Theoretical Neuroscience

Gustavo Patow
IMAE - UdG

May 23, 2022

Abstract

In this assignment we will verify the results obtained by Montbrió and Pazó in their 2020 paper. It consists of three exercises: the simulation of a single QIF neuron, the simulation of a population of these neurons, and finally the verification of the firing-rate equations presented in the paper.

1 Introduction

As we discussed in class, the development of population models is a key issue in Theoretical/Computational Neuroscience, as they enable the creation of whole-brain models which allow a direct simulation of the brain and could help in clinical practice.

In the year 2015 Montbrio, Pazó and Roxin [MPR15] developed a derivation of a set of exact macroscopic equations for a network of spiking neurons. Their results revealed that the spike generation mechanism of individual neurons introduces an effective coupling between two biophysically relevant macroscopic quantities, the firing rate and the mean membrane potential, which together govern the evolution of the neuronal network. The resulting equations exactly describe all possible macroscopic dynamical states of the network, including states of synchronous spiking activity. A few years later, in 2020, Montbrió and Pazó [MP20] generalized their previous results to the case of electrical synapses, which are thought to play a major role in setting up neuronal synchronization.

In this assignment we will verify the results in the 2020 paper. Although the math in the paper itself is quite complex, the results are quite clear and the figures within are extremely clear. Here we are going to reproduce its main findings, and check their results.

2 Tasks

This assignment is composed of three tasks: the implementation of a single QIF neuron, the implementation of a large population ($N = 1e4$ neurons) and the final verification of their firing-rate equations.

2.1 QIF Neuron

The first step is to write the code for a single, disconnected quadratic-integrate-and-fire neuron.

$$\begin{gathered} \tau \frac{du}{dt} = u^2 + I^{ext} \\ \text{if } u > u_p \text{then } u \leftarrow -u_r \end{gathered} \tag{1}$$

Where τ is the membrane's time constant, and I^{ext} represents an external current. Because of the quadratic nonlinearity in this equation, u may escape to infinity in a finite time. To prevent this, the QIF model incorporates a resetting rule, the same we have seen for the Leaky-Integrate-and-Fire (LIF) neuron: Each time the neuron's membrane potential u reaches the peak value u_p, the neuron emits a spike and the voltage is reset to $-u_r$. In the paper, for convenience the authors define a positive, real parameter characterizing spike asymmetry as the ratio (here we consider $u_p, u_r > 0$):

$$a \equiv \frac{u_p}{u_r}$$

To recover the neuron used in the 2015 paper, we simply have to set $a = 1$, as it had a symmetric behavior.

With respect to the constants, we are going to set $I^{ext} = 1$ and $\tau = 10ms$. We are going to integrate the equation between 0 and $80ms$, with an Euler scheme with time steps of $dt = 10^{-4}ms$. Yes, really small time steps!

We will simulate this for two different values of the asymmetry ration, $a = 1$ and $a = 4$. As a result, we should get something similar to Figure 1 in the paper.

