Spiking Neuron and Neuronal Network

Izhikevich neuron model

Emil Zvarík
28.4.2021
Computational Cognitive Neuroscience
Faculty of mathematics, physics and informatics
Comenius University in Bratislava

INTRODUCTION

In this work, we aimed to replicate models of different cortical neurons and neuronal network presented in the paper by Izhikevich (2003). For this purpose, we have created python code, which can be found in the Appendix. We have successfully modelled every kind of neuron mentioned in the paper as well as the network of randomly connected 1000 neurons.

Following two sections present the models of neurons and neuronal network respectively.

PART A: MODELLING FIRING PATTERNS OF NEURONS

Below, we replicate the behaviour of different cortical neurons as mentioned in the paper. Our figures plot membrane voltage in blue, injected current in green and u parameter in orange. We note interesting observations and encountered problems below some of the figures.

In every model, we have used time step of 0.1.

Excitatory neurons

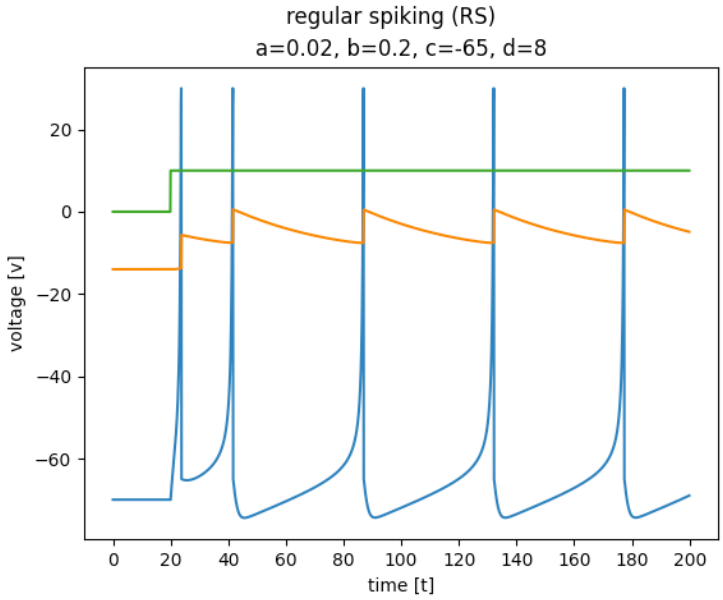

Figure 1 - Regular spiking (RS)

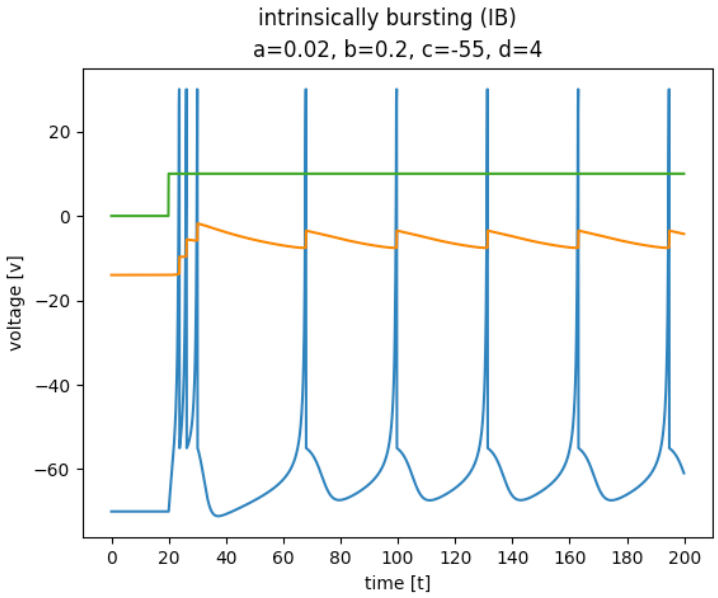

Figure 2 - Intrinsically bursting (IB)

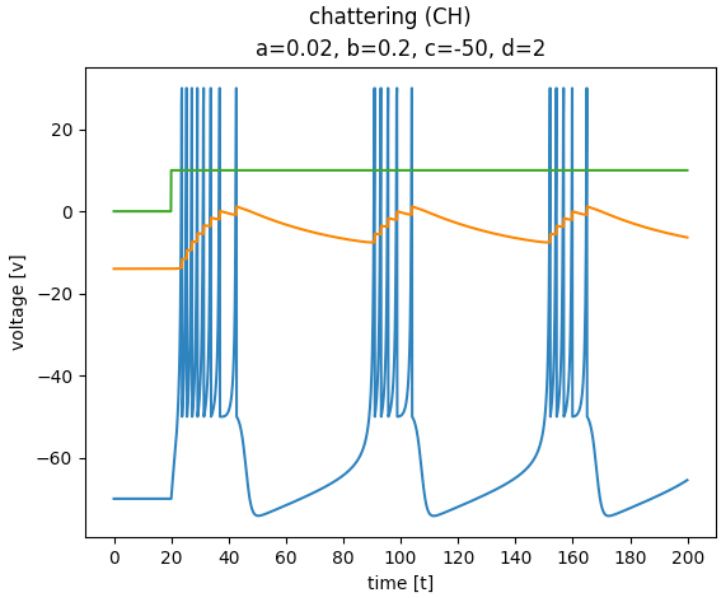

Figure 3 – Chattering (CH)

Note: Frequency of bursts in our figure is 4 bursts per 200 ms, which is 20 Hz. In the paper, Izhikevich claims that the inter-burst frequency can be as high as 40 Hz.

