NEURON - tutorial C (part 3)

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

Lubica Benuskova

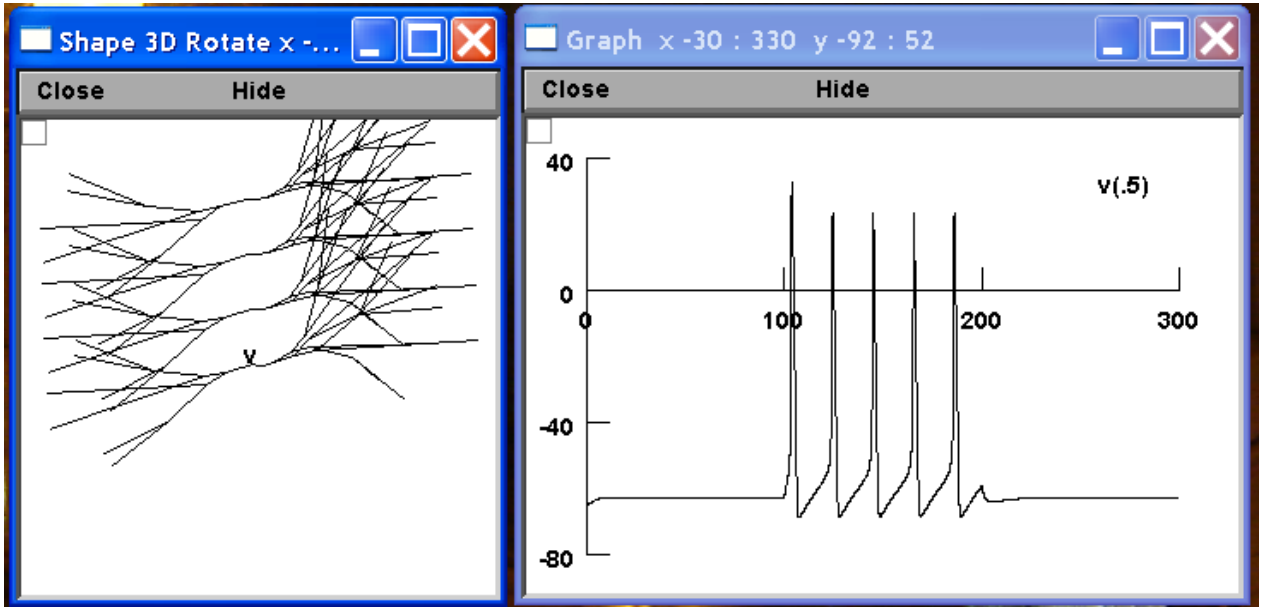
Lecture 7

How to connect neurons using **`NetCon`**

Centre for Cognitive Science Bratislava

What we've got so far: `sthC2.hoc`

- The final four neurons, each with a full dendritic tree morphology are shown here in a shape plot (image on the left).
- Next to it is the voltage trace in one of the model neurons, i.e., **`SThcells[0].soma`**, as a result of the current pulse injection (recall the cells are not connected yet).

