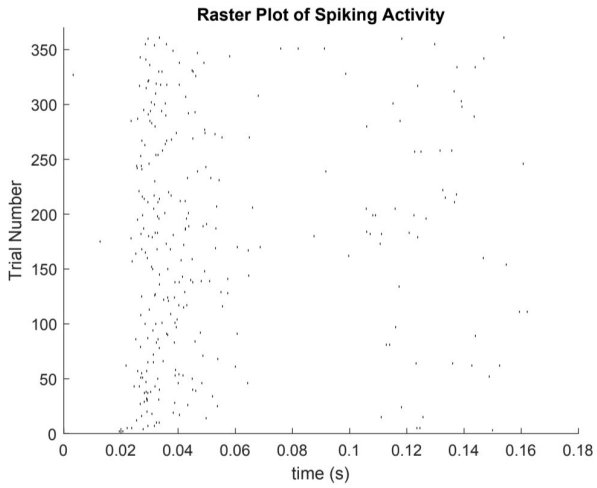
Neuro 120 HW #2: Data Analysis

Team: Andrew Sullivan, Sanjay Patil, Harriet Tieh, and Saloni Vishwakarma

October 18, 2018

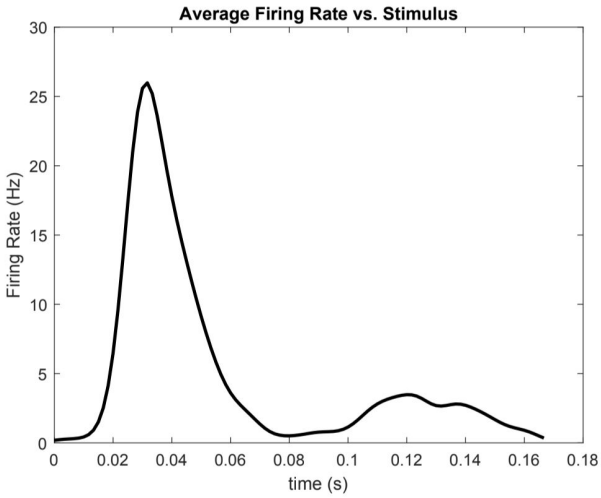
1. **Auditory neuroplasticity**

    a. The raster plot below describes the response of a single neuron to the exposure stimulus:



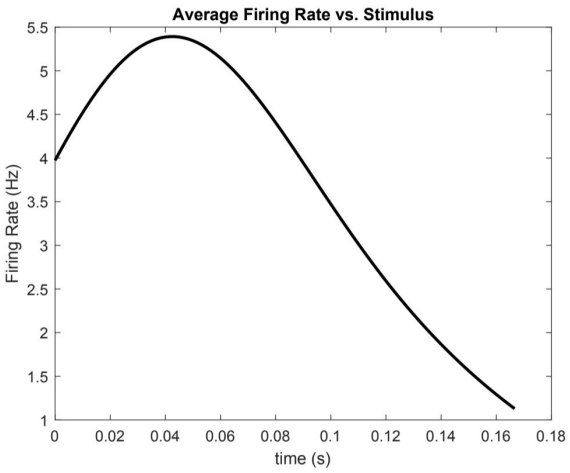
    Yes, this neuron appears to respond to the stimulus. For each of the trials, there appears to be spiking activity in response to the exposure stimulus, beginning at approximately 0.03 seconds after the presentation. A second, smaller increased firing rate also appears at approximately 0.12 s.

    b. Plot of the average firing rate for this neuron in response to the stimuli by smoothing with a Gaussian kernel. Use a Gaussian of standard deviation σ = 5ms.

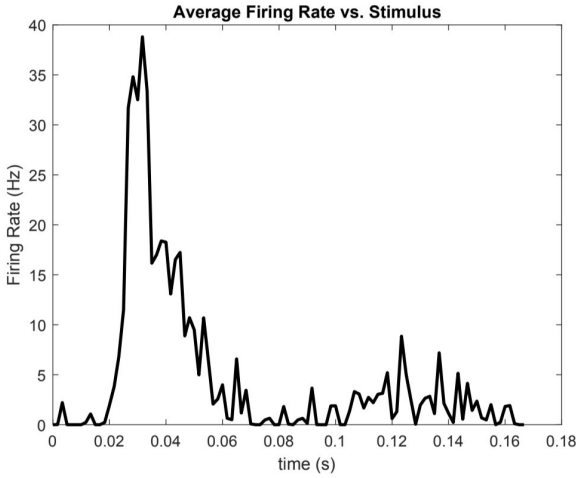


c. Generate plots for σ = 50ms and σ = 0.5ms. What is the impact of very large or very small standard deviations?

