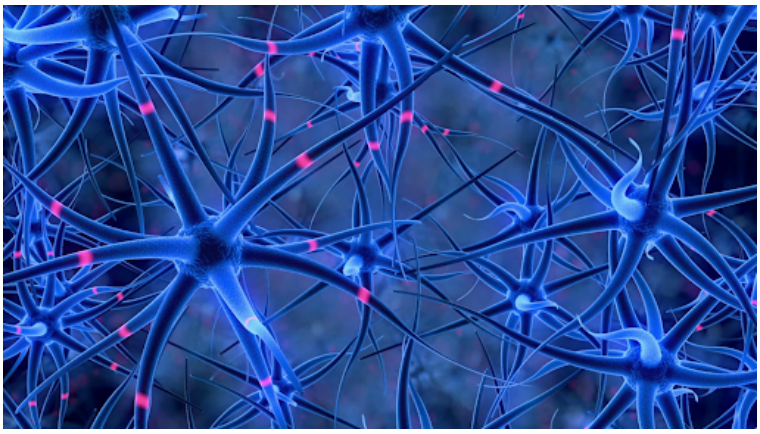
Modeling a Neuron Network

Project 2

Team22 :
Angelica Reitelli,
Aloïs Thomas,
Charlotte Coulon,
Laura Gonzalez

Autumn 2020



Contents

1 Aim of the program

The program is based on a simplified model of neurons by Eugene Izhikevich. This model reproduces spiking and bursting behavior of different types of cortical neurons (RS, IB, CH and LTS, FS). According to the Izhikevich model, one neuron has: 2 variables that are time-dependent: the membrane potential v and the recovery variable u a "firing" state when v reaches 30 mV, in this case u and v instantaneously reset to new values:

$$v(t) = c \tag{1}$$

$$u(t) = u(t) + d \tag{2}$$

The temporal evolution is described by two equations:

$$\frac{\mathrm{dv}}{\mathrm{dt}}(t) = 0.04v^2(t) + 5v(t) + 140 - u(t) + I(t) \tag{3}$$

$$\frac{\mathrm{du}}{\mathrm{dt}}(t) = a(bv(t) - u(t)) \tag{4}$$

The cellular properties of a neuron are characterized by four dimensionless parameters a, b, c and d in order to implement a large diversity of neuron types. The parameter a describes the time scale of the recovery variable u whereas the parameter b describes its sensitivity to the subthreshold fluctuations of the membrane potential v. Moreover, the parameter c describes the after-spike reset value of the membrane potential v caused by the fast high-threshold K+ conductances and finally the parameter d describes after-spike reset of the recovery variable u caused by slow high-threshold Na+ and K+ conductances.

This program will represent spiking and bursting behaviors of each type of neurons as presented in the simple model of Izhikevich below taken from the article of Izhikevich himself:

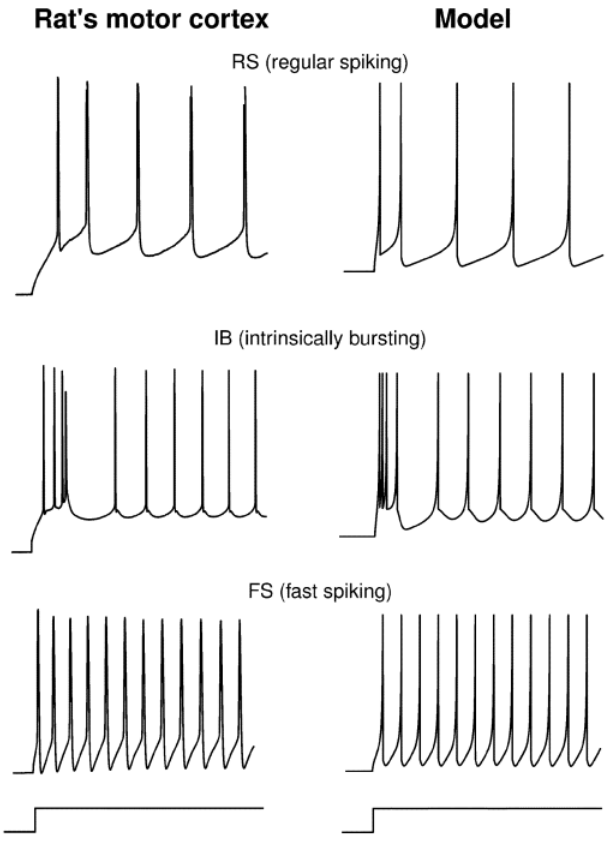
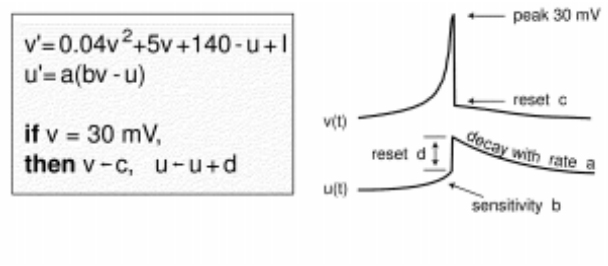
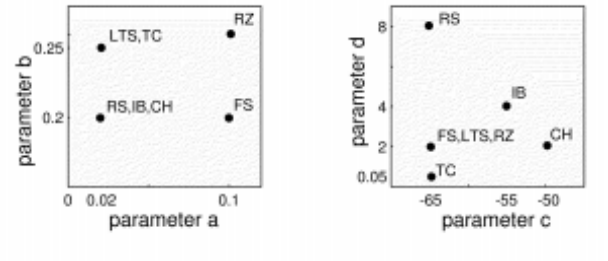


