FNS user manual

Website: http://www.fnsneuralsimulator.org
GitHub: http://github.com/fnsneuralsimulator
Reference paper: "FNS: an event-driven spiking neural network simulator based on the LIFL neuron model" (2020, under review); arXiv link

Contents

1. Software installation **1**
- 1.1 Regular installation 1
- 1.2 Run using Docker 1

2. Simulation process **2**
- 2.1 Simulation procedure 2
- 2.2 Visualization of neural activity 3
  - 2.2.1 MATLAB 4
  - 2.2.2 Python 4
  - 2.2.3 Animation with Gephi 5

3. How to design a simulation **6**
- 3.1 Experiment command 6
  - 3.1.1 Commands for the Docker version of FNS 8
- 3.2 An in-depth look on FNS files, folders and fields. 8
- 3.3 Simulation issues 13

4. Simulation examples **15**

References **16**

1. Software installation

1.1 Regular installation

1. Install *Java (SE) JDK*[1] from the JAVA official page. Add the *bin* directory of *JDK* to the system environment variable PATH, and set the JAVA_HOME environment variable to point to the *jdk* folder.
2. Download the FNS package and unzip it.
3. That's it! You are ready to use FNS.

Optional. If you want to re-compile the software for some reasons (e.g., testing purposes), please follow the steps below:

- download and install MAVEN. It can be done from the official page
- As already done for *Java JDK*, add also the *bin* directory of MAVEN to the PATH environment variable. For Linux users Maven can be also downloaded from the official distribution repository;
- open *powershell* and go to the FNS root;
- run the following command (please keep your computer connected to the internet):

```
PS C:\FNS_folder> .\compile.bat
```

Linux users can re-compile by running:

```
~/FNS_folder$ ./compile
```

NOTE: In general, all the commands shown in this guide can be used in Linux with the exception of substituting the slash with the backslash. The user will be informed about possible differences between the commands for the two operating systems.

1.2 Run using Docker

Alternatively, you can run FNS with Docker, using the public Docker Hub image.
If you do not have Docker installed, please follow the Docker installation instructions for your system here: https://docs.docker.com/install/
Please find the commands to execute the Docker version of FNS in Par.3.1.1.

[1] we recommend version 11 onwards.

