This is a tutorial explaining how to use the Matlab\textsuperscript{®} code NAC\textunderscore Fundamentals.m, available at the site: http://www.neuralassembly.org/downloads/.

The program can simulate basic concepts of neural assembly computing (NAC). The code has a section (1.CONFIG) in which you can change simulation parameters.

1. **Simulation mode, the number of assemblies and neurons per assembly**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numAssb</td>
<td>Assign the number of assemblies simulated</td>
</tr>
<tr>
<td>nNa</td>
<td>Assign the number of neurons in each assembly</td>
</tr>
<tr>
<td>nN</td>
<td>Internal variable containing the total of simulated neuroids (nN=nNa*numAssb)</td>
</tr>
<tr>
<td>simMode</td>
<td>simMode=0 simulate synfire chains, simMode=1 simulate polychronous groups</td>
</tr>
</tbody>
</table>

You can assign the number of assemblies and the number of neurons/assembly by changing `numAssb` and `nNa`, respectively. The number of neurons in network (`nN`) is calculated. Assemblies are indexed from 0...(nN-1). Be careful for not indexing an assembly greater than specified in `numAssb`.

Two types of assemblies (or coalitions) can be simulated: synfire chains and polychronous groups. You can change the simulation mode by changing `simMode`. If synfire is chosen (`simMode`=0) the program make the assembly neuroids\footnote{Following Cruse, I used the term neuroids when designating artificial neuron models, while the term neuron I used mainly for biological neuron cells. See Hilk Cruse, “Neural Networks as Cybernetic Systems. 3rd and revised edition. 2010. Available at: http://www.brains-minds-media.org/archive/1841.} fire simultaneously. If polychronous groups (PG) is selected (`simMode`=1) then neurons in an assembly fire time-locked, but scattered and not synchronized.

2. **Neuroid model and types of neuroids**

One purpose of the program is to verify which neuron types are eligible for participating in NAC. The Izhikevich's Simple Model neuron was chosen because this model can reproduce the behavior of a range of natural neurons. In this neuroid model you can adjust four parameters ($a$, $b$, $c$, and $d$; see IEEE\textsuperscript{-}TNN(2004)15:1063-1070, or here) for changing a neuron behavioral response.

In order to make simple the neuron type selection the program uses the look-up table (LUT) `Pars[]`, which contains a set of built-in parameters. User has only to point which neuron he/she intend to use.
### 3. Creating the topologies

We claim that two main components are responsible for NAC: the *propagation delay* (axon + synaptic + dendritic) among the participant neurons, and the *synaptic weights* of the each interconnection among neurons. Plasticity may be responsible for keep the NAC dynamics working properly. Plasticity is not treated here, and it is the subject of other program and other set of experiments.

You create the network structural topology by changing these two components in the matrices $DA$ (which controls the *axonial delay*) and $Sw$ (which controls the *synaptic weights*). The program creates these two $nN,nN$ matrices ($DA$ & $Sw$) and you define how neurons are delayed and connected. In the matrices, lines represent $i$ (pre-synaptic) neuroids and columns representing $j$ (post-synaptic) neuroids.

It would be a tedious job to interconnect all neuroids once both the synaptic weight and the propagation delay must be assigned. In order to make this job easier, the program offers a topologic description alternative. User describes the network topologic structure in terms of assemblies ($i$) that interacts with assemblies ($j$) in a matrix as follow:

$$\text{Topol}=[i_1 j_1 s_1 d_1 ip_1; i_2 j_2 s_2 d_2 ip_2; ... i_n j_n s_n d_n ip_n];$$

Where:

- $i_x$ denote the pre-synaptic assembly. The assembly index “0” (zero) is used for input stimulus.
- $j_x$ denote the pos-synaptic assembly. Note that the index ‘x’ stands for an assembly number, not an individual neuron.
- $s_x$ denote the synaptic weight among all $i_x \rightarrow j_x$ neuroids;
- $d_x$ denote the average axonal propagation delay from all $i_x \rightarrow j_x$ neuroids. In synfire simulation $d_x$ is exactly the delay time between the assembly $i_x$ to $j_x$; but when simulating PG there is a “scattering factor” controlled by $fatorDispGP$, which randomly create particular time relations among neurons from $i_x$ to $j_x$. In this case, $d_x$ is the mean time delay among the two assemblies.
- $ip_x$ denote the $j_x$ *neuroid type* (it is an index for the matrix $Pars[]$). Pay attention because the user can only change the type of the post-synaptic neurons.

