# Computational Cognitive Neuroscience. Lab 1

Simple Neuron Models. January 2025

**Lecturer:** Peggy Seriès
**Teaching Assistant:** Lars Werne

## Tutorial Objectives

In this tutorial, you will learn to do the following:

- Implement a model of a single neuron in Python (or Matlab)
- Simulate the model numerically and plot the results
- Interpret your findings and their relevance to biology
- Understand where analytical approximations may be used alongside numerical simulation and how the two approaches complement each other

## Introduction

In this tutorial, you will code and simulate a simple neuron model that we discussed at length in Lecture 3: the leaky integrate and fire neuron. You will explore how the spike statistics and membrane potential dynamics of the model depend on the statistics of the input the neuron receives and how analytical approximations can provide insights into this behaviour in limiting situations.

## Getting Started

Before starting, ensure you have Python installed or available on the computer you intend to use:

- If you are in a Computer Lab, Python should already be installed.
- If not, you can install Python following the directions here: https://www.python.org/about/gettingstarted/.
- Matlab users are also welcome to use it for this tutorial. Solutions will primarily be released in Python, but Matlab solutions can be made available upon request.

### Python Knowledge for This Tutorial

You should be familiar with the following concepts in Python/NumPy/Matplotlib (or their Matlab equivalents). If these are not familiar, review the Python tutorial available on the CCN Learn page:

- Arrays (e.g., creating an array of zeroes, vectors, matrices)
- Indexing (e.g., accessing elements of an array, logical indexing, etc.)
- Loops (e.g., `for`, `if-then`, nested loops)

- Plotting (e.g., commands such as `matplotlib.subplot`, `xlim`, `ylim`, etc.)
- Histograms (e.g., `matplotlib.hist`)
- Mathematical operations (e.g., scalar addition, multiplication, `exp`)
- Random variables (e.g., `numpy.random.randn` for normal distribution)
- Differences (e.g., `numpy.diff` for differences between consecutive elements of an array)

## Part 1: Setting up the Model for Passive Dynamics

We first consider how to numerically simulate the passive membrane potential dynamics of a simple neuron model – the leaky integrate and fire neuron.

The membrane potential of the leaky integrate and fire model obeys the following equation while below the spiking threshold:

$$\tau \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - E_m) + \frac{I_{\text{ext}}}{g_m}.$$

Our goal is to solve this equation numerically using the Euler method. This will allow us to explore how the membrane potential evolves over time and how it eventually stabilizes under various conditions. You can review Lecture 3, Slide 22, for a refresher. Follow these steps:

1. **Set up parameters**:
   - Assign values to the parameters: $\tau$, $E_m$, $I_{\text{ext}}$, $g_m$.
   - Define the number of iterations, $N_t = 10000$, and the timestep, $\delta t = 0.1\,\text{ms}$, for a total simulation time of 1 second.
2. **Initialize arrays**:
   - Create an array of size $[N_t + 1, 1]$ to store the values of the membrane potential.[1]
3. **Simulate the dynamics**:
   - Use a `for` loop to compute the values of the membrane potential at each time step.
   - Start the simulation with the initial condition $V = E_m$ and the following parameters:
     - time constant $\tau = 10\,\text{ms}$
     - resting potential $E_m = -70\,\text{mV}$
     - membrane conductance $g_m = 1\,\text{S}$
     - external current $I_{\text{ext}} = 20\,\text{A}$
4. **Plot and analyze results**:
   - Check that your simulation runs without errors.
   - Plot the membrane potential as a function of time.
   - Inspect the plot. Did the simulation behave as expected? How can you tell?

[1] After $N_t$ iterations you will have $N_t + 1$ time points

5. **Explore parameter variations**:

   - Vary the input current, time constant, and initial condition. [2] Observe how each affects the evolution of the membrane potential.

6. **Determine steady-state behavior**: In the simulations above, you should observe that the membrane potential converges to a stable, final value (the "steady-state" or "equilibrium" potential). This value of $V$ is reached when $\frac{\mathrm{d}V}{\mathrm{d}t} = 0$ in the equation:

$$\tau \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - E_m) + \frac{I_{\text{ext}}}{g_m}.$$

   - Derive an analytical equation for the steady-state membrane potential.
   - Compare the derived equation with your simulations. Do the results agree?

## Part 2: Incorporating Spiking Into the Model

Next, we will add spikes to the model and observe how they change its behavior. To model spikes, we introduce the following spike-reset rule:

$$\text{If } V(t) \geq V_{\text{threshold}}, \text{ then set } V(t) \to V_{\text{reset}}.$$

1. **Implement the spike-reset rule**:

   - Modify your code to include the spike-reset rule by adding an `if-then` condition inside the loop over time steps.
   - Ensure that spike times are stored in a separate array for later analysis.

2. **Set parameters for spiking**:

   - Use the following parameter values for spiking behavior:
     - Threshold potential: $V_{\text{threshold}} = -50\,\text{mV}$
     - Reset potential: $V_{\text{reset}} = -75\,\text{mV}$

3. **Run and analyze simulations**:

   - Run the modified simulation with different values of:
     - Input current ($I_{\text{ext}}$)
     - Time constant ($\tau$)
   - Plot the membrane potential for each case.
   - Analyze how changes in these parameters influence spiking activity. What patterns do you observe?

[2] It may help if you modify the code to allow you to simulate the model under multiple parameter values in one go. One way to do this is to write a for loop around your simulation that iterates over parameters (although for loops are inefficient in Python - if you convert the set of equations into a vector equation it will run faster). You can create a membrane potential array of size $[N_t + 1, N_{parameters}]$ and store all the simulations in this array.

## Part 3: Adding Noise to the Input

In this part, we will explore how the neuron behaves when driven by noisy, fluctuating input. Initially, we return to the case of purely passive dynamics, as in Part 1. To simulate this, you can modify your spiking code by setting $V_{\text{threshold}} = \infty$, which ensures the neuron never spikes.

The noisy input is modeled as:

$$I_{\text{ext}}(t) = I_0 + \sigma\xi(t),$$

where $\xi(t)$ is a normally distributed random variable with zero mean and unit variance, drawn independently at each timestep.[3]

1. **Simulate passive dynamics with noisy input**:
   - Set $\sigma = 50$ and simulate with different values of $I_0$.
   - Plot the membrane potential for each input current value.
   - Analyze the plots. How would you expect the noise to influence spiking activity?
2. **Incorporate the spike-reset rule and analyze interspike interval distributions**:
   - Reintroduce the spike-reset rule by setting $V_{\text{threshold}} = -50\,\text{mV}$.
   - Vary both $I_0$ and $\sigma$. Compute interspike interval (ISI) distributions for different parameter combinations.
   - Use `matplotlib.hist` to create histograms of the ISI distributions for various parameter values.
   - Discuss how the shape of the ISI distribution depends on the input parameters. If necessary, increase the number of simulation timesteps to obtain higher sample sizes for better analysis.

[3] In Python, use `np.random.randn` to generate the noise term $\eta$ independently at each simulation time step. When simulating Gaussian white noise numerically with discrete time steps $\delta t$, one has to scale the noise variance by $1/\delta t$. If you do not do this, the noise strength will change as you change $\delta t$. This is a property of approximating a continuous-time noise process with a discrete-time simulation. To correct for this, the Euler update for this Gaussian white noise model should be implemented as:

$$V_{t+1} = V_t + \frac{\delta t}{\tau}\left(-(V_t - E_m) + I_0 + \frac{\sigma}{\sqrt{\delta t}}\eta_t\right),$$

where $\eta_t \sim \mathcal{N}(0, 1)$.