Homework 4: Poisson neuron models

In this homework, you will absorb several key concepts of neural variability ("noise") and the code ("signal") using the Poisson neuron model and the linear-nonlinear-Poisson (LNP) model. You will use a pseudorandom number generator to draw samples from a Poisson distribution. You will also learn a couple of new concepts along the way. Most of the code needed is provided in this document with just a few missing details marked with [[MODIFY THIS]]. You will also need to download 4 '.mat' data files from brightspace. Note that not all code needs modification. Also, you do not need to fully understand the plotting code.

Carefully follow the instructions in order. The instructions are designed to build understanding required for the subsequent steps. Submit the answers inserted into this document and converted to a PDF.

Due: **Oct 8, 2022, Saturday, 11:59 PM ET via Brightspace**

Name: **Amrita Shah**

Theoretically, the mean and the variance of a Poisson distribution are equal, making the Fano factor 1. Empirically, given a collection of observations, one can estimate the Fano factor by plugging in the sample mean and sample variance. **Insert the appropriate expressions in the provided MATLAB code to estimate the Fano factor. Draw 10 pseudorandom samples from a Poisson neuron model with $\lambda = 5.5$ and T = 1. Run the code segment a few times without fixing the random seed and observe the fluctuation of the estimated Fano factor. Increase the number of samples (trials) to 1000 and repeat.** (Use the skeleton code below)

```
%rng(2478); % set a fixed random seed [[you may uncomment this and modify]]
T = 1;          % total duration in seconds
lambda = 5;     % (spikes/sec) mean firing rate [[Q1a: MODIFY THIS]]
nTrial = 1000;    % number of samples to draw [[Q1a: MODIFY THIS]]

Lambda = lambda * T; % parameter for the Poisson distribution
y = poissrnd(Lambda, nTrial, 1); % draw nTrial number of pseudo-random
Poisson samples

mu = mean(y); % estimate sample mean
s2 = var(y);  % estimate sample variance
FF = (s2/mu); % estimate sample Fano factor [[Q1a: MODIFY THIS]]

fprintf('mean [%.3f] variance [%.3f] Fano factor [%.3f]\n', mu, s2, FF);
```

Q1a. [2 pt] Describe the variation you observe in the Fano factor. For which number of samples does the Fano factor deviate most from the theoretical value? (1-2 sentences)

The Fano Factor deviates greatly from the theoretical value of 1 when 10 samples are drawn, with values from 0.065 to 2.647 in just a few runs. There is little fluctuation and deviation from 1 when 1000 samples are drawn.

The sum of two independent Poisson random variables is again Poisson distributed with the sum of corresponding mean parameters. Therefore, this process of summation of Poisson random variables with a small rate can be repeated to produce a Poisson distribution with a large mean parameter.

For example, if there are 1000 independent Poisson random variables all with an identical mean parameter $\Lambda_s = 0.01$, then drawing a sample from each Poisson distribution and adding them together would be equivalent to drawing a single sample from a Poisson distribution with $\Lambda = 10$.

Q1b. [2 pt] Consider a small time window size T = 0.001 sec (1 ms), and a mean firing rate of $\lambda = 10$ spikes/sec. The spike count of a Poisson neuron with this rate will follow a Poisson distribution with y_i = Poisson(Λ) where $\Lambda = \lambda \cdot T = 0.01$. **Calculate the probabilities of having (a) 0 spikes, (b) 1 spike, or (c) 2 or more spikes? (show the derivation for each case)** [Hint: if it's not 0 or 1, then it's more than 2; use the law of total probability (the sum of all probabilities must equal 1.)]

$$\Pr[\mathrm{Y} = \mathrm{k}] = \frac{\Lambda^k e^{-\Lambda}}{k!}$$
$$\Pr[\mathrm{Y} = 0] = \frac{0.01^0 e^{-0.01}}{0!} = 0.99005$$
$$\Pr[\mathrm{Y} = 1] = \frac{0.01^1 e^{-0.01}}{1!} = 0.00990$$
$$\Pr[\mathrm{Y} \geq 2] = 1 - 0.99005 - 0.00990 = 4.9668 * 10^{-5}$$

The Poisson neuron model only gives you spike counts, and not the pattern of spikes within the interval T. Let's try to fix this. You probably noticed that the probability of getting more than 2 spikes is pretty small. Using the summation property, this provides a way to simulate spike trains from the Poisson neuron model – break a large $\Lambda = \lambda T$ on a large interval T into many smaller intervals with small $\Lambda_s$such that the probability of getting 2 or more spikes is vanishingly small. Once this is achieved, the Poisson random draws will be mostly 0, and sometimes 1, mimicking a spike train generation! This limit of dividing a Poisson distribution leads to a random process called the *Poisson process*. The Poisson process is used in neuroscience as a model of spike train statistics.

