## Table of Contents

# James Skelly, ECG703 Sp 2021 Final Exam

**Instructor: Dr. Latifi**

**Due Date: May 13, 2021**

```
close all
format compact
```

## Part 0: Importing and Plotting Given Data

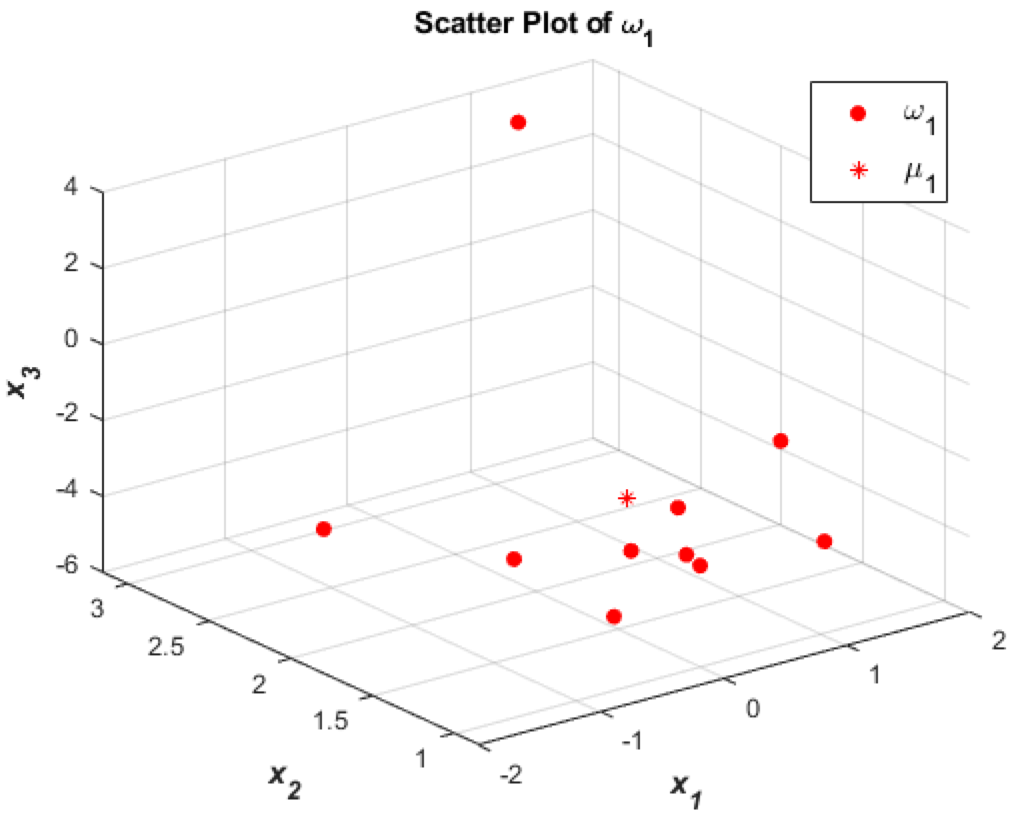
In this section, the given data for class 1, class 2, and class 3 was taken from the given PDF and input to an excel sheet. The **importfile** function was generated to extract each of the 3 features for each of the 3 classes into individual column vectors for use in the program.

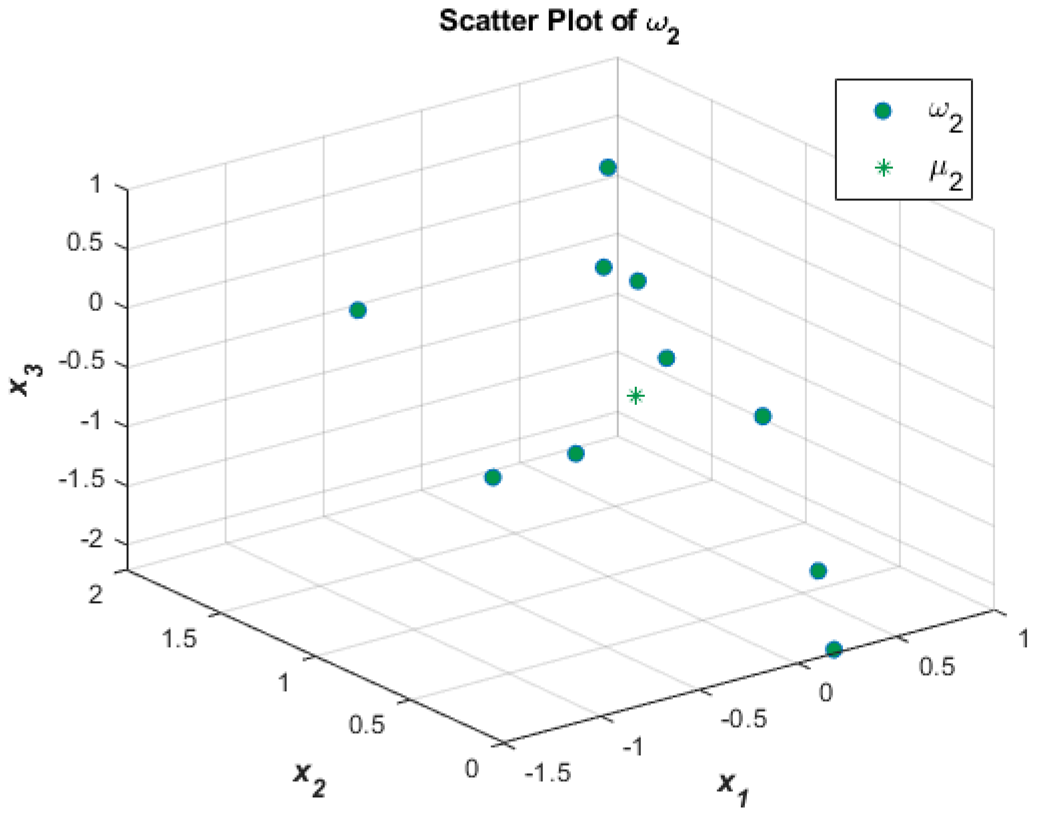
```
% Import Data from Excel Sheet
[x1w1, x2w1, x3w1, x1w2, x2w2, x3w2, x1w3, x2w3, x3w3]...
    = importfile('Data.xlsx');

% Separate imported data by class
w1 = [x1w1, x2w1, x3w1];
w2 = [x1w2, x2w2, x3w2];
w3 = [x1w3, x2w3, x3w3];

% Compute the mean of each class
mu1 = mean(w1);
mu2 = mean(w2);
mu3 = mean(w3);
```