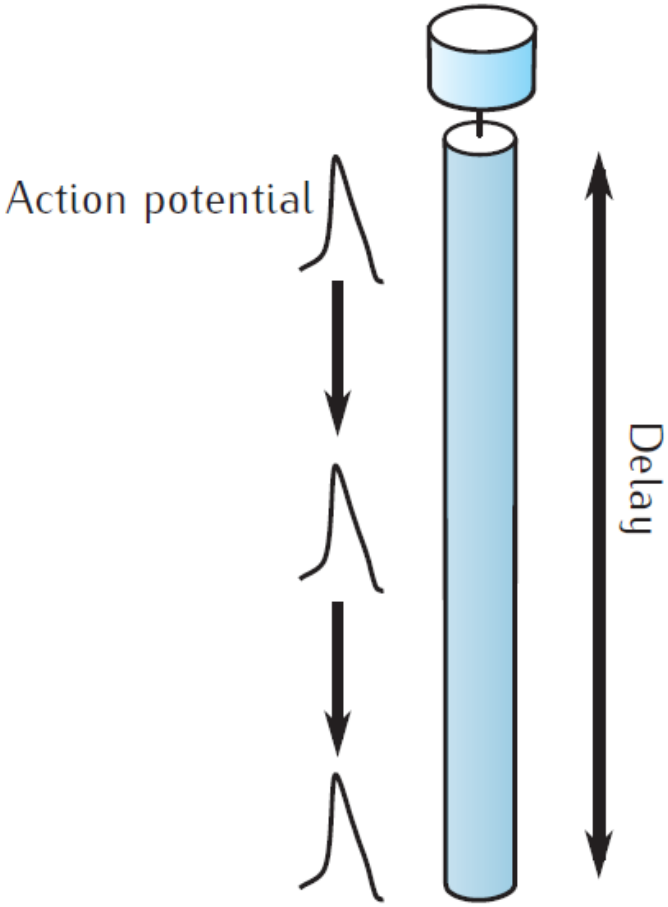


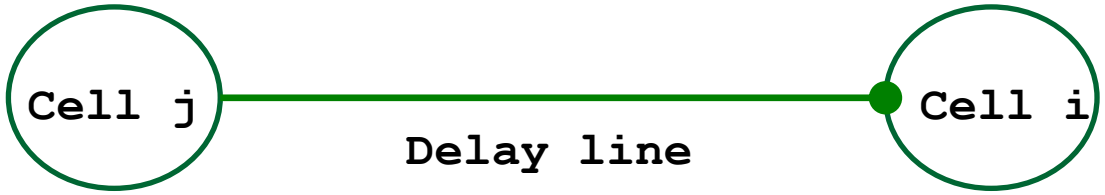
Connecting neurons together

- Networks of neurons are formed by connecting neurons via synapses.
- The signal that passes from the efferent (i.e. sending) neuron to the receiving neuron is the action potential (spike).
- One possibility is to work with a multi-compartmental model of the axon along which the spike propagates from one neuron to another. This is computationally expensive and often unnecessary.
- Spikes are stereotypical and the information content of signals passing from one neuron to another is carried by the times and frequency of spikes, rather than the voltage waveform of a spike.

The delay line

- The common approach is to treat the inter-neuronal signal to be the presence or absence of a spike.
- A spike is initiated in the soma and then propagates along the axon. This can be modelled as a delay line, which specifies the time Δt taken for the spike to travel from the soma to synapse.
- By using this so-called event-based approach, we can dramatically reduce the amount of computation.





- The voltage in the soma of the presynaptic cell *j* is continuously monitored. If the voltage goes over a certain threshold value, this means occurrence of an output spike (action potential).

- The delay line then signals this occurrence to the synaptic contact on the postsynaptic neuron *i* after some time delay Δt.

- The delay is calculated based on the known constant velocity of spike propagation along axons and the distance between the two neurons as Δt = distance / velocity.
  - The nerve conduction velocity (0.5 – 120 m/s) depends on type of a neuron, myelination, temperature and age.

Connecting neurons together – NetCon

- First, we must add an additional public object variable to our neuron template, the **`nclist,`** to be accessible from outside of the template.

- Then, we must declare a new object variable **`nclist`** that will refer to a list that contains an arbitrary number of NetCon objects.

- So, now we begin our subthalamic neuron template with:

```
begintemplate SThCell

public soma, treeA, treeB, nclist

create soma, treeA[1], treeB[1]

objectvar f, nclist
```

Connecting neurons together – **List**

- Then we continue with the init() procedure like this:

```
proc init() {
        local i,me,child1,child2
        create soma
        nclist = new List()

        ......
}
endtemplate SThcell
```

- In NEURON, a **List** is an object that holds a list of other objects. The advantage of a list is that we don't have to specify in advance how big it will grow, as we have to for an array (like `dend[ndend]`).

Continuation of `sthC3.hoc`

- After defining the Sthcell template, we continue the code with defining the array of neurons

```
nSThcells = 4

objectvar SThcells[nSThcells]

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

- Next, we let only the neuron **`SThcells[1]`** to receive IClamp current, instead of all 4 neurons.

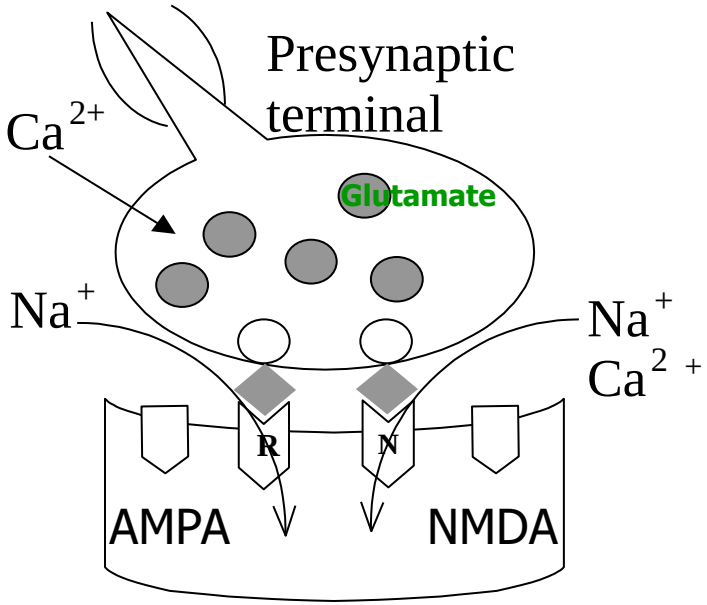
Just one stimulating electrode

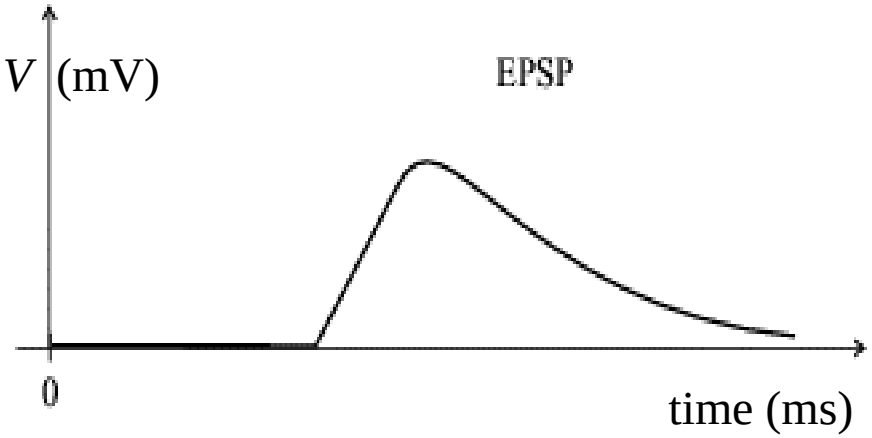
- Let only the neuron **SThcells[1]** will receive IClamp current. The modified piece of code looks like this:

```
objectvar stim[nSThcells]

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

- **Now is the time to deal with synapses proper.**

Excitatory synapses

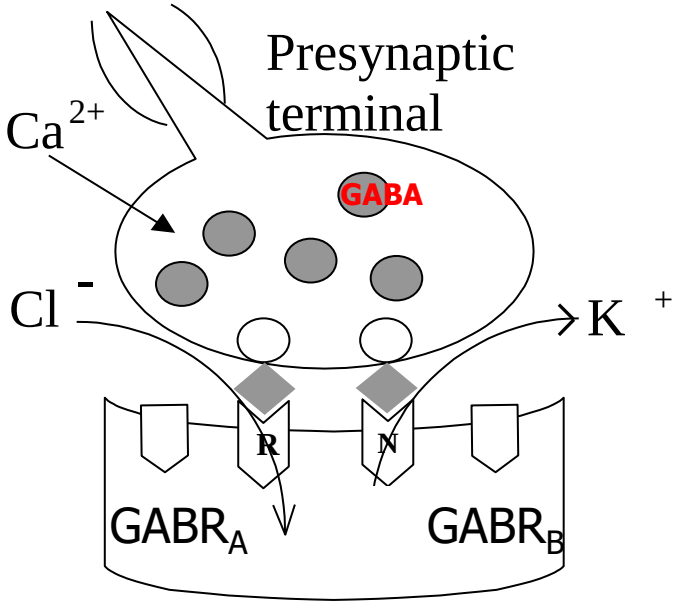


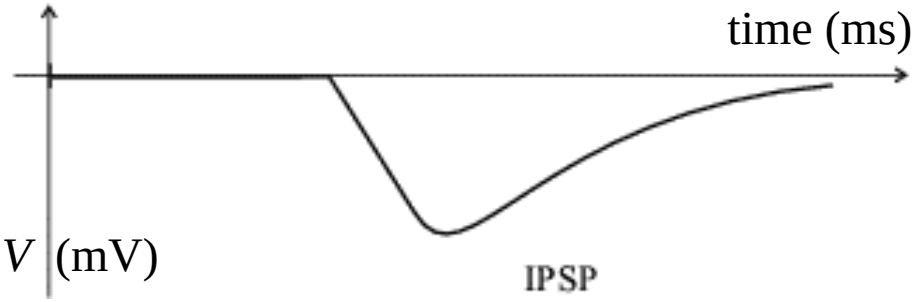


- Neutrotransmitter: glutamate.
- Postsynaptic receptors called AMPA and NMDA are associated with ion channels for Na^+ or for Na^+ and Ca^{2+}, respectively.
- When we measure the electric potential at the postsynaptic membrane, we see a positive deviation from the resting potential, which is called an **excitatory postsynaptic potential (EPSP).**

Inhibitory synapses

- Neutrotransmitter: GABA
- Postsynaptic receptors $GABR_A$ and $GABR_B$ for ion channels for Cl^- and K^+, respectively.
- When we measure the electric potential at the postsynaptic site, we see a negative deviation from the resting potential, called an **inhibitory postsynaptic potential (IPSP).**





Postsynaptic potential (PSP = I_{syn} R_m)

- PSP (either EPSP or IPSP) is the result of electric current *I* that flows through the receptor-gated ion channels and obeys the equation:

$$I_{syn}(t) = g_{syn}(t)\left(V(t) - E_{syn}\right)$$

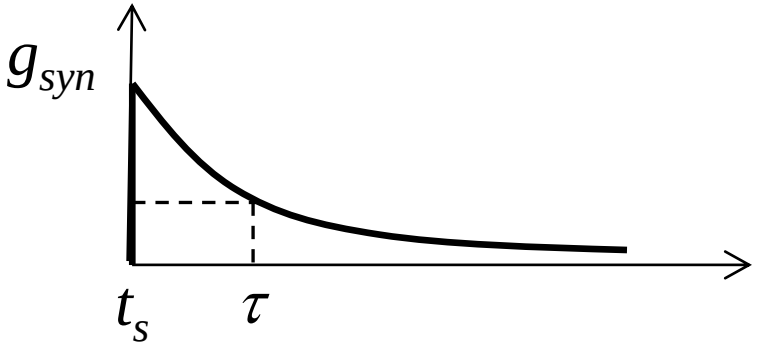
- Where the effect of neurotransmitter binding to and opening the postsynaptic receptors/ion channels is the conductance change, g_{syn}.