Let's verify that this process works by breaking down the $\lambda = 5.5$ spikes/sec and T = 1 second process into 100 bins of duration 10 ms. Modify and run the code segment below (it continues from the code from Q1a). Note that y is a (nTrial x nBin) matrix of independent random samples, and sum(y,2) takes the summation over the 2nd dimension, i.e., it sums over the nBins, leaving nTrial number of outputs.

```
%% divide the Poisson distribution for duration T into smaller bins
% to verify the sum property, and understand the **Poisson process**
dt = 0.01;          % bin size in seconds [[Q1c: MODIFY THIS]]
nBin = ceil(T/dt); % number of bins within T, rounded up

y = poissrnd(lambda * dt, nTrial, nBin);
y2 = sum(y, 2); % add the Poisson samples for smaller time bins together

mu = mean(y2);
s2 = var(y2);
FF = (s2/mu);  % estimate sample Fano factor [[Q1b: MODIFY THIS]]

fprintf('mean [%.3f] variance [%.3f] Fano factor [%.3f]\n', mu, s2, FF);

% optional plotting of the spike trains
fig = figure(4000); clf;
imagesc(y < 1); axis xy; colormap('gray'); set(gca, 'TickDir', 'out');
xlabel('time (ms)');
ylabel('trials');
ylim(0.5 + [0, min(nTrial,50)]); % just the first 50 trials
% note that the spikes may not show up correctly if your figure window is
too small. Try zooming in if you suspect there are more spikes.
```

Q1c. [2 pt] This approach to emulating the Poisson process consists of many independent Poisson distributions with small mean. **Try dt = 100 ms, 10 ms, 1ms, 0.1 ms. Which window sizes of the time bins makes "the probability of not spiking" in each bin is greater than 99%.** You can do this either empirically through trial and error, or use the Poisson probability equation (or MATLAB function `poisspdf`).

$\Lambda = \lambda * \mathrm{dt}\ (\text{in seconds})$ $\quad\quad$ $\Pr[\mathrm{Y}=0] = \frac{\Lambda^0 e^{-\Lambda}}{0!} = e^{-\Lambda}$

$\Lambda = 5.5 * 0.1 = 0.55$ $\quad\quad$ $\Pr[\mathrm{Y}=0] = e^{-0.55} = 0.5769$

$\Lambda = 5.5 * 0.01 = 0.055$ $\quad\quad$ $\Pr[\mathrm{Y}=0] = e^{-0.055} = 0.9465$

$\Lambda = 5.5 * 0.001 = 0.0055$ $\quad\quad$ $\Pr[\mathrm{Y}=0] = e^{-0.0055} = 0.9945$

$\Lambda = 5.5 * 0.0001 = 0.00055$ $\quad\quad$ $\Pr[\mathrm{Y}=0] = e^{-0.00055} = 0.9994$

The 1ms and 0.1ms window sizes of the time bins make the probability of not spiking greater than 99%. The probability of getting 0 spikes is 99.45% if the time window is 1ms, and 99.94% if the time window is 0.1ms.

The time interval between consecutive action potentials is called the *inter-spike interval (ISI)*. In electrophysiological recordings of neurons, neuroscientists often characterize a neuron's firing pattern with the distribution of ISIs. Since the Poisson process approximation allows generating spike timings corresponding to the Poisson neuron, let's plot the ISI distribution. **Run the following segment of code to collect the ISI distribution and plot it (It continues from the code in Q1c).**

```
%% ISI distribution (Q1d)
% Collect the inter-spike intervals from each trial
isi = [];
sts = cell(nTrial, 1);
for kTrial = 1:nTrial
    st = find(y(kTrial, :)); % find the index of all the non-zero bins
    st = st * dt; % convert array index to spike times
    isi = [isi, diff(st)]; % take the difference between the times and
collect them
    sts{kTrial} = st;
end

% generate a plot of ISI distribution
fig = figure(591); clf;
histogram(isi * 1000, ceil(numel(isi)/15))
title('inter-spike interval distribution');
xlabel('ISI (ms)');
ylabel('occurrence');
```

Q1d. [3 pt] What is the most likely ISI (in the unit of ms)? [Note that the histogram can be variable due to insufficient number of spikes or trials. Simulate the Poisson process a few times to get a feel of the underlying ISI distribution shape which is smooth.] How would your answer change if the time bin size became smaller? (1-2 sentence)

The most likely ISI value is within the interval of 0 to 3.046 ms. As the bin size becomes smaller, the underlying smooth curve of the ISI distribution remains the same. The ISI is most likely to be within the first bin, and closest to zero.