Figure 1: The simple model (1), (2) can reproduce firing patterns of neurons recorded from the rat's motor cortex.



(a) Evolution of the tension v and relaxation variable u



(b) parameters a,b,c,d

Figure 2: Each type of neuron shows a voltage response depending on specific values of the parameters a,b,c,d

The program simulates a neuron network that is composed of a large population of neurons that make random one-way-connections to each other. Each connection has a random strength that symbolizes the number of synapses in a real-life model. The neurons can either be 'inhibitory' (FS, LTS) or 'excitatory' (RS, IB, CH). The input produced by firing neurons to other connected neurons is computed by the synaptic current I(t) that is calculated at each time stamp.

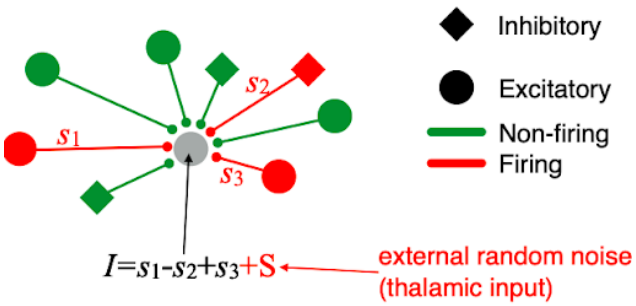


Figure 3: Image of a simplified representation of a neuron network taken from the course slides (slides week6, page n°25)

2 Design of the program

In order to conceive this program, we have created 5 classes: Neuron, NeuronNetwork, Simulation, Random and Error.

Simulation.h
This is the main class of the program. It constructs the neuron network, manages user inputs and produces all the outputs. To store the user's information we used the TCLAP library and to handle the possible errors we created the class Error described below. To produce the outputs, we created some methods to print the spikes, parameters and samples and finally a method to run the simulation.

NeuronNetwork.h
This class modelizes the neural network of the simulation. It has the vector of neurons as an attribute and

thus can iterate on it, calculate and set for each neuron the values that are dependent from the surrounding neurons such as the connections each neuron makes or the current I. Moreover, the dynamics of the simulation are generated by the method update which changes the values of the membrane potential and relaxation values solving a differential equation.

Neuron.h
This class represents every specific neuron in our simulation and groups together its characteristics. We find some fixed attributes such as its type, quality or cellular properties a, b, c, d. The set of connections this neuron forms is also defined here. Then, there are some variable attributes such as the membrane potential, the recovery variable or the firing state. The methods in this class are essentially getters and setters.

Error.h
This class is created to handle the errors of the program. It heritates from a subclass of std::exception and will produce a specific output depending on the type of error.

Random.h
This class is based on the standard random generators of c++. We used specifically the generator Mersenne twister *mt19937*. The methods produce some random values following an uniform, normal or poisson distribution.

Additional files
The "main.cpp" creates and runs a simulation. It catches the potential errors. It is also here where we create our only random generator Random* _RNG, externally declared in the Random class.
The "constants.h" file groups together the default parameters and description texts for the TCLAP. The error class is also written in this file.

