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 12, 2020

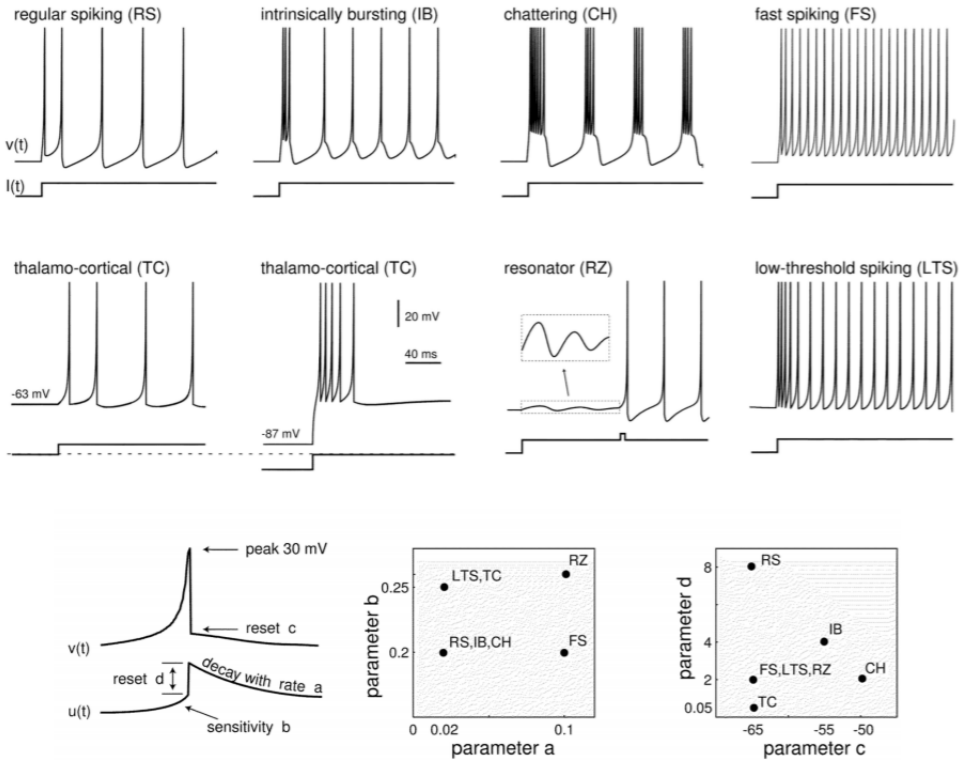
1 Program presentation

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 aim of this project is to simulate a network of N neurons. The connections between neurons are created randomly: each neuron n receives the inputs from d(n) (variable Random Poisson distribution with mean) other neurons chosen at random, each link has an intensity l that is a random number (uniform distribution between 0 and 2L). The links and their intensity are chosen at the start of the simulation and do not change during the simulation.
Each neuron is represented by two time functions: the membrane potential, v(t) and the recovery variable, y(t).
There are different types of neurons, 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)

They also have 4 parameters: a,b,c and d. Represented in the figure below.