- *V* is the actual (momentary) value of transmembrane potential.

- E_{syn} is the reversal potential of those ion channels (Na, K, Cl, Ca) that mediate a given synaptic current in the postsynaptic membrane.

NEURON has two PSP functions **ExpSyn** and **Exp2Syn**

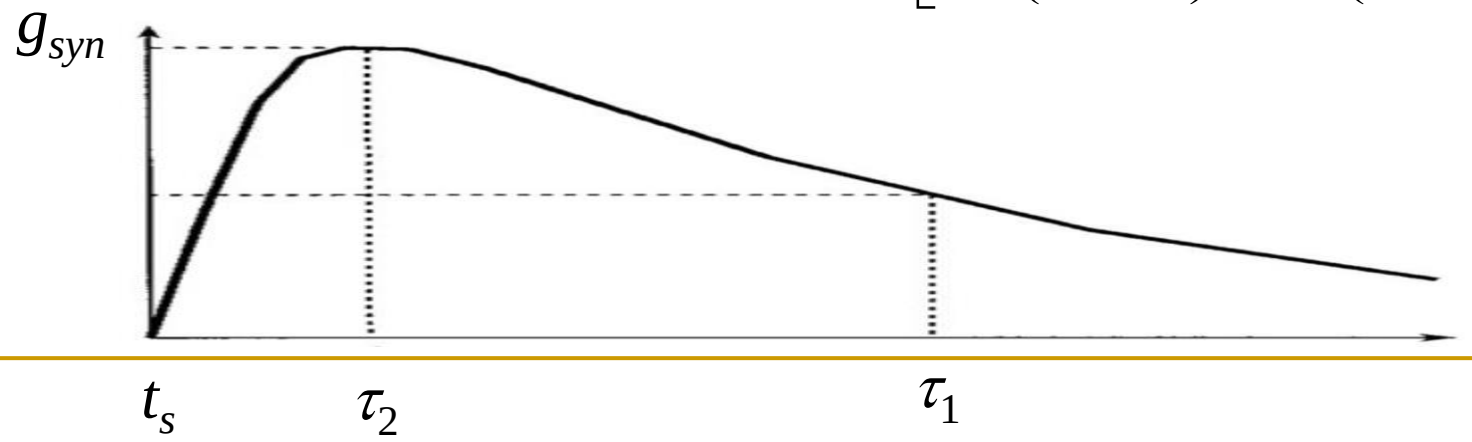
- Single exponential **ExpSyn**



$$g_{syn}(t) = g_{\max} \exp\left(-\frac{t - t_s}{\tau}\right)$$

- Double exponential **Exp2Syn**

$$g_{syn}(t) = g_{\max} \frac{\tau_1 \tau_2}{\tau_1 - \tau_2}\left[\exp\left(-\frac{t - t_s}{\tau_1}\right) - \exp\left(-\frac{t - t_s}{\tau_2}\right)\right]$$



ExpSyn : notes on synaptic conductance g_{max}

- In **ExpSyn,** the synaptic weight parameter is the peak amplitude of the synaptic conductance g_{max}. This has two implications.
- First, the weight parameter for a conductance change should be always non-negative, i.e. synaptic conductance g_{max} >= 0.
- Second, whether the synapse is inhibitory or excitatory depends on whether the reversal potential lies above or below the spike threshold, which is around – 50 mV.
- Synaptic weight g_{max} will be defined later in a new NetCon object.

Placing a synapse

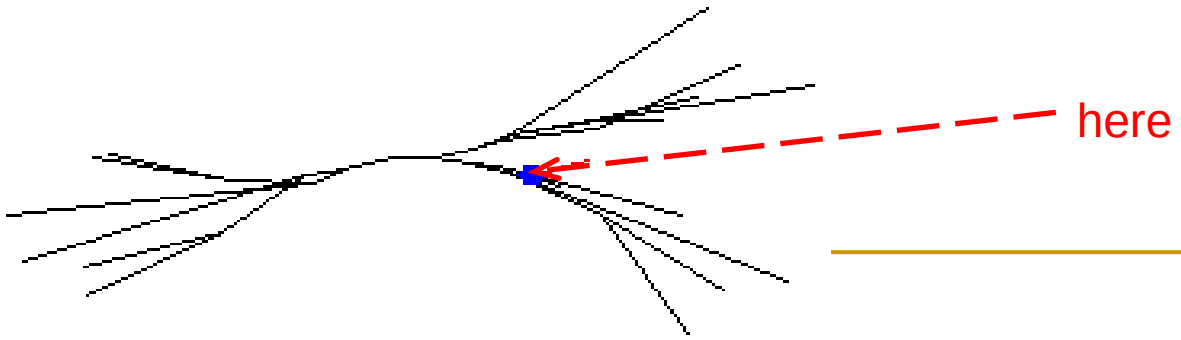
- In order to connect the neurons, we must create synapse objects. A synapse is an object that can be positioned anywhere on a neuron.

```
maxsyn = 10
objectvar syn[maxsyn]
```

- Now, we want to connect the stimulated neuron **SThcells[1]** to model neuron **SThcells[0]** and observe EPSP at the its soma.

```
SThcells[0].treeA[7] syn[0] = new ExpSyn(0)
```

- A synapse 0 will be placed at the branch 7 of treeA of the neuron 0:



ExpSyn : variables

- When we create a new instance of ExpSyn, we introduce these variables (with certain default values for an excitatory synapses):

```
syn.tau // decay time constant in ms
syn.e  // reversal potential in mV
syn.i // synaptic current in nA
```

- If we want to make an **inhibitory synapse**, we have to set a new value for this reversal potential, e.g., **syn.e = –60** should work.