**Example:**

This is an example of how you can construct a topology:

$$\text{Topol}=[0 1 \text{sw0} \text{tBD} \text{dNT}; 1 2 \text{sw0} \text{tBD} \text{dNT}; 2 1 \text{sw0} \text{tBD} \text{dNT}; ... 1 3 \text{sw0/2} 1.5*\text{tBD} \text{dNT}; 2 3 \text{sw0/2} \text{tBD/2} \text{dNT};];$$

By declaring this matrix user creates the following structural topology:

Let us analyze the fist line vector ($0 1 \text{sw0} \text{tBD} \text{dNT;}$): it means the assembly $0$ (neuroids from $1...nNa$) is connected to the assembly $1$, the synaptic weight among all their neurons is assumed to be in
the variable $sw_0$, the mean propagation delay from assembly 0 to 1 is into the variable $tBD$, and the neuroid type for the assembly 1 is indexed by $dNT$.

The assembly 1 is composed by the neuroids number ($nNa+1$) to ($2*nNa$); and subsequent assemblies have the same indexing rule. In this sense, a neuroid $nZ$ is a neuron of the assembly $Z$ if:

$$Z.nNa + 1 \leq nZ \leq (Z + 1).nNa$$

In this program, rarely users have to index a neuroid separately; but it can be necessary if some kind of failure experiment is intended.

In the example, the first line vector forces the program to construct the relation among all neuroids from assembly 0 to assembly 1. The next vectors, subsequently, create the following relations: assembly 1 is connected to assembly 2; and then the assembly 2 is connected back to assembly 1. It means the assemblies 1 and 2 feedback each other. They form a recurrent bistable loop, a bistable assembly.

In the next two line vectors the assembly 1 is connected to the assembly 3 by a synaptic weight that is half of the default $sw_0$. Note also that the delay from 1→3 is 50% greater (1.5*$tBD$) than the delay from 1→2. In the next line vector the assembly 2 is also connected to the assembly 3 by the same synaptic weight ($sw_0/2$), but the delay from assembly 2→3 is only 50% ($tBD/2$) of the default delay time. It means that when the spikes from 1 reach the assembly 3 it is coincident to the spikes coming from 2. As the synaptic weight of these connections are both half of the default value, one assembly alone should not fire the assembly 3, but the coincidence 1 AND 2 may have great chance of firing the assembly 3.

This is yet a rough way to construct the topology, and it is necessary to create a friendly interface for doing this job; but that’s enough for now for simulating all the basal aspects in NAC.

4. Synaptic weights

The default value for synaptic weights is calculated at the beginning of the program as a function of the number of neurons present in an assembly. The value $sw_0$ is calculated to guarantee that a neuron can spike when excited by all synaptic inputs of an assembly interconnected by this synaptic weight.

Users are encouraged to try any values for synaptic weights in order to test the conditions for the logic functions really work. As said above, this program has not an automatic way for changing the synaptic weights during the simulation because it belongs to the domain of plasticity. Plasticity (mainly STDP) is the subject of other set of experiments. This program is only intended to show how the basics in NAC can construct all functions, the bistable assemblies, and the inhibitory (NOT) functions.