σ = 50ms

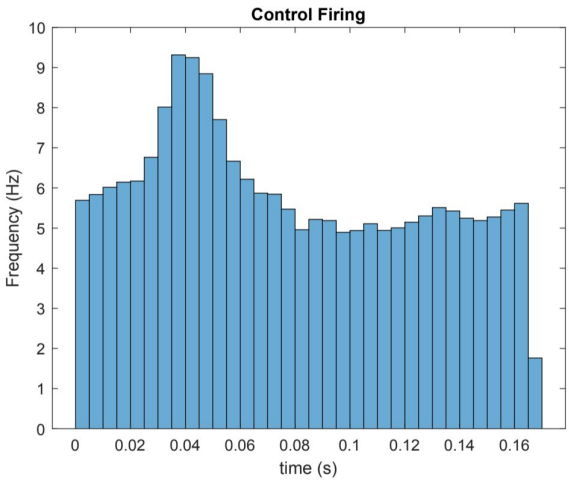


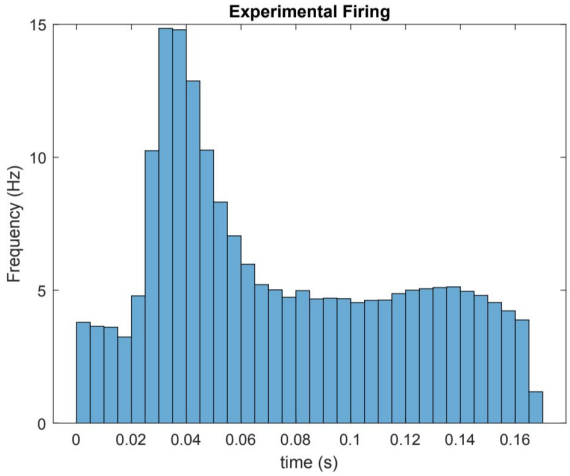
σ = 0.5ms



Having a larger standard deviation renders the curve much smoother over time, and also results in a smaller average firing rate in response to stimuli at each time frame. This larger deviation also obscures the second smaller peak in firing rate seen in the 5ms and 0.5ms cases. Conversely, a small standard deviation, as seen in the 0.5ms case, yields a noisier curve (many local extrema) that partially masks the two peaks.

d. So far we have examined a single neuron. Now let's see if these trends hold more generally, and investigate how neural responses might have changed between the control and experimental groups. Compute and plot a grand average post stimulus time histogram for all spikes and all recorded neurons, for both the control and experimental groups. That is, count the average number of spikes falling into a set of time bins (here the average is over the stimuli and the number of neurons from which we have responses), divided by the bin width to yield a firing rate in Hz. Use a bin width of 5ms.

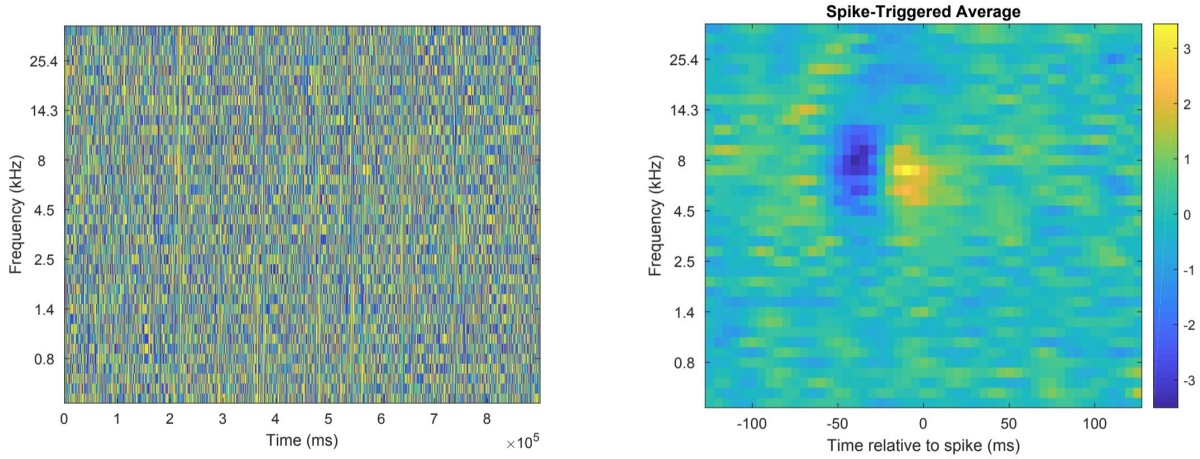




e. What differences do you observe between the experimental and control groups? Did neurons become more or less selective to the exposure stimulus? Did the precision of responses change? Are there any differences in responses after the initial peak response?

In the control group, the peak in the neural responses to the stimuli was less extreme than that of the experimental group, and the average firing rate before and after the spike is greater than that of the experimental group (~5-6 Hz compared to ~3-5 Hz). In the experimental group, the neurons demonstrated increased selectivity and more precise responses to the exposure stimulus—as seen in the large spike in firing frequency in response to the stimulus to ~15 Hz, and then a quick return to the baseline ~5 Hz firing frequency. In comparison to the ~9 Hz to ~6 Hz differential of the control neurons, this return to a lower baseline indicates that the experimental neurons are more tuned to the exposure stimulus.

f. Generate a spike-triggered average for this neuron that extends 125 ms into the past and 125ms into the future. That is, for each spike, cut out a segment of the stimulus corresponding to the 250ms window centered on the spike, and average these together. The portion of the STA that is 125ms into the future is a useful sanity check: the stimulus in the future cannot impact the spiking behavior of the neuron, so we expect to see an STA that is approximately zero for these future times.



The first figure is a spectrogram that displays the most used frequencies over time, the second figure displays the STA for the neuron.

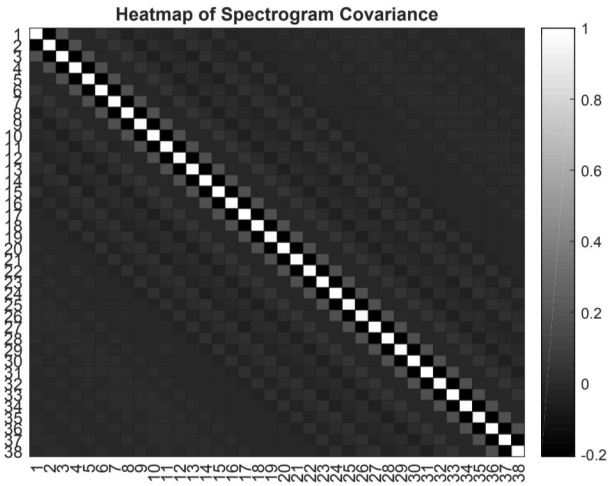
g. What frequency is the neuron most selective to?

