Chekireb-Pack Antoine
Rivet Victoria
Rodriguez Valentin
Eglin Arthur

NEURON NETWORK

FROM BIOLOGY TO INFORMATIC MODELISATION OR HOW TO MODEL A NEURON NETWORK

EPFL

About the program

We wish to model a neuronal network to study its behaviour, the interactions between its neurons and its evolution in time. The aim of this program is to simulate a neuronal network based on a simplified model of the cortical neurons made by Eugene Izhikevich. Each neuron is represented by two time-depending functions, one for the membrane potential and one for the relaxation variable:

$$\frac{dv}{dt}(t) = 0.04v^2(t) + 5v(t) + 140 - u(t) + I(t), \qquad \frac{du}{dt}(t) = a\big(bv(t) - u(t)\big), \qquad (1)$$

When v(t) is bigger than 30mV, the neuron is in firing state and the program then sets the values as followed:

$$v(t) = c, \qquad u(t) = u(t) + d \qquad (2)$$

The cellular properties of a neuron are characterized by a type and four parameters:

- **a** – the time scale of the recovery variable **u**.
- **b** – the sensitivity of the recovery variable **u** to the subthreshold fluctuations of the membrane potential **v.**
- **c** – the after-spike reset value of the membrane potential **v** caused by the fast high-threshold K^+ conductance.
- **d** – the after-spike reset of the recovery variable **u** caused by slow high-threshold Na^+ and K^+ conductance.

Inhibitory types:

- Fast spiking – **FS**
- Low threshold spiking – **LTS**

Excitatory types:

- Regular spiking – **RS**
- Intrinsically bursting – **IB**
- Chattering – **CH**

Connections between neurons are created randomly and each neuron receives inputs of $d(n) \sim Poisson(\lambda)$ other neurons. Each link has an intensity $I \sim Unif(0, 2L)$. Links and their intensity are chosen at the beginning of the simulation and do not change overtime.

The synaptic stream $I(t)$ is the sum of 3 terms: the thalamic input, the excitatory stream, and the inhibitor stream. It is calculated as follows, with M_E the number of linked excitatory neurons in firing state and M_I the number of linked inhibitory neurons in firing state, l is the strength of the link:

$$I_n(t) = wJ_n(t) + 0.5 \sum_{m \in M_E(n,t)} l(n,m) - \sum_{m \in M_I(n,t)} l(n,m), \qquad (3)$$

The thalamic input $J_n(t) \sim Norm(0,1)$ act as an external noise, with $w = 2$ if the neuron is inhibitory and $w = 5$ if the neuron is excitatory.

The program starts with all the potentials $v = -65mV$ and $u = bv$ and makes these values evolve using equations (1) and (2). At every time-step, the stream arriving to each neuron is computed with the bond strength of each firing neuron linked to this neuron.

The following parameters can be chosen by the user:

- The **mode** of the simulation (constant model, basic model, over dispersed model). By default, the program will choose the basic model (Izhikevich model).
- A parameter **delta** (if entered, gives a more realistic – uniform – distribution of the parameters **a, b, c, d**)
- The **name** of the output files
- The **number** of neurons
- The **time** of the simulation (total number of time-steps)
- The mean number of connections for a neuron (**connectivity**)
- The mean bond strength between neurons (connection **intensity**)
- The **proportions** of each neuron type or simply the proportion of excitatory neurons

The output files of the program are the following:

- _spikes → A chart with **N columns** (neuron number) and **T lines** (time) representing the neurons in firing mode by a 1 and neuron not in firing mode by a 0 at every time-step.
- _parameters → A chart with the values **a, b, c, d, neuron type, degree** (number of linked neurons) and **valence** ($= 0.5 \sum_{m \in M_E(n,t)} l(n,m) - \sum_{m \in M_I(n,t)} l(n,m)$, from equation (3))
- _sample_neurons → A chart with the values of **v(t)**, **u(t)** and **I(t)** for one neuron of each type at every time-step.

