BE 521: Homework 5

Vision

Spring 2021

42 points

Due: Tuesday, 03/02/2021 10 pm

Objective: Visual responses and likelihood

Shubhankar Patankar

V1 Dataset

In this homework, you will work with data from 18 cells recorded from mouse primary visual cortex (also known as V1). Cells in this area are responsive to specific angles. Hence, a common stimulation paradigm is to show the subject a sinusoidal grating drifting at a specific angle (see Figure 1). This data was collected

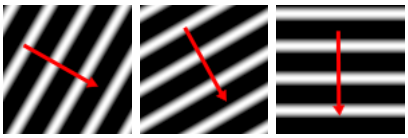


Figure 1: Example sinusoidal drifting grating at (in order left to right) 30, 60, and 90 degrees, where the red arrows shows the direction of the drift.

and graciously provided by Daniel Denman in the Contreras Lab, University of Pennsylvania. The file `mouseV1.mat` contains two variables: `neurons`, a cell array representing all the times that each of the 18 cells fired a spike during the approximately 7 minute long experiment, and `stimuli`, which provides the time (first column, in milliseconds) that a given angle (second column) was presented in this experiment. Note that each stimulus in `stimuli` is presented for exactly 2 seconds, after which a gray screen is presented for approximately 1.5 seconds (therefore each trial is approximately 3.5 seconds in duration.)

1 Stimulus Response (11 pts)

In this section, you will explore the response of the cells to different stimulus angles.

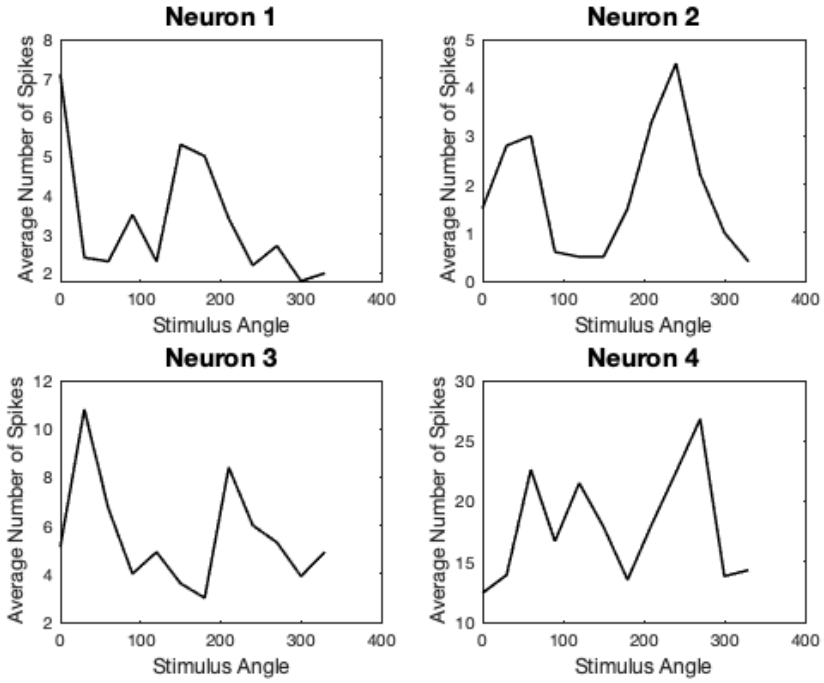
1. How many unique grating angles, m, are there in `stimuli`? (1 pt)

```
cd('/Users/sppatankar/Developer/BE-521')
addpath(genpath('Homework_5'));
load('mouseV1.mat')

num_neurons = length(neurons);
num_stimuli = length(stimuli);
all_angles = unique(stimuli(:, 2)); % unique() sorts in ascending order
num_unique_angles = length(all_angles)
```

```
num_unique_angles =

    12
```