2. Simulation process

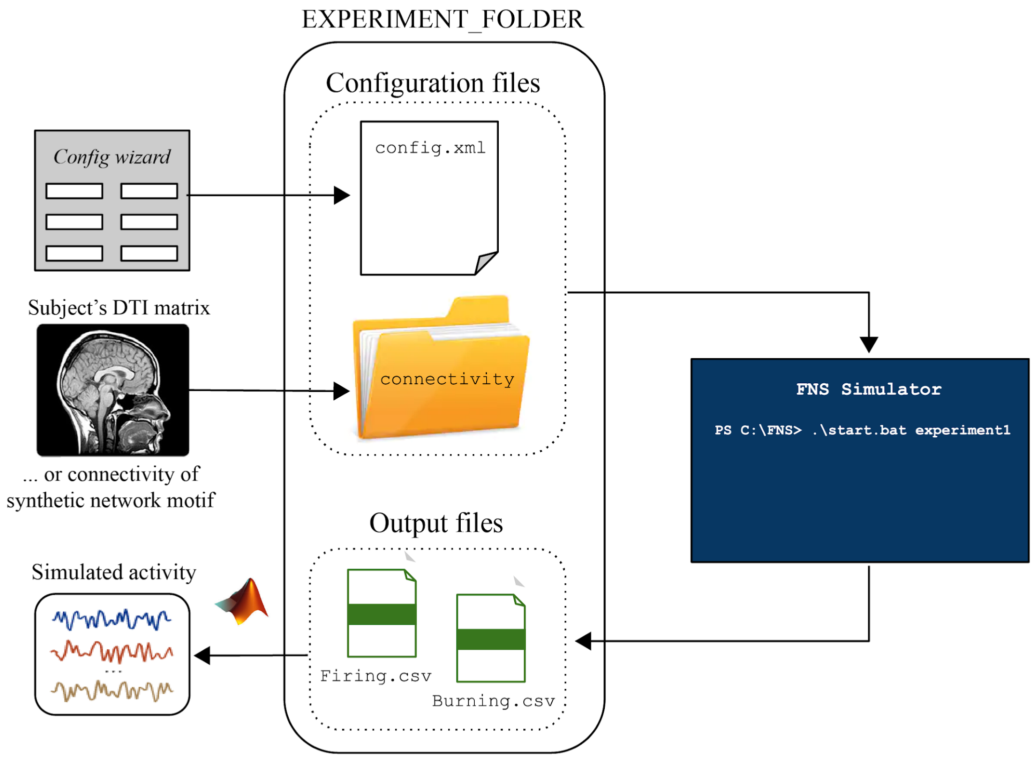


Fig.1: Three steps in order to obtain simulated neural activity. A) Preparation of the input data: config.xml file (manually or through the *Config wizard* online tool), and connectivity folder (real brain data extracted using DTI technique, or artificial network motifs); B) Simulation through FNS; C) Reconstruction of the electrophysiological-like signals using the FNS output files (**`Firing.csv`** and/or **`Burning.csv`**, introduced in Par 2.1), using the matlab scripts present in the FNS GitHub repository.

2.1 Simulation procedure

a. To launch a new simulation, the user has to type the so-called *experiment command* from the FNS folder. The *experiment command* has the following structure (for Windows and Linux, respectively):

```
PS C:\FNS_folder> .\start.bat [SIMULATION_FOLDER\EXPERIMENT] [SWITCHES]
```

```
~/FNS_folder$ ./start [SIMULATION_FOLDER/EXPERIMENT] [SWITCHES]
```

where **`[SIMULATION_FOLDER]`** is the folder which contains the simulation packages, and **`[EXPERIMENT]`** is the package which contains the set of *configuration files* for a single simulation (i.e., the file **`config.xml`** and the folder **`connectivity`**).
Then press *enter* to start the simulation.
The reader can find a detailed description of the *experiment commands'* fields in Par.3.

b. During the simulation, FNS displays 3 sets of simulation data: *initialization*, *execution*, *simulation stats*:

   Initialization.
   On the basis of the file `config.xml` and those present in the folder `connectivity`, *nodes* (neuronal populations, or regions) and *edges* (fibre tracts) of the network are created. Inner states of all neurons are initialized to random and subthreshold values (i.e., uniformly distributed between 0 and 1).

   Execution.
   The program proceeds with the simulation of consecutive temporal slices, displaying information about the progress of the process.

   Simulation stats.
   At the end of the overall simulation the following data are shown to the user:
   - Overall duration of the simulation (in terms of execution time);
   - Time employed by the *initialization* procedures (extraction of data from the configuration files and synthesis of the structures).
   - Minimum tract length among all the inter-node connections (in mm);
   - Duration of the cycle time during the simulation (in terms of simulated time);
   - Total number of inter-node connections;
   - Number of missed fires, i.e., fires that have been discarded due to an unsuitable sizing of the BOP. Note that, since by default we set a *cycle time* slightly smaller than the BOP, missed fires are naturally avoided;
   - Curve *goodness*, referred to the gamma distribution generated for the *intra-node connection lengths* (just in case negative values are generated).

c. FNS stores information about the activity of the network in the folder `SIMULATION_FOLDER\EXPERIMENT\OUTPUT`. Network activity is organized in two .CSV files, one for the departing spikes (*firing events*) and one for the incoming spikes (*burning events*) [1], with reference to the node, or set of nodes considered:
   - `burning.csv` : contains the data of burning events (neurons participate as receiver)[2].
   - `firing.csv` : contains the data of firing events (neurons participate as emitter).

   Considering a node composed of *n* neurons [*0* to *n-1*], possible additional *external inputs* will be regarded as pertaining to the same node, with their *neuron id* starting from *n*.

