Assignment 2: Clustering and MoG

Tian Yang 200054403

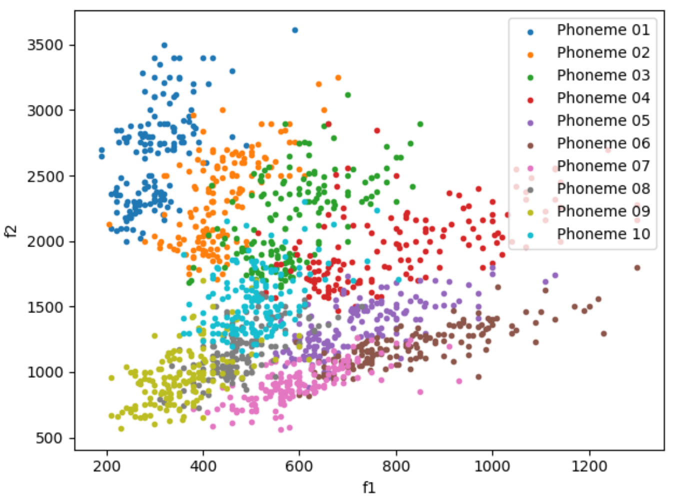
1. Introduction

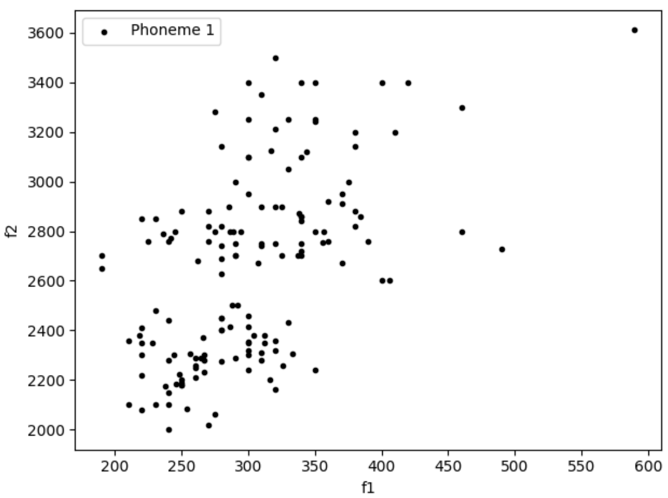
2. MoG Modelling using the EM Algorithm

Task 1.
Load the dataset to your workspace

```
########################################
# 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
```

```
# Create array containing only samples that belong to phoneme 1
# X_phoneme_1 = np.zeros((np.sum(phoneme_id==1), 2))
# X_phoneme = ...
########################################
X_phoneme_index = np.where(phoneme_id == p_id)
X_phoneme_1 = np.take(X_full,X_phoneme_index,axis=0).reshape(X_phoneme_index[0].size,2)
```

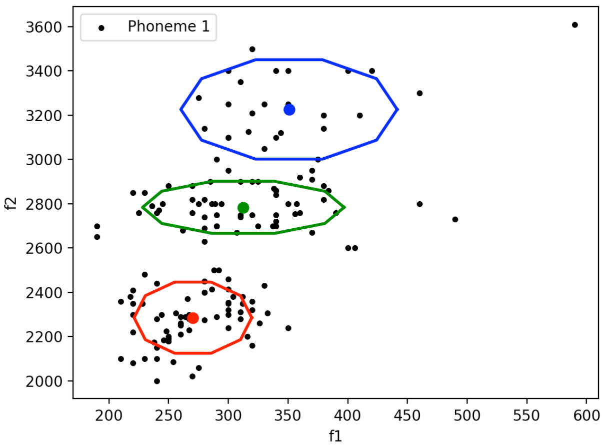




Task 2.

