Data-Driven Neural Dynamics Problem set

Instructions:

- Please turn in a Python or MATLAB script that solves each problem/sub-problem in a separate code block. Include comments designating:
  - What problem is being solved in each block
  - What the approach is for that problem (e.g., what is being computed in significant computation blocks/lines of code)
- To navigate the materials for MATLAB (for Problem 3 especially) please start with the "NOTES.pdf" document and "Some key lines in MATLAB format.m". You do not have to start from scratch!

1. **Problem 1: Importance of likelihood selection.** Consider an experimental setup where we can present an N-dimensional stimulus $\boldsymbol{x} \in \mathbb{R}^N$ that elicits a response in a recorded neural population. For one of these neurons, assume that we measure a response $r(\boldsymbol{x})$ for a neuron with a tuning curve $\boldsymbol{g}$ that is defined as the number of spikes in 1 s after the presentation of the stimulus.

   A common model for the response distribution of this neuron is the Poisson model where the response r is distributed as

   $$P(r|\boldsymbol{x}) = \text{Poisson}\left(e^{\langle \boldsymbol{g}, \boldsymbol{x} \rangle}\right),$$

   where $\langle \boldsymbol{g}, \boldsymbol{x} \rangle$ denotes the inner product between two vectors.

   - **Part A** Write MATLAB code that generates samples from this distribution. As a baseline, consider the tuning curve given by

     ```
     >> g = gausswin(N,5).*(cos(2*pi*(0:(N-1))/10).');
     ```

     and sample stimuli given by

     ```
     >> X = 2*rand(N,1);
     ```

     Plot the distribution of r for a single draw of $\boldsymbol{x}$ as a histogram (i.e., generate a single random vector $\boldsymbol{x}$ and use this to draw multiple samples of $r|\boldsymbol{x}$ to generate the histogram). Try the same with drawing different $\boldsymbol{x}$ repeatedly.

     Hint: Recall the properties of a Poisson distribution: the mean and the variance are the same (λ), and this value λ is used to calculate the values that comprise the probability mass function (PMF): $P(r = k) = \frac{\lambda^k e^{-\lambda}}{k!}$.
   - **Part B** Now generate a single response draw for each of M different stimuli $\boldsymbol{x}_m$, $m = 1, ..., M$, where M should be a parameter in your code that is easily adjustable. The response vector should now be an M dimensional vector $\boldsymbol{r} \in \mathbb{R}^M$ with a corresponding matrix $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_M]$ containing all of your generated stimuli. Often in higher dimensional settings we linearize models, in this

case assuming that $\boldsymbol{r} \approx \boldsymbol{X}\boldsymbol{g}$ with added independent, identically distributed (*i.i.d.*) Gaussian noise. Set up a probabilistic relationship (likelihood) of $\boldsymbol{r}$ conditioned on $\boldsymbol{g}$ under a Gaussian noise assumption. Specifically, assume that $\boldsymbol{r} = \boldsymbol{X}\boldsymbol{g} + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ is a mean-zero, Gaussian noise vector with variance $\sigma^2\boldsymbol{I}$. Use the likelihood derived to set up a maximum likelihood inference of $\boldsymbol{g}$ given the responses and stimuli (hint: think least-squares). How close is the estimate of $\boldsymbol{g}$ to the real $\boldsymbol{g}$?

- **Part C** Now use the actual known likelihood (in this case) to estimate $\boldsymbol{g}$ by setting up an optimization program using likelihood defined above. You should focus on defining the minimization program and then using the `fminunc` function in MATLAB.

  Infer the tuning curve using $M = N$ samples. How accurate are the curves? What about for $M = 2N$? $M = N/2$? Plot the estimates from Part C and Part B together. What do you notice about the estimates as the number of samples gets large?

- **Part D** Adding in priors: Now let's add some prior knowledge about the tuning curve. For the Gaussian case, add a Gaussian prior $p(\boldsymbol{g}) \sim \mathcal{N}(0, \sigma^2\boldsymbol{I})$. Recall that with priors we change from a maximum likelihood to a maximum *a-posteriori* (MAP) estimation. Use the definition of the MAP estimate and Bayes rule to derive the estimator. How does this change the optimization? How about the estimated tuning curve?

  For the Poisson case, add a "smoothing" term $p(\nabla\boldsymbol{g}) \sim \mathcal{N}(0, \sigma^2\boldsymbol{I})$ (following the same procedure). How does this affect the optimization? How about the estimated tuning curve?

- **Part E** Parameter regimes: Finally you should have code that generates a number of samples M that probe an N dimensional receptive field. You should be able to change these parameters as well as the stimulus strength A (multiplier of X) as well as the level of regularization (i.e., the variance of the prior σ). Vary these parameters. What happens at low signal levels? How many samples do you need if $A = 0.01$? Compare these observations with the histograms of the spike counts.