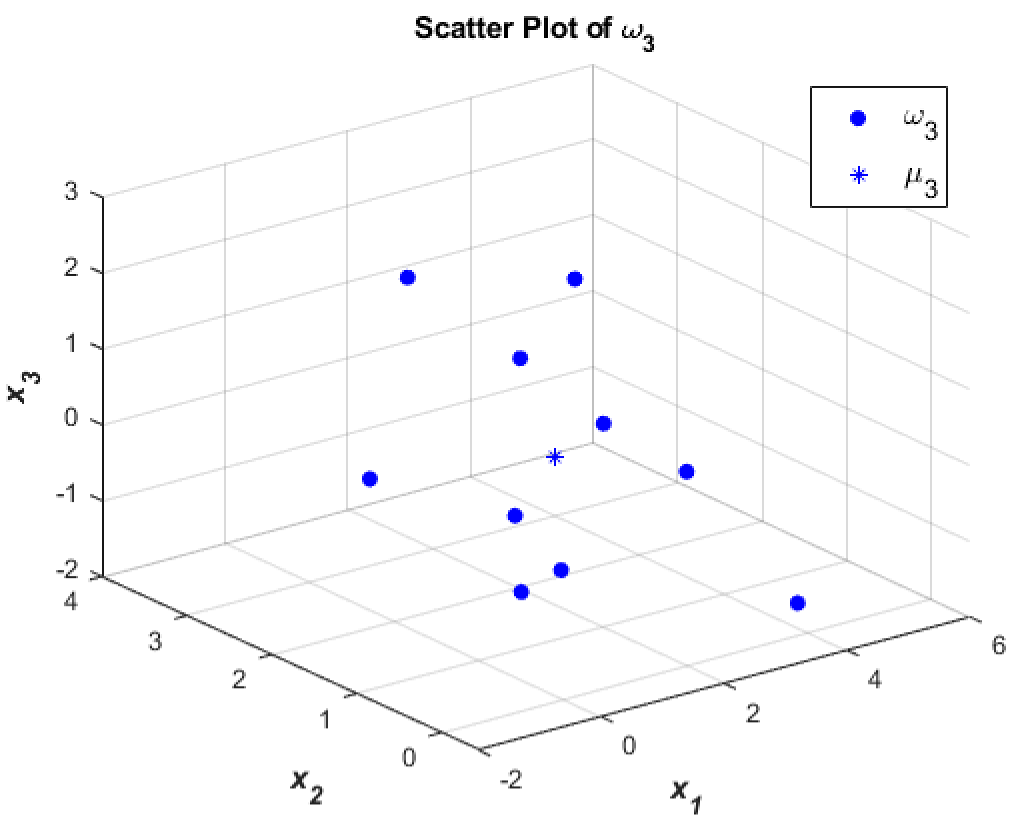
```
% Obtain centralized data by subtracting the mean from each class
w1_c = w1 - mu1;
w2_c = w2 - mu2;
w3_c = w3 - mu3;
```

## P0. Plotting

```
% 3D plot of given data for class 1
figure
scatter3(w1(:,1),w1(:,2),w1(:,3),'red','filled')
title('Scatter Plot of \omega_{1}')
xlabel('\itx_{1}', 'FontSize', 12, 'FontWeight', 'bold')
ylabel('\itx_{2}', 'FontSize', 12, 'FontWeight', 'bold')
zlabel('\itx_{3}', 'FontSize', 12, 'FontWeight', 'bold')
hold on
scatter3(mu1(1),mu1(2),mu1(3),'*', 'red')
legend('\omega_{1}','\mu_{1}',  'location', 'ne', 'FontSize', 12)

% 3D plot of given data for class 2
green = [0 0.6 0.3];
figure
scatter3(w2(:,1),w2(:,2),w2(:,3),'MarkerFaceColor',green)
title('Scatter Plot of \omega_{2}')
xlabel('\itx_{1}', 'FontSize', 12, 'FontWeight', 'bold')
ylabel('\itx_{2}', 'FontSize', 12, 'FontWeight', 'bold')
zlabel('\itx_{3}', 'FontSize', 12, 'FontWeight', 'bold')
hold on
scatter3(mu2(1),mu2(2),mu2(3),'*','MarkerEdgeColor',green)
legend('\omega_{2}','\mu_{2}', 'location', 'ne', 'FontSize', 12)

% 3D plot of given data for class 3
figure
scatter3(w3(:,1),w3(:,2),w3(:,3),'blue','filled')
title('Scatter Plot of \omega_{3}')
xlabel('\itx_{1}', 'FontSize', 12, 'FontWeight', 'bold')
ylabel('\itx_{2}', 'FontSize', 12, 'FontWeight', 'bold')
zlabel('\itx_{3}', 'FontSize', 12, 'FontWeight', 'bold')
hold on
scatter3(mu3(1),mu3(2),mu3(3),'*', 'blue')
legend('\omega_{3}', '\mu_{3}', 'location', 'ne', 'FontSize', 12)
```


Scatter Plot of $\omega_1$
$\omega_1$
$\mu_1$
$x_3$
$x_2$
$x_1$
4
2
0
-2
-4
-6
3
2.5
2
1.5
1
-2
-1
0
1
2


