Table of Contents

James Skelly, ECG703 Sp 2021 Midterm Exam

Instructor: Dr. Latifi

Due Date: March 30, 2021

```
close all
```

Problem 1: Logistic Regression

The data comprises data related to studies on real participants in relation to heart disease. The data was cleansed in STATA resulting in the original data of 4,239 data entries being reduced to 3,749.

Two datasets of 3 features each were selected after examining plots of the data.

P1. Importing the Data from Excel

```
[diaBP1, BMI1, heartRate1, prevalentHyp1] =
 importHeartData('HeartData_01.xlsx');
```

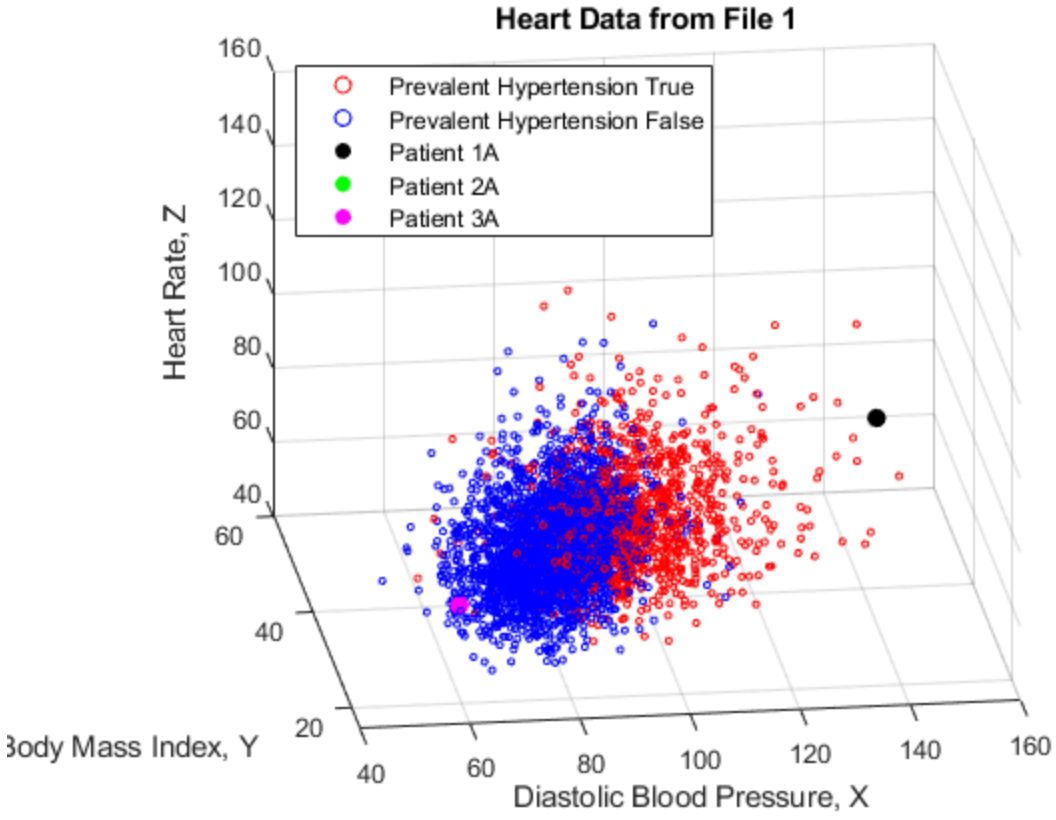
```
[totChol2, sysBP2, glucose2, prevalentHyp2] =
 importHeartData('HeartData_02.xlsx');
```

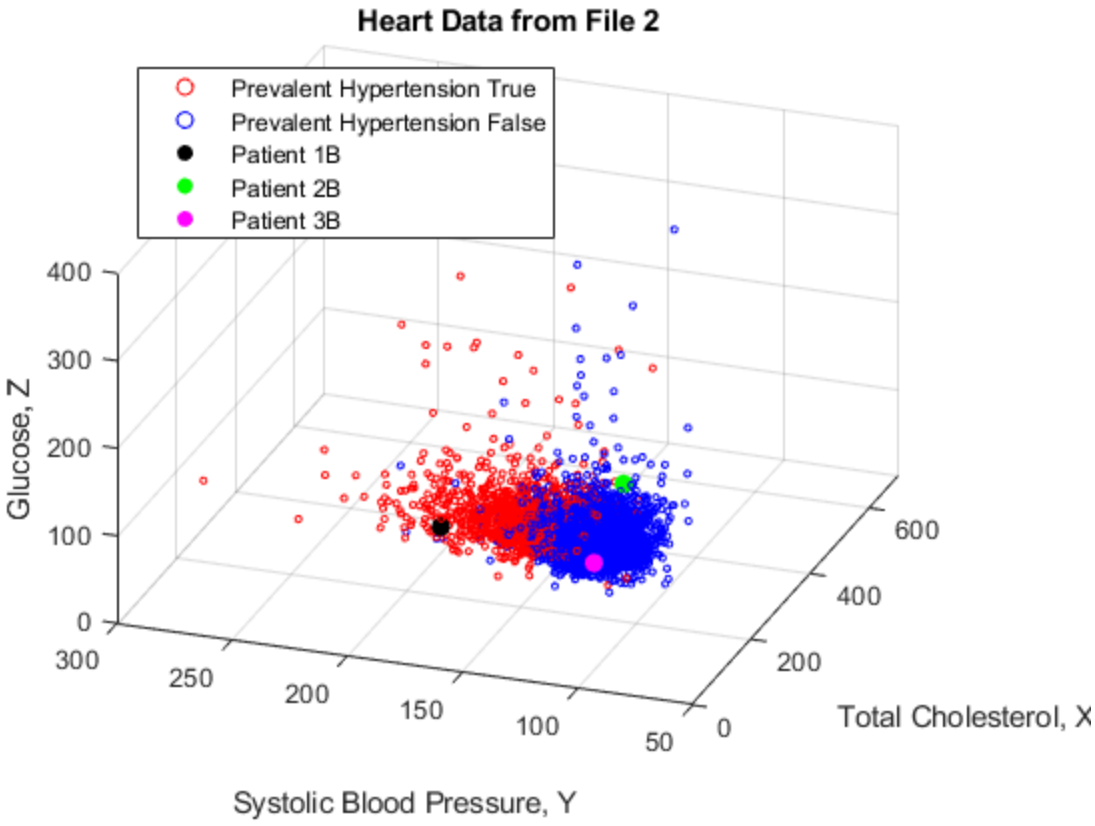
P1. Plotting Data for Visualization

```
% Store data from excel in one matrix for heart data 1
HeartData_01(:,2) = diaBP1;
HeartData_01(:,3) = BMI1;
HeartData_01(:,4) = heartRate1;
HeartData_01(:,5) = prevalentHyp1;

% Store data from excel in one matrix for heart data 2
HeartData_02(:,2) = totChol2;
HeartData_02(:,3) = sysBP2;
HeartData_02(:,4) = glucose2;
HeartData_02(:,5) = prevalentHyp2;

% Obtain the number of rows in the data matrices for later looping
[nRows1,~] = size(HeartData_01);
[nRows2,~] = size(HeartData_02);

% Set the first data column to all ones for w0
HeartData_01(:,1) = ones(nRows1,1);
HeartData_02(:,1) = ones(nRows2,1);

% Generate matrices to hold prevalent hypertension true/false data
 points
pH1True = zeros(10,5);
pH1False = zeros(10,5);

% Initialize values for true and false counts
trueCount = 1;
falseCount = 1;

% For heart data 1, store true and false hypertension data separately
for i = 1:nRows1
    if HeartData_01(i,5) == 1
        pH1True(trueCount,:) = HeartData_01(i,:);
        trueCount = trueCount + 1;
    else
        pH1False(falseCount,:) = HeartData_01(i,:);
        falseCount = falseCount + 1;
        HeartData_01(i,5) = -1;
    end
end

% Generate matrices to hold prevalent hypertension true/false data
 points
pH2True = zeros(10,5);
pH2False = zeros(10,5);

% Initialize values for true and false counts
```

```
trueCount = 1;
falseCount = 1;

% For heart data 2, store true and false hypertension data separately
for i = 1:nRows2
    if HeartData_02(i,5) == 1
        pH2True(trueCount,:) = HeartData_02(i,:);
        trueCount = trueCount + 1;
    else
        pH2False(falseCount,:) = HeartData_02(i,:);
        falseCount = falseCount + 1;
        HeartData_02(i,5) = -1;
    end
end

sizePt = 6;
sizePt2 = 50;
HypTrueLegend = 'Prevalent Hypertension True';
HypFalseLegend = 'Prevalent Hypertension False';

% Plot data for given patients, training data for heart data 1
patient1a = [1; 140; 33; 95];
patient2a = [1; 80; 27; 70];
patient3a = [1; 60; 21; 65];

figure('Name', '3D Plot of Heart Data File 1')
scatter3(pH1True(:,2), pH1True(:,3), pH1True(:,4), sizePt, 'red')
view([-7.511,25.704])
title('Heart Data from File 1');
xlabel('Diastolic Blood Pressure, X');
ylabel('Body Mass Index, Y');
zlabel('Heart Rate, Z');
hold on
scatter3(pH1False(:,2), pH1False(:,3), pH1False(:,4), sizePt, 'blue')
scatter3(patient1a(2), patient1a(3), patient1a(4),
 sizePt2, 'black', 'filled')
LP1 = 'Patient 1A';
scatter3(patient2a(2), patient2a(3), patient2a(4),
 sizePt2, 'green', 'filled')
LP2 = 'Patient 2A';
scatter3(patient3a(2), patient3a(3), patient3a(4),
 sizePt2, 'magenta', 'filled')
LP3 = 'Patient 3A';
legend(HypTrueLegend, HypFalseLegend, LP1, LP2,
 LP3, 'Location', 'Northwest')
hold off

% Plot data for given patients, training data for heart data 2
patient1b = [1; 160; 180; 95];
patient2b = [1; 311; 120; 110];
patient3b = [1; 133; 110; 90];

figure('Name', '3D Plot of Heart Data File 2')
scatter3(pH2True(:,2), pH2True(:,3), pH2True(:,4), sizePt, 'red')
```

```
view([-70.278,34.716])
title('Heart Data from File 2');
xlabel('Total Cholesterol, X');
ylabel('Systolic Blood Pressure, Y');
zlabel('Glucose, Z');
hold on
scatter3(pH2False(:,2), pH2False(:,3), pH2False(:,4), sizePt, 'blue')
scatter3(patient1b(2), patient1b(3), patient1b(4),
 sizePt2, 'black', 'filled')
LP1b = 'Patient 1B';
scatter3(patient2b(2), patient2b(3), patient2b(4),
 sizePt2, 'green', 'filled')
LP2b = 'Patient 2B';
scatter3(patient3b(2), patient3b(3), patient3b(4),
 sizePt2, 'magenta', 'filled')
LP3b = 'Patient 3B';
legend(HypTrueLegend,
 HypFalseLegend,LP1b,LP2b,LP3b,'Location','northwest')
hold off
```





P1. Part A.