The neuron is most selective to the frequency ~7 Hz, as the high-intensity (orange-yellow) spot right before the spike highlights the frequencies most associated with the spike.

h. According to the STA, would the neuron's peak response be higher to a constant tone at its preferred frequency, or to a brief tone pip? Approximately how long should a tone pip be to evoke the largest response?

According to the STA, the neuron's peak response would be higher to a brief tone pip than a constant tone at its preferred frequency. The tone pip should be about 10ms long. Any longer stimulus would overlap with the low-intensity (blue) region, likely diminishing the stimulus' effect and decreasing the likelihood of a significantly increased firing rate in the neuron.

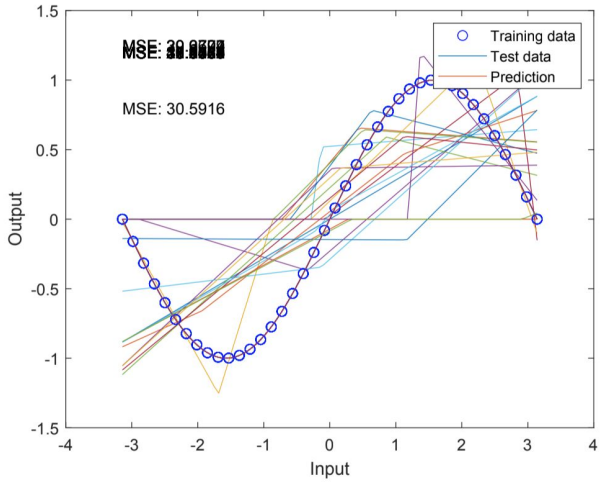
i. The STA is sensitive to stimulus correlations, and is only the optimal linear filter if the stimulus correlations are white (the identity matrix). Plot the stimulus correlation matrix (stim_spectrogram*stim_spectrogram'). Is it close to an identity matrix?



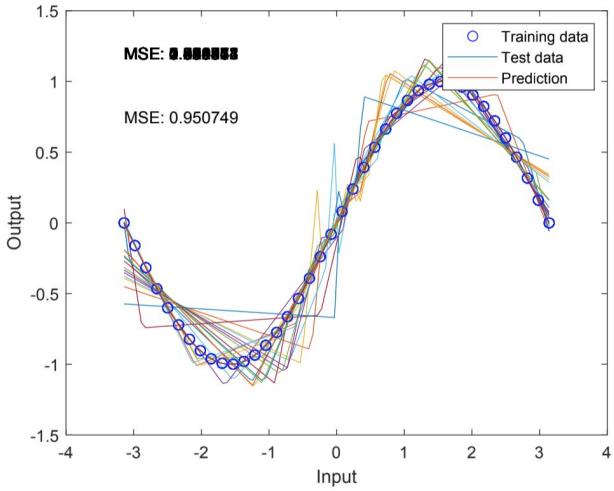
Yes, it is close to the identity matrix, but the presence of small fluctuations (the gray regions) outside of the main diagonal suggest the presence of some structure (i.e. correlation) in the stimulus.

2. **Random neural networks and overfitting in regression.**
    a. Plot several example predictions for Nh =2,10,26.

