Exercise sheet 01

Alessandro Rizzi & Matteo Zortea

Group G3-A8

Tasks solved

Ex1: a) b) c) d)
Ex2: a) b) c) d)
Ex3: a) b)

Exercise 1

a)

The human brain consumes a power P of 20W, is composed of $N_n = 10^{11}$ neurons that fires with a frequency ν of 1Hz, and $N_s = 10^{15}$ synapses. Thus, the energy consuption per action potential is:

$$E_{\text{act}} = \frac{P}{N_n \nu} = 2^{-10}\text{J},$$

and the energy consuption per synaptic event is:

$$E_{\text{syn}} = \frac{P}{N_s \nu} = 2^{-14}\text{J}.$$

b)

The energy consuption of the simulation per action potential is:

$$E_{\text{act,K}} = \frac{Pt}{N_n \nu} = 2.6\text{J},$$

and the energy consuption per synaptic event is:

$$E_{\text{syn,K}} = \frac{P}{N_n N_{\text{connection}} \nu} = 0.004\text{J}.$$

c)

If we scale the K simulation to the brain sixe in term of spikes we obtain a consuption of

$$P_{\text{K,spikes}} = P_K \frac{N_{\text{neur,brain}}}{N_{\text{neur,K}}} = 1.02\text{GW}.$$

If we now scale it per number of synapses we get

$$P_{\text{K,syn}} = P_K \frac{N_{\text{syn,brain}}}{N_{\text{neur,k}} N\text{connections,k}} = 1.7\text{GW}.$$

d)

If we could speed up the K supercomputer maintaining the energy/flop constant, to simulate a human brain (in term of spikes) we would need:

$$P_{\text{K,speeded}} = P_{\text{K,spikes}} \frac{t_{\text{k}}}{t_{\text{brain}}} = 2.4 \times 10^{12}\text{W}.$$

To do so, I would need approximately the power generated by 1375 nuclear power plants.

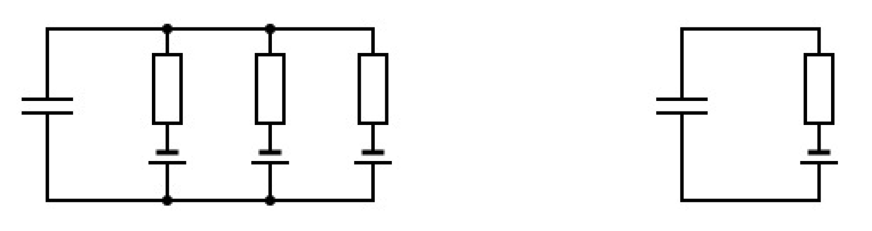
Exercise 2

a)

Let us consider the circuit on the left

```
In [1]: from IPython.display import Image
        Image(filename='Circuit.jpeg')
```

Out[1]:



By applying Thevenin's theorem we can reduce the circuit to an equivalent one (on the right)

The equivalent voltage E_{eq} is the open-circuit voltage on C_{eq} and g_{eq} is the equivalent conductance seen from the load.
We can also make use of the superposition principle to calculate E_{eq}. By keeping only E_1 active, using Ohm's law on the resistances $R_i = 1/g_i$ we get

$$E_{eq}^{(1)} = E_1 \frac{R_2//R_3}{R_1 + R_2//R_3} = E_1 \frac{\frac{R_2 R_3}{R_2+R_3}}{R_1 + \frac{R_2 R_3}{R_2+R_3}} = \frac{E_1}{1 + \frac{R_1}{R_2} + \frac{R_1}{R_3}} = \frac{E_1}{1 + \frac{g_2}{g_1} + \frac{g_3}{g_1}}$$

By applying the same procedure to the other two generators one gets an expression for the equivalents resting potentials

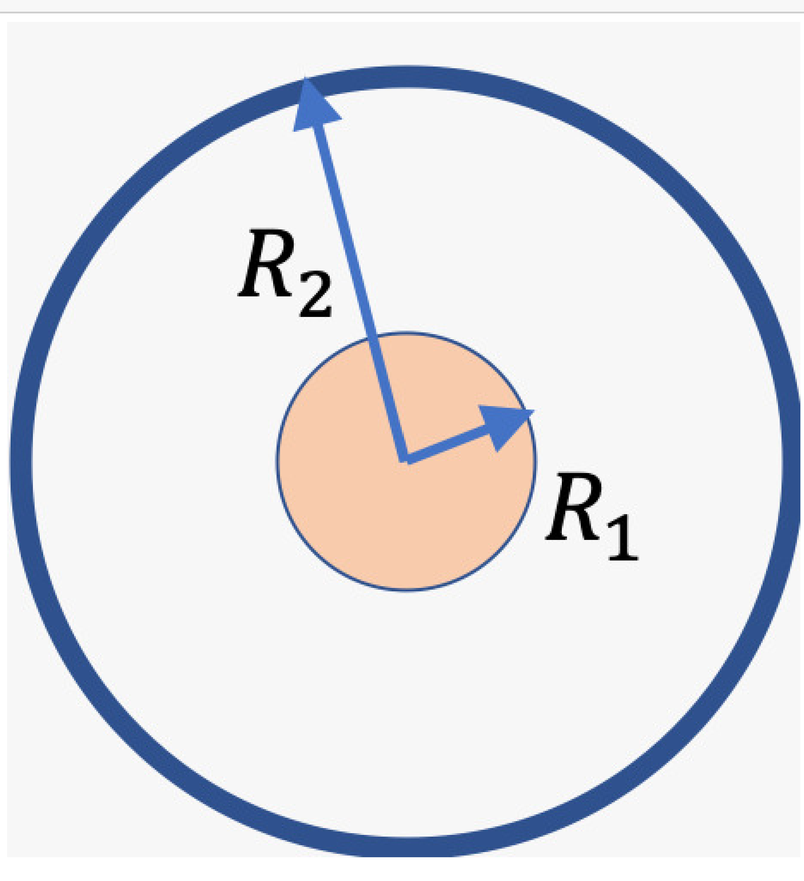
$$E_{eq} = E_{eq}^{(1)} + E_{eq}^{(2)} + E_{eq}^{(3)} = \frac{E_1}{1 + \frac{g_2}{g_1} + \frac{g_3}{g_1}} + \frac{E_2}{1 + \frac{g_1}{g_2} + \frac{g_3}{g_2}} + \frac{E_3}{1 + \frac{g_1}{g_3} + \frac{g_2}{g_3}} = \\ = \frac{g_1 E_1 + g_2 E_2 + g_3 E_3}{g_1 + g_2 + g_3}$$

The equivalent potential is thus a weigthed average of all the potentials with their conductance as a weight.

b)

```
In [2]: from IPython.display import Image
        Image(filename='spherical.jpeg')
```

Out[2]:



The capacitance of a spherical capacitor is

$$C = \frac{4\pi\epsilon}{\frac{1}{R_1} - \frac{1}{R_2}} = 4\pi\epsilon \frac{R_1 R_2}{R_2 - R_1}$$

In our case $\epsilon = 3\epsilon_0$, $R_1 = 10\,\mu\text{m}$ and $R_2 = 10.005\,\mu\text{m}$, hence $R_1 R_2 \approx R_1^2$ and $R_2 - R_1 \approx 5\text{nm}$.

By popping in the numerical values one gets

$$C \simeq 2.670\,\text{nF}.$$

The area of the membrane is

$$A = 4\pi R_2^2 \simeq 1.257 \times 10^{-9}\,\text{cm}^2.$$

Hence, the capacitance for unit area is

$$C_A = \frac{C}{A} \simeq 2.124 \frac{\text{F}}{\text{m}^2} = 212.4 \frac{\mu\text{F}}{\text{cm}^2}$$

c)

We use Nernst's equation:

$$V = \frac{k_b T}{q} \ln \frac{[\text{Na}^+]_{\text{in}}}{[\text{Na}^+]_{\text{out}}}.$$

We want V' to be $V + 10$mV. Assuming that $[\text{Na}^+]_{\text{out}}$ remains constant during the process, we can write:

$$\frac{k_b T}{q} \ln \frac{[\text{Na}^+]_{\text{in}}}{[\text{Na}^+]'_{\text{in}}} = 10\text{mV}.$$

Then, assuming we are at room temperature and knowing that the charge of an ion is the charge of one electronwe find:

$$\frac{[\text{Na}^+]_{\text{in}}}{[\text{Na}^+]'_{\text{in}}} = \exp\left(\frac{10\text{mV}e}{k_b T}\right) = 1.47.$$

The total number of ions inside the cell was:

$$N_{\text{in}} = [\text{Na}^+]_{\text{in}} V_{\text{cell}} = [\text{Na}^+] \frac{4}{3} \pi R_1^3 = 1.26 \times 10^{11}.$$

Thus the number of ions that needs to cross the membrane is:

$$N_{\text{cross}} = (1 - 1.47^{-1}) N_{\text{in}} = 4.0 \times 10^{10}.$$

Comparing N_{cross} with N_{in} we get:

$$\frac{N_{\text{cross}}}{N_{\text{in}}} \approx 0.3$$

.

d)

We can use Nernst's equation to calculate the reversal potential

$$(\Delta V)_{\text{Ca}} = \frac{RT}{zF} \ln\left(\frac{[\text{Ca}^{2+}]_{out}}{[\text{Ca}^{2+}]_{in}}\right)$$

where $z = 2$ is the number of electrons exchanged per process, F is Faraday's constant and R it the ideal gas constant. By inserting the values one gets

$$(\Delta V)_{\text{Ca}} \simeq 140\text{ mV}$$

Exercise 3

a)

Let us first solve the differential equation analitically

$$y'(t) = \frac{1}{\tau}(E_L - y(t)) + \frac{I_e}{C_M}$$

where $\tau = g_L/C_M$ is a time constant.
Let us introduce the quantities

$$A(t) = \int_0^t \frac{1}{\tau}\, dt' = \frac{t}{\tau}$$

$$B(t) = \int_0^t \frac{1}{\tau} \left[E_L + \frac{I_e(t')}{g_L}\right] e^{t'/\tau}\, dt' = \int_0^t \frac{1}{\tau}\, E_L e^{t'/\tau}\, dt' + \tau \int_0^t \frac{\Theta(t'-100)}{g_L} e^{t'/\tau}\, dt' = \\ = E_L\left(e^{t/\tau} - 1\right) + (e^{t/\tau} - e^{100/\tau})\theta(t-100)$$

The solution is given by

$$y(t) = e^{-A(t)}\,[y_0 + B(t)] = e^{-t/\tau}\left[E_L(e^{t/\tau} - 1) + e^{(t-100)\tau}\Theta(t-100)\right] = E_L(1 - e^{-t/\tau}) \\ + (1 - e^{(100-t)/\tau})\Theta(t-100)$$

The Euler integrator is useful to numerically integrate equations of the type

$$\dot{y}(t) = f(t, y(t))$$

One approximates the derivative with a finite difference and obtains

$$y_{n+1} = y_n + h\, f(t_n, y_n)$$

where h is the step length.

```
In [3]: '''
        h: step length
        RHS: right hand side function of the differential equation (can also be vector valued)
        t: time at step n
        y: value of the function at step n
        params: parameters to pass to RHS given as an array
        '''
        def euler_step(h, RHS, t, y, params):
            return y + h*RHS(t, y, params)
```

In our case the right hand side function (RHS) of the differential equation is

$$f(t, y(t)) = \frac{1}{\tau}[\Theta(t-100) - y(t) + E_L]$$

where $\theta(t-100)$ is the Heaviside function and where we have set $g_L = 1$.

```
In [4]: def RHS1(t, y, params):
            tau = params[0]
            EL = params[1]
            if t > 100:
                return (EL - y + 1)/tau
            else:
                return (EL - y)/tau
```

Let us fix some constants for the problem

```
In [5]: h_vals = [30, 20, 10, 5, 0.1] # various step lengths
        t_sim = 200 # simulation time
        tau = 10
        EL = 0 # Resting potential
        y0 = 0 # Initial value
```

The analytical solution is

```
In [6]: def analytical_sol(t, tau, EL):
            y = []
            for tval in t:
                if tval < 100:
                    y.append( EL*(1 - np.exp(-tval/tau)))
                else:
                    y.append(EL*(1 - np.exp(-tval/tau)) + (1-np.exp((100-tval)/tau)))
            return y
```

We can now run the integration for the proposed step lengths

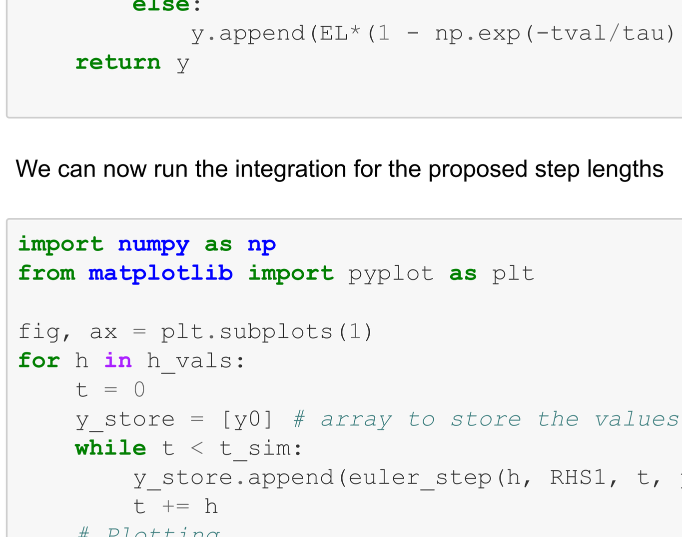
```
In [7]: import numpy as np
        from matplotlib import pyplot as plt

        fig, ax = plt.subplots(1)
        for h in h_vals:
            t = 0
            y_store = [y0] # array to store the values. first value is the initial condition
            while t < t_sim:
                y_store.append(euler_step(h, RHS1, t, y_store[-1], [tau, EL]))
                t += h
            # Plotting
            t_vals = np.linspace(0, t_sim, len(y_store))
            ax.plot(t_vals, y_store, label="h="+str(h))

        ax.set(xlabel="t [ms]")
        ax.set(ylabel="y(t)")
        ax.plot(t_vals, analytical_sol(t_vals, tau, EL), label="analytic", linestyle='dashed') # plot of the a
        nalytical solution

        ax.legend()

        axR = ax.twinx()
        axT = ax.twiny()
        ax.tick_params(direction='in')
        axT.tick_params(direction='in')
        axR.tick_params(direction='in')
        axR.yaxis.set_major_formatter(plt.NullFormatter())
        axT.xaxis.set_major_formatter(plt.NullFormatter())
        plt.show()
```



b)

We now want to use the Euler method to solve a 2nd order ODE, namely the one describing the harmonic oscillator:

$$\ddot{x} = -x.$$

To do so, we decompose the equation in a set of two first order ODEs:

$$y := \dot{x} \quad \Rightarrow \quad \dot{y} = -x \quad \text{and} \quad \dot{x} = y.$$

```
In [8]: h_vals = [1, 0.1, 1E-5] # various step lengths
        t_sim = 10# simulation time
        x0 = 1
        y0 = 0 # Initial values
```

```
In [9]: def euler_step(h, RHS, t, y):
            return y + h*RHS

        fig, ax = plt.subplots(1)
        for h in h_vals:
            t = 0
            x_store = [x0]
            y_store = [y0] # array to store the values. first value is the initial condition
            while t < t_sim:
                x_store.append(euler_step(h, y_store[-1], t, x_store[-1]))
                y_store.append(euler_step(h, -x_store[-1], t, y_store[-1]))
                t += h
            # Plotting
            t_vals = np.linspace(0, t_sim, len(x_store))
            ax.plot(t_vals, x_store, label="h="+str(h))

        ax.set(xlabel="t")
        ax.set(ylabel="y(t)")
        ax.plot(t_vals, np.cos(t_vals), label="analytic", linestyle='dashed') # plot of the analytical solutio
        n

        ax.legend()
        axR = ax.twinx()
        axT = ax.twiny()
        ax.tick_params(direction='in')
        axT.tick_params(direction='in')
        axR.tick_params(direction='in')
        axR.yaxis.set_major_formatter(plt.NullFormatter())
        axT.xaxis.set_major_formatter(plt.NullFormatter())
        plt.show()
```