Compilation and launch of the program

In order to use the program, the user must clone the git repository using the following command on the terminal:

```
git clone https://gitlab.epfl.ch/sv_cpp_projects/team_33.git
```

Then to build and compile the program, the user must use the CMake tool with the following commands in the repository created by the git clone command:

rm -rf build	→ removes an eventual old build repository
mkdir build	→ creates a new build repository (for the compilation and output files)
mkdir doc	→ creates a new doc repository (for the documentation files)
cd build	→ enters into the build repository
cmake ..	→ creates the necessary CMake files based on the CMakeLists.txt
make	→ compile the different files of the program
make doc	→ generates the Doxygen documentation (https://www.doxygen.nl/index.html)
make test	→ runs the different unit tests of the program (Google tests – C++ unit testing library: https://github.com/google/googletest)

To launch the program, the user has several possibilities. Each must be entered in the build repository:

- Run the program without parameters input and be guided by the user-friendly interface (this will run a simplified version of the program)
- Run the program with parameters input following the TCLAP help with the command:
  ./Neurons -h
  ex : ./Neurons -M O -d 0.1 -O test10000 -T 'IB:0.2,LTS:0.4,CH:0.2' -I 10 -C 40 -t 500 -N 10000

The above command will simulate a network with 10'000 neurons for 500 milliseconds in the over dispersed mode with the uniform distribution mode of the **a, b, c, d** parameter. The results will be printed in three output-files detailed above.

To read the documentation, the user can open the html documentation file, by executing the following command (in the build repository):

- **xdg-open ../doc/html/index.html** (for Linux Ubuntu)
- **open ../doc/html/index.html** (for Mac OS)
- **"..\doc\html index.html"** (for Windows)

Otherwise, the user can manually open the file **index.html** in the html repository of the doc repository.

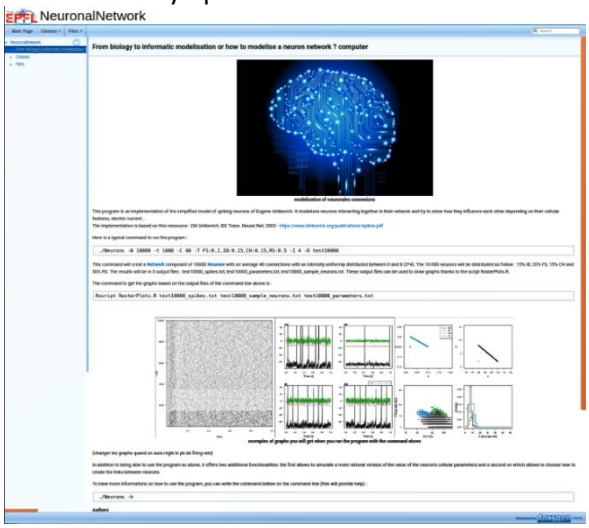

Figure 1 : Doxygen Documentation

The unit tests were designed for all non-trivial methods of each class. They check the methods' behaviour and assure their correctness. One of the most challenging tests was the test for the dimension of the spike output file. The aim of the test was to check the dimension of the data file and the content of the data (it should only include 0 and 1, except for the first column which enumerates the time-steps). Our choice of implementation was to convert it into a matrix of int that we could test:

```
TEST(OutputFile, DimensionCheck)
{
    Simulation simulation;
    simulation.run();
    std::string name(_OUTFILE_NAME_);
    name+="_spikes.txt";
    std::ifstream config_file(name);
    std::string item, line, key;
    std::vector<std::vector<int>> output_matrice;
    if (config_file.is_open()) {
        while (std::getline(config_file, line)) {
            std::stringstream line_outfile(line);
            std::getline(line_outfile,key, ' ');
            if(key.empty()) {
                continue;
            } else {
                std::vector<int> output_line;
                for (size_t n(0);(std::getline(line_outfile,item,' '))
                                        &&(n<line.size()); n++) {
                    EXPECT_TRUE(item=="0" or item=="1");
                    output_line.push_back(stoi(item));
                }
                output_matrice.push_back(output_line);
            }
        }
        config_file.close();
    }
    size_t simTime(_SIMULATION_TIME_);
    size_t nbNeurons(_NEURON_NUMBER_);
    EXPECT_EQ(output_matrice.size(),simTime);
    for(size_t i(0); i<output_matrice.size(); ++i) {
        EXPECT_EQ(output_matrice[i].size(), nbNeurons);
    }
}
```

Creates a default simulation and runs it.

