Task 6: Fit tuning curves & population analysis

Due date: Monday, May 30, 11:59 AM

Task

This week we will further analyze how the neurons' firing rates are modulated by the direction of motion of the grating. To analyze a few more neurons, download `NDA_task6_data.mat`. The file contains the sorted spike times of 41 neurons in the usual format (cells 26–29 are the ones from last week). To look at the responses, use last week's functions.

1. **Estimate firing rates.** Compute the spike counts of each neuron for each direction of motion and trial in `getSpikeCounts()`. The result should be a matrix $\boldsymbol{x}$, where x_{jk} represents the spike count of the j^{th} response to the k^{th} direction of motion (i.e. each column contains the spike counts for all trials with one direction of motion).

   Figure 1: For a selection of neurons plot the average responses to all conditions. Also indicate the variability of the responses in each condition in your plot.

2. **Fit cosine tuning curve.** We will start by fitting the simplest tuning curve possible, a cosine, to the response of each neuron. To do this, project the vector of average spike counts, $m_k = 1/n \sum_j x_{jk}$ on a complex exponential with two cycles, $v_k = 1/4 \exp(2i\theta_k)$, where θ_k is the k^{th} direction of motion (in radians). Denoting the projection by $q = \boldsymbol{m}^{\mathrm{T}}\boldsymbol{v}$, the tuning curve is then given by $f_k = \langle \boldsymbol{m} \rangle + \bar{q}v_k + q\bar{v}_k$, where $\langle \boldsymbol{m} \rangle$ is the overall average spike count. Implement this procedure in the function `fitCos()`.

   Figure: Add the fitted tuning function to your plot in Figure 1. What aspects of the tuning curve does the cosine fit not capture?

3. **Test for significant orientation selectivity.** Based on the above approach, implement a permutation test to quantitatively assess whether a neuron is orientation selective. To do so, estimate the distribution of $|q|$ under the null hypothesis that the neuron fires randomly by running 1000 iterations where you repeat the same calculation as above but on a random permutation of the trials (that is, randomly shuffle the entries in the spike count matrix $\boldsymbol{x}$). The fraction of iterations for which you obtain a value more extreme than what you observed in the data is your *p*-value. Implement this procedure in the function `testTuning()`.

   Figure 2: Illustrate this procedure for one of the cells from above. Plot the sampling distribution of $|q|$ and indicate the value observed in the real data in your plot. How many cells are tuned at $p < 0.01$?

4. **Fit tuning curve using von Mises model.** To capture the non-linearity and direction selectivity of the neurons, we will now fit a modified von Mises function to the data:

$$f(\theta) = \exp\left[\alpha + \kappa\left(\cos\left(2(\theta - \phi)\right) - 1\right) + \nu(\cos(\theta - \phi) - 1)\right]$$

   Here, θ is the stimulus direction. Implement the von Mises function in `tuningCurve()` and plot it to understand how to interpret its parameters $(\phi, \kappa, \nu, \alpha)$. As a first step we will do a least squares fit using Matlab's `lsqcurvefit`. Implement the fitting in `fitLS()`. Select two cells that show nice tuning to test you code.

   Figure: Add the least squares fit to your plot in Figure 1.

5. **Fit tuning functions using Poisson noise model.** Using the above procedure works well in most situations, but the noise model is not ideal. Since spike counts are non-negative and their variance is usually proportional to their mean, the Poisson distribution tends to be a better model. Write down the log likelihood assuming Poisson noise and the tuning functions, differentiate it with respect to the parameters $(\phi, \kappa, \nu, \alpha)$. Implement this calculation in `poissonNegLogLike()` and fit the model to the data using maximum likelihood. Implement the fitting in the function `fitML()`. Instead of using the built-in Matlab minimizer `fminunc` (which is slow) we suggest you use Carl Rasmussen's `minimize` (it's in the same zip file as the function interfaces). Note that it is important that you work on spike counts, not rates. Compare your fits. Do they differ from the least squares solution?

   Figure: Add the maximum likelihood fit under the Poisson model to Figure 1.

Tips

- You can implement an inline function in Matlab using this syntax:
  ```
  f = @(p, theta) = p(1) * theta + p(2)
  ```
  This example obviously implements a linear function in θ with parameters p_1 and p_2. You evaluate the function by calling
  ```
  f(p,0:0.1:2*pi)
  ```
- For the last task, numerically verify the gradients you calculated before running the optimizer.
- As you are optimizing non-convex functions, local optima are a potential issue. It may therefore help to initialize the parameters sensibly before using `lsqcurvefit` and `minimize`.