Find the probability of prevalent hypertension for the following patients using HeartData_01.

```
% Generate x matrix, y vector for heart data 1
x_01(:,1) = HeartData_01(:,1);
x_01(:,2) = (HeartData_01(:,2));
x_01(:,3) = (HeartData_01(:,3));
x_01(:,4) = (HeartData_01(:,4));
y_01 = HeartData_01(:,5);

% Compute the transpose of X
xt_01 = transpose(x_01);

% Generate the initial weight vector and its transpose
w_01 = zeros(4,1);
wt_01= transpose(w_01);

% Initialize sum for error measure and gradient to 0
sum_01g = 0;
sum_01e = 0;

% Set learning rate
nLearn = 0.01;

% Set number of iterations and allocate memory for weights, gradients,
```

```
% errors on each iteration
nIterations = 1000;
weights = zeros(4,nIterations);
gradients = zeros(4,nIterations);
errors = zeros(1, nIterations);


for i = 1:nIterations

        sum_01g = 0;
        sum_01e = 0;

    for n = 1:nRows1

        % Compute the sum used to calculate the gradient
        sum_01g = sum_01g + ((y_01(n)*xt_01(:,n))/
(1+exp(y_01(n)*wt_01*xt_01(:,n))));

        eInSample_01 = log(1 + exp(-1*y_01(n)*wt_01*xt_01(:,n)));
        sum_01e = sum_01e + eInSample_01;

    end

% Compute the gradient from the sum
del_Ein = (-1/nRows1)*sum_01g;

% Compute the in sample error
Ein = (1/nRows1)*sum_01e;

% Compute the norm of the gradient
norm_del_Ein = norm(del_Ein);

% Compute the varying learning rate
nRate = nLearn/norm_del_Ein;

% Update the weights
w_01 = w_01 - (nRate*(del_Ein));
wt_01 = transpose(w_01);

% Monitor the weights and gradients
weights(:,i) = w_01;
gradients(:,i) = del_Ein;
errors(i) = Ein;

end

probHyp_patient1a = 1/(1+exp(-1*wt_01*patient1a));
probHyp_patient2a = 1/(1+exp(-1*wt_01*patient2a));
probHyp_patient3a = 1/(1+exp(-1*wt_01*patient3a));
```

P1. Part B.

Find the probability of prevalent hypertension for the following patients using HeartData_02.

```
% Generate x matrix, y vector for heart data 2
x_02(:,1) = HeartData_02(:,1);
x_02(:,2) = HeartData_02(:,2);
x_02(:,3) = HeartData_02(:,3);
x_02(:,4) = HeartData_02(:,4);
y_02 = HeartData_02(:,5);

% Compute the transpose of X
xt_02 = transpose(x_02);

% Generate the initial weight vector and its transpose
w_02 = zeros(4,1);
wt_02= transpose(w_02);

% Initialize sum for error measure and gradient to 0
sum_02g = 0;
sum_02e = 0;

% Set learning rate
nLearn2 = 0.001;

% Set number of iterations and allocate memory for weights, gradients,
% errors on each iteration
nIterations2 = 600;
weights2 = zeros(4,nIterations);
gradients2 = zeros(4,nIterations);
errors2 = zeros(1, nIterations);


for i = 1:nIterations

        sum_01g2 = 0;
        sum_01e2 = 0;

    for n = 1:nRows2

        % Compute the sum used to calculate the gradient
        sum_01g2 = sum_01g2 + ((y_02(n)*xt_02(:,n))/
(1+exp(y_02(n)*wt_02*xt_02(:,n))));

        eInSample_02 = log(1 + exp(-1*y_02(n)*wt_02*xt_02(:,n)));
        sum_01e2 = sum_01e2 + eInSample_02;

    end

% Compute the gradient from the sum
del_Ein2 = (-1/nRows2)*sum_01g2;

% Compute the in sample error
Ein2 = (1/nRows2)*sum_01e2;

% Compute the norm of the gradient
norm_del_Ein2 = norm(del_Ein2);
```

```
% Compute the varying learning rate
nRate2 = nLearn2/norm_del_Ein2;

% Update the weights
w_02 = w_02 - (nRate2*(del_Ein2));
wt_02 = transpose(w_02);

% Monitor the weights and gradients
weights2(:,i) = w_02;
gradients2(:,i) = del_Ein2;
errors2(i) = Ein2;

end

probHyp_patient1b = 1/(1+exp(-1*wt_02*patient1b));
probHyp_patient2b = 1/(1+exp(-1*wt_02*patient2b));
probHyp_patient3b = 1/(1+exp(-1*wt_02*patient3b));
```

P1. Outputs and Probability Values for Given Patients

```
fprintf('\n')
fprintf('Final Weights:\n');
fprintf('\n')
disp('Heart Data 1 Weights:');
disp(w_01)
disp('Heart Data 2 Weights:');
disp(w_02)

fprintf('\n');
fprintf('Part A Data:\n');
fprintf('Probability of Hypertension for Patient 1A = %0.2f\n',
 probHyp_patient1a);
fprintf('Probability of Hypertension for Patient 2A = %0.2f\n',
 probHyp_patient2a);
fprintf('Probability of Hypertension for Patient 3A = %0.2f\n',
 probHyp_patient3a);
fprintf('\n');
fprintf('Part B Data:\n');
fprintf('Probability of Hypertension for Patient 1B = %0.2f\n',
 probHyp_patient1b);
fprintf('Probability of Hypertension for Patient 2B = %0.2f\n',
 probHyp_patient2b);
fprintf('Probability of Hypertension for Patient 3B = %0.2f\n',
 probHyp_patient3b);
```

```
Final Weights:

Heart Data 1 Weights:
   -0.0206
    0.0738
```

```
   -0.0855
   -0.0543

Heart Data 2 Weights:
   -0.0042
   -0.0162
    0.0456
   -0.0352


Part A Data:
Probability of Hypertension for Patient 1A = 0.91
Probability of Hypertension for Patient 2A = 0.44
Probability of Hypertension for Patient 3A = 0.29

Part B Data:
Probability of Hypertension for Patient 1B = 0.91
Probability of Hypertension for Patient 2B = 0.03
Probability of Hypertension for Patient 3B = 0.42
```

P1. Concluding Remarks

For case (a), the results are intuitive. We can observe from the figure **Heart Data from File 1** that higher diastolic blood pressure results in a higher probability of prevalent hypertension, while BMI and Heart Rate do not seem to play a large role. Since the main determining factor appears to by diastolic blood pressure, it is intuitive that that the patient with the highest diastolic blood pressure, patient 1, has the highest probability of prevalent hypertension while the patient with the lowest diastolic blood pressure, patient 3, has the lowest probability of prevalent hypertension.

For case (b), the results are a bit less intuitive. We can observe again from the figure **Heart Data from File 2** that higher systolic blood pressure results in a higher probability of prevalent hypertension, while total cholesterol and glucose levels play a smaller role. Since the main determining factor is systolic blood pressure, it is intuitive that the patient with the highest systolic blood pressure, patient 1, has the highest probability of prevalent hypertension. We would then expect that, since systolic blood pressure appears to play the largest role, patients 2 and 3 should have similarly low probability of having prevalent hypertension, but this is not the case.

Overall, it is clear from the data (before even performing a logistic regression) that high diastolic and systolic blood pressure are the greatest signs that a patient is likely to have prevalent hypertension. The other variables included may play a small role, but none greater than blood pressure. This data could be used to predict the potential risk of other sorts of health problems as well, such as diabetes.