- Thus, to create an **inhibitory** synapse, we write

```
SThcells[0].treeA[7] syn[0] = new ExpSyn(0) syn.e = -60
```

Creating a new **NetCon** object

- To create a new **NetCon** object, we use the command format:

```
new NetCon(&source_v,synapse,threshold,delay,weight)
```

- ***source_v*** is the source voltage (e.g., **SThcells[1].soma**);
- ***synapse*** is the object variable that refers to the synaptic object receiving the events (in our case ***syn[0]***);
- ***threshold*** is the threshold value, which the ***source_v*** must reach for it to be considered that a spike has occurred;
- ***delay*** is the connection delay in milliseconds, and
- ***weight*** is the connection weight strength of the synapse = $g_{\max}$.

Appending created synapse

- To append **SThcells[1]** to the dendritic branch 7 of **treeA** on subthalamic neuron **SThcells[0]** we add the command:

```
SThcells[1].soma SThcells[0].nclist.append(new
NetCon(&v(1), syn[0], -20, 1, 0.5))
```

- First, this command accesses **SThcells[1].soma**
- then the **nclist** of **SThcells[0]** has a new NetCon appended.
- This NetCon object has a source voltage of **SThcells[1].soma**, which is read through **&v(1)**.
- The NetCon object is applied to **syn[0]** which we have already attached to `SThcells[0].treeA[7]`.
- Our threshold for action potentials is –20mV, our delay 1ms, and our synaptic weight 0.5.

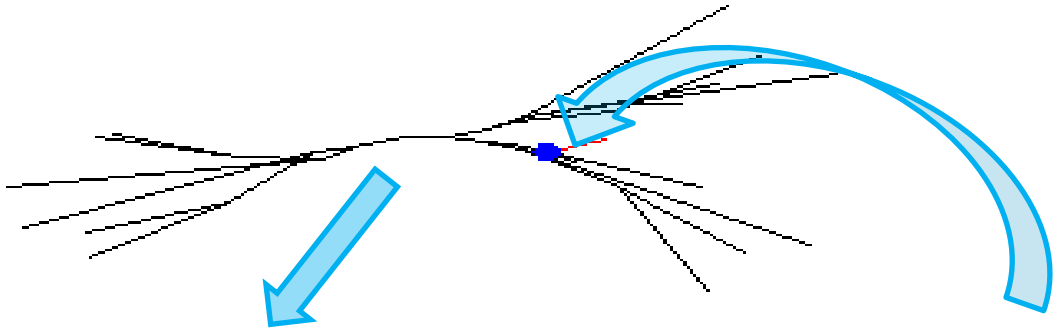
Last lines of **sthC3.hoc**

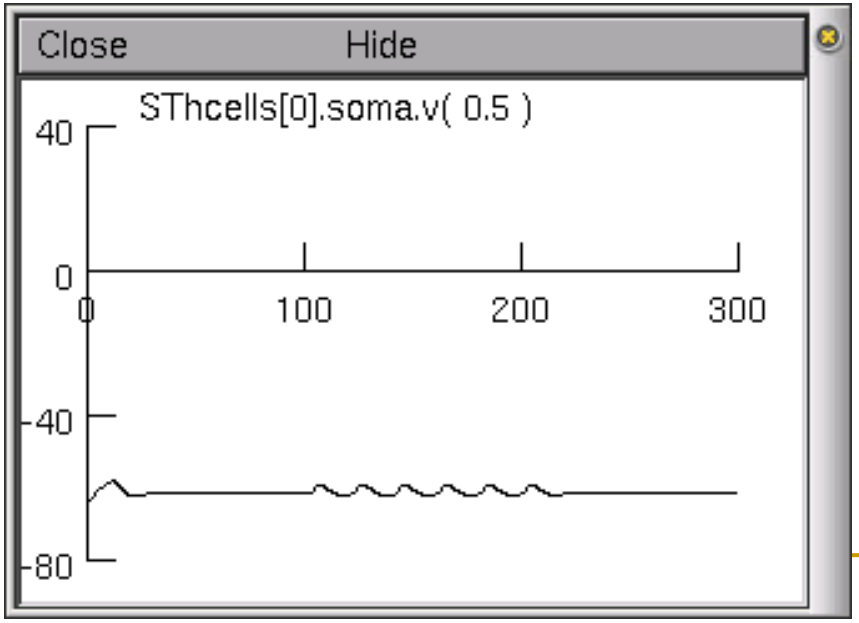
- Thus, the final lines read:

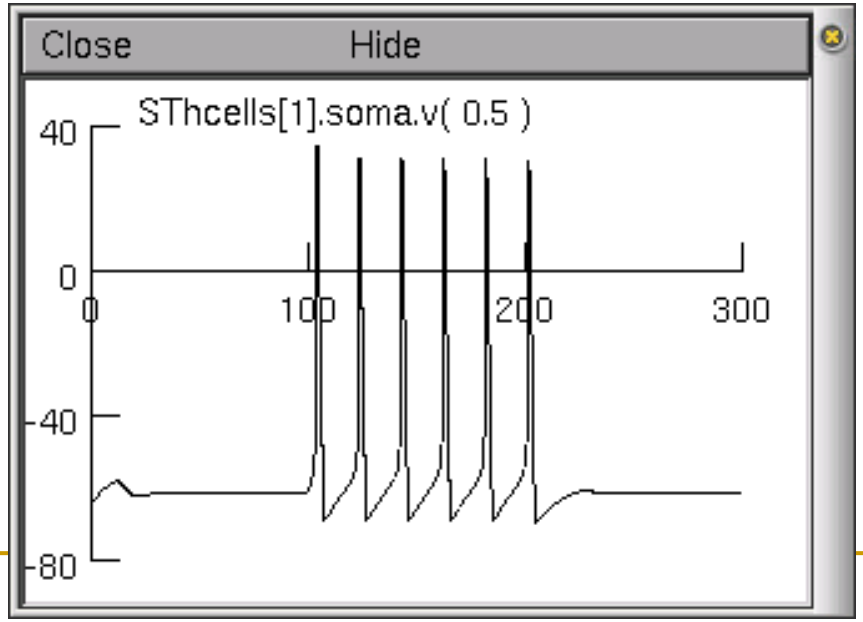
```
maxsyn = 10
objectvar syn[maxsyn]

//creating new synapses and appending them

SThcells[0].treeA[7] syn[0] = new ExpSyn(0)
SThcells[1].soma SThcells[0].nclist.append(new
NetCon(&v(1), syn[0], -20, 1, 0.5))

access SThcells[0].soma
tstop = 300
```

Simulation of **`sthC3.hoc`**

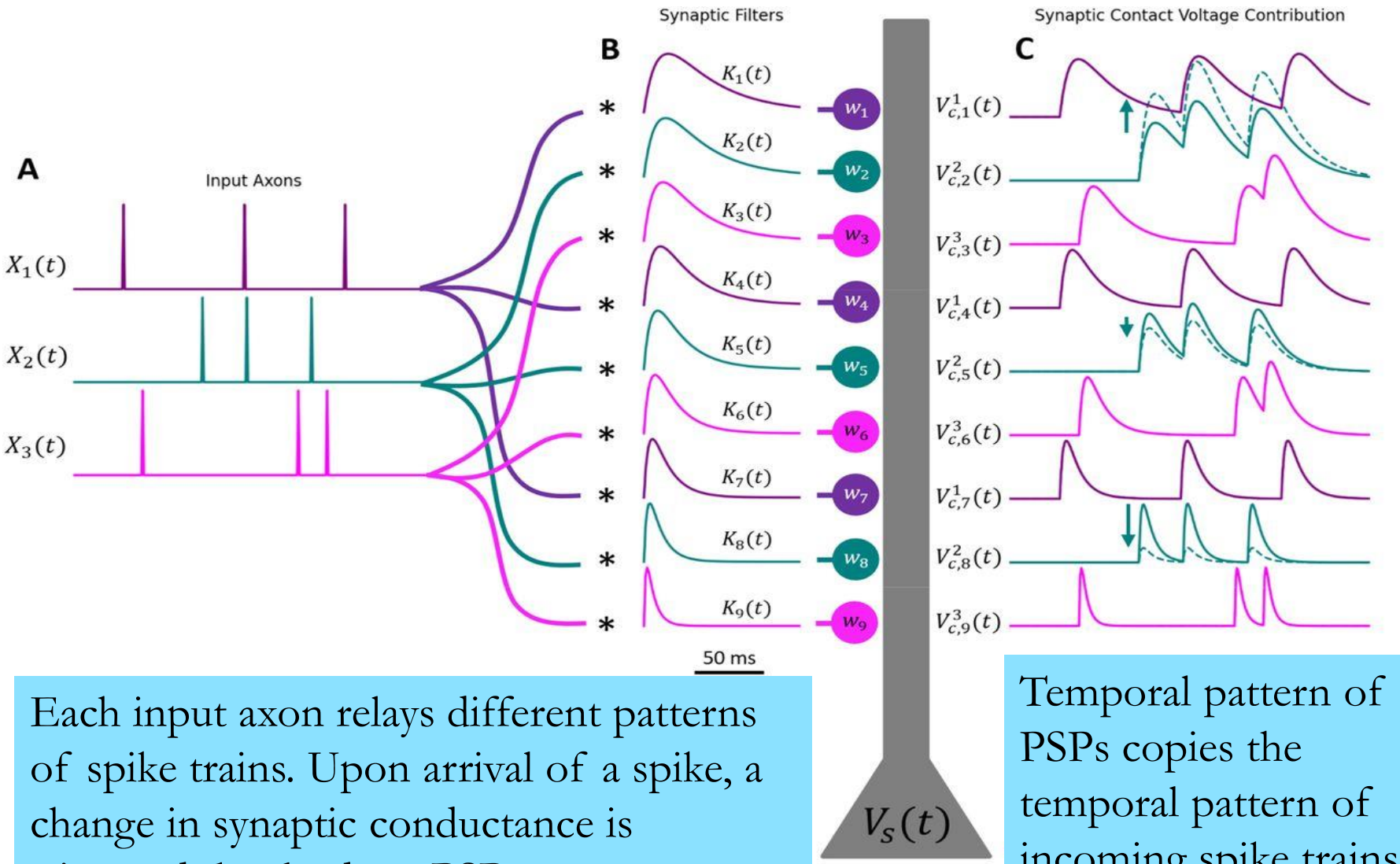
- If we run the simulation and plot the voltage at **`STheells[0].soma`** we see EPSPs resulting from the spikes of neuron **`STheells[1]:`**