2. A *tuning curve* is frequently used to study the response of a neuron to a given range of input stimuli. To create tuning curves for this data, calculate the average number of spikes each cell fires in response to each grating angle. Store the result in an $18 \times m$ dimensional matrix, where each element represents the response of a single neuron to a particular input stimulus angle, with each neuron assigned a row and each angle assigned a column. In a 2×2 Matlab subplot, plot the tuning curve for the first four cells. Place the stimulus angle on the x-axis and the number of spikes on the y-axis. (6 pts)

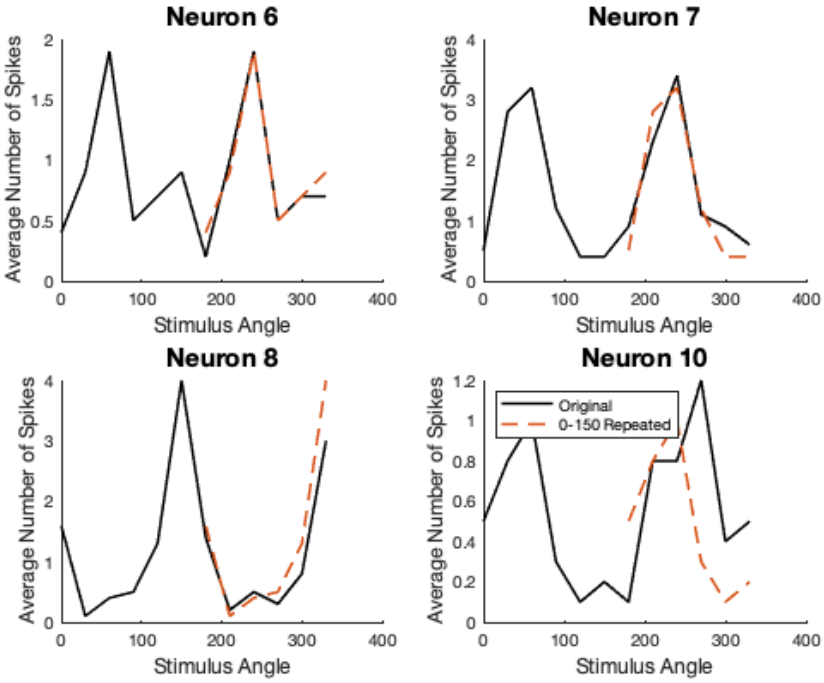
```
stim_durations = diff(stimuli(:, 1));
stim_durations(end + 1) = round(mean(stim_durations));

tuning_curves = zeros(num_neurons, num_unique_angles);

for i = 1:num_unique_angles
    curr_angle = all_angles(i);
    % Find the indices when the current angle is used for stimulation.
    curr_angle_idx = find(stimuli(:, 2) == curr_angle);
    % Initialize an empty array to store how many spikes each neuron fires
    % for each use of this angle.
    angle_response = zeros(num_neurons, length(curr_angle_idx));
    for j = 1:num_neurons
        curr_neuron = neurons{1, j};
        % Loop over each use of the current angle to find how often each
        % neuron fired in response.
        for k = 1:length(curr_angle_idx)
            stim_start_time = stimuli(curr_angle_idx(k), 1);
            stim_end_time = stim_start_time + stim_durations(curr_angle_idx(k));
            spikes = curr_neuron((curr_neuron > stim_start_time) ...
                & (curr_neuron < stim_end_time));
            angle_response(j, k) = length(spikes);
        end
    end
    % Take the mean of the number of spikes for each neuron.
    tuning_curves(:, i) = mean(angle_response, 2);
end

figure;
for i = 1:4
    subplot(2, 2, i)
    plot(all_angles, tuning_curves(i, :), 'LineWidth', 1.5, 'Color', [0, 0, 0])
    xlabel('Stimulus Angle', 'FontSize', 12);
    ylabel('Average Number of Spikes', 'FontSize', 12);
    title(sprintf('Neuron %d', i), 'FontSize', 15);
end
```



(a) Look through the tuning response curves of each of the 18 cells. How reasonable is it to assume that the response of a given cell to angle θ is the same as its response to angle $\theta + 180$? Include at least a few tuning curves to back up your answer. (2 pts)

