ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

# Programming Report

Project 2: Neuron Network

*Group 12:*
Constance De Trogoff
Lola Bardel, Alexandra Preda, Clara Rossignol

December 11, 2020

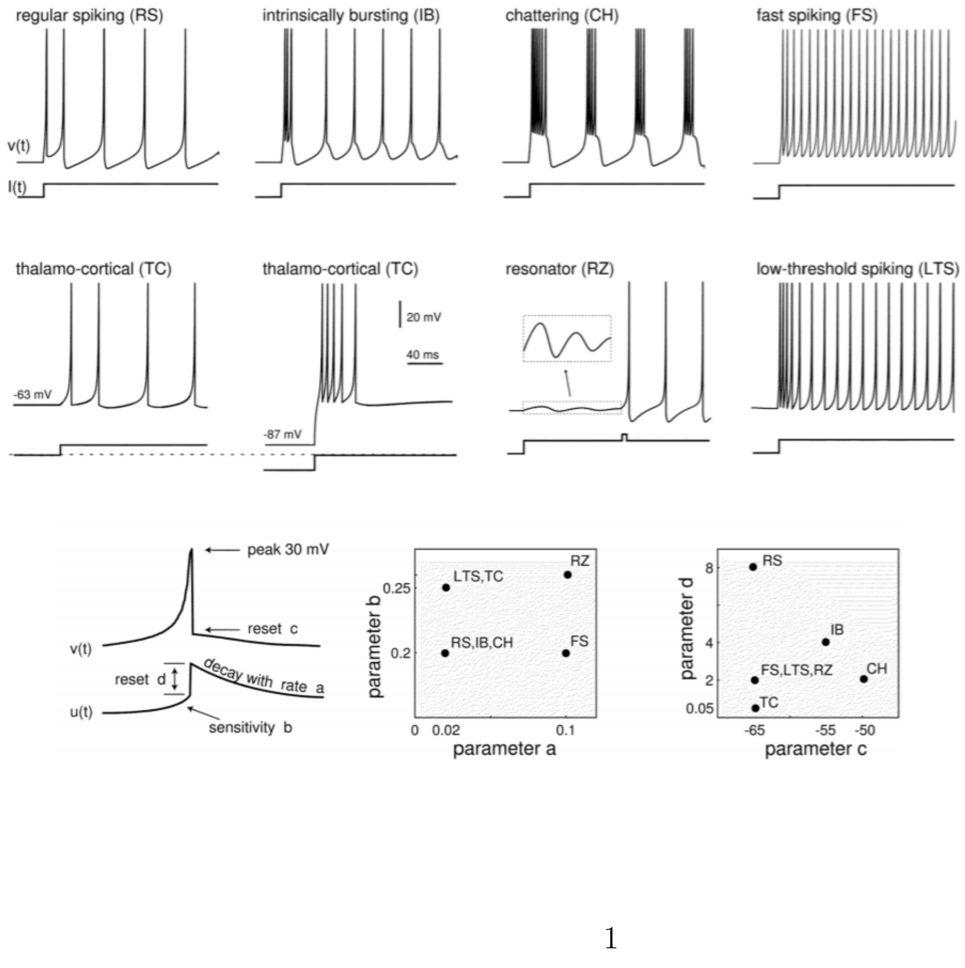
# 1 Program presentation

## 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:

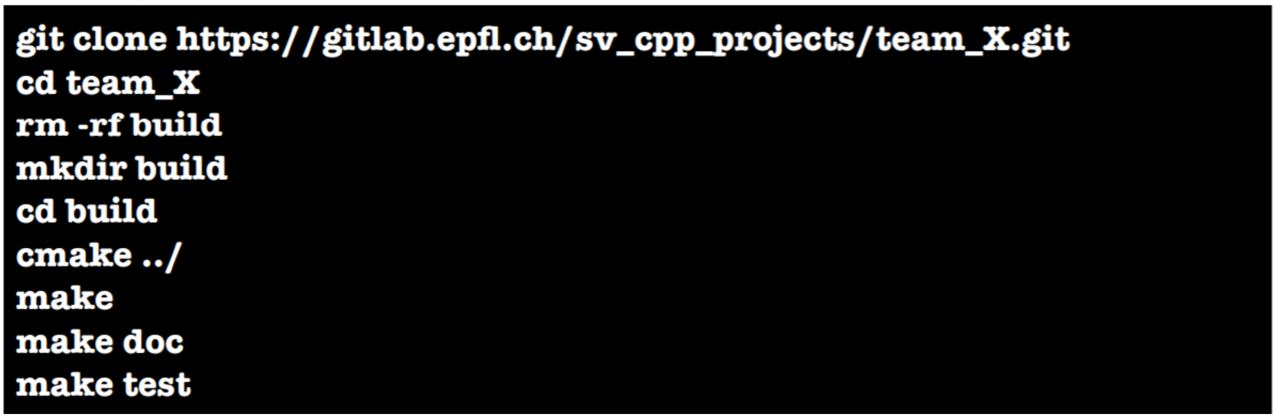
- excitatory neurons:**RS** (regular spiking), **IB** (intrinsically bursting) **CH** (chattering), **TC** (thalamo-cortical) and **RZ**(resonator)
- inhibitory: **FS** (fast spiking) and **LTS** (low-threshold spiking)