Inhibitory neurons

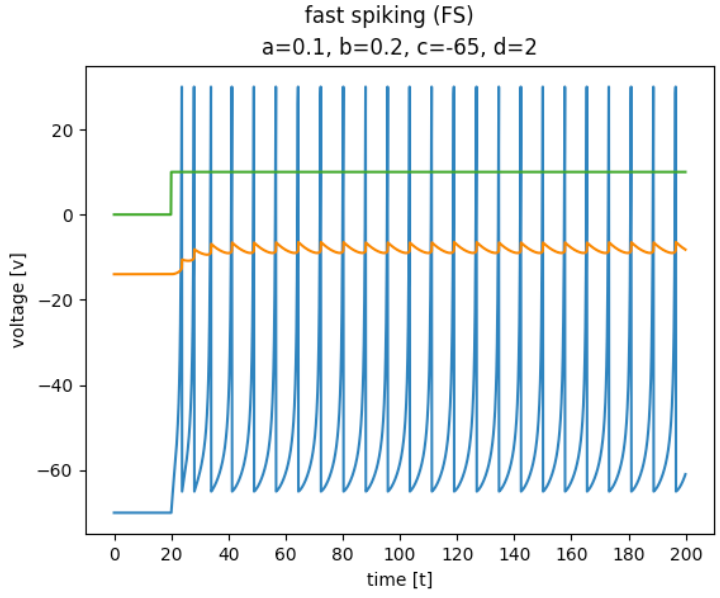

Figure 4 - Fast spiking (FS)

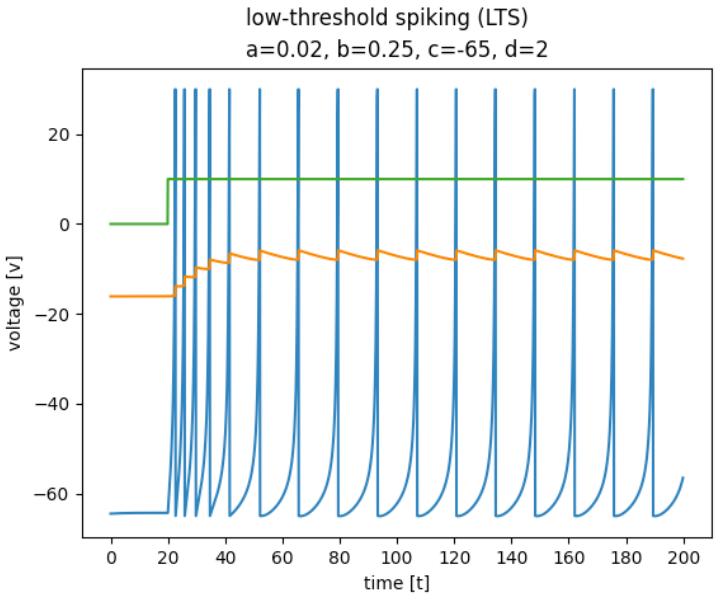

Figure 5 - Low-threshold spiking (LTS)

Thalamo-cortical neurons

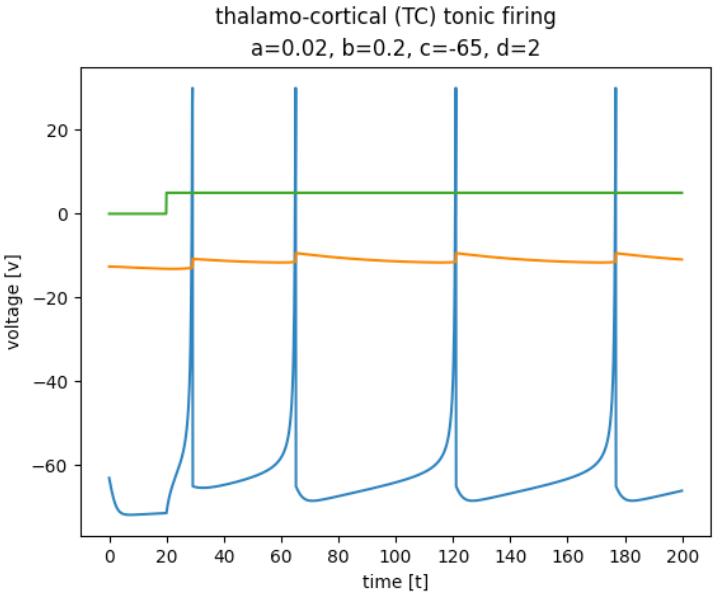

Figure 6 - Thalamo-cortical (TC) tonic firing

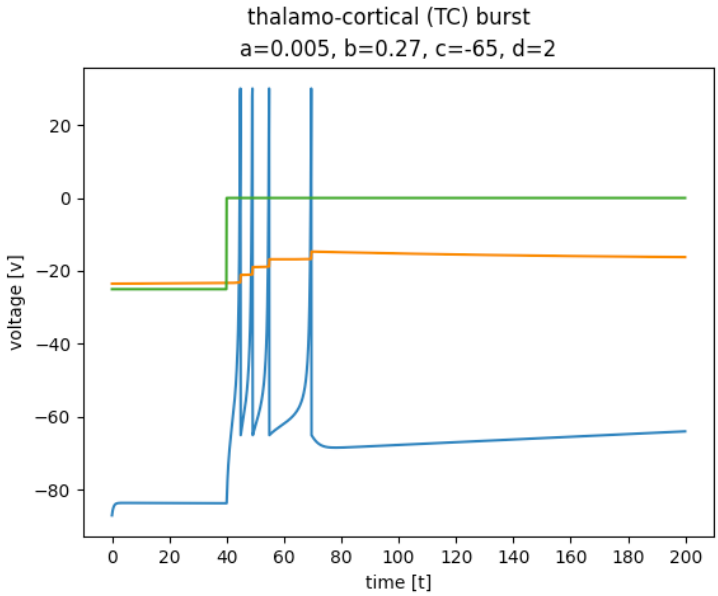

Figure 7 - Thalamo-cortical (TC) burst

Note: Model of this neuron was for us harder to simulate. We started the simulation with *v0* = -87 as was is in the paper. Also, the initial injected current was I0 = -25, to keep the membrane voltage low. As soon as the injected current was turned off (set to 0), we got the bursting behaviour by setting the *b* parameter to *b* = 0.27, which increased the sensitivity of the neuron and thus increased the number of spikes. However, then we had to decrease the *a* parameter to *a* = 0.005 to move the spikes closer together and keep upcoming spikes further from the burst.