Scatter Plot of $\omega_2$
$\omega_2$
$\mu_2$
$x_3$
$x_2$
$x_1$
1
0.5
0
-0.5
-1
-1.5
-2
2
1.5
1
0.5
0
-1.5
-1
-0.5
0
0.5
1



## Part 1: Obtaining Correlation/Covariance Matrices

Obtain the correlation tables for the 3 features of each class of w1, w2, and w3. (20 pts)

```
% obtain the size of each class
[nRowsW1, nColsW1] = size(w1);
[nRowsW2, nColsW2] = size(w2);
[nRowsW3, nColsW3] = size(w3);

% compute the covariance matrices for each class
covW1 = (1/(nRowsW1-1))*(transpose(w1_c)*w1_c);
covW2 = (1/(nRowsW2-1))*(transpose(w2_c)*w2_c);
covW3 = (1/(nRowsW3-1))*(transpose(w3_c)*w3_c);

% obtain the correlation matrices for each class
corrW1 = corrcov(covW1);
corrW2 = corrcov(covW2);
corrW3 = corrcov(covW3);
```

## P1. Outputs

```
% Display Correlation, Covariance Matrices for class
fprintf('\n')
fprintf('Correlation Matrix for Class 1\n')
disp(corrW1)
fprintf('\n')
fprintf('Correlation Matrix for Class 2\n')
disp(corrW2)
fprintf('\n')
fprintf('Correlation Matrix for Class 3\n')
disp(corrW3)
fprintf('\n')
fprintf('Covariance Matrix for Class 1\n')
disp(covW1)
fprintf('\n')
fprintf('Covariance Matrix for Class 2\n')
disp(covW2)
fprintf('\n')
fprintf('Covariance Matrix for Class 3\n')
disp(covW3)
fprintf('\n')
```

```
Correlation Matrix for Class 1
    1.0000    0.3195    0.1492
    0.3195    1.0000    0.4815
    0.1492    0.4815    1.0000

Correlation Matrix for Class 2
    1.0000    0.1099   -0.0957
    0.1099    1.0000    0.6844
   -0.0957    0.6844    1.0000

Correlation Matrix for Class 3
    1.0000    0.4632   -0.3888
    0.4632    1.0000   -0.0354
   -0.3888   -0.0354    1.0000

Covariance Matrix for Class 1
    1.1842    0.2454    0.3946
    0.2454    0.4981    0.8261
    0.3946    0.8261    5.9101

Covariance Matrix for Class 2
    0.3025    0.0356   -0.0470
    0.0356    0.3479    0.3600
   -0.0470    0.3600    0.7953

Covariance Matrix for Class 3
    3.9972    1.2784   -1.1309
    1.2784    1.9055   -0.0711
   -1.1309   -0.0711    2.1171
```

## P1. Remarks

The covariance matrices are required in order to determine the eigen values and eigen vectors. Covariance matrices give us an idea of how much a change in one variable or feature results in change of another feature in a dataset. In this case, we only have three features, so our correlation and covariance matrices are 3x3. The covariance matrices are used to determine the eigen values and vectors in part 2 for the PCA method.

## Part 2: Performing PCA to Reduce Dimensionality

Use the PCA method to reduce the 3 dimensions for each class to 2 dimensions. Determine the eigen values and eigen vectors and map the 3D data to 2D data by picking 2 of the 3 features in each class. Show the new table but with 2 features for each class. (30 pts)

```
% Determine eigen values and eigen vectors from covariance matrices
[eigVectsW1, eigValsW1] = eig(covW1);
[eigVectsW2, eigValsW2] = eig(covW2);
[eigVectsW3, eigValsW3] = eig(covW3);

% Extract eigen vectors with the largest eigen values for PCA
[PC1_w1,PC2_w1] = GetPCs(eigValsW1,eigVectsW1);
[PC1_w2,PC2_w2] = GetPCs(eigValsW2,eigVectsW2);
[PC1_w3,PC2_w3] = GetPCs(eigValsW3,eigVectsW3);

% Set eigen vectors with largest eigen values as principal components
W1_EigVects = [PC1_w1, PC2_w1];
W2_EigVects = [PC1_w2, PC2_w2];
W3_EigVects = [PC1_w3, PC2_w3];

% Obtain vectors for plotting features with the greatest variance
p1x_w1 = [ 0 W1_EigVects(3,1)];
p1y_w1 = [ 0 W1_EigVects(3,2)];
length1_w1 = norm(W1_EigVects(3,1:2));
p2x_w1 = [ 0 W1_EigVects(2,1)];
p2y_w1 = [ 0 W1_EigVects(2,2)];
length2_w1 = norm(W1_EigVects(2,1:2));
p3x_w1 = [ 0 W1_EigVects(1,1)];
p3y_w1 = [ 0 W1_EigVects(1,2)];
length3_w1 = norm(W1_EigVects(1,1:2));
Lengths_w1 = [length3_w1,length2_w1,length1_w1];

p1x_w2 = [ 0 W2_EigVects(3,1)];
p1y_w2 = [ 0 W2_EigVects(3,2)];
length1_w2 = norm(W2_EigVects(3,1:2));
p2x_w2 = [ 0 W2_EigVects(2,1)];
p2y_w2 = [ 0 W2_EigVects(2,2)];
length2_w2 = norm(W2_EigVects(2,1:2));
p3x_w2 = [ 0 W2_EigVects(1,1)];
p3y_w2 = [ 0 W2_EigVects(1,2)];
length3_w2 = norm(W2_EigVects(1,1:2));
Lengths_w2 = [length3_w2,length2_w2,length1_w2];
```

```
p1x_w3 = [ 0 W3_EigVects(3,1)];
p1y_w3 = [ 0 W3_EigVects(3,2)];
length1_w3 = norm(W3_EigVects(3,1:2));
p2x_w3 = [ 0 W3_EigVects(2,1)];
p2y_w3 = [ 0 W3_EigVects(2,2)];
length2_w3 = norm(W3_EigVects(2,1:2));
p3x_w3 = [ 0 W3_EigVects(1,1)];
p3y_w3 = [ 0 W3_EigVects(1,2)];
length3_w3 = norm(W3_EigVects(1,1:2));
Lengths_w3 = [length3_w3,length2_w3,length1_w3];
```

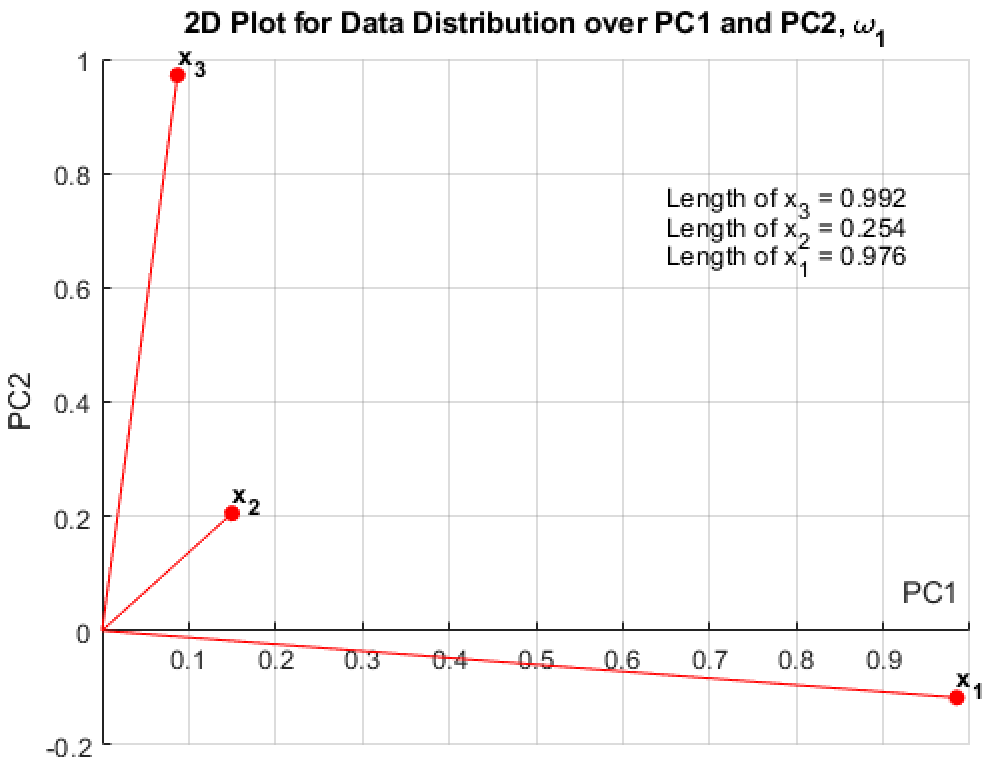
## P2. Outputs

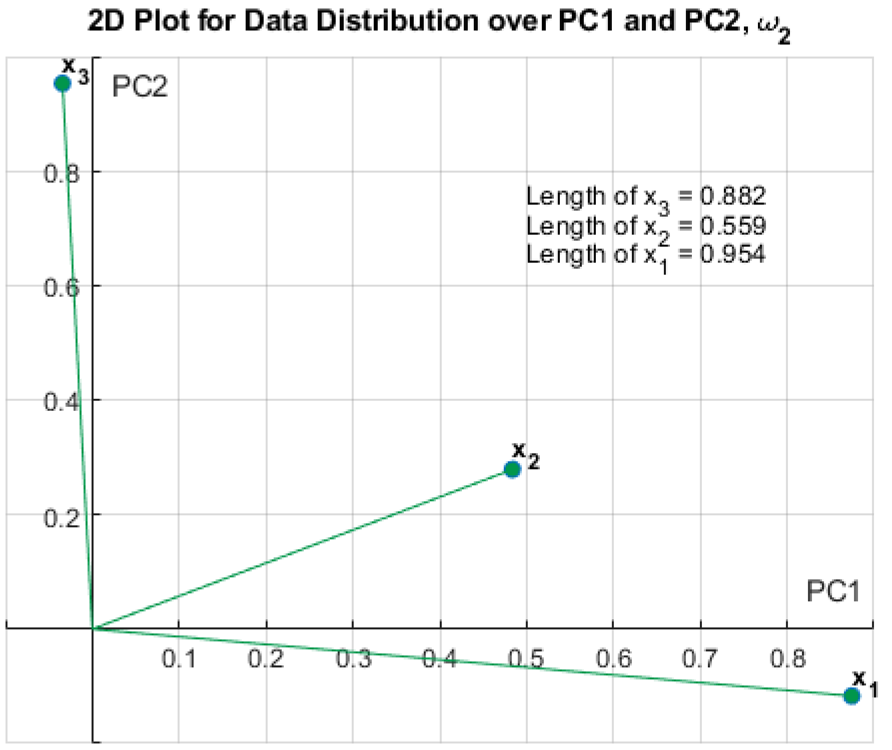
```
% Display Eigen Vectors, Eigen Values for each class
fprintf('\n')
fprintf('Eigen Vectors for Class 1\n')
disp(eigVectsW1)
fprintf('\n')
fprintf('Eigen Vectors for Class 2\n')
disp(eigVectsW2)
fprintf('\n')
fprintf('Eigen Vectors for Class 3\n')
disp(eigVectsW3)
fprintf('\n')
fprintf('Eigen Values for Class 1\n')
disp(eigValsW1)
fprintf('\n')
fprintf('Eigen Values for Class 2\n')
disp(eigValsW2)
fprintf('\n')
fprintf('Eigen Values for Class 3\n')
disp(eigValsW3)
fprintf('\n')
```