Recovers the output of the default simulation in a ifstream.

Creates storage strings to read the output and a matrix to store the dimension of the output file.

Reads the output file, check the values of the output file data and translate the data into a matrix of int.

Compare the size of the matrix with the time of the simulation (one line per millisecond). Compare the size of a line of the matrix with the number of neurons (one column per neuron).

Figure 2 : Test DimensionCheck

To run the unit tests, the user can use the **make test** command in the terminal, but this only informs if tests are passed or not. To run the detailed tests and see the result of each unit test, the user must execute the test program by the following command: **./testAll**

Conception of the program

To model the neuronal network, we designed 3 main classes (Simulation, Network and Neurone) and two utilitarian ones (RandomNumbers and constants.h, containing SimulError). The main class are inspired by the reality of the anatomy of a brain (Simulation) with cortical areas (Network) composed of neurons (Neurone). This implementation choice allows a simpler conception of the program and a more realistic one, since every class has its own role in the program.

The graph below shows the interactions between one another.

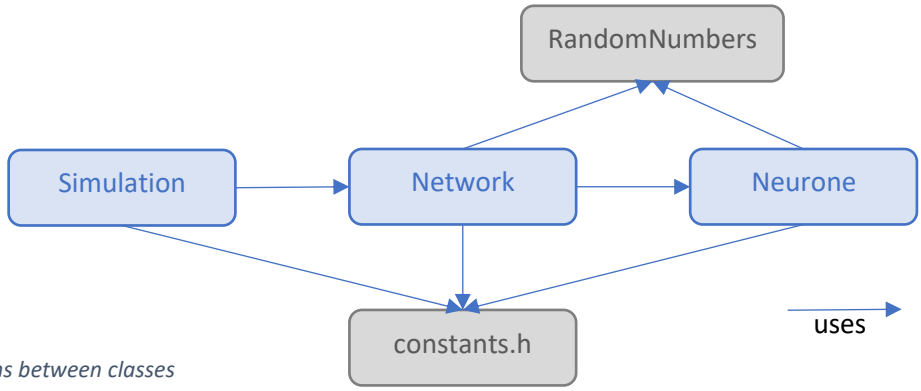

Figure 3 : Relations between classes

Simulation: Simulation is the driving class of the program; it manages the user's specified parameters and builds the **Network** of neurons. Then the run() method, brings life to the **Network** by incrementing the simulation time until it reaches the time limit. At each time-step, the neuronal network will be updated, and prints the results in 3 output files.

Network: The **Network** class represents the environment in which the neurons evolve and interact. It contains a set of neurons of different types that are linked randomly with one another. The dynamic of the Network is such that it updates every neuron at each time-step using the method update() to update its firing state. When the Network is initialized, all the Neurons and the links between them are created. During the simulation, neurons cannot die and links cannot be changed or created

Neurone: The Neurone class is the smallest unit-class of the program; it gives a simple model of a neuron. A neuron is represented by a set of parameters, some specifically defined for each type:

- The membrane potential → v(t)
- The relaxation variable → u(t)
- The cellular properties → a, b, c, d
- The type → FS, RS, LTS, CH, IB
- The state of action → firing / not firing

It also contains a list of every interaction that other neurons have with it.

Random: The Random class is a utility class used to randomly generate values for the different classes of the program. It can return values based on the following distribution: uniform (double), normal, Poisson and exponential. Its algorithms are based on the C++ random library.

Constants.h: This file contains the general constants used throughout the program and the SimulError class, a base class for errors thrown in the program. It also contains the data-structures used in some classes.

Generated graphics

To better visualize the results of the simulation, the user can generate graphics with the data of the output-files (the following examples are obtained by the command line of page 2). There are 3 graphics that can be made, one for each output-file. To do this, the user must enter the following line in the terminal (in the build repository):

Rscript ../RasterPlots.R test10000_spikes.txt test10000_sample_neurons.txt test10000_parameters.txt

1) **test10000_spiles.txt** is a graph with the neurons as ordinate and the time as abscissa, where each neuron in firing state is marked with a black dot/stroke.