Other

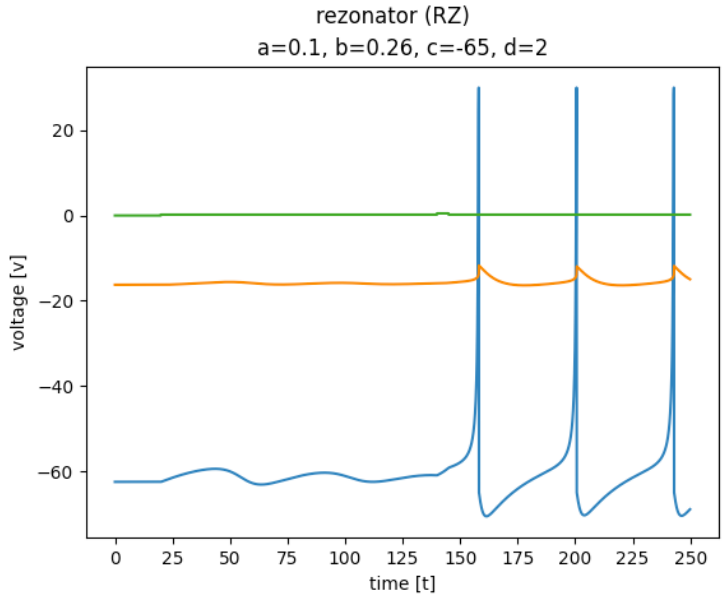

Figure 8 - Resonator (RZ)

Note: When simulating behaviour of the resonator, we had to change the initial voltage to *v0* = 62.5 and use very small injected current I1 = 0.2, I2 = 0.5. Also, interestingly, the timing of the change in injected current from I1 to I2 mattered – the change had to start on the "downslope" of the membrane voltage curve, if it would start on the upslope, it would not trigger the spiking behaviour.

PART B: MODELLING NETWORK OF NEURONS

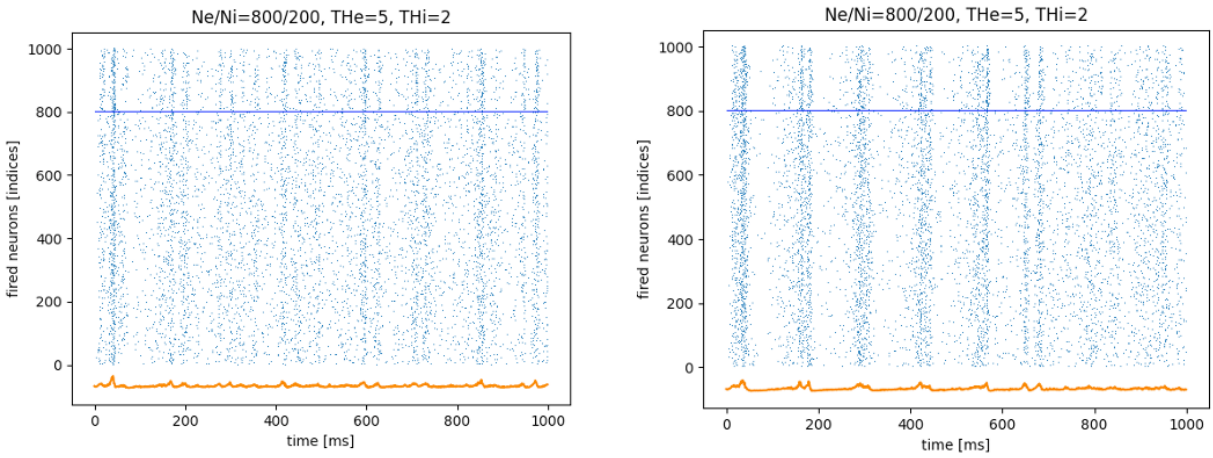

Figure 9 – Default network of neurons, different runs

Note on plots in this section

Every plot in this section displays firing of one neuron as a blue pixel. Blue line separates inhibitory neurons from excitatory neurons and orange curve plots average voltage of all neurons. Above each plot, values of *Ne*, *Ni*, *THe* and *THi* are displayed. *Ne* means number of excitatory neurons, *Ni* number of inhibitory neurons and *THe* and *Thi* are weight of the thalamic input for excitatory and inhibitory neurons respectively.

General observations

We have successfully implemented model from Izhikevich's paper in Python 3. Even with default (same as those from the paper) parameters, significant variations among specific runs can be seen (see Figure 9 above). What seems to be a regularity is strong synchronization at the beginning and lower later. We attribute this to neurons having the same initial membrane voltage. The difference between runs is most probably caused by the random thalamic input.

Ratio of excitatory and inhibitory neurons

From our observations, the grater the ratio of excitatory neurons, the greater the synchronisation. Below, we display what happens when we gradually increment the ratio of the excitatory neurons with a step of ten – the synchronisation becomes so strong that the plotted average activity of the neurons looks like activity of one neuron. In other words, almost all neurons fire roughly at the same time. This happens as soon as when the ratio is 830/170 or 840/160.

Increasing ratio of excitatory neurons

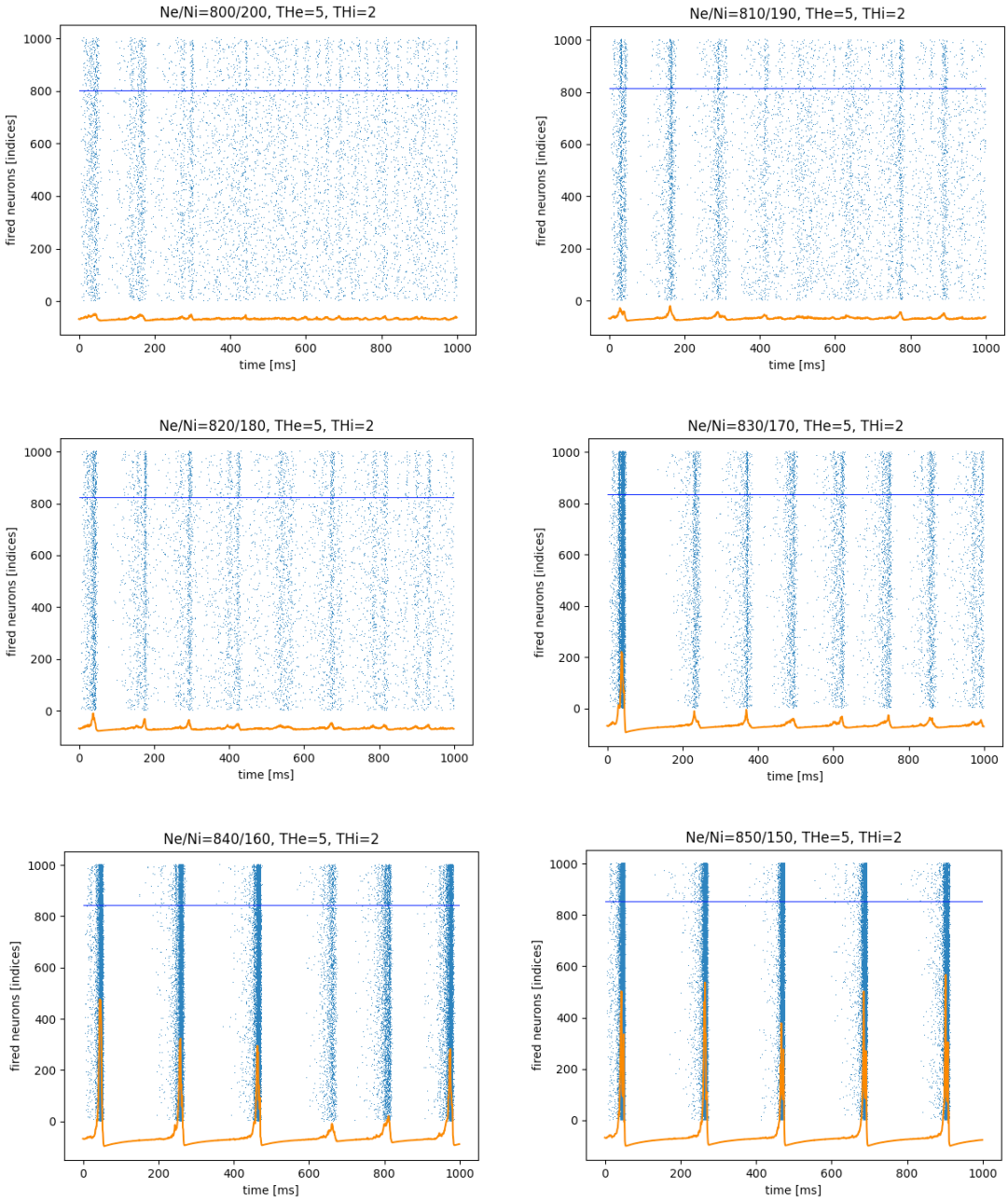

Figure 10 - Network of neurons, increasing ratio of excitatory neurons

Increasing ratio of inhibitory neurons

Here, we increase the ratio of inhibitory neurons similarly to previous section.

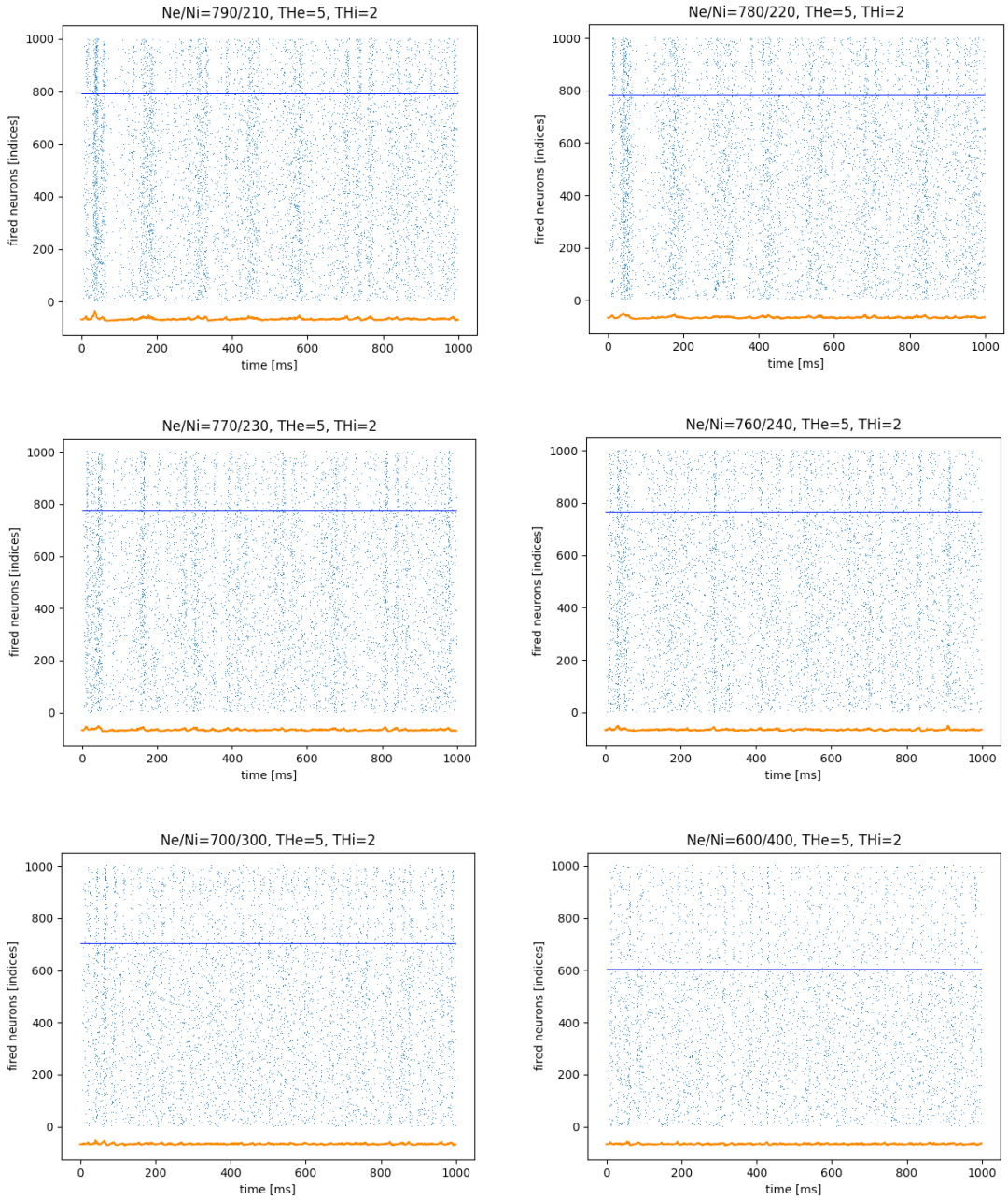

Figure 11 - Network of neurons, increasing ratio of inhibitory neurons

When we go even further with the ratio favouring the inhibitory neurons, we can see not only that the inhibitory neurons fire less frequently, but they also suppress the firing of excitatory neurons (which they do by the definitions, but it is nice to see that it is also happening on our plot).

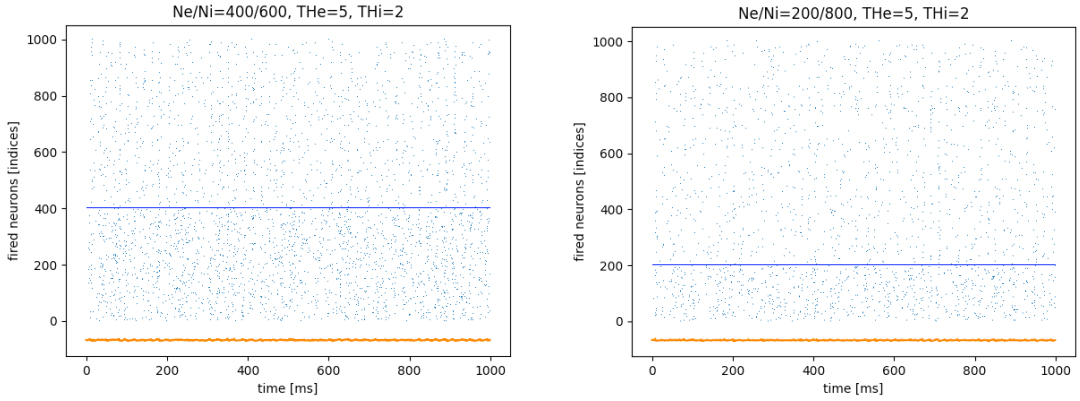

Figure 12- Network of neurons, strongly favouring the ratio of inhibitory neurons

Influence of thalamic noise

We can observe, that increasing the thalamic noise increases the activity of the neurons and the other way around, which is obvious, because the more noise, the more input.

Changing the thalamic noise for the excitatory neurons

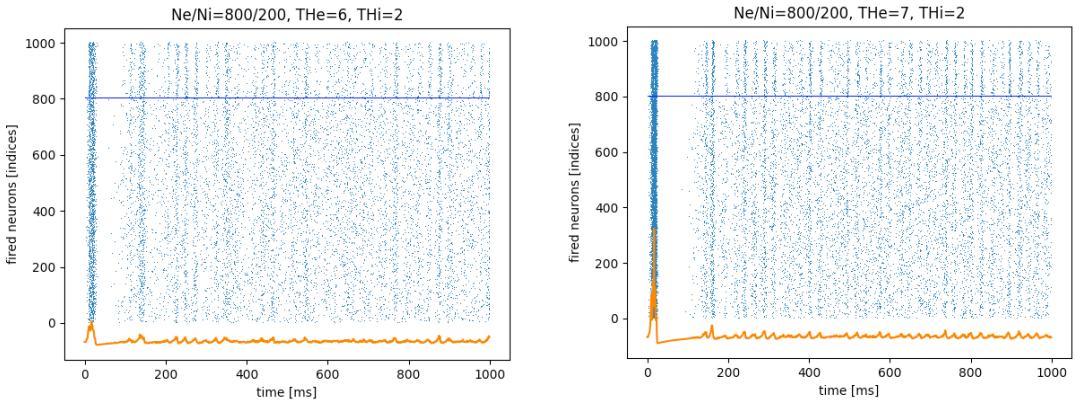

Figure 13 - Network of neurons, increasing the thalamic noise for excitatory neurons

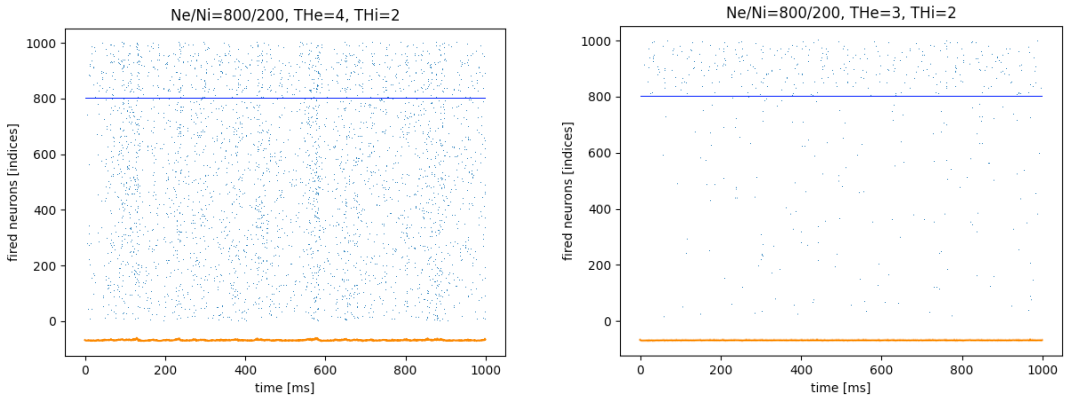

Figure 14 - Network of neurons, decreasing the thalamic noise for excitatory neurons

Increasing the thalamic noise for only the excitatory neurons results in more firing, with higher frequency, but less synchronisation.

Changing the thalamic noise for the inhibitory neurons

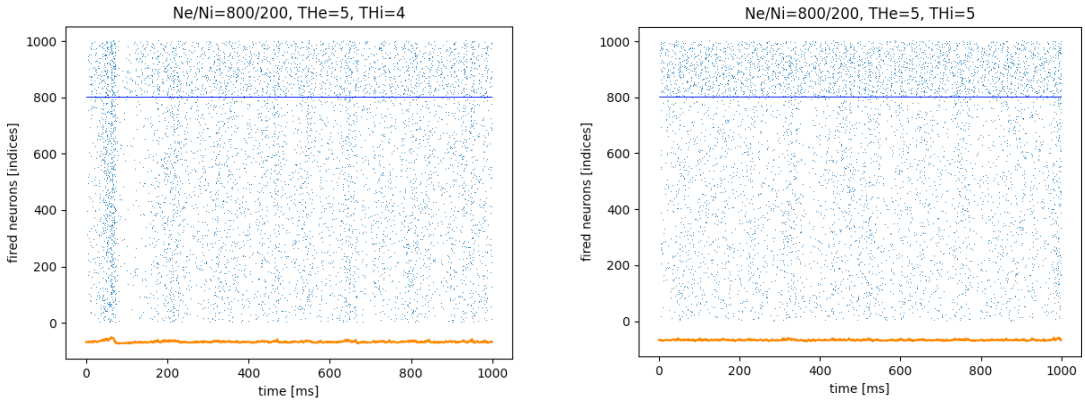

Figure 15 - Network of neurons, increasing the thalamic noise for inhibitory neurons

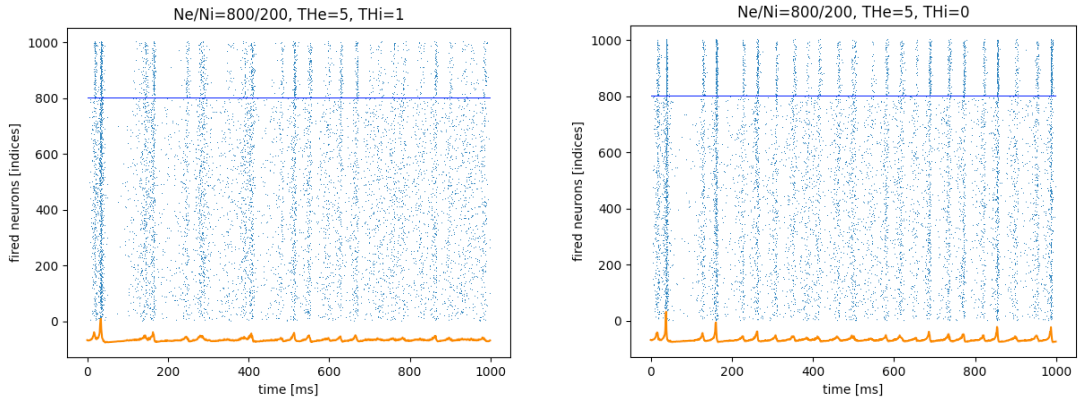

Figure 16 - Network of neurons, decreasing the thalamic noise for inhibitory neurons

Increasing the thalamic noise for only the inhibitory neurons results in greater activity of the inhibitory neurons, but smaller activity in excitatory neurons and weaker synchronisation.

Decreasing the thalamic noise for only the inhibitory neurons causes higher synchronisation with higher frequency.

Increasing the thalamic noise for both the excitatory and inhibitory neurons

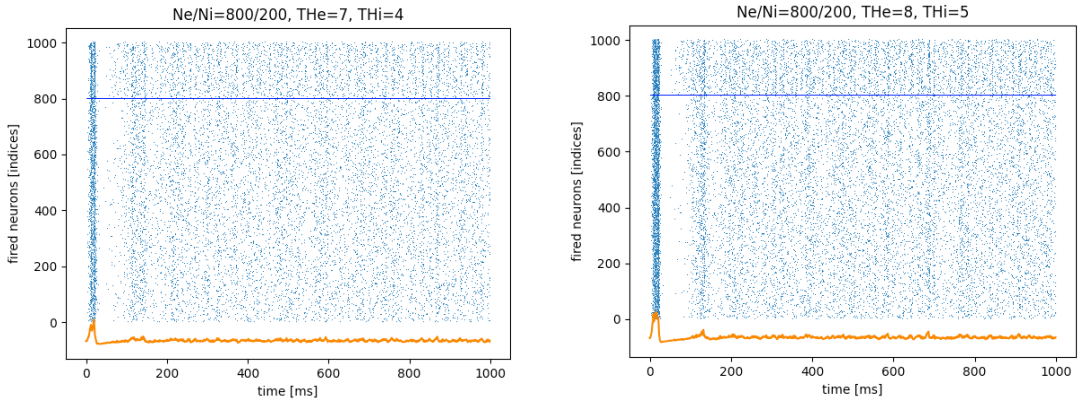

Figure 17 - Network of neurons, increasing the thalamic noise for all neurons

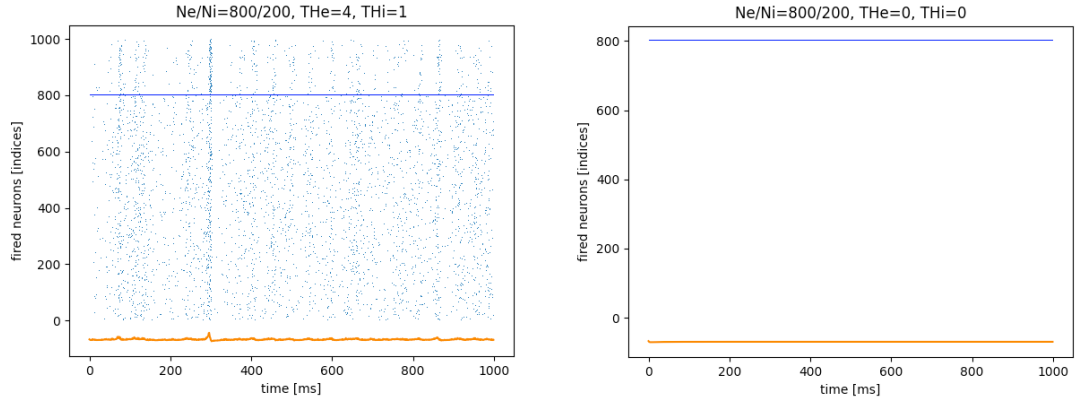

Figure 18 - Network of neurons, decreasing the thalamic noise for all neurons

Increasing the overall noise causes more activity, but almost no synchronisation. Interesting is the high initial synchronisation at around time 0, which is probably cause by the non-random initial starting membrane voltage of 65 mV.

Decreasing the noise simply decreases the activity of all neurons, and turning the noise off results in no firing.

Final notes

The periodic synchronisation seems to be influenced by the inhibitory neurons. When they are in low number, the excitatory neurons take charge and there is nothing in their way to fire synchronously. Also, when the inhibitory neurons receive less thalamic input, they are dependent from the input from excitatory neurons and thus they fire with them synchronously.

Analogously, letting the inhibitory neurons take charge, either by increasing the thalamic input they receive or their number, they cause the network to fire asynchronously.

REFERENCES

E. M. Izhikevich, "Simple model of spiking neurons," in *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569-1572, Nov. 2003, doi: 10.1109/TNN.2003.820440.