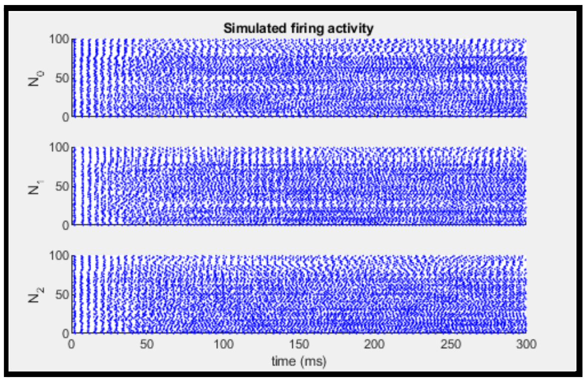
2.2 Visualization of neural activity

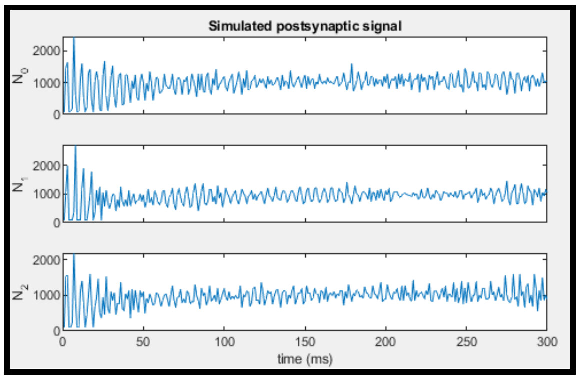
Using the output files firing.csv and burning.csv, the user can easily visualize both spiking activity and post-synaptic activity, respectively. Some visualization scripts are available in a proper GitHub folder. Moreover, we allow the user to explore network structure and activity in Gephi - specialized graph visualization software.

We introduce some examples below.

[2] Note that in the `burning.csv` file not all the burning times are listed in increasing time order, due to the specific BOP-based technique adopted (see [1] for details).

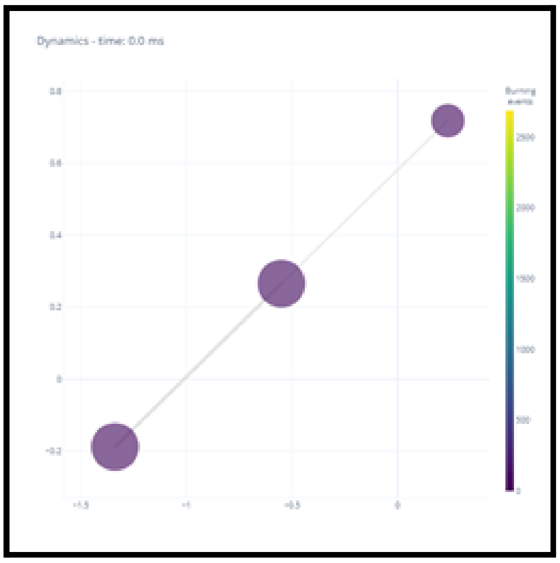
2.2.1 MATLAB

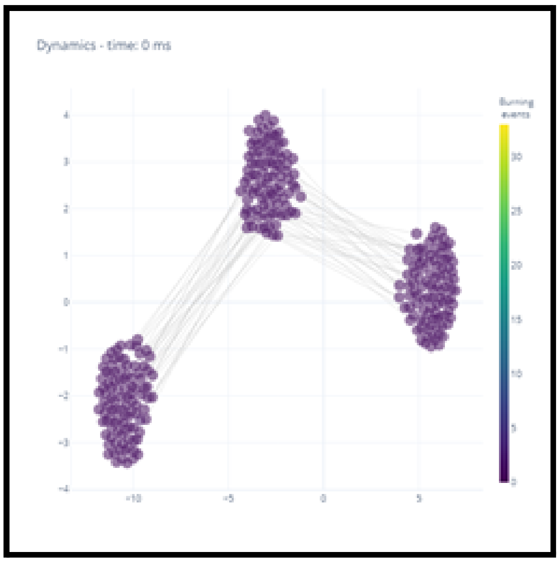




The user can easily visualize both *raster plot* and *local field potential*, evoked at the specified nodes using proper visualization scripts (`FNS_spiking.m` and `FNS_postsynaptic.m`, respectively).