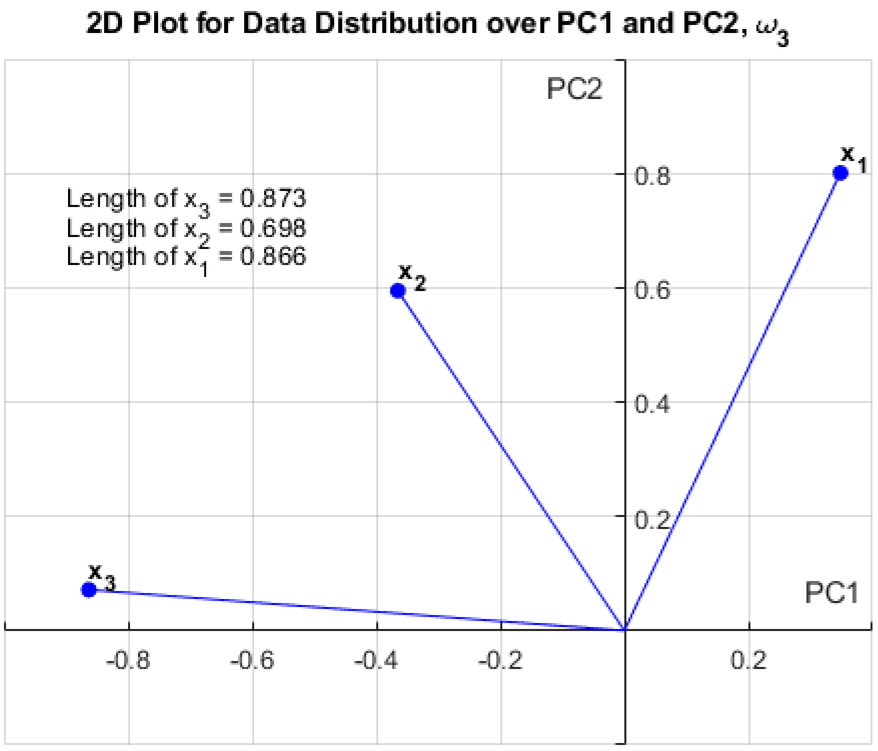
```

Eigen Vectors for Class 1
   -0.2197    0.9717    0.0871
    0.9672    0.2053    0.1498
   -0.1277   -0.1171    0.9849

Eigen Vectors for Class 2
   -0.3008    0.9531   -0.0343
    0.8295    0.2792    0.4837
   -0.4706   -0.1170    0.8745

Eigen Vectors for Class 3
    0.4997    0.0715   -0.8633
   -0.7162    0.5947   -0.3653
    0.4873    0.8008    0.3483

Eigen Values for Class 1
    0.3333         0         0
         0    1.1885         0
         0         0    6.0707

Eigen Values for Class 2
    0.1307         0         0
         0    0.3187         0
         0         0    0.9963

Eigen Values for Class 3
    1.0619         0         0
         0    1.9634         0
         0         0    4.9945
```

## P2. Plotting

```
% Plot of feature variance in direction of principal components, class
 1
figure
scatter(W1_EigVects(:,1),W1_EigVects(:,2),'red','filled')
title('2D Plot for Data Distribution over PC1 and PC2, \omega_{1}')
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
grid on
hold on
line(p1x_w1,p1y_w1, 'Color', 'red')
line(p2x_w1,p2y_w1, 'Color', 'red')
line(p3x_w1,p3y_w1, 'Color', 'red')
text(p1x_w1(2),p1y_w1(2)+0.03, 'x_{1}', 'FontWeight', 'bold')
text(p2x_w1(2),p2y_w1(2)+0.03, 'x_{2}', 'FontWeight', 'bold')
text(p3x_w1(2),p3y_w1(2)+0.03, 'x_{3}', 'FontWeight', 'bold')
text(0.65, 0.75, sprintf('Length of x_{3} = %0.3f', length1_w1))
text(0.65, 0.7, sprintf('Length of x_{2} = %0.3f', length2_w1))
text(0.65, 0.65, sprintf('Length of x_{1} = %0.3f', length3_w1))
xlabel('PC1')
ylabel('PC2')

% Plot of feature variance in direction of principal components, class
 2
figure
scatter(W2_EigVects(:,1),W2_EigVects(:,2),'MarkerFaceColor',green)
title('2D Plot for Data Distribution over PC1 and PC2, \omega_{2}')
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
grid on
hold on
line(p1x_w2,p1y_w2, 'Color', green)
line(p2x_w2,p2y_w2, 'Color', green)
line(p3x_w2,p3y_w2, 'Color', green)
text(p1x_w2(2),p1y_w2(2)+0.03, 'x_{1}', 'FontWeight', 'bold')
text(p2x_w2(2),p2y_w2(2)+0.03, 'x_{2}', 'FontWeight', 'bold')
text(p3x_w2(2),p3y_w2(2)+0.03, 'x_{3}', 'FontWeight', 'bold')
text(0.5, 0.75, sprintf('Length of x_{3} = %0.3f', length1_w2))
text(0.5, 0.7, sprintf('Length of x_{2} = %0.3f', length2_w2))
text(0.5, 0.65, sprintf('Length of x_{1} = %0.3f', length3_w2))
xlabel('PC1')
ylabel('PC2')

% Plot of feature variance in direction of principal components, class
 3
figure
scatter(W3_EigVects(:,1),W3_EigVects(:,2),'blue','filled')
title('2D Plot for Data Distribution over PC1 and PC2, \omega_{3}')
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
grid on
hold on
line(p1x_w3,p1y_w3, 'Color', 'blue')
line(p2x_w3,p2y_w3, 'Color', 'blue')
line(p3x_w3,p3y_w3, 'Color', 'blue')
text(p1x_w3(2),p1y_w3(2)+0.03, 'x_{1}', 'FontWeight', 'bold')
text(p2x_w3(2),p2y_w3(2)+0.03, 'x_{2}', 'FontWeight', 'bold')
text(p3x_w3(2),p3y_w3(2)+0.03, 'x_{3}', 'FontWeight', 'bold')
text(-0.9, 0.75, sprintf('Length of x_{3} = %0.3f', length1_w3))
text(-0.9, 0.7, sprintf('Length of x_{2} = %0.3f', length2_w3))
text(-0.9, 0.65, sprintf('Length of x_{1} = %0.3f', length3_w3))
xlabel('PC1')
ylabel('PC2')
ylim([-0.2 1])
```


2D Plot for Data Distribution over PC1 and PC2, $\omega_1$
Length of $x_3$ = 0.992
Length of $x_2$ = 0.254
Length of $x_1$ = 0.976
$x_3$
$x_2$
$x_1$
PC1
PC2
1
0.8
0.6
0.4
0.2
0
-0.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


2D Plot for Data Distribution over PC1 and PC2, $\omega_2$
Length of $x_3$ = 0.882
Length of $x_2$ = 0.559
Length of $x_1$ = 0.954
$x_3$
$x_2$
$x_1$
PC1
PC2
0.8
0.6
0.4
0.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8



## P2. Remarks

By examining the magnitudes of each feature (x1, x2, and x3) of each class in the plot of principal component 2 vs. principal component 1, we can determine which feature is the least relevant and eliminate that feature to reduce the number of dimensions in our data from 3 to 2. From the figures of this section, we can observe that the magnitude of the x2 feature in all three classes is the least relevent. Therefore, we can eliminate this feature without losing much information, and operate on 2D data for our SVM classification in part 3.

## Part 3: Soft-Margin SVM for Classification

Derive a SVM solution (specify the boundary and the margin) for dividing the following classes. Make sure to check the data separability for each case and specify whether you used Hard SVM or Soft SVM. Plot the data for each case, show the decision boundaries and mark the support vectors. (30 pts)

```
% Set class1, class2, and class3 data matrices with feature x2 removed
class1 = [x1w1,x3w1];
class2 = [x1w2,x3w2];
class3 = [x1w3,x3w3];

% Obtain the weights for the best boundary line using soft margin SVM
[w1_1,w2_1,bias1,m1,dy1,sv1] = GetWeights(class1,class2);
[w1_2,w2_2,bias2,m2,dy2,sv2] = GetWeights(class1,class3);
[w1_3,w2_3,bias3,m3,dy3,sv3] = GetWeights(class2,class3);
```

```
% Set the x-axis linear space and y values for plots of each boundary
 line
xplot = -30:0.05:30;
yplot1 = (-1*(w1_1/w2_1)*xplot)-(bias1/w2_1);
yplot2 = (-1*(w1_2/w2_2)*xplot)-(bias2/w2_2);
yplot3 = (-1*(w1_3/w2_3)*xplot)-(bias3/w2_3);

xp = [xplot, fliplr(xplot)];
inBetween1 = [yplot1-dy1, fliplr(yplot1+dy1)];
inBetween2 = [yplot2-dy2, fliplr(yplot2+dy2)];
inBetween3 = [yplot3-dy3, fliplr(yplot3+dy3)];
```

## P3. Outputs

```
% Display margins, support vectors for each SVM solution
fprintf('\n')
fprintf('Margin for Class 1/Class 2\n')
disp(m1)
fprintf('\n')
fprintf('Support Vectors for Class 1/Class 2\n')
disp(sv1)
fprintf('\n')
fprintf('Margin for Class 1/Class 3\n')
disp(m2)
fprintf('\n')
fprintf('Support Vectors for Class 1/Class 3\n')
disp(sv2)
fprintf('\n')
fprintf('Margin for Class 2/Class 3\n')
disp(m3)
fprintf('\n')
fprintf('Support Vectors for Class 2/Class 3\n')
disp(sv3)
fprintf('\n')
```

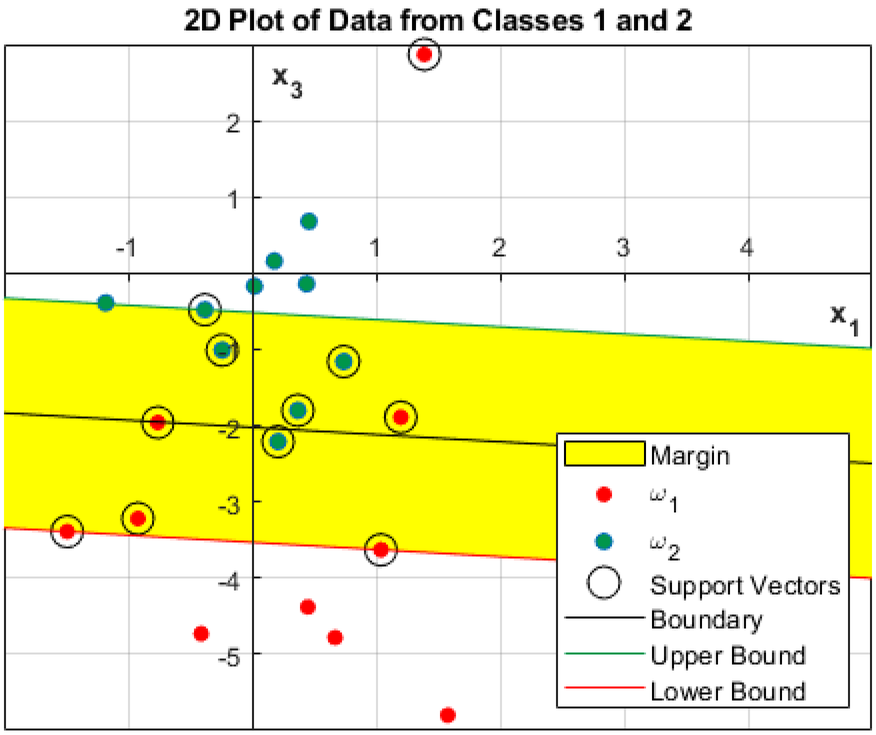
```

Margin for Class 1/Class 2
    1.5013

Support Vectors for Class 1/Class 2
    1.0400   -3.6300
   -1.4900   -3.3900
    1.3900    2.8700
    1.2000   -1.8900
   -0.9200   -3.2200
   -0.7600   -1.9600
    0.2100   -2.2100
    0.3700   -1.8000
   -0.2400   -1.0100
    0.7400   -1.1600
   -0.3800   -0.4800

Margin for Class 1/Class 3
```

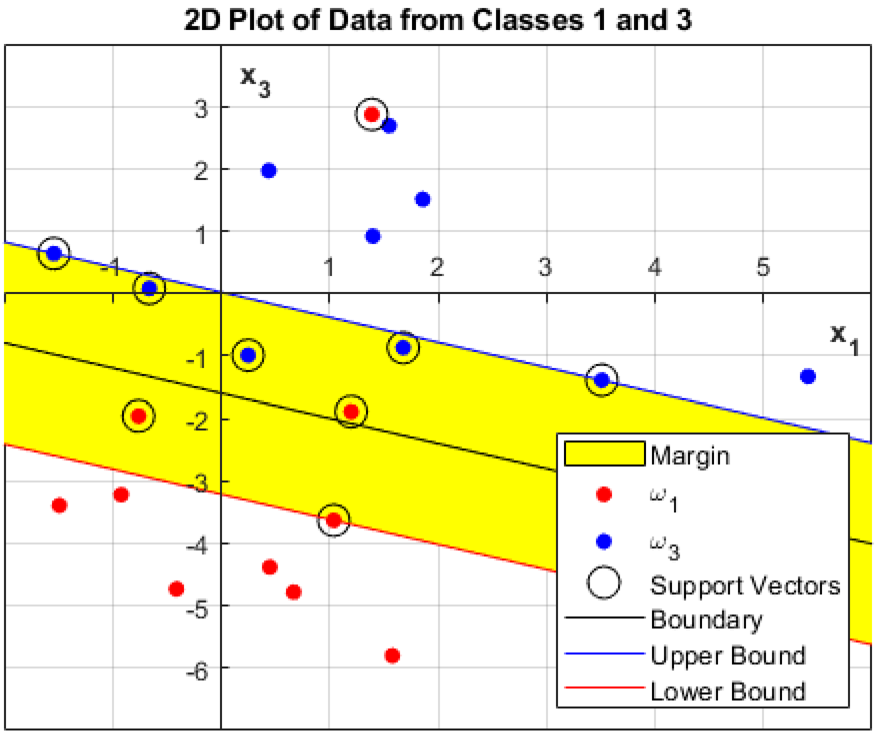
```
    1.4999

Support Vectors for Class 1/Class 3
    1.0400   -3.6300
    1.3900    2.8700
    1.2000   -1.8900
   -0.7600   -1.9600
   -1.5400    0.6400
    1.6800   -0.8700
    3.5100   -1.3900
    0.2500   -0.9900
   -0.6600    0.0800

Margin for Class 2/Class 3
    1.1295

Support Vectors for Class 2/Class 3
    0.1800    0.1600
    0.7400   -1.1600
   -0.3800   -0.4800
    0.0200   -0.1700
    0.4400   -0.1400
    0.4600    0.6800
   -1.5400    0.6400
    1.6800   -0.8700
    3.5100   -1.3900
    1.4000    0.9200
    0.4400    1.9700
    0.2500   -0.9900
   -0.6600    0.0800
```

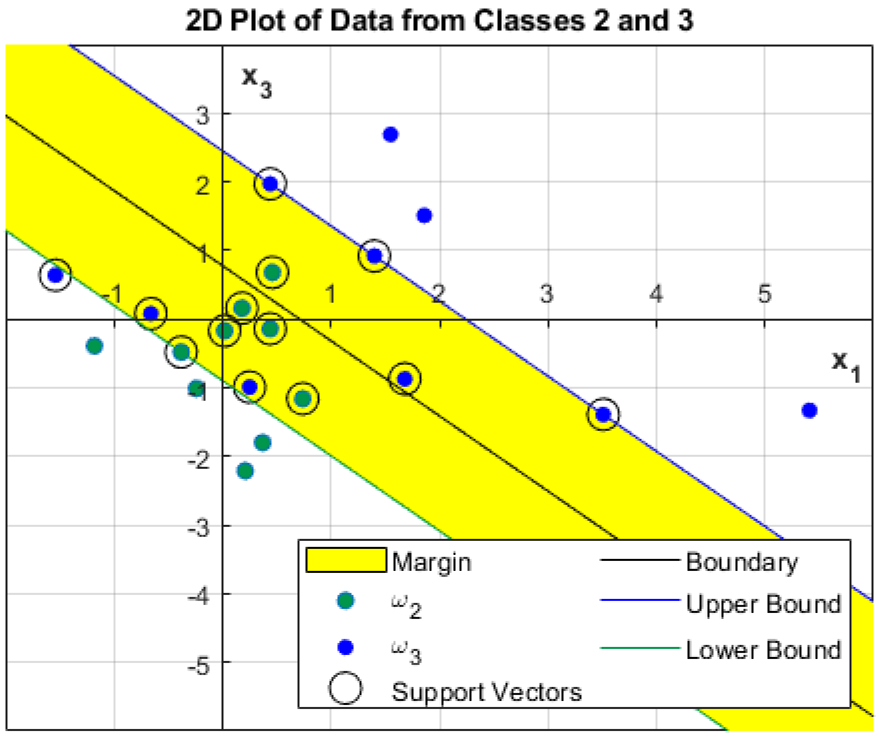
## P3. Plotting

```
% 1. SVM Plot for classes 1 and 2
figure
fill(xp,inBetween1,'yellow');
hold on
scatter(x1w1,x3w1,'red', 'filled')
title('2D Plot of Data from Classes 1 and 2')
xlim([-2 5])
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
scatter(x1w2,x3w2,'MarkerFaceColor',green)
xlabel('x_{1}','FontWeight','Bold')
ylabel('x_{3}','FontWeight','Bold')
plot(sv1(:,1),sv1(:,2),'ko','MarkerSize',12)
plot(xplot,yplot1,'black')
plot(xplot,yplot1+dy1,'Color',green)
plot(xplot,yplot1-dy1,'red')
grid on
```

```
legend('Margin','\omega_{1}', '\omega_{2}','Support
 Vectors','Boundary','Upper Bound',...
    'Lower Bound','FontSize',10,'location','se')


% 2. SVM Plot for classes 1 and 3
figure
fill(xp,inBetween2,'yellow');
hold on
scatter(x1w1,x3w1,'red', 'filled')
title('2D Plot of Data from Classes 1 and 3')
xlim([-2 6])
ylim([-7 4])
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
scatter(x1w3,x3w3,'blue', 'filled')
xlabel('x_{1}','FontWeight','Bold')
ylabel('x_{3}','FontWeight','Bold')
plot(sv2(:,1),sv2(:,2),'ko','MarkerSize',12)
plot(xplot,yplot2,'black')
plot(xplot,yplot2+dy2,'blue')
plot(xplot,yplot2-dy2,'red')
grid on
legend('Margin','\omega_{1}', '\omega_{3}','Support
 Vectors','Boundary','Upper Bound',...
    'Lower Bound','FontSize',10,'location','se')

% 3. SVM Plot for classes 2 and 3
figure
fill(xp,inBetween3,'yellow');
hold on
scatter(x1w2,x3w2,'MarkerFaceColor',green)
title('2D Plot of Data from Classes 2 and 3')
ylim([-6 4])
xlim([-2 6])
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
scatter(x1w3,x3w3,'blue', 'filled')
xlabel('x_{1}','FontWeight','Bold')
ylabel('x_{3}','FontWeight','Bold')
plot(sv3(:,1),sv3(:,2),'ko','MarkerSize',12)
plot(xplot,yplot3,'black')
plot(xplot,yplot3+dy3,'blue')
plot(xplot,yplot3-dy3,'Color',green)
grid on
legend('Margin','\omega_{2}', '\omega_{3}','Support
 Vectors','Boundary','Upper Bound',...
    'Lower Bound','FontSize',10,'NumColumns',2,'location','se')
```


