ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Programming Report

Project 2: Neuron Network

Group 12:
Lola Maïa Lou Bardel, Constance Laure Marie Géraldine Gabrielle De Trogoff Coatallio,
Alexandra-Elena Preda, Clara Rossignol

December 18, 2020

1 Program presentation

1.1 Description of the project

This is an implementation of the model of E.M. Izhikevich ([Simple Model of Spiking Neuron, IEE Trans. Neural Net., 2003]
https://www.izhikevich.org/publications/spikes.pdf
The goal of this project is to simulate the Neo-cortical neurons of the mammalian brain. The model takes into account the membrane potential and the recovery variable (which relates the activation of K+ionic currents and inactivation of Na+ionic currents) of a neuron. When the membrane potential exceeds a certain threshold, the neuron is spiking and can affect the neurons to which it is connected. Depending on four parameters a, b, c, and d, the model reproduces spiking and bursting behavior of known types of cortical neuron such as:

- excitatory neurons: **RS** (regular spiking), **IB** (intrinsically bursting) **CH** (chattering), **TC** (thalamo-cortical) and **RZ**(resonator)
- inhibitory: **FS** (fast spiking) and **LTS** (low-threshold spiking)

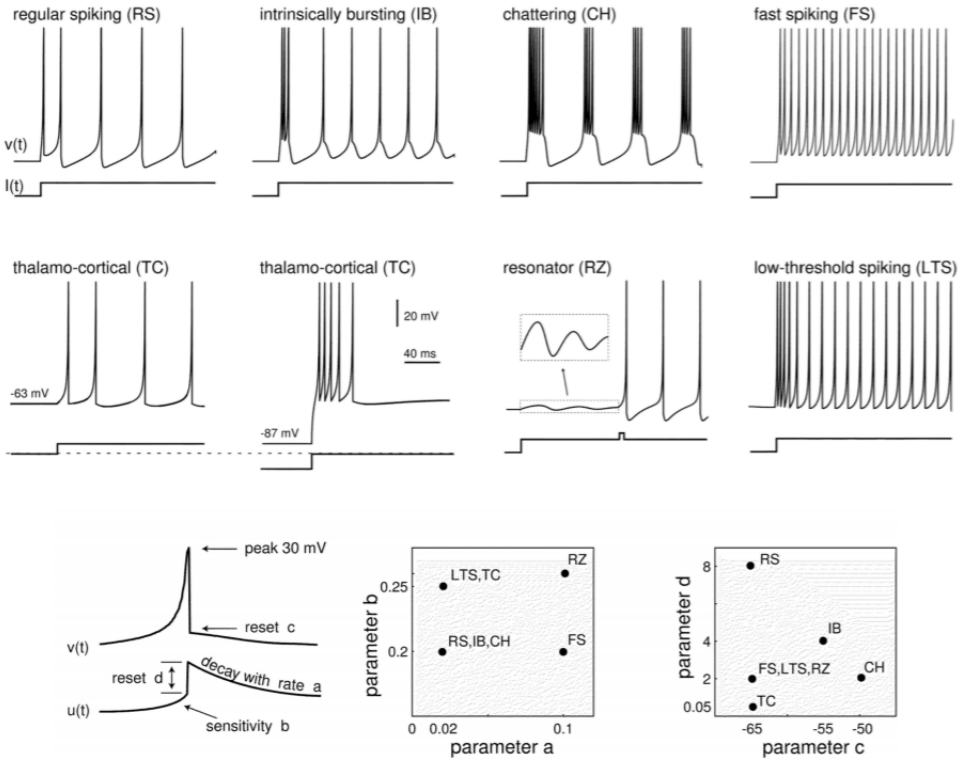


Figure 1: Here we can see each inset from all different types of neurons which shows a voltage response to a step of dc-current I=10. The parameter **a** describes the time scale of the recovery variable. The parameter **b** represents the sensitivity of the recovery variable. The parameter **c** corresponds to the after-spike reset value of the membrane potential caused by the fast high threshold K+ conductance. The parameter **d** describes after-spike reset of the recovery variable caused by slow high-threshold Na+ and K+ conductance.

The connections between neurons are created randomly: each neuron receives the inputs from a random number of other neurons chosen at random with a random intensity. Parameters for random distributions can be specified by the user as we precise in the next part.