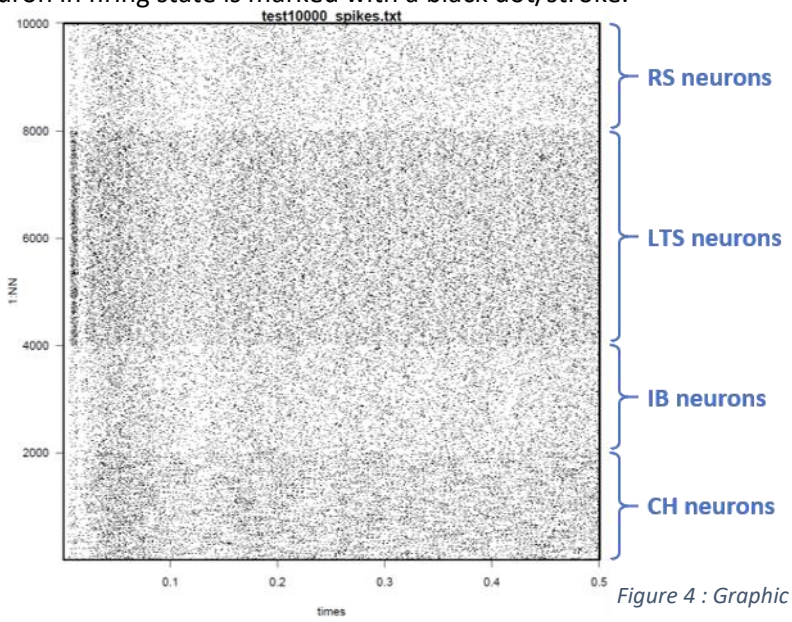

Figure 4 : Graphic 1 firing representation

2) **test10000_sample_neuron.txt** is a graph that shows the evolution a single neuron of each type during the entire simulation time. The relaxation variable is represented in **red**, the membrane potential in **black** and the synaptic stream in **green**. We see that for the inhibitory neurons (here LTS), the synaptic stream has a smaller variance because of a smaller w value than excitatory neurons (RS, IB, CH). The graphs shape depends on the network population. The more inhibitory neurons there are, the less neurons will be firing. Moreover, this graph allows to see the characteristic firing pattern of each neuron type. CH neurons can fire stereotypical burst of closely spaced spikes, which is represented by a sequence of vertical black lines. Whereas RS neurons are characterized by few spikes with short inter-spikes periods and then an increasing period. LTS neurons have a lower spiking threshold, therefore their firing rate is higher than the one of the other neuron types. This shows the reality near aspect of the program.

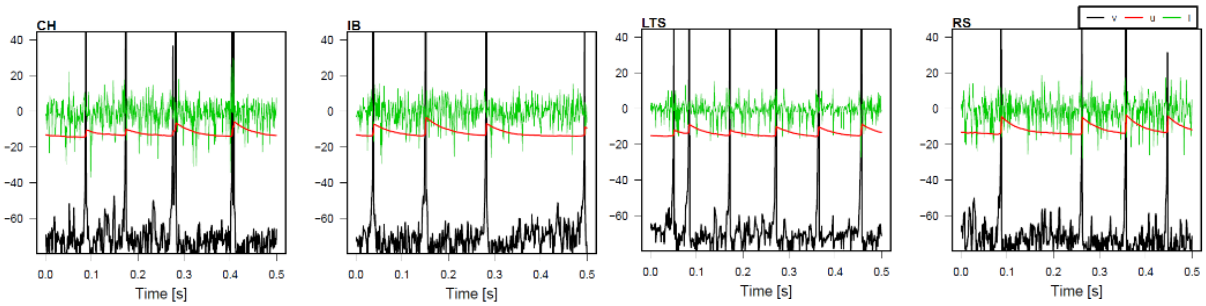

Figure 5 : Graphic 2 sample neuron

3) **test10000_parameters.txt** contains four graphics. The first two show the four parameters a, b, c, d expressed in function of each other for each type. In the example case, this shows the uniform distribution of the parameters with a noise coefficient between 1-**delta** and 1+**delta**.
The third shows the firing rate (number of spikes per seconds) in function of the coefficient of variation of the inter-spike interval. This shows how the inter-spike interval of each neuron type affects its firing rate. These two variables are inversely proportional, which means that the higher the inter-spike interval, the lower the firing rate. Most of the neurons are not often firing since the graph is mainly is the south-east quarter, showing high inter-spikes interval and thus low rate. This is mainly affected by the number of links each neuron receives; indeed, this parameter affects the most the firing rate, which correlates with the reality.
The fourth shows the statistic density of the firing rates of each type. It is a gaussian-type curve with mean and standard deviation depending on the parameters of the types. It shows the distribution of the neuron according to their firing rate. We can see that RS and IB neurons have approximatively all the same firing rate, the distribution is not spread, whereas the one of LTS and CH are more spread, showing that neurons have different behaviours.