It is pretty reasonable to assume that the response of a cell to angle θ is the same as its response to angle $\theta + 180$. In the plots that follow, the response in the domain from 0 to 180 is reproduced and overlaid over the 180 to 360 domain in red ink. We see that the true response in that domain is highly correlated with the overlaid response.

```
plot_neurons = [6, 7, 8, 10];
figure;
for i = 1:length(plot_neurons)
    neuron_ID = plot_neurons(i);
    subplot(2, 2, i);
    hold on
    plot(all_angles, tuning_curves(neuron_ID, :), ...
        'LineWidth', 1.5, 'Color', [0, 0, 0])
    plot(all_angles(1:6) + 180, tuning_curves(neuron_ID, 1:6), ...
        'LineWidth', 1.5, 'LineStyle', '--')
    hold off
    xlabel('Stimulus Angle', 'FontSize', 12);
    ylabel('Average Number of Spikes', 'FontSize', 12);
    title(sprintf('Neuron %d', neuron_ID), 'FontSize', 15);
end
legend('Original', '0-150 Repeated', ...
    'Location', 'NorthWest');
```



(b) Does this assumption have any physiological justification (given what you know about the types of cells in V1)? (2 pts)

V1 neurons are highly sensitive to stimulus orientation. The region is comprised of simple and complex cells. Simple cells respond to specific orientations in the visual receptive field, whereas complex cells integrate inputs from multiple simple cells. Since the stimuli being presented are invariant to 180 degree rotations, it makes sense that the spike recordings from V1 cells are also invariant to them.

2 Neural Decoding (31 pts)

Suppose we would like to work backwards - that is, for a given neural response, can we predict what the grating angle was? This process is called "neural decoding," and is especially of interest to the BCI motor control community (as we'll see in a later homework). In this section, we will try out an approach which is detailed in Jazayeri & Movshon 2006 [1]. The method we will use involves finding the maximum likelihood of the data.

Here, the data is the number of spikes s_i that cell i fires when the subject sees a stimulus with grating angle θ. One way to think about our likelihood function is to ask the question "given a stimulus angle θ, how many spikes would I expect this cell to fire?" We can represent this number of spikes s_i using a Poisson process with parameter $f_i(\theta)$ for a stimulus θ, where f_i represents neuron i's tuning function. A Poisson distribution is often used to model count data that occurs at a constant rate, and in this case the rate is given by $f_i(\theta)$. In other words, our likelihood function $L_i(\theta)$ for each neuron i is the probability $p(s_i|\theta)$ of neuron i firing s_i spikes for a given value of θ. The idea in this method is to calculate the log likelihood[2] function of each neuron and then add them all together to get the log likelihood function of the entire population

[1] A copy of this paper is included if you are curious, but we walk you through the method in this homework.
[2] log = natural log unless otherwise specified

of (n) neurons. We often work with the *log* likelihood because it allows adding of probabilities instead of multiplying, which can lead to numerical problems.

