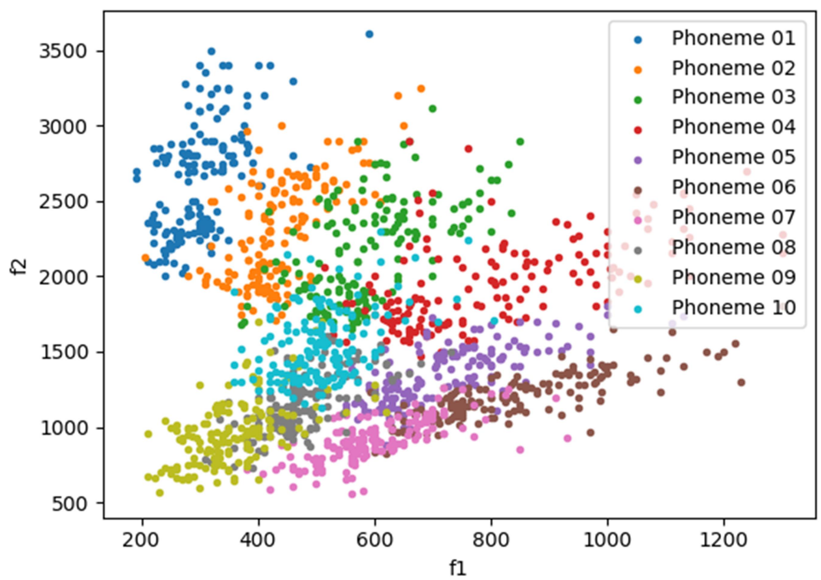
Task 1:

```
##########################################
# Write your code here
# Store f1 in the first column of X_full, and f2 in the second column of X_full
X_full[:, 0] = f1
X_full[:, 1] = f2
##########################################/
```

The scatter plot of the whole is given below:



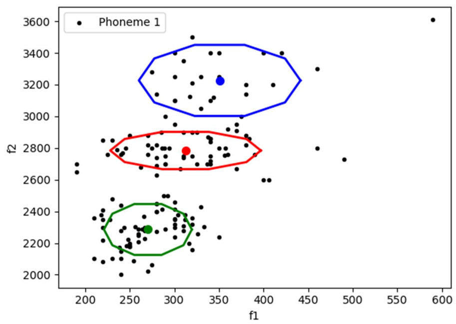
Task 2:

```
##########################################
# Write your code here

# Create an array named "X_phoneme", containing only samples that belong to the
chosen phoneme.
# The shape of X_phoneme will be two-dimensional. Each row will represent a sample
of the dataset, and each column will represent a feature (e.g. f1 or f2)
# Fill X_phoneme with the samples of X_full that belong to the chosen phoneme
# To fill X_phoneme, you can leverage the phoneme_id array, that contains the ID
of each sample of X_full

# X_phoneme = ...
X_phoneme = np.empty((0, 2))
for i, phoneme in enumerate(phoneme_id):
    if phoneme == p_id:
        X_phoneme = np.append(X_phoneme, [X_full[i]], axis=0)
##########################################/
```

When p_id =1 k=3 (first run):



Implemented GMM | Mean values

[312.59350304 2783.90779909]

[270.39518811 2285.46523098]

[350.84638986 3226.39325972]

Implemented GMM | Covariances

[[3562.56238066   0.   ][0.   7659.188771]]

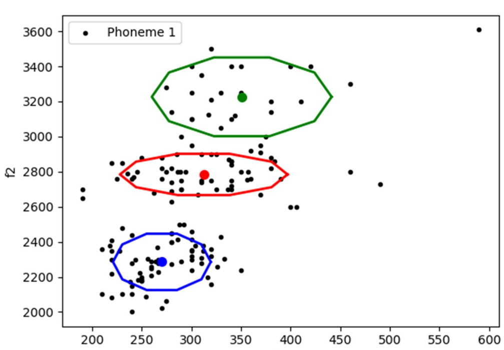
[[1213.7388661   0.   ][0.   14278.40519955]]

[[4103.14624508   0.   ][0.   27815.66503816]]

Implemented GMM | Weights

[0.3810266 0.43514408 0.18382931]

When p_id =1 k=3 (second run):



Implemented GMM | Mean values

[312.59101059 2783.89686544]

[350.84442157 3226.33350825]

[270.39520267 2285.46536284]

Implemented GMM | Covariances

[[3562.60125052   0.   ] [0.   7657.7035147]]

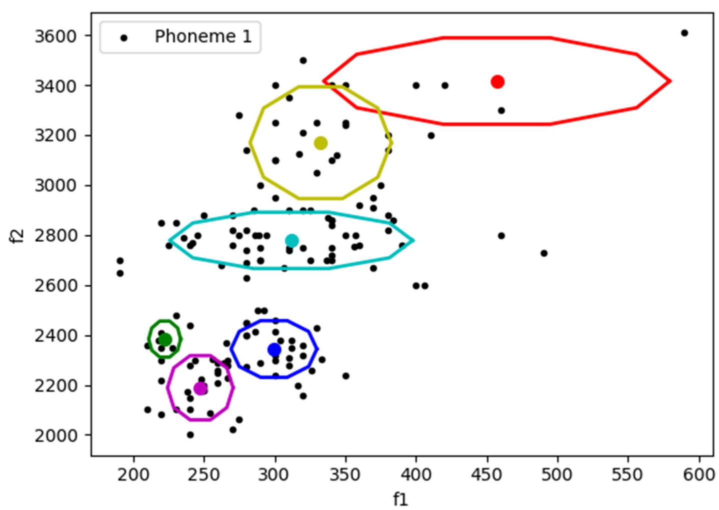
[[4102.84591673   0.   ] [0.   27831.0479474]]

[[1213.73838111   0.   ] [0.   14278.42200152]]

Implemented GMM | Weights

[0.38099188 0.18386375 0.43514437]

When p_id =1 k=6 (first run):



Implemented GMM | Mean values

[457.14320288 3415.82177028]

[222.36367131 2382.92101417]

[299.6330761   2343.36332949]

[311.77726537 2777.95278992]

[247.53252273 2188.32578441]

[332.66932759 3169.00645794]

Implemented GMM | Covariances

[[7484.99212445   0.   ] [0.   16533.9016334]]

[[64.00858485   0.   ] [0.   2870.89190502]]

[[460.05800562   0.   ] [0.   7116.26720108]]

[[3692.2548788   0.   ] [0.   7011.56713729]]

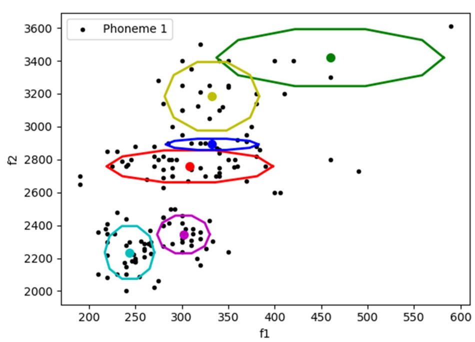
[[269.03977159   0.   ] [0.   9147.51644731]]

[[1247.51982296   0.   ] [0.   27683.1835309]]

Implemented GMM | Weights

[0.02629111 0.04534958 0.21249325 0.36616425 0.17626342 0.17343838]

When p_id =1 k=6 (second run):



Implemented GMM | Mean values

[308.60976519 2758.0317891]

[459.77104476 3419.02023515]

[332.61275026 2891.72596839]

[243.85000432 2234.58313928]

[301.87761932 2344.00505671]

[332.49274772 3184.14542689]

Implemented GMM | Covariances

[[4011.11629444   0.   ] [0.   5207.62247962]]

[[7487.77566098   0.   ] [0.   16567.12558234]]

[[1248.55474281   0.   ] [0.   639.46746022]]

[[356.12559769   0.   ] [0.   14192.45134232]]

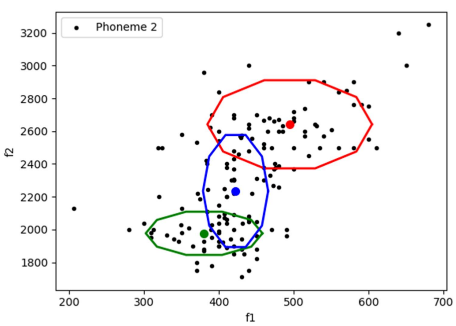
[[401.26925074   0.   ] [0.   7253.81322169]]

[[1295.26815791   0.   ] [0.   23994.64923993]]

Implemented GMM | Weights

[0.31407893 0.02536212 0.05901031 0.23552469 0.19879136 0.16723258]

P_id = 2, K=3 (first run):



Implemented GMM | Mean values

[494.57771034 2641.8306782]

[380.35335024 1977.16145931]

[422.35296791 2234.89239865]

Implemented GMM | Covariances

[[6028.37376199   0.   ] [0.   39911.38562194]]

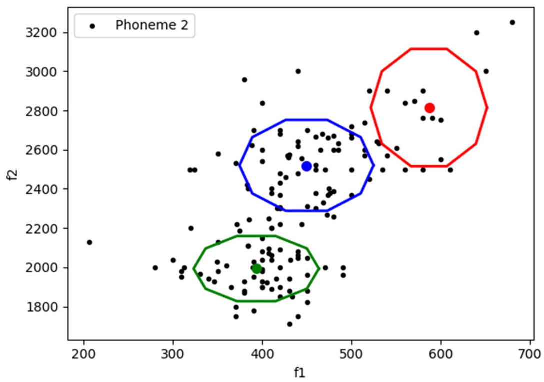
[[3027.48266711   0.   ] [0.   9593.12601103]]

[[944.55931253   0.   ] [0.   64502.90397744]]

Implemented GMM | Weights

[0.37647676 0.27015809 0.35336515]

<u>P_id = 2, K=3 (second run):</u>



Implemented GMM | Mean values

[586.56168038 2814.17424277]

[393.45280373 1993.08179073]

[449.67524905 2519.68449523]

Implemented GMM | Covariances

[[2111.57496187   0.   ] [0.   49424.41868097]]

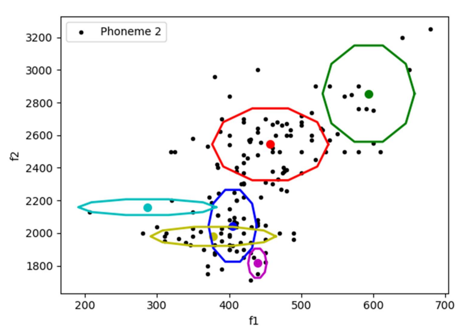
[[2454.91328052   0.   ] [0.   15263.55476211]]

[[2809.27550477   0.   ] [0.   29720.48875153]]

Implemented GMM | Weights

[0.09488306 0.43516579 0.46995115]

P_id = 2 K=6 (first run):



Implemented GMM | Mean values

[457.24289991 2543.87935991]

[593.8295612   2854.61520169]

[404.98324083 2044.87675142]

[286.29771138 2158.88168211]

[439.34033977 1815.47974558]

[378.04695926 1980.06288649]

Implemented GMM | Covariances

[[3257.79740262   0.   ] [0.   26770.38459488]]

[[2038.74208693   0.   ] [0.   48064.10801198]]

[[558.83532712   0.   ] [0.   26752.4727024]]

[[4590.91337817   0.   ] [0.   1279.80765044]]

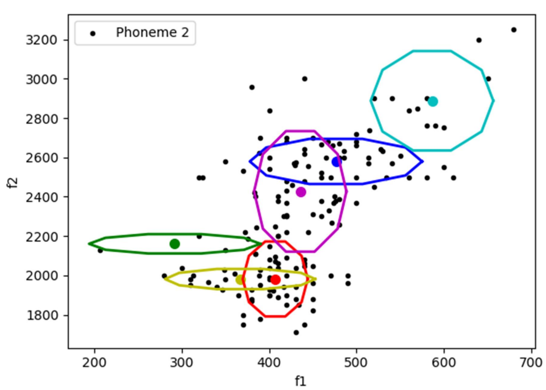
[[81.75642993   0.   ] [0.   4396.71886222]]

[[3788.19773094   0.   ] [0.   1868.02552831]]

Implemented GMM | Weights

[0.45180658 0.07614803 0.28813074 0.01669957 0.02787813 0.13933695]

P_id = 2 K=6 (second run):



Implemented GMM | Mean values

[406.98613146 1982.12860533]

[291.97025457 2160.84266224]

[476.41831724 2579.33469862]

[586.38895296 2887.7442048]

[435.60464722 2427.03828092]

[366.79343113 1981.71303184]

Implemented GMM | Covariances

[[692.43079563 0.] [0. 20024.37275315]]

[[4804.54574526 0.] [0. 1335.54765549]]

[[4853.24273494 0.] [0. 7269.11402696]]

[[2456.96614448 0.] [0. 35269.04255639]]