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



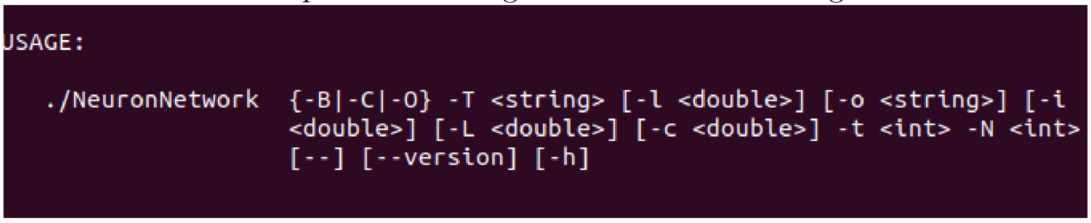
## 1.2 Compilation and commands

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. 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 thalamic input,**i**:, **L**:mean intensity, **c**:mean connectivity,**t** represents the simulation duration (in time steps),**N** represents the number of neurons and **T** represents the neuron types previously mentioned in the description of the project.

There are two different modes depending on the command line. The first command is:
**./NeuronNetwork -B -t 500 -N 10000 -T "RS:0.7,FS:0.3" -c 40 -L 10** (1)
Which will start the standard mode. If the command is, instead:
**./NeuronNetwork -B -t 1000 -N 5000 -T "FS:0.2,RS:0.6,CH:0.2" -c 70 -L 5** (2)
the program will enter in the second mode which will include all types of neurons.
Note that in order to have a good visualization of the network, one can change the L(mean intensity) or c(mean connectivity) .

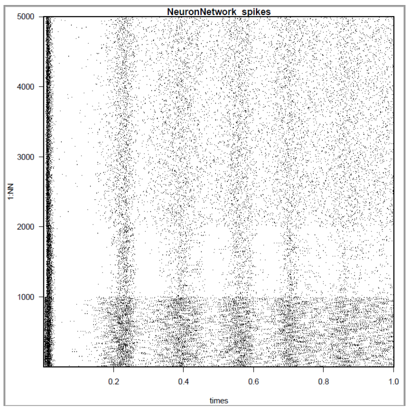
When you run the program, you will generate 3 output files: spikes, parameters and sample_neurons

## 1.3 Visualisation

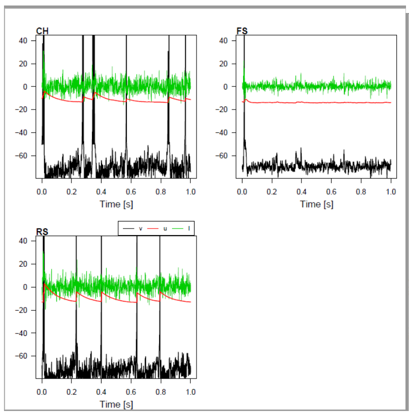
In order to visualize the simulation, the user can open a terminal and write the following command, after opening the directory team_12 (cd team_12):

**Rscript RasterPlots.R spikes sample_neurons parameters**

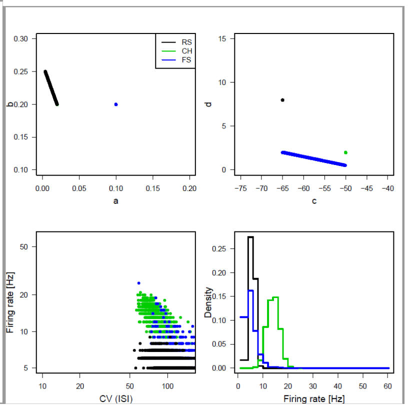
The graphics will look like the following ones:



(a) Plot 1.



(b) Plot 2.



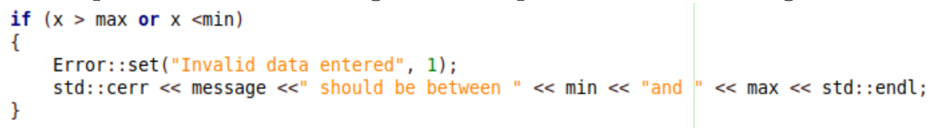
(c) Plot 3.

The plots are created using command (2).Each point in the first plot represents a firing neuron,second plots represents the actual spikes and the third plot represents the parameters of the given neurons.

## 2 Implementation

In order to implement this project, we created 4 main classes:
`Random`
This class allows us to generate a random noise.
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 an enumerated type, followed by 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.In order to calculate the current, we have a method that multiplies with 0 if.. This will explain why the execution time is a few seconds.
The execution time of the program with the command no (2) will be around 14 seconds. The reason for that is the design of the method currentCalculation(). This method is meant to calculate the current by multiplying ....

`Network`
It will construct the neurons, it will set the connections. And it will be updated.
`Simulation`
It manages the user's inputs, defines the simulation parameters, then runs the simulation and prints the results to the output streams. The parameters of the simulation are: the time steps, the number of neurons, the types of neurons mentioned in program description (that will be parsed using the method readTypesProportions), the mean connectivity and the mean intensity.
`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:

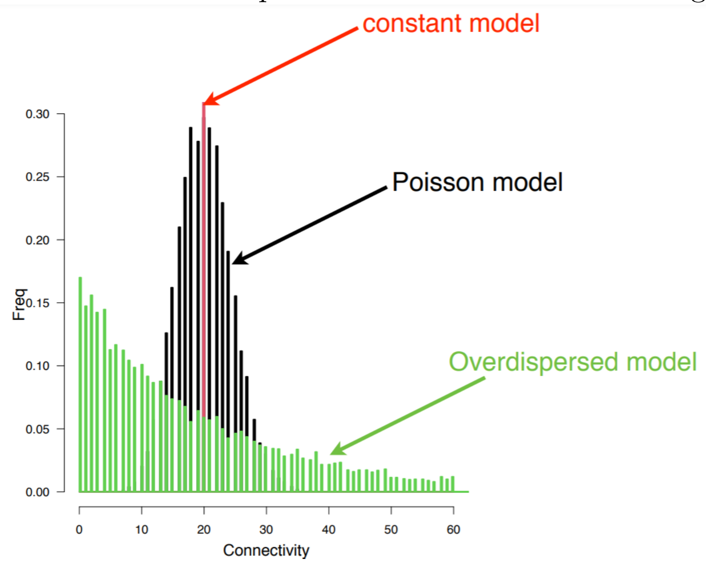


```
if (x > max or x <min)
{
    Error::set("Invalid data entered", 1);
    std::cerr << message <<" should be between " << min << "and " << max << std::endl;
}
```

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

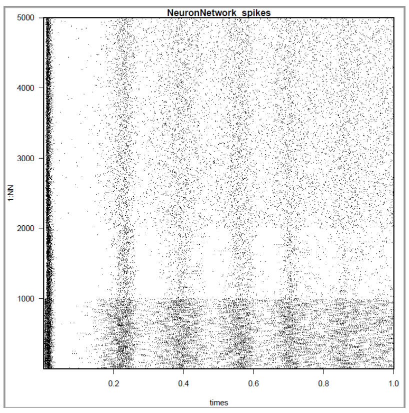
## 3 Extensions

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:

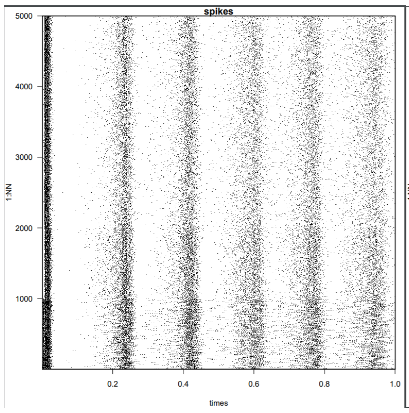


In order to better understand the difference between the 3 types of models, one can look at the 3 images presented below, created using the command (2) and changing the type of model.

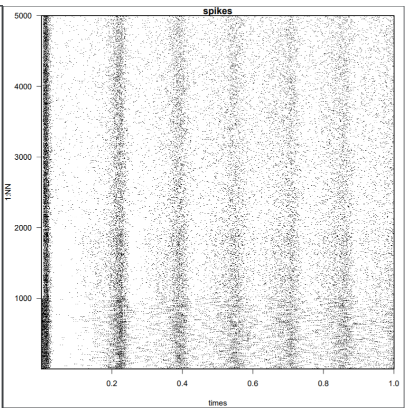
One can observe that the 3 models are different.



(a) Basic model.



(b) Constant model.



(c) Over-dispersed model.

## 4 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**:it checks if the created neuron is firing
- **neuron_types**:checks if the created neuron is inhibitory or not
- **update**: checks if the membrane_potential and the recovery_variable are the correct ones for a given type of neuron
- **current_calculation**:

**Network** :

- **prpoprtionconstructor**:
- **setConnections**: checks if the connections are good, using EXPECT_NEAR()
- **simpleConstructor**:

**Simulation**:

- **readTypesProportions**:
- **checkInBound**: