Neuroinformatics — lab exercises manual

authors:
Eduard Bakstein (TA), Daniel Novák (Lecturer)
http://aid.felk.cvut.cz, Prague 2015-2026

February 16, 2026

Abstract

This is supportive text for labs of Neuroinformatics course at Czech Technical University in Prague, Faculty of Electrical Engineering. Exercises for each task are provided in Matlab language.

We further recommend the book [3], which is nice and easy to read. A good summary is provided by [2]. Particular topics are covered by [1], [4] Some advanced material is covered by [1], [4].

Contents

Bibliography

[1] Michael L. Hines Nicholas T. Carnevale. *The Neuron Book*. Cambridge University Press, 2006.

[2] Dale Purves. *Neuroscience*. Sinauer Associates, Inc., 5th. edition edition, 2011.

[3] Thomas Trappenberg. *Fundamentals of Computational Neuroscience*. Oxford University Press, USA, June 2010.

[4] Werner M. Kistler Wulfram Gerstner. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

0 Mathematical apparatus

We will introduce numerical apparatus, which will be applied during activity modelling of neurons.

0.1 Numerical solution of differential equations

As in other fields dealing with dynamic systems, we will use differential equations throughout this course. Due to the fact that their exact analytical solution can often be difficult or even impossible to obtain, we introduce the approximate numerical solutions in this exercise.

Euler's method

Let's consider the first order differential equation

$$\frac{\delta x}{\delta t} = f(x,t), \tag{0.1}$$

Euler's first order method consist of discretization $\frac{\delta x}{\delta t} = \frac{\Delta x}{\Delta t}$. Lets' take:

$$\begin{aligned} \Delta x &= x(t+\Delta t) - x(t) = x(t_2) - x(t) \\ \Delta t &= t_2 - t, \end{aligned}$$

then we can express eq (0.1) as

$$\frac{\Delta x}{\Delta t} = f(x(t),t)$$

finally

$$x(t+\Delta t) = x(t) + \Delta t f(x(t),t)$$

We can approximate the solution by taking into account the slope of the line at that point. However, if the slope is dependent on t, this will lead to a very rough approximation. In this case, we can refine the solution by other parameters of the Taylor series according to the formula:

$$x(t+\Delta t) = x(t) + \Delta t \frac{\delta x}{\delta t} + \frac{1}{2}(\Delta t)^2 \frac{\delta^2 x}{\delta t^2} + O, \tag{0.2}$$

where O represents all members of the higher orders. Therefore, the second order model is extended by the curvature (2nd order derivative). This approximation should be closer to the analytical solutions.

Runge-Kutta method

In more sophisticated methods, we can additionally refine the solution by estimation not in the point x or $x + \Delta x$, but in the middle of this interval - the so-called *„midpoint method"*. The resulting value should be more representative and lead to a more accurate estimate. Moreover, if we do not have analytical expression of parameters of the Taylor expansion of higher orders, we can again estimate parameters of high orders using numerical methods.

The actual Runge-Kutta method for numerical integration is the 4th order, which combines estimation in the middle of the interval with a numerical estimate of higher order. It can therefore approximate solution of arbitrary functions. This leads to a higher computational cost, but - as we shall see - leads to very accurate estimates of the solution. Runge-Kutta method is implemented in Matlab function `ode45()`.

In python, you can use `solve_ivp` from `scipy.integrate` with mcode method='RK45'.

Exercise 0.1 Our task is numerical approximation of the following differential equation

$$\frac{dx}{dt} = t - x + 1, \tag{0.3}$$

initial conditions: x (0) = 1 using Euler's method (1 and 2 order) and Runge-Kutta methods.

The analytical solution of this equation has the form $x = t + e^{-t}$ [1]. Solve the equation in the interval $\langle 0, 5)$, choose $\Delta t = 0.02$.

[1] You can check e.g. at http://www.wolframalpha.com/, question: „solve differential equation x '= ... "

Task 0.1 (2 b) Plot the solution $x(t) = f(t)$ for $t = 0, \ldots, 2s$ using Euler's method of first and second order [2] (X_{Euler}) and plot into graph along with the analytical solution.

Task 0.2 (1 b) Solve the task using Runge-Kutta x_{Runge} and add the solution to the same graph.

Hints: in the function `ode45`, the first parameter is a callback function, which desribes the equation to be solved. In this case the callback can be easily defined using so called *function handle* and *anonymous function*[3] as `ode_func = @(t,x,flag) 1-x+t;`

Task 0.3 (1.5 b) Plot the mean absolute error over the interval $\langle 0, 1 \rangle$ of each numerical method as a function of the size of the integration step Δt.

Task 0.4 (1.5 b) Plot error of each numerical method at time $t = 1$ as a function of the size of the integration step $\Delta t \in \langle 0.001, 1 \rangle s$.

[2] when solving the second order Euler method you can proceed in two ways. 1) Derivating analytic function $f(x, t)$. 2) estimate value of $f'(x, t)$ using the slope $f(x, t)$ between points t and $t +$ *Deltat*

[3] see for example. http://www.mathworks.com/help/matlab/matlab_prog/creating-a-function-handle.html

1 Neuron models

In this block, we will describe properties of the action potential and behavior of individual neurons. We will implement several membrane and neuron models of differing level of complexity and investigate how the simplifying assumptions affect the resulting action potentials or spike-trains.

1.1 Model of membrane and synapse: simplest case

Considering that the action potentials propagate as changes in electrical potential on the cell membrane, it seems natural to model the membrane as an equivalent electrical circuit. The simplest model that we introduce in this exercise works with single type of ion channels only - the leakage chloride channels (always open) - and consists of an RC circuit and a voltage power source, representing the membrane resting potential (*Nernst potential*) . As we shall see, its response to the input excitation represents only a part of the true membrane behavior but still useful for understanding the basic concept.

Exercise 1.1 (RC model) Model the membrane behavior using RC model - see figure 1. We assume that the membrane is an infinite surface with electrical properties defined per unit area: capacitance $[\mu F/cm^2]$ and channel conductivity $[ms/cm^2$, *millisiemens per square centimeter*$]$ [4]. We can imagine we are performing an experiment with a very small sample of the membrane of the area A, which we stimulate with current I_{stim}.

Our stimulation membrane current I_{stim} is a rectangular pulse signal $10pA$ lasting $20mS$. We will use the μA as the basic unit for this assignment, making the peak stimulation current $I^{pk}_{stim} = 10pA = 10^{-5}\mu A$. Note that on the contrary, I_{Cl} and I_C are defined as current densities in $\mu A/cm^2$[5].

The membrane parameters are the following:

capacitance: $C_m = 1\ \mu F/cm^2$,
conductance: $g_{Cl} = 0.3\ mS/cm^2$,
time constant : $\tau = C_m/g_{Cl}$,
membrane area : $A = 1 \cdot 10^{-6}\ cm^2$
Nernst potential of Cl: $V_{Cl} = -68\ mV$,
initial conditions: $V(0) = -68\ mV$, $I_{Cl}(0) = 0\ \mu A/cm^2$, $I_C(0) = 0\ \mu A/cm^2$.

Task 1.1 (5 b) Plot the time courses of $V(t)(= \phi_{in} - \phi_{out})$, $I_{Cl}(t)$, $I_C(t)$ over time interval $\langle 0, 40\rangle ms$, $\Delta t = 0.01ms$

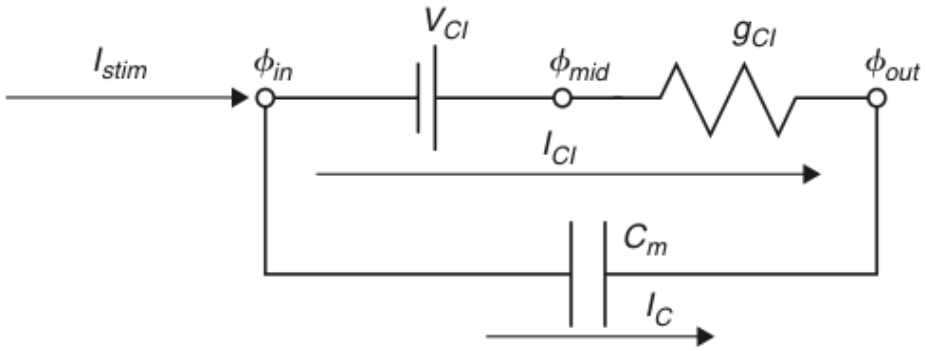


Figure 1: Membrane model with Cl leakage channel

[4]The conductance is an inverse value of resistance, $G = R^{-1}$, as well as its unit, siemens, is inverse value of Ohm, $[S] = [\Omega^-1]$. Conductivity and resistivity are then corresponding values per unit area (in this case of the membrane).

[5]As I_{Cl} and I_C are current densities, it would be more appropriate to use ϱ instead of I. However, we will stick to the convention used in [3] and use I.

Hence

$$I_C(t) = C_m \frac{dV}{dt}(t) \tag{1.1}$$

$$I_C(t) = \frac{I_{stim}(t)}{A} - I_{Cl}(t) \qquad \rightarrow I_{stim} = A \cdot I_C(t) + A \cdot I_{Cl}(t) \tag{1.2}$$

$$I_{Cl}(t) = g_{Cl}(V(t) - V_{Cl}) \tag{1.3}$$

We can express the time constant of the RC circuit as

$$\tau = \frac{C_m}{g_{Cl}} \tag{1.4}$$

Combining the above equations we get

$$C_m \frac{dV}{dt} = \frac{I_{stim}(t)}{A} - g_{Cl}(V(t) - V_{Cl}) \tag{1.5}$$

Or expressed using time constant τ from 1.4 we get

$$\tau \frac{dV}{dt} = V_{Cl} - V(t) + \frac{I_{stim}(t)}{A \cdot g_{Cl}} \tag{1.6}$$

Euler's method (forward)

$$\tau \frac{V(j) - V(j-1)}{\Delta t} = V_{Cl} - V(j-1) + \frac{I_{stim}(j-1)}{A \cdot g_{Cl}} \tag{1.7}$$

$$V(j) = V(j-1) + \frac{\Delta t}{\tau}[V_{Cl} - V(j-1) + \frac{I_{stim}(j-1)}{A \cdot g_{Cl}}] \tag{1.8}$$

Exercise 1.2 (EPSP model) The task is analysis of a model depicted in figure 2. This model incorporates so called „excitatory postsynaptic potential" (EPSP) simulating behavior of membrane dendrite of the postsynaptic neuron after receiving excitation. Synapse is modeled by variable conductance g_{syn}. In other words: the model contains additional channels, whose conductance is neurotransmitter-controlled and further voltage and time-dependent (literally in the moment of receiving neurotransmitter the channels are opening, hence their conductance is increasing - see result of eq (1.9)) with time constant $\tau_{syn} = 1\ mS$. The parameters are: stimulation current $I_{stim} = 0$, $V_{syn} = 10\ mV$. At time $t = 1\ ms$ the neurotransmitter is released, hence $g_{syn}(1 + \Delta t) = 1$. Initial conditions $V(1) = 0$, $I_{syn} = 0$, $g_{syn}(1) = 0$, $g_L = 1$.

Additionally to the model presented in the previous task (RC model), we model the response of ion channels on the post-synaptic membrane to a neurotransmitter, released by the pre-synaptic neuron. We assume immediate opening of all ion channels after the neurotransmitter released at t_{pre} and additional reaction time t_{delay}. Then, individual channels close randomly at constant rate. The gated channels are thus modeled according to:

$$\tau_{syn} \frac{dg_{syn}(t)}{dt} = -g_{syn}(t) + \delta(t - t_{pre} - t_{delay}), \tag{1.9}$$

where $\delta(t)$ is a dirac impulse at time t. In this case, we omit the stimulation current and the Nernst potential of leakage chloride channels $V_{Cl} = 0$, as well as the stimulation current.

Task 1.2 (0 b) Download the file `Ex12_epsp.m` / `1_2_epsp.ipynb` from the course website. Plot dependence $V(t)$, $I_{Cl}(t)$, $I_C(t)$, $I_{syn}(t)$. Explain trends and compare them with the previous task 1.1.

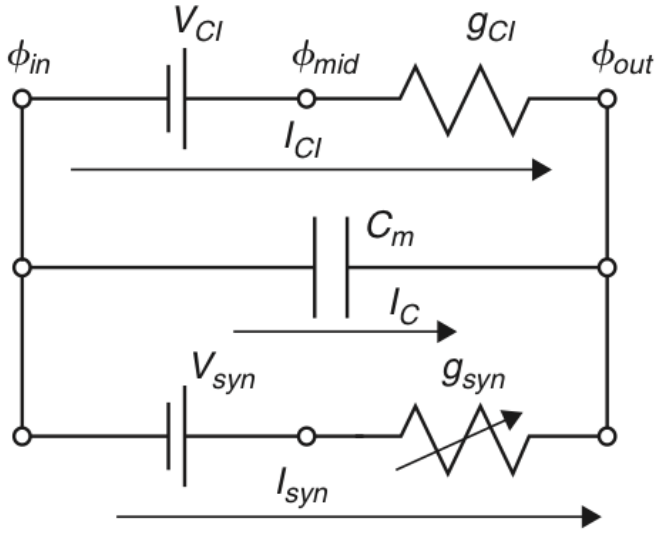


Figure 2: EPSP: Synapse model with leakage Cl channel

1.2 Mebrane and synapse modelling 2: Hodgkin-Huxley

Exercise 1.3 (Hodgkin-Huxley) The main aim of this exercise is analysis of the Hodgkin-Huxley(HH) model depicted in Figure 3. Compared to the previous models, HH incorporates ion channels whose conductivity is dependent on both time and voltage: the Na^+ a K^+ channels. The model is based on the following set of equations:

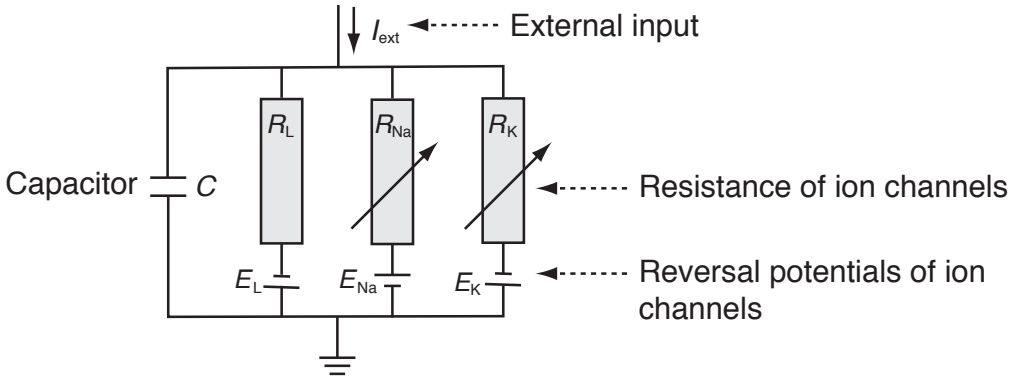


Figure 3: Hodgkin-Huxley model. Each branch represents resting potential a and resistivity of a particular channel: E_L, R_L – always open channels („Leakage channels"), E_{Na}, R_{Na} – voltage-controlled sodium channels, E_K, R_K – voltage-controlled potassium channels.

Current through ion channels can be described by Ohm's law:

$$I_{ion} = \hat{g}_{ion}(V - E_{ion}) \qquad \left(= (V - E_{ion})/\hat{R}_{ion}\right), \tag{1.10}$$

where $\hat{g}_{ion}$ is the maximum conductance of ion channel

Next, we introduce auxilary variables dependent on voltage and time $n(V,t)$, $m(V,t)$, $h(V,t)$, resulting in the following conductances:

$$\hat{g_K}(V,t) = g_K n^4 \tag{1.11}$$
$$g_{\hat{N}a}(V,t) = g_{Na} m^3 h \tag{1.12}$$

Combining eq.1.12-1.10 we get

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = -g_{\mathrm{K}}n^4(V - E_{\mathrm{K}}) - g_{\mathrm{Na}}m^3h(V - E_{Na}) - g_{\mathrm{L}}(V - E_{\mathrm{L}}) + I_{ext}(t) \tag{1.13}$$

Next, we define time constants

$$\tau_{\mathrm{n}}(V)\frac{\mathrm{d}n}{\mathrm{d}t} = -[n - n_0(V)] \tag{1.14}$$

$$\tau_{\mathrm{m}}(V)\frac{\mathrm{d}m}{\mathrm{d}t} = -[m - m_0(V)] \tag{1.15}$$

$$\tau_{\mathrm{h}}(V)\frac{\mathrm{d}h}{\mathrm{d}t} = -[h - h_0(V)] \tag{1.16}$$

For each variable we get

$$\frac{dx}{dt} = -\frac{1}{\tau_x(V)}[x - x_0(V)], \qquad x \in \{n, m, h\} \tag{1.17}$$

and after substituting Euler's numerical integration we get

$$x\left(t + \Delta t\right) = \left(1 - \frac{\Delta t}{\tau_x}\right)x(t) + \frac{\Delta t}{\tau_x}x_0. \tag{1.18}$$

Under the initial conditions:

$$x(0) = \alpha \cdot \tau_x, \qquad \tau_x = \frac{1}{\alpha + \beta}, \qquad x \in \{n, m, h\} \tag{1.19}$$

$$\alpha_n = \frac{10 - V}{100\left(e^{\frac{10-V}{10}} - 1\right)}, \qquad \beta_n = 0.125e^{-\frac{V}{80}} \tag{1.20}$$

$$\alpha_m = \frac{25 - V}{10\left(e^{\frac{25-V}{10}} - 1\right)}, \qquad \beta_m = 4e^{-\frac{V}{18}} \tag{1.21}$$

$$\alpha_h = 0.07e^{\frac{V}{20}}, \qquad \beta_h = \frac{1}{e^{\frac{30-V}{10}} + 1} \tag{1.22}$$

Task 1.3 (0 b) Download model implementation file `Ex13_HH.m` / `1_3_hh.ipynb` from course website. Visualize time output of the model (membrane voltage V), which is stimulated by constant current $I_{ext} = 30\mu A/cm^2$ (default settings). The simulation starts at time -30ms to get steady system. Plot output for time $0 - 100ms$.

Task 1.4 (1 b) Expand the visualization by adding time trend of conductance g for each channel type (describe different time series using `legend`)

Task 1.5 (2 b) Visualize dependence of firing frequency on external constant current I_{ext} - so called *activation or transfer function*. Set the input current I_{ext} in the range $0 - 15$ $\mu A/cm^2$. To obtain the result, compute the firing frequency after reaching the steady state (hint: use fce `diff`, `find` alt. `findpeaks`). What shape does the activation function resemble? For which values of the current I_{ext} will the shape of the activation function change significantly (discuss the result)?

Task 1.6 (2 b) Adding noise to the input current will change behaviour of the model. Visualize the membrane voltage V and activation function as a response to input current with added noise. Use $I_{ext} = 30\mu A/cm^2$ and white noise (`randn`) with standard deviation 60 and zero mean. For plotting activation function use $I_{ext} = \langle 0, 15\rangle \mu A/cm^2$, noise with standard deviation 30. How did the resulting output change? Try out for different noise levels.

Task 1.7 (2 b) Plot ISI histogram for the input current $I_{ext} = 30\mu A/cm^2$ and noise with standard deviation 60. Compare to the case with zero noise and discuss the results.

1.3 Spike train modelling

We showed in the previous tasks, how does noise on the input affect temporal and frequency characteristics of a neuron. Noise signal on neuron input is much more realistic than constant current - mainly due to the fact that neuron input is formed by a summary signal from many synapses of pre-synaptic neurons, connected to its dendrites. In this section, we will simulate such signal and use it as input to a neuron model.

You have learned in the lectures, that interspike intervals (ISI) of typical cortical neurons exhibit log-normal distribution. Before we start modelling spiketrains, let us verify this on a model.

Exercise 1.4 (LIF neuron model and Poisson spiketrain) The complex models from previous exercises, such as Hodgkin-Huxley, are able to model the action potential shape based on physiological modelling of ion channels' behavior. If we want to model more complex neural networks, even simpler models (such as Wilson model) are overly complex for such application. When modelling neural networks, the accurate shape of action potential is not as important (we will see the shape of the post-synaptic potential can be simply modelled using α functions) as is the spike timing. For these purposes, the *Leaky integrate and fire* neuron is often used. The basics of this model includes simple integrator, which is reset to resting potential when a preset threshold is exceeded.

The LIF model uses the simple RC membrane representation, defined previously in Eq. 1.6, which can be simply represented as:

$$\tau \frac{dV}{dt} = V_{Cl} - V(t) + I_{stim}(t) \cdot R_{Cl}, \tag{1.23}$$

where the $V(t)$ is membrane voltage over time, I_{stim} the external stimulation current and V_{Cl} and R_{Cl} the half-cell potential of the always-open ion channels and their rezistivity, respectively. The τ is the time constant defined as the product of membrane capacitance and rezistivity $\tau = C * R_{Cl}$.

Apart from this basic dynamics, the model includes a defined threshold voltage V_Θ. Whenever this threshold voltage is achieved, a spike is emitted and the membrane voltage is reset to the resting potential.

Task 1.8 (1 b) Download LIF model from the course website - `Ex14_lif.m` / `1_4_lif.ipynb`. Show model output for constant input current (default setting). Plot output voltage, along with spike times (= reset times) over time. Compute firing rate and display an ISI histogram.

Task 1.9 (2 b) Modify previous task by applying input noise with mean 12 and standard deviation 100, instead of constant current (function `randn`). Simulate for times in the range $(0 - 10s\rangle$ and integration step $dt = 0.01$. Set bin number to 60 when computing histograms. Convert histograms to probability density function estimates ($\sum pdf = 1$).

Task 1.10 (2 b) Fit lognormal distribution to the data from the previous task. The lognormal distribution is given by:

$$pdf^{lognormal}(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(log(x)-\mu))^2}{2\sigma^2}} \tag{1.24}$$

One possible solution would be to fit the analytical pdf to ISI histogram from the previous task. This, however, would be completely wrong, as one point in the histogram does not mean one, but many data points (think about this for a while!).

Thus, we will use the function `lognfit`, which estimates lognormal distribution parameters using maximum likelihood estimates. Based on these estimates, calculate the lognormal pdf and display along with the histogram. (You will probably have to adjust the bin number to get a good representation)

2 Real data analysis, modelling of neuronal populations

In this block, we will analyze real μEEG (*micro-EEG*) data, recorded from brains of Parkinson's disease patients. We will compare the data to our simulations from the previous block and we will analyze correlation with other biosignal - the electrooculogram (EOG).

Further on in this section, we will simulate more complicated network of artificial neurons and (so called *spiking network*) and analyze its behavior under varying conditions.

2.1 Spiketrain modelling II

Exercise 2.1 (Lognorm spiketrain) We showed in the last exercise how does LIF neuron model respond to a noisy input. In this task, we will show that a more representative signal, produced as a combination of presynaptic potentials, really resembles gaussian noise in many ways.

You know from the lectures, that the *Poisson spiketrain*, - a spiketrain where the number of events per unit time follows poisson distribution - has exponential distribution of interspike intervals (ISIs). Further, the physiological constraints prevent neurons from firing earlier than after the *refractory period* - a fact we have to incorporate in our implementation.

In this example, we will generate a set of spiketrains using exponential distribution, taking refractory period into account, and mix them to create an artificial $muEEG$ signal.

Task 2.1 (1 b) Generate 100 000 ISI intervals from the exponential distribution with mean $E[X] = 50ms$ (hint: use `exprnd` function). Display ISI histogram.

Remove all ISI values lower than the refractory period of 1ms. Display ISI histogram and compare to the one before refractory period filtering. How many spikes were removed? Compute mean firing rate and coefficient of variation before and after filtering and compare.

Convert the resulting ISI ingervals to a spiketrain (logical vector with ones at spike positions hint: `cumsum`). Use time step of $dt = 0.1ms$. Check for errors by plotting the result

Task 2.2 (2 b) To convert binary spiketrain to time course, we will use convolution with $\alpha - function$, which will play the role of a post-synaptic spike template.

Generate time course of the $\alpha - function$ according to:

$$I(t) = c \cdot t \cdot e^{-\frac{t}{t_{peak}}} \tag{2.1}$$

$$c = \frac{I_{peak}}{t_{peak}} e^1 \tag{2.2}$$

where $I_{peak} = 0.5$, $t_{peak} = 1ms$. Compute the function values for times in the interval $t \in \langle 0, 30 \rangle ms$, the time step should be again $dt = 0.1ms$. Plot the output. After assessing the result, you may reduce the time range if appropriate.

Convolve the alpha function with Poisson spiketrain from the previous task (`conv`). Check the result again by plotting.

Task 2.3 (2 b) Divide the time course from previous task into 1000 signals of the same length(`reshape`). In this task, we will simulate mixing of presynaptic potentials (represented by the simulated neurons), at dendrites. The resulting summary signal will be given by a linear combination of the individual spiketrains. Coefficients will be given by weights w, which will be modelled as random from exponential distribution with $\lambda = 1$.

In order to have better control of the amplitude of the resulting signal, we will normalize the weights so that their sum equals a constant k. Set the value $k = 400$.

Combine the 1000 spiketrains into one signal, using a linear combination with weights w. You can represent the linear combination as matrix multiplication (elegant + much faster in Matlab).

Compare the resulting signal to gaussian noise (time and distribution). What are the key differences?

Task 2.4 (1 b) Apply the signal from previous task to the input of LIF neuron from Exercise and observe the output. What changes do you observe when you change the normalizing constant $k = 150$. What does the ISI histogram look like, now? Investigate and comment behavior of the LIF neuron.

2.1.1 Artificial and real signal - comparison

Exercise 2.2 (Real signal) We have now generated artificial signal as a combination of Poisson spiketrains, convoluted with alpha function. In these tasks, we will compare the simulated signal to real μEEG, recorded from basal ganglia of a Parkinson's disease patient during deep brain surgery. We will show, that this simple method of data generation is not far from reality.

Task 2.5 (1 b) Use artificial signal generated in the previous task. It is a time course with sampling rate $f_s = 10kHz$. Download μEEG data from the course website. The .mat file contains variable `realmEEG` with 1s recording of activity around Thalamus ($f_s = 24kHz$). Display both signals in parallel plots (with correct time coordinates). Investigate both signals.

Task 2.6 (1 b) The real signal, used in the previous task, was an extracellular recording. Conversely, the signal we generated was a simulation of EPSP signal (that is why we used the alpha function)! In this task, we will generate artificial extracellular signal in a similar way. Use real spike shapes, downloaded from the course website instead of the alpha function. There are 5 spike shapes in the data sample - use random spike shape and random amplitude for each simulated neuron. In your simulation, use the same sampling frequency as in the real signal, i.e. $f_s = 24kHz$. Generate the composite signal from at least 100 (better around 1000) simulated neurons and compare it to the real microelectrode recording.

Task 2.7 (2 b) From visual comparison of time courses of real and simulated data, we get a rough idea of how similar these two signals are. Your next task is to try out some quantitative comparisons. Choose and implement appropriate methods of signal comparison. You can use e.g.: frequency spectrum, amplitude distribution etc.

2.2 Real data processing: Spike sorting

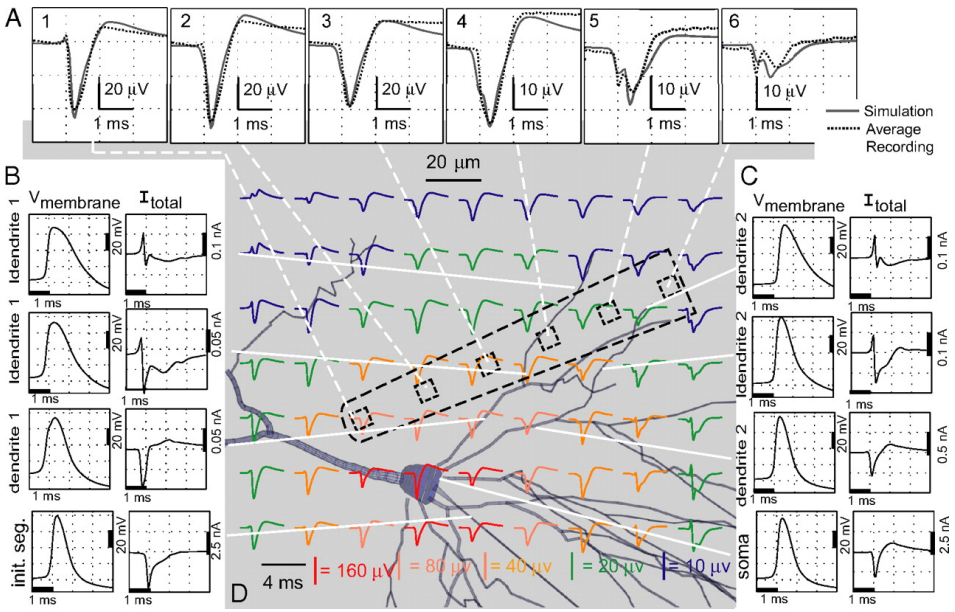
Exercise 2.3 (Spike sorting) In this exercise we will focus on spike detection and spike sorting (i.e.) in real μEEG data.

Spikes and background noise

The real μ EEG data we will use in this exercise were recorded using microelectrodes placed in the brain of patients suffering from Parkinson's disease, specifically in the area of the thalamus. As we know, the recorded signal consists of activity of a large number of neurons in the vicinity of the electrodes - this is source of a background noise[6] - from which we can get an approximate idea of the activity of the population in the area. However, we are usually interested in neurons in the vicinity of the electrode - here we can recognize individual action potentials of individual neurons and thus accurately analyze their activity. The recorded AP's (*spikes*) from near neurons have higher amplitude and we are usually able to distinguish them from the background - this procedure is called *spike detection*. Then, to assign individual spikes to different neurons we need a method called *spike sorting*, which usually uses unsupervised learning techniques based on the spike shape.

Spike sorting

Suppose we have a μEEG signal with detected spikes. Our goal is to classify spikes to individual neurons. Due to the spatial arrangement of different neurons relative to the electrode (plus possibly their individual characteristics) the shapes of recorded spikes originating in different neurons differ. This can be used to sort spikes on the basis of their shape. Given that the correct solution is unknown and we can not know (the number of neurons in the vicinity of the electrode can not be detected by the available imaging techniques, as well as correct assignment of individual spikes to the neurons), we need to use unsupervised learning methods, namely clustering. Regarding the choice of features describing the shape of individual spikes, we have a great range of options, including amplitude, duration, as well as various transformations - such as wavelet, fourier and many more.



Dependence of AP's shape on location to electrodes. [7]

In this task we will use for spike detection a method designed by R.Q. Quiroga that is available from WaveClus toolbox[8]. The methods detects each AP based on amplitude tresholding. The treshold is determined by the number of standard deviation. Furthremore, signal is filtered by band-pass filter in range $300 - 3000Hz$. You can download a script `cv7_amp_detect.m`.

[6] Frequently the noise carries the vast majority of the signal energy.

[8] source code is available here http://www.vis.caltech.edu/~rodri/Wave_clus/Wave_clus_home.htm. Those who are interested can find more information here, specially in „Introduction")

Task 2.8 (1 b) Download `Ex23_spikesorting_data` and load it into the MATLAB. Sampling frequency is again $fs = 24kHz$. Plot the data and roughly estimate the spikes position. How long is record (in seconds)?

Download a script for spike detection `Ex23_amp_detect.m`. Launch the sript using microelectrode data and plot the time series along with spikes detected with the treshold used for detection (output variable is `index` and treshold is `thr`). .

Task 2.9 (1 b) Variable `spikes` returns spikes detected in rows sorted by max value. Plot a few first spikes and estimate the number of neurons. Design at least one feature that will be used for spikesorting. Set manually threshold and sort the spikes according your feature. Plot the average spike for each group.

Task 2.10 (3 b) Now just imagine the task when we need to sort the signal without manually setting the treshold. Suggest and calculate at least one more feature. Plot the spikes in feature space and color them based on your sorting from the previous task. Was your threshold calculated well?

Cluster your features by k-means algorithm (function `kmeans` from Statistics Toolbox). Estimate the number of clusters and show results in feature space. Next, show all signals and average signal for each cluster.

Analyse clustering results. Do you consider your features as sufficient? Is there any possible improvement? Was any feature standardization necessary? If so, how did it contribute to clustering results? Comment on these issues in your report.

2.3 Analysis of real data: Eye movements

Exercise 2.4 (Eye movements) In this exercise, we will show how to analyze reaction of single neuron to a stimulus. The data comes from experiments, where we evaluate connection between eye movements and neuronal activity in human Basal gangial. If the activity of given neuron correlates with eye movements, we should see a change in its firing pattern when eye movements are present. The data come from ten repetitions of the following experiment: The subject is looking at a PC screen, which is black at the beginning. At given time, a white cross appears and the task of the subject to focus at it. Electrooculogram (EOG) is recorded throughout the experiment using electrodes placed near subject's eyes, as well as micro EEG activity in the area of basal ganglia. Once the experiment is over, the data goes through spike detection and spike sorting and neurons, correlating with EOG are identified.

Task 2.11 (2 b) Download exercise data from the course website. Data contain recording of activity from singe neuron in ten repetionions of the same experiments + corresponding EOG. The variable `firingTimes` contains spike times of given neuron, relative to the time of stimulus (cross appears). Variables `eogTimes` and `eogVals` specify corresponding EOG signal for each repetition.

Note: plotting of EOG and corresponding spike times can be done as follows:

```
plot(eogTimes{1},eogVals{1});
hold on;
stem(firingTimes{1},ones(size(firingTimes{1})))
```

Plot firing of the neuron over time, aligned with the stimulus (time 0). What can we say about the activity? How many times did the neuron fire during all repetitions before and after the stimulus?

Task 2.12 (3 b) Use an appropriate way to aggregate activity over all repetitions. (e.g. number of spikes in time windows, instantaneous firing rate, etc.). How does the activity change right after the stimulus? Is there some trend even before the stimulus? How long does it take for the activity to drop and how long to return to original level? Be creative!

3 Information coding and transfer in the brain

In this block, we will model some of the processes taking part in transferring and storing information in the brain. We will take a look at one of the weight adaptation (i.e. learning) mechanisms, simulate some small spiking networks and more.

3.1 Hebbian learning

All simulations that we did until now were using fixed synaptic weights - such approach ignores the *synaptic plasticity*, i.e. the adaptation of neuron's input weights. One model for weight adaptation is based on correlation of pre-synaptic and post-synaptic firing and is based on the theory by Donald Hebb.

Exercise 3.1 (Hebbian learning) This exercise will be based on the codes implemented for 2.1 („Lognorm spiketrain"), which will be adapted to include synaptic weight adaptation based on Hebb's learning. Simply put, the theory states that synapses, that fired shortly before the post-synaptic spike was emitted, are strengthened, while those firing after the post-synaptic spike are weakened.

The weight adaptation will be based on the following equation:

$$w_{n+1} = w_n + \delta w - f_{decay}, \tag{3.1}$$

where w_n are actual synaptic weights, f_{decay} is forgetting constant and δw is weight change computed according to:

$$\begin{aligned} \delta w &= \alpha_{learn} \cdot pm \cdot e^{-pm\frac{\Delta t}{\tau}} && (3.2)\\ \Delta t &= t_{post} - t_{pre} && (3.3)\\ pm &= sign(\Delta t) && (3.4) \end{aligned}$$

Task 3.1 (2 b) Implement function for weight adaptation based on 3.2. Let the parameters be $\tau = 40$, $\alpha = 2$, display the function in the range $\langle -100, 100 \rangle ms$ (do not forget axis labels). Comment the result.

Task 3.2 (4 b) In this task, we will use code for LIF neuron, fed with a large number of pre-synaptic spiketrains from exercise 2.1. Implement code for Hebbian weight adaptation. All synaptic weights will be adapted every time the LIF neuron fires. Use constatnts from the previous task, set forgetting constant to $f_{decay} = 0.01$ a and set initial sum of weigthts to $k = 600$.

Display time course of the summary pre-synaptic potential in time, including post-synaptic spikes, generated by the LIF neuron. Display the changes of weights over time (you can use e.g. `imagesc` function). How do the weights change over time? What is the change during the first 500ms? And during the following time? What effect does the value of forgetting constant have? Experiment with parameter settings and comment the results.

3.2 Simulated spiking networks

The previous exercises focused mostly on modelling and examination of single-neuron models and their behavior. We showed factors that affect the variability of firing pattern of a neuron. It is assumed that firing pattern and spike timing is actually the main information carrier in the brain. To get a bit closer to the behavior of real neural networks, we will study artificial spiking neural networks, composed by many neurons.

Exercise 3.2 (Spontaneous activity) In this exercise, we will model spontaneous activity of a random neural network, composed by Izhikewich neurons - we will reproduce outputs of original Izhikewich research paper (available online: `http://www.izhikevich.org/publications/spikes.pdf`). The result will be a very brief text document, summarizing results of individual tasks.

Task 3.3 (2 b) Study Izikhewich model from the original research paper. Explain, what do the variables u and v mean and what do the constants a, b, c, d do.

Study the bottom two lines of Fig 2. What do these plots represent? What mainly affects the time courses?

Task 3.4 (2 b) Copy and paste source code of simulation of a random network of 1000 neurons from the paper to matlab. Run the code and investigate it thorougly. Answer the following simple questions:

1. What types of neurons (and how many) have been used in the simulation
2. What unit do the constants a, b, c, d have and what does their size correspond to?
3. What represents the constant S and which way does it implement different neuron types?
4. How is the input current of different neuron types modelled?
5. Which part of the code represents resetting the membrane voltage to resting potential when a threshold voltage is achieved, as defined by Equation (3)?

Task 3.5 (2 b) Extend the implementation of the paper: add storing of post-synaptic potentials of individual neurons (it already stores firing times). Extend the program with the following (put the output figures in the report)

1. Display the time course of output voltage on 2 neurons of different type (select whichever you want). Are there any key differences in the time course?
2. Display power spectral density (e.g. `pwelch`) of averaged output voltage of the whole population. Restrict the spectrum to the range $1 - 100Hz$. What can we say about the character of the spectrum? What is the activity like in terms of brain waves (How about e.g. the alpha range: $8 - 13Hz$)? Place your comments into the report.

3.3 Self-organizing maps (SOM)

In the previous exercises, we have understood and experimented with weight adaptation - the network learning process. In this exercise, we will show a different mechanism how information may be represented in the nervous system.

The idea is based on experiments performed in of associative learning, observed in human visual cortex by Hubel and Wiesel in 1960's (Nobel prize 1981), who showed how neighboring neurons in visual cortex, organized into so-called hypercolumns, react to similar stimuli. In one of their discoveries, they showed how neigboring neurons react to visual stimuli of different orientation (line angle).

In this task, we will model a self-organizing map, which builds on this principle: that the neighboring neurons should „fire" for similar stimuli. This exercise will be based on the *self-organizing maps* (*SOM*, sometimes also *Kohonen networks/maps*), which implement this idea in a simple way. As we will see, the SOMs can be used for data clustering and unsupervised learning or data projections. As opposed to all previous models, are not spiking-networks, working in discrete time steps.

SOM

Compared to k-means, we can imagine SOM as a projection of source data to a space of lower dimension. This can be useful for data visualization or aggregation. SOM is composed of neurons, distributed in space. Apart from position in space, each neuron is characterized by a weight vector of the same dimension as the input data (which is equal to the feature count).

The aim of SOM learning is to achieve similar response of neighboring neurons to given stimulus. At the beginning, all weights are initialized randomly[9]. For each input sample (=vector of feature values), we identify the neuron with weight vector closest to the input sample. This neuron is declared a winner. Weight vectors of all neurons in the neighborhood of the winner vector are adapted towards current input vector. The result is heavily dependent on the neighborhood function, defining weight adaptation step for given distance from the winner neuron. During learning, input data are presented to the network in multiple cycles, so that higher network consistency is ensured.

When the network is applied (i.e. during classification phase), weights are held fixed. For each input sample, distance between the input vector and each neuron's weight vector is calculated. Typically, this distance is visualized on a color scale. Based on this visualization, it can be defined which region of the network responds to specific input type.

We will use Matlab's builtin Neural Network Toolbox.

Exercise 3.3 (SOM) We will make use of the classic Fisher's iris dataset http://en.wikipedia.org/wiki/Iris_flower_data_set, representing blossom properties of three species of iris (flower), which are rather hard to distinguish visually.