In order to obtain inhibitions between assemblies the user can assign negative values for the synaptic weights. In the previous example, the assembly 3 would be responsible for inhibit the assembly 2 if the user insert the line vector into the matrix Topol[]:

```
...; 3 2 -sw0 5 22;...
```

In this case, all the neurons in the assembly 3 are inhibitory as they are connected to the assembly 2 by a synaptic weight $-sw_0$. The delay from 3→2 is 5 milliseconds, and the neuron is the fast spiking neuron type (index 22 in the LUT Pars[]).
### 5. Timing parameters

The following parameters are related to the timing in simulation:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tsim</code></td>
<td>Time of simulation (in milliseconds). The resolution of the simulator is 1 ms</td>
</tr>
<tr>
<td><code>maxPD</code></td>
<td>Maximum propagation delay (ms). See note **.</td>
</tr>
<tr>
<td><code>tBD</code></td>
<td>Average time basis for delays between assemblies ((i)\rightarrow(j)) in (ms)</td>
</tr>
<tr>
<td><code>tINev</code></td>
<td>Time (ms) for the first input event occurrence (in assembly 0)</td>
</tr>
<tr>
<td><code>tiINev</code></td>
<td>Time interval (ms) for repeating the input event, after the first input event.</td>
</tr>
<tr>
<td><code>tGReset</code></td>
<td>Time (ms) for occurrence of a global reset (an inhibitory burst at the top most assembly). If <code>tGReset=0</code> no global reset is executed.</td>
</tr>
<tr>
<td><code>ftINevRnd</code></td>
<td>Flag: =1 add random delays to each input event (0=don't randomize)</td>
</tr>
<tr>
<td><code>dispFactorPG</code></td>
<td>Dispersion factor for polychronous groups. In relation to the <code>tBD</code>, this factor randomly spread (+ or -) the spiking time difference for the PG neuroids.</td>
</tr>
</tbody>
</table>

** These parameters are self explanatory; but the `maxPD` parameter requires explanation. The program records all spikes in order to create the raster plot at the end of simulation. The parameter `maxPD` controls how the program goes back into the “past” for searching fired neuroids. In other words, each one millisecond in time advance (each iteration) forces the program to turn back `maxPD` milliseconds and check for all fired neuroids during the period:

\[
(tci - maxPD) \leq t \leq (tci - 1)
\]

where `tci` denotes the time of the current iteration; and `t` is the loop variable used for seeking the fired neuroids during such interval. The `maxPD` parameter is somehow connected to the `dispFactorPG`. For instance, suppose one have used the maximum delay among assemblies = `tBD`. For `dispFactorPG=2` it is required `maxPD= tBD+12`, and for `dispFactorPG=3` it may be necessary to use `maxPD= tBD+16`. It means that an assembly probably fired at `tBD` ms ago, but the dispersion around this time may be \(\pm 12\) or \(\pm 16\) ms respectively. Searching on the past fired neuroids has to include these spikes.

Note that `tBD` should never be greater than `maxPD`, because `tBD` controls the mean delay time from one assembly to the other, and the program must search in the spiking history for a time greater than this mean delay time. User must also pay attention when creating the `Topol[]` matrix, in order to not use a delay between assemblies greater than `maxPD`.

Note also that as you increase the `maxPD` parameter you are increasing the computational demand.
6. Interpreting the raster plot

Running the program as is, at the end of simulation, a figure like the following one will be plotted. Then, change the default neuroid type \((dNT)\) from 1-23 skipping the types 16, 18, 19 and 20; for these neuroids are special (see figure in [http://www.izhikevich.org/publications/whichmod.htm](http://www.izhikevich.org/publications/whichmod.htm)).

![Raster plots](image)

**Figure-1. Two raster plots for NAC with synfire chains.** (A) The left raster plot is similar for the neuroid types \((dNT)=\): 1, 5, 6, 9, 12, 21, and 23. (B) The raster plot at right is similar for the neuroid types 2, 3, 4, 8, 10, 11, 14, 15, 17, and 22. The neuroids type 7 and 23 also present raster plot similar to B, but the assembly 5 cannot fire.

Starting from the bottom, the first five neuroids \((nNa=5)\) belongs to the assembly 0 (excitatory stimuli, or input stimuli), which fires at the time \(tINev=10\) ms, and repeat at 510 ms \((tillNev=500)\). After 40 ms \((tBD=40)\) the assembly 0 triggers the assembly 1 (neuroids 6…10). After another tBD ms the assembly 1 triggers assembly 2, that triggers assembly 3 that feedback and triggers assembly 2. It means that the bistable assembly 2-3 remain firing each other.

