Name: Sayed Sohail Pasha Peerzade

Student Number: 220541549

Assignment Number: Assignment 2 (Density Estimation)

Module Code: ECS708U/ECS708P

Report for Assignment 2 (Density Estimation)

Q1. Produce a plot of F1 against $F2$ (You should be able to spot some clusters already in this scatter plot.). Comment on the figure and the visible clusters [2 marks]

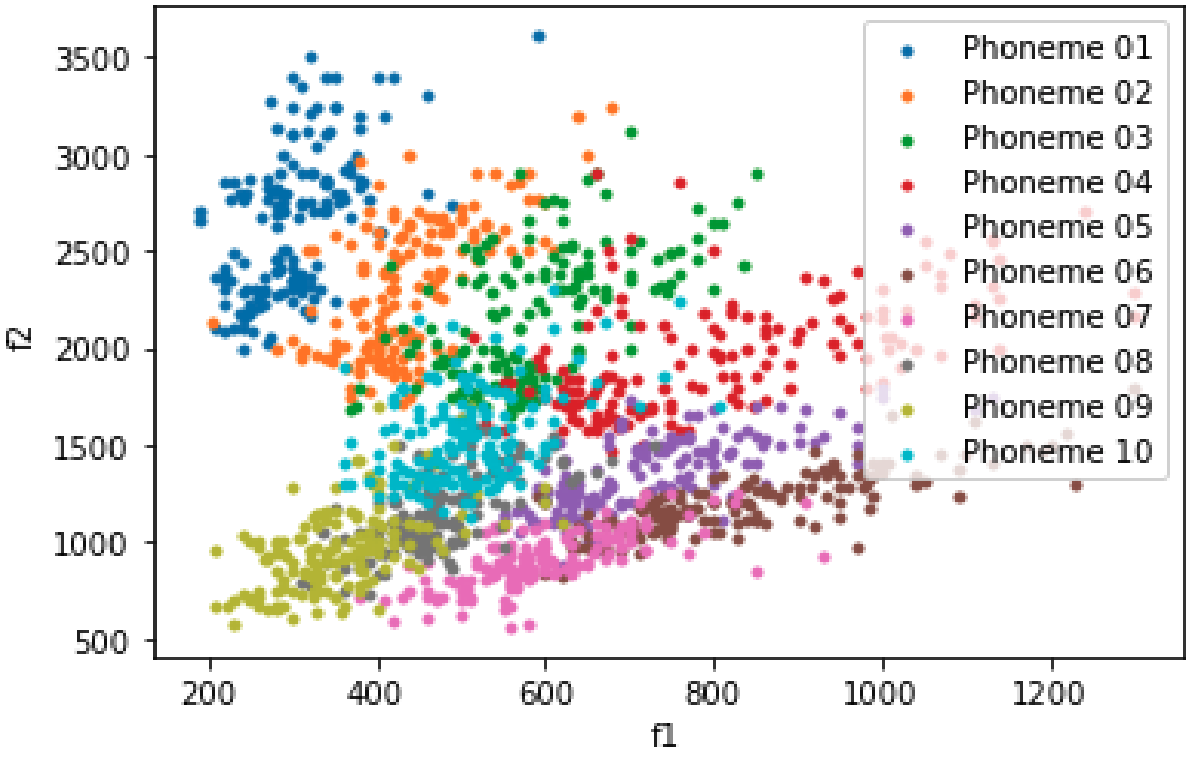


Figure 1: Plot of fundamental frequencies, F1 against F2 for all Phonemes

From the above plot of fundamental frequencies F1 against F2 it can be inferred that the plots for most of the phonemes lie isolated without overlapping with other phonemes, hence most of the phonemes may be easily separated.

Phoneme 1 (Blue) and Phoneme 2(orange) can be easily separated as they do not have areas overlapping the other phonemes. These sounds can be distinguished even without using the other fundamental frequencies.