Task 3.6 (2 b) Perform clustering of the iris dataset using SOM. Visualize the results. Follow this procedure:

1. Load iris dataset from matlab repository using `load iris_dataset`. This will load the variables `irisInputs` (feature values) and `irisTargets` (true labels - iris species, annotated by an expert)
2. Initialize the SOM: `net = selforgmap([8 8], "topologyFcn","hextop");`. What do the individual parameters mean? What does `topologyFcn` mean and what possibilities does the toolbox offer? Try out different topologies and visualize them using `plotsomtop`.
3. Learn the network on iris data using `net = train(net,irisInputs);`
4. Visualize position of individual neurons in the dimension of their weights using `plotsompos`
5. Visualize hit counts of individual neurons using `plotsomhits`. Show the same plots for different classes (iris species, specified by `irisTargets`). Compare the plots
6. Try out other possibilities of SOM visualization (you can use the GUI window which opens during learning)

[9] Better initialization can be achieved e.g. by principal component analysis (PCA) or heuristic knowledge about the data.

In the second part of the exercise, we will do clustering of vowels using SOM and visualization of the results.

Task 3.7 (3 b) Download publicly available dataset „Vowel Recognition“ (originally from UCI repository, now on course website). You need specifically the file`vowel-context.data`, other files are useful for data understanding.

Load the file to matlab using `D = importdata filePath/vowel-context.data`. Data instances (examples) are in rows, attributes (features) in columns. We will need the 4th-13th feature. We will handle the last column separately - these are the data labels. We will also throw away the first three features. We will transpose the data (').

The next step is SOM initialization using reasonable parameters and learning. Observe the results for different classes and compare the results. The procedure is similar to the previous exercise 3.6.

4 Bioinspired and neuromorphic algorithms to model visual attention

In this block, we investigate an interesting and actual application scenario of the spiking networks for the purpose of attention modeling within the area of neuromorphic algorithms.

4.1 Playing around with bioinspired and neuromorphic algorithms

Robotics is entering a new era of intelligence, where traditional approaches to perception and computation are no longer sufficient to meet the demands of real-time, energy-efficient, and adaptive systems. To enable more complex behaviours and facilitate the seamless integration of robots into daily human environments, it is imperative to significantly reduce the power consumption of robotic applications.

Event-driven sensing and neuromorphic computing offer disruptive solutions to these challenges by mimicking how biological brains perceive and process their surroundings. Unlike conventional systems that process data at fixed intervals, event-driven systems react only when a change occurs in the environment. This shift enables low-latency, low-power, and highly efficient processing, essential for autonomous robots, smart city infrastructure, and space exploration, fields where quick adaptation, energy efficiency, and data reduction are key.

Event-based information processing has gained significant attention from the scientific community and media due to the promising results of early studies. Neuromorphic computing, inspired by biology, is driving innovations in robotics, artificial intelligence, and low-power computing, enabling efficient real-world applications.

Tutorials can be found here: https://github.com/GiuliaDAngelo/CTU-EDNeuromorphic/

Exercise 4.1 (Tutorials 1 and 2) Students of the Neuroinformatics course are welcome to complete any tutorials, provided in the Neuromorphic tutorial above. **Only answers for Tutorials 1 and 2 are required.**. Answer the questions in the respective tutorials via the answer questionnaire.

Tutorial 1 (A) - Time Window for Event-Based Data Visualization

This tutorial introduces students to the fundamentals of event-based vision using a time window visualization method. The provided Python script loads Dynamic Vision Sensor (DVS) data using the Bimvee library and displays events in real-time using OpenCV. Events are grouped into fixed-duration time windows, offering insights into how asynchronous vision systems capture motion and dynamic scenes. The task encourages students to understand the properties of event data and experiment with window duration to visualize fast-changing scenes.

1. **Understanding Event Cameras:** How do event cameras differ from traditional frame-based cameras in terms of capturing dynamic scenes, and what advantages do they provide for analyzing fast-moving objects or changes in the environment?

2. **Data Extraction and Processing:** After loading the event data, we extract the x and y coordinates, timestamps, and polarities of the events. How might these different attributes be useful in understanding the behavior of moving objects within a scene, and what additional processing steps could enhance the analysis of this data?

3. **Interactive Visualization:** The script visualizes events in real-time within specified time windows. How might adjusting the window period affect the visualization of events, and what strategies could be employed to ensure that significant events are not missed or overwhelmed by noise during the visualization process?

Tutorial 1 (B) - Sliding Window for Event-Based Data Visualization

This tutorial builds upon the previous one by implementing a sliding window approach. Instead of refreshing the event display at fixed time intervals, the script maintains a rolling window of events, continuously updating the visualization. Students explore the effect of window size and step on real-time event visualization, learning how this approach captures continuous motion more effectively. The task emphasizes strategies for tuning parameters to balance temporal resolution and noise filtering.

1. **Sliding Window Technique:** How does the implementation of a sliding time window impact the way we visualize event data? What considerations should be made regarding the duration of the sliding window to ensure meaningful representations of the scene are captured?

Tutorial 1 (C) - Fixed Event Count for Event-Based Data Visualization

Here, the tutorial introduces a different strategy: visualizing a fixed number of events per frame, regardless of time. Students explore how grouping events by count rather than time affects perception of movement and object tracking. This method allows consistent processing loads and is particularly relevant in neuromorphic computing applications. The task encourages reflection on the tradeoffs between time-based and count-based processing for dynamic scene understanding.

1. **Understanding Event Grouping:** How does the fixed event count visualization approach differ from time-based visualization methods? What are the potential benefits and drawbacks of using a fixed number of events per visualization window in terms of capturing dynamic scenes?
2. **Real-Time Visualization Implications:** What challenges might arise when visualizing events in fixed batches, particularly in a dynamic environment? How could the choice of batch size (e.g., 1000 events) affect the responsiveness and accuracy of the visualization?

Tutorial 2 - Loading IBM DVS Gesture Dataset

This tutorial guides students through loading and visualizing the DVS Gesture dataset. Using Python, event data are converted into image frames based on polarity (ON/OFF events), allowing students to understand how gestures appear in the DVS domain. The task involves experimenting with different user trials and time window settings to analyze gesture features and sensor responses. This exercise helps bridge the gap between raw event streams and higher-level interpretation.

1. Try experimenting with different values for `user_trial` and `time_window`. How do these changes affect the visualization and interpretation of the data?