2.2.2 Python



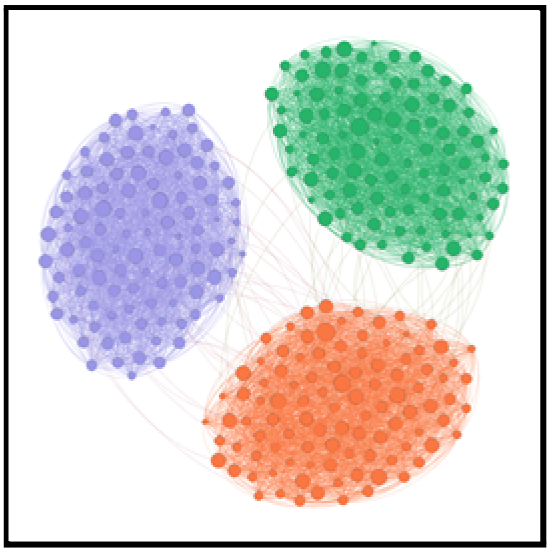


FNSvisualization.py is an option if you want a structural based visualization: plot network's activity over time with a theoretical spatial distribution of nodes. Choose between two levels of analysis neuronal and regional (averaging neurons' activity).

Specific Python modules have to be installed in order to be able to run the script[3], which can be installed directly through the execution of `pip install -r requirements.txt`

[3] The modules needed for the execution of `FNSvisualization.py` are: *python-igraph* (≥ 0.8.2), *pandas* (≥ 1.1.0), *plotly* (≥ 4.9.0), *numpy* (≥ 1.15.4), *opencv-python* (≥ 4.4.0.42) *requests* (>=2.24.0), *fsspec* (≥ 0.8.4)

2.2.3 Animation with Gephi



To use Gephi software to explore network structure and activity dynamics: export FNS data in Gephi compliant files[4], then import them in Gephi (*Data Laboratory section -> Nodes|Edges -> Import Spreadsheet*) to create the network, visualize, analyze and animate it.

Once data is imported, you can go to the *Overview* tab and visualize nodes. Apply a layout (e.g., *Fruchterman Reingold*) and press *run* to spread the nodes. Then customize the appearance in the upper left box. To animate the data, choose nodes' colour by ranking (attribute: *events*), click on the button aside *apply* to choose *Auto-apply*; finally, tap on *Enable Timeline* at the bottom of the screen, *shorten time window* and *play*.

[4] We are working on an option for FNS (I.e. the switch "-g") to generate these files directly as output of the simulation. By now, we provide a python script (I.e. `gephiConvert.py`) to transform FNS matlab compliant output (obtained with the switch **-m**) into two Gephi compliant files: `gephi_nodes.csv` and `gephi_edges.csv`.

3. How to design a simulation

3.1 Experiment command

As described in Par.2, the *experiment command* is articulated as follows:

- The starter `.\start.bat` (mandatory);
- The `SIMULATION_FOLDER\EXPERIMENT` path (mandatory);
- The `SWITCH(ES)` (optional).

The `EXPERIMENT` folder contains the input files (neuroanatomy of the network), and is where output files will be collected. It has to be structured as follows:

1. subfolder `CONNECTIVITY`, containing all the parameters to describe the network edges (Fig.2, in blue), containing the following files:

    - `Ne_xn_ratio.txt` (mandatory)
    - `conn_type.txt` (mandatory)
    - `mu_lambda.txt` (mandatory)
    - `mu_omega.txt` (mandatory)
    - `alpha_lambda.txt` (optional)
    - `sigma_omega.txt` (optional)

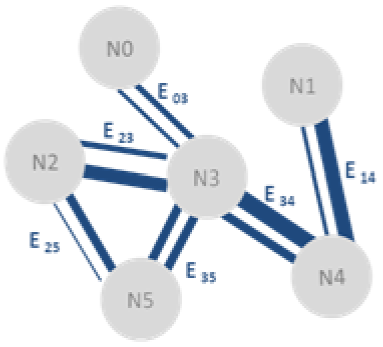


Fig.2: Scheme for the configuration of network edges (in blue). For simplicity in the figure the term E_{ab} indicates both the edge from *a* to *b* and that from *b* to *a*. Nevertheless, in FNS edges are, in general, non-symmetrical.

2. file `config.xml`, containing all the parameters of the network nodes (Fig.3, in blue), and some global parameters. The file config.xml can be generated by the user, or obtained through the dedicated config wizard.

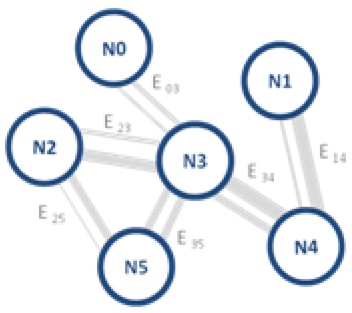


Fig.3: Scheme for the configuration of network nodes (in blue).

Finally, the `SWITCH(ES)`, which can be one or more among the following:

- `-f` enables *faster algorithms* at different levels, in return for some approximations (i.e., plasticity exponentials, underthreshold decay exponential, etc.);
- `-n` followed by the *list of node of interests* (NOI) for which to store the output data (i.e., the firing events for which the declared nodes are sender, and the burning events for which the declared nodes are receiver). If this switch is not present, the entire set of nodes will be considered for the generation of output data. The reader can find in Par. 3.2. the method implemented in FNS to specify a list of NOIs;
- `-m` provides as output a set of *Matlab-compliant CSV* output files, in addition to the output CSVs described in Par.2. The user can use the Matlab scripts present in the GitHub repository to obtain the electrophysiological-like signal from the CSV files.
- `-r` enables reduced CSV files, i.e., outputs that indicate only spiking events and inner states of the neurons. In this case, the output of the simulation will be `firing_r.csv` file (which will contain the columns 1,2,3,5 of the normal `firing.csv` file.and `burning_r.csv` (which will contain the columns 1,4,5,8 of the normal `burning.csv` file).
- `-g` in addition to the normal output, produces a *Gephi-compliant CSV* output file.

EXAMPLE:

For Windows and Linux, respectively:

```
PS C:\FNS_folder> .\start.bat [SIMULATION_FOLDER\EXPERIMENT] -m
```

```
~/FNS_folder$ ./start [SIMULATION_FOLDER/EXPERIMENT] -m
```

A description of the switches can be obtained by FNS by calling the help, i.e., digiting `.\start.bat -h` or `./start -h` (for Windows and Linux, respectively)

Before to start the first simulation, the reader is advised to take note of the simulation issues described in Par 3.3.

3.1.1 Commands for the Docker version of FNS

To run FNS with Docker, you can use the public Docker Hub image.
Please navigate the terminal until the FNS folder (where you placed the `[SIMULATION_FOLDER]`) and type the following command (consider the prefix `sudo` for Linux privileges):

```
docker run --rm -v
$(pwd)/[SIMULATION_FOLDER]:/usr/local/fns/[SIMULATION_FOLDER] -it -e
JAVA_OPTS="" --name fns fnsneuralsimulator/fns-simulator:latest fns
[SIMULATION_FOLDER/EXPERIMENT][SWITCHES]
```

Some useful notes:

- replace `-it` with `-d` if you prefer to detach and run FNS in the background;
- specify the field `JAVA_OPTS` in case you need to modify the Java heap size.

To make sure you are using the latest version of Docker, type:

```
docker pull fnsneuralsimulator/fns-simulator:latest
```

3.2 An in-depth look on FNS files, folders and fields.

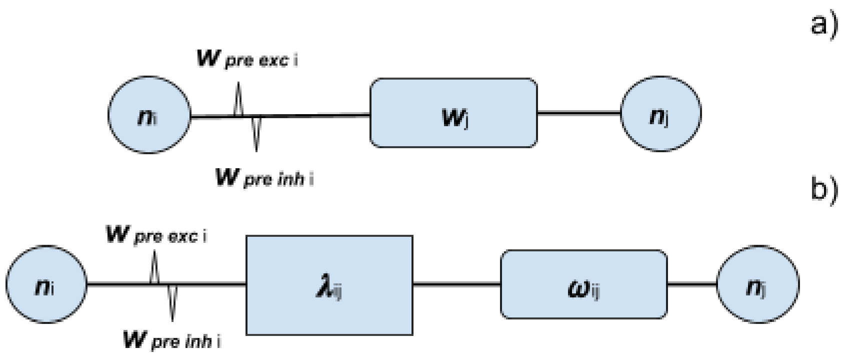


Fig.4: Scheme of intra-node (a) and inter-node (b) connection between two neurons.

a. File `config.xml`:

Within this file we find the section **<fns_config>**. At this level the user can define the global parameters, i.e., those parameters referred to all network nodes. Such section consists of the following fields:

Field name	Argument	Meaning
<stop>	Number (decimal, positive)	Duration of the simulation (in ms).
<avg_neuronal_signal_speed>	Number (decimal, positive)	Axonal conduction speed (in m/s).
<serialize_after>	Number (integer, positive)	Update period of output files (in terms of firing events).
<lif>	*true, false*	Neuron model adopted (LIFL or LIF)
<exp_decay>	*true, false*	Type of underthreshold behavior adopted (exponential or linear).
<glob_n>	Number (integer, positive)	Number of neurons.
<glob_rewiring_P>	Number (decimal): [*0,1*]	*Small world* rewiring probability.
<glob_k>	Number (even integer, positive)	Mean degree. Note that k has to be smaller than n.
<glob_R>	Number (decimal): [*0,1*]	Ratio between excitatory neurons and total number of neurons.
<glob_Bn>	Number (integer, positive)	Burst cardinality factor (number of spikes for each burst).
<glob_IBI>	Number (decimal, positive)	Inter-burst interval (in ms).
<glob_mu_w_exc>	Number (decimal, positive)	Intra-node post-synaptic weight (at the dendrites of excitatory targets).
<glob_mu_w_inh>	Number (decimal, positive). The program will automatically take the negative modulus of the value introduced.	Intra-node post-synaptic weight (at the dendrites of inhibitory targets).
<glob_sigma_w_exc>	Number (decimal, positive)	Standard deviation of the intra-node weights.
<glob_sigma_w_inh>	Number (decimal, positive)	Standard deviation of the intra-node weights.
<glob_w_pre_exc>	Number (decimal, positive)	Pre-synaptic excitatory amplitude (at the output of excitatory senders).

<glob_w_pre_inh>	Number (decimal). The program will automatically take the negative modulus of the value introduced.	Pre-synaptic inhibitory amplitude (at the output of inhibitory senders
<glob_external_inputs_number>	Number (integer, positive)	Number of external inputs (EIs). Each external neuron is associated to the related node, using a sequential wiring order (i.e., $EI_1(N_1)$ -> $n_1(N_1)$, $EI_2(N_1)$ -> $n_2(N_1)$,).
<glob_external_inputs_outdegree>	Number (integer, positive)	Number of target neurons associated to each EI.
<glob_external_inputs_type>	Number (integer): {*0;1;2*}	Type of external inputs (0=Poisson distribution; 1=constant spike train; 2=noise).
<glob_external_inputs_time_offset >	Number (decimal, positive)	Start of the external stimulation (in ms).
<glob_external_inputs_timestep>	Number (integer, positive)	Firing interval between spikes generated by the same EI (in ms). In case of noise and Poisson-distributed input, it has to be intended as the mean interval between two spikes.
<glob_external_inputs_fireduration>	Number (integer, positive)	Stop of the external stimulation (in ms).
<glob_external_inputs_amplitude>	Number (decimal)	External input amplitude.
<glob_plasticity>	*true, false*	Plasticity on/off.
<glob_plasticity_eta_plus	Number (decimal: [*0,1*]	LTP learning rate.
<glob_plasticity_eta_minus>	Number (decimal): [*0,1*]	LTD learning rate.
<glob_plasticity_tau_plus>	Number (decimal, positive)	LTP decay constant.
<glob_plasticity_tau_minus	Number (decimal, positive)	LTD decay constant.
<glob_plasticity_to>	Number (decimal, positive)	STDP timeout constant.
<glob_w_max>	Number (decimal, positive)	Upper bound for the postsynaptic weights

Note that the program rectifies numbers that are not inserted in the required format (removal of fractional digits where 'integer' numbers are required. Where 'even integer' numbers are required the program approximates to the previous admitted number.

The **<fns_config>** section also includes two subsections:

- the **<global_neuron_manager>** subsection, where the user can specify the neuron parameters of the generic network node:

Field name	**Argument**	**Meaning**
<D_exc>	Number (decimal): [*0,1*]	Decay constant for excitatory neurons (ms^{-1}). *
<D_inh>	Number (decimal): [*0,1*]	Decay constant for inhibitory neurons (ms^{-1}). *
<c>	Number (decimal, positive)	Threshold constant.
<t_arp>	Number (decimal, positive)	Absolute refractory period (ms).
<a>	Number (decimal)	Latency curve center distance (LIFL neuron constant set to 1 by default, see [1]).
<b>	Number (decimal)	Latency curve x-axis intersection (LIFL neuron constant set to 0 by default, see [1]).