However we do see some considerable overlapping between between Phoneme 9(Green) and Phoneme 8 (Gray), Phoneme 6 (brown) and Phoneme (07), Phoneme 3 (Green) and Phoneme 4 (Red) and Phoneme 3 (Green) and Phoneme 10 (Sky Blue). To distinguish these sounds we may require other fundamental frequencies (F3, F4)

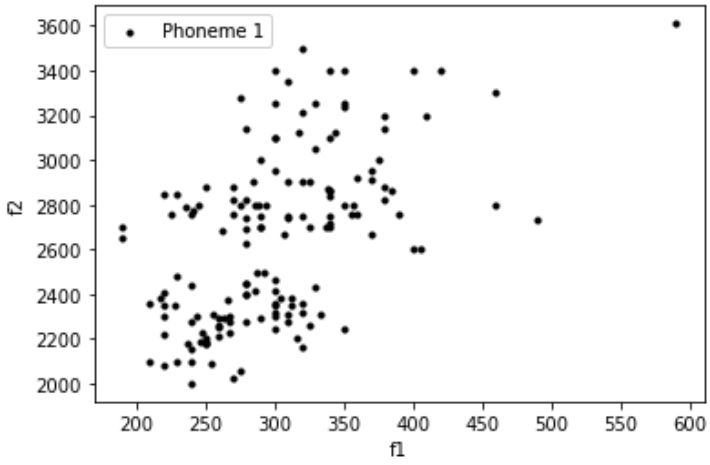


Figure 2: Plot of Fundamental frequencies, F1 against F2 for Phoneme 1

Q2. Run the code multiple times for K=3, what do you observe? Use figures and the printed MoG parameters to support your arguments [2 mark]

The MoG is trained using the Estimation Maximization algorithm for 100 Iterations. Before the EM algorithm the Guasssian parameters are initialized to random values. Mean (mu) is picked randomly from data samples, Covariance Matrix (s) is initialized with zeros and Weights are initially set as 1/k.

The EM step of the algorithm starts by soft assignment of these values (mu, s, p) by finding the lower bound of the log-likeihood.

In the Maximization step, these Guassian parameters which were set to a lower bound are then further maximized.

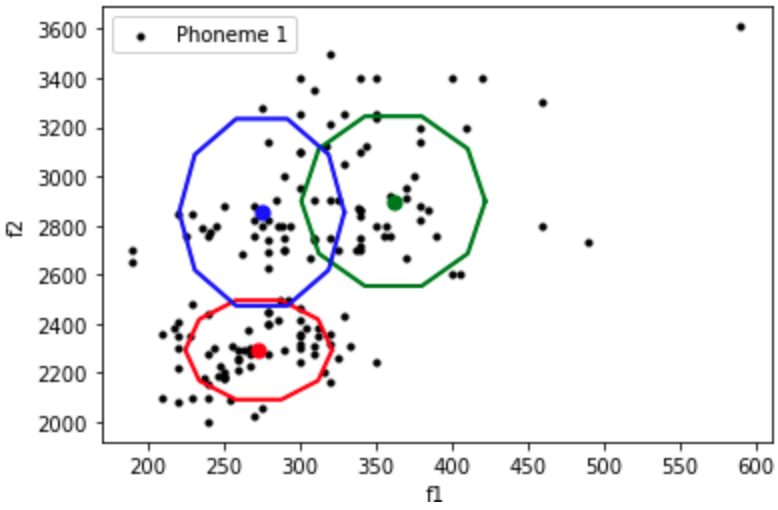
By running the code for 100 iterations we can see that,

Iteration 1:

```
Iteration 001/100
Mean values
[ 276.29565 2340.5652 ]
[ 319.56888 2990.8176 ]
[ 332.79654 2926.1926 ]

Covariances
[[ 1880.70637029     0.        ]
 [    0.         32791.62899211]]
[[ 3553.55352272     0.        ]
 [    0.         59908.26722447]]
[[ 4304.0416827      0.        ]
 [    0.         51825.66751681]]

Weights
[0.50123337 0.24291949 0.25584714]
```



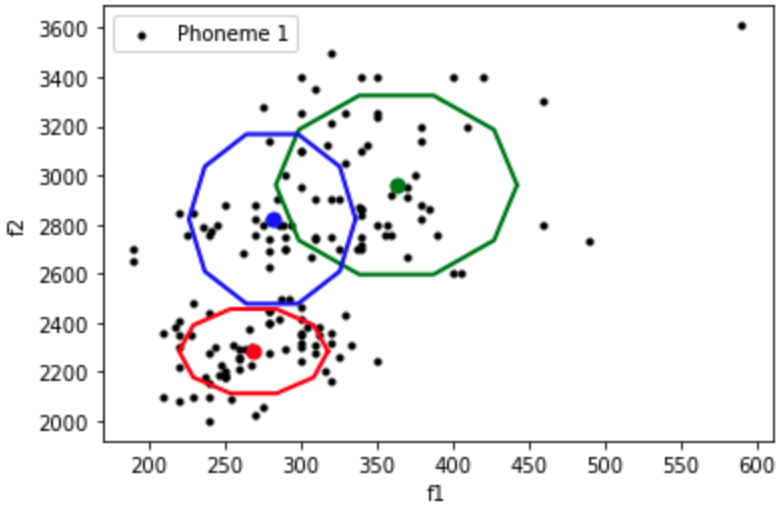
In the first iteration the Guassian parameters (mu, s, p) are set by soft initialization. We can see that there is overlapping in the classifications. Blue, red and green covariances are being overlapped.

Iteration 2:

```
Iteration 002/100
Mean values
[ 270.48697 2328.945  ]
[ 323.63455 2967.935  ]
[ 334.686  2913.9883]

Covariances
[[ 1318.20265087     0.        ]
 [    0.         30869.60340064]]
[[ 3064.91999667     0.        ]
 [    0.         63739.10171695]]
[[ 4701.4559361      0.        ]
 [    0.         57809.3108181]]

Weights
[0.47726145 0.25183261 0.27090594]
```



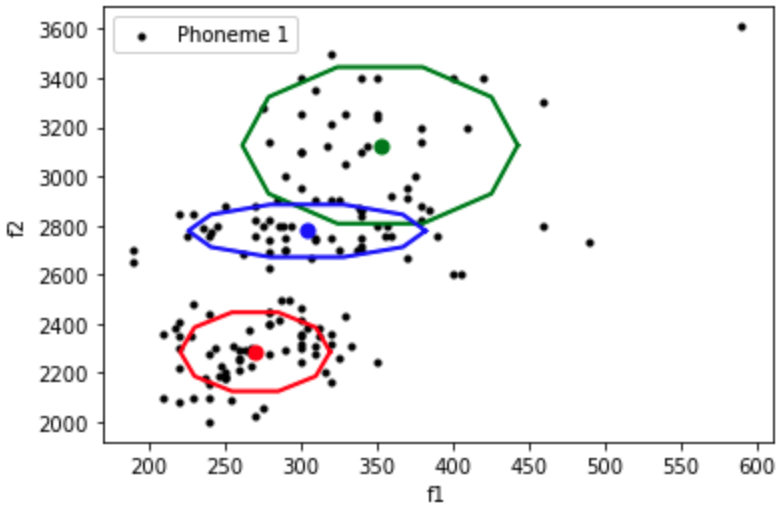
In iteration 2 the Maximisation step of the algorithm starts to update the Gaussian parameters to fit the model more likely. We can see that the red circle has started to separate from the other two circles.

Iteration i-th:

```
Mean values
[ 269.4754 2317.3953]
[ 321.99216 2949.4905 ]
[ 334.25482 2911.534  ]

Covariances
[[ 1233.83938569     0.         ]
 [    0.         27286.16209791]]
[[ 2466.09131098     0.         ]
 [    0.         62541.31855282]]
[[ 5253.33601148     0.         ]
 [    0.         63440.16600303]]

Weights
[0.45969617 0.26199899 0.27830484]
```



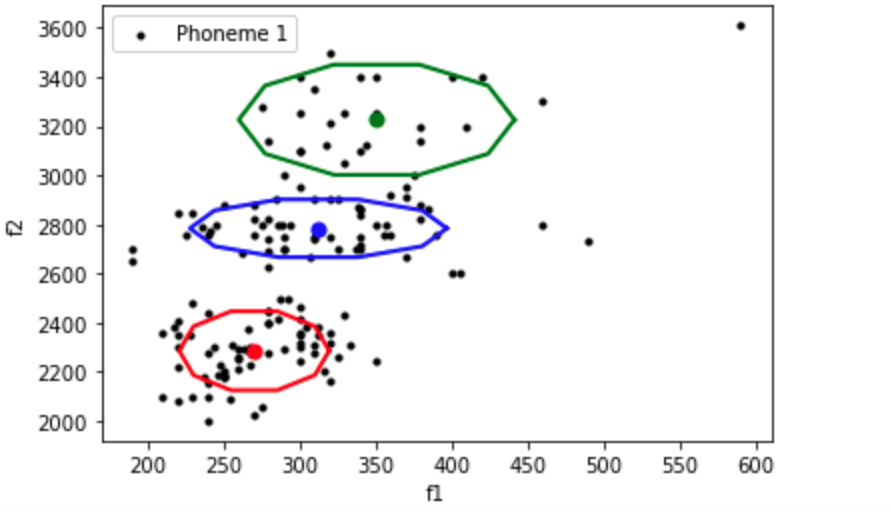
After running the model for some i iterations we can clearly see that the clusters have started to separate from one and another.

Final:

```
Iteration 100/100
Mean values
[ 270.3952 2285.4653]
[ 312.59125 2783.898  ]
[ 350.8446 3226.3394]

Covariances
[[ 1213.73843494     0.        ]
 [    0.         14278.42029989]]
[[3562.59743769    0.        ]
 [   0.          7657.84896997]]
[[ 4102.87537456     0.        ]
 [    0.         27829.54223973]]

Weights
[0.43514434 0.38099528 0.18386038]
```



After running the model for 100 iterations we can clearly see that the clusters have separated from each other and the data has been classified.

Q5. Use the 2 MoGs (K=3) learnt in tasks 2 & 3 to build a classifier to discriminate between phonemes 1 and 2, and explain the process in the report [4 marks] ¶

To discriminate between Phonemes 1 and Phonemes 2:

1. We first extract GMM parameters from the respective files for both the phonemes

```
npy_filename = 'data/GMM_params_phoneme_{:02}_k_{:02}.npy'.format(p_id, k)
if os.path.isfile(npy_filename):
    GMM_parameters_phoneme_1 = np.load(npy_filename, allow_pickle=True)
    GMM_parameters_phoneme_1 = np.ndarray.tolist(GMM_parameters_phoneme_1)
else:
    raise('File {} does not exist.'.format(npy_filename))
mu = GMM_parameters_phoneme_1['mu']
s = GMM_parameters_phoneme_1['s']
p = GMM_parameters_phoneme_1['p']
```

```
npy_filename2 = 'data/GMM_params_phoneme_{:02}_k_{:02}.npy'.format(p_id, k)
GMM_parameters_phoneme_2 = np.load(npy_filename2, allow_pickle=True)
GMM_parameters_phoneme_2 = np.ndarray.tolist(GMM_parameters_phoneme_2)
mu_2 = GMM_parameters_phoneme_2['mu']
s_2 = GMM_parameters_phoneme_2['s']
p_2 = GMM_parameters_phoneme_2['p']
```

2. Then we execute the get_predictions( ) function to obtain the predicted values for both phonemes 1 and 2.

```
Z_phoneme_1 = get_predictions(mu, s, p, X)
pred_1 = Z_phoneme_1.astype(np.float32)
pred_1 = pred_1.sum(axis=1)
```

```
Z_phoneme_2 = np.zeros((N,k)) # shape Nxk
Z_phoneme_2 = get_predictions(mu_2, s_2, p_2, X)
pred_2 = Z_phoneme_2.astype(np.float32)
pred_2 = pred_2.sum(axis=1)
```

3. Create a prediction class array to store both the predictions and then store them into an array y_pred.

```
predClass = []
for p1,p2 in zip(pred_1, pred_2):
    if p1 > p2:
        predClass.append(0.0)
    else:
        predClass.append(1.0)
```

```
y_pred = np.array(predClass)
```

4. Compute y_true, an array containing the actual values.

```
y_true = np.concatenate((np.zeros((np.sum(phoneme_id==1))),np.ones((np.sum(phoneme_id==2)))),axis =0)
```

5. Calculate accuracy of the predictions by using y_pred and y_true values.

```
def get_accuracy(y_true, y_pred):
    correct_predicted = 0
    for true_label,predicted in zip(y_true,y_pred):
        if true_label == predicted:
            correct_predicted +=1
    accuracy_score = correct_predicted / len(y_true)
    return accuracy_score * 100
```

6. Calculate the mis-classification error by creating a confusion matrix.

```
confusion_matrix = metrics.confusion_matrix(y_true, y_pred)
TP = confusion_matrix[1, 1]
TN = confusion_matrix[0, 0]
FP = confusion_matrix[0, 1]
FN = confusion_matrix[1, 0]
print()
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('Mis-classification error using GMMs with {} components: {:.2f}%'.format(3, classification_error*100))
```

```
Accuracy using GMMs with 3 components: 95.07%

Mis-classification error using GMMs with 3 components: 4.93%
              precision    recall  f1-score   support

    Phoneme 1       0.94      0.96      0.95       152
    Phoneme 2       0.96      0.94      0.95       152

     accuracy                           0.95       304
    macro avg       0.95      0.95      0.95       304
 weighted avg       0.95      0.95      0.95       304


Mis-classification error using GMMs with 3 components: 4.93%
```