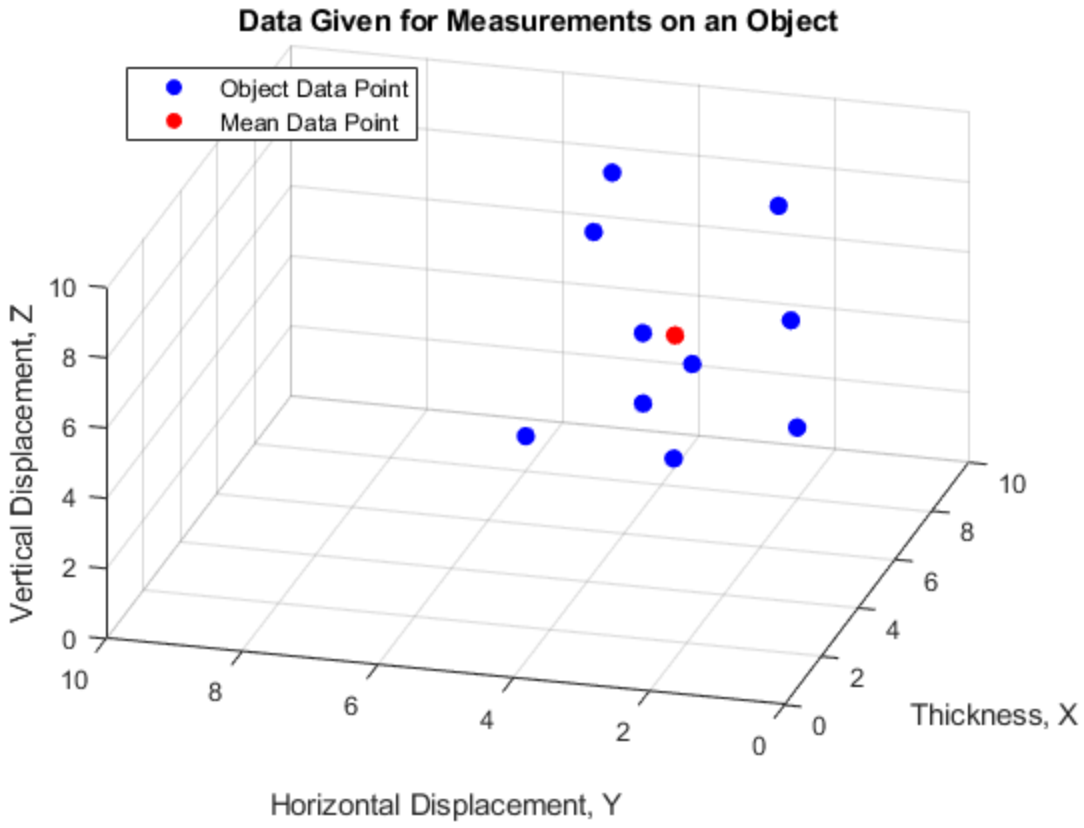
Problem 2: Principal Component Analysis

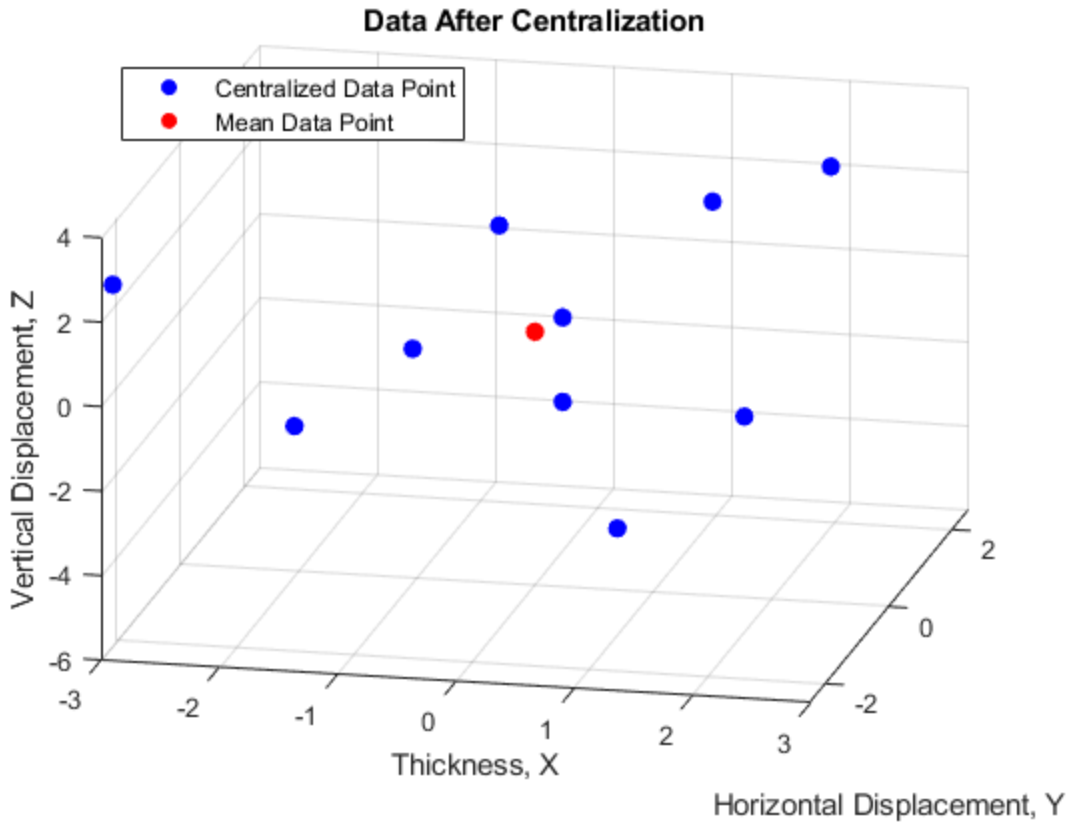
Consider the following 3-variable dataset with 10 data points. Each point consists of 3 measurements on an object: thickness, horizontal displacement, and vertical displacement.

```
% Let X have the form [x1, x2, x3]

X = [7 4 3;
     4 1 8;
     6 3 5;
```

```
    8 6 1;
    8 5 7;
    7 2 9;
    5 3 3;
    9 5 8;
    7 4 5;
    8 2 2];

% Compute the tranpose of X
XT = transpose(X);

% Generate vector containing size of data
[nRows, nCols] = size(X);

% Split data by column for later computation
x1 = X(:,1);
x2 = X(:,2);
x3 = X(:,3);

% Obtain the means of each column of data
mu = mean(X);
mu1 = mu(1);
mu2 = mu(2);
mu3 = mu(3);

% Compute the centralized features
x1_centralized = x1 - mu1;
x2_centralized = x2 - mu2;
x3_centralized = x3 - mu3;

% Store centralized features in centralized data matrix
X_centralized = [x1_centralized, x2_centralized, x3_centralized];

% Plot the given data and the mean value of the data
figure('Name', 'Initial Plot of Given 3D Data')
scatter3(X(:,1),X(:,2),X(:,3), 50, 'blue', 'filled')
view([-74.7,35.571])
L1 = 'Object Data Point';
hold on
title('Data Given for Measurements on an Object');
xlabel('Thickness, X');
ylabel('Horizontal Displacement, Y');
zlabel('Vertical Displacement, Z');
xlim([0 10]);
ylim([0 10]);
zlim([0 10]);
xticks(0:2:10)
yticks(0:2:10)
zticks(0:2:10)
scatter3(mu1, mu2, mu3, 50, 'red', 'filled')
L2 = 'Mean Data Point';
legend(L1, L2, 'location', 'northwest')
grid on
hold off
```

```
% Plot the centralized data
figure('Name', 'Plot of Centralized 3D Data')
scatter3(x1_centralized,x2_centralized,x3_centralized,
50, 'blue', 'filled')
view([12.600,25.041])
L1c = 'Centralized Data Point';
hold on
title('Data After Centralization');
xlabel('Thickness, X');
ylabel('Horizontal Displacement, Y');
zlabel('Vertical Displacement, Z');
scatter3(0, 0, 0, 50, 'red', 'filled')
L2c = 'Mean Data Point';
legend(L1c, L2c, 'location', 'northwest')
grid on
hold off
```





P2. Part A.

Compute the correlation/covariance matrix.

```
% Compute the covariance matrix and corresponding correlation matrix
covX = (1/(nRows-1))*(transpose(X_centralized)*X_centralized);
corrX = corrcov(covX);
```

P2. Part B.

Determine the eigen values and select 2 eigen vectors which correspond to the 2 largest eigen values.

```
% Compute the eigen values and eigen vectors from the covariance
matrix
[eigVects,eigVals] = eig(covX);

% Obtain the principal components by extracting the maximum eigen
values
[vals, indeces] = max(eigVals);
[PC1val, index1] = max(vals);
vals(index1) = 0;
[PC2val, index2] = max(vals);

PC1 = eigVects(:,index1);
PC2 = eigVects(:,index2);
```

```
myEigVects = [PC1, PC2];
```

P2. Part C.

Construct the projection matrix.

```
% Compute the projection matrix from the data and the principal
components
magMatrix = [X_centralized*PC1, X_centralized*PC2];
Xproj = magMatrix * transpose(myEigVects);
```

P2. Part D.

Transform the data to the 2-variate dataset.

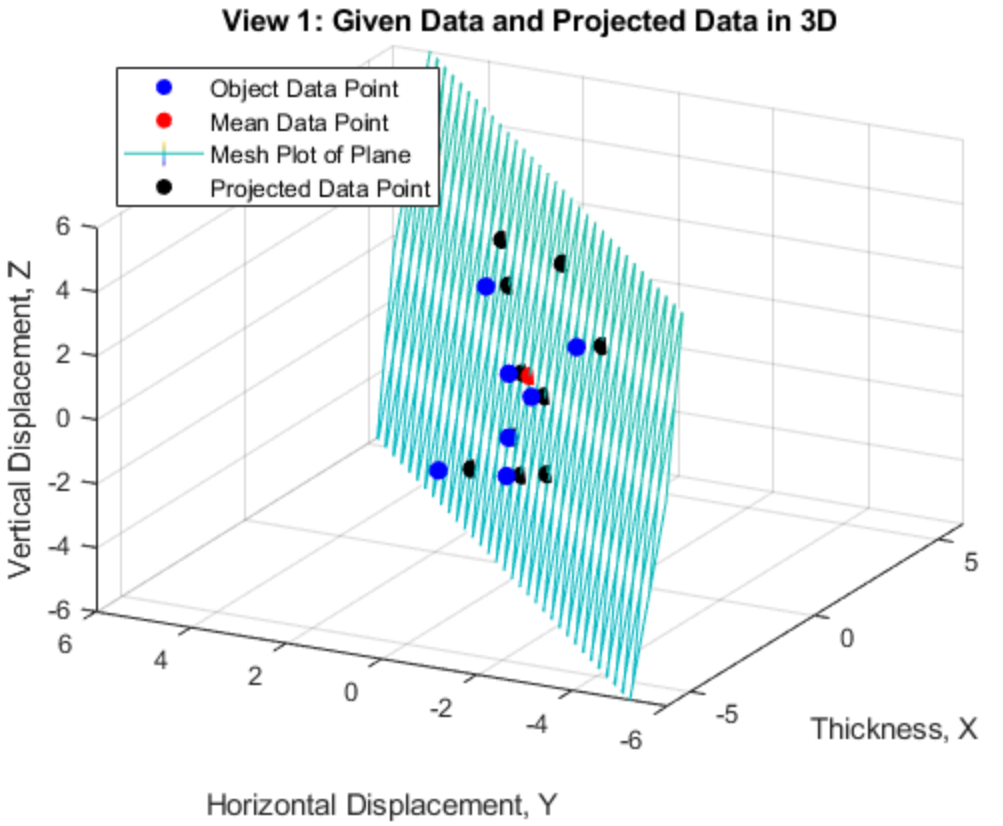
```
% Obtain the 2D data from the 3D projected data.
data_3D = Xproj;
data_2D = [Xproj(:,1),Xproj(:,2)];
```

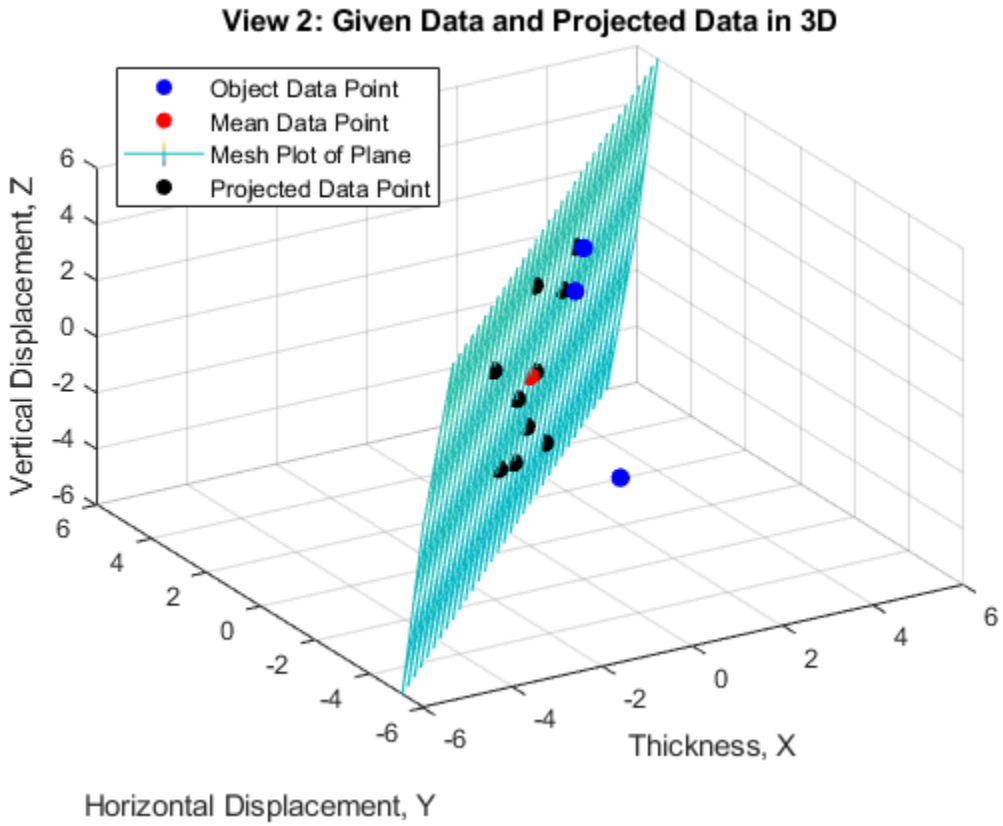
P2. Part E.

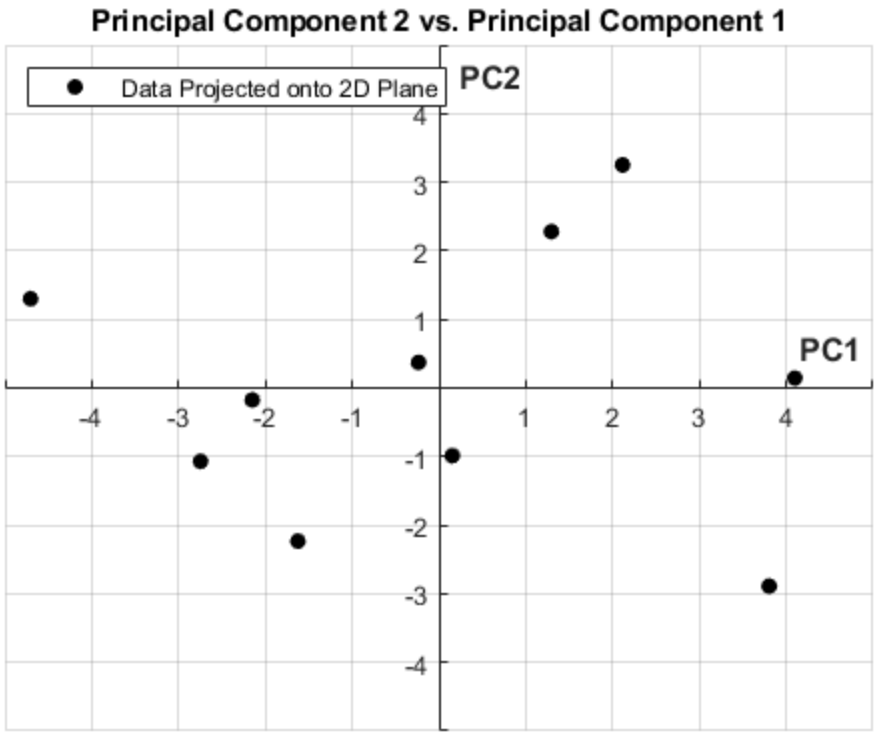
Plot the 3D data set and the resultant 2D dataset by color coding the variables.

```
% Store 3 points from the projected data to generate a plane
P1 = Xproj(1,:);
P2 = Xproj(2,:);
P3 = Xproj(3,:);

normal = cross(P1-P2, P1-P3);

% Create symbolic variables for x, y, and z to generate plane
equation
syms x y z
P = [x, y, z];

% Compute the plane equation
planeFunction = dot(normal, P-P1);
zplane = solve(planeFunction, z);

% Plot the first view of the projected data onto the plane
figure('Name', 'PCA Plot (1) of Projected 3D Data onto Plane')
scatter3(x1_centralized,x2_centralized,x3_centralized,
50, 'blue', 'filled')
view([-62.4,28.22])
L1 = 'Object Data Point';
hold on
title('View 1: Given Data and Projected Data in 3D');
xlabel('Thickness, X');
ylabel('Horizontal Displacement, Y');
zlabel('Vertical Displacement, Z');
xlim([-6 6]);
ylim([-6 6]);
zlim([-6 6]);
```

```
scatter3(0, 0, 0, 50, 'red', 'filled')
L2 = 'Mean Data Point';
grid on
fmesh(zplane)
scatter3(Xproj(:,1),Xproj(:,2),Xproj(:,3), 50, 'black', 'filled')
L3 = 'Projected Data Point';
legend(L1, L2,'Mesh Plot of Plane', L3, 'location', 'northwest')
hold off

% Plot the second view of the projected data onto the plane
figure('Name', 'PCA Plot (2) of Projected 3D Data onto Plane')
scatter3(x1_centralized,x2_centralized,x3_centralized,
50, 'blue', 'filled')
view([-31.156,35.092])
L1 = 'Object Data Point';
hold on
title('View 2: Given Data and Projected Data in 3D');
xlabel('Thickness, X');
ylabel('Horizontal Displacement, Y');
zlabel('Vertical Displacement, Z');
xlim([-6 6]);
ylim([-6 6]);
zlim([-6 6]);
scatter3(0,0,0, 50, 'red', 'filled')
L2 = 'Mean Data Point';
grid on
fmesh(zplane)
scatter3(Xproj(:,1),Xproj(:,2),Xproj(:,3), 50, 'black', 'filled')
L3 = 'Projected Data Point';
legend(L1, L2,'Mesh Plot of Plane', L3, 'location', 'northwest')
hold off


figure('Name', 'Principal Component Analysis Final 2D Plot')
scatter(magMatrix(:,1),magMatrix(:,2), 'black', 'filled')
title('Principal Component 2 vs. Principal Component 1')
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
xlabel('PC1', 'FontSize', 12, 'FontWeight', 'bold')
ylabel('PC2', 'FontSize', 12, 'FontWeight', 'bold')
xlim([-5 5]);
ylim([-5 5]);
grid on
legend('Data Projected onto 2D Plane', 'location', 'nw')
```


View 1: Given Data and Projected Data in 3D
Object Data Point
Mean Data Point
Mesh Plot of Plane
Projected Data Point
Vertical Displacement, Z
Horizontal Displacement, Y
Thickness, X


View 2: Given Data and Projected Data in 3D
Object Data Point
Mean Data Point
Mesh Plot of Plane
Projected Data Point
Vertical Displacement, Z
Horizontal Displacement, Y
Thickness, X



P2. Outputs

```
fprintf('\n')
fprintf('Principal Component Vectors, PC1 and PC2:\n')
disp(myEigVects)
fprintf('\n')
fprintf('Final Magnitudes (PC1, PC2):\n')
disp(magMatrix)
fprintf('\n')
```

```
Principal Component Vectors, PC1 and PC2:
   -0.1376    0.6990
   -0.2505    0.6609
    0.9583    0.2731


Final Magnitudes (PC1, PC2):
   -2.1514   -0.1731
    3.8042   -2.8875
    0.1532   -0.9869
   -4.7065    1.3015
    1.2938    2.2791
    4.0993    0.1436
   -1.6258   -2.2321
    2.1145    3.2512
```

```
   -0.2348    0.3730
   -2.7464   -1.0689
```

P2. Concluding Remarks

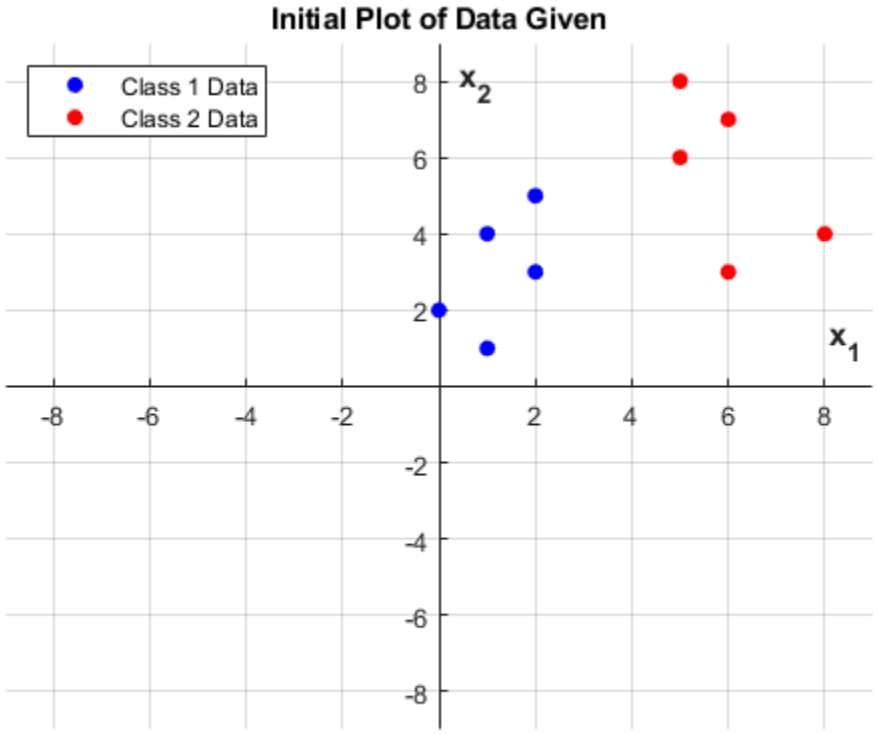
For the principal component analysis problem, the eigen values and eigen vectors were obtained to select the principal components, or the axes along which the variance of the data was the greatest. Once the principal components PC1 and PC2 were obtained, the data points were projected onto the 2D plane created from PC1 and PC2. Finally, the data points were plotted in 2D where PC1 is the x-axis and PC2 is the y-axis. The purpose of PCA is dimensionality reduction for faster computation and better data visualization. In this case, the data given was 3D, and could therefore be visualized with no need for dimensionality reduction. However, the methods used previously to solve this problem can be extended to much higher dimensional data to reduce the dimensionality for faster computationa and data visualization which would otherwise not be possible, since we can only see in 3D.