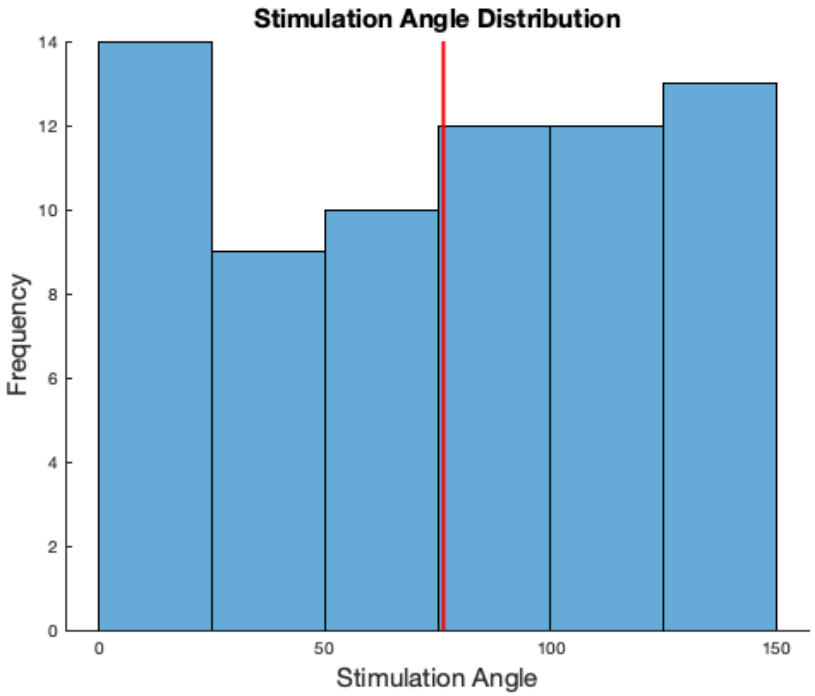
$$
\begin{aligned}
p(s_i|\theta) \sim Pois(f_i(\theta)) &= \frac{f_i(\theta)^{s_i}}{s_i!} e^{-f_i(\theta)} && \text{(Poisson probability density)} \\
Ł_i(\theta) &= p(s_i|\theta) && \text{(Likelihood of a given neuron firing at } s_i\text{)} \\
Ł(\theta) &= \prod_{i=1}^{n} p(s_i|\theta) && \text{(Joint likelihood of all n neurons)} \\
\log Ł(\theta) &= \sum_{i=1}^{n} \log L_i(\theta) = \sum_{i=1}^{n} \log p(s_i|\theta) && \text{(Take log)} \\
&\propto \sum_{i=1}^{n} s_i \log f_i(\theta) && \text{(evaluation of PDF and simplifying)}
\end{aligned}
$$

Thus, we can define the log likelihood for each neuron i as the log of its tuning curve $f_i(\theta)$ times the number of spikes s_i it fires for a particular stimulus θ, and the population log likelihood is simply the summation across all cells. This tells us that, given a set of tuning curves $f_i(\theta)$, we can compute the likelihood of observing our data s. But we already have those tuning curves for each cell from question 1.2, so all we need to know for a new (hidden) stimulus is how many spikes each neuron fires. Let $\mathbf{s}$ be the n-dimensional column vector of the number of spikes each cell fires after the subject is presented with a new stimulus θ' and let $\mathbf{F}$ be the $n \times m$ matrix representing the tuning curves of each neuron at each of the m stimuli (for us, m is the number of stimuli between 0 and 150 degrees because we assume that all neurons respond equally to θ and $\theta + 180$ degrees.) We can then compute the log likelihood of the new stimulus θ' easily using the inner product of $\mathbf{s}$ and $\mathbf{F}$: `L = s'*log(F)`.

1. Compute the matrix `F` by recalculating the tuning curves you calculated in question 1.2 using only the **first 70** trials (this is akin to our "training" data). You will use the remaining 50 trials (as "testing" data) to make predictions. Make a histogram of the number of stimulation angles for the first 70 trials to ensure that each angle (0 to 150) is presented at least a few times. (4 pts)

```
% 1) Use only the first 70 trials to compute tuning curves
% 2) Merge data for angles 180-330 with 0-150

stim_durations = diff(stimuli(:, 1));
stim_durations(end + 1) = round(mean(stim_durations));

tuning_curves_train_all = zeros(num_neurons, num_unique_angles);

for i = 1:num_unique_angles
    curr_angle = all_angles(i);
    % Find the indices when the current angle is used for stimulation.
    % Note that we are looking for the current angle only in the first 70
    % stimulation trials.
    curr_angle_idx = find(stimuli(1:70, 2) == curr_angle);
    % Initialize an empty array to store how many spikes each neuron fires
    % for each use of this angle.
    angle_response = zeros(num_neurons, length(curr_angle_idx));
    for j = 1:num_neurons
        curr_neuron = neurons{1, j};
        % Loop over each use of the current angle to find how often each
        % neuron fired in response.
        for k = 1:length(curr_angle_idx)
            stim_start_time = stimuli(curr_angle_idx(k), 1);
            stim_end_time = stim_start_time + stim_durations(curr_angle_idx(k));
            spikes = curr_neuron((curr_neuron > stim_start_time) ...
                & (curr_neuron < stim_end_time));
```

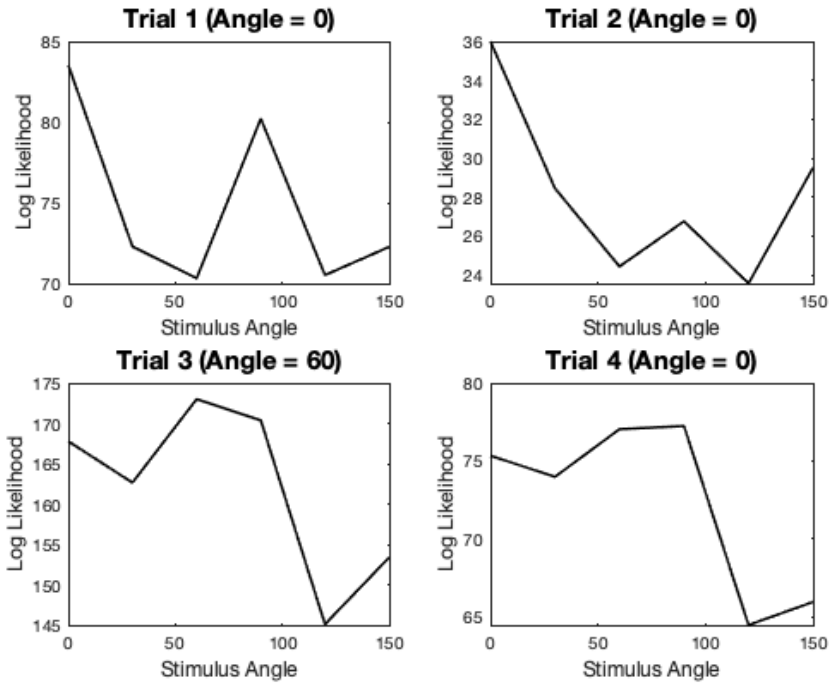
```
            angle_response(j, k) = length(spikes);
        end
    end
    % Take the mean of the number of spikes for each neuron.
    tuning_curves_train_all(:, i) = mean(angle_response, 2);
end

F = (tuning_curves_train_all(:, 1:6) + ...
    tuning_curves_train_all(:, 7:12)) ./ 2;

% Histogram of stimulation angles for the first 70 trials
F_stim_angles = stimuli(1:70, 2);
F_stim_angles(F_stim_angles > 150) = F_stim_angles(F_stim_angles > 150) - 180;
figure;
hold on
histogram(F_stim_angles, length(unique(F_stim_angles)));
plot([mean(F_stim_angles); mean(F_stim_angles)], ...
    repmat(ylim', 1, 1), '-r', 'LineWidth', 2)
hold off
xlabel('Stimulation Angle', 'FontSize', 15);
ylabel('Frequency', 'FontSize', 15);
title('Stimulation Angle Distribution', 'FontSize', 15);
```



2. For the 50 "testing" trials, compute a $n \times 50$ matrix `S` where each row represents the number of spikes one neuron fired in response to a each of the 50 trials. With this, you can easily compute the log likelihood functions for all the trials at once with the command: `L_test = S'*log(F)`. (Hint: add a small number to `F` to avoid taking the log of 0)