[[1397.36912917 0.] [0. 51867.87722396]]

[[3691.62083334 0.] [0. 1455.0549978]]

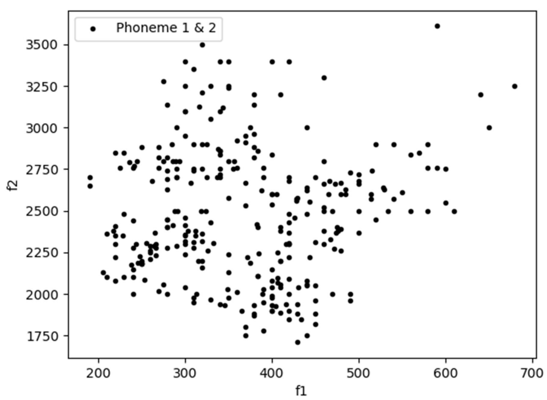
Implemented GMM | Weights

[0.27888925 0.0178867 0.21865879 0.07830013 0.2960006 0.11026452]

Observation: Different mixtures of Gaussians are obtained when the algorithm is run multiple times.

Task 3:

Given below is the scatter plot for



Code:

```
##########################################
# Write your code here
# Store f1 in the first column of X_full, and f2 in the second column of X_full
X_full[:, 0] = f1
X_full[:, 1] = f2
#########################################/
```

The code snippet given above stores f1 and and f2 in the 0th and 1st column of X_full

```
##########################################
# Write your code here

# Create an array named "X_phonemes_1_2", containing only samples that belong to
```

```
phoneme 1 and samples that belong to phoneme 2.
# The shape of X_phonemes_1_2 will be two-dimensional. Each row will represent a
sample of the dataset, and each column will represent a feature (e.g. f1 or f2)
# Fill X_phonemes_1_2 with the samples of X_full that belong to the chosen
phonemes
# To fill X_phonemes_1_2, you can leverage the phoneme_id array, that contains the
ID of each sample of X_full

# X_phonemes_1_2 = ...

# y is the target phoneme value
y = np.array([])
X_phonemes_1_2 = np.empty((0, 2))
for i, phoneme in enumerate(phoneme_id):
    if phoneme == 1 or phoneme == 2:
        X_phonemes_1_2 = np.append(X_phonemes_1_2, [X_full[i]], axis=0)
        y = np.append(y, phoneme)
########################################/
```

The code snippet given above is used to store the only the phoneme with id = 1 or 2. The array 'y' has the actual phoneme id for the respective row in the X_phonemes_1_2. This will be used later for calculating accuracy.

```
#########################################
# Write your code here
# Get predictions on samples from both phonemes 1 and 2, from a GMM with k
components, pretrained on phoneme 1
# Get predictions on samples from both phonemes 1 and 2, from a GMM with k
components, pretrained on phoneme 2
# Compare these predictions for each sample of the dataset, and calculate the
accuracy, and store it in a scalar variable named "accuracy"

# as dataset X, we will use only the samples of the chosen phoneme
X = X_phonemes_1_2.copy()
# get number of samples
N = X.shape[0]
# change k here
k = 6

GMM_file_01 = 'data/GMM_params_phoneme_01_k_{:02}.npy'.format(k)
# Loading data from .npy file
GMM_parameters_1 = np.load(GMM_file_01, allow_pickle=True)
GMM_parameters_1 = GMM_parameters_1.item()

mu_1 = GMM_parameters_1['mu']
s_1 = GMM_parameters_1['s']
p_1 = GMM_parameters_1['p']

# Initialize array Z that will get the predictions of each Gaussian on each sample
Z_1 = np.zeros((N,k))

Z_1 = get_predictions(mu_1, s_1, p_1, X)
sum1 = Z_1.sum(axis=1)

GMM_file_02 = 'data/GMM_params_phoneme_02_k_{:02}.npy'.format(k)
# Loading data from .npy file
GMM_parameters_2 = np.load(GMM_file_02, allow_pickle=True)
GMM_parameters_2 = GMM_parameters_2.item()
```

```
mu_2 = GMM_parameters_2['mu']
s_2 = GMM_parameters_2['s']
p_2 = GMM_parameters_2['p']

# Initialize array Z that will get the predictions of each Gaussian on each sample
Z_2 = np.zeros((N,k))

Z_2 = get_predictions(mu_2, s_2, p_2, X)
sum2 = Z_2.sum(axis=1)

prediction = np.array([])


for i in range(y.size):
    if sum1[i] > sum2[i]:
        prediction = np.append(prediction, 1)
    else:
        prediction = np.append(prediction, 2)

accuracy = (sum(np.equal(prediction, y)) / y.size) * 100

#########################################/
```

The code snippet given above does the following steps:

1) Copies relevant sample (p_id =1 or 2) to X
2) Gets the size of the dataset (N)
3) Initializes K (change k here for the whole program)
4) Loads the GMM 1 parameters to mu_1, s_1, and p_1
5) Gets the prediction and stores it in z_1
6) Sums the rows of the prediction to get the total probability for the data point and stores in sum 1
7) Repeats the points 4 to 6 for GMM 2 and gets sum 2
8) Compares sum1 and sum2. The p_id = 1 if sum1 > sum2 else p_id = 2
9) Calculates the accuracy by comparing prediction array with y array. (the resuts obtained are given below)

Result:

Accuracy using GMMs with 6 components: 95.72%

Accuracy using GMMs with 3 components: 96.38%

Observation:

The accuracy of GMM with 3 components is higher than GMM with 6 components as the GMM with 6 components is prone to overfitting the data points. Hence the accuracy of the data will decrease.

Task 4:

```
#########################################
# Write your code here
# Store f1 in the first column of X_full, and f2 in the second column of X_full
X_full[:, 0] = f1
```

```
X_full[:, 1] = f2
#########################################/
#########################################
# Write your code here

# Create an array named "X_phonemes_1_2", containing only samples that belong to
phoneme 1 and samples that belong to phoneme 2.
# The shape of X_phonemes_1_2 will be two-dimensional. Each row will represent a
sample of the dataset, and each column will represent a feature (e.g. f1 or f2)
# Fill X_phonemes_1_2 with the samples of X_full that belong to the chosen
phonemes
# To fill X_phonemes_1_2, you can leverage the phoneme_id array, that contains the
ID of each sample of X_full

# X_phonemes_1_2 = ...
X_phonemes_1_2 = np.empty((0, 2))
for i, phoneme in enumerate(phoneme_id):
    if phoneme == 1 or phoneme == 2:
        X_phonemes_1_2 = np.append(X_phonemes_1_2, [X_full[i]], axis=0)
#########################################/
```

```
#########################################
# Write your code here

# Create a custom grid of shape N_f1 x N_f2
# The grid will span all the values of (f1, f2) pairs, between [min_f1, max_f1] on
f1 axis, and between [min_f2, max_f2] on f2 axis
# Then, classify each point [i.e., each (f1, f2) pair] of that grid, to either
phoneme 1, or phoneme 2, using the two trained GMMs
# Do predictions, using GMM trained on phoneme 1, on custom grid
# Do predictions, using GMM trained on phoneme 2, on custom grid
# Compare these predictions, to classify each point of the grid
# Store these prediction in a 2D numpy array named "M", of shape N_f2 x N_f1 (the
first dimension is f2 so that we keep f2 in the vertical axis of the plot)
# M should contain "0.0" in the points that belong to phoneme 1 and "1.0" in the
points that belong to phoneme 2
#########################################/

grid = np.zeros((N_f1, N_f2, 2))

for i in range(N_f1):
    for j in range(N_f2):
        grid[i][j] = [min_f1+i, min_f2+j]

GMM_file_01 = 'data/GMM_params_phoneme_01_k_{:02}.npy'.format(k)
# Loading data from .npy file
GMM_parameters_1 = np.load(GMM_file_01, allow_pickle=True)
GMM_parameters_1 = GMM_parameters_1.item()

mu_1 = GMM_parameters_1['mu']
s_1 = GMM_parameters_1['s']
p_1 = GMM_parameters_1['p']

GMM_file_02 = 'data/GMM_params_phoneme_02_k_{:02}.npy'.format(k)
# Loading data from .npy file
GMM_parameters_2 = np.load(GMM_file_02, allow_pickle=True)
GMM_parameters_2 = GMM_parameters_2.item()
```

```
mu_2 = GMM_parameters_2['mu']
s_2 = GMM_parameters_2['s']
p_2 = GMM_parameters_2['p']

N = grid.shape[0]
# change k here
k = 3
# Initialize array Z that will get the predictions of each Gaussian on each sample
Z_1 = np.zeros((N,k))
# Initialize array Z that will get the predictions of each Gaussian on each sample
Z_2 = np.zeros((N, k))
prediction = np.zeros(N)
temp = np.zeros((N_f1, N_f2))
for i in range(N_f2):
    # as dataset X, we will use only the samples of the chosen phoneme
    X_temp = grid[:,i]
    # get number of samples

    Z_1 = get_predictions(mu_1, s_1, p_1, X_temp)
    sum1 = Z_1.sum(axis=1)

    Z_2 = get_predictions(mu_2, s_2, p_2, X_temp)
    sum2 = Z_2.sum(axis=1)

    for j in range(N):
        if sum1[j] > sum2[j]:
            prediction[j] = 0.0
        else:
            prediction[j] = 1.0
    temp[:,i] = prediction

M = temp.T
print(M)
#################################################
```