Problem 3: Linear Discriminant Analysis

Consider the following 2D dataset:

```
C1 = 'blue';
X_C1 = [0, 2;
        1, 1;
        1, 4;
        2, 3;
        2, 5];

C2 = 'red';
X_C2 = [5, 6;
        5, 8;
        6, 3;
        6, 7;
        8, 4];

% Declare and populate variables to store size of data matrices
[nRows1,~] = size(X_C1);
[nRows2,~] = size(X_C2);

% Plot the given data in a 2D plane
figure('Name', 'Plot of C1 and C2 Data Before LDA');
scatter(X_C1(:,1),X_C1(:,2),C1,'filled')
hold on
title('Initial Plot of Data Given');
xlabel('x_{1}', 'FontSize', 12, 'FontWeight', 'bold');
ylabel('x_{2}', 'FontSize', 12, 'FontWeight', 'bold');
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
xlim([-9 9]);
ylim([-9 9]);
scatter(X_C2(:,1),X_C2(:,2),C2,'filled')
```

```
grid on
legend('Class 1 Data', 'Class 2 Data', 'location', 'northwest')
hold off

% Compute the mean for each class

mu1 = mean(X_C1);
mu2 = mean(X_C2);
```



P3. Part A.

Find the direction V along which the two classes are best discriminated using the LDA. (Get the mu1, mu2, Sw, Sb, V, W, and y following the procedure in lecture 9).

```
% Initialize matrix to store thetas, J(v) ratios to check solution
resultMatrix = zeros(180,2);

x = linspace(-10, 10, 10000);

for theta = 1:180

    % obtain x, y components of a vector in the direction of theta
    [xv, yv, m] = GenerateLineFromAngle(theta);

    % declare vector v, the current direction vector
    v = [xv;
```

```
     yv];

% Project C1, C2 points onto generated line

% Generate matrices to hold the projected points' coordinates
Y_C1 = zeros(5,2);
Y_C2 = zeros(5,2);

% Generate vectors to hold the projected points' magnitudes
Y_Mag1 = zeros(5,1);
Y_Mag2 = zeros(5,1);

% Populate the Y vector, matrix for class 1
for i = 1:nRows1

    Y_Mag1(i) = dot(transpose(v),X_C1(i,:));
    [xp1,yp1] = ProjectPoint(Y_Mag1(i),theta);
    Y_C1(i,1) = xp1;
    Y_C1(i,2) = yp1;

end

% Populate the Y vector, matrix for class 2
for j = 1:nRows2

    Y_Mag2(j) = dot(transpose(v),X_C2(j,:));
    [xp2,yp2] = ProjectPoint(Y_Mag2(j),theta);
    Y_C2(j,1) = xp2;
    Y_C2(j,2) = yp2;

end

% Compute projected mean for each class

mu1_tilda = dot(transpose(v),mu1);
mu2_tilda = dot(transpose(v),mu2);


% Compute scatter for each class

sProj1 = 0;
sProj2 = 0;

for i = 1:nRows1

    sProj1 = sProj1 + ((Y_Mag1(i) - mu1_tilda)^2);
    sProj2 = sProj2 + ((Y_Mag2(i) - mu2_tilda)^2);

end

% Normalize by both sProj1 and sProj2

J1 = (((mu1_tilda - mu2_tilda)^2)/(sProj1 + sProj2));
```

```
    % Compute the separate class scatter matrices S1 and S2

    s1 = zeros(2,2);
    s2 = zeros(2,2);

    for i = 1:nRows1

        s1 = s1 + (transpose((X_C1(i,:) - mu1)))*(X_C1(i,:) - mu1);
        s2 = s2 + (transpose((X_C2(i,:) - mu2)))*(X_C2(i,:) - mu2);

    end



    % Define the within class & between class scatter matrix
    sW = s1 + s2;
    sB = transpose(mu1-mu2)*(mu1-mu2);

    % Compute alternate form of J ratio
    J2Num = transpose(v)*sB*v;
    J2Den = transpose(v)*sW*v;

    J2 = J2Num/J2Den;

    % Store final ratio in result matrix for checking solution
    resultMatrix(theta,1) = theta;
    resultMatrix(theta,2) = J2;
end

% After taking derivative of J and setting equal to zero, v can be
 found.
vFinal = (mu1-mu2)/sW;
thetaFinal = FindOptimalTheta(vFinal);
```

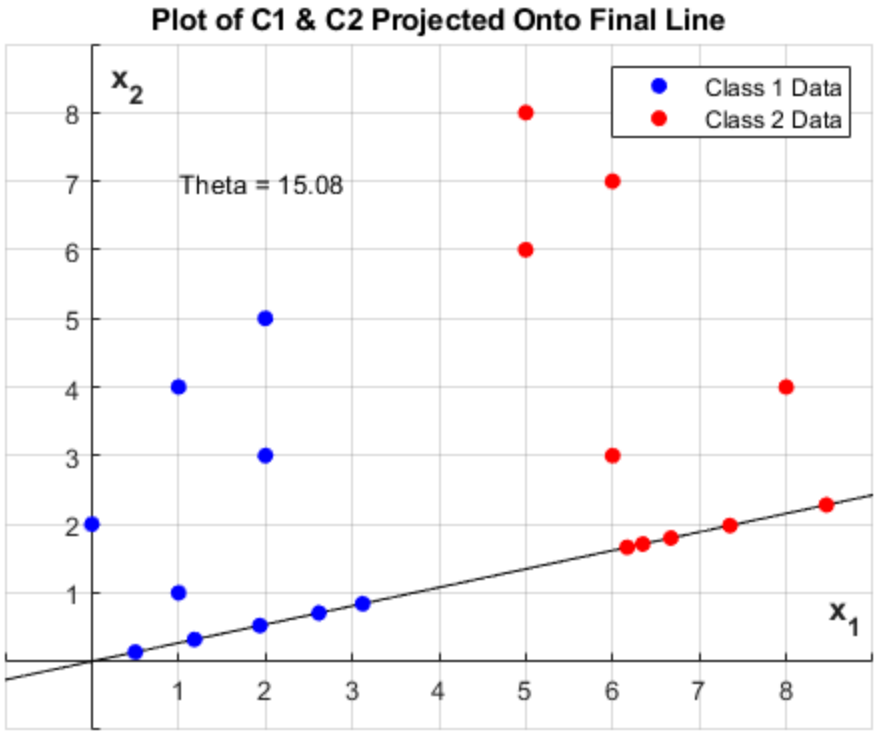
P3. Part B.

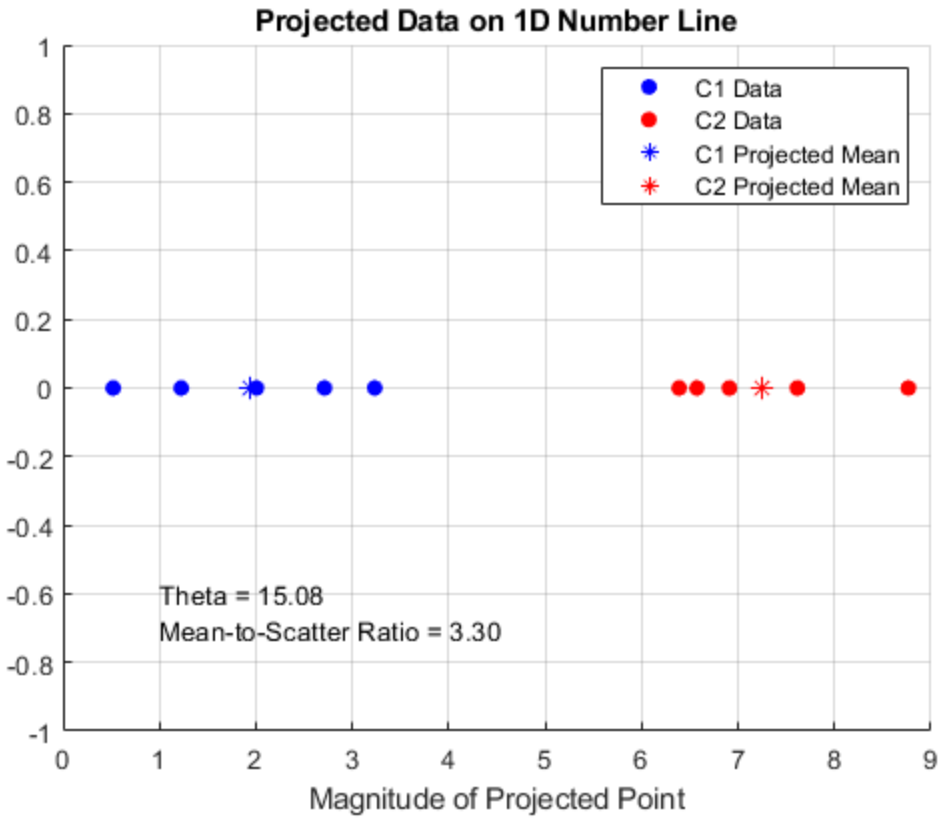
Plot the data before and after transformation using different colors for the two classes.

```
[maxVal, index] = max(resultMatrix(:,2));

% Generate matrices to hold the projected points' coordinates
Y_C1f = zeros(5,2);
Y_C2f = zeros(5,2);

% Generate vectors to hold the projected points' magnitudes
Y_Mag1f = zeros(5,1);
Y_Mag2f = zeros(5,1);

[xf, yf, mf] = GenerateLineFromAngle(thetaFinal);

vf = [xf;
      yf];
```

```
yOpt = mf*x;

% Compute final projected mean for each class

mu1_tildaf = dot(transpose(vf),mu1);
mu2_tildaf = dot(transpose(vf),mu2);

% Populate the Y vector, matrix for class 1 and optimal theta
for i = 1:nRows1

    Y_Mag1f(i) = dot(transpose(vf),X_C1(i,:));
    [xp1,yp1] = ProjectPoint(Y_Mag1f(i),thetaFinal);
    Y_C1f(i,1) = xp1;
    Y_C1f(i,2) = yp1;

end

% Populate the Y vector, matrix for class 2 and optimal theta
for j = 1:nRows2

    Y_Mag2f(j) = dot(transpose(vf),X_C2(j,:));
    [xp2,yp2] = ProjectPoint(Y_Mag2f(j),thetaFinal);
    Y_C2f(j,1) = xp2;
    Y_C2f(j,2) = yp2;

end

figure('Name', 'Projecting Data onto Final LDA Line');
scatter(X_C1(:,1),X_C1(:,2),C1,'filled')
hold on
title('Plot of C1 & C2 Projected Onto Final Line');
xlabel('x_{1}', 'FontSize', 12, 'FontWeight', 'bold');
ylabel('x_{2}', 'FontSize', 12, 'FontWeight', 'bold');
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
xlim([-1 9]);
ylim([-1 9]);
scatter(X_C2(:,1),X_C2(:,2),C2,'filled')
plot(x, yOpt, 'black')
scatter(Y_C1f(:,1),Y_C1f(:,2),C1,'filled')
scatter(Y_C2f(:,1),Y_C2f(:,2),C2,'filled')
grid on
legend('Class 1 Data', 'Class 2 Data', 'location', 'northeast')
text(1,7,sprintf('Theta = %0.2f', thetaFinal));
hold off


figure('Name', 'Plot of Projected Data on 1D Number Line')
scatter(Y_Mag1f, zeros(5,1), 'blue', 'filled')
title('Projected Data on 1D Number Line')
xlabel('Magnitude of Projected Point')
hold on
```

```
scatter(Y_Mag2f, zeros(5,1), 'red', 'filled')
scatter(mu1_tildaf,0,60,'*','blue')
scatter(mu2_tildaf,0,60,'*','red')
grid on
legend('C1 Data', 'C2 Data', 'C1 Projected Mean', 'C2 Projected Mean')
text(1,-0.6,sprintf('Theta = %0.2f', thetaFinal));
text(1,-0.7,sprintf('Mean-to-Scatter Ratio = %0.2f', maxVal));

hold off
```

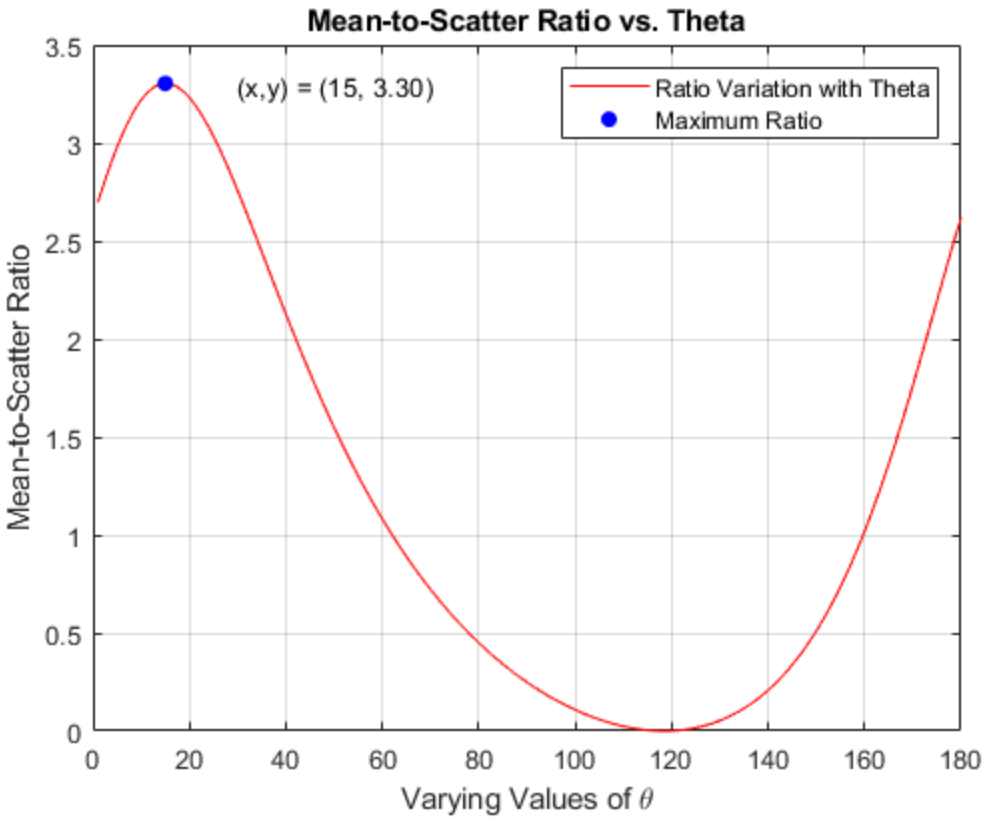




P3. Final Plot to Validate Solution

```
figure('Name', 'Plot to Check Final Solution')
plot(resultMatrix(:,1),resultMatrix(:,2),'red')
hold on
title('Mean-to-Scatter Ratio vs. Theta')
xlabel('Varying Values of \theta')
ylabel('Mean-to-Scatter Ratio')
scatter(index, maxVal, 'blue', 'filled');
grid on
legend('Ratio Variation with Theta', 'Maximum Ratio');
text(30,3.3,sprintf('(x,y) = (%i, %0.2f)', index, maxVal));


fprintf('X component of final vector v: %0.2f\n', vFinal(1));
fprintf('Y component of final vector v: %0.2f\n', vFinal(2));
fprintf('Final theta value: %0.2f\n', thetaFinal);
```

```
X component of final vector v: -0.60
Y component of final vector v: -0.16
Final theta value: 15.08
```



P3. Outputs

```
fprintf('\n')
fprintf('Mean of Class 1 before Projection:\n')
disp(mu1)

fprintf('\n')
fprintf('Mean of Class 2 before Projection:\n')
disp(mu2)

fprintf('\n')
fprintf('Mean of Class 1 after Projection:\n')
disp(mu1_tildaf)

fprintf('\n')
fprintf('Mean of Class 2 after Projection:\n')
disp(mu2_tildaf)

fprintf('\n')
fprintf('Within-Class Scatter:\n')
disp(sW)

fprintf('\n')
fprintf('Between-Class Scatter:\n')
disp(sB)
```

```
fprintf('\n')
fprintf('Final Direction Vector for Projection:\n')
disp(vFinal)

fprintf('\n')
fprintf('Final Projection Magnitudes:\n')
fprintf('Class 1.\n')
disp(Y_Mag1f)
fprintf('Class 2.\n')
disp(Y_Mag2f)

fprintf('\n')
fprintf('Final Projection Points (x,y):\n')
fprintf('Class 1.\n')
disp(Y_C1f)
fprintf('Class 2.\n')
disp(Y_C2f)
```

```
Mean of Class 1 before Projection:
    1.2000    3.0000


Mean of Class 2 before Projection:
    6.0000    5.6000


Mean of Class 1 after Projection:
    1.9392


Mean of Class 2 after Projection:
    7.2503


Within-Class Scatter:
    8.8000   -3.0000
   -3.0000   27.2000


Between-Class Scatter:
   23.0400   12.4800
   12.4800    6.7600


Final Direction Vector for Projection:
   -0.6006   -0.1618


Final Projection Magnitudes:
Class 1.
    0.5203
    1.2257
```

```
    2.0062
    2.7116
    3.2319

Class 2.
    6.3888
    6.9091
    6.5739
    7.6145
    8.7652


Final Projection Points (x,y):
Class 1.
    0.5024    0.1354
    1.1835    0.3189
    1.9371    0.5219
    2.6182    0.7055
    3.1207    0.8408

Class 2.
    6.1688    1.6621
    6.6712    1.7975
    6.3475    1.7103
    7.3523    1.9810
    8.4633    2.2804
```

P3. Concluding Remarks

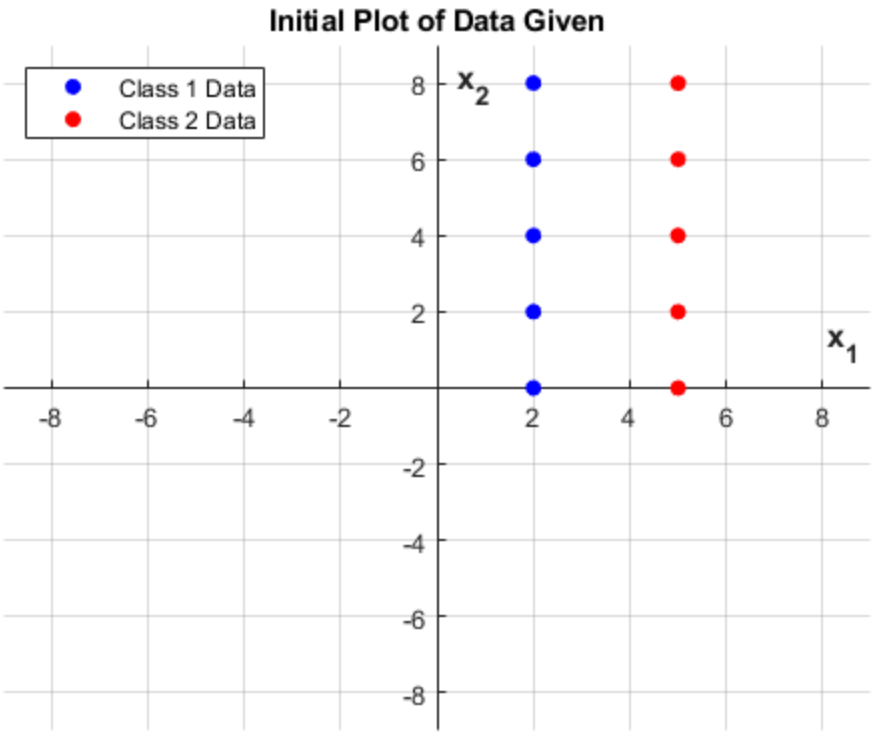
The code above is packed into a large for loop that iterates over all integer values of theta between 0 and 180. The reason for this is to obtain the final plot, **Mean-to-Scatter Ratio vs. Theta.** The algorithm itself computes a directional vector **v** which yields the direction of the line which best separates or classifies the given data. From this directional vector, the theta value or the angle above the x-axis can be computed. The for loop is in place so that the algorithm can run over all possible integer values of theta and the ratio for each theta can be computed. The final plot then shows the peak ratio which corresponds to some theta value. If that theta value matches the theta of the vector **v** obtained by the algorithm, this proves that the algorithm's chosen vector is the correct vector the maximize the mean-to-scatter ratio.