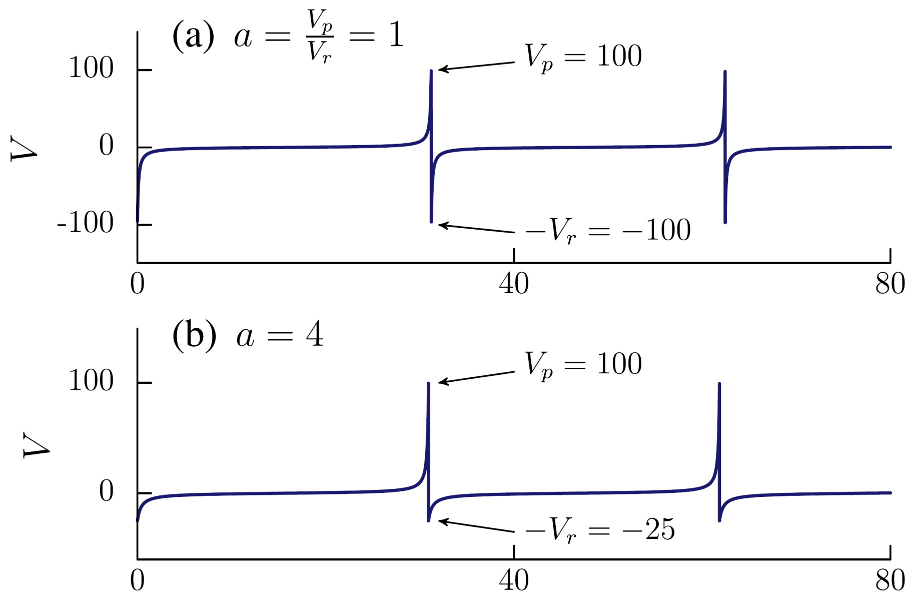

Figure 1: Time series of an oscillatory QIF neuron, Equation 1, with (a) symmetric, and (b) nonsymmetric resetting rule. Taken from [MP20].

2.2 Population of QIF Neuons

Now, to generate a population, the first thing we need to do is to add sub-indexes $j = 1, ..., N$ to label all the neurons in the population, to distinguish one neuron from the other. Also, we are going to modify Equation 1 above to add a couple more terms:

$$\begin{gathered}\tau \frac{du_j}{dt} = u_j^2 + I_j^{ext} + J\tau r(t) + g(v(t) - u_j) \\ \text{if } u > u_p \text{then } u \leftarrow -u_r\end{gathered} \tag{2}$$

here, and the parameters I_j^{ext} represent external currents that are taken from some prescribed probability distribution. We will discuss this point later on. Electrical synapses typically connect inhibitory neurons and hence J should be thought of as a negative parameter. Without loss of generality, we will set $J = 0$, as it has the same practical effect as a, see next section. Finally, g is the electrical coupling, of strength $g \geq 0$ (for this assignment let's use a value of $g = 2.5$,) that diffusely couples each neuron's membrane potential with the mean membrane potential:

$$v(t) = \frac{1}{N} \sum_{j=1}^{N} u_j(t)$$

that is, we should average all the voltages at time t. This should be done immediately after we have computed the new state for the entire population.

Now, we will assume an "all-to-all" connectivity, which implies that we do not need to check whether two neurons are connected, and that all neurons receive all spikes. Thus, to compute the *average firing rate* $r(t)$ we simply need to count all spikes between the current time t and some $t - \tau_s$ in the past. Mathematically, this is written as:

$$r(t) = \frac{1}{N\tau_s} \sum_{j=i}^{N} \sum_{k} \int_{t-\tau_s}^{t} dt' \delta(t' - t_j^k)$$

where t_j^k is the time of the k-th spike of the j-th neuron, $\delta(t)$ is a mathematical function called Dirac delta, and which in this context, together with the integral symbol, means that we should count 1 if there was a spike, and 0 otherwise. That is, actually $\int_{t-\tau_s}^{t} dt' \delta(t' - t_j^k)$ simply means 1 or 0, depending on whether there was a spike or not in the time interval between the current time t and the past $t - \tau_s$. We will choose a very small value for $\tau_s \ll 1$, for instance, $\tau_s = 1e - 3\tau$. Simply speaking, it means:

$$r(t) = \frac{1}{N\tau_s} \sum_{j=i}^{N} s_j(t) \tag{3}$$

where s_j is the number of spikes emitted by the j-th neuron between times t and $t - \tau_s$. As you can see, we are being very formal here, as Montbrió and Pazó did.

External input I_j^{ext}: Here we will assume the external input has a heterogeneous, quenched component as well as a common time-varying component. In our case, this means that, for a given neuron j, with $j \in [0, MMX)$, we will set the current *cmat* for each j as a time-dependent component and an additional local component. For this exercise, the external time-dependent component will be constant ($\eta_0 = 1$), so *cmat* will be generated with the following code:

$$inc = (2.0 * (j + 1) - MMX - 1)/(MMX + 1.)$$
$$cmat = \eta_0 + \Delta * \tan(\pi/2. * inc)$$

with $\eta_0 = 1$ and $\Delta = 1$ and MMX is the number of neurons we are simulating, in our case, 10000.

