Cara Evangeline Chandra Mohan
190539421

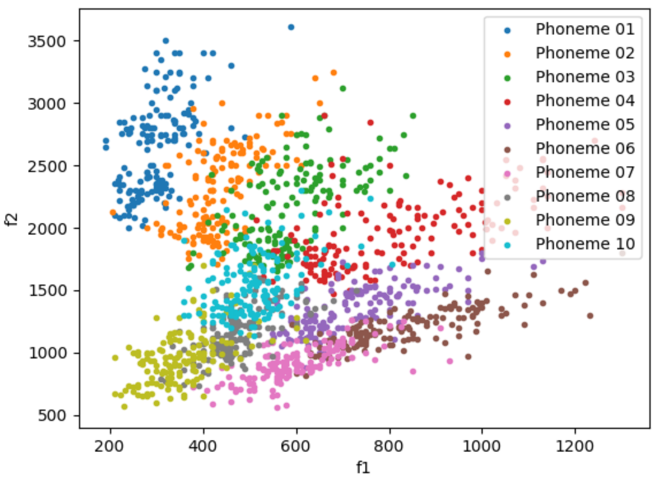
MACHINE LEARNING ASSIGNMENT - 2

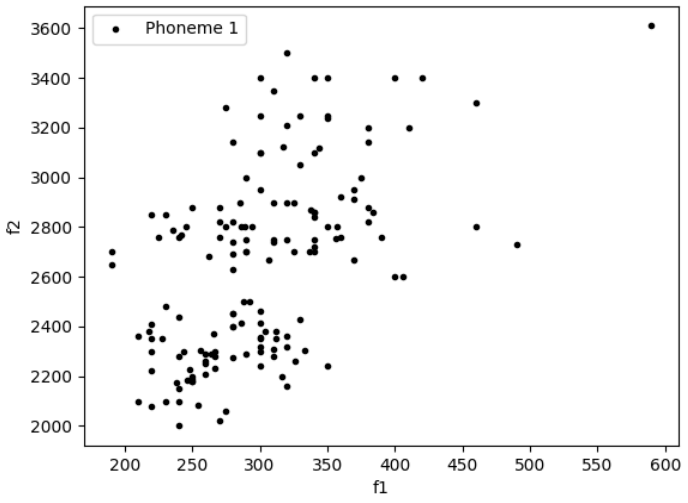
1.)Load the dataset to your workspace. We will only use the dataset for F1 and F2, arranged into a 2D matrix where the first column will be F1 and the second column will be F2. Using the code in task_1.py, produce a plot of F1 against F2. (You should be able to spot some clusters already in this scatter plot.)Include in your report the corresponding lines of your code and the plot.

<u>Lines of code:</u>

```
# Write your code here
# Store f1 in the first column of X_full, and f2 in the second column of X_full
for i in range(len(X_full)):
   X_full[i, 0] = f1[i]
   X_full[i, 1] = f2[i]
----------------------------------------------------------------------------------------------------
# Create array containing only samples that belong to phoneme 1
X_phoneme_1 = np.zeros((np.sum(phoneme_id==1), 2))
j = 0
for i in range(len(phoneme_id)):
      if phoneme_id[i] == 1:
          X_phoneme_1[j, 0] = f1[i]
          X_phoneme_1[j, 1] = f2[i]
          j = j+1
```

<u>Plot:</u>





Cara Evangeline Chandra Mohan
190539421

2.)Train the data for phonemes 1 and 2 with MoGs. You are provided with python files task_2.py, get_predictions.py and plot_gaussians.py. Include in your report the lines of code you wrote, and results that illustrate the learnt models.

Lines of Code:

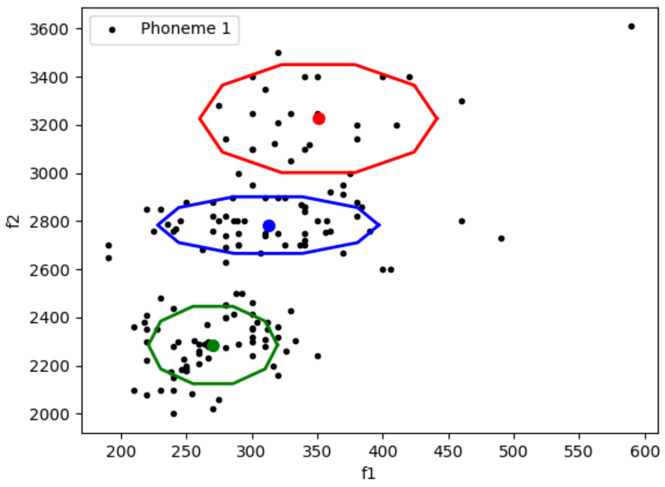
```
# Write your code here
# Store f1 in the first column of X_full, and f2 in the second column of X_full
for i in range(len(X_full)):
   X_full[i, 0] = f1[i]
   X_full[i, 1] = f2[i]
------------------------------------------------------------------------------------------
# number of GMM components
k = 3
# you can use the p_id variable, to store the ID of the chosen phoneme that will be used (e.g. phoneme 1, or phoneme 2)
p_id = 1
# Create an array named "X_phoneme", containing only samples that belong to the chosen phoneme.
X_phoneme = np.zeros((np.sum(phoneme_id==p_id), 2))
j = 0
for i in range(len(phoneme_id)):
     if phoneme_id[i] == p_id:
         X_phoneme[j, 0] = f1[i]
         X_phoneme[j, 1] = f2[i]
         j = j+1
```

For p_id = 1 and k = 3:

Output:

```
Implemented GMM | Mean values
[ 350.84461639 3226.33942727]
[ 270.39520123 2285.46534981]
[ 312.59125756 2783.89794792]

Implemented GMM | Covariances
[[ 4102.87564279     0.        ]
 [    0.         27829.52384048]]
[[ 1213.73842907     0.        ]
 [    0.         14278.42034286]]
[[3562.59741398    0.        ]
 [   0.         7657.85050199]]
```



Cara Evangeline Chandra Mohan
190539421

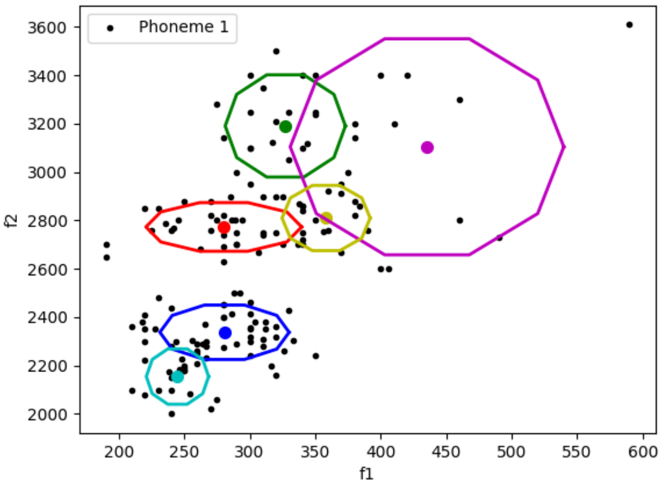
```
Implemented GMM | Weights
[0.18386034 0.43514434 0.38099532]
```

For p_id = 1 and k = 6:

Output:

```
Implemented GMM | Mean values
[ 280.22261838 2773.2121153 ]
[ 327.18825074 3190.91477118]
[ 280.77000293 2337.54855211]
[ 244.86057212 2154.31760682]
[ 435.39339013 3104.62181771]
[ 358.33912297 2810.05893689]

Implemented GMM | Covariances
[[1767.58246947    0.        ]
 [   0.         5587.98158067]]
[[ 1053.40144454     0.        ]
 [    0.         24700.08547544]]
[[1217.81324369    0.        ]
 [   0.         7000.8466357 ]]
[[ 282.97894619    0.        ]
 [   0.         7245.97362334]]
[[  5455.58478058      0.        ]
 [     0.         110351.63080648]]
[[  566.13854926     0.        ]
 [    0.         10065.94047009]]
```



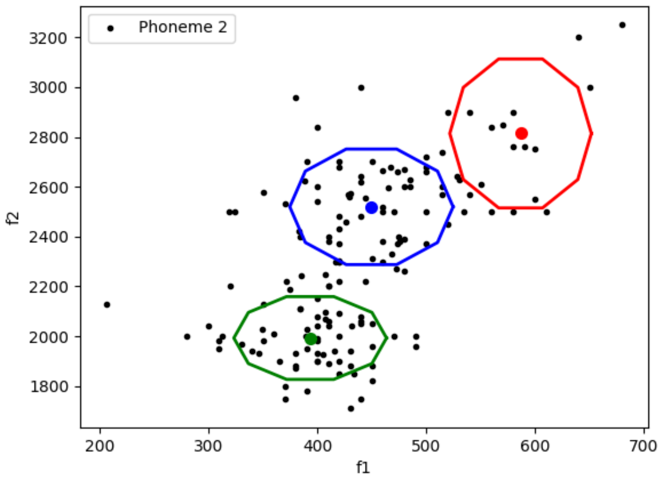
```
Implemented GMM | Weights
[0.23959994 0.15182757 0.30852128 0.12547192 0.05892473 0.11565456]
```

For p_id = 2 and k = 3:

Output:

```
Implemented GMM | Mean values
[ 586.56854146 2814.20063257]
[ 393.45313    1993.08507982]
[ 449.67862186 2519.69254665]

Implemented GMM | Covariances
[[ 2111.37544631     0.        ]
 [    0.         49424.39755058]]
```

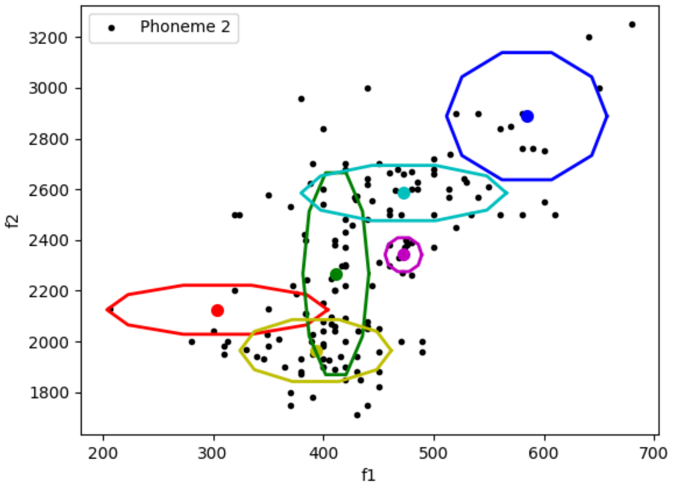


```
[[ 2454.89611799     0.        ]
 [    0.         15264.18925333]]
[[ 2809.51380436     0.        ]
 [    0.         29719.72369546]]
```

Implemented GMM | Weights
```
[0.09486833 0.4351722  0.46995947]
```

For p_id = 2 and k = 6:

Output:



Implemented GMM | Mean values
```
[ 303.96998537 2125.21538968]
[ 411.40342447 2267.23190801]
[ 584.44701231 2888.38274019]
[ 472.91517569 2585.39308096]
[ 472.63254627 2342.34102904]
[ 393.03178033 1964.38112376]
```

Implemented GMM | Covariances
```
[[5005.3600321     0.        ]
 [   0.         5155.33727723]]
[[  446.54906717     0.        ]
 [    0.         87538.52053573]]
[[ 2643.37484641     0.        ]
 [    0.         34745.52583589]]
[[4333.23437466    0.        ]
 [   0.         6566.22263362]]
[[ 138.23577474    0.        ]
 [   0.         2461.18438796]]
[[2351.74172986    0.        ]
 [   0.         8209.10617183]]
```

Implemented GMM | Weights
```
[0.0233877  0.27533015 0.07973558 0.2706006  0.06672262 0.28422336]
```

get_predictions.py

$$p(x) = \sum_{k=1}^{K} \frac{p(c_k)}{(2\pi)^{\frac{D}{2}} det(|\Sigma_k|)^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right)$$ -- Formula 1

```
mu_i = mu[i,:]                                   # i ----> for every cluster/gaussian distribution
mu_i = np.expand_dims(mu_i, axis=1)
mu_i_repeated = np.repeat(mu_i, N, axis=1)
X_minus_mu = X - mu_i_repeated.transpose()
inverse_s = scipy.linalg.pinv(s[i])
inverse_s = np.squeeze(inverse_s)
s_i_det = scipy.linalg.det(s[i])
x_s_x = np.matmul(X_minus_mu, inverse_s)*X_minus_mu
Z[:,i] = p[i]*(1/np.power( ((2*np.pi)**D) * np.abs(s_i_det), 0.5 )) * np.exp(-0.5*np.sum(x_s_x,
axis=1))
```

According to formula 1, in get_predictions.py we calculate the probability of a point belonging to a particular gaussian distribution belonging to a phoneme.

- mu_i calculates the mean of the dimensions of a particular cluster
- s[i] is the covariance matrix of a particular gaussian distribution
- s_i_det is the determinant of the covariance matrix
- X is the dataset points
- p[i] is the weight given to a particular gaussian distribution

Formula 1 calculates the probablity of a point belonging to a particular phoneme whereas get_predictions.py calculates the probablity of a point belonging to a gaussian distribution of a given phoneme.

3.)Use the 2 MoGs (K=3) learnt in Task 2 to build a classifier to discriminate between phonemes 1 and 2. Classify using the Maximum Likelihood (ML) criterion (feel free to hack parts from the code in task_3.py, and utilize the get_predictions.py file so that you calculate the likelihood of a data vector for each of the two MoG models). Calculate the mis-classification error. Remember that a classification under the ML compares p(x; θ_1), where θ_1 are the parameters of the MoG learnt for the first phoneme, with p(x; θ_2), where θ_2 are the parameters of the MoG learnt for the second phoneme. Repeat this for K=6 and compare the results. Include in your report the lines of the code that your wrote, explanations of what the code does and comment on the differences on the classification performance.

<u>Lines of Code:</u>

```
# Write your code here
# Store f1 in the first column of X_full, and f2 in the second column of X_full
for i in range(len(X_full)):
    X_full[i, 0] = f1[i]
    X_full[i, 1] = f2[i]
```

#Pretrained models ---------------> I have used a function call to model.py which is similar to task_2.py to get the trained model parameters

```
k = 3
```

Cara Evangeline Chandra Mohan
190539421

```
mu1, p1, s1 = model(k, 1)
mu2, p2, s2 = model(k, 2)
-----------------------------------------------------------------------------------------------------
# Create an array named "X_phonemes_1_2", containing only samples that belong to phoneme 1 and samples that belong to phoneme 2.
n1 = np.sum(phoneme_id==1)
n2 = np.sum(phoneme_id==2)
X_phonemes_1_2 = np.zeros((n1+n2, 2))
j = 0
for i in range(len(phoneme_id)):
     if phoneme_id[i] == 1:
         X_phonemes_1_2[j, 0] = f1[i]
         X_phonemes_1_2[j, 1] = f2[i]
         j = j+1
for i in range(len(phoneme_id)):
     if phoneme_id[i] == 2:
         X_phonemes_1_2[j, 0] = f1[i]
         X_phonemes_1_2[j, 1] = f2[i]
         j = j+1
-------------------------------------------------------------------------------------------------------
# Write your code here
X = X_phonemes_1_2.copy()
# get number of samples
N = X.shape[0]
# get dimensionality of our dataset
D = X.shape[1]
Z1 = np.zeros((N,k))
Z2 = np.zeros((N,k))
# Get predictions on samples from both phonemes 1 and 2, from a GMM with k components, pretrained on phoneme 1
Z1 = get_predictions(mu1, s1, p1, X)
# Get predictions on samples from both phonemes 1 and 2, from a GMM with k components, pretrained on phoneme 2
Z2 = get_predictions(mu2, s2, p2, X)
Z1_new = np.zeros((Z1.shape[0],1))
Z2_new = np.zeros((Z1.shape[0],1))
for i in range(Z1.shape[0]):
   Z1_new[i] = Z1[i, 0] + Z1[i, 1] + Z1[i, 2]
   Z2_new[i] = Z2[i, 0] + Z2[i, 1] + Z2[i, 2]
# Compare these predictions for each sample of the dataset, and calculate the accuracy, and store it in a scalar variable named "accuracy"
correct_classify = 0
for i in range(len(X_phonemes_1_2)):
   if i < len(X_phonemes_1_2)/2:
        if Z1_new[i] > Z2_new[i]:
            correct_classify = correct_classify+1
    else:
        if Z2_new[i] > Z1_new[i]:
            correct_classify = correct_classify+1
accuracy = (correct_classify/len(X_phonemes_1_2))*100
error = (1-(correct_classify/len(X_phonemes_1_2)))*100
print('Accuracy using GMMs with {} components: {:.2f}%'.format(k, accuracy))
print('Misclassification error using GMMs with {} components: {:.2f}%'.format(k, error))
```

Code Explaination:

- model.py is a file similar to task_2.py which is used in task_3.py to train the data of phoneme_1 and phoneme_2 individually. Once the data is trained we get the corresponding mean, covariance and weight (mu1, s1, p1 and mu2, s2, p2) of the gaussian distributions of the respective phonemes.
- Now, we build data with both phoneme_1 and phoneme_2 datapoints(phoneme_1_2)
- We predict the cluster that phoneme_1_2 belongs to by using get_predictions() and using the trained parameters mu1, s1, p1 and mu2, s2, p2 and store it in Z1 and Z2 respectively.
- Now, we sum up the probablities (Z1_new and Z2_new) of a point belonging to a gaussian distribution of a phoneme to get the probablity of the point belonging to a particular phoneme.
- Compare the probablities Z1_new and Z2_new to see whether a point belongs to phoneme_1 or phoneme_2
- Accuarcy = Correctly_classified/Total_number_of_tuples

Result:

- Accuracy using GMMs with 6 components: 94.08%
  Misclassification error using GMMs with 6 components: 5.92%
- Accuracy using GMMs with 3 components: 95.07%
  Misclassification error using GMMs with 3 components: 4.93%

The accuarcy of GMM with 3 component is better than GMM with 6 components because k=6 tries to overfit the data leading to more misclassification error than k=3.

4.)Create a grid of points that spans the two datasets (i.e., the dataset that contains phoneme 1, and the dataset that contains phoneme 2). Classify each point in the grid using one of your classifiers (i.e., the MoG that was trained on phoneme 1 and the MoG that was trained on phoneme 2). Display the classification matrix.Include the lines of code in your report, comment them, and display the classification matrix.

Lines of Code:

```
#Pretrained models ---------------> I have used a function call to model.py which is similar to
```

Cara Evangeline Chandra Mohan
190539421

task_2.py to get the trained model parameters

```
k = 3
mu1, p1, s1 = model(k, 1)
mu2, p2, s2 = model(k, 2)
```

# Create an array named "X_phonemes_1_2", containing only samples that belong to phoneme 1 and samples that belong to phoneme 2.

```
n1 = np.sum(phoneme_id==1)
n2 = np.sum(phoneme_id==2)
X_phonemes_1_2 = np.zeros((n1+n2, 2))
j = 0
for i in range(len(phoneme_id)):
      if phoneme_id[i] == 1:
          X_phonemes_1_2[j, 0] = f1[i]
          X_phonemes_1_2[j, 1] = f2[i]
          j = j+1
for i in range(len(phoneme_id)):
      if phoneme_id[i] == 2:
          X_phonemes_1_2[j, 0] = f1[i]
          X_phonemes_1_2[j, 1] = f2[i]
          j = j+1
```

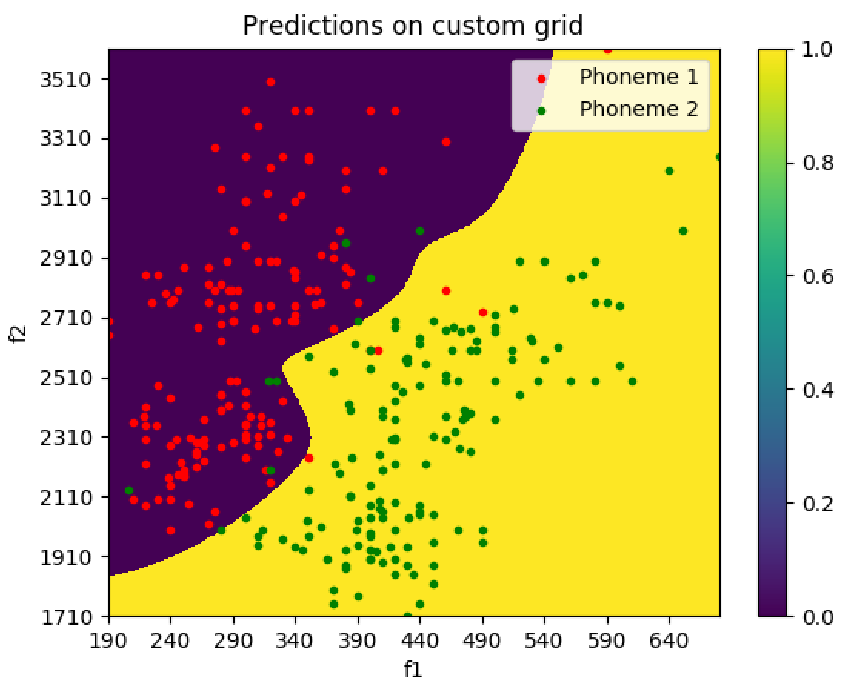
```
# Write your code here--------Classification Matrix
M = np.zeros((N_f2, N_f1))
# Create a custom grid of shape N_f1 x N_f2
x = np.linspace(min_f1, max_f1, N_f1)
y = np.linspace(min_f2, max_f2, N_f2)
x1, y1 = np.meshgrid(x, y, sparse = True)
Z1 = np.zeros((1,k))
Z2 = np.zeros((1,k))
a = np.zeros((1, 2))
for i in range(N_f2):
    for j in range(N_f1):
        a[0, 0] = x1[0, j]
        a[0, 1] = y1[i, 0]
        Z1 = get_predictions(mu1, s1, p1, a)
        Z2 = get_predictions(mu2, s2, p2, a)
        Z1_new = np.zeros((Z1.shape[0],1))
        Z2_new = np.zeros((Z1.shape[0],1))
        Z1_new = Z1[0, 0] + Z1[0, 1] + Z1[0, 2]
        Z2_new = Z2[0, 0] + Z2[0, 1] + Z2[0, 2]
        if Z1_new > Z2_new:
          M[i, j] = 0
        else:
```

```
        M[i, j] = 1
print(M)
```

Explaination of code:

- Similar to task_3, we have used model.py for obtaining trained parameters.
- X_phoneme_1_2 dataset is formed
- For building up the classification matrix, we predict the class of X_phoneme_1_2 using get_predictions() using trained parameters and we compare Z1_new and Z2_new and obtain the phoneme for every point in X_phoneme_1_2
- If a point in X_phoneme_1_2 belongs to phoneme_1 then M=0 and if a point belongs to phoneme_2 then M=1

Result:(For K = 3)



Classification Matrix:

```
[[1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 ...
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]]
```

Cara Evangeline Chandra Mohan
190539421

5.)In the code of task_5.py a MoG with full covariance matrices is fit to the data. Now, create a new dataset that will contain 3 columns, as follows: X = [F1, F2, F1+F2]
Fit a MoG model to the new data. What is the problem that you observe? Explain why. Suggest ways of overcoming the singularity problem and implement them.
Include the lines of code in your report, and graphs/plots so as to support your observations.

Lines of Code: (Note: Red -Changes Made, Black - Already Present, Green - Comments)

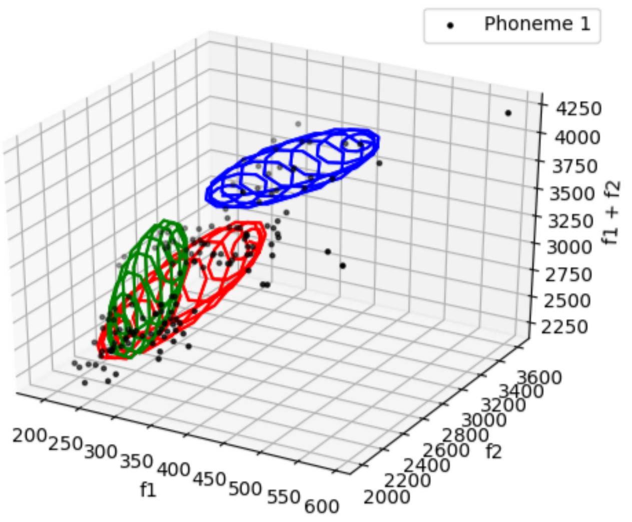
```
# Create an array named "X_phoneme", containing only samples that belong to the chosen
phoneme.
X_phoneme = np.zeros((np.sum(phoneme_id==1), 3))
j = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == 1:
        X_phoneme[j, 0] = f1[i]
        X_phoneme[j, 1] = f2[i]
        X_phoneme[j, 2] = f1[i]+f2[i]
        j = j+1
-------------------------------------------------------------------------------------------------------------
#Fitting the model using EM method
for t in range(n_iter):
    print('Iteration {:03}/{:03}'.format(t+1, n_iter))
    # Do the E-step
    for i in range(k):
        s_i_det = scipy.linalg.det(s[i])
        if np.abs(s_i_det) == 0:
          s[i] = [[1, 1000, 1000],[1000, 1, 1000],[1000, 1000, 1]]
    Z = get_predictions(mu, s, p, X)
    Z = normalize(Z, axis=1, norm='l1')
    # Do the M-step:
    for i in range(k):
        mu[i,:] = np.matmul(X.transpose(), Z[:,i])/np.sum(Z[:,i])
        # We will fit Gaussians with full covariance matrices:
        mu_i = mu[i,:]
        mu_i = np.expand_dims(mu_i, axis=1)
        mu_i_repeated = np.repeat(mu_i, N, axis=1)
        term_1 = X.transpose() - mu_i_repeated
        term_2 = np.repeat(np.expand_dims(Z[:,i], axis=1), D, axis=1) * term_1.transpose()
        s[i,:,:] = np.matmul(term_1, term_2)/np.sum(Z[:,i])
        p[i] = np.mean(Z[:,i])
    ax1.clear()
        # plot the samples of the dataset, belonging to the chosen phoneme (f1, f2, f1+f2 |
phoneme 1 or 2)
```

```
    plot_data_3D(X=X, title_string=title_string, ax=ax1)
    # Plot gaussians after each iteration
    plot_gaussians(ax1, 2*s, mu)
```

<u>Code Explaination and Problem Faced:</u>

- Problem faced here is singularity problem where at some point of time the determinant of covariance matrix becomes zero leading to infinte value of probablity.
- To avoid this, when the determinant of covariance matrix becomes zero and gaussian model collapses, we can declare covariance to be some large value and continue with the optimization.
- The changes in code is highlighted in red where we check for 0 detereminant and intialize the covariance matrix to large value except the diagonal elements which is intialized to 1

<u>Result:</u>



```
Implemented GMM | Mean values
[ 295.56941637 2539.32054322 2834.88995959]
[ 248.25110848 2534.69087278 2782.94198126]
[ 347.01936445 3246.66997089 3593.68933534]
Implemented GMM | Covariances
[[  2812.01142007   7844.50482216  10656.51624223]
 [  7844.50482216  83333.27307563  91177.77789779]
 [ 10656.51624223  91177.77789779 101834.29414002]]
[[ 1435.8376539  -5098.8794608  -3663.0418069 ]
 [-5098.8794608  75287.45964579 70188.58018499]
 [-3663.0418069  70188.58018499 66525.5383781 ]]
[[ 4921.3229024   6507.86827847 11429.19118087]
 [ 6507.86827847 29131.16221737 35639.03049584]
 [11429.19118087 35639.03049584 47068.22167671]]
Implemented GMM | Weights
[0.79792769 0.04761518 0.15445713]
```