Q6. Repeat for K=6 and compare the results in terms of accuracy. [2 mark].

```
Accuracy using GMMs with 3 components: 95.07%

Mis-classification error using GMMs with 3 components: 4.93%
              precision    recall  f1-score   support

    Phoneme 1       0.94      0.96      0.95       152
    Phoneme 2       0.96      0.94      0.95       152

     accuracy                           0.95       304
    macro avg       0.95      0.95      0.95       304
 weighted avg       0.95      0.95      0.95       304
```

The model when run with k=3 has an accuracy of 95.07% and mis-classification error of 4.93%

```
Accuracy using GMMs with 6 components: 96.38%
              precision    recall  f1-score   support

   Phoneme 1       0.95      0.97      0.96       152
   Phoneme 2       0.97      0.95      0.96       152

    accuracy                           0.96       304
   macro avg       0.96      0.96      0.96       304
weighted avg       0.96      0.96      0.96       304


Mis-classification error using GMMs with 3 components: 3.62%
```

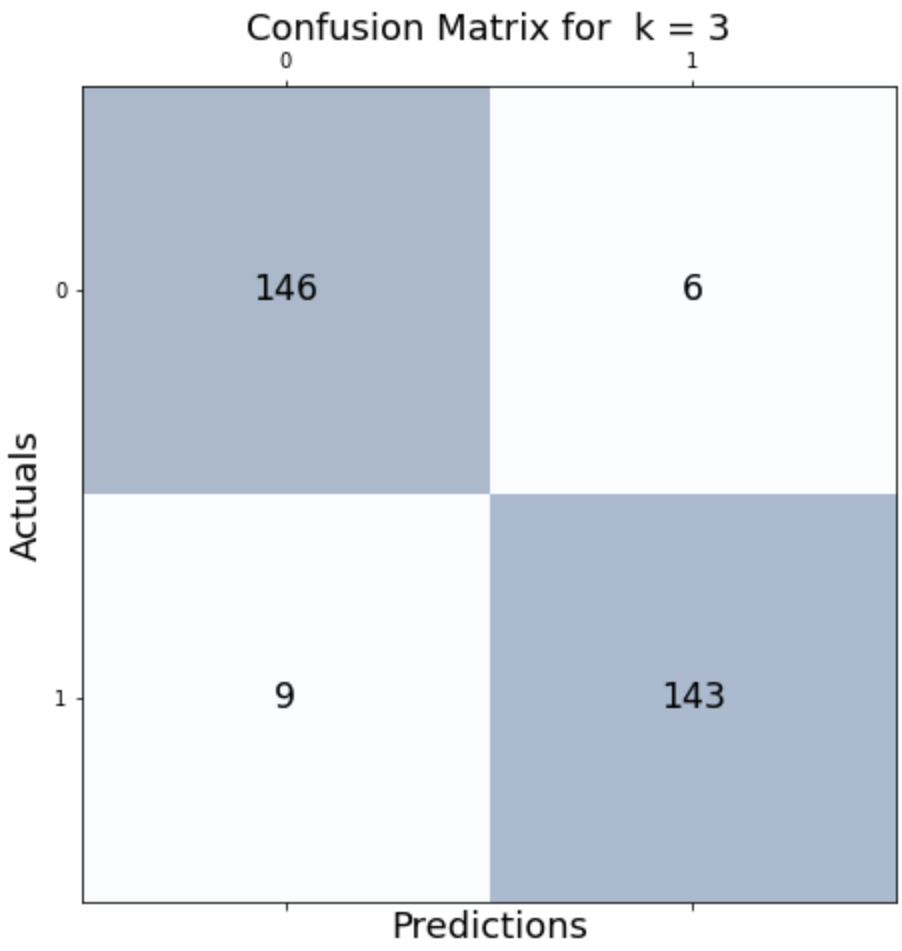
The model when run with k=6 has an accuracy of 96.38% and mis-classification error of 3.62%.

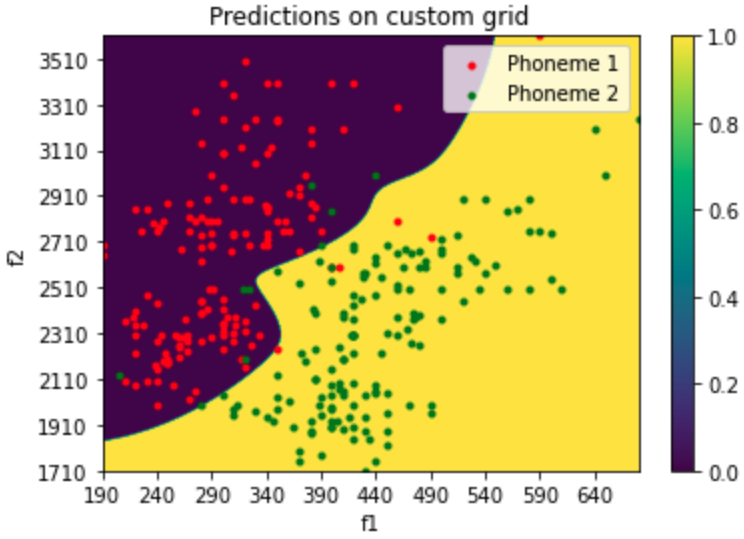
From the above accuracies we can infer that both the models are adept to classify phonemes 1 & 2. However we achieve slightly greater accuracy for K=6 than K=3.

Q7. Display a "classification matrix" assigning labels to a grid of all combinations of the F1 and F2 features for the K=3 classifiers from above. Next, repeat this step for K=6 and compare the two. [3 marks]

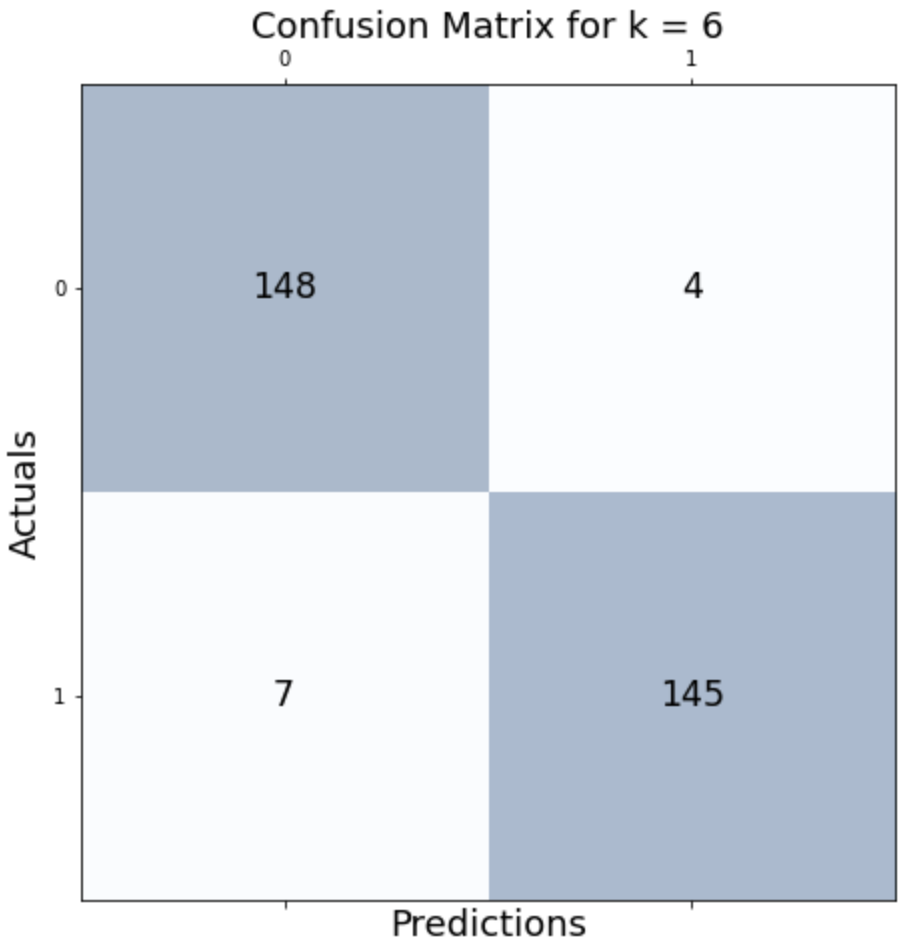
For K=3

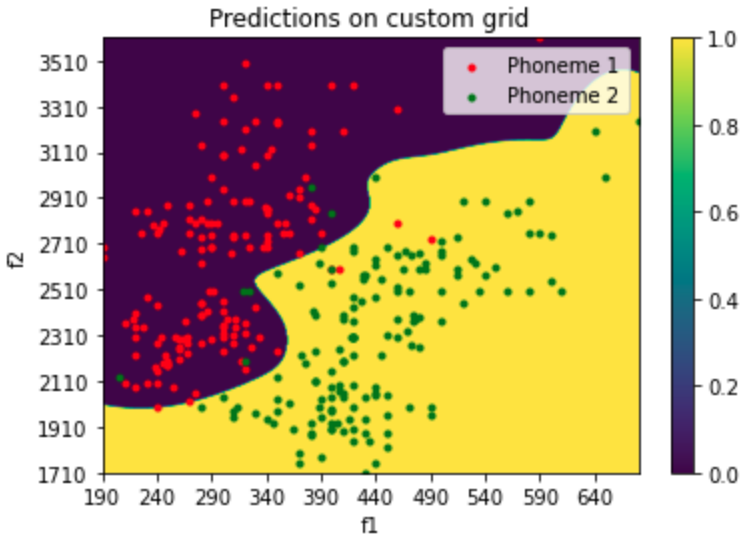
As we can see from the below images, the grid clearly displays the classification of Phonemes 1 and 2 . In the confusion matrix we can see that there are 6 and 9 wrong predictions for Phonemes 1 and 2 which can also be seen in the custom grid where we can see some of the data points of Phoneme 1 and Phoneme two have crossed the boundary line. For these data points constitute to the misclassification errors.


Confusion Matrix for k = 3
0
1
0
1
146
6
9
143
Actuals
Predictions


Predictions on custom grid
Phoneme 1
Phoneme 2
3510
3310
3110
2910
2710
2510
2310
2110
1910
1710
f2
190
240
290
340
390
440
490
540
590
640
f1
1.0
0.8
0.6
0.4
0.2
0.0

For k=6


Confusion Matrix for k = 6
0
1
0
148
4
Actuals
1
7
145
Predictions



We can see that the number of data points crossing the boundary are less and the misclassification is also lower for K = 6

Q8. Try to fit a MoG model to the new data. What is the problem that you observe? Explain why it occurs [2 marks]

```
X[:, 0] = f1
X[:, 1] = f2
X[:, 2] = f1 + f2
X = X.astype(np.float32)
X_phoneme = np.zeros((np.sum(phoneme_id==p_id), 2))
X_phoneme = X[phoneme_id == p_id, :]
X = X_phoneme.copy()
```

```
n, msg_dtype)
    107     if (
    108         allow_nan
    109         and np.isinf(X).any()
    110         or not allow_nan
    111         and not np.isfinite(X).all()
    112     ):
    113         type_err = "infinity" if allow_nan else "NaN, infinity"
--> 114         raise ValueError(
    115             msg_err.format(
    116                 type_err, msg_dtype if msg_dtype is not None else X.dtype
    117             )
    118         )
    119 # for object dtype data, we only check for NaNs (GH-13254)
    120 elif X.dtype == np.dtype("object") and not allow_nan:

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

The issue was caused because initially the non diagonal entries of covariance matrices were set to 0. These Matrices are called singular matrices and the inverse of a singular matrix is zero. When the inverse of these singular matrices is used for any calculation we get Value Error.

In our previous model the covariance matrices were being updated in the Maximization step of EM of GMM. This only happens if F1 and F2 are independent. But in this case we are updating the dataset to contain 3 values, F1, F2 and F1 + F2. The vector F1+F2 is combined from F1 and F2, this makes the vectors not linearly independent. Thus we encounter the singularity issue.

Q9. Suggest ways of overcoming the singularity problem and implement one of them. Show any training outputs in the report and discuss. [3 marks]

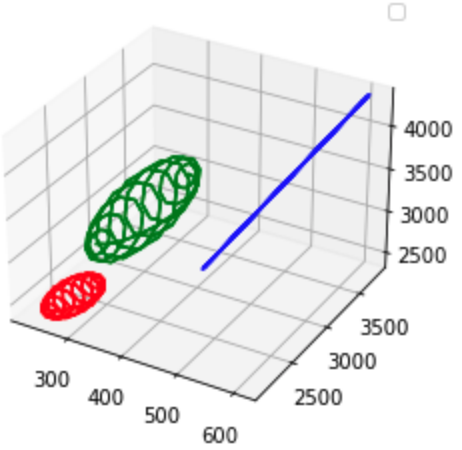
The singularity problem can be solved by the following ways:

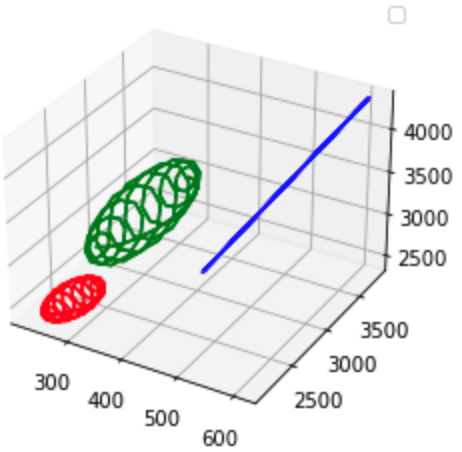
1. By introducing a small noise value to the matrix to prevent it from getting zero, and causing error during inversion.
2. Re initializing or Re setting the mean and variance whenever singularity occurs.
3. Using MAP(Maximum posterior) instead of MLE by adding a prior.

```
# Solving the singularity problem by making sure that diagonal entries of covarience matrix do not reach zero
# Multipled with 0.001 to force the diagonal entries to never reach zero.
s[i,:,:] += 0.001 * np.identity(D)
```

The first solution has been used to solve the singularity problem by making sure that entires of the covariance matrix do not reach zero, by adding a small noise(0.001) .

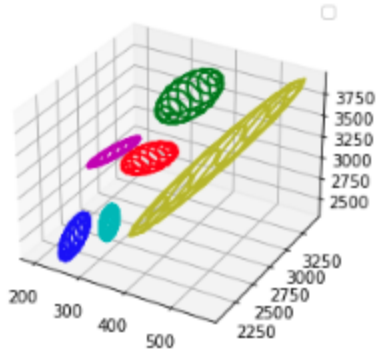
For K=3

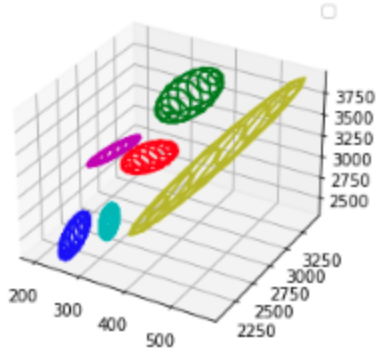





```
Implemented GMM | Mean values
[ 270.35016 2271.1824  2541.5325 ]
[ 315.7896 2877.8364 3193.6262]
[ 540.0022 3170.019  3710.0212]
Implemented GMM | Covariances
[[ 1187.87990158  1264.71779122  2452.5966928 ]
 [ 1264.71779122 13008.08932515 14272.80611637]
 [ 2452.5966928  14272.80611637 16725.40380917]]
[[ 3023.81011706  5190.79835429  8214.60747135]
 [ 5190.79835429 70425.37137705 75616.16873132]
 [ 8214.60747135 75616.16873132 83830.77720269]]
[[  2500.00099528  21999.99995846  24499.99995374]
 [ 21999.99995846 193600.00063447 215599.99959293]
 [ 24499.99995374 215599.99959293 240100.00054668]]
Implemented GMM | Weights
[0.38461053 0.60223215 0.01315732]
```

K = 6






```
Implemented GMM | Mean values
[ 321.2314 2789.7302 3110.9614]
[ 341.15964 3224.889   3566.0486 ]
[ 244.85951 2235.8257  2480.6853 ]
[ 303.9684 2350.935  2654.9036]
[ 238.11461 2806.398   3044.5125 ]
[ 462.87085 2838.94    3301.8108 ]
Implemented GMM | Covariances
[[ 1541.42760124   935.4057931   2476.83239435]
 [  935.4057931   7035.90672748  7971.31152056]
 [ 2476.83239435  7971.31152056 10448.14491492]]
[[ 2037.09193496  1240.06189824  3277.1528332 ]
 [ 1240.06189824 19770.41329315 21010.47419139]
 [ 3277.1528332  21010.47419139 24287.62802459]]
[[ 3.60969227e+02 -3.50262108e+02  1.07061190e+01]
 [-3.50262108e+02  1.34113931e+04  1.30611300e+04]
 [ 1.07061190e+01  1.30611300e+04  1.30718371e+04]]
[[ 314.10738116 -774.60867211 -460.50229095]
 [-774.60867211 7343.80342463 6569.19375251]
 [-460.50229095 6569.19375251 6108.69246155]]
[[  951.83314376  2217.97535322  3169.80749697]
 [ 2217.97535322  7245.71804423  9463.69239744]
 [ 3169.80749697  9463.69239744 12633.50089442]]
[[  5500.02398159  27867.52759965  33367.55058124]
 [ 27867.52759965 159073.29391507 186940.82051472]
 [ 33367.55058124 186940.82051472 220308.37209596]]
Implemented GMM | Weights
[0.29009627 0.17018554 0.2477264  0.18484769 0.07308011 0.03406398]
```