* Considering how they are mathematically defined, in the case of exponential decay, the greater D, the slower the decay; in the case of linear decay, the greater D, the steeper the decay (see Fig.5, and read [1] for the mathematical relations).

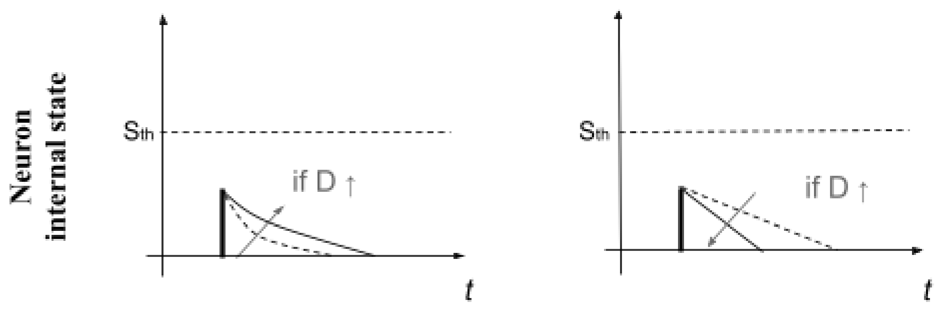


Fig.5: Effect of the decay constant increase on the underthreshold behavior, in the cases of exponential and linear decay. D>0.

This version of FNS is based on the basic *leaky integrate-and-fire with latency* (LIFL) configuration. The user can refer to the reference paper [1] for the mathematical aspects of this neuron model.

- the **<node>** subsection(s), where the user can redefine parameters for some single nodes. Such section(s) consist of the majority of the fields definable in the **<fns_config>** section (except for the fields <stop>, <avg_neuronal_signal_speed>, <lif> and <exp_decay>, which are the same for all the nodes of the simulated network). Such node-specific parameters can be defined by removing the "**<glob_ ... >**" prefix to the respective global parameters (e.g., <n>, <rewiring_P>, <k>, and so on). The user can redefine the node-specific neuron parameters through the subsection **<neuron_manager>**, to be included within the section **<node>**. Differently to the other parameters, for these fields the field names into the .xml file remains unchanged. Note that the neuron parameters of specific nodes are reconfigurable even partially, i.e. only a part of the parameters of the specific node can be specified (FNS will auto-complete the other local parameters with the values specified in the first **<neuron manager>** section).
The user can refer to the `config.xml` example file available in the GitHub repository for an easy understanding.

b. Subfolder `CONNECTIVITY`:

File name	**Content**
`Ne_xn_ratio.txt`	The value of the matrix *Ne_xn_ratio(R,C)* indicates the connections to be created from the neurons of node *R* to the neurons of node *C*, multiplied by the number of neurons of node *R* [5]. Types of both sender and receiver neurons are specified through the file *conn_type.txt* (described below).
`mu_omega.txt`	Mean of the inter-node post-synaptic weights.
`sigma_omega.txt` (optional)	Standard deviation of the inter-node post-synaptic weights.
`mu_lambda.txt`	Mean parameter of the inter-node lengths (mm).
`alpha_lambda.txt` (optional)	Shape parameter of the inter-node lengths.
`conn_type.txt`	Through this matrix the user can specify the types of sender and receiver neurons (excitatory/inhibitory/mixed). The generic R·C matrix element denotes the types of both sender (row index) and receiver (column index) neurons, by the following values: 0= from *exc&inh* to *exc&inh*;

[5] Edge cardinality is expressed as *Ne/xn* ratio in order to simplify the network scalability when parametric simulation approaches are adopted. For example: a value of 0.8 applied to an edge which transmitter node has 100 neurons of the type specified as sender in the file `conn_type.txt` (e.g., excitatory), will result in 80 (excitatory) connections directed to the receiver node.

	1= from *exc&inh* to *exc*; 2= from *exc&inh* to *inh*; 3= from *exc* to *exc&inh*; 4= from *exc* to *exc*; 5= from *exc* to *inh*; 6= from *inh* to *exc&inh*; 7= from *inh* to *exc*; 8= from *inh* to *inh*.