1.2 Compilation and commands

In order to use the program, the user has to clone the repository from GitLab, using the command here. To open the program, in the terminal type the following commands, where X is replaced by 12:

```
git clone https://gitlab.epfl.ch/sv_cpp_projects/team_X.git
cd team_X
rm -rf build
mkdir build
cd build
cmake ../
make
make doc
make test
```

The command **make doc** generates the Doxygen documentation of the program. In order to find more details about the implementation, one can go to the folder doc->html->annoted.htm .
For this program there is no need to specify any input file. To run a command that will create a Neuron Network, the user can type: **./NeuronNetwork -h**
and the terminal will print something similar to the following list:

```
USAGE:

   ./NeuronNetwork  {-B|-C|-O} -T <string> [-l <double>] [-o <string>] [-i
                    <double>] [-L <double>] [-c <double>] -t <int> -N <int>
                    [--] [--version] [-h]
```

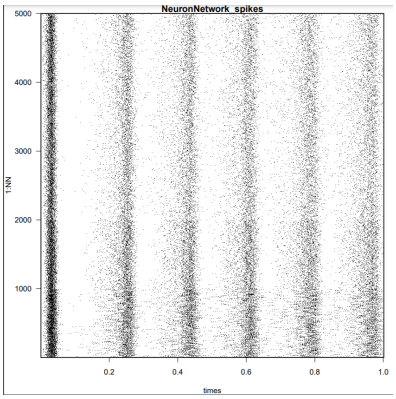
where **B,C,O** represent the type of network model: basic, constant or overdispersed respectively, **l** represents the standard deviation of the thalamic input,**i**:proportion of inhibitory neurons, **L**:average connections' intensity, **c**:average connectivity,**t** number of time steps,**N** represents the number of neurons and **T** represents the neuron types previously mentioned in the description of the project.

The parameters **c** and **L** and **l** are optional, as they are default parameters in our program. The default parameter for c is 1, for L it is 4 and for l is 1. **i** is also an optional parameter but if its value is not precised in the command line, the program will run without using i.

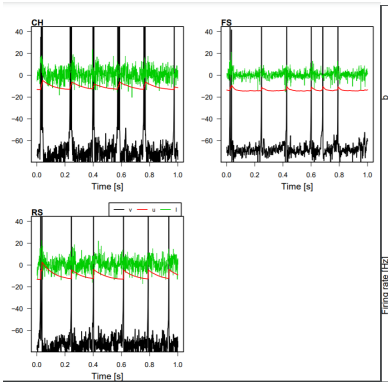
Concerning the neuron type proportions, the user can add proportions for all existing types. If he doesn't specify the RS proportion, the program will add RS neurons in order to reach a total proportion of 1. In the same way, if the user doesn't specify the FS proportion but specifies the total inhibitor's proportion, the program will add FS neurons in order to reach the total inhibitor's proportion.
In this manner, for a standard mode, the user can launch the program with only FS and RS types in four ways:
./NeuronNetwork -B -t 500 -N 1000 -T "RS:0.7,FS:0.3"
or **./NeuronNetwork -B -t 500 -N 1000 -i 0.3 -T" "**
or **./NeuronNetwork -B -t 500 -N 1000 -i 0.3 -T "RS:0.7"**
or **./NeuronNetwork -B -t 500 -N 1000 -T "FS:0.3"**
For a more complex network, the user can also launch the program with all types of neurons in four ways, but we only specify two as it would be redundant otherwise:
./NeuronNetwork -B -t 500 -N 1000 -T "IB:0.2,FS:0.3, CH:0.1, LTS:0.1, RZ:0.1, RS: 0.2"
or **./NeuronNetwork -B -t 500 -N 100 -i 0.4 -T "IB:0.2, CH:0.1, LTS:0.1, RZ:0.1"**
When the user runs the program, he will generate 3 output files: spikes, parameters and sample_neurons.

1.3 Visualisation

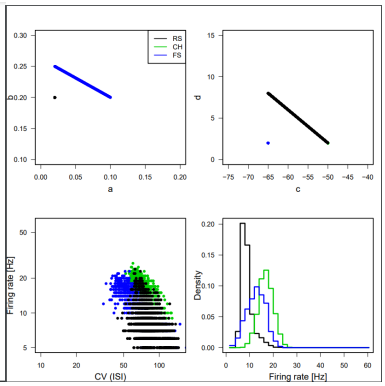
In order to visualize the simulation, the user can open a terminal and write the following command, after opening the directory team_12 (cd team_12):
Rscript ../RasterPlots.R NeuronNetwork_spikes NeuronNetwork_sample_neurons NeuronNetwork_parameters
Note that the prefix-name of the files can change based on the name used after **o**. The graphics will look like the following ones:



(a) Plot 1.



(b) Plot 2.



(c) Plot 3.