   (a) Plot the likelihood functions for the first four testing trials in a 2×2 subplot. In the title of each plot, give the trial number (1, 2, 3, or 4) and the true stimulation angle. Make sure to label your axes correctly. (5 pts)

```
test_stim = stimuli(71:end, :); % all test stimulation data
test_stim_angles = test_stim(:, 2); % test stimulation angles
% Replace angles such that theta + 180 = theta
test_stim_angles(test_stim_angles > 150) = ...
    test_stim_angles(test_stim_angles > 150) - 180;
% Generate duration information for the stimuli
test_stim_durations = diff(test_stim(:, 1));
test_stim_durations(end + 1) = round(mean(test_stim_durations));

s = zeros(num_neurons, length(test_stim));
test_angles = zeros(1, length(test_stim));

for i = 1:length(test_stim)
    stim_start_time = test_stim(i, 1);
    stim_end_time = stim_start_time + test_stim_durations(i);
    test_angles(i) = test_stim(i, 2);
    for j = 1:num_neurons
        curr_neuron = neurons{1, j};
        spikes = curr_neuron((curr_neuron > stim_start_time) ...
            & (curr_neuron < stim_end_time));
        s(j, i) = length(spikes);
    end
end

LL = s' * log(F);

figure;
for i = 1:4
    subplot(2, 2, i)
    plot(unique(F_stim_angles), LL(i, :), 'LineWidth', 1.5, 'Color', [0, 0, 0])
    xlabel('Stimulus Angle', 'FontSize', 12);
    ylabel('Log Likelihood', 'FontSize', 12);
    title(sprintf('Trial %d (Angle = %d)', i, test_stim_angles(i)), ...
        'FontSize', 15);
end
```



(b) How well do these four likelihood functions seem to match the true stimulation angle? Explain in a few sentences. (3 pts)

Panels 1, 2 and 3 in the figure above are plots of likelihood functions where the *argmax* over the stimulus angles matches the true stimulus angle. However, the MLE method does not get the stimulus angle right for panel 4.

(c) Compute the maximum likelihood estimate (MLE) for each of the 50 trials. This is another way of asking which angle θ has the highest probability.

$$\hat{\theta}_{MLE} = \arg\max_{\theta} \; L(\theta) = \arg\max_{\theta} \; \log L(\theta)$$

In what percentage of the 50 trials did your MLE correctly predict the stimulation angle [0-150]? (5 pts)

```
pred_angles = zeros(1, length(test_stim));
all_angles_sym = all_angles(1:6); % sym refers to rotational symmetry

for i = 1:length(test_stim)
    [~, max_idx] = max(LL(1, :));
    pred_angles(i) = all_angles_sym(max_idx);
end

accuracy = sum(test_angles ~= pred_angles)/length(test_angles)
```

```
accuracy =

    0.9600
```

(d) In a few sentences, discuss how well this method worked for predicting the input angle from the response of the 18 neurons. What might you change in the experimental protocol to try and get a better prediction? (3 pts)

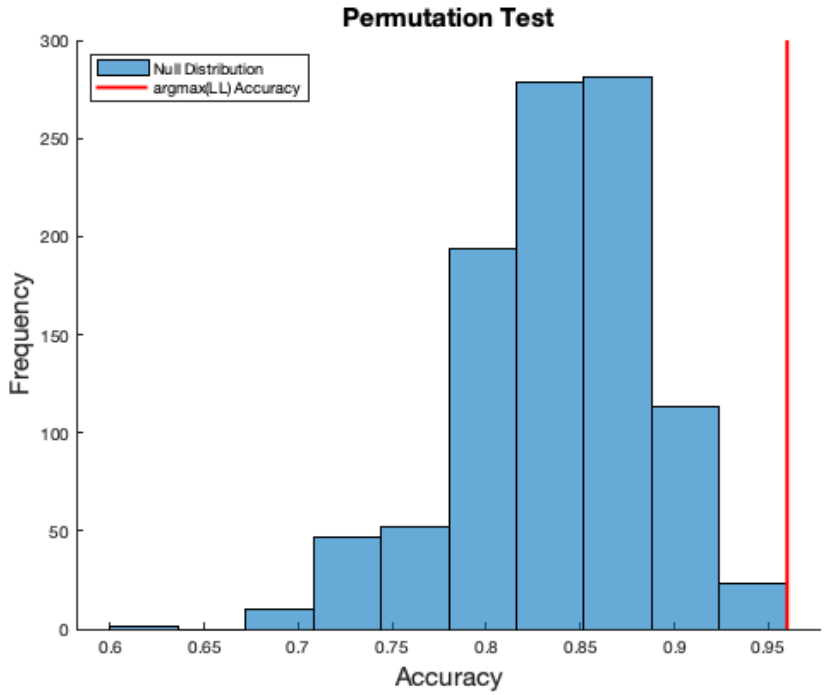
This method works incredibly well in predicting the stimulus angle from neural responses. Based on the likelihood traces in the figure for 2(a), and in particular for Neuron 4, gathering more data for intermediate stimuli may help make the curves smoother as well as improve the signal-to-noise ratio. Additionally, we could augment the data with sinusoidal gratings that drift at different rates. This information could act as an extra feature when inferring the likelihood functions.