The above code snippet does the following:

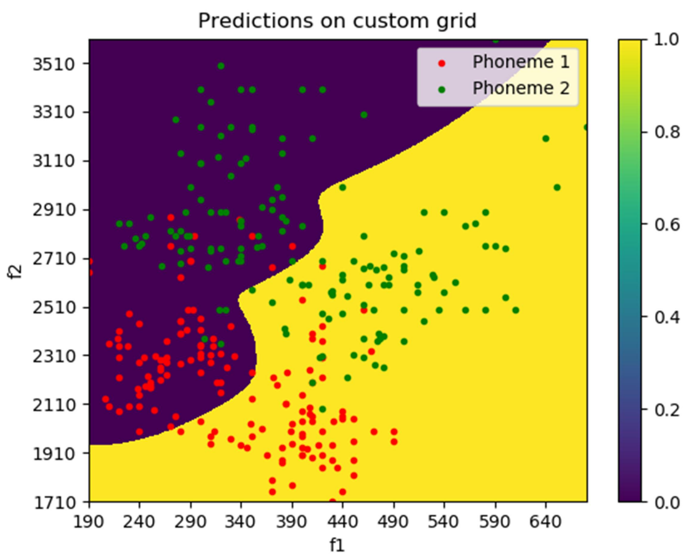
1) Initialise the grid with size N_f1*N_f2*2
2) Assign a 2d array (ordered pair of f1 and f2) at each position in the grid ranging from min f1 to max f1 and min f2 to max f2
3) Loads the GMM 1 parameters to mu_1, s_1, and p_1
4) Loads the GMM 2 parameters to mu_2, s_2, and p_2
5) Initialise z_1, z_2, prediction and temp array
6) Run a loop for every column in the grid
7) Pass the columns through the different MoGs and get the sum of their predictions in sum1 and sum2
8) Assign 0.0 to prediction at index i if sum1[i] > sum2[i] i.e. if the data point belongs to phoneme 1 else assign 1.0
9) Store the above value at temp's column
10) Transpose the temp matrix to get the classification matrix M

Result:

f1 range: 190-680 | 490 points

f2 range: 1710-3610 | 1900 points

```
[[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.]]
```



Task 5:

```
############################################
# Write your code here
# Store f1 in the first column of X_full, f2 in the second column of X_full and f1+f2 in the third column of X_full
X_full[:, 0] = f1
X_full[:, 1] = f2
X_full[:, 2] = f1 + f2
###########################################/
```

```
############################################
# Write your code here

# Create an array named "X_phoneme", containing only samples that belong to the chosen phoneme.
# The shape of X_phoneme will be two-dimensional. Each row will represent a sample of the dataset, and each column will represent a feature (e.g. f1 or f2 or f1+f2)
# Fill X_phoneme with the samples of X_full that belong to the chosen phoneme
# To fill X_phoneme, you can leverage the phoneme_id array, that contains the ID of each sample of X_full

# X_phoneme = ...
```

```
X_phoneme = np.empty((0, 3))
for i, phoneme in enumerate(phoneme_id):
    if phoneme == p_id:
        X_phoneme = np.append(X_phoneme, [X_full[i]], axis=0)
########################################/
```

Singularity:

Singularity in a likelihood function is when a component collapse on a data point. It is a form of over fitting. The variance becomes zero which in this case leads to singular covariance matrix. When the variance is zero, the likelihood of the component becomes infinity and hence the model over fits.

There are two ways to overcome singularity:

1) Resetting the mean and variance when singularity occurs
2) Using MAP instead of MLE