Temporal patterns of PSPs at individual synapses



Each input axon relays different patterns of spike trains. Upon arrival of a spike, a change in synaptic conductance is triggered that leads to PSP.

Temporal pattern of PSPs copies the temporal pattern of incoming spike trains.

Networks of neurons: design questions

- The most common properties that are investigated in network models are the patterns of firing within the neural network and how such patterns contribute to processing of the incoming stimulation and/or how they are modified through specific synaptic learning rules.

- Having a full-scale model of a given brain area is usually computationally infeasible b/c there are millions of neurons and billions of synapses.

- Thus, we must deal with how to **downsize** their numbers and yet accurately model real network dynamics / behaviour.

Scaling neuronal numbers

- Suppose our network is going to be one-tenth ($1/10^{th}$) the size of the brain area we are modelling.

- Assume this area contains three cell types – excitatory neurons that make up 80% of the cell population, and two types of inhibitory interneurons, each constituting about 10% of the cell population.

- It is important to scale neuronal numbers to retain these relative proportions of cells of different types (80:10:10) in our $1/10^{th}$ -sized model.

- Interestingly, simulations have shown that networks containing the same cell types, but in different proportions, can show significantly different spiking behaviour (Földy et al., 2003, 2005).

Level of detail

- Another major decision is to choose at which level of detail to model the individual neurons, whether to include a full-morphology dendritic tree or reduced-morphology with the reduced number of dendrites.

- For a large-scale network with thousands, or hundreds of thousands of neurons, we may choose to use the simplified integrate & fire neurons (AdEx, Izhikevich).

- Even the construction of the realistic multi-compartmental model of a single neuron involves a range of choices concerning number of sections and segments, identifying and coding different types and distributions of ion channels.

Scaling synapses

- Next, we should determine the pattern of connectivity in the network, i.e. which neurons connect with which ones, types and distribution of synapses, choose ExpSyn functions and their parameters.

- In our model network, excitation and inhibition for each type of neurons should be as close as possible to that experienced by real neurons in vivo.

- Given that we have let's say $1/10^{th}$ of synapse numbers, we scale up the maximum synaptic conductance of each connection by a factor of ten, i.e. we make synapses 10 times stronger in order to get an equivalent magnitude of excitation and inhibition.

Positioning neurons in space

- Real neurons have a particular location within the brain, and connectivity patterns between neurons are often distance-dependent (the timing in delay lines depends on distances).

- To capture these patterns, it may be necessary to place our model neurons in virtual space.

- In many instances, say, a cortical column or other small part of the cortex, it may be reasonable to assume that connectivity is completely uniform (e.g. every neuron connects to every other neuron) or that there is a fixed probability that one neuron makes contact with another neuron. In this case the precise spatial location of a neuron is not relevant and can be ignored.

Positioning neurons in space

- In general, though, we will need to lay our cells out in some 2D or 3D arrangement that reflects the physiological layout.

- Typically, this is done with a regular spacing between cells like on a grid. Then, when forming connections between cells, the probability that an efferent cell forms a connection onto a target cell can be a function of the distance between them.

- This function is often an exponential or Gaussian function so that the probability of connection decreases with distance.

- Each choice involves a compromise over the level of biological detail to include.

Variability in cell properties

- Majority of existing neuronal network models contain populations of cells with completely uniform properties, including morphology and membrane physiology. This does not reflect the variation seen within biological neurons and may lead to artifacts in network behaviour.