3. It is important to show that your findings are not a result of chance. One way to demonstrate this is using a "permutation test." Here, we will perform a permutation test by randomly reassigning new grating angles to the 50 test responses and then calculating how often the new grating angles match the true stimulation angles.

(a) Simulate the chance prediction (the "null" distribution) by randomly reassigning the stimulation angles 1000 times. For each permutation, calculate the percent accuracy of each label. Create a histogram of the 1000 accuracy measurements for the null distribution. Add a red vertical line with the accuracy calculated from using the Jazayeri method above. (4 pts)

```
num_iters = 1000; % 1000 permutations
accuracy_perm = zeros(1, num_iters);
for i = 1:num_iters
    test_angles_perm = datasample(unique(F_stim_angles), length(pred_angles))';
    accuracy_perm(i) = sum(test_angles_perm ~= pred_angles)/length(test_angles_perm);
end
```

```
F_stim_angles = stimuli(1:70, 2);
F_stim_angles(F_stim_angles > 150) = F_stim_angles(F_stim_angles > 150) - 180;
figure;
hold on
histogram(accuracy_perm, 10);
plot([accuracy; accuracy], ...
    repmat(ylim', 1, 1), '-r', 'LineWidth', 2)
hold off
xlabel('Accuracy', 'FontSize', 15);
ylabel('Frequency', 'FontSize', 15);
legend('Null Distribution', 'argmax(LL) Accuracy', ...
    'Location', 'NorthWest');
title('Permutation Test', 'FontSize', 15);
```



(b) Is the null distribution what you expected? Explain. (1 pt)

It makes sense that the null distribution resembles a bell curve reflecting the random assignments that are made for the labels. It is pretty surprising, however, that the mean of this Gaussian distribution is at roughly 0.85. This suggests that simply randomly selecting labels for the stimulus angles is also likely to perform really well making the permutation test all the more important.

(c) What is the probability that your actual accuracy measurement comes from the null-distribution? That is, calculate the fraction of permutation samples with accuracy *more extreme* relative to the mean of the null distribution (less than your measurement if your value is less than the mean, or more than your measurement if your value is more than the mean). (2 pts)

```
p_value = sum(accuracy_perm > accuracy)/num_iters
```

```
p_value =

     0
```

(d) What would the probability be if your accuracy had been 25%? (1 pt)

```
p_value_25 = sum(accuracy_perm > 0.25)/num_iters
```

```
p_value_25 =

     1
```

(e) What is this value typically called? (1 pt)
This probability is typically called a p-value.

4. The tuning curves and neuron responses to a new trial were calculated using the number of spikes each neuron fired after the stimulation. But what if a particular neuron just happens to fire a lot and fires even more when it gets a stimulus to which it's tuned? Those neurons could "swamp" the log likelihood estimate given in Equation 1 by virtue of just having a larger average s_i and $f_i(\theta)$. How might we correct for this type of behavior? Suggest a possible method. (2 pts)

   One approach might be to z-score the neural response information. In other words, scale every neuron's response to have zero mean and unit standard deviation. This way, neurons that might have the potential to swamp the likelihood cannot do so.