3 Compilation and launch of the program

In order to make use of the program, the user has to clone the git repository using the following command on its terminal: git clone `https://gitlab.epfl.ch/sv_cpp_projects/team_22.git`

Then to build and compile the program these are the following commands that should be done in the terminal:

cd team_22
rm -rf build
mkdir build
cd build
cmake ..
make
make doc
make test

Make doc generates the Doxygen documentation of the program and make test allows to generate the unit testing library for C++, Google tests.

If the user wants to execute the program he can use TCLAP. The user can write the command *./Neurons -h* in the terminal to see the detail of each parameter he has to enter, this is the output that he will see:

```
USAGE:

   ./Neurons  [-D <double>] [-E] [-T <string>] [-L <double>] [-C <int>] [-P
              <double>] -N <int> -t <double> [--] [--version] [-h]


Where:

   -D <double>,  --delta <double>
     Noise of the parameters a,b,c,d

   -E,  --ext
     Use of the extension

   -T <string>,  --Proportions <string>
     Proportions of the different neurons (write RS:x , FS:y with x and y
     being the corresponding proportions )

   -L <double>,  --intensity_conn <double>
     Average intensity of the connections

   -C <int>,  --conn_mean <int>
     Average connectivity

   -P <double>,  --prop_exciter <double>
     Proportion of excitatory neurons

   -N <int>,  --nb_neur <int>
     (required)  Number of neurons to simulate

   -t <double>,  --duration_sim <double>
     (required)  Time of the simulation (in ms)

   --,  --ignore_rest
     Ignores the rest of the labeled arguments following this flag.

   --version
     Displays version information and exits.

   -h,  --help
     Displays usage information and exits.


   Simulation of a neuron network
```

Figure 4: *TCLAP : detail of each parameter*

A simple example of a complete TCLAP command is:
./Neurons -t 1000 -N 10000 -T "IB:0.15, FS:0.2, CH:0.15, RS:0.5" -L 6 -C 60

This command will generate three files in the folder **build** called *parameters.txt*, *sample_neurons.txt* and *spikes.txt*.

The following files are an example of what they should look like:

```
        IB.v    IB.u    IB.I    FS.v    FS.u    FS.I    CH.v    CH.u    CH.I    RS.v    RS.u    RS.I
1       -69.651 -13     -6.98078        -67.0225        -13.2147        -0.233437       -67.7842        -13     -1.66218        -62.6177        -13     -5.41529
2       -76.3306        -13.0186        -0.0497376      -69.2214        -13.2536        -0.159825       -71.1064        -13.0111        -0.675412       -70.8867
        -12.9905        -5.74246
3       -73.0737        -13.0636        8.82107 -70.3801        -13.3312        0.455927        -71.8895        -13.0353        1.99431 -75.8457        -13.0142
        8.87795
4       -64.5935        -13.0946
2.37451 -70.3624        -13.4238        -1.87955        -69.9898        -13.0622        -3.21743        -65.6719        -13.0573        0.11866
```

Figure 5: *sample_neurons.txt*

There should be 1000 lines like these in this file if t = 1000. This file shows for each time stamp the values of the potential v, the relaxation u and the synaptic current I of each type of neuron chosen.

```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 6: *spikes.txt*

This file contains 1000 lines and 10'001 columns. A 0 means that the neuron is not firing and 1 means that it is firing. At the beginning of each line the time stamp is written : 1, 2 ... 1000.