2D Plot of Data from Classes 1 and 2
x3
x1
Margin
ω1
ω2
Support Vectors
Boundary
Upper Bound
Lower Bound


2D Plot of Data from Classes 1 and 3
x3
x1
Margin
ω1
ω3
Support Vectors
Boundary
Upper Bound
Lower Bound



## P3. Remarks

In this section, the weights of the boundary line are obtained using soft-margin SVM. The support vectors are circled in each of the three plots. Support vectors for a soft-margin SVM are defined as any points on the edges, inside the margins, or misclassified points. There are far more support vectors in a soft-margin SVM than there are in a hard-margin SVM. The reason why we need to use soft-margin SVM is because the data is not absolutely linearly separable. We need to allow for misclassifications in order for our boundary line to be the best classifier. Thus, soft-margin SVM is used for each of the 3 cases in the plots above from part 3.

## Part 4. Organization, Clarity, Readability Note

Organization, clarity and readability of the solution (see the posted solutions to homework assignments). (20 pts)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% NO CODE NEEDED FOR THIS SECTION %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

## Summary of Process and Final Remarks

The data was first taken from the PDF given and input into an excel file. An **importfile** function was used to pull the data from excel into MATLAB and each feature for each class was stored as a separate column vector. The data was then concatenated by class to form three distinct matrices, one for each of the three classes. Once the data was separated by class, the correlation and covariance matrices were computed for each class. Each covariance matrix was then used to obtain the eigen values and eigen vectors for the class datasets. Using these eigen values and eigen vectors, the principal components were determined, and the feature which contributed the least variance to the data was eliminated. This feature happened to be feature 2 (x2) so x2 was removed, leaving features x1 and x3 behind to form 2-dimensional data.

Once the 2-dimensional data was pulled, soft-margin support vector machine solutions were obtained to best separate the data. Soft-margin SVM's were necessary since the data was not absolutely linearly separable. In soft-margin SVM solutions, the support vectors are defined as all misclassified data points, all data within the margins, and all data lying on the edge of the margins. These data points were circled, and are labeled **support vectors.** The margins were also filled in with yellow. The final margins and list of support vectors were displayed in the final publication above.

In conclusion, the soft-margin SVM is much more useful than the hard-margin SVM because data in real life is more likely to overlap thanks to noise and other parameters. The hard-margin SVM is a special case which is not as useful because data is not typically linearly separable in the real world.

## Functions

```
function [PC1,PC2] = GetPCs(eigVals,eigVects)

    [vals,~] = max(eigVals);
    [~, index1] = max(vals);
    vals(index1) = 0;
    [~, index2] = max(vals);
    PC1 = eigVects(:,index1);
    PC2 = eigVects(:,index2);

end

function [w1,w2,b,m,dy,sv] = GetWeights(c1,c2)

    % Create column for y values in class data
    c1(:,3) = ones(10,1);
    c2(:,3) = -1*ones(10,1);

    % Generate data matrix
    data = [c1;c2];

    % Create SVM solution for binary classification using Quadratic
    % Programming
    mdl = quadprog2(data(:,1:2),data(:,3));

    % Extract weights from SVM solution
    w1 = mdl.Beta(1);
    w2 = mdl.Beta(2);
    b = mdl.Bias;

    % Compute the margin
    m = 1/norm(mdl.Beta);

    % Compute the offset for the support vectors from the boundary
 line
    theta1 = rad2deg(atan(mdl.Beta(1)/mdl.Beta(2)));
    theta2 = 90 - theta1;
    dy = m/(sin(deg2rad(theta2)));

    sv = mdl.SupportVectors;

end
```

*Published with MATLAB® R2018b*