PSYCH-GA.2211/NEURL-GA.2201 – Fall 2017
Mathematical Tools for Neural and Cognitive Science

Homework 6

Due: 19 Dec 2017
(late homeworks penalized 10% per day)

See the course web site for submission details. Please: don't wait until the day before the due date... start *now*!

1. **Reverse Correlation.** Download from the course web page this function

   ```
   [spikes, stimuli] = runGaussNoiseExpt(kernel, duration)
   ```

   that simulates a white noise (reverse correlation) experiment. The `kernel` is a spatial weighting vector, and `duration` specifies the total number of random stimuli that will be shown. The function returns `spikes`, a binary vector indicating which stimuli produced spikes, and `stimuli`, a matrix whose rows contain the simuli.

   (a) Generate a 100-sample response vector by running the function on the spatial kernel

   ```
   [-1 -2 0; -2 0 2; 1 2 0]/6
   ```

   This is a matrix, which you'll need to stretch out into a column vector (i.e., `kernel(:)`) before passing it into the function. Note that this kernel is unit-norm (sum of squares of weights is one). Plot (on two subplots of the same figure) the linear filter response to the stimuli (you should be able to compute this with a single matrix multiplication!) and the spike train, both as functions of time. Do you see a relationship between these? Display a 2D scatter plot of the raw stimulus intensities at positions 1 and 6, which should look like samples of a 2D Gaussian. On top of this (use `hold on`), plot in red only the stimulus intensities that produced spikes. Where in this 2D stimulus space do the spikes occur? Does this match your expectations?

   (b) Now compute the spike-triggered average (STA) for the simulated data from this "experiment". Rescale it to have unit norm, reshape it into a matrix, and display it as a grayscale image. Display the true kernel next to it (use `subplot`). How similar is the STA to the true kernel? Characterize the error of the STA, as a function of the duration of the experiment. For durations $100, 200, 400, 800, 1600, 3200, 6400$, run the experiment 100 times, compute the mean STA across these 100 runs, and subtract this from the true kernel, and compute the sum of this difference kernel (the estimation bias). Also, subtract the computed mean from the 100 STAs, and compute the average squared error over these (the estimation variance). Plot the bias and square root of the variance as functions of the duration – you might want to look at a log-log plot (matlab has a function `loglog`). What do you conclude about how the bias and variance behave, as a function of the amount of data?

   (c) Repeat the exercise above, but use the function `runBinNoiseExpt`, which uses binary noise instead of Gaussian noise. Compare these plots to those for the Gaussian case. How do they differ? Why?

(d) Estimate the nonlinearity of the response. Take the stimuli, spikes, and STA from the experiment with duration 1600, project the stimuli onto the STA. Sort these projections from highest to lowest (using matlab's `sort` function) and re-order the spike vector to maintain correspondence (the sort function will give you the indices of the sorted values). For each consecutive group of 200 indices, compute the average projected stimulus value, and the average number of spikes (you should collect these in two vectors of length 16), and plot these against each other. You should see an estimate of the spiking nonlinearity.

(e) Replot this nonlinearity with error bars, computed by bootstrapping. Draw a set of 1600 random integer indices in the range [1:1600], and use these to resample the projection and spike vectors, and recompute the nonlinearity for this bootstrap-resampled data. Note that unlike the case above, you'll need to do this with a set of fixed position bins, so that you can average across multiple bootstrap samples. Do this 100 times, to get 100 estimated nonlinearities. Plot the mean nonlinearity, and standard deviation, as points with error bars (use the matlab function `errorbar`).

2. **Simulating a 2AFC experiment.** Consider a two-alternative forced choice (2AFC) psychophysical experiment in which a subject sees two stimulus arrays of some intensity on a trial and must say which one contains the target. (One and only one contains the target.) Her probability of being correct on a trial is:

$$p_c(I) = 1/2 + 1/2\Phi(I; \mu, \sigma)$$

where $\Phi(I; \mu, \sigma)$ is the cumulative distribution function of the Gaussian (`normcdf` in matlab) with mean μ and standard deviation σ evaluated at I. The function $p_c(I)$ is known as the *psychometric function*. (Minor note, somewhat subtle: This setup only makes sense if I is on a logarithmic scale, e.g., $I = k \log C$, where C is stimulus contrast.)

(a) Plot two psychometric functions, for $\{\mu, \sigma\}$ equal to $\{5, 2\}$ and $\{4, 3\}$. (Use $I = [1 : 10]$). Describe the difference between these. If you increase μ, how does the curve change? If you increase σ, how does the curve change? (If you are not sure, make more plots with different parameter values.) What is the range of $p_c(I)$? Explain why this range is appropriate.

(b) Write a function `C=simpsych(mu,sigma,I,T)` which takes two vectors `(I,T)` of the same length, containing a list of intensities and the number of trials for each intensity, respectively, simulates draws from $p_c(I)$, and returns a vector, C, of the same length as I and T, which contains the number of trials correct out of T, at each intensity I.

(c) Illustrate the use of `simpsych` with `T=ones(1,7)*100` and `I=1:7` for $\mu = 4$ and $\sigma = 1$. Plot `C ./ T` vs `I` (as points) and plot the psychometric function $p_c(I)$ (as a curve) on the same graph.

(d) Do the same with `T=ones(1,7)*10` and plot the results (including the psychometric function). What is the difference between this and the plot of the previous question?

3. **Fitting a psychometric function.** Now we'll flip things around, and use this probabilistic model as a means of fitting/analyzing a simulated data set.

(a) Write a function `nll = nloglik(mu,sigma,I,T,C)` that returns the negative of the log likelihood of parameters `mu` and `sigma`, for data set `I,T,C`.

(b) Generate a contour plot (function contour, using 50 lines) of the negative log likelihood of the data set from part (c) of the previous problem, for all pairs of `mu` from `muall = [2:0.2:10]` and a `sigma` from `sigmaall = [0.5:0.2:6]`. What is the approximate location of the best fitting pair of parameters from this plot (determined visually)?

(c) Use the function `fminsearch` to find the more precise values `mu, sigma` that minimize the function `nloglik(mu, sigma,)`. Three notes: first, the syntax for calling `nloglik` within `fminsearch` is a bit odd:
`fminsearch(@(x) nloglik(x(1),x(2),I,T,C), <startpoint>).`
Here, the `@` notation is used to create a temporary function, with argument `x` a vector containing the two variables being optimized (mean and stdev). Second, `fminsearch` minimizes rather than maximizes, which is why we must use the *negative* of the log likelihood. Third, you'll need to specify a start point for the search – for this problem, `[2,2]` is a reasonable choice.

(d) A variant of `fminsearch, fminunc`, also returns the *Hessian* (the matrix of second derivatives) of the negative log likelihood at the optimal `mu` and `sigma`. The inverse of the Hessian provides an estimate of the covariance matrix of the parameter estimates. Use this to determine 95% confidence intervals on each parameter (Hint: The marginal standard errors for each parameter are the square roots of the diagonal entries of the inverse Hessian; a 95% confidence interval is the mean plus or minus 1.96 standard errors.) Do the true parameter values (4 and 1) fall within the confidence intervals? Note: `fminunc` is less robust than `fminsearch`, and if the optimizer strays too far from the true values, there may be numerical problems due to overflow of the likelihood; in this case, try a different starting point.

(e) Produce a second set of confidence intervals for the parameters using a bootstrap method. For each of the 7 intensities, resample 100 trials (correct or incorrect) from the 100 trials of that intensity in the original data, with replacement. Refit the model to the resampled data using `fminsearch`. Plot the histograms (function hist) of `mu` and `sigma` estimates obtained over 500 such resampled datasets, and define your confidence intervals as the region between the 2.5th and 97.5th percentiles of these distributions. How well do these values agree with those from 2D?

4. **Classification (decision) in a 2-dimensional space**. Load the file `fisherData.mat` into your MATLAB environment. The file contains two data matrices, `data1` and `data2`, whose rows contain hypothetical normalized responses of 2 mouse auditory neurons to different stimuli – The first matrix contains responses to dogs barking, and the second are responses to cat vocalizations. You would like to know whether the responses of these two neurons could be used by the mouse to differentate the two types of sound. We'll implement three *classifiers*.

(a) First consider the linear discriminant corresponding to the difference in means of the two data sets. Convince yourself (write the math) that this solution is the Maximum Likelihood classifier under the assumption that the data are drawn from Gaussian distributions with identity (or any multiple of the identity) covariance. Visualize the solution by generating a binary image showing the classification output. Specifically, use `meshgrid` to generate X and Y coordinate images covering the a region that extends a bit beyond the range of the data, compute images of the two Gaussians, with mean matching the correponding data sets and identity covariance, and then calculate the

binary classifier by comparing the two Gaussian images (g1 ¡ g2). Display the image using `image` (make sure to provide x and y coordinates), and plot the data on top of the image. Now compute the discriminant vector (compute the differnece of the means of each data set, and normalize to unit length). Plot this on the same graph as the decision image, plot the linear discriminant vector, verifying that it is perpendicular to the decision boundary (use `axis equal` to make the units on the two axes the same). What fraction of the points are correctly classified by this classifier?

(b) Now use Fisher's Linear Discriminant, which maximizes the average squared between-class mean distance, while minimizing the sum of within-class squared distances (see Notes on regression). This classifier is the ML solution when the data are drawn from Gaussian distributions with different means, but the same covariance matrix (which need not be a multiple of the identity!). Estimate the common covariance by averaging together the covariances of the two data matrices. Repeat the plotting exercises of part (a) to visualize the solution. Again, what fraction of the points are correctly classified by this classifier?

(c) Finally, develop a Quadratic Classifier, that computes the Maximimum likelihood decision rule assuming the data are drawn from two different Gaussian distributions (i.e., each with its own mean and covariance). Specifically, estimate the mean and covariance of data measured for each condition, and calculate the classifier that chooses the class of each data point based on which of the two Gaussians has higher probability at that location (write out the math). Repeat the plotting exercises of part (a) (except now there's no discriminant vector to plot). Calculate the fraction of correctly classified data points. Which of the three classifiers is best? Do you think that would always be the case, or are there data scenarios in which you might choose to use one of the inferior classifiers?