AMS 332/BIO 332/NEU 536
– Project for Module 5 –

*** Remember to include: 1. your code (your `*.m` files) and 2. a statement of collaboration. ***

Part A: Poisson spike trains and basic measures of neural activity (25/50 points)

1. (Spike trains) Generate $N = 50$ Poisson spike trains with rate $\lambda = 10$ spikes/s, each lasting $T = 5$ seconds. These will represent a fictitious dataset comprising recordings from the same neuron in $N = 50$ trials of a fictitious experiment (one spike train for each trial). To generate the spike trains, generate first the **inter-spike intervals** (ISIs), and then obtain the spike times by *cumulating* the ISIs.
   Here is the outline of the procedure:
   - Use the fact that the ISIs of a Poisson spike train are exponentially distributed with parameter λ, i.e., `ISI=-log(u)/`λ, where `log()` denotes the *natural* logarithm in Matlab, and the random variable '`u`' follows a uniform distribution between zero and one. The instruction `u=rand(n,1)` will generate n such random deviates. For the actual number of spikes to be generated, see **Box 1.**
   - The ISIs are the differences between successive spike times, $\{(t_2-t_1),(t_3-t_2),\ldots,(t_n-t_{n-1})\}$, where $t_1, t_2, \ldots, t_n$ are the spike times. Obtain the vector of spike times, $\mathbf{t} \equiv \{t_1, t_2, \ldots, t_n\}$, by cumulating the ISIs: `t=cumsum(ISI)`.
   - Repeat the above procedure $N = 50$ times, once for each spike train.

Box 1: how many spikes?

How many spikes should you generate for each spike train? If you generate $\lambda \cdot T = 50$ ISIs for each spike train, their cumulative sum will sometimes exceeds $T = 5$ s and sometime fall short of it – for example, the latest spike time may occur at $t = 4.23$s, and perhaps the next spike after that would have occurred at a time $t = 4.76 < T = 5$ s and you will have missed it! This happens because the Poisson process is a stochastic process and thus sometimes you may get e.g. 47 spikes in 5 seconds, and sometimes you may get 54 spikes, and so on. How can you generate the right amount of spikes in each trial to cover a $T = 5$ s interval? One way would be to generate ISIs until their cumulative sum exceeds T, discard the very last spike, and stop there. Another way is to generate more spikes than needed, for example $2\lambda T = 100$ ISIs *in each trial*, and then remove all those spike times exceeding $T = 5$s. Recall that within 2 standard deviations, one expects around $\lambda T + 2\sqrt{\lambda T} \approx 64$ spikes within $T = 5$ s, hence $2\lambda T = 100$ should be plenty.

If you want to store all your spike trains as columns of the same matrix – say, a matrix $S_{100\times N}$ which contains N columns and 100 *spike times* in each column –, you can replace the spike times exceeding T with `NaN`, which is a Matlab constant standing for '`not-a-number`': given the matrix S that contains *all* spike times, this is accomplished by

```
>> ind=find(S>T);
>> S(ind)=NaN;
```

These two lines of code can be combined into a single line:

```
>> S(S>T)=NaN;
>> S % have a look at S
```

Once this is done, you can visualize all N spike trains in a **raster plot** (see the next exercise):

```
>> plot(S,1:N,'.k'); % raster plot
```

Additional help on the generation of Poisson spike trains can be found in the lecture notes on the stochastic theory of reaction kinetics.

Matlab tip ⇒ Recall that you can always get information about Matlab functions by typing '`help function_name`' from the command line; for example, '`help rand`' will provide information on function `rand`. Other Matlab concepts and functions that are very useful for this project are: `find, length, NaN, isnan`, and ∼. The symbol ∼ ('not') negates the outcome. For example, once you work out what `isnan(X)` does to a vector `X`, test what ∼`isnan(X)` does. □

2. (Raster plot) Perform a **raster plot** of these N spike trains. This is a plot showing each spike train on a different horizontal line (from 1 to N), with the spike times on each line as a sequence of dots, each line stacked above the previous one (see **Box 1** for help). Label the horizontal axis as 'time' (from zero to T sec) and the vertical axis as 'trial number' (from 1 to N). **Include the raster plot in your report.**

3. (Firing rate) Calculate the **average firing rate** across trials for the whole simulation interval (T sec) as

$$f = \frac{\text{number of total spikes across trials}}{N \cdot T}. \tag{1}$$

Confirm that your estimated firing rate f is very close to the theoretical firing rate λ.