Now, let us look at the topology matrix:

\[
\text{Topol}=[0 \ 1 \ 0 \ 2 \ 3; 1 \ 1 \ 1 \ 2 \ 3; 2 \ 2 \ 3 \ 3 \ 3; 3 \ 3 \ 3 \ 3 \ 3; 4 \ 4 \ 4 \ 4 \ 4]
\]

The first line in the matrix describes the connections \((0\rightarrow1, 1\rightarrow2, 2\rightarrow3, 3\rightarrow2)\) explained above, all of them with the same synaptic weights and the same delay among assemblies. In the second line of the matrix we see that the assembly 2 triggers the assembly 4 after \(tBD/2\) ms, and the assembly 3 also triggers the assembly 4 after \(tBD/2\) milliseconds. It results in the assembly 4 firing twice faster than assemblies 1 or 2. It means also that the assembly 4 fires because assembly 2 or assembly 3 triggers it; it can be denoted by \(k_4=k_2+k_3\) (the assembly 4, the kernel \(k_4\), fires due to \(k_2\) OR \(k_3\)).

The third line define the connections between the assemblies 2→5 and 3→5. Note that the delay for spikes from 2 to reach assembly 5 is \(1.25*tBD+4\), while the delay for spikes from assembly 3 to reach the assembly 5 is only \(0.25*tBD-2\). It means these spikes coincide in assembly 5, but as the AND or NAND functions are quite sensitive to time coincidence, time fine tuning was necessary (+4 ms to spikes from 2→5 and -2 ms to spikes from 3→5). Note also that the synaptic weights for 2→5 and for 3→5 are...
0.5*sw0. Alone, neither assembly 2 nor 3 is able to fire the assembly 5; but the time coincidence of their spikes triggers the assembly 5. We can then say that \( k_5 = k_2 \cdot k_3 \) (\( k_5 \) depends on firing of \( k_2 \) AND \( k_3 \)).

The assembly 6 is activated by the global reset round 350 ms (\( t_{GReset} = 364 \)). In the last line of the matrix \( \text{Topol}[] \) we see that assembly 5 has an inhibitory connection to assembly 2. The assembly also has an inhibitory connection to assembly 2. Neither assembly 5 nor assembly 6 is able to apply a “veto”, an inhibition to assembly 2 alone. The assembly 6 has no delay (only 1ms) for trigger the assembly 2, and when the time delay matches with the assembly 5 they can inhibit assembly 2. They perform a \( NAND \) operation, once \( k_2 = (k_5 \cdot k_6)' \). This is a (stochastic) logic function more difficult to be adjusted, and it does not happen to all neuroid types without changing the synaptic weights or adjusting timing delays.

If you change the simulation mode for polychronous groups (\( \text{simMode}=1 \)); you can also change the number of neuroids per assembly (e.g. \( n_{Na}=9 \)), and run the program for all neuroid types (by changing \( dNT \)), you can obtain figures like these:

![Figure-2. Two raster plots for NAC using polychronous groups.](image)

(A) Note that neuroids do not fire synchronously, but time-locked. (B) All neuron types (from 1 to 15) respond execute the task on the above topology, although it is necessary to adjust the parameters \( sw_0 \) (the default synaptic weights calculated by \( sw_0 = K/n_{Na} \)) and fine tuning the delays (as in \( 1.25 \cdot t_{BD} + 4 \)) when executing \( AND / NAND \) functions.

This is the same topology and the same functionality of the previous example. The matrix \( \text{Topol}[] \) is exactly the same. So, it is expected the assemblies execute the same logical functions. The user will note that the raster plots practically never repeats, differently from the synfire simulations.

The program creates new delay relations every time it is executed. Note that the “shape” of each assembly is maintained because the delay relation persists during all the simulation.

When working with polychronous groups the stochastic nature of the logic functions performed by the assemblies become more evident. The user can see that some neurons may failure, but the functionality, the overall shape of the assemblies remain the same. Some neuroid types (e.g. 4, 14) cause the simulation to create raster plots with “burst-like” shapes.

The raster plots are much more realistic, more biologically plausible, although the synaptic weights must be more fine-tuned. Probably this fine-tuning is executed by plasticity mechanisms, which is under investigation and will be the subject of an incoming tutorial.