```
Type    a       b       c       d       Inhibitory      degree  valence
CH      0.02    0.2     -50     2       0       58      227.907
CH      0.02    0.2     -50     2       0       59      208.591
CH      0.02    0.2     -50     2       0       50      188.992
CH      0.02    0.2     -50     2       0       61      226.703
CH      0.02    0.2     -50     2       0       62      289.438
CH      0.02    0.2     -50     2       0       63      202.311
CH      0.02    0.2     -50     2       0       58      129.931
CH      0.02    0.2     -50     2       0       57      225.102
CH      0.02    0.2     -50     2       0       83      316.765
CH      0.02    0.2     -50     2       0       56      197.949
CH      0.02    0.2     -50     2       0       61      226.06
CH      0.02    0.2     -50     2       0       74      279.718
CH      0.02    0.2     -50     2       0       55      231.435
CH      0.02    0.2     -50     2       0       54      220.654
CH      0.02    0.2     -50     2       0       82      235.191
CH      0.02    0.2     -50     2       0       64      189.983
CH      0.02    0.2     -50     2       0       60      215.255
CH      0.02    0.2     -50     2       0       65      252.889
CH      0.02    0.2     -50     2       0       62      238.361
CH      0.02    0.2     -50     2       0       65      261.725
```

Figure 7: *parameters.txt*

This file should contain lines like these for each type of neuron that we want to compute (in this case: IB, FS, CH, RS). It shows the parameters of each neuron, if they are inhibitory (1) or not (0) as well as their degree (number of connections towards the neuron) and their valence (sum of intensities of these connections).

4 Outputs

If the user wants to see the evolution of the parameters or the spiking and bursting of neurons in function of time, he can enter the command:

Rscript ../res/RasterPlots.R spikes.txt sample_neurons.txt parameters.txt

This command will generate three pdf files called *Rplot001.pdf*, *Rplot002.pdf* and *Rplot003.pdf* in the folder **build**, here is an example of what the files should look like (if the command is the same as in the example in part.3). The file 1, represents each spike of neurons in function of time. One point signifies that a neuron was firing at this point in time. The file 2, represents the evolution of the potentials v, u and the current I in function of time for one neuron of each type. A firing state is represented by a black spike in the graph. The file 3, represents the evolution of the parameter a, b, c and d during the simulation. The two other graphs represent statistics concerning the neurons, the coefficient variation and the firing rate.

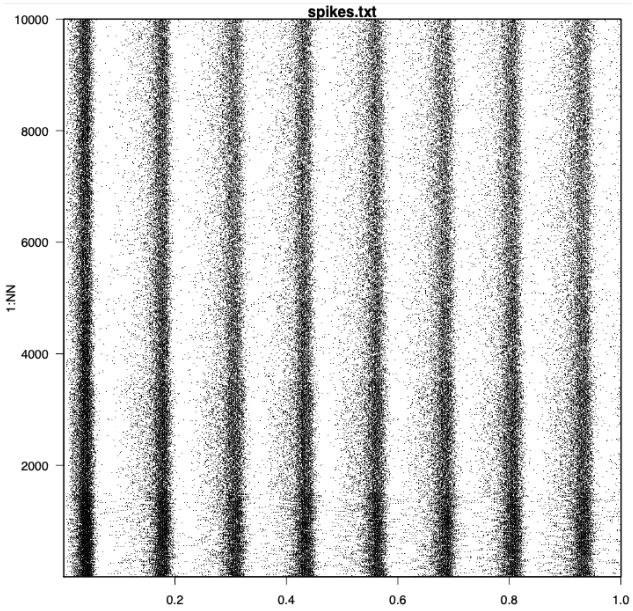


Figure 8: *File 1- Evolution of spikes in time*

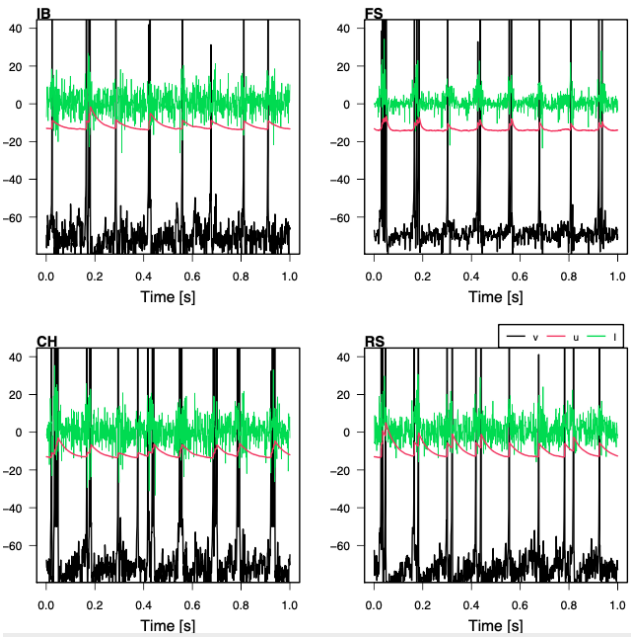


Figure 9: *File 2- Evolution of v,u,I in time for one neuron of each type*

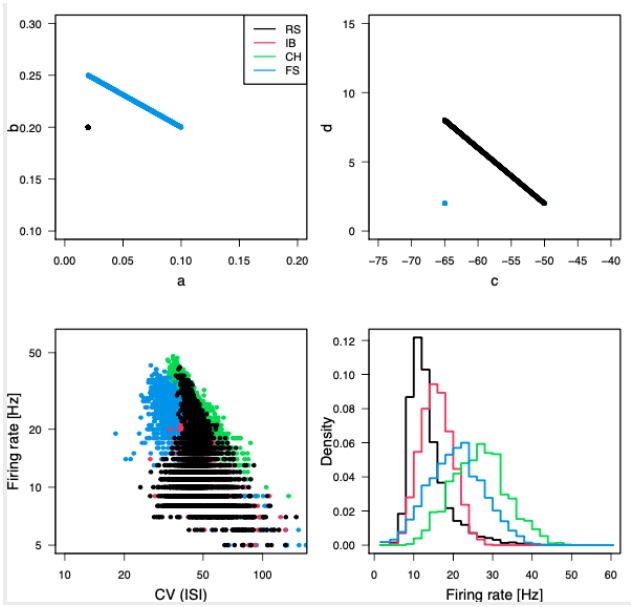


Figure 10: *File 3- Evolution of parameters a,b,c,d, statistics concerning the neurons, the coefficient variation and the firing rate*

5 Tests

We decided to create unit tests for all the functions in the class Random, Simulation, NeuronNetwork and Neuron if the functions exceeded three lines and were essential to the proper functioning of the model.

In order to run the tests the user has to write this command in the terminal :
./NeuronsTest
And this should be the output :

```
[==========] Running 9 tests from 4 test suites.
[----------] Global test environment set-up.
[----------] 1 test from Random
[ RUN      ] Random.uniform
[       OK ] Random.uniform (0 ms)
[----------] 1 test from Random (0 ms total)

[----------] 2 tests from Simulation
[ RUN      ] Simulation.cmdLine
[       OK ] Simulation.cmdLine (16 ms)
[ RUN      ] Simulation.test_read_proportions
[       OK ] Simulation.test_read_proportions (3 ms)
[----------] 2 tests from Simulation (19 ms total)

[----------] 4 tests from NeuronNetwork
[ RUN      ] NeuronNetwork.set_connections
[       OK ] NeuronNetwork.set_connections (1 ms)
[ RUN      ] NeuronNetwork.I
[       OK ] NeuronNetwork.I (0 ms)
[ RUN      ] NeuronNetwork.update
[       OK ] NeuronNetwork.update (0 ms)
[ RUN      ] NeuronNetwork.getValence
[       OK ] NeuronNetwork.getValence (0 ms)
[----------] 4 tests from NeuronNetwork (2 ms total)

[----------] 2 tests from Neuron
[ RUN      ] Neuron.checkConnection
[       OK ] Neuron.checkConnection (0 ms)
[ RUN      ] Neuron.setinitialattributs
[       OK ] Neuron.setinitialattributs (0 ms)
[----------] 2 tests from Neuron (0 ms total)

[----------] Global test environment tear-down
[==========] 9 tests from 4 test suites ran. (21 ms total)
[  PASSED  ] 9 tests.
```

Figure 11: *Units tests in our program : they all pass*

We would like to detail one test that was implemented:
NeuronNetwork.set_connections

```
TEST(NeuronNetwork, set_connections) {

    std::vector<Neuron*> neurons;

    for (size_t i(0); i < 40; ++i){
        neurons.push_back(new Neuron("FS", i));
    }


    double lambda(2);
    double intensity(3.0);
    NeuronNetwork N(neurons, lambda, intensity);
    double S(0);
    int nb_connections(0);
    int nb_neurons(0);

    N.set_connections();

    for (size_t i(0); i<neurons.size(); ++i){
        nb_neurons=nb_neurons+1;
        EXPECT_TRUE(!neurons[i]->getConnections().empty());

            for (auto j : neurons[i]->getConnections()){
                nb_connections=nb_connections+1;
                EXPECT_TRUE(j.first || neurons[i]->getId());
                EXPECT_LE(j.first, neurons.size()-1);
            }
            S+=nb_connections;
            nb_connections = 0;
    }

    double nb_connections_mean = S/nb_neurons;
    EXPECT_NEAR(nb_connections_mean, lambda, sqrt(lambda));

}
```

Figure 12: *Test in NeuronNetwork : setconnections*

Explanation of the test:

At first, a vector of 40 FS neurons are created (the type does not matter here since we are checking only the connections). Then, a NeuronNetwork is created using this vector with a mean connectivity as well as an intensity of connections (those values are chosen randomly). The method set_connections() is then called on this Neuron Network. The first test **EXPECT_TRUE** is done to verify that each neuron is connected to other ones. The second **EXPECT_TRUE** is used to verify that a neuron is not connected to itself. The **EXPECT_LE** is to verify that it does not have more connections than the number of neurons in the vector. Finally, we check with **EXPECT_NEAR** that the mean number of connections for each neuron is close to the mean connectivity chosen, using a confidence interval of +-sqrt(lambda) from the Poisson random variable that is used to compute the number of connections for each neuron.

6 Documentation

The documentation of the program has been done with Doxygen and can be executed via the terminal with the command: *make doc*. To access the documentation, the user has to find the file *index.html* in the folder **doc**. When he clicks on it a web page should open and this should be what it looks like:

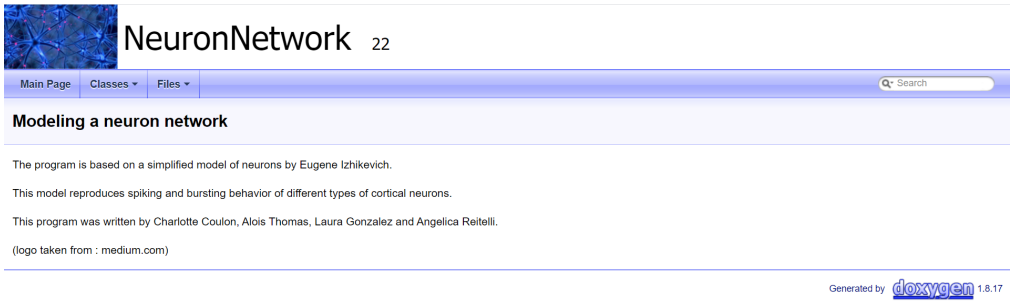


Figure 13: *Documentation : main page*

The user can browse the web page to access the details of each function of the program.

7 Extension

We decided to add an extension to our program. The extension provides parameter noise that can be controlled by the user. If the user wants to execute the program with the extension, the parameter -E in TCLAP has to be written in the command line. If the extension is activated then each parameter (a,b,c,d) of each type of neuron (RS, FS, IB, CH, LTS) is created with a noise r that is chosen at random uniformly with a tunable parameter delta chosen by the user in the command line with -D. This extension allows to implement a more rational model for all parameters of the neurons.

An example of a TCLAP command with the extension activated is : *./Neurons -t 1000 -N 10000 -T "IB:0.15, FS:0.2, CH:0.15, RS:0.5" -L 6 -C 60 -E -D 0.03*

The outputs should look like :

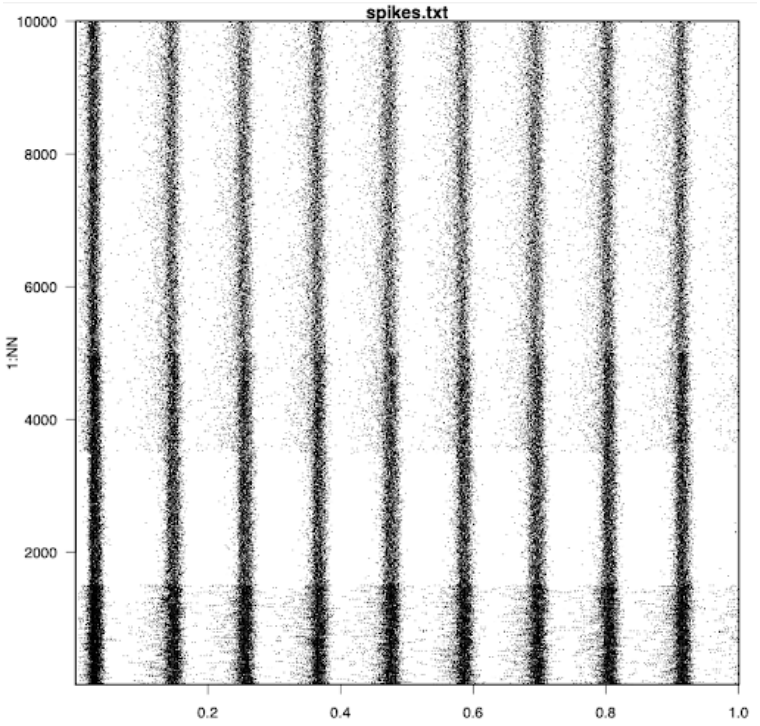


Figure 14: *File 1-Evolution of spikes in time*

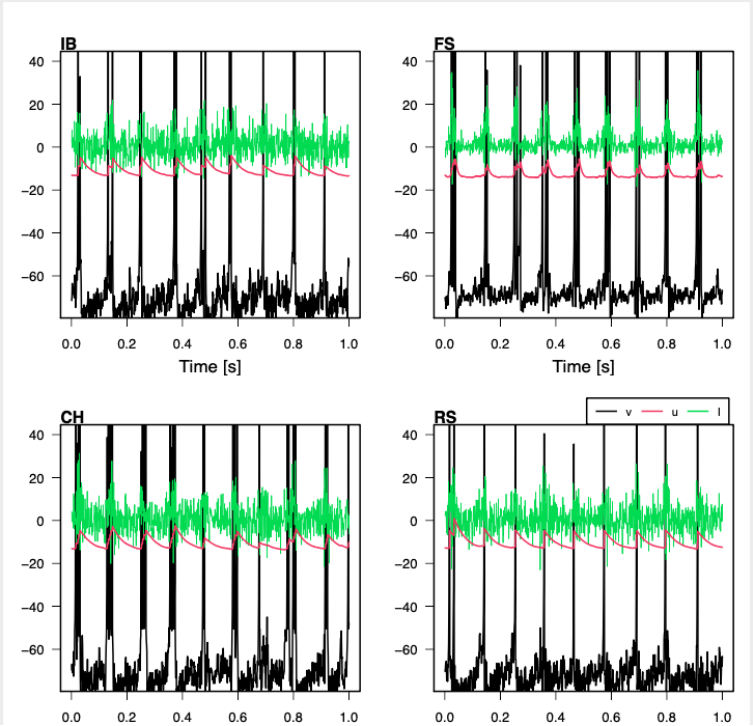


Figure 15: *File 2- Evolution of v,u,I in time for one neuron of each type*

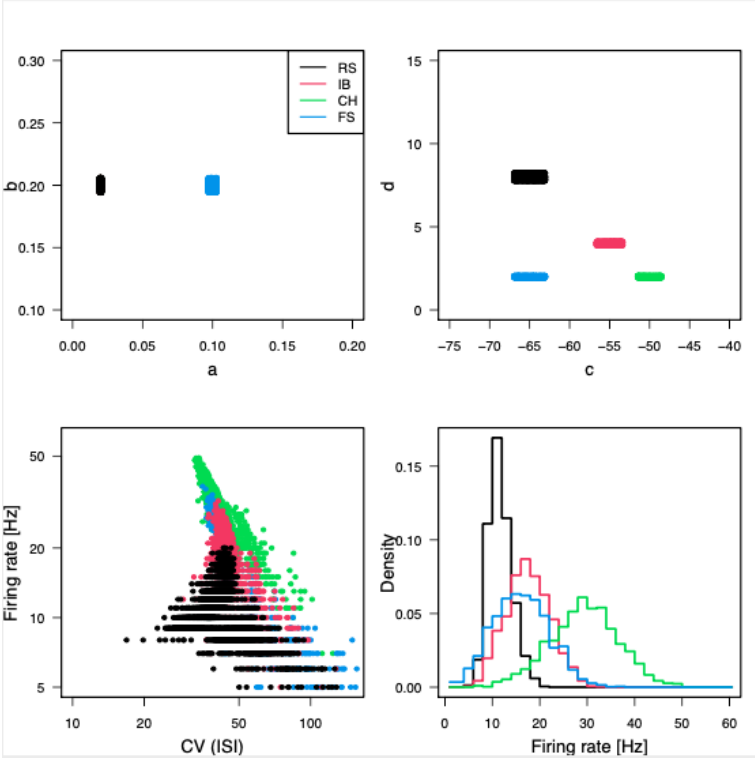


Figure 16: *File 3- Evolution of parameters a,b,c,d, statistics concerning the neurons, the coefficient variation and the firing rate*