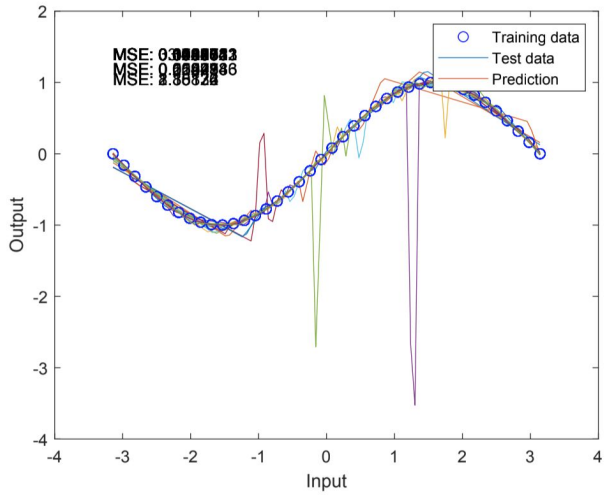
Nh = 2 neurons



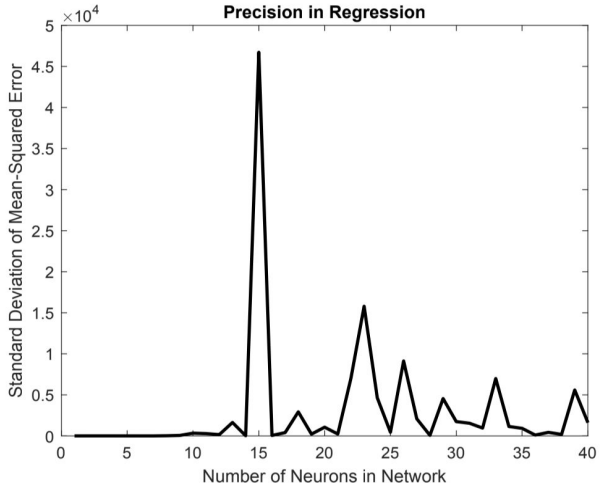
Nh = 10 neurons

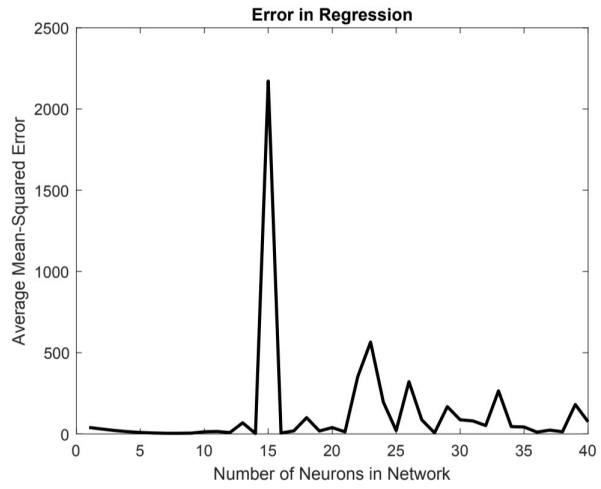


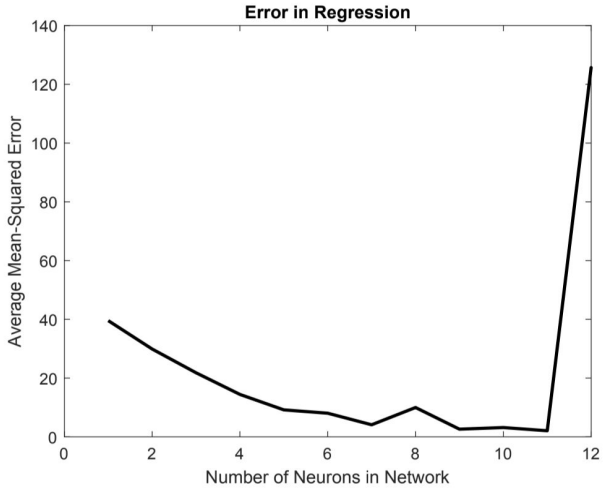
Nh = 26 neurons



b. Train networks of different sizes in the range Nh = [1 40]. This is the overdetermined regime where you have more training samples than parameters in your model. Plot the average mean squared test error as a function of Nh. You'll want to average over many repetitions at each size (≥ 500 repetitions). What would you say is the approximate best size model? How do things go wrong when the model is too small or too big?



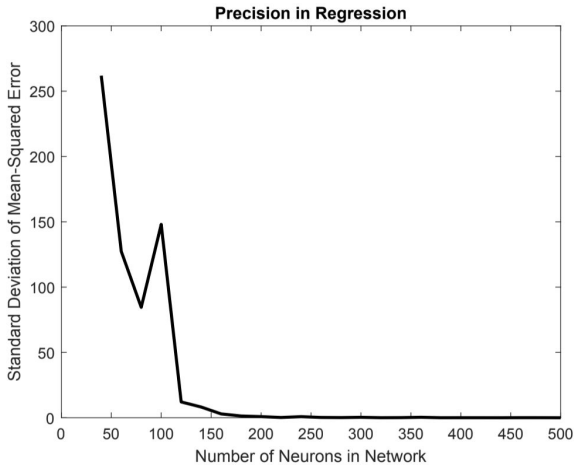


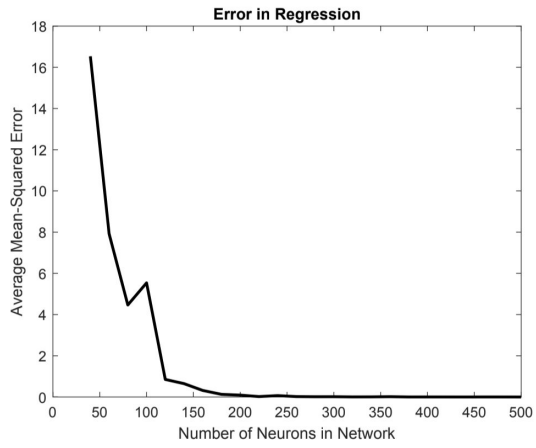


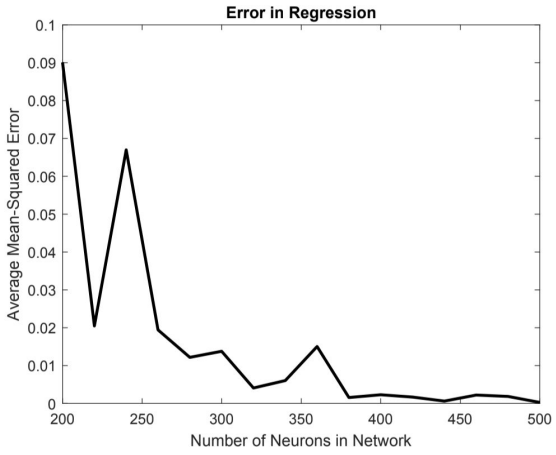
The approximate best size model for the network would be around 10 neurons, where the mean-squared error is closest to 0 in our trials. After around 15 neurons are in the network, you start to see dramatic spikes in the average mean-squared error and the standard deviation of the mean-squared error, indicating an increased regression error. We additionally include the plot of the standard deviation for the mean-squared error to demonstrate that the large spikes in the average mean-squared error are due to increased volatility in the trials when n is larger than 15, rather than similar, large errors between trials. For models that are too small (below ~10 neurons), there is also an increase in average mean-squared error, perhaps because these models are too simple to sufficiently capture the form of the underlying function.

c. Now throw caution to the wind and train some massive networks in the range nh = [40 500]. This is the underdetermined regime, where you have fewer training samples than parameters in your model. In underdetermined models, there is an infinite space of solutions that will fit the training points exactly. By using the pseudoinverse in the linear

regression equation, we are picking the solution with smallest norm. Plot the average mean squared test error as a function of Nh. What would you say is the approximate best model size? How variable is the performance of the really big networks compared to smaller networks?