Bonus Problem on Linear Disciminant Analysis

Consider the following 2D dataset:

```
C1 = 'blue';
X_C1 = [2, 0;
        2, 2;
        2, 4;
        2, 6;
        2, 8];

C2 = 'red';
X_C2 = [5, 0;
        5, 2;
```

```
        5, 4;
        5, 6;
        5, 8];

% Declare and populate variables to store size of data matrices
[nRows1,nCols1] = size(X_C1);
[nRows2,nCols2] = size(X_C2);

% Plot the given data in a 2D plane
figure('Name', 'Plot of C1 and C2 Data Before LDA');
scatter(X_C1(:,1),X_C1(:,2),C1,'filled')
hold on
title('Initial Plot of Data Given');
xlabel('x_{1}', 'FontSize', 12, 'FontWeight', 'bold');
ylabel('x_{2}', 'FontSize', 12, 'FontWeight', 'bold');
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
xlim([-9 9]);
ylim([-9 9]);
scatter(X_C2(:,1),X_C2(:,2),C2,'filled')
grid on
legend('Class 1 Data', 'Class 2 Data', 'location', 'northwest')
hold off

% Compute the mean for each class

mu1 = mean(X_C1);
mu2 = mean(X_C2);
```



Bonus. Part A.

Find the direction V along which the two classes are best discriminated using the LDA. (Get the mu1, mu2, Sw, Sb, V, W, and y following the procedure in lecture 9).

```
% Initialize matrix to store thetas, J(v) ratios to check solution
resultMatrix = zeros(600,2);

x = linspace(-10, 10, 10000);

for theta = 1:600

    % obtain x, y components of a vector in the direction of theta
    [xv, yv, m] = GenerateLineFromAngle(theta);

    % declare vector v, the current direction vector
    v = [xv;
         yv];

    % Project C1, C2 points onto generated line

    % Generate matrices to hold the projected points' coordinates
    Y_C1 = zeros(5,2);
    Y_C2 = zeros(5,2);

    % Generate vectors to hold the projected points' magnitudes
```

```
Y_Mag1 = zeros(5,1);
Y_Mag2 = zeros(5,1);

% Populate the Y vector, matrix for class 1
for i = 1:nRows1

    Y_Mag1(i) = dot(transpose(v),X_C1(i,:));
    [xp1,yp1] = ProjectPoint(Y_Mag1(i),theta);
    Y_C1(i,1) = xp1;
    Y_C1(i,2) = yp1;

end

% Populate the Y vector, matrix for class 2
for j = 1:nRows2

    Y_Mag2(j) = dot(transpose(v),X_C2(j,:));
    [xp2,yp2] = ProjectPoint(Y_Mag2(j),theta);
    Y_C2(j,1) = xp2;
    Y_C2(j,2) = yp2;

end

% Compute projected mean for each class

mu1_tilda = dot(transpose(v),mu1);
mu2_tilda = dot(transpose(v),mu2);


% Compute scatter for each class

sProj1 = 0;
sProj2 = 0;

for i = 1:nRows1

    sProj1 = sProj1 + ((Y_Mag1(i) - mu1_tilda)^2);
    sProj2 = sProj2 + ((Y_Mag2(i) - mu2_tilda)^2);

end

% Normalize by both sProj1 and sProj2

J1 = (((mu1_tilda - mu2_tilda)^2)/(sProj1 + sProj2));


% Compute the separate class scatter matrices S1 and S2

s1 = zeros(2,2);
s2 = zeros(2,2);

for i = 1:nRows1

    s1 = s1 + (transpose((X_C1(i,:) - mu1)))*(X_C1(i,:) - mu1);
```

```
            s2 = s2 + (transpose((X_C2(i,:) - mu2)))*(X_C2(i,:) - mu2);

        end

        % create a small epsilon to give the matrices full rank
        epsilon = 0.001;

        s1 = s1 + epsilon;
        s2 = s2 + epsilon;


        % Define the within class & between class scatter matrix
        sW = s1 + s2;
        sB = transpose(mu1-mu2)*(mu1-mu2);

        % Compute alternate form of J ratio
        J2Num = transpose(v)*sB*v;
        J2Den = transpose(v)*sW*v;

        J2 = J2Num/J2Den;

        % Store final ratio in result matrix for checking solution
        resultMatrix(theta,1) = theta;
        resultMatrix(theta,2) = J2;
    end

    % After taking derivative of J and setting equal to zero, v can be
     found.
    vFinal = (mu1-mu2)/sW;
    thetaFinal = FindOptimalTheta(vFinal);
```

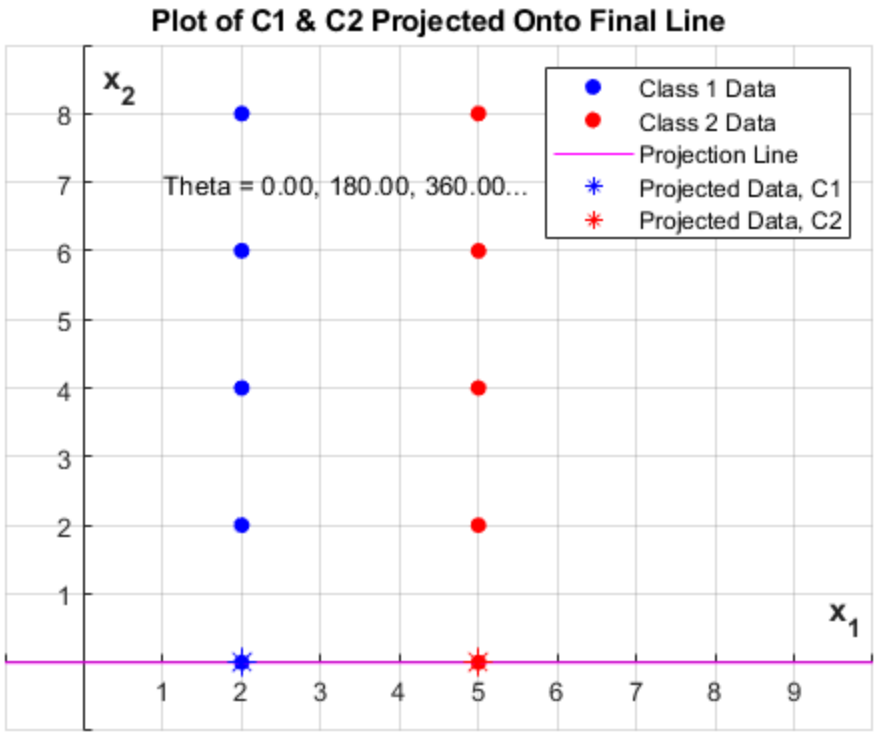
Bonus. Part B.