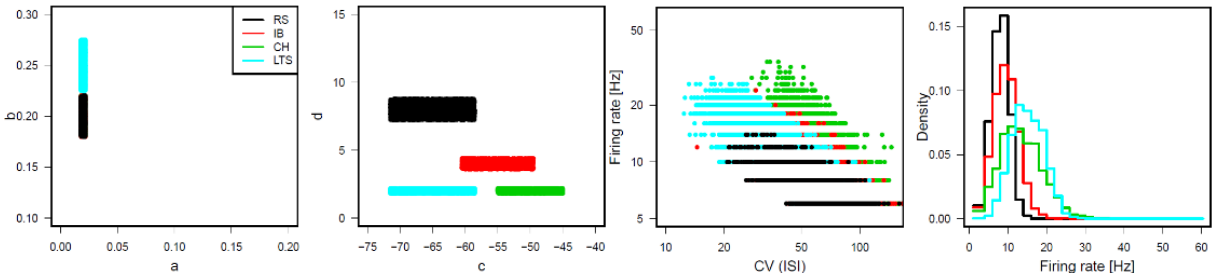

Figure 6 : Graphic 3 Neuron parameters

Discussion about the program features

First, we will show the effect of the different modes of the program on the connectivity patterns and what are the consequences. In order to do it, we will execute the following command line, each time with a different mode:

./Neurons -N 10000 -t 1000 -C 100 -I 4 -P 0.8 -M B or **O** or **C**

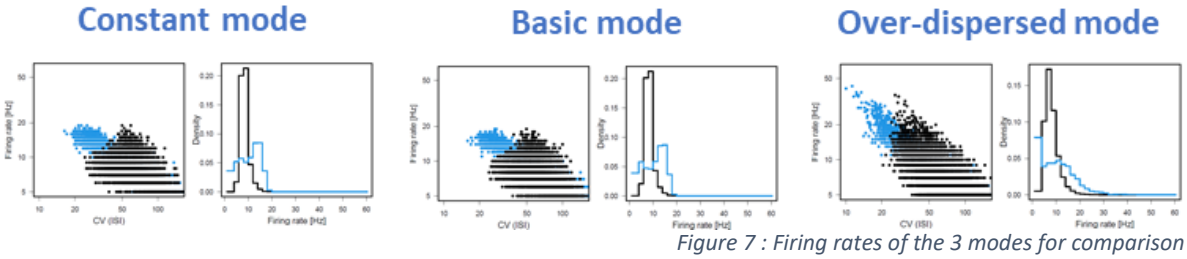

Figure 7 : Firing rates of the 3 modes for comparison

As explained in point 3) above, in a realistic simulation (with more excitatory neurons than inhibitory ones), the more connections the higher the firing rate. For the constant mode, each neuron got 100 connections, so its firing rate should be approximatively constant. For the basic mode, neurons received between 80 and 120 connections, the extreme connection values are very near the mean, so the firing rate should also be mostly constant. For the over-dispersed mode, neurons received approximatively between 4 and 300 connections, which is very scattered, so the firing rate should be very spread out. (note : number of connections can be found in the _parameters output file)

As we can see on the two graphics, the constant and the basic modes are very similar, with no great variation of the firing rate. The firing rate of the over-dispersed mode is very spread out. So, our assumptions above are verified.