All these six files consist in adjacency matrices, which values specify parameters of the edge directed from raw index to column index.
Note that the files `sigma_omega.txt` and `alpha_lambda.txt` are optional; if not present, FNS will consider homogeneous inter-node lengths and weights, with the values present in `mu_omega.txt` and `mu_lambda.txt`, respectively.
FNS supports asymmetrical matrices, in order to be able to model different edge values for the two directions. Values pertaining to the main diagonals of all the matrices are not taken in account.
Note that the present version of the software does not support instantaneous (zero-length) inter-node connections.

c. Nodes list:
In the case that the user prefers to record in the output CSV files the data of a subset of NOIs, the list of nodes has to be specified in the *experiment command*, after the switch *-n* in squared brackets, as in the following example :

```
... [SIMULATION_FOLDER\EXPERIMENT] -n [3, 25, 13, 12]
```

In this case, the output files will be:
- `node_x-y-z-..._burning.csv`, which contains the data of burning events in which neurons pertaining to selected NOIs (x,y,z, ...) participate as receivers.
- `node_x-y-z-..._firing.csv`, which contains the data of firing events in which neurons pertaining to selected NOIs (x,y,z, ...) participate as transmitters.

Alternatively, if we are interested in the events pertaining to all nodes, is not necessary to specify the list of nodes: FNS will record the activity of all network nodes for default.

d. Output files:
After each experiment the output files will be stored into the folder `[EXPERIMENT/output]`. This folder will be automatically generated at the first simulation.

3.3 Simulation issues

a. FNS allows to perform batteries of sequential simulations without the assiduous intervention of the user, through the generation of a file named "`battery.bat`", structured as in the following example:

```
call .\start.bat [SIMULATION_FOLDER/EXPERIMENT_1] [SWITCHES]
call .\start.bat [SIMULATION_FOLDER/EXPERIMENT_2] [SWITCHES]
call ...
```

The battery can be launched by typing `.\battery.bat` as experiment command. For Windows and Linux the user can use, respectively:

```
PS C:\FNS_folder> .\battery.bat
```

```
~/FNS_folder$ ./battery
```

b. Before launching a simulation, please check that the files in the subfolder `CONNECTIVITY` are correctly formatted, e.g.:
- decimals have to be expressed with point and not with comma,
- do not use tabs but spaces.

c. If an out-of-memory error occurs during a simulation (JAVA *garbage collection*), please consider to increase the maximum heap size, using the environment variable _JAVA_OPTIONS (*-Xms* is used for the minimum heap size, *-Xmx* for the maximum heap size):
- Check the heapsize interval by typing (for Windows and Linux respectively):

```
java -XX:+PrintFlagsFinal -version | findstr /R /C:"HeapSize"
```

```
java -XX:+PrintFlagsFinal -version | grep HeapSize
```

- Increase the heapsize: insert as environment variable (user variable) “_JAVA_OPTIONS” and set, for example, `-Xms1G` and `-Xmx8G` as fields.

d. If a problem is encountered during an experiment, use Ctrl+c to block its execution in the powershell.

e. Regarding the external stimulation, when the field <external_inputs_outdegree> is set to 1, each external generator sends an independent process to its target network neuron, using one-to-one connections with the target neurons taken sequentially with respect to their neuron order; if <external_inputs_outdegree> is greater than 1, the target neurons are chosen randomly from those of the belonging node.

4. Simulation examples

Some network examples are present in the GitHub repository. Some configuration presets are now in the folder FNS-simulation_examples of the FNS GitHub repository, including:

- *single node*
- *resonance pair*, please refer to [2], [3];
- *dynamical relaying*, please refer to [4];

References

[1] Susi G, Garces P, Paracone E, Cristini A, Salerno M, Maestu F, Pereda E. *FNS: an event-driven spiking neural network simulator based on the LIFL neuron model* (submitted). PDF available at arXiv [1801.00864] arXiv link

[2] Maslennikov O.V and Nekorkin V.I, 2014. *Modular networks with delayed coupling: Synchronization and frequency control.* Phys. Rev. E 90.

[3] Gollo L, Mirasso C, Sporns O, Breakspear M, 2014. *Mechanisms of zero-lag synchronization in cortical motifs*. PLOS computational biology 10 (4).

[4] Vicente R, Gollo LL, Mirasso CR, Fischer I, Pipa G, 2008. *Dynamical relaying can yield zero time lag neuronal synchrony despite long conduction delays*. Proceedings of the National Academy of Sciences USA 105 (44).