Lab 7, 8, and 9 Report

Introduction

Supervised algorithms are an effective way to classify and decode data to make predictions on new information. With these algorithms, neural signals can be taken to anticipate how hands move, if someone is having a seizure, or what letter someone wants to type. Classifiers deliver fast bit rates and categorize data into several categories. Continuous decoders normally use a variant of linear regression (LR) to predict a wide range of values. In this lab, the classifiers Naive Bayes and linear discriminant analysis are implemented to use neural firing rates to predict hand direction or grip for a monkey. Additionally, LR, ridge regression, LASSO, and Kalman filters are continuous decoders used to predict the x and y position, and x and y velocity using training data.

Methods

The first step in this lab consisted of implementing the Naive Bayes algorithm. The "firingrate.mat" data given in class is used which consists of firing rates from 95 neurons in 182 trials for 8 different reaching positions for a monkey. In Naive Bayes there is a probabilistic model of the distributions that the data is being drawn from and that it is only able to organize data into a set number of categories. Naive Bayes algorithm works by utilizing Bayes Rules, Equation 1. Equation 2, which describes how the Naive Bayes classifier works, looks for the maximum likelihood target given the spike data. A poisson distribution was assumed which presumes a distribution of integer counts oriented in bell curves with a greater variance for higher means. This is a fair assumption as neurons usually act this way. Naive Bayes also assumes that the different features of sample x are independent from one another for a given class. This is usually not true but has been shown to not have a great effect on decoder performance. To implement the algorithm the firingrate.mat data was imported into MATLAB and split into training and testing data. All algorithms in this lab used MATLAB. The training data consisted of the first 91 trials including all of the neurons and possible directions while the testing data consisted of the last 91 trials. The classifier was then trained on the training data. The mean for each feature-class (neuron-direction) pair of the training data was recorded. A doubly nested for loop was then created going through the 91 trails and the 8 targets. The feature was saved for the neurons in each iteration for the trail and target of the testing data. For each feature the probability density of observing that feature in that direction was found. This was done by going through 8 target guesses and 95 neurons and using Equation 3 to find the probability of each guess and picking the maximum probability. Normally the probabilities are multiplied like in Equation 2, however since they can be very small numbers, the log of the probabilities was taken and summed to avoid floating point errors. The poisspdf() function was also utilized to find the poisson density given the feature of the neuron and mean for each neuron at its target guess. The accuracy of the predictions was evaluated by seeing which predictions were correct (target 1 equaled 1, target 2 equaled 2 and so on). The accuracy was determined as the number of correct predictions divided by the total number of predictions. To ensure that the algorithm was working correctly, a spoofed data set was created and tested with. This was done

by taking the mean of firingrate.mat across the number of trials and across the 8 predictions. An empty matrix of 95x128x8 values was created and the poissrnd() function was used to generate random numbers from the poisson distribution and neuron mean for all the trials and targets of each neuron. The confusion matrix was run through the algorithm and the accuracy recorded.

The next step looked at implementing a linear discriminant analysis algorithm. Linear discriminant analysis (LDA) conceptually makes a hyperplane separating two clusters of data. LDA is a classifying algorithm meaning it can only estimate into a discrete number of classes. Equation 4 is what is used to determine what class the data belongs to and Equation 5 determines centroids. If the data is independent, the centroids can be subtracted to find their relationship. Linear discriminants put all the data together and get one covariance matrix. This assumes that the data is legitimately correlated and makes the decision boundary a line. LDA assumes that the data is normally distributed, linearity, independence, homoscedasticity, and each of the classes is assumed to have identical covariance matrices. The data from "ecogclassifydata.mat" given in class was loaded into MATLAB which contained average power values from 0.5 sec prior to movement to 1.5 sec after movement in 60 -120 Hz bands for 38 trails with data recorded from 27 electrode pairs. The group variable indicates which finger was squeezing. The classify function was used and given the test data, training data, training classes and 'linear' parameter to ensure a linear discriminant was used. The test data and training data were changed so that each trial was able to be the test data while the rest was the training data. The correct predictions divided by the number of predictions was done to calculate the accuracy. To determine where the errors occurred Figure 1 was made to show what percentage of the time a finger is classified as itself and what percentage of time it was classified as another finger.

A LR algorithm was then implemented. The basic equation for LR is given by equation 6. LR is a model that assumes a linear relationship between the input and output variables. LR also assumes that the variance of the errors is constant for all independent variables, there is little or no multicollinearity, and the observations are independent. In this lab, testing data is used to find a decoder, B, that can be multiplied by the input data Y, to find the predicted output variable X. The objective function for LR is given by Equation 7. The data used was from "contdata95.mat" given in class and has an X variable where there are columns of X position, Y position, X velocity, and Y velocity at 31413 time points. There is also a Y variable that has firing rates for 95 recorded units. A delay was added to the data to result in 950 total features by adding duplicated columns for each column that were rotated down incrementally with 0s replacing numbers lost at the top. Inserting this time lag is only helpful for "offline" testing (not testing done on a person live) because it will cause a person's previous firing rate to affect their current action. The data was then split into training data containing the first 15707 time points and testing data containing the time points form 15708 until the end. Equation 8 was used with the training data to find the B matrix. Equation 6 was then used with the testing data and B matrix to find the predicted values. Figures 2 and 3, show the plots of the predicted X and Y positions with the real values. The mean squared error (MSE) for each position was calculated by taking the

sum of the squared difference of the means for the predicted and actual. The corr2() function was also used to find the correlation of the predicted values.

LR was extended to use ridge regression. Ridge regression is very similar to LR but it introduces a regularization term that should counter overfitting. Ridge regression has similar assumptions as LR: linearity, constant variances, and independence. The objective function is now described by Equation 9, and Equation 10 describes how to find the B term with ridge regression. The same data with the time lag that was in the LR was used for this part and split up the same way. 15 lambda values equally spaced from 0 to 0.1 were evaluated to see the effects on correlation and MSE. Each lambda value was looped through and the B matrix was calculated using Equation 14 with an N of 15706, training data, and an identity matrix with sides of 951. The MSE and correlation for each position was then calculated like in LR. The MSE and correlation were plotted vs. lambda in Figures 4 & 5 and the lambda value that gave the most accurate results was determined.

A modified version of ridge regression was used called least absolute shrinkage and selection operator (LASSO). LASSO does variable selection and regularization in an attempt to make more accurate predictions. It uses the L_1 norm instead of the L_2 norm. Equation 11 describes the objective function for LASSO. The assumptions for LASSO are the same as ridge regression: linearity, constant variances, and independence. The same lagged time data used for linear and ridge regression was used for this part. The first 15707 data points were set as the training data, the second half were used as testing data, and the best lambda value found in the ridge regression was used. MATLAB's lasso() function was used with the training data and the best-found lambda value to get the B matrix for each prediction (X position, Y position, X velocity, Y velocity). The MSE was found using the immse() function and the correlation was found using the corr2() function.

A Kalman filter was used to incorporate neural data, physics data, and some noise to predict motion. The model assumes that the current state is a linear function of the previous state and the noise is normally distributed. The same data with the time lag that was used in the LR was used for this part and split the same way. Xt was created with transposing all but the last element in the first half of the positions and velocities of the training data and x_{t-1} was made by taking everything but the first term for the positions and velocities of the training data. Then the A matrix, the relationship between the previous x and next, was calculated with equation 12. C, the relationship between the neural data and x data, was calculated with equation 13 with the training data. W, the noise term with physics and Q the noise term for the neural data were created in equations 14 and 15. A loop through all time points of the testing data was done. In each iteration Equation 16 was used to update the posterior state estimate of x at time t, Equation 17 to update the posterior error covariance of x, Equation 18 to update the Kalman gain (best proportion of the neural guess to include compared to the best physics guess), Equation 19 to update x, and Equation 20 to update the error covariance. The MSE and correlation coefficients were calculated for each parameter and the predictions were plotted in Figures X and Y along with the real variable.

Results

In the first part of this lab a Naive Bayes algorithm was used to classify what direction a monkey would put his hand in using training data and neural firing rates. This resulted in a 97.39% accuracy when the model was used on the testing data. This shows that the model was working well classifying almost all of the testing data correctly. To ensure that the model is working properly, it was also tested on a spoof data set that randomly generated numbers for each neuron based on a poisson distribution using the mean of all the trails and directions for each neuron. This resulted in a 12.43% accuracy rate which makes sense as there are 8 possible classifications random data should result in a ⅛ accuracy. Naive Bayes is a fast program that is easy to implement and as shown can predict data really well sometimes. However, one disadvantage is that it assumes that the features are independent which works in this case but may not always work. Additionally, if it sees a categorical variable that was not shown in the dataset it assigns it a zero probability. Lastly, since it is a classifier, it can only categorize the data into a set number of groups and won't work as well for continuous data as other methods.

LDA was also used to determine which finger was squeezing given power values from electrode pairs. The accuracy of the predictions was found to be 71.79%. This is less accurate than the Naive Bayes algorithm. A confusion matrix was also created in Figure 1 that showed the percentage of time a finger is classified as itself and what percentage it was classified as another finger. Figure 1 shows that group 3 had the lowest percentage of classifying itself only doing so about 40% of the time. Group 1 had the best percentage of classifying itself doing so 100% of the time. Groups 2,4, and 5 classify themselves about 85% of the time. LDA is a quick and simple classifier although it has some limitations. LDA assumes a normal distribution, which may not always be the case. If the other assumptions mentioned above are not met it can decrease the accuracy of the predictor. LDA does not work great with small sample sizes. LDA is also a classifying algorithm so it has a set number of groups to categorize the data.

LR was implemented to predict the position and velocity of a hand based on training data and firing rates. Figure 2 shows the predicted X position and real X position of the hand over 250 ms and Figure 3 shows the same for Y position. In Figure 2 the line for predicted X value is pretty close to the real X values. The predicted X values are much more noisy especially in between movements it does not stay flat like the real values but whenever there is a spike the predicted values spike as well and to a similar magnitude. There are similar observations for Figure 3. The predicted Y values follow the real Y values pretty closely. There is also a lot more noise as it goes up and down when the arm is not reaching but the predicted values also line up with the spikes and to a similar magnitude.The MSE and the correlation coefficient of all the parameters are reported in Table 1. The MSE of the positions are about the same and the velocities are about the same but the positions are much lower. The predictions for position were on average closer to the real values than the predictions for velocities. Correlation says how related two variables are. The correlations for position are a little higher than the correlations for velocity meaning that the positions are more closely related to the actual values than the velocities. Even though the positions had better predictions than the velocity, the velocities were

still able to come close to the real values. LR is a great tool that is easy to implement to forecast continuous data and the output measurements are easy to interpret. However, LR assumes a linear relationship between dependent and independent variables and that there is independence. Outliers can also have large effects on the results and LR is prone to underfitting.

Ridge regression, an extension of LR, was implemented on the same data as before for the same purpose. Neuron firing rates were used to predict X and Y position and velocity. Different regularization terms were tested to see what would give the best prediction. Figure 4 and 5 shows the MSE and correlation respectively of different lambda values for the X position. Although the differences in MSE and correlation are small, we see that a lambda value of about .0286 gives the lowest MSE and highest correlation which means that it made the most accurate predictions. Using a lambda of 0.0286 the MSE and the correlation coefficient of all the parameters are reported in Table 2 . The regularization term reduces overfitting by penalizing the B matrix for having large numbers. Ridge regression and LR are very similar algorithms. They use almost identical equations but ridge regression introduces the regularization term to get better predictions. We see here that ridge regression gives slightly lower MSEs which means there is less error, and slightly higher correlations, meaning that the predictions are more correlated to the actual values, but not by a significant amount. There are similar limitations as LR such as assuming linearity and independence and having to find the best lambda.

Another extension of LR, LASSO, was used to calculate the same values on the same data as the previous two regression methods. Neuron firing rates were used to predict X and Y positions and velocities. LASSO incorporates regularization and variable selection to improve the accuracy of the LR. Lasso was done with the same best regularization term as the ridge regression, 0.0286, for each of the parameters. The MSE and the correlation coefficient of all the parameters are reported in Table 3. The MSEs are higher than in the linear and ridge regression however the correlation is also higher. This shows that LASSO might be better or worse depending on the use but it does provide a closer correlation to the actual data than in linear or ridge regression. The terms in the B matrix in LASSO often look smaller than the terms in the B matrix in ridge regression. This is likely from the regularization that is done in LASSO and may help to have the outliers not have a significant impact on the predictions. There are similar limitations as LR such as assuming linearity and independence and having to find the best lambda.

Finally, a Kalman filter was also implemented to use neuron firing rates to predict hand motion. Kalman filters use physics and neural data to estimate predictions about the motion. A cycle of “Predict”, “Innovate”, and “Update” were repeated that integrated a form of LR with noise components and the MSE and correlation coefficients were calculated. The MSE and the correlation coefficient of all the parameters are reported in Table 4. We can see that the MSEs are higher than most of the other algorithms and the correlations are lower than most of the other methods. This means that the Kalman filter performed worse in general than the other algorithms. This may be because it assumes there is a linear relationship between the previous and new parameters and the neural firing rate and parameters which is probably not true. It also

assumes the noise is Gaussian distributed. However, Kalman filters are still useful because they help maintain real movement, like making sure the predictions don't teleport.

Naive Bayes and LDA are the two classifying algorithms worked with in this lab. Both of the algorithms need data to train on, run fast, and are only able to categorize the data into a discrete number of categories. Naive Bayes works more with probabilities and assumes a poisson distribution of training data to find the most probable category on the training data. LDA tried to separate the data into clusters that could be used to predict where the test data would fit. In this case the Naive Bayes algorithm performed better than LDA but LDA had less data.

Several continuous decoders were used. Unlike the classifying algorithms, these algorithms can predicate much more than a set number of categories. LR was the first method implemented that assumed a linear relationship between the hand data and neuron firing rate and used this relationship found on training data to predict on test data. Ridge regression is almost the same but has a regularization term to prevent overfitting. LASSO was also used and is similar to LR but shrinks the points with variable selection and regularization. Kalman filters use LR, neural data, and physics data to try to make a more realistic prediction. LR is the basis for the other linear decoders in this report. MSE and correlations were used to calculate accuracy and some methods improve slightly on LR but LR does really well. For the X position LASSO had the highest correlation but ridge regression had the lowest MSE.

Discussion

Supervised algorithms are useful tools that use training data to predict information. There is a wide array where these applications can be used and in this lab neural firing rates were used to predict the movement of a monkey's hand. This can be useful for the people whose limbs are not working properly. The person's neural activity can be recorded and a prosthetic device can be set to move in the direction it wants. These algorithms could also record neural data and detect an episode is about to occur such as a seizure.

This lab started out with a Naive Bayes algorithm and LDA which both were used to classify the direction a monkey was going to move from 8 categories based on its neural data. Naive Bayes used Bayes rule with probabilities and LDA finds a line to split the data in distinct groups. Naive Bayes had a higher accuracy than LDA and these algorithms can only classify into a certain number of groups, but both are fast running and easy to implement. These were used to predict the direction of hand movement and hand orientation in this lab. These classifiers can be used in the future to categorize what key someone would want to press on a keyboard or if a person is at high risk of diseases, and even help identify facial features.

Then LR, ridge regression, LASSO, and Kalman filters were implemented as continuous decoders. Continuous decoders give more possible values the algorithm can predict such as here the X and Y position and velocity could be predicted. Each variation tries to make the method more accurate in different ways. The data learned from these algorithms can be used to help control prosthetic arms for someone without limbs. These algorithms can also be used in conjunction with visual prosthetics to track people's movements, and recording neural data in the auditory complex to interpret hearing.

Appendix

Tables

	X position	Y position	X Velocity	Y velocity
Mean Squared Error	129.3543	135.9860	3002.2	2935.4
Correlation	.9141	.8859	.8696	.8387

Table 1. Mean squared error and correlation coefficient of linear regression parameters.

	X position	Y position	X Velocity	Y velocity
Mean Squared Error	128.8856	135.7369	2992.8	2927.7
Correlation	.9144	.8860	.8699	.8391

Table 2. Mean squared error and correlation coefficient of ridge regression parameters with a best found lambda of .0284.

	X position	Y position	X Velocity	Y velocity
Mean Squared Error	138.1741	210.1580	3183.1	3056.9
Correlation	.9153	.8870	.8703	.8395

Table 3. Mean squared error and correlation coefficient of LASSO.

	X position	Y position	X Velocity	Y velocity
Mean Squared Error	171.7940	176.4908	3713.2	3647
Correlation	.8860	.8501	.8415	.7962

Table 4. Mean squared error and correlation coefficient of Kalman filter.

Figures

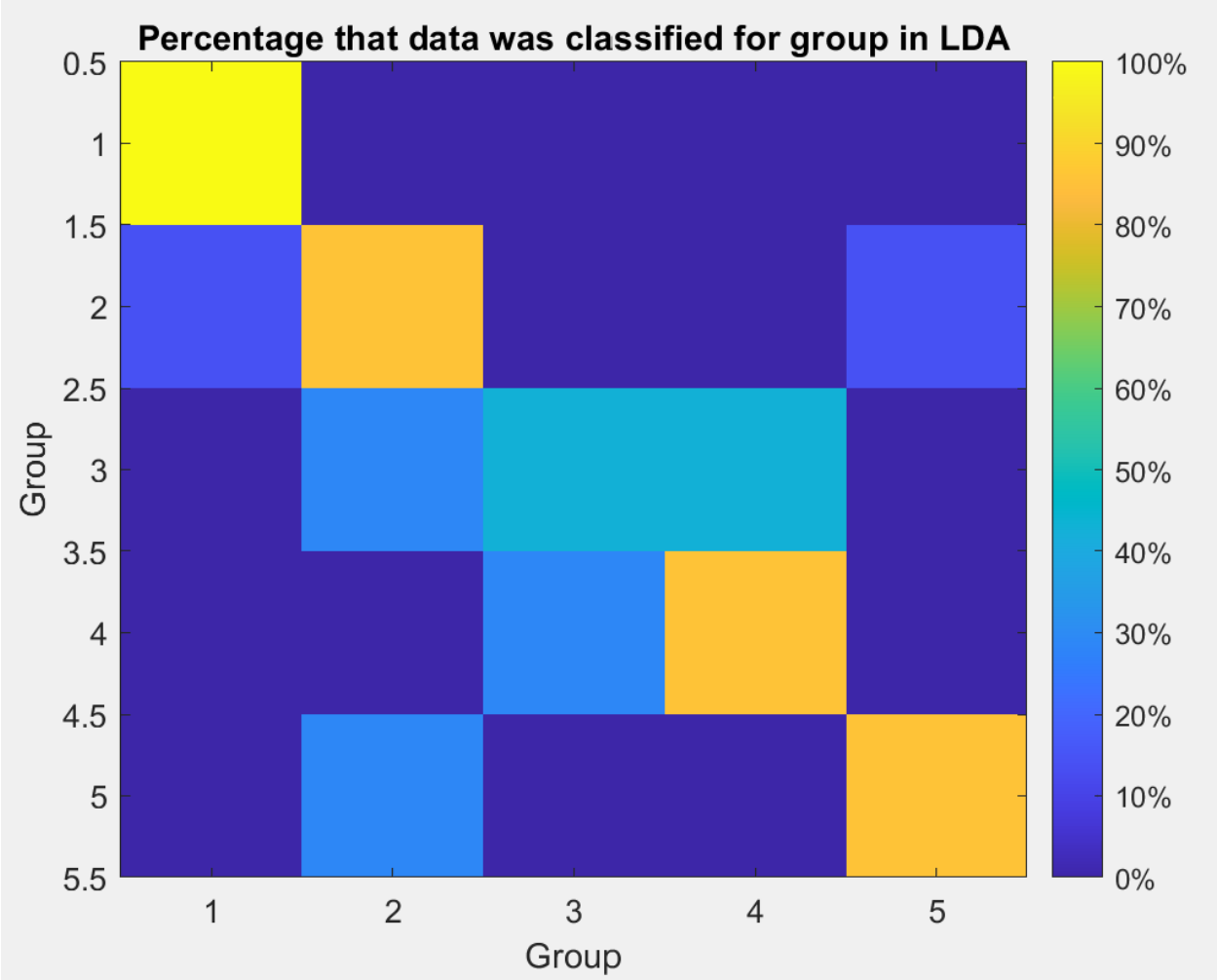

Figure 1. Percentage that data was classified for groups in LDA.

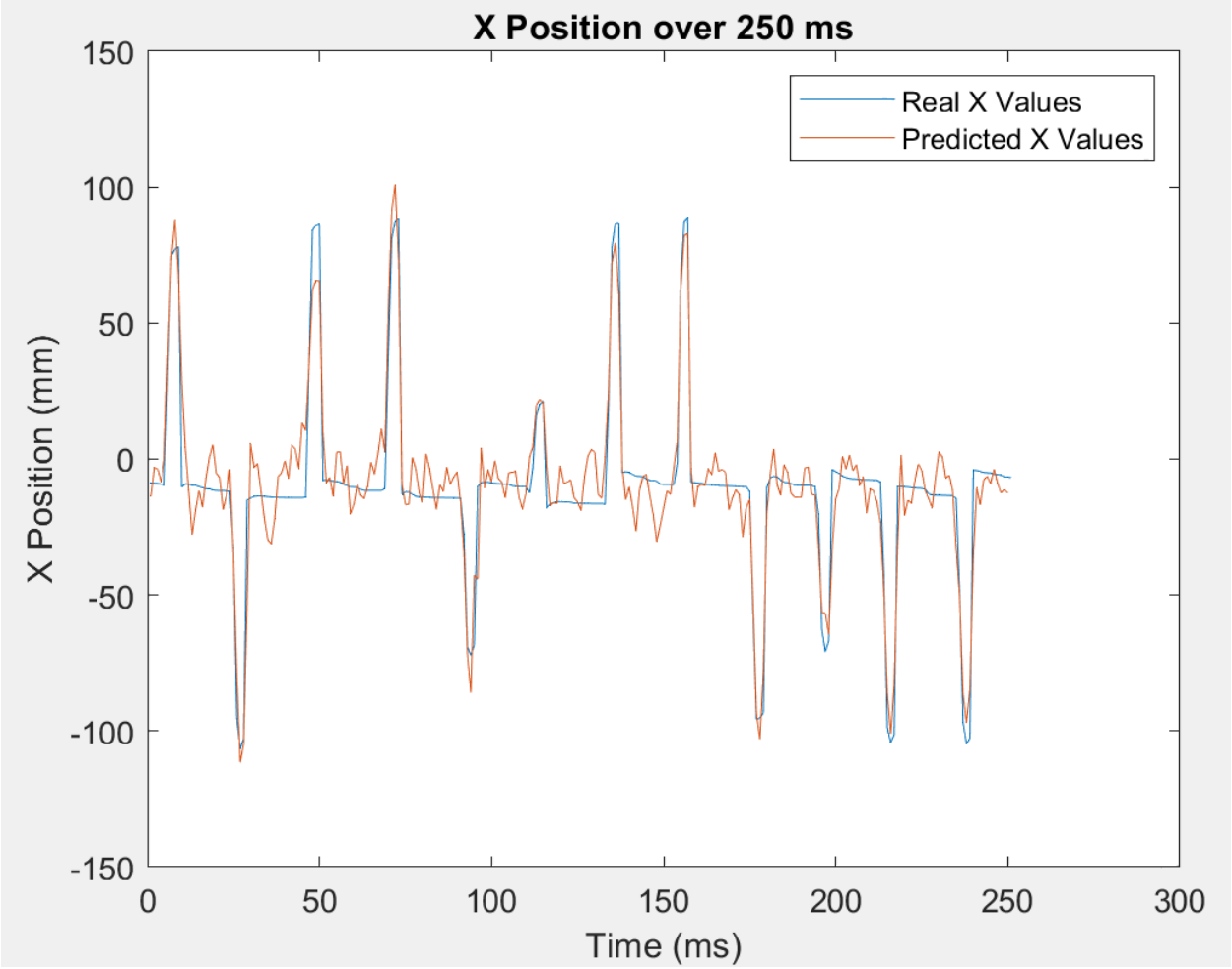

Figure 2. Predicted X position and real X position over 250 ms using linear regression.

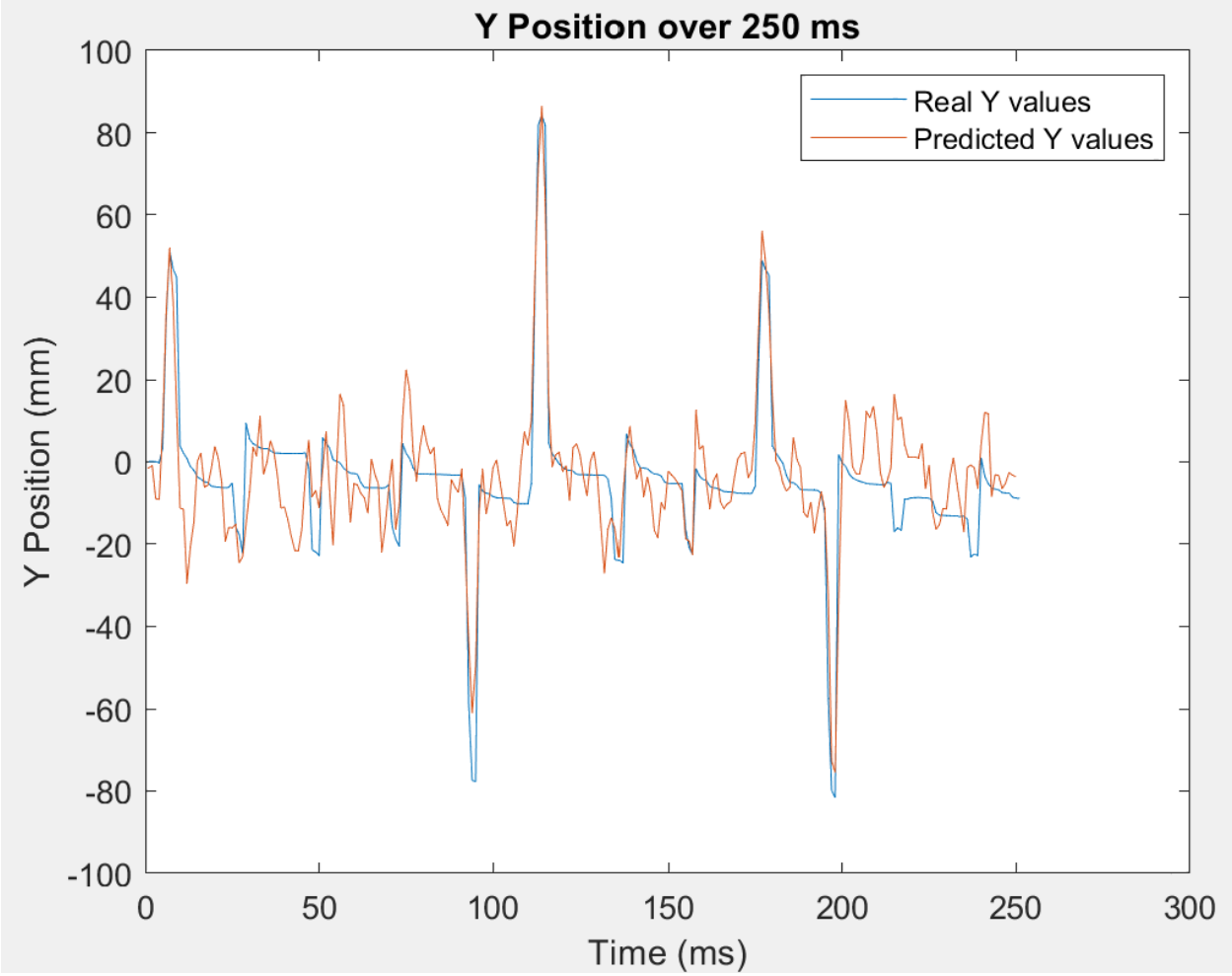

Figure 3. Predicted Y position and real Y position over 250 ms using linear regression.

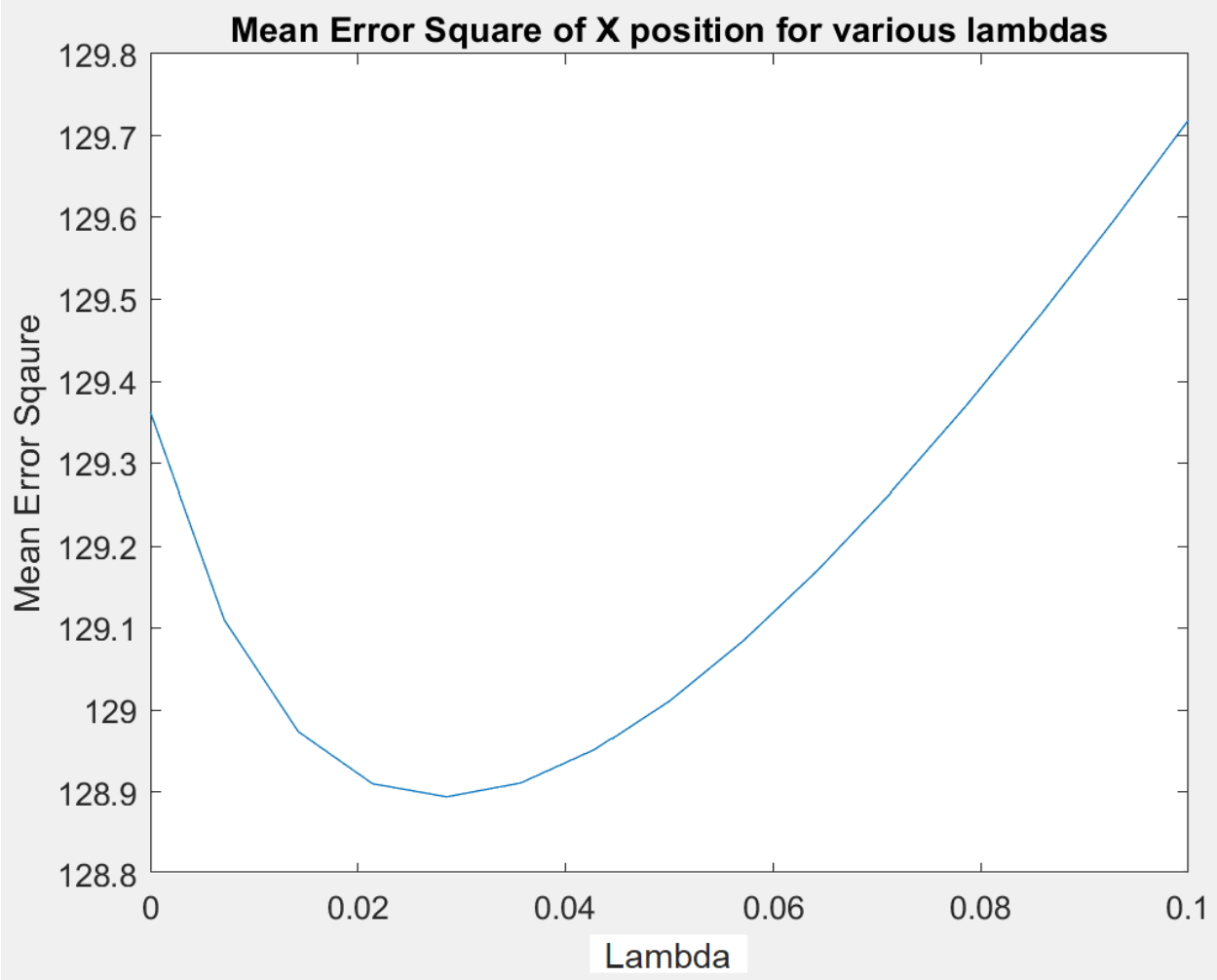

Figure 4. Sweep of mean square error for different lambdas in ridge regression.

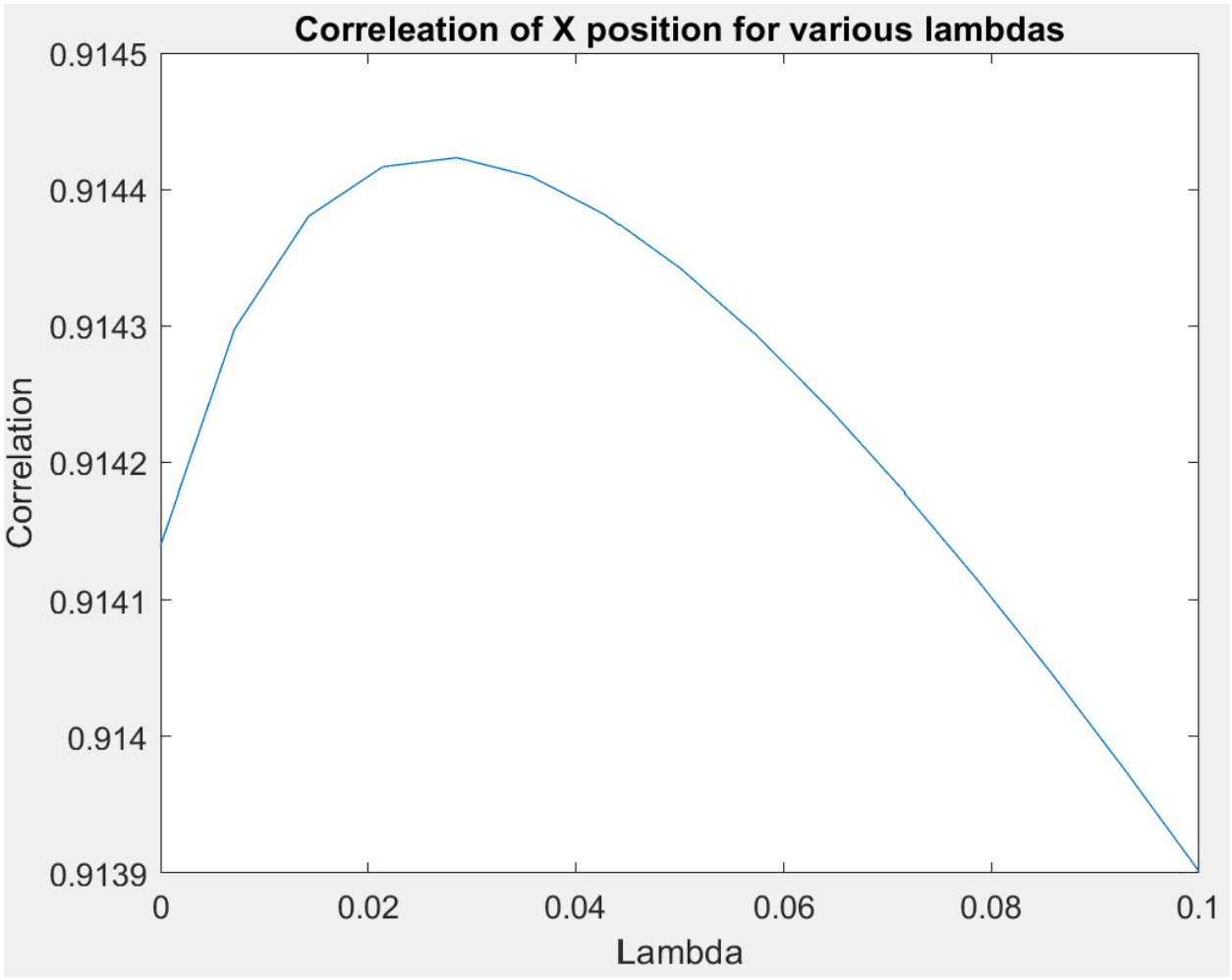

Figure 5. Sweep of correlations for different lambda values in ridge regression.

Equations

Equation 1

$$p(\vec{x} \mid \vec{y}) = \frac{p(\vec{y} \mid \vec{x})p(\vec{x})}{p(\vec{y})}$$

Equation 1. Bayes Rule used to calculate probability given prior information.

$$\hat{f}(x) = \arg\max_{k} \left[\prod_{j=1}^{D} \hat{g}_k^{(j)}(x) \right]$$

Equation 2. X is a vector of features, D is the number of features in x, and $\hat{g}_k^{(j)}$ is the estimated probability distribution of features j for class k.

$$\sum_{j=1}^{D} \log \hat{g}_{k=1}^{(j)}(X_i)$$

Equation 3. Total probability using logs because the numbers can be small.

$$y(\vec{x}) = \vec{w}^T \vec{x} + w_0$$

Equation. 4. Calculates discriminate in LDA. y(x) says what x belongs to, x is the data to be classified, w^T is what is fit during training, and w0 is the bias term to determine if y(x) is > or < 0.

$$\vec{m}_k = \frac{1}{N_k} \sum_{n \in k} x_n$$

Equation 5. Centroid of cluster of data.

$$\vec{x} = \vec{y}B$$

Equation 6. Linear regression equation.

$$\min_{\vec{w},b} \left[\frac{1}{n} \sum_{i=1}^{n} (\vec{w}^T \vec{x}_i + b - y_i)^2 \right]$$

Equation 7. Objective function for linear regression. x_i is a vector of features for observation i, y_i is the measured outcome of observation i, and w_i and b are estimated parameters of the linear model.

$$B = \left(\vec{y}^T \vec{y} \right)^{-1} \vec{y}^T \vec{x}$$

Equation 8. Equation to find B matrix with testing data for linear regression.

$$\min_{\vec{w},b}\left[\frac{1}{n}\sum_{i=1}^{n}(\vec{w}^T\vec{x}_i+b-y_i)^2+\lambda\left|\left|\vec{w}\right|\right|_2^2\right]$$

Equation 9. Objective function for linear regression.where $||w||_2$ is the L_2 norm of w.

$$\beta=(X^TX+N\lambda I)X^TY$$

Equation 10. Equation to find B matrix with testing data for ridge regression. Lambda is the regularization term, N is the total number of training data points, and I is the identity matrix.

$$\min_{\vec{w},b}\left[\frac{1}{n}\sum_{i=1}^{n}(\vec{w}^T\vec{x}_i+b-y_i)^2+\lambda\left|\left|\vec{w}\right|\right|_1^2\right]$$

Equation 11. Objective function for LASSO.

$$A=X_tX_{t-1}^T(X_{t-1}X_{t-1}^T)^{-1}$$

Equation 12. Kalman filter A equation. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

$$C=YX_t^T(X_tX_t^T)^{-1}$$

Equation 13. Kalman filter C equation. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

$$W = \frac{1}{\#\,pts - 1}(X_t - AX_{t-1})(X_t - AX_{t-1})^T$$

Equation 14. Kalman filter W equation. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

$$Q = \frac{1}{\#\,pts}(Y_t - CX_t)(Y_t - CX_t)^T$$

Equation 15. Kalman filter Q equation. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

$$\hat{x}_{t|t-1} = A\hat{x}_{t-1}$$

Equation 16. Kalman filter previous posterior state estimate of x at time t update (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008).

$$P_{t|t-1} = AP_tA^T + W$$

Equation 17. Kalman filter previous posteriori error covariance of x (Kalman, 1960, Wu, Black, 2006, Kim… Donoghue, 2008).

$$K_t = P_{t|t-1}C^T(CP_{t|t-1}C^T + Q)^{-1}$$

Equation 18. Kalman gain equation (Kalman, 1960, Wu, Black, 2006, Kim… Donoghue, 2008).

$$\hat{x}_t = \hat{x}_{t|t-1} + K_t(y_t - C\hat{x}_{t-1})$$

Equation 19. Kalman filter x update equation. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

$$P_t = (I - K_t C)P_{t|t-1}$$

Equation 20. Kalman filter error covariance equation update. (Kalmna, 1960, Wu, Black, 2006, Kim… Donoghue, 2008)

Code

```
Lab 7 Part 1
data = load("firingrate.mat");
load("firingrate.mat")
training = data.firingrate(:,1:91,:);
testing = data.firingrate(:,92:182,:);
%% Testing Training Data

meanFeatureClass = mean(training,2);
predicted = zeros(91,8,1);
for trial =1:91
   for target = 1:8
      feature = training(:,trial,target);
      probabilities = zeros(1,8);
      for targetGuess = 1:8
         totalProbability = 0;
         for neuron = 1:95
            totalProbability = totalProbability +
log(poisspdf(feature(neuron),meanFeatureClass(neuron,1,targetGuess)));
         end
         probabilities(targetGuess) = totalProbability;
      end
      [val, index] = max(probabilities);
      predicted(trial,target) = index;
   end
end

totalNum = 0;
numCorrect = 0;
for trial = 1:91
   for target = 1:8
      totalNum = totalNum + 1;
      if(predicted(trial, target) == target)
         numCorrect = numCorrect + 1;
      end
   end
end
resultTraining = numCorrect/totalNum;

%% Testing Test Data
```

```
meanFeatureClassTrain = mean(training,2);
predictedTest = zeros(91,8,1);
for trial =1:91
   for target = 1:8
      feature = testing(:,trial,target);
      probabilities = zeros(1,8);
      for targetGuess = 1:8
         totalProbability = 0;
         for neuron = 1:95
            totalProbability = totalProbability +
log(poisspdf(feature(neuron),meanFeatureClassTrain(neuron,1,targetGuess)));
         end
         probabilities(targetGuess) = totalProbability;
      end
      [val, index] = max(probabilities);
      predictedTest(trial,target) = index;
   end
end

totalNum = 0;
numCorrect = 0;
for trial = 92:182
   for target = 1:8
      totalNum = totalNum + 1;
      if(predictedTest(trial-91, target) == target)
         numCorrect = numCorrect + 1;
      end
   end
end
resultTest = numCorrect/totalNum

%% Testing Spoof Data


overallMean = mean(data.firingrate,2);
meanMean = mean(overallMean,3);
spoof = zeros(95,182,8);
for i = 1:95
   for j = 1:182
```

```
        for k = 1:8
            spoof(i,j,k) = poissrnd(meanMean(i));
        end
    end
end

meanFeatureClassTrain = mean(training,2);
predictedSpoof = zeros(182,8,1);
for trial =1:182
    for target = 1:8
        feature = spoof(:,trial,target);
        probabilities = zeros(1,8);
        for targetGuess = 1:8
            totalProbability = 0;
            for neuron = 1:95
                totalProbability = totalProbability +
log(poisspdf(feature(neuron),meanFeatureClassTrain(neuron,1,targetGuess)));
            end
            probabilities(targetGuess) = totalProbability;
        end
        [val, index] = max(probabilities);
        predictedSpoof(trial,target) = index;
    end
end

totalNum = 0;
numCorrect = 0;
for trial = 1:182
    for target = 1:8
        totalNum = totalNum + 1;
        if(predictedSpoof(trial, target) == target)
            numCorrect = numCorrect + 1;
        end
    end
end
resultSpoof = numCorrect/totalNum;

Lab 7 Part 2
load("ecogclassifydata.mat")
trainingData = powervals(1:38,:);
```

```
testData = powervals(39,:);
%size(trainingData)
%size(testData)
%size(group(1:38))
testClass = classify(testData, trainingData, group(1:38), 'linear');
%size(trainingData(38:37,:))
%size(powervals(40:39,:))
%size(vertcat(powervals(1:0,:), powervals(1:39,:)))
predicted = zeros(39,1);
numCorrect = 0;
for i=1:39
   trainingData = vertcat(powervals(1:-1+i,:), powervals(i+1:39,:));
   testData = powervals(i,:);
   trainingClasses = vertcat(group(1:-1+i,:), group(i+1:39,:));
   testClass = classify(testData, trainingData, trainingClasses, 'linear');
   if(testClass == group(i))
      numCorrect = numCorrect + 1;
   end
   predicted(i) = testClass;
end
accuracy = numCorrect/39
confusion = confusionmat(group, predicted)
sum = 0;
total = 0;
for i = 1:5
   for j = 1:5
      if i == j
         sum = sum + confusion(i,j);
      end
      total = total + confusion(i,j);
      confusion(i,j) = confusion(i,j)/7 * 100;
   end
end
sum/total
%size(confusion)
%figure
%pcolor(confusion)
%colorbar

%figure
```

```
imagesc(confusion)
%title("")
c = colorbar;
c.Ruler.TickLabelFormat='%g%%';
%heatmap(confusion)
%colormap default
title("Percentage that data was classified for group in LDA")
xlabel("Group")
ylabel("Group")

Lab 8
load("contdata95.mat");
binData = ones(31413, 950);

allOnes = ones(31413, 1);

for i = 1:95

    binData(:,10*i-9) = Y(:,i);

    binData(:,10*i-8) = circshift(Y(:,i),1);
    binData(1,10*i-8) = 0;

    binData(:,10*i-7) = circshift(Y(:,i),2);
    binData(1:2,10*i-7) = 0;

    binData(:,10*i-6) = circshift(Y(:,i),3);
    binData(1:3,10*i-6) = 0;

    binData(:,10*i-5) = circshift(Y(:,i),4);
    binData(1:4,10*i-5) = 0;

    binData(:,10*i-4) = circshift(Y(:,i),5);
    binData(1:5,10*i-4) = 0;

    binData(:,10*i-3) = circshift(Y(:,i),6);
    binData(1:6,10*i-3) = 0;

    binData(:,10*i-2) = circshift(Y(:,i),7);
    binData(1:7,10*i-2) = 0;
```

```
    binData(:,10*i-1) = circshift(Y(:,i),8);
    binData(1:8,10*i-1) = 0;

    binData(:,10*i) = circshift(Y(:,i),9);
    binData(1:9,10*i) = 0;

end

binData = horzcat(allOnes, binData);

training = binData(1:15707, :);
testing = binData(15708:end, :);

B = inv(transpose(training)* training) * transpose(training)* X(1:15707,:);
Xprediction = testing* B;

figure(1)
plot(X(15708:15708+250,1))
hold on
plot(Xprediction(1:250,1))
title("X Position over 250 ms")
xlabel("Time (ms)")
ylabel("X Position (mm)")
legend("Real X Values", "Predicted X Values")
hold off

figure(2)
plot(X(15708:15708+250,2))
hold on
plot(Xprediction(1:250,2))
title("Y Position over 250 ms")
xlabel("Time (ms)")
ylabel("Y Position (mm)")
legend("Real Y values", "Predicted Y values")

figure(3)
plot(X(15708:15708+250,3))
hold on
plot(Xprediction(1:250,3))
```

```
title("X Velocity over 250 ms")
xlabel("Time (ms)")
ylabel("X velocitiy (mm/ms)")
legend("Real X velocity values", "Predicted X velocity values")

figure(4)
plot(X(15708:15708+250,4))
hold on
plot(Xprediction(1:250,4))
title("Y Velocity over 250 ms")
xlabel("Time (ms)")
ylabel("Y velocitiy (mm/ms)")
legend("Real Y velocity values", "Predicted Y velocity")



%size(training)
%size(B)
%size(training*B)
%size(Xprediction)
%error = training*B-Xprediction
%error = transpose(training*B-Xprediction)*(B*training-Xprediction)
%errorFirst = transpose(B)*transpose(training)*training*B;
%errorSecond = 2*transpose(training)*X(1:15707,:);
%errorThird = transpose(X)*X;
%error = errorFirst - errorSecond;
%MSEXPOS = sum((Xprediction(:,1)-X(15708:end,1)).^2)/15707;

meanSquaredErrorXpos = immse(Xprediction(:,1),X(15708:end,1));
MSExPos = sum((Xprediction(:,1)-X(15708:end,1)).^2)/15707;
meanSquaredErrorYpos = immse(Xprediction(:,2),X(15708:end,2));
MXEyPos = sum((Xprediction(:,2)-X(15708:end,2)).^2)/15707;
meanSquaredErrorXvel = immse(Xprediction(:,3),X(15708:end,3));
MSExVel = sum((Xprediction(:,3)-X(15708:end,3)).^2)/15707;
meanSquaredErrorYvel = immse(Xprediction(:,4),X(15708:end,4));
MSEyVel = sum((Xprediction(:,4)-X(15708:end,4)).^2)/15707;

corrXpos = corr2(Xprediction(:,1),X(15708:end,1));
corrYpos = corr2(Xprediction(:,2),X(15708:end,2));
corrXvel = corr2(Xprediction(:,3),X(15708:end,3));
corrYvel = corr2(Xprediction(:,4),X(15708:end,4));
```

```
lambda = linspace(0,.1,15);
%t = ["Xpos Correlation", corrXpos];

MESlambdas = zeros(1,15);
Corrlambdas = zeros(1,15);

%disp(t)
%Br = inv(transpose(training)* training + lambda(1)*eye(951)*(15707*951)) *
transpose(training)* X(1:15707,:);

for i = 1:15
   %Make sure N is correct
   Btemp = inv(transpose(training)* training + lambda(i)*eye(951)*(15706)) *
transpose(training)* X(1:15707,:);
   XpredictionTemp = testing* Btemp;
   %meanSquaredErrorXposTemp = immse(XpredictionTemp(:,1),X(15708:end,1));
   meanSquaredErrorXposTemp = sum((XpredictionTemp(:,1)-X(15708:end,1)).^2)/15707;
   %meanSquaredErrorYposTemp = immse(XpredictionTemp(:,2),X(15708:end,2));
   meanSquaredErrorYposTemp = sum((XpredictionTemp(:,2)-X(15708:end,2)).^2)/15707;
   %meanSquaredErrorXvelTemp = immse(XpredictionTemp(:,3),X(15708:end,3));
   meanSquaredErrorXvelTemp = sum((XpredictionTemp(:,3)-X(15708:end,3)).^2)/15707;
   %meanSquaredErrorYvelTemp = immse(XpredictionTemp(:,4),X(15708:end,4));
   meanSquaredErrorYvelTemp = sum((XpredictionTemp(:,4)-X(15708:end,4)).^2)/15707;
   print1 = [i, "Mean squared error Xpos", meanSquaredErrorXposTemp];
   print2 = [i, "Mean squared error Ypos", meanSquaredErrorYposTemp];
   print3 = [i, "Mean squared error Xvel", meanSquaredErrorXvelTemp];
   print4 = [i, "Mean squared error Yvel", meanSquaredErrorYvelTemp];

   MESlambdas(1,i) = meanSquaredErrorXposTemp;

   corrXposTemp = corr2(XpredictionTemp(:,1),X(15708:end,1));
   corrYposTemp = corr2(XpredictionTemp(:,2),X(15708:end,2));
   corrXvelTemp = corr2(XpredictionTemp(:,3),X(15708:end,3));
   corrYvelTemp = corr2(XpredictionTemp(:,4),X(15708:end,4));

   Corrlambdas(1,i) = corrXposTemp;

   print5 = [i, "Correlation Xpos", corrXposTemp];
   print6 = [i, "Correlation Ypos", corrYposTemp];
```

```
    print7 = [i, "Correlation Xvel", corrXvelTemp];
    print8 = [i, "Correlation Yvel", corrYvelTemp];
%{
    disp(print1)
    disp(print2)
    disp(print3)
    disp(print4)
    disp(print5)
    disp(print6)
    disp(print7)
    disp(print8)
%}
end

bestLambda = lambda(5);

Btemp = inv(transpose(training)* training + bestLambda*eye(951)*(15706)) *
transpose(training)* X(1:15707,:);
Xprediction1= testing* Btemp;

Bempty = inv(transpose(training)* training + 0*eye(951)*(15706)) * transpose(training)*
X(1:15707,:);
XpredictionEmpty = testing * Bempty;
ridgeNoLambdaMSE = immse(XpredictionEmpty(:,1),X(15708:end,1));

%ridgeMeanSquaredErrorXposTemp = immse(Xprediction1(:,1),X(15708:end,1));
ridgeMeanSquaredErrorXposTemp = sum((Xprediction1(:,1)-X(15708:end,1)).^2)/15707;
%ridgeMeanSquaredErrorYposTemp = immse(Xprediction1(:,2),X(15708:end,2));
ridgeMeanSquaredErrorYposTemp = sum((Xprediction1(:,2)-X(15708:end,2)).^2)/15707;
%ridgeMeanSquaredErrorXvelTemp = immse(Xprediction1(:,3),X(15708:end,3));
ridgeMeanSquaredErrorXvelTemp = sum((Xprediction1(:,3)-X(15708:end,3)).^2)/15707;
%ridgeMeaanSquaredErrorYvelTemp = immse(Xprediction1(:,4),X(15708:end,4));
ridgeMeaanSquaredErrorYvelTemp = sum((Xprediction1(:,4)-X(15708:end,4)).^2)/15707;
print1 = ["Mean squared error Xpos best lambda", ridgeMeanSquaredErrorXposTemp];
print2 = ["Mean squared error Ypos best lambda", ridgeMeanSquaredErrorYposTemp];
print3 = ["Mean squared error Xvel best lambda", ridgeMeanSquaredErrorXvelTemp];
print4 = ["Mean squared error Yvel best lambda", ridgeMeaanSquaredErrorYvelTemp];


ridgeCorrXposTemp = corr2(Xprediction1(:,1),X(15708:end,1));
```

```
ridgeCorrYposTemp = corr2(Xprediction1(:,2),X(15708:end,2));
ridgeCorrXvelTemp = corr2(Xprediction1(:,3),X(15708:end,3));
ridgeCorrYvelTemp = corr2(Xprediction1(:,4),X(15708:end,4));

print5 = ["Correlation Xpos best lambda", ridgeCorrXposTemp];
print6 = ["Correlation Ypos best lambda", ridgeCorrYposTemp];
print7 = ["Correlation Xvel best lambda", ridgeCorrXvelTemp];
print8 = ["Correlation Yvel best lambda", ridgeCorrYvelTemp];

disp(print1)
disp(print2)
disp(print3)
disp(print4)
disp(print5)
disp(print6)
disp(print7)
disp(print8)

figure(5)
plot(X(15708:15708+250,2))
hold on
plot(Xprediction1(1:250,2))
title("Y Position over 250 ms")
xlabel("Time (ms)")
ylabel("Y - Position (mm)")
legend("Real Y values", "Predicted Y values")

figure(6)
plot(X(15708:15708+250,3))
hold on
plot(Xprediction1(1:250,3))
title("X Velocity over 250 ms")
xlabel("Time (ms)")
ylabel("X veloctiy (mm/ms ?)")
legend("Real X velocity values", "Predicted X velocity values")

figure(7)
plot(lambda, MESlambdas)
title("Mean Error Square of X position for various lambdas")
xlabel("Lambda")
```

```
ylabel("Mean Error Sqaure")


figure(8)
plot(lambda, Corrlambdas)
title("Correleation of X position for various lambdas")
xlabel("Lambda")
ylabel("Correlation")

Lab 9 Part 1
load("contdata95.mat");
binData = ones(31413, 950);

allOnes = ones(31413, 1);

for i = 1:95

    binData(:,10*i-9) = Y(:,i);

    binData(:,10*i-8) = circshift(Y(:,i),1);
    binData(1,10*i-8) = 0;

    binData(:,10*i-7) = circshift(Y(:,i),2);
    binData(1:2,10*i-7) = 0;

    binData(:,10*i-6) = circshift(Y(:,i),3);
    binData(1:3,10*i-6) = 0;

    binData(:,10*i-5) = circshift(Y(:,i),4);
    binData(1:4,10*i-5) = 0;

    binData(:,10*i-4) = circshift(Y(:,i),5);
    binData(1:5,10*i-4) = 0;

    binData(:,10*i-3) = circshift(Y(:,i),6);
    binData(1:6,10*i-3) = 0;

    binData(:,10*i-2) = circshift(Y(:,i),7);
    binData(1:7,10*i-2) = 0;
```

```
    binData(:,10*i-1) = circshift(Y(:,i),8);
    binData(1:8,10*i-1) = 0;

    binData(:,10*i) = circshift(Y(:,i),9);
    binData(1:9,10*i) = 0;

end

binData = horzcat(allOnes, binData);

training = binData(1:15707, :);
testing = binData(15708:end, :);

lambda = linspace(0,.1,15);

%Btemp = inv(transpose(training)* training + lambda(5)*eye(951)*(15706)) *
transpose(training)* X(1:15707,:);
%XpredictionTemp = testing* Btemp;

B1 = lasso(training, X(1:15707, 1), 'Lambda', lambda(5));
Xprediction1 = testing*B1;

B2 = lasso(training, X(1:15707, 2), 'Lambda', lambda(5));
Xprediction2 = testing*B2;

B3 = lasso(training, X(1:15707, 3), 'Lambda', lambda(5));
Xprediction3 = testing*B3;

B4 = lasso(training, X(1:15707, 4), 'Lambda', lambda(5));
Xprediction4 = testing*B4;


%lassoMeanSquaredErrorXpos = immse(Xprediction1(:),X(15708:end,1));
%MSExPos = sum((Xprediction(:,1)-X(15708:end,1)).^2)/15707;
lassoMeanSquaredErrorXpos = sum((Xprediction1(:,1)-X(15708:end,1)).^2)/15707;
%lassoMeanSquaredErrorYpos = immse(Xprediction2(:),X(15708:end,2));
lassoMeanSquaredErrorYpos = sum((Xprediction2(:,1)-X(15708:end,2)).^2)/15707;
%lassoMeanSquaredErrorXvel = immse(Xprediction3(:),X(15708:end,3));
lassoMeanSquaredErrorXvel = sum((Xprediction3(:,1)-X(15708:end,3)).^2)/15707;
%lassoMeanSquaredErrorYvel = immse(Xprediction4(:),X(15708:end,4));
```

```
lassoMeanSquaredErrorYvel = sum((Xprediction4(:,1)-X(15708:end,4)).^2)/15707;

lassoCorrXpos = corr2(Xprediction1(:),X(15708:end,1));
lassoCorrYpos = corr2(Xprediction2(:),X(15708:end,2));
lassoCorrXvel = corr2(Xprediction3(:),X(15708:end,3));
lassoCorrYvel = corr2(Xprediction4(:),X(15708:end,4));
```

Lab 9 Part 2

```
load("contdata95.mat");
binData = ones(31413, 950);

allOnes = ones(31413, 1);

for i = 1:95

    binData(:,10*i-9) = Y(:,i);

    binData(:,10*i-8) = circshift(Y(:,i),1);
    binData(1,10*i-8) = 0;

    binData(:,10*i-7) = circshift(Y(:,i),2);
    binData(1:2,10*i-7) = 0;

    binData(:,10*i-6) = circshift(Y(:,i),3);
    binData(1:3,10*i-6) = 0;

    binData(:,10*i-5) = circshift(Y(:,i),4);
    binData(1:4,10*i-5) = 0;

    binData(:,10*i-4) = circshift(Y(:,i),5);
    binData(1:5,10*i-4) = 0;

    binData(:,10*i-3) = circshift(Y(:,i),6);
    binData(1:6,10*i-3) = 0;

    binData(:,10*i-2) = circshift(Y(:,i),7);
    binData(1:7,10*i-2) = 0;
```

```
    binData(:,10*i-1) = circshift(Y(:,i),8);
    binData(1:8,10*i-1) = 0;

    binData(:,10*i) = circshift(Y(:,i),9);
    binData(1:9,10*i) = 0;

end

binData = horzcat(allOnes, binData);

training = transpose(binData(1:15707, :));
testing = transpose(binData(15708:end, :));

Xt = transpose(X(1:(15707-1),:));
Xt1 = transpose(X(2:15707,:));

A  = Xt*transpose(Xt1)*inv(Xt1*transpose(Xt1));
C = training(:,1:15706)*transpose(Xt)*inv(Xt*transpose(Xt));
%firstDeterm = det(C*transpose(C));
W = 1/(15707-1) *(Xt-A*Xt1)*transpose(Xt-A*Xt1);
Q = (1/15707)*(training(:,1:(15707-1)) - C*Xt)*transpose(training(:,1:(15707-1))-C*Xt);

Pt1 = W;
xt1 = transpose(X(15708,:));
predict = zeros(15706,4);
for i = 1:15706
    xtt1 = A * xt1;
    Ptt1 = A*Pt1*transpose(A)+W;
    %tempOriginal = C*Ptt1*transpose(C)+Q;
    %temp = inv(C*transpose(C)*Ptt1);
    withQ = C*Ptt1*transpose(C)+Q;
    withoutQ = C*Ptt1*transpose(C);
    determ = det(C*Ptt1*transpose(C)+Q);
    Kt = Ptt1*transpose(C)*inv(C*Ptt1*transpose(C)+Q);
    xt = xtt1 + Kt*(testing(:,i)-C*xtt1);
    Pt = (eye(4)-Kt*C)*Ptt1;
    predict(i,1) = xt(1,1);
    predict(i,2) = xt(2,1);
    predict(i,3) = xt(3,1);
    predict(i,4) = xt(4,1);
```

```
    xt1 = xt;
    Pt1 = Pt;
    i

end

figure(1)
%plot(X(15708:15708+250,1))
plot(X(15708:end,1))
hold on
plot(predict(15708:end,1))
title("X Position over time")
xlabel("Time (ms)")
ylabel("X Position (mm)")
legend("Real X Values", "Predicted X Values")
hold off

figure(2)
%plot(X(15708:15708+250,2))
plot(X(15708:end,2))
hold on
%plot(predict(1:250,2))
plot(predict(1:end,2))
title("Y Position over time")
xlabel("Time (ms)")
ylabel("Y Position (mm)")
legend("Real Y values", "Predicted Y values")

figure(3)
%plot(X(15708:15708+250,3))
plot(X(15708:end,3))
hold on
%plot(predict(1:250,3))
plot(predict(1:end,3))
title("X Velocity over time")
xlabel("Time (ms)")
ylabel("X veloctiy (mm/ms)")
legend("Real X velocity values", "Predicted X velocity values")
```

```
figure(4)
%plot(X(15708:15708+250,4))
plot(X(15708:end,4))
hold on
%plot(predict(1:250,4))
plot(predict(1:end,4))
title("Y Velocity over time")
xlabel("Time (ms)")
ylabel("Y veloctiy (mm/ms)")
legend("Real Y velocity values", "Predicted Y velocity")

kalmanMSExPos = sum((predict(:,1)-X(15708:end,1)).^2)/15707;
kalmanMXEyPos = sum((predict(:,2)-X(15708:end,2)).^2)/15707;
kalmanMSExVel = sum((predict(:,3)-X(15708:end,3)).^2)/15707;
kalmanMSEyVel = sum((predict(:,4)-X(15708:end,4)).^2)/15707;
kalmanCorrXpos = corr2(predict(:,1),X(15708:end,1));
kalmanCorrYpos = corr2(predict(:,2),X(15708:end,2));
kalmanCorrXvel = corr2(predict(:,3),X(15708:end,3));
kalmanCorrYvel = corr2(predict(:,4),X(15708:end,4));
```