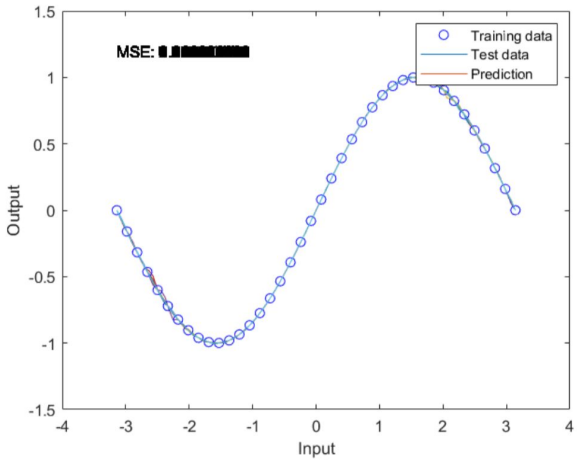




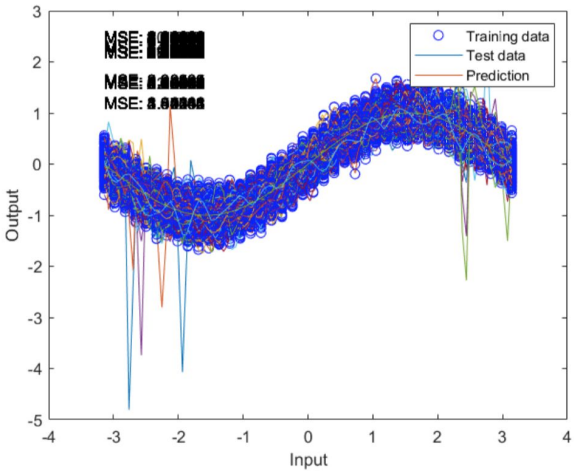
The approximate best model size would be anything larger than 200 neurons, and ideally larger than 400 neurons, depending on computational constraints and desired precision. Really large networks have a much smaller average mean-squared error and standard deviation of mean-squared error in comparison to smaller networks, and thus perform better (more accurately). Thus, really large networks have lower variability in performance compared to smaller networks, at the expense of computational efficiency.

d. Investigate a model with Nh = 500, but set the label noise ε = 0.2. What is the typical MSE with and without label noise? Do the predictions with label noise look good? Now adjust the regularization parameter $\lambda \approx 1e{-}4$. How does this change the MSE and the predictions with and without noise?

MSE with no noise



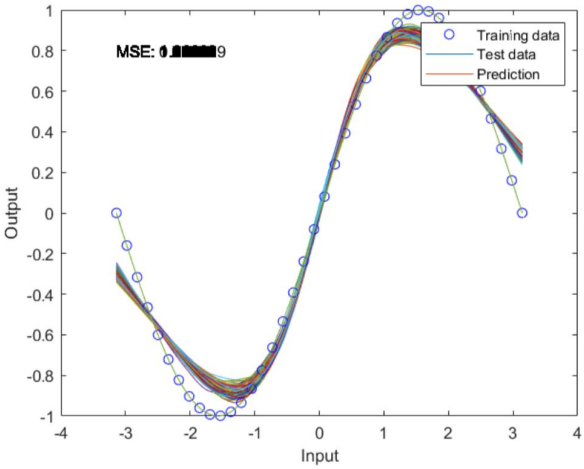
MSE with label noise ε = 0.2



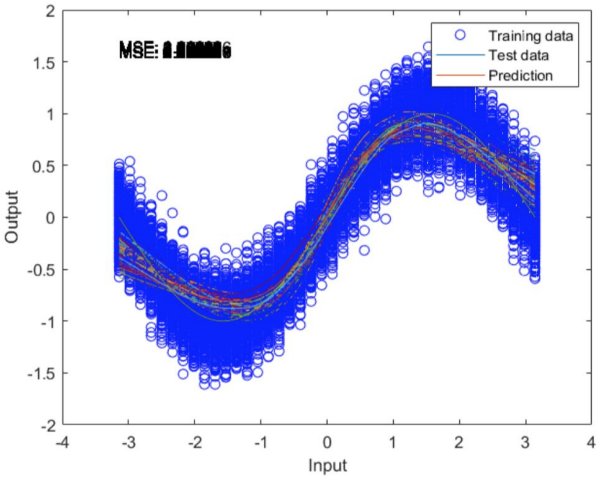
With no noise, the prediction is aligned with the test data and the training data. With the label noise parameter set to ε = 0.2, there is more variability between the oscillating prediction and test data.

With regularization parameter $\lambda \approx 1e{-}4$

MSE with no noise



MSE with noise ε = 0.2



While adjusting the regularization parameter to $\lambda \approx 1e{-}4$ in a system with no noise decreases the accuracy of the prediction, adding the parameter to a system with noise promotes considering only the most important weights, thereby reducing the influence of the noise on the prediction.

e. Based on your results, does it make sense for neural networks to massively expand the dimension of their inputs? What regime (in terms of label noise and number of neurons

relative to amount of training data) would you argue is the relevant regime for the learning problems faced by brain?

Yes. As seen in part c, larger model dimensions increase accuracy of stimulus reconstruction. In context of the brain, more aspects of the stimulus are represented by a higher dimensional network and available for interpretation. We expect that the number of neurons relative to amount of training data relevant to the brain is likely comparable to the 400 neurons used in part c) to give an accurate low-error reconstruction of the stimulus, as this seems to be the minimum regime required to achieve the accuracy observed in cognitive processes. This is based on our assumption that the brain is extremely effective at reconstructing stimuli based on a discrete set of observations (i.e., with a mean-squared error of 0.01 or less). We also expect that the brain encounters a significant level of noise with each signal, and that this would correspond to a relatively high magnitude of label noise, such as the $\varepsilon = 0.2$ parameter used above in the case of the sin function "signal."

3. **Image demixing: visual cocktail party problem**
   a. Covariance matrix and PCA:

Below are the inline functions written for mean subtraction and covariance calculations, as well as the eigenvalue/vector generation of the covariance matrix.

```
load toWhiten.mat

m=inline('A-mean(A,1)');                                    %inline function for mean subtraction
covar=inline('(1/(size(A,1)-1))*transpose(A)*(A)');           %inline function for covariance calculation

mean_sub=m(toWhiten);
co=covar(mean_sub);
[V,D]=eig(co);                                              %find eigenvalues and eigenvectors of covariance matrix
```

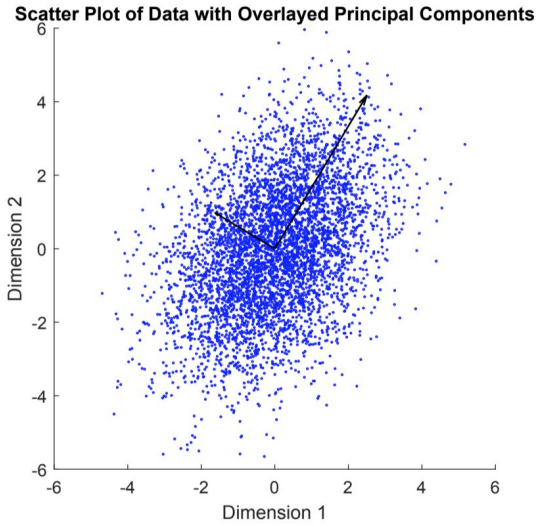
i. Write an inline function that removes the mean from a data matrix. Check by checking the mean of the output.

Using the inline function constructed for mean-subtraction yields final column means on the order of 10^{-14} (essentially zero, as expected).

ii. Without using MATLAB's built-in covariance function, write a function that calculates the covariance matrix of the data. Check against the built-in function. Tips: 1) This can be done with an inline function as well. 2) Remember that you need to divide by n − 1 to bias correct the covariance estimates.

The difference between Matlab's covariance function and the user-made covariance function on the mean-subtracted data is on the order of 10^{-15} (essentially zero, as expected).

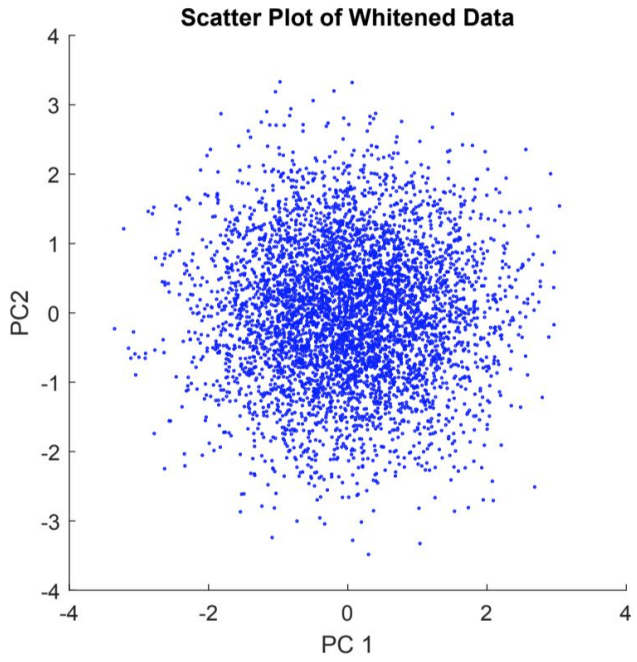
iii. Use MATLAB's built-in eig function to determine the eigenvalues and eigenvectors of the covariance matrix. Overlay the principle components of the data on a scatter plot of the data.



iv. Look at the dimension of the covariance matrix. Comment on conditions (values of D and N) for which the diagonalization will be time consuming. Compare the time required for singular value decomposition (MATLAB's svd function) with the time required for calculation and diagonalization of the covariance matrix by running both on random 1000 × 5000 and 5000 × 1000 matrices.

We predict that diagonalization will be most time consuming for cases where the number of dimensions of the data, D, is largest, as the input to the eigenvalue decomposition function is the D x D covariance matrix. We tested this prediction by finding average values over ten trials for four different conditions: singular value decomposition (SVD) for a random 1000 x 5000 (N x D) matrix, diagonalization of the same matrix, SVD of a random 5000 x 1000 matrix, and diagonalization of the same matrix. This analysis yielded average run times of 0.238 s, 36.9 s, 0.218 s, and 0.565 s, respectively, consistent with our expectation that a larger dimensionality of inputs results in significantly increased diagonalization time. This also highlights the utility of SVD, as this decomposition did not experience the same dependence on dimensionality.

b. Whitening the data: Check by plotting a scatter plot of the toWhiten data after whitening. Given how you constructed your whitened data, how can you extract the first principle components from the whitened data? (don't over think it!)



The data is centered at 0 and symmetrical, which indicates that the data is whitened. The code for this whitening can be found in Whiten.m

The first principal components can be extracted from the whitened data (Y) by left multiplying the pseudoinverse of the initial data matrix (X) with the whitened data matrix, i.e. by reversing the transformation performed during the whitening, Y=XW. This would yield a matrix of weights (W) in which the columns represent the eigenvectors of the covariance matrix of X divided by the square roots of their corresponding eigenvalues. As the eigenvectors used in the whiten process are normalized (i.e. have Euclidean norm equal to 1), taking the norm of each column of the weight matrix yields a value equal to $1/\lambda^{1/2}$, from which each eigenvalue (λ) and their corresponding eigenvectors may be extracted.

c. ICA:

ICA was performed by whitening the data using the code in Whiten.m, feeding this result into the provided learnWeights.m code, and multiplying the whitened data matrix by the resulting weight matrix.

i. Plot the mixed images.

Figure 1: **Figure 2:** **Figure 3:**

ii. Whiten the images using you function from part (a). Plot the whitened images.

Figure 1: **Figure 2:** **Figure 3:**

iii. The function learnWeights will learn a transformation (applied to the right of the data matrix in this case) that maximize the kurtosis of the extracted components. Natural images have non-Gaussian pixel statistics, so this transformation will find images that are as far from the mixture as possible. Run this function on the whitened images, apply the transformation to the whitened images, and plot the resulting output. Whitening prior to learning makes learning easier; try learning without whitening first to see what happens.

"Unmixed" images that result from the transformation function applied to the data WITHOUT whitening.

Figure 1: **Figure 2:** **Figure 3:**

“Unmixed” images that result from the transformation function applied to the data WITH whitening.

Figure 1:

Figure 2:

Figure 3:

Whitening the data prior to learning makes the transformation much more accurate and allows for better separation of the independent components (i.e., the three unmixed images).

Appendix:

Average_Firing.m

```
t=0:1/600:1/6;
std=5/1000;
fr=zeros(length(stimulus_start_times)-1,length(t));
normdist=0;
fr_avg=zeros(length(t),1);
for i=1:length(stimulus_start_times)-1
    times=st{i};
    n=length(times);
        for j=1:n
            normdist=normdist+normpdf(t,times(j),std);
        end
        fr(i,:)=normdist;
        normdist=0;
end

for i=1:length(t)
    fr_avg(i)=sum(fr(:,i))/(length(stimulus_start_times)-1);
end
plot(t,fr_avg,'color','k','linewidth',2);
xlabel('time (s)');
ylabel('Firing Rate (Hz)');
title('Average Firing Rate vs. Stimulus');
```

```
%time increment vector
%standard deviation for Gaussian filter
%firing rate for each stimulus presentation

%average firing rate for each time point
%over all stimulus presentations
%vector of spike times associated with given presentation

%for each spike
%apply normal distribution evaluated at all time points centered at spike

%firing rate is sum of all distributions from all spikes




%calculate average firing rate at each time point
```

ElapsedTime.m

```
t_el=zeros(n,4);

for i=1:n

r=rand(1000,5000);
r2=rand(5000,1000);
tic;

S=svd(r);
t_el(i,1)=toc;
tic;

r_meansub=r-mean(r,1);
cov_r=(1/size(r,1))*transpose(r)*r;
[Vr,Dr]=eig(cov_r);
t_el(i,2)=toc;
tic;

S2=svd(r2);
t_el(i,3)=toc;
tic;

r2_meansub=r2-mean(r2,1);

cov_r2=(1/size(r2,1))*transpose(r2)*r2;
[Vr2,Dr2]=eig(cov_r2);
t_el(i,4)=toc;

end

t_elapsed=(1/n)*sum(t_el,1);
```

```
%matrix containing rows of elapsed time for each operation for 10 trials



%script for testing speed of SVD and diagonalization
%generate random matrices
%mark down current time

%perform SVD and calculate elapsed time



%perform diagonalization and calculate elapsed time




%repeat with second matrix
```

Nn_regression_2.m

```
function[mse,w] = nn_regression(Nh)
```

```
%make function that takes in number of neurons in network
%returns mean-squared error and weight vector of synaptic strengths
```

```
%% Set up parameters
N = 40; % Number of training samples
epsilon = 0.0; % Amount of label noise
%%Nh = 20;
lambda = 0;
```

```
%% Make dataset
target_fn = @(t) sin(t);
x = linspace(-pi,pi,N);
y = target_fn(x) + epsilon*randn(size(x));

Ntest = 100;
x_test = linspace(-pi,pi,Ntest);
y_test = target_fn(x_test);

Ni = 2;
```

```
%% Compute network activity

J = randn(Nh,Ni)/Nh;

h = J*[x; ones(1,N)];
h(h<0)=0;

h_test = J*[x_test; ones(1,Ntest)];
h_test(h_test<0)=0;
```

```
%% Now train linear regression to map from h to y

w = y*h'*pinv(h*h'+lambda*eye(length(h*h')));

y_pred = w*h_test;

mean_squared_error = norm(y_test-y_pred).^2;
mse=mean_squared_error;
```

PCA.m

```
m=inline('A-mean(A,1)');                                    %inline function for mean subtraction
covar=inline('(1/(size(A,1)))*transpose(A)*(A)');           %inline function for covariance calculation

mean_sub=m(toWhiten);
co=covar(mean_sub);
[V,D]=eig(co);                                              %find eigenvalues and eigenvectors of covariance matrix

scatter(mean_sub(:,1),mean_sub(:,2),1,'b');                 %plot data
hold on;
quiver(0,0,1.5*D(1,1)*V(1,1),1.5*D(1,1)*V(2,1),'color','k','LineWidth',1);  %plot principal components
quiver(0,0,1.5*D(2,2)*V(1,2),1.5*D(2,2)*V(2,2),'color','k','LineWidth',1);
pbaspect([1 1 1]);                                          %set aspect ratio for x and y axes 1:1
xlabel('Dimension 1');
ylabel('Dimension 2');
title('Scatter Plot of Data with Overlayed Principal Components');
```

PSTH2.m

```
ctrl=(spikes_control);
exp=(spikes_exp);
width=5/1000;                                               %histogram bin width
tstim=[0 1/6];
numbin=ceil((tstim(2)-tstim(1))/width);                     %total bin number

j=1;

while j<=length(ctrl)                                       %for each spike in control dataset
    if ctrl(j)>=1/6                                         %if spike time is greater than length of stimulus
       ctrl(j)=ctrl(j)-1/6;                                 %subtract stimulus length until within 1 stimulus
    else
        j=j+1;                                              %go to next spike
    end
end

j=1;

while j<=length(exp)                                        %repeat for experimental dataset
    if exp(j)>=1/6
       exp(j)=exp(j)-1/6;
    else
        j=j+1;
    end
end

ctrl=sort(ctrl);                                            %sort data vectors in ascending order
exp=sort(exp);

histogram(ctrl,'BinWidth',width,'Normalization','pdf');     %use matlab histogram function with normalization
xlabel('time (s)');                                         %by total number of points and bin width
ylabel('Frequency (Hz)');                                   %to get firing rate distribution for both control and experiment
title('Control Firing');
figure;
histogram(exp,'BinWidth',width,'Normalization','pdf');
xlabel('time (s)');
ylabel('Frequency (Hz)');
title('Experimental Firing');
```

Raster.m

```
spikes_single_unit=sort(spikes_single_unit);                                %make sure spikes are sorted in ascending order
if length(stimulus_start_times)==360
    stimulus_start_times=[stimulus_start_times 360/6];
end
i=1;
j=1;
st=cell(length(stimulus_start_times),1);                                    %create cell to store data for each stimulus
st=st';
while j<=length(spikes_single_unit)                                         %for each spike
    if spikes_single_unit(j)<stimulus_start_times(i)                        %if the spike is before the current start time
        line([spikes_single_unit(j)-stimulus_start_times(i-1) spikes_single_unit(j)-stimulus_start_times(i-1)],[i-1 i+1],'color','k');
        st{i}=[st{i} spikes_single_unit(j)-stimulus_start_times(i-1)];      %draw a line, update array for stimulus i with spike onset
        j=j+1;                                                              %go to next spike
    else
        i=i+1;                                                              %if not, all spikes up until current time have been included
    end                                                                     %go to next time point
end
ylim([0 370]);
xlabel('time (s)');
ylabel('Trial Number');
title('Raster Plot of Spiking Activity');
```

Regression.loop.m

```
n=1000;                                                                     %script for looping through neural net regression
test=1:1:40;                                                                %vector of Nh to test

mse=zeros(length(test),length(n));                                          %matrix of returned mse for each test condition and trial
w=cell(length(test),1);                                                     %cell for each test condition with each matrix containing
                                                                            %values for each w over all trials
for i=1:length(test)
    for j=1:n
        [mse(i,j),w{i}(j,:)]=nn_regression_2(test(i));
    end
end

mse_avg=mean(mse,2);                                                        %calculate average mse and standard deviation of mse
mse_std=std(mse,0,2);
plot(test,mse_avg);
figure;
plot(test,mse_std);
```

Sta_estimation_2.m

```
clear all

% Load DMR stimulus specrogram and spiking responses from one neuron
load dmr_experiment

% Plot spectrogram of stimulus
plot_spectrogram(stim_spectrogram, stim_time, stim_freq)
```

```
%% Generate STA
t_past = 125; % in ms
t_future = 125; % in ms
sampling_rate = mean(median(diff(stim_time)));
sta_time = (-t_past/1000):sampling_rate:(t_future/1000);
sta_freq = stim_freq;
sta_stim = zeros(length(sta_freq),length(sta_time),length(spikes));         %3D array with each 2D matrix the frequencies at time points in window

for k=1:length(spikes)                                                      %over all spikes
    t=spikes(k)+sta_time(1);                                                %find first time point in window (tspike-125 msec)
    tdiff=abs(stim_time-t);                                                 %evaluate difference between all time points and first point in window
    [M,I]=min(tdiff);                                                       %find index of closest value to include in stimulus window
    sta_stim(:,:,k)=stim_spectrogram(:,I:(I+length(sta_time)-1));           %extract stimulus window from -125 to 125 msec around spike k
end
sta = sum(sta_stim,3)/length(spikes);                                       %average 2D window over all spikes

% Plot results
figure(2)
plot_spectrogram(sta, sta_time, sta_freq);
xlabel('Time relative to spike (ms)')
ylabel('Frequency (kHz)');
title('Spike-Triggered Average');
colorbar

figure;
corr=(1/size(stim_spectrogram,2))*stim_spectrogram*stim_spectrogram';       %calculate covariance of spectrogram
heatmap(corr,'Colormap',gray,'Xlabel','','YLabel','');                      %plot on gray heatmap
title('Heatmap of Spectrogram Covariance');
```

Whiten.m

```
function[Whitened] = Whiten(toWhiten)                                   %returns whitened dataset

m=inline('A-mean(A,1)');
covar=inline('(1/(size(A,1)))*transpose(A)*(A)');

mean_sub=m(toWhiten);                                                   %mean subtract data and find covariance matrix
co=covar(mean_sub);
[Vunsort,D]=eig(co);                                                    %find eigenvalues and eigenvectors
dia=diag(D);                                                            %extract eigenvalues from diagonal matrix for sorting
[val,I]=sort(dia,'descend');                                            %sort in descending order of eigenvalues
D=diag(val);                                                            %construct new diagonal matrix
V=zeros(length(D));
for i=1:length(I)                                                       %construct new eigenvector matrix corresponding to eigenvalues
    V(:,i)=Vunsort(:,I(i));
end

Proj=mean_sub*V;                                                        %project mean-subtracted data into principal component space
for i=1:length(D)                                                       %scale each dimension by 1/sqrt(eigenvalue)
    Proj(:,i)=Proj(:,i)/sqrt(D(i,i));
end
Whitened=Proj;
scatter(Proj(:,1),Proj(:,2),1,'b');                                     %plot whitened matrix in first two PCs
pbaspect([1 1 1]);
xlabel('PC 1');
ylabel('PC2');
title('Scatter Plot of Whitened Data');
```