Q1e. [2 pt] Discuss how the Poisson process deviates from the biological neuron regarding the minimum ISI (1-3 sentences). [Hint: Recall that a real neuron has an absolute refractory period.]

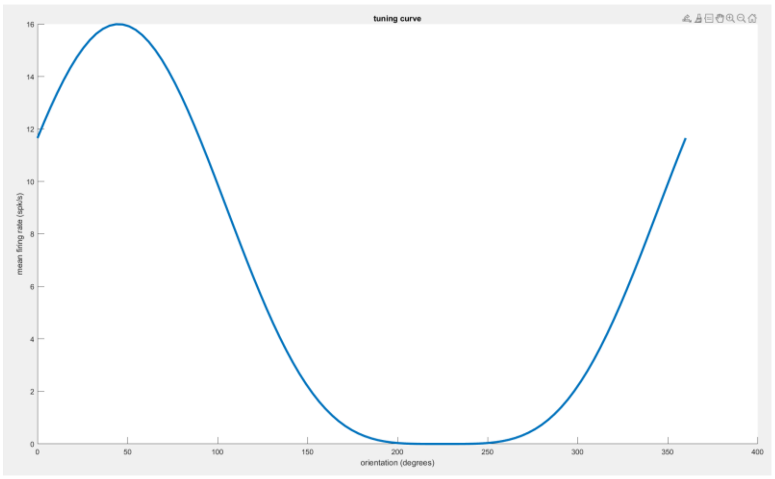
The Poisson process results in ISI values of 0 ms. If there is no inter-spike interval, this means two spikes occurred simultaneously. In the biological neuron, there is an absolute refractory period which makes it impossible for two spikes from one neuron to occur at the same time, so the ISI value can never be zero.

Q2 is about estimating the tuning curve from data generated from a Poisson neuron model.

In MATLAB, you can use *an anonymous function* to define quick one liner functions. This is perfect for defining tuning curves in this exercise. The syntax for anonymous function starts with `@(x)` followed by the mathematical expression that uses x. For example, a function that adds 1 to the current value can be written as `f = @(x) x + 1;`. Once defined, you can call the function as usual, e.g., `f(4)`. Here's an example run in the MATLAB command window:

```
>> fun = @(x) sqrt(x + 347)

fun =
  function_handle with value:
    @(x)sqrt(x+347)

>> fun(3)
ans =
  18.708286933869708
```

Q2a. [1 pt] Write a MATLAB function to implement the tuning curve $\lambda = f(x) = 4(cos\,(x - 45) + 1)^2$ within the code provided below. Insert the generated plot as the answer. [Hint: use MATLAB function `cosd` for cosine in degrees and `.^` for element-wise power]



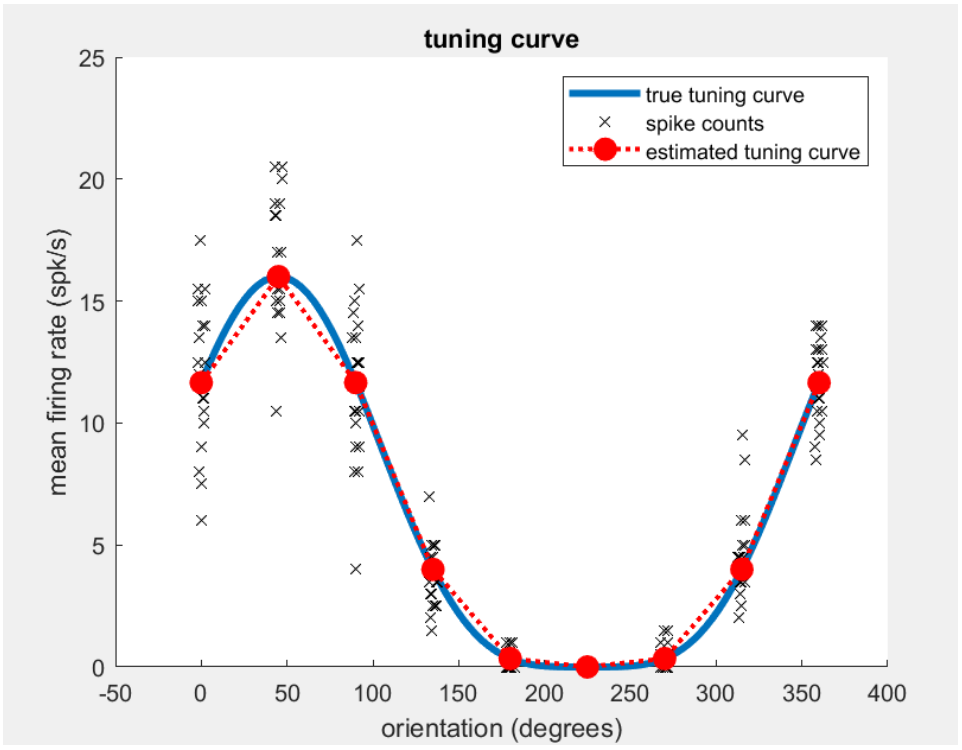
```
%% setup the true tuning curve
% the commented examples below are for fun:
f = @(x) 5*(sind((x-11))/2 + 0.5).^2 + 0.6;  % a broad tuning curve
%f = @(x) 4*(cosd((x-150))/2 + 0.5).^10 + 0.6; % a narrow tuning curve
%f = @(x) exp(2 * cosd(x + 30).^2 - 0.5); % exponentiated exp-cos-square
%f = @(x) 2 * (x/60).^2 + 1; % quadratic
%f = @(x) 4*((cosd(x-45) + 1).^2); % [[Q2a: UNCOMMENT and MODIFY THIS]]

xr = linspace(0, 360); % generate uniform grid from 0 to 360 degrees

fig = figure(3471); clf; hold all;
ph1 = plot(xr, f(xr), '-', 'LineWidth', 3);
ylabel('mean firing rate (spk/s)');
xlabel('orientation (degrees)');
```

```
title('tuning curve');
```

A tuning curve is the mean firing rate (in units of spikes per second) as a function of stimulus input. For the model neuron from Q2a, we consider 0 to 360 degrees as our stimulus range. Since we can't stimulate every possible angle, we'll choose multiples of 45 degrees, and estimate coarsely the tuning curve for those stimuli only. The code below will generate spike counts corresponding to the stimuli for repeated trials according to a Poisson neuron model with tuning curve you defined in Q2a.

```
%% Simulate spike counts and estimate tuning curve
stimuli = [0, 45, 90, 135, 180, 225, 270, 315, 360]; % equiv to 0:45:360
T = 2;        % [[Q2c: MODIFY THIS]]
nTrial = 20; % [[Q2c: MODIFY THIS]]

nStim = numel(stimuli); % number of stimuli
meanSpikeCount = zeros(nStim, 1); % prepare to save the mean spike counts
fig = figure(3471); % plot on the same figure as tuning curve

for kStim = 1:nStim % for each stimuli
    stim = stimuli(kStim);

    % simulate the Poisson neuron model
    lambda = f(stim);
    y = poissrnd(T * lambda, nTrial, 1);

    % estimate the mean spike count, pretending we don't know f
    meanSpikeCount(kStim) = lambda*T/2;  %% [[Q2b: MODIFY THIS]]

    % plot individual trials (use random jitter to enhance visualization)
    ph2 = plot(stim + 5 * (rand(nTrial, 1) - 0.5), y / T, 'xk');
end

% plot the estimated curve
ph3 = plot(stimuli, meanSpikeCount, 'o:r', 'MarkerFaceColor', 'r',
'MarkerSize', 8, 'LineWidth', 2);
legend([ph1, ph2, ph3], 'true tuning curve', 'spike counts', 'estimated
tuning curve');
```

Q2b. [3 pt] Insert the correct piece of code above to estimate the tuning curve from spike counts generated by the Poisson neuron. (Units must be spikes per second) Insert the generated plot as the answer.



Q2c. [extra credit] Vary the duration and/or the number of trials and observe the quality of the estimated tuning curve change. Describe and explain your observations. Discuss the possible implications in experimental neuroscience (short paragraph).

Altering the duration T by a certain factor alters the mean firing rate on the estimated curve by that factor. When T is doubled from 2 to 4, the mean firing rate at 45 degrees is doubled from 16 to 32. When T is halved from 2 to 0.5, the mean firing rate is halved from 16 to 8. The longer duration allows the neuron more time to receive signal and fire more spikes, while the shorter duration has the opposite effect. In experimental neuroscience, it is very important to derive the optimal values for experimental parameters, as they can greatly alter the results.

Let's use the linear-nonlinear cascade applied to the receptive field model of V1 we used in HW2. Load the 25 x 25 receptive field (RF) from the "rf.mat" file. We will use $g(x) = e^{x/2-0.1}$as our nonlinearity. Also provided are various test stimuli along different parameters in the 25 x 25 image space. Run the following code which will produce 3 tuning curves corresponding to 3 different 1-dimensional parameters that control the stimulus. The tuning curves are evaluated for the corresponding stimuli input plotted below the tuning curve. The x-label of the tuning curve provides a hint as to what is being changed. Note that most of the lines are for plotting.

```
% Load the receptive field
load('rf');

% setup the linear-nonlinear cascade
g = @(z) exp(z/2 - 0.1); % nonlinearity
f = @(x) g(sum(rf .* x, 'all'));

xFiles = {'x_theta', 'x_phase', 'x_sf'}; % mat file names with the stimuli

for kFile = 1:numel(xFiles)
    load(xFiles{kFile});     % load the stimulus images (stored in variable
'x')
    nX = numel(x);           % how many stimuli?
    lambda = zeros(nX, 1);   % prepare to store the mean firing rates

    fig = figure(5470 + kFile); clf
    subplot(2, nX+1, nX + 2);
    imagesc(rf); colormap('gray'); axis square; title('RF');

    for k = 1:nX % for each stimulus image
        subplot(2, nX+1, nX + k + 2);
        imagesc(x{k}); colormap('gray'); axis square % plot the stimulus
        title(paramRange(k));

        lambda(k) = f(x{k});   % <--- Key line: evaluate the tuning curve
    end

    subplot(2, nX+1, 2:(nX+1));
    plot(paramRange, lambda, 'o-');
    axis tight; xlabel(paramStr); ylabel('firing rate'); title('tuning
curve');
end
```

Q3a. [3 pts] If you didn't know the RF, but only had access to the provided collection of stimuli and the corresponding tuning curves, would it have been enough to characterize this feature that this neuron is sensitive to? Argue on both sides if possible. (3-5 sentences)

The provided collection of stimuli and corresponding tuning curves allow one to characterize the optimal angle of the edge this neuron detects, optimal spatial frequency, and optimal phase shift. Using the peaks of these tuning curves where the firing rate is at its highest, it is possible to construct an example image of what this neuron is sensitive to which includes the optimal values of all the parameters tested above. The tuning curves allow one to distinguish the optimal point of each stimulus feature. However, some points on the tuning curve have very similar firing rates for very different values on the x-axis, so one disadvantage may be that two stimuli could cause identical firing rates in the neuron. Most importantly, three tuning curves are most likely not enough to discern all of the features the neuron is sensitive to, and more parameters would need to be tested.

Q3b. [2 pts] The set of all possible black-and-white stimuli are enormous $2^{25 \cdot 25} \approx 1.4 \times 10^{188}$. The 3 stimuli sets are only a small portion. However, often the LNP model with the correct RF and nonlinear function can predict well the response to all possible stimuli. **Explain the advantage of using RF estimation techniques such as the spike-triggered-average (STA) for neuroscientists compared to characterizing a neuron's tuning curves to many different possible collections of stimuli parametrized as in Q3a.**

Using RF estimation techniques such as spike-triggered-average (STA) is more advantageous to neuroscientists, because this technique involves estimating the receptive field from only stimulus images which induce spikes. This also offers information about various features of the receptive field at once. The latter method involves finding the optimal states of individual parameters by running through very large amounts of stimuli images, of which only a small amount will offer information about the neuron's receptive field. In the above example, 27 stimulus images offered information about 3 features, based on the neuron having a high firing rate in response to about 4 of the images. On the other hand, averaging 27 pixel images that caused the neuron to fire would have created a more reliable receptive field in less time and using less energy.

Q3c. [extra credit] Simulate from the LNP with the LN part given in Q3a for 10000 trials on random stimuli and spike counts generated with the code below. Estimate the receptive field using STA and compare with the true receptive field. Paste your code and comparison figure below.

```
[nx, ny] = size(rf);
nTrial = 10000;
s = randn(nx, ny, nTrial);           % random gaussian noise stimulus
sv = reshape(s, [nx * ny, nTrial]); % vectorize stimulus for speed

%% simulate LNP
l = exp(rf(:)' * sv - 1.3);  % firing rate corresponding to each stimulus
y = poissrnd(l);             % Poisson spike count generation
```

```
figure(1); clf; colormap('gray');
staSaved = cell(nNeuron, 1);

for kNeuron = 1:nNeuron
    sta = zeros(nx, ny);
    y = neuralResponses(kNeuron, :);

    for t = 1:nT
```

```
        frame = rf(:, t);
        sta = sta + frame * y(t);
    end

    sta = sta / sum(y, 'all');

    subplot(5,2,kNeuron);
    imagesc(sta);
    axis square
    caxis([-1, 1] * max(abs(sta(:))));
    colorbar;
    title(kNeuron)

    staSaved{kNeuron} = sta;
end
```