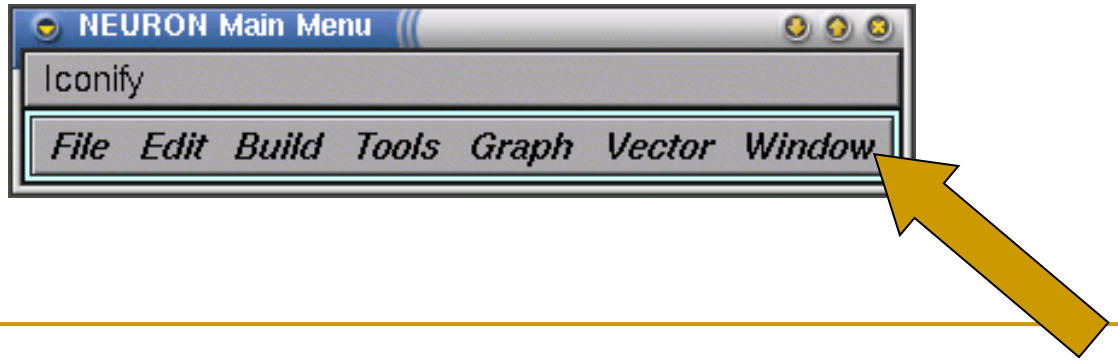
- A better approach is to introduce variance into one or more cellular properties, including membrane resistance, resting membrane potential and ion channel densities – **population modelling**.

- Experimental estimates of these parameters may be available that indicate the magnitude of variance in a biological population. That is, variations in electrophysiological responses may be available from the literature or can be randomly artificially generated.

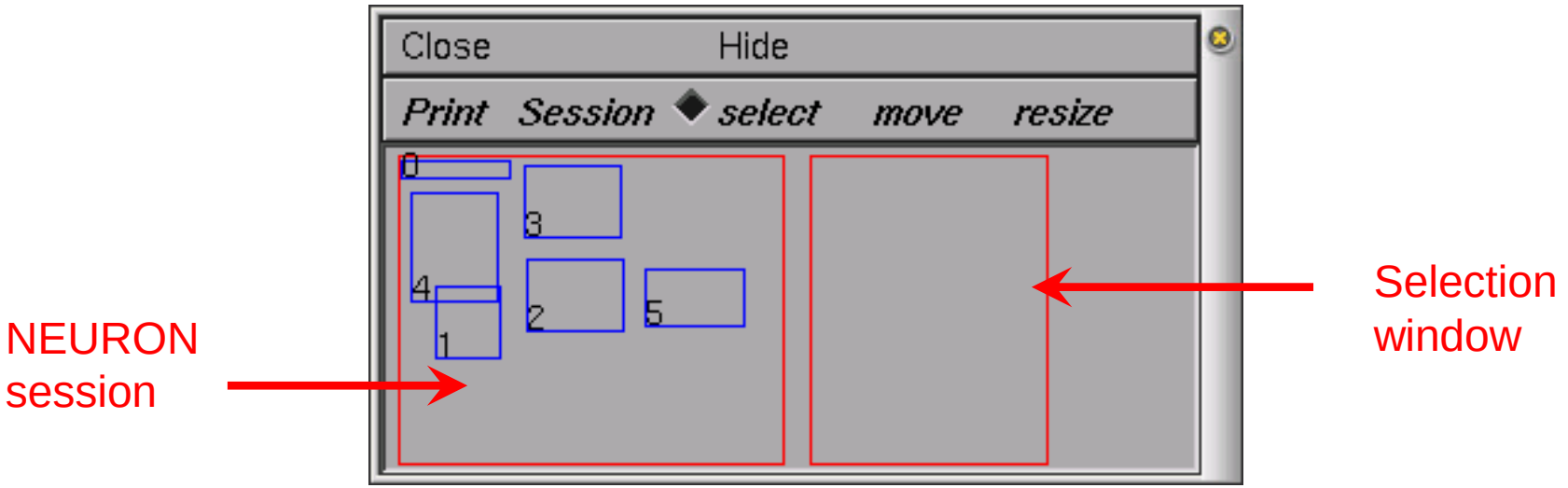
Notes on the print & file window manager

- This function enables you to
  - print selected windows from your simulation and
  - store the whole session, so that the next time you can continue where you have stopped.

- To open the *Print & File Window Manager*, select *Print & File Window Manager* from the *Window* menu on the *Main menu*.



The print & file window manager



- The left most of the two red rectangles in this window represents the entire NEURON display. Each smaller blue rectangle (with a number in it) represents one of NEURON's windows.

- The second red rectangle represents a sheet of paper--we will call this the Selection rectangle. It is used to print selected windows to a file or printer and to save selected windows in a session file.

Saving sessions

- After creating several graphs, you may want to save the windows you have created (i.e., graphs and panels) to a file so that you can recall them later.

- NEURON allows you to save either all or selected windows to a session by selecting the *Save selected* or *Save all* option of the Session menu in the *Print & File Window Manager*.

- Save all will save the position and contents of all NEURON's windows. Save selected will save only those windows that are currently selected in the Selection rectangle in *Print & File Window Manager*. Either of these options will pop up a window, in which you can enter the filename of your saved session.

Retrieving sessions

- If we save our session to a file (e.g., **`sthC.ses`**), we can
  - either load the session each time we load our program by selecting the *Retrieve* option of the *Session* menu in the *Print & File Window Manager*,
  - or we can have our program automatically load our session for us. To do this, we need to add the following at the very end of our program:

```
xopen("sthC.ses")
```

  - where **`sthC.ses`** is the name of the session we saved. The next time we start our program, the session with our graphs and menus will automatically be loaded into NEURON.