The plots are created using the command:
./NeuronNetwork -B -t 1000 -N 5000 -T "FS:0.2,RS:0.6,CH:0.2" -c 70 -L 5 .
The first plot has N columns and t lines. Each point in the first plot represents a firing neuron. The second plots represents the evolution of the intensity of the current, the recovery variable and the **membrane potential** in function of time for each type of neuron. The third plot represents the parameters of the given neurons (a,b,c,d), as well as a Density vs Firing rate graph and a Firing rate vs CV graph, where CV is the *Coefficient of variation of the inter-spike interval* equal to std dev/mean.
The plots are influenced by **c, L, l** and **T**. The way the connections are set depends on the parameters **c** and **L**. If the user increases **c**,the neurons in the first plot will gather into certain denser areas of the plot, while decreasing it will make the neurons in the first plot more evenly spread out. If one decreases **L**, the dots will be more spread out and more distinguishable, while increasing it will make an abundance of firing neurons, making the the dots look like lines, second plot will present denser tighter and in the third plot the density will be 0. And, finally, if **l** is increased, the spikes in the second plot will be very high and dense, since **l** affects the current calculation.

2 Implementation

In order to implement this project, we created 4 main classes:

`Random`: This class allows us to generate random numbers.
It provided us functions that were running usual distributions as the exponential one and the uniform used in the algorithm.

`Neuron`: A neuron is defined by the 4 parameters: a,b,c,d and whether or not it is inhibitory. It is aware of all neurons from which it is connected, spikes when the membrane potential exceeds the threshold of 30 mV and updates its membrane potential and recovery variable.

`Network`: It consists of the simulation of the network of interconnected neurons. First the Network creates all neurons according to the needed types' proportions, and parses all connections between them. Then it sets all neurons' behaviours and identifies which one will be discharging during the simulation. The Network is also in charge of making all neurons' parameters and stimulation visible in the output files and graphs.

`Simulation`: It manages the user's inputs, defines the simulation parameters, then runs the simulation and prints the results to the output streams.

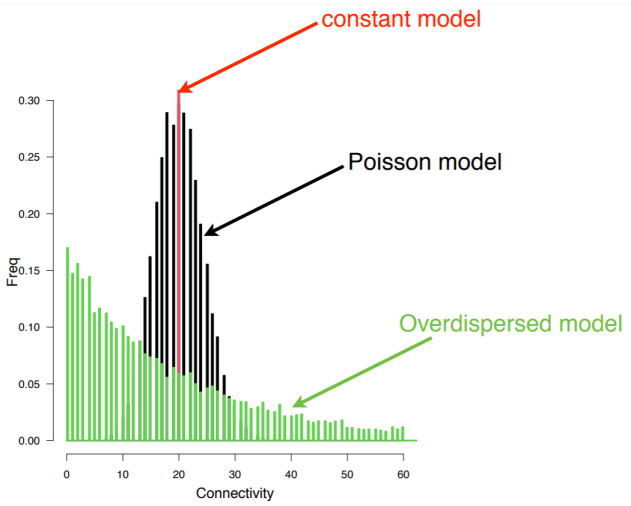
We also took into account the general **error handling**. In general, the user can write any command in the terminal: if that command is wrong in any way (i.e the numbers are negative, the proportions are bigger than 1 or the sum of the proportions is bigger than 1) or there is a corrupted output file, the corresponding error message will be printed, thanks to a method called set that creates a message with a given code. When it encounters the error, it will throw the error message.

3 Performance

To ease the calculation of the current for a neuron, printing neuron's parameters and to make them faster, we decided to order the set of connections of a Neuron in such a way that inhibitory neurons come first. We also know the exact position of the last inhibitory neuron. As a consequence, we never have to test whether a neuron is an inhibitor or not.

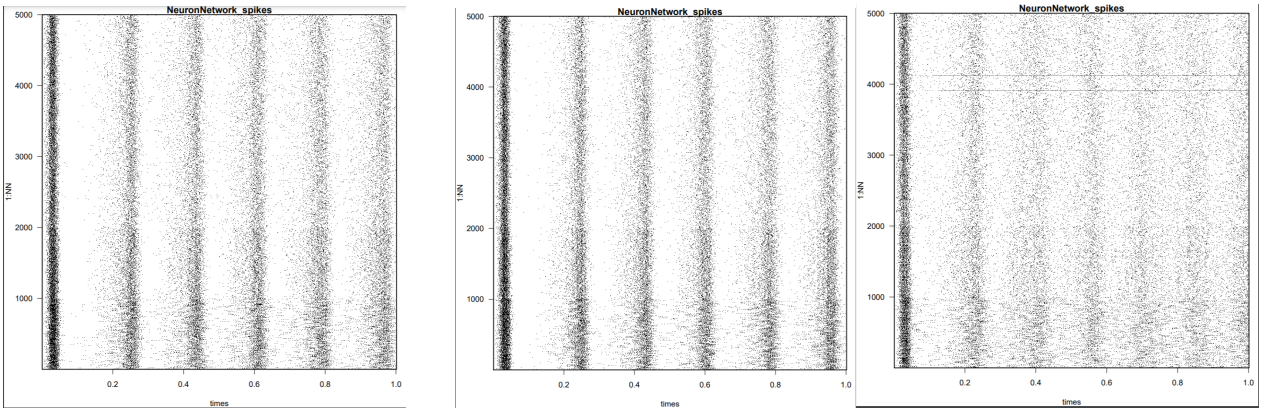
4 Extensions

Regarding the extensions, we decided to add different network models.Until now, we only used a basic model, namely the mean connectivity is fixed and each neuron has n-Pois(c) connections. In order to implement this extension, the classes `ConstNetwork`, `DispNetwork` were created. For the Constant model of the network, each neuron has exactly c connections, while for the Overdispersed model, for each neuron we draw a random average connectivity i-exp(1/connection). We want to study the effect of changing the connectivity pattern. This means that the patterns will follow the following graph:



In order to better understand the difference between the 3 types of models, one can look at the 3 images presented below, created using the command used in section 1.3 and changing the type of model.

One can observe that image (b) is the most homogeneous out of all the 3 plots because all neurons have the same **c**.The plot (a) is the one following plot (b), in terms of homogeneity.In the image (c), it can be observed that the plot is the least homogeneous one, due to the change in connectivity.



(a) Basic model. (b) Constant model. (c) Over-dispersed model.

5 Tests

The test file in this project contains the tests, all included in test/main.cpp. We used Google Test(gtest) which is a unit testing library for the C++ programming language, based on the xUnit architecture (from wikipedia). The tests are meant to test all the non-trivial methods in the program (i.e.:methods with more than 3 code lines).
Simulation:

- **readTypesProportions**: checks if it correctly reads a string containing neuron type proportions and if it parses properly proportions into a map
- **checkMatchingProportions**: checks if it correctly verifies incompatible proportions. Because we expect the checkTypes method to throw an error, we use the test "EXPECT_ANY_THROW"
- **checkInBound**: checks if it detects out of bound values: if those values are higher or smaller than a given minimum and a given maximum. Because we expect the checkTypes method to throw an error, we use the test "EXPECT_ANY_THROW"

Neuron:

- **neuronTypes**: checks if all neurons have been created with the correct type and if they are inhibitor or not as they should be. Verifies if they have their membrane potential and their recovery variable according to their types
- **update**: checks if the calculation of the recoveryVariable is correct. Verifies if the membranePotential and the recoveryVariable are reinitialized if the neuron is firing.
- **currentCalculation**: checks if the current is well computed. Because the calculation includes a random variable, we calculated the average value given by the current of 100 neurons with thalamic input of 1. Each one of those have connections made with two non-inhibitory neurons and all connections have an intensity of 10. As half of the contribution of non inhibitory

neurons is considered in the calculation of the current and as it is added to a random value from a normal distribution, we assume that the result should have a mean of 10. Since it is testing an average, the final value will not be exactly equal to the expected result,therefore we use the test "EXPECTED NEAR" and give an absolute error of 0.5.

- **setConnections**: checks if the value of nInhibitory is well set and if the vector of Connections newly created is of the right size.

Network :

- **proportionConstructor**: makes sure that all neurons in the network are created in the good proportions.
- **indexes**: verifies if all indexes really correspond to the last neuron of each type in the neurons' set of a Network.
- **setConnections**: checks if all neurons have the right number of connections and if a connection has the right intensity value.
  As each neuron has its number of connections calculated with a Poisson distribution with an average given, we calculated the mean value of the connections' number of 100 networks, each one of them containing 5000 neurons.
  As each connection has its intensity calculated with a uniform distribution with an average given, we calculated the mean value of the connection's intensity of 100 networks, each one of them containing 5000 neurons with an average of 100 connections.
  Since it is testing averages, the final values will not be exactly equal to the expected result, therefore we use the test "EXPECTED NEAR" and give an absolute error of 1 for the mean connectivity and of 0.03 for the mean intensity

ConstNetwork:

- **setConnections**: test the number of connections per neuron. Because every neuron should have the same number of connections, only 3 neurons are created. We checked for each one of those if their number of connections is correct

DispNetwork:

- **setConnections**: checks if all neurons have the right number of connections. To do the test, we proceed in the same way as we did to test the setConnections of Network except that we do not verify the average intensity as it should not be different from the basic Network. The given absolute error is 1.