4. (PSTH) Next, divide the total time T in $N_b = T/\Delta t$ bins, each of length $\Delta t = 0.2$ s. Calculate the firing rate in each bin, i.e., in the first time bin $T_1 = [0, 0.2]$, in the second time bin $T_2 = [0.2, 0.4]$, and so on, until the firing rate in the last time bin $T_{25} = [4.8, 5.0]$ has also been determined. This will give you the set of firing rates

$$f(T_i) = \frac{\text{number of total spikes in } T_i \text{ across trials}}{N \cdot \Delta t}. \tag{2}$$

Plot $f(T_i)$ as a function of the middle points t_i^* of time bins T_i (for example, the middle point of time bin $T_1 = [0, 0.2]$ is $t_1^* = \frac{1}{2}(0+0.2) = 0.1$ s). Such a plot is called peri-stimulus time histogram (PSTH). **Include the PSTH plot in your report**.

5. (CV) Compute the **coefficient of variability** (CV) of the inter-spike intervals (ISIs) in each trial. The CV quantifies the variability of the ISIs in each trial and is defined as the ratio of the standard deviation to the mean of the ISIs: in each trial $k = 1, \ldots, N$,

$$CV_k = \frac{\text{StDev(ISI)}_k}{\text{Mean(ISI)}_k}. \tag{3}$$

Plot CV_k as a function of trial number k and **include the plot in your report**.
Next, average CV_k across trials,

$$\langle CV \rangle = N^{-1} \sum_{k=1}^{N} CV_k, \tag{4}$$

and check that $\langle CV \rangle \approx 1$, the theoretical CV of Poisson spike trains.

Matlab tip $\Rightarrow$ Given the vector of spike times $\mathbf{t} = \{t_1, t_2, \ldots, t_n\}$, the ISIs can be conveniently obtained as `ISI=diff(t)`. Mean, standard deviation and variance of a vector `x` in Matlab can be obtained as `mean(x)`, `std(x)`, and `var(x)`, respectively. □

6. (Fano factor) Quantify the **trial-to-trial variability** of this dataset by computing the so-called **Fano Factor** (FF). The FF is defined as the ratio of the variance to the mean (across trials) of the spike count:

$$FF = \frac{\text{Var}_{trials}(\text{count})}{\text{Mean}_{trials}(\text{count})}. \tag{5}$$

Theoretically, for Poisson spike trains FF=1. Is that confirmed (at least approximately) in your data?

7. Repeat Exercises 5. and 6. for increasing interval durations T (in seconds). Use

$$T = \{10, 100, 200, 400, 800, 1600, 3200, 6400\}. \tag{6}$$

For each value of T, take the CV of a single spike train (for example, the first) and the FF of the set of 50 spike trains. Plot both the CV and the FF values as a function of T in the same plot and **include the plot in your report** (note: do NOT plot the spike rasters!). Does the CV visibly converge to 1 as T increases? Does the FF? (You may want to run your code a few times before committing to a conclusion).

Part B: Perceptron learning (25/50 points)

Build a perceptron with N input units x_j, $j = 1, \ldots, N$, and one output unit

$$y = \text{sgn}(\mathbf{w} \cdot \mathbf{x}), \tag{7}$$

(use Matlab function `sign()`), where $\mathbf{w}$ is the vector of synaptic weights, $\mathbf{x}$ is the vector of input units $x_j = \pm 1$, and $\mathbf{w} \cdot \mathbf{x} \doteq \sum_{j=1}^{N} w_j x_j$ (both $\mathbf{w}$ and $\mathbf{x}$ are vectors with N elements).

Initialize all synaptic values to zero and recall that the (component-wise) learning rule is

$$\Delta w_j = \eta(y^t - y)x_j, \tag{8}$$

where y^t is the desired value in response to $\mathbf{x}$, y is the actual output, η is the learning rate, and the weights are updated after each pattern presentation. In all exercises below, use a learning rate of $\eta = 1$.
Make sure to present all patterns multiple times and in random order; by default, stop the algorithm after $T = 500$ pattern presentations.