2. **Problem 2: Dimensionality reduction**

   Consider a dataset where M neurons are recorded during repeated trials of a reach task. The resulting data is a number of matrices (one per trial) that is $N \times T$ (number of neurons by time - in 1 ms bins) where $\boldsymbol{X}_{it}$ represents the number of spikes for neuron i during the time-bin t. This problem will consider the dimensionality reduction approach of finding the underlying patterns of neural activity common across trials. Specifically we will first look at standard PCA before and after processing (to highlight the benefit of priors over spike-rates). We will then look at applying an external package that finds smooth spike-rates underlying the neural activity automatically (Gaussian Process Factor Analysis).

   - **Part A** Load the data from `sample_dat.mat`. Compute the peristimulus time histogram (PSTH) of the data. Plot the PSTHs and note any trends. Now pre-process the data by using a Gaussian Process (GP) prior to smooth the data. A GP prior smooths the data by creating a correlation structure that makes nearby points in time have similar values. Specifically, if we treat each neuron's PSTH as $\boldsymbol{x}_n$, which is a T-dimensional vector, the prior distribution is $\boldsymbol{x}_n \sim \mathcal{N}(0, \boldsymbol{K})$, where the covariance matrix $\boldsymbol{K}$ is defined as $\boldsymbol{K}_{ij} = A * e^{-(t_i - t_j)^2/l}$. Use the same approach as Problem 1 to get an estimate of the smoothed PSTH per neuron. Compare the two sets of PSTHs visually and remark on any details you notice. TIP: This computation might take a minute to run depending on your computer. Test on one neuron first before running on all the neurons (this is a good habit to get into with computational work regardless!)

   - **Part B** Compute the Principal components of the PSTH matrix, both with and without smoothing. Look at the temporal principal components. Specifically, plot (side-by-side) the top 3 components as a 3D line plot. What do you notice?

- **Part C** Look in the GPFA folder. Identify how to run GPFA by looking at the demo files included. Run GPFA on the data with a bin-size of 1ms (instead of the default 20ms). GPFA returs per-trial representations of neural activity. Use the function provided in the gpfa code directory `plot3D` to plot the time-courses for all of the trials. What do you notice?
- **Part D** Given trial-by-trial variability, what do you think could be reasonable explanations for the observed deviations of the low-dimensional neural trajectory. Pick one of these hypotheses and describe a theoretical way to test a prediction made by that hypothesis.

3. **Problem 3: Learning a data-driven Linear Dynamical System (LDS)**

- **Part A:** Find the data associated with the jPCA paper (included as exampleData.mat). Load up the data, and read the NOTES.pdf file to identify which data matrix corresponds to "condition 27". What do the values in this matrix represent? Plot the data as a heatmap.
- **Part B:** In this problem we will set up the optimization to learn a linear system guiding a population of neurons. For one dataset write out the probabilistic model corresponding to the discrete approximation of $d\boldsymbol{x}_t/dt = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ is independently identically distributed Gaussian noise with zero mean and standard deviation $\sigma^2\boldsymbol{I}$. For the discretization, use the simple approximation to the differential: $d\boldsymbol{x}_t/dt \approx (\boldsymbol{x}_t - \boldsymbol{x}_{t-1})/dt$. Now what is the same solution when given k datasets that all follow the same dynamics? (Hint: think about each dataset as independent: what's an important property of independent random variables with respect to their probabilities?). The derived likelihood represents $P(\boldsymbol{x}|\boldsymbol{A})$. Use this to set up a ML estimate of $\boldsymbol{A}$. Code up in MATLAB the solution you derived and run on the jPCA dataset. Reconstruct the data by taking each $\boldsymbol{x}_t$ and predicting $\boldsymbol{x}_{t+1}$ by taking one step given the dynamical system you found (via the discrete equation you derived above). Compute the reconstruction error of the data (norm of the difference between the estimated $\boldsymbol{x}_{t+1}$ and the real $\boldsymbol{x}_{t+1}$), and plot the histogram of these errors across all time-points.
- **Part C:** Repeat part B but consider only the top 6 principal components of each matrix X. REMEMBER: PCA projection includes a mean subtraction step! This is very important in projecting the data! Compute again the reconstruction error of the data given the learned dynamical system (with a one-step prediction), and plot the histogram of errors.
- **Part D:** The learned dynamics should provide a projection of the data onto a given space. Take one of the first 6 datapoints in each dataset in the jPCA datasets, and use the learned dynamics from Part C to extrapolate the neural system forward in time. Plot the first two dynamical principal dimensions of the projection: how do they look?
- **Part E:** Now run the jPCA algorithm on the same data. Look at the code and notes provided by the authors and find the right lines of code to run on the dataset. Find the learned dynamics and project the data forward as with Part D. Use "subplot" to plot the projected neural activity next to each other. What key differences do you see between the dynamics? What differences between the dynamical system matrices do you see? Are there similarities or differences e.g., in the eigenspectra?