APPENDIX

Our code for task A:

```
import math
import matplotlib.pyplot as plt


class Neuron:

    def __init__(self, a, b, c, d, v0, time, time_step, i_times):
        # i_times: iterable in form of number couples: at what time should
what stimulation start
        # e.g.: (0,0,20,10) means at time 0, start with curret 0, at time
20, start with current 10
        #                     (and continue till the end)
        if len(i_times) == 0 or len(i_times) % 2 != 0 or i_times[0] != 0:
            raise Exception("i_times argument must start with 0, has even
length and have at least 2 numbers in it.\n"
                            "e.g. (0, 10) or (0, 0, 20, 10) etc. are
correct, () or (2) or (10, 5) etc. are incorrect")
        self.time = time  # whole simulation duration
        self.h = time_step  # time step
        self.T = math.ceil(self.time / self.h)  # overall number of steps

        # model parameters
        self.a = a  # typical value is a = 0.02
        self.b = b  # typical value is b = 0.2
        self.c = c  # typical value is c = -65
        self.d = d  # typical value is d = 2

        # create lists of size T filled with zeros
        self.v = [0] * self.T
        self.u = [0] * self.T
        self.i = []
        ind = 0
        for t in range(self.T):
            if ind < len(i_times) and t == math.ceil(i_times[ind] /
self.h):
                I = i_times[ind+1]
                ind += 2
            self.i.append(I)
        # set v and u in time 0
        self.v[0] = v0
        self.u[0] = self.b * self.v[0]  # we do not know why this is as it
is

        self.integrate()
        self.plot()

    def integrate(self):
        for t in range(self.T - 1):
            if self.v[t] < 30:  # if v is less than apex, do update v and u
acording to (1) and (2):
                # v' = 0.04v^2 + 5v + 140 - u + I:
                dv = (0.04 * self.v[t]**2) + 5 * self.v[t] + 140 -
self.u[t] + self.i[t]
                self.v[t + 1] = self.v[t] + dv * self.h
                du = self.a * (self.b * self.v[t] - self.u[t])  # u' = a *
(b * v - u)
                self.u[t + 1] = self.u[t] + du * self.h
```

```
            else:  # if v is over apex, do after-spike resetting of u and v
according to (3):
                self.v[t] = 30
                self.v[t + 1] = self.c  # v ← c
                self.u[t + 1] = self.u[t] + self.d  # u ← u + d

    def plot(self):
        plt.plot(self.v)  # voltage on plot in blue
        plt.plot(self.u)  # u on plot in orange
        plt.plot(self.i)  # injected current on plot in green
        plt.suptitle(name)
        plt.title('a={}, b={}, c={}, d={}'.format(self.a, self.b, self.c,
self.d))
        plt.xticks(range(0, self.T+1, self.T//10), range(0, self.time+1,
self.time//10))
        plt.xlabel('time [t]')
        plt.ylabel('voltage [v]')
        plt.show()


# Neuron(a, b, c, d, v0, time, time_step, i_times)
# typical values: a = 0.02, b = 0.2, c = -65, d = 2
# ================================================

# name = "regular spiking (RS)"
# c = -65 mV, d = 8
# params = {'a': 0.02, 'b': 0.2, 'c': -65, 'd': 8, 'v0': -70, 'time': 200,
'time_step': 0.1, 'i_times': (0, 0, 20, 10)}

# name = "intrinsically bursting (IB)"
# c = -55 mV, d = 4
# params = {'a': 0.02, 'b': 0.2, 'c': -55, 'd': 4, 'v0': -70, 'time': 200,
'time_step': 0.1, 'i_times': (0, 0, 20, 10)}

# name = "chattering (CH)"
# c = -50 mV, d = 2
# params = {'a': 0.02, 'b': 0.2, 'c': -50, 'd': 2, 'v0': -70, 'time': 200,
'time_step': 0.1, 'i_times': (0, 0, 20, 10)}

# name = "fast spiking (FS)"
# a = 0.1
# params = {'a': 0.1, 'b': 0.2, 'c': -65, 'd': 2, 'v0': -70, 'time': 200,
'time_step': 0.1, 'i_times': (0, 0, 20, 10)}

# name = "low-threshold spiking (LTS)"
# b = 0.25
# params = {'a': 0.02, 'b': 0.25, 'c': -65, 'd': 2, 'v0': -64.5, 'time':
200, 'time_step': 0.1,
#           'i_times': (0, 0, 20, 10)}

# name = "thalamo-cortical (TC) tonic firing"
# v0 = -63 mV
# params = {'a': 0.02, 'b': 0.2, 'c': -65, 'd': 2, 'v0': -63, 'time': 200,
'time_step': 0.1, 'i_times': (0, 0, 20, 5)}

# name = "thalamo-cortical (TC) burst"
# v0 = -87 mV
# params = {'a': 0.02, 'b': 0.2, 'c': -46, 'd': 2, 'v0': -87, 'time': 200,
'time_step': 0.1, 'i_times': (0, -25, 40, 0)}

name = "rezonator (RZ)"
```

```
# a = 0.1, b= 0.26
params = {'a': 0.1, 'b': 0.26, 'c': -65, 'd': 2, 'v0': -62.5, 'time': 250,
'time_step': 0.1,
          'i_times': (0, 0, 20, 0.2, 140, 0.5, 145, 0.2)}

Neuron(*params.values())
```

Our code for task B:

```
import numpy as np
from pylab import rand, randn
import matplotlib.pyplot as plt

TIME = 1000

Ne = 800  # number of excitatory neurons
Ni = 1000-Ne  # number of inhibitory neurons

THe = 5
THi = 2

# rand(N) creates an array of size N filled with random numbers from [0,1)
interval, uniformly distributed
re = rand(Ne)
ri = rand(Ni)

# creates arrays of size Ne + Ni
a = np.r_[0.02 * np.ones(Ne), 0.02 + 0.08 * ri]
b = np.r_[0.2 * np.ones(Ne), 0.25 - 0.05 * ri]
c = np.r_[-65 + 15 * re ** 2, -65 * np.ones(Ni)]
d = np.r_[8 - 6 * re ** 2, 2 * np.ones(Ni)]
# creates matrix S of synaptic connections
S = np.c_[0.5 * rand(Ne + Ni, Ne), -rand(Ne + Ni, Ni)]

v = -65 * np.ones(Ne + Ni)  # Initial values of v
u = b * v  # Initial values of u ( u = b * -65)
firings = np.zeros((0, 2))  # spike timings will be in this array, used in
plot
avg_v = np.zeros(TIME)

for t in range(TIME):  # simulation of 1000 ms
    I = np.r_[THe * randn(Ne), THi * randn(Ni)]  # thalamic input, normally
distributed random values
    fired = np.flatnonzero(v >= 30)  # indices of neurons which have fired
    if any(fired):
        firings = np.vstack((firings, np.c_[t + 0 * fired, fired]))  #
remember spike timings for plot
        v[fired] = c[fired]  # v ← c for those neurons which have fired
        u[fired] = u[fired] + d[fired]  # u ← u + d for those neurons which
have fired
        I = I + S[:, fired].sum(1)  # gets the input for each neuron from
all neurons that have fired
    v = v + 0.5 * (0.04 * v ** 2 + 5 * v + 140 - u + I)  # step 0.5 ms
    v = v + 0.5 * (0.04 * v ** 2 + 5 * v + 140 - u + I)  # for numerical
stability
    u = u + a * (b * v - u)
    avg_v[t] = np.average(v)
```

```
plt.plot(firings[:, 0], firings[:, 1], ',')
plt.plot(Ne + 0 * v, ',', color="blue")
plt.plot(avg_v)
plt.title(f'Ne/Ni={Ne}/{Ni}, THe={THe}, THi={THi}')
plt.xlabel('time [ms]')
plt.ylabel('fired neurons [indices]')
plt.show()
```