1.2 Compilation and commands

In order to use the program, the user has to clone 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, but 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]
```

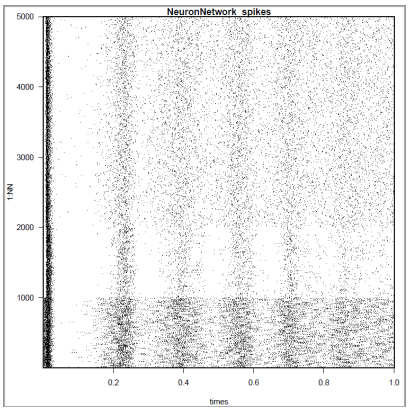
where **B,C,O** represent the type of network model: basic, constant or overdispersed respectively, **l** represents the standard deviation of the thalamic input,**i**:proportion of inhibitory neurons, **L**:average connections' intensity, **c**:average connectivity,**t** number of time steps,**N** represents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

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 did specify the total inhibitor's proportions, the program will add FS neurons in order to reach the total inhibitor's proportions. There are two different modes depending on the command line. In this manner, 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"**
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"**

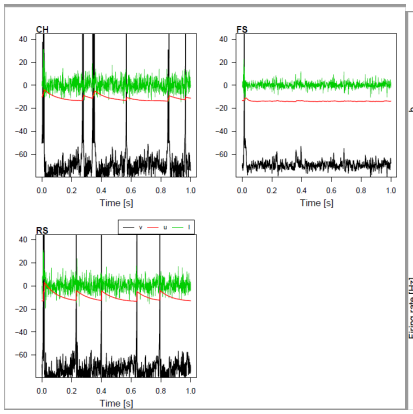
When the user runs the program, he will generate 3 output files: spikes, parameters and sample_neurons.

1.3 Visualisation

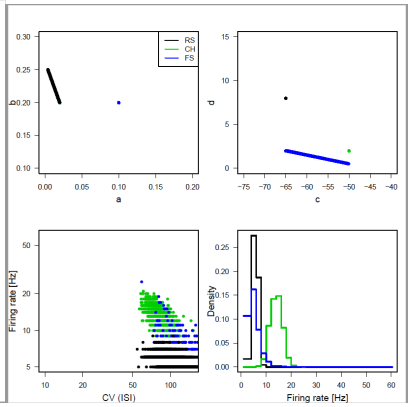
In order to visualize the simulation, the user can open a terminal and write the following command, after opening the directory team_12 (cd team_12):
Rscript RasterPlots.R spikes sample_neurons parameters
The graphics will look like the following ones:



(a) Plot 1.



(b) Plot 2.



(c) Plot 3.

The plots are created using command:
./NeuronNetwork -B -t 1000 -N 5000 -T "FS:0.2,RS:0.6,CH:0.2" -c 70 -L 5 .
Each point in the first plot represents a firing neuron.The second plots represents the actual spikes:the green lines represent the intensity of the current, the red ones represent the recovery variable and the black ones, the membrane potential for each type of neuron as they evolve in time.The third plot represents the parameters of the given neurons, as well as Density vs Firing rate graph.

2 Implementation

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

`Random`: This class allows us to generate a 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 4 parameters:a,b,c,d and whether or not is inhibitory. A structure was implemented to do that, since the the first 4 parameters are doubles and the inhibitory is a bool. In order to represent every type of neurons, we implemented a map that has the type of the neuron as a key and the.The current is calculated for each neuron in this class.The connectivity of each neuron is set in this class.

`Network`: It will construct the neurons, it will set the connections for the neurons,

suing the methods setConnection and setNeuronConnections.It will be updated.And it also contains 3 methods that are called in simulation that will help print the output files.

`Simulation`: It manages the user's inputs, defines the simulation parameters, then runs the simulation and prints the results to the output streams. In order to get an output from the command, the method readTypesProportions was implemented.
`Main`: This class runs the simulation and catches the potential errors.
`Error handling`: In order to handle the errors, we created a class called Error. In this class each error type has a specific exit code. A good example of error handling in our code is the following one:

```
if (x > max or x <min)
{
    Error::set("Invalid data entered", 1);
    std::cerr << message <<" should be between " << min << "and " << max << std::endl;
}
```

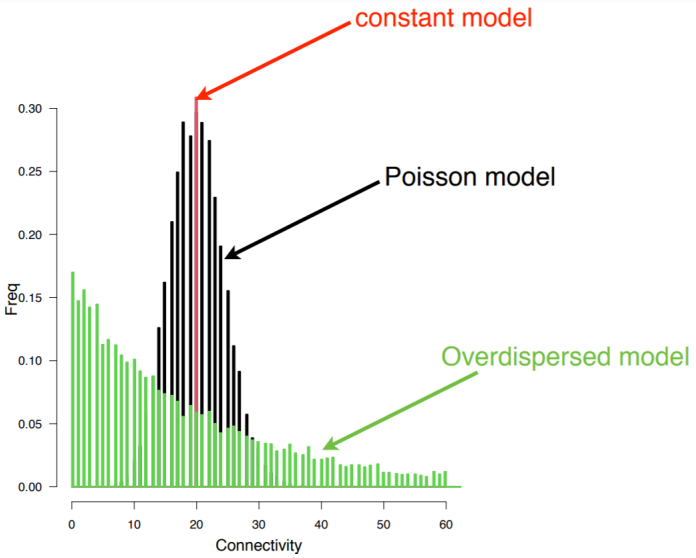
Set is a method of class error that will throw a message an exception number.

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.

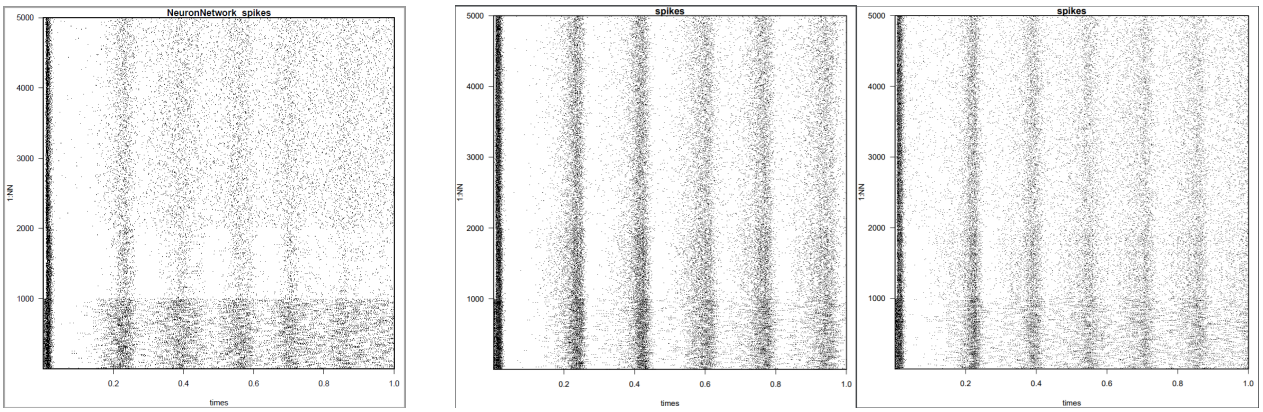
4 Extensions

Regarding the extensions, we decided to add different network models. 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,each neuron has connections and for each neuron draw a random avg connectivity iẽxp(1/connection) We want to study the effect of changing the connectivity pattern. What it means is 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 (2) and changing the type of model.



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

One can observe that the 3 models are different.Here we have to explain how.

5 Tests

The test file in this project contains the tests, all included in test/main.cpp. We used Google Test(gtest) 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).
Neuron:

- **create_neuron**:checks if a neuron is created
- **neuron_types**:checks if all neurons have been created with the correct type and if they are inhibitor or not as they should be.
- **update**: checks if the membrane_potential and the recovery_variable are the correct ones for a given type of neuron
- **current_calculation**: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.

Network :

- **prpoprtion constructor**:make sure that all neurons in network are created in the good proportions. Verify also if all indexes really corresponds to the last neuron of each type in the set of neurons 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 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 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 to complete for the mean connectivity and of to complete for the mean intensity

Simulation:

- **readTypesProportions**:checks if it correctly reads a string containing neuron type proportions and if it parses properly proportions into a map
- **checkTypes**:checks if it correctly verify 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"

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. In the DispNetwork model, each neuron should have a random variable calculated with an exponential distribution as average connectivity, from which we calculate the number of connections with a Poisson distribution. To do the test, we proceed in the same way as we did to test the setConnections of Network except that we do no verify the average intensity as it should no be different from the basic Network. The given absolute error is 1.