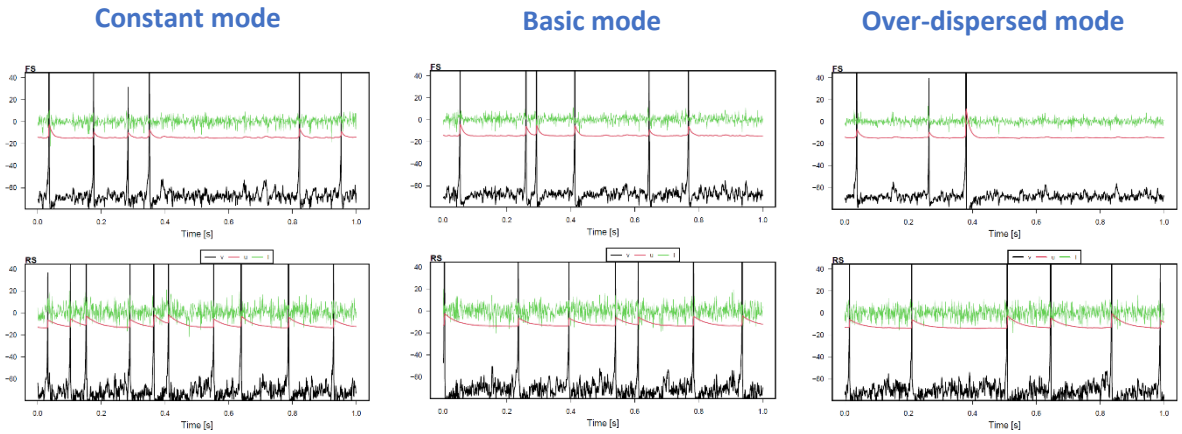

Figure 8 : Firing pattern for a sample in each mode

Even though modes constant and basic are similar, we can distinguish them by looking at the above graphs. Neurons FS and RS in basic mode have here respectively 84 and 88 connections, this explains why they are less often firing than neurons in constant mode. Finally, neurons in over-dispersed mode FS and RS have here respectively 33 and 47 connections, this low number of connections compared to the mean is possible only in this mode and explain the low firing rate compared to the 2 other modes.

Secondly, we will explain how the results of each neuron type depend on the parameters used in this model. As the results suggest (using the above graphs), inhibitory and excitatory neurons have different behaviours.

Primarily, we can clearly see that the excitatory neurons have a higher firing rate than the inhibitory neurons. It can be explained by the value of the thalamic input (noise) **w,** which is 5 for excitatory neurons and 2 for inhibitory ones. So, the synaptic current will be greater for excitatory neurons.

Moreover, the value of the cellular parameter **c** for RS is way lower than for FS so the reset value of the membrane potential for RS is closer to the firing threshold than for FS and will therefore take less time for RS to hit this threshold, thus they will fire more frequently.

We also observed, through trials and errors, that changing the user parameters will greatly change the output that we observe: for example, the number of connections **C** will greatly affect the firing rate of the neurons, more connections mean a higher firing rate. In fact, the firing rate can be influenced by each parameter since the model is very sensitive. Increasing the connectivity will increase the synaptic current so depending on the proportion of excitatory and inhibitory neurons, the firing rate will decrease (if many inhibitory neurons) or increase (if many excitatory neurons).

Also, the intensity of the connections will impact the rate: higher intensity means higher current, so we will be able to see the same effects as explained above.

The cellular parameters (**a**, **b**, **c** and **d**) have a great impact because they are determining terms in the evolution equation of the neurons (equation **(1)** and **(2)**).

Finally, the thalamic input can affect the firing rate too because it impacts the synaptic current. For example: if you change the amplitude of the thalamic noise by $J_n(t) \sim Norm(0,5)$, the firing rate will linearly change from 10 Hz to 50 Hz.

$J_n(t) \sim Norm(0,1)$ $J_n(t) \sim Norm(0,5)$

Figure 9 : Thalamic input changes comparison

References

Documentation

E. M. Izhikevich, "Simple model of spiking neurons," in *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569-1572, Nov. 2003, doi: 10.1109/TNN.2003.820440.

Figures

First page : https://www.numerikare.be/fr/actualites/e-health/l-intelligence-artificielle-aussi-efficace-que-le-diagnostic-des-medecins-que-retenir-de-la-meta-analyse-du-lancet-nbsp.html#
EPFL Logo : https://www.epfl.ch/about/overview/fr/identite/
Project Logo : https://www.shutterstock.com/fr/search/neuron+logo
Last page : https://healthitanalytics.com/features/what-is-deep-learning-and-how-will-it-change-healthcare