1. (Boolean functions) Start with $N = 2$ and try to solve the AND, OR and XOR problems,[1] where

   - y^t=AND$(x_1, x_2) = 1$ if and only if both $x_1 = 1$ and $x_2 = 1$, otherwise $y^t = -1$.
   - y^t=OR$(x_1, x_2) = 1$ if either $x_1 = 1$ or $x_2 = 1$, otherwise $y^t = -1$.
   - y^t=XOR$(x_1, x_2) = 1$ if either $\{x_1 = -1, x_2 = 1\}$, or $\{x_1 = 1, x_2 = -1\}$, otherwise $y^t = -1$.

   Make sure to use a 3D representation of the inputs, i.e., express each input pattern as $\{x_1, x_2, x_3\}$, where x_3 is always set to -1.[2]
   Answer the following questions:

   (i) How many steps does it take, approximately, to converge to the solution for the AND and the OR problems?

   (ii) What happens in the case of the XOR problem? Does the algorithm converge to perfect performance (i.e., correct classification of *all* input patterns)? If not, what kind of behavior do you observe? Does it help to increase the number of pattern presentations to 1000 or more?

   To answer these questions, plot the performance vs. presentation number (see **Box 2**).
   Attach a few plots to your report to accompany your explanations.

> BOX 2: PLOTTING PERFORMANCE VS. PRESENTATION NUMBER
>
> Define performance as +1 for a correct classification and 0 for a misclassification after each pattern presentation, and then plot the performance vs. presentation number using the `'.'` option in function `plot()`.
>
> *Matlab tip* ⇒ Make your plot more readable with `ylim([-0.2 1.2])`. □

2. (Random patterns) Generate $M = 40$ random vectors, each with $N = 50$ elements taking ± 1 values according to a uniform distribution (see **Box 3**). Assign $M/2$ patterns randomly to class '$y^t = +1$' and the remaining to class '$y^t = -1$'. Then let the perceptron try to learn them.

   (i) What do you observe? Does the perceptron learn to separate them in the desired classes after 100 steps? After 1000 steps? After 5000 steps?

   (ii) What conclusions do you derive on the linear separability (or lack thereof) of your random vectors?

   (iii) Do your results change if you use a different learning rate, e.g., $\eta = 0.1$ or $\eta = 10$?

   (iv) If the algorithm converges to perfect performance, report the *average* number of pattern presentations until convergence (n_{conv}, see **Box 4**), where the average is across multiple runs: $\langle n_{conv} \rangle_{runs}$. This means

[1] Note: these are three separate, independent problems.
[2] This corresponds to expressing condition $\mathtt{w_1 x_1 + w_2 x_2} \geq \theta$ as $\mathtt{w_1 x_1 + w_2 x_2 + w_3 x_3} \geq 0$, where $\mathtt{x_3} = -1$ (for all input patterns) and $\mathtt{w_3} \equiv \theta$ (parameter to be tuned by the learning algorithm). See the lecture notes for details.

that you will have to repeat the exercise several times (='runs') to obtain $\langle n_{conv} \rangle_{runs}$. Use a new group of M patterns in each run; use at least 10 runs and 1000 presentations in each run. If any, report the number of runs for which the criterion for convergence defined in Box 4 was *not* reached.

Attach a few plots of the performance vs. presentation number to accompany your explanations as done in exercise 1.

(v) What happens if you repeat (iv) with $M = 90$? To allow the comparison, use $100 \times M$ presentations for $M = 40$ and $1,000 \times M$ presentations for $M = 90$, and report the mean number of presentations *for each pattern*, i.e., $\langle n_{conv} \rangle_{runs}/M$.

Box 3: generation of random vectors

One way to do this is to use Matlab function `randi()`: `X=randi(2,N,M)-1;` This will create an $N \times M$ matrix with M random vectors of N '0/1' elements, where each column is a random vector. Then you can easily map these '0/1' values into ± 1 values (but the results of the exercise should not depend on whether you use the '0/1' or the '± 1' representation for the input patterns – as long as you keep the ± 1 representation for the targets y^t, consistent with Eq. 7).

Box 4: criterion for convergence

n_{conv} is the number of pattern presentations until the performance becomes correct (+1) for *all* subsequent pattern presentations. We can stop after a sequence of presentations containing all patterns at least once, all of which have been correctly classified. However, for practical purposes, we will **define n_{conv} as the first of 200 consecutive correct classifications**.