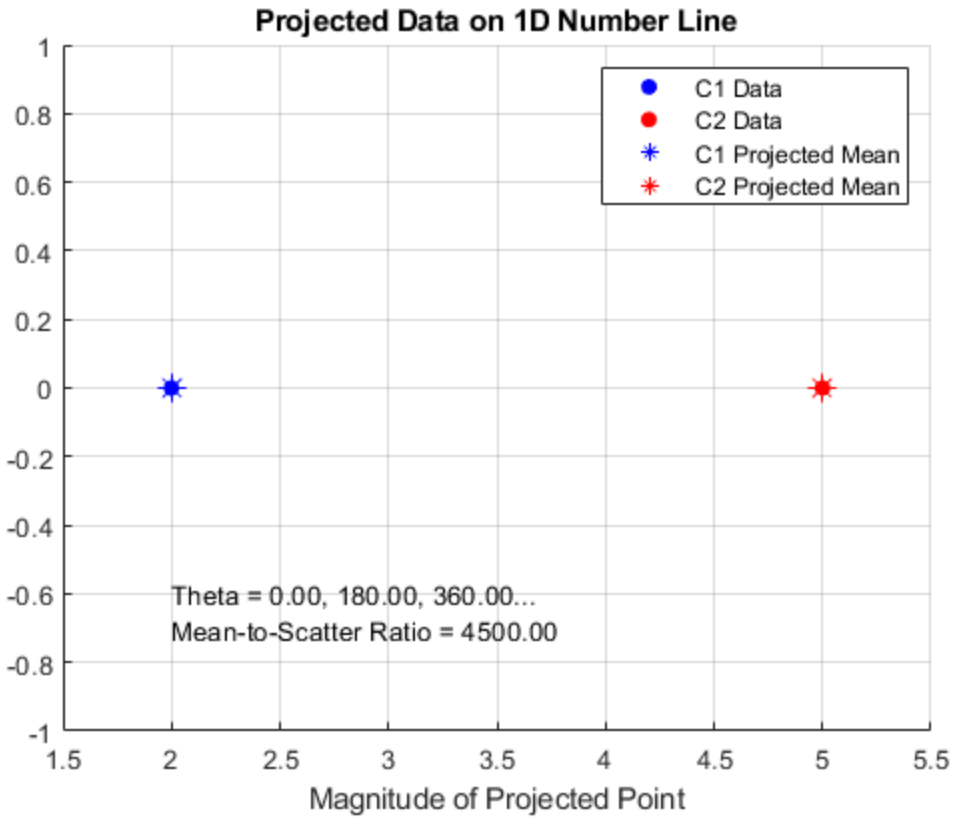
Plot the data before and after transformation using different colors for the two classes.

```
[maxVal, index] = max(resultMatrix(:,2));

% Generate matrices to hold the projected points' coordinates
Y_C1f = zeros(5,2);
Y_C2f = zeros(5,2);

% Generate vectors to hold the projected points' magnitudes
Y_Mag1f = zeros(5,1);
Y_Mag2f = zeros(5,1);

[xf, yf, mf] = GenerateLineFromAngle(thetaFinal);

vf = [xf;
      yf];

yOpt = mf*x;

% Compute final projected mean for each class
```

```
mu1_tildaf = dot(transpose(vf),mu1);
mu2_tildaf = dot(transpose(vf),mu2);

% Populate the Y vector, matrix for class 1 and optimal theta
for i = 1:nRows1

    Y_Mag1f(i) = dot(transpose(vf),X_C1(i,:));
    [xp1,yp1] = ProjectPoint(Y_Mag1f(i),thetaFinal);
    Y_C1f(i,1) = xp1;
    Y_C1f(i,2) = yp1;

end

% Populate the Y vector, matrix for class 2 and optimal theta
for j = 1:nRows2

    Y_Mag2f(j) = dot(transpose(vf),X_C2(j,:));
    [xp2,yp2] = ProjectPoint(Y_Mag2f(j),thetaFinal);
    Y_C2f(j,1) = xp2;
    Y_C2f(j,2) = yp2;

end

figure('Name', 'Projecting Data onto Final LDA Line');
scatter(X_C1(:,1),X_C1(:,2),C1,'filled')
hold on
title('Plot of C1 & C2 Projected Onto Final Line');
xlabel('x_{1}', 'FontSize', 12, 'FontWeight', 'bold');
ylabel('x_{2}', 'FontSize', 12, 'FontWeight', 'bold');
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
xlim([-1 10]);
ylim([-1 9]);
scatter(X_C2(:,1),X_C2(:,2),C2,'filled')
plot(x, yOpt, 'magenta')
L1 = 'Projection Line';
scatter(Y_C1f(:,1),Y_C1f(:,2),100,'*',C1)
P1 = 'Projected Data, C1';
scatter(Y_C2f(:,1),Y_C2f(:,2),100,'*',C2)
P2 = 'Projected Data, C2';
grid on
legend('Class 1 Data', 'Class 2 Data', L1, P1,
 P2,'location', 'northeast')
text(1,7,sprintf('Theta = %0.2f, %0.2f, %0.2f...', thetaFinal,
 thetaFinal+180, thetaFinal+360));
hold off


figure('Name', 'Plot of Projected Data on 1D Number Line')
scatter(Y_Mag1f, zeros(5,1), 'blue', 'filled')
title('Projected Data on 1D Number Line')
xlabel('Magnitude of Projected Point')
```

```
hold on
scatter(Y_Mag2f, zeros(5,1), 'red', 'filled')
scatter(mu1_tildaf,0,100,'*','blue')
scatter(mu2_tildaf,0,100,'*','red')
grid on
legend('C1 Data', 'C2 Data', 'C1 Projected Mean', 'C2 Projected Mean')
text(2,-0.6,sprintf('Theta = %0.2f, %0.2f, %0.2f...', thetaFinal,
 thetaFinal+180, thetaFinal+360));
text(2,-0.7,sprintf('Mean-to-Scatter Ratio = %0.2f', maxVal));

hold off
```

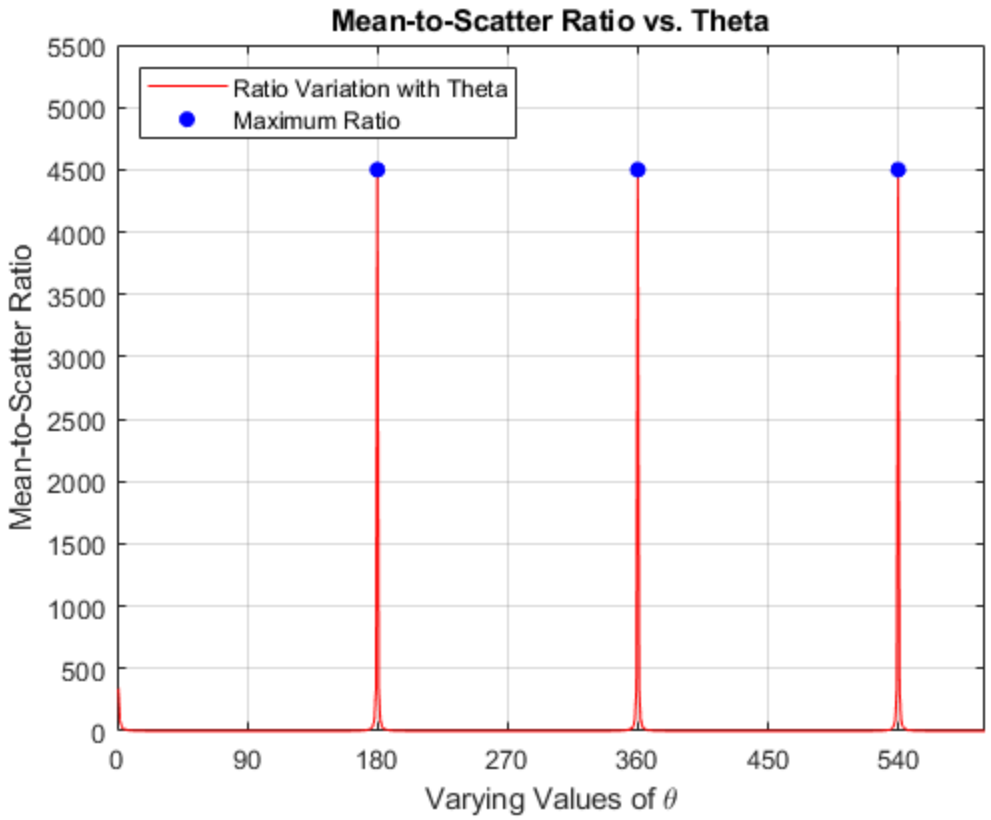




Bonus. Final Plot to Validate Solution

```
figure('Name', 'Plot to Check Final Solution')
plot(resultMatrix(:,1),resultMatrix(:,2),'red')
hold on
title('Mean-to-Scatter Ratio vs. Theta')
xlabel('Varying Values of \theta')
ylabel('Mean-to-Scatter Ratio')
xticks(0:90:720);
ylim([0 5500]);
scatter(index, maxVal, 'blue', 'filled');
scatter(index-180, maxVal, 'blue', 'filled');
scatter(index-360, maxVal, 'blue', 'filled');
grid on
legend('Ratio Variation with Theta', 'Maximum
 Ratio', 'location', 'nw');

fprintf('X component of final vector v: %0.2f\n', vFinal(1));
fprintf('Y component of final vector v: %0.2f\n', vFinal(2));
fprintf('Final theta value: %0.2f\n', thetaFinal);
```

```
X component of final vector v: -1500.04
Y component of final vector v: 0.04
Final theta value: 0.00
```



Bonus. Outputs

```
fprintf('\n')
fprintf('Mean of Class 1 before Projection:\n')
disp(mu1)

fprintf('\n')
fprintf('Mean of Class 2 before Projection:\n')
disp(mu2)

fprintf('\n')
fprintf('Mean of Class 1 after Projection:\n')
disp(mu1_tildaf)

fprintf('\n')
fprintf('Mean of Class 2 after Projection:\n')
disp(mu2_tildaf)

fprintf('\n')
fprintf('Within-Class Scatter:\n')
disp(sW)

fprintf('\n')
fprintf('Between-Class Scatter:\n')
disp(sB)
```

```
fprintf('\n')
fprintf('Final Direction Vector for Projection:\n')
disp(vFinal)

fprintf('\n')
fprintf('Final Projection Magnitudes:\n')
fprintf('Class 1.\n')
disp(Y_Mag1f)
fprintf('Class 2.\n')
disp(Y_Mag2f)

fprintf('\n')
fprintf('Final Projection Points (x,y):\n')
fprintf('Class 1.\n')
disp(Y_C1f)
fprintf('Class 2.\n')
disp(Y_C2f)
```

```
Mean of Class 1 before Projection:
     2     4


Mean of Class 2 before Projection:
     5     4


Mean of Class 1 after Projection:
    2.0001


Mean of Class 2 after Projection:
    5.0001


Within-Class Scatter:
    0.0020    0.0020
    0.0020   80.0020


Between-Class Scatter:
     9     0
     0     0


Final Direction Vector for Projection:
   1.0e+03 *

   -1.5000    0.0000


Final Projection Magnitudes:
Class 1.
```

```
    2.0000
    2.0000
    2.0001
    2.0001
    2.0002

Class 2.
    5.0000
    5.0000
    5.0001
    5.0001
    5.0002


Final Projection Points (x,y):
Class 1.
    2.0000    0.0000
    2.0000    0.0000
    2.0001    0.0001
    2.0001    0.0001
    2.0002    0.0001

Class 2.
    5.0000    0.0001
    5.0000    0.0001
    5.0001    0.0001
    5.0001    0.0001
    5.0002    0.0001
```

Bonus. Concluding Remarks

The bonus problem is similar to problem 4, except that a small epsilon value needs to be added to the scatter matrices in order to give them full rank (make their inverse matrices exist). By adding a small epsilon value to each component of the matrix, the matrix is invertible and we see that the data is best separated by a line with theta of 0, 180, 360, and all other multiples of 180. This is an intuitive result since the data in once class has all the same x values and the data in the other class has all the same x values. The best way to project two data sets who all have the same x values is to project them straight down onto the x axis. We can see from the final plot that the ratio reaches a maximum at every multiple of 180 degrees for theta.

Functions

```
function thetaOpt = FindOptimalTheta(v)

    magVfinal = sqrt(((v(1))^2)+((v(2))^2));
    arg = abs(v(1))/magVfinal;
    thetaOptRad = acos((arg));
    thetaOpt = rad2deg(thetaOptRad);

end

function [x, y, m] = GenerateLineFromAngle(theta)
```

```
    x = cos(deg2rad(theta));
    y = sin(deg2rad(theta));

    m = y/x;

end

function [xnew, ynew] = ProjectPoint(mag, theta)

    xnew = mag * cos(deg2rad(theta));
    ynew = mag * sin(deg2rad(theta));

end
```

Published with MATLAB® R2018b