Initialization: The same way, we have to initialize our neurons. Here we are going to assume there was no current before $t = 0$, so the system is in a stationary state, following a distribution called *Lorentzian*, which means we will initialize our neuronal population by giving each neuron with index j an initial voltage given by

$$inc = (2.0 * (j + 1) - MMX - 1)/(MMX + 1.)$$
$$u = u_0 + r_0 * \pi * \tau * \tan(\pi/2. * inc)$$

with $u_0 = 1$ and $r_0 = 0.015$.

Counting spikes: As we said above, for each neuron we need to compute the number of spikes between a previous tine $t - \tau_s$ and the current time t. But remember we need to work with samples, not time, that is, we need to "translate" τ_s to positions into an array, which means that the actual value you have to use to get the indices must be converted as we did for the other values, dividing by *dt* to get the positions in the array. Do not forget converting it to an integer to prevent strange errors. Finally, note that we are doing this for *all* neurons simultaneously, so we really do not need to store the individual spikes, we only need to store $r(t)$, and its values are the ones that need to be averaged between $t - \tau_s$ and t.

Plotting: To avoid further problems, when plotting, simply use the count of spikes as we did in Equation 3, but without diving by $N\tau_s$, as we just want the firing rate, not the averaged one.

2.3 Verification of the Population model

The macroscopic dynamics of neuronal ensembles have been extensively studied through computational models of large networks of recurrently coupled spiking neurons, including Hodgkin-Huxley-type conductance-based neurons, as well as simplified neuron models, such as the QIF neurons we studied in previous sections. In parallel, researchers have sought to develop statistical descriptions of neuronal networks, mainly in terms of a macroscopic observable that measures the mean rate at which neurons emit spikes, the firing rate. These descriptions, called firing-rate equations (FREs), have been proven to be extremely useful in understanding general computational principles underlying functions such as memory, visual processing, motor control, or decision making.

Montbrió and coworkers [MPR15, MP20], proposed a method to derive the FREs for networks of heterogeneous, all-to-all coupled quadratic integrate-and-fire (QIF) neurons, which is exact in the thermodynamic limit, i.e., for large numbers of neurons.

$$\begin{aligned} \tau \frac{dr}{dt} &= \frac{\Delta}{\tau\pi} + 2ru - gr \\ \tau \frac{du}{dt} &= u^2 + \bar{\eta} - (\pi\tau r)^2 + (J + g\ln a)\tau r \end{aligned} \tag{4}$$

Where $\bar{\eta}$ is the average of the input current, which in practice is $\bar{\eta} = \eta_0 + I(t)$, with η_0 is the same as above (i.e., 1), and $I(t)$ is the external current, which we will set, again, to 0. As a result, we should get, for different values of a and v_p, the plots in Figure 2.

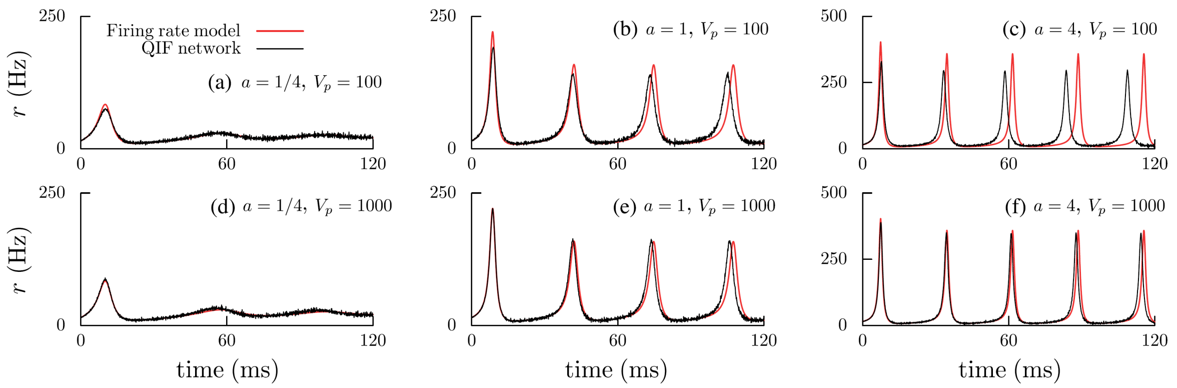

Figure 2: Time series of $r(t)$ for FREs in Equation 4 (red) and network Equations 2 with $N = 10^4$ neurons (black). Taken from [MP20].

2.4 Some comments about the assignment

Note 1: You can implement these assignments in any programming language you may want to, e.g., Python, Java, C++, etc. I have used Python with the *numpy* library and it has been quite an easy ride. On Moodle you will find a document explaining *numpy* for those who have never used it.

Note 2: Although for the first exercise you need to keep track of the neuron's individual voltages, for the other two this may be overkill, as keeping track of all the voltages for $1e4$ neurons is a bit too large in terms of memory. Remember that we only need the *average* of the voltages at each point in time, as well as the *total* of all the spikes. Actually, we need even less than that, as we will group data into bins, see nxt point.

Note 3: Same as before, although for the implementation of the first exercise you will need to keep track of the *whole* time-series to plot them, for the second and third ones you will need to *aggregate* samples into bins, and plot those bins. To help you with this, below you can find some *Python* code.

Note 4: Please, do not complicate yourselves more than necessary. Keep the code simple and straightforward to read. This is not about fancy code, but about doing the simulations.

3 Some useful code

This section provides some bits of code that I have found to be useful for this assignment. It uses the *numpy* library, imported with

```
import numpy as np
```

Now, the first function is *timeSeries2bins*, which is useful for later processing, such as plotting, because our series have 50k and even more elements, which is a bit too much for python plotting libraries. Thus, we need to reduce the quite long time-series (1.2e6) to a more reduced number of bins (1.2e3). In this code, op could be either *np.average* if we want to compute the averages of the values inside the bin, which is what we want to do with voltages; or *np.sum*, which is perfect for counting spikes. Now, we have to specify the size of our bin interval, in seconds. I used 0.1 seconds as *binInterval*, which plays well with our $1e-4ms$ step for integration.

```
def timeSeries2bins(ts, dt, binInterval, op):
    count = len(ts)
    numBins, binSize = np.divmod(count, binInterval/dt)
    sorted = ts.reshape((int(numBins), int(binInterval/dt)))
    bins = op(sorted, axis=1)
    return bins
```

For plotting, I always recommend *pyplot*, another great library for Python that works with numpy. Here you have an example from my own code, where the variable *values* is a pair (a,b) with a and b are two arrays of the same size we want to plot, simultaneously (for example, for the network and for the FRE). *ylim* is a variable used to force some plotting limits in the vertical direction. I personally used *ylim=(0,250)* to match the results in the original paper.

```
import matplotlib.pyplot as plt
# ================ simple utility plotting functions
def plot_one(values, ylabel='V', ylim=None):
    plt.plot(values[0])
    plt.plot(values[1])
    if ylim is not None:
        plt.ylim(ylim)
    plt.title('QIF network')
    plt.xlabel("time")
    plt.ylabel(ylabel)
    plt.show()
```

Finally, if you want to plot multiple subplots into a single one, you should use the *subplots* in *pyplot*, which generates as many axis as requested (in the example below, just 2), and then you plot at each axis independently. Please observe that, for some reason, they have changed the names of some functions when plotting a single graph and when using subplots(e.g., *ylabel* vs. *set_ylabel*), so be careful. Here you have a very simple example, that plots two binned time-series, *u1tr* and *u2tr*:

```
def plot_2(u1tr, u2tr):
    fig, axs = plt.subplots(2, sharex=True)
    axs[0].plot(u1tr)
    axs[1].plot(u2tr)
    plt.xlabel('Time step')
    axs[0].set_ylabel('Voltage [mV]')
    axs[1].set_ylabel('Voltage [mV]')
    plt.suptitle("Time Series of an Oscillatory QIF Neuron")
    plt.show()
```

References

[MP20] Ernest Montbrió and Diego Pazó. Exact mean-field theory explains the dual role of electrical synapses in collective synchronization. *Physical Review Letters*, 125(24), December 2020.

[MPR15] Ernest Montbrió, Diego Pazó, and Alex Roxin. Macroscopic description for networks of spiking neurons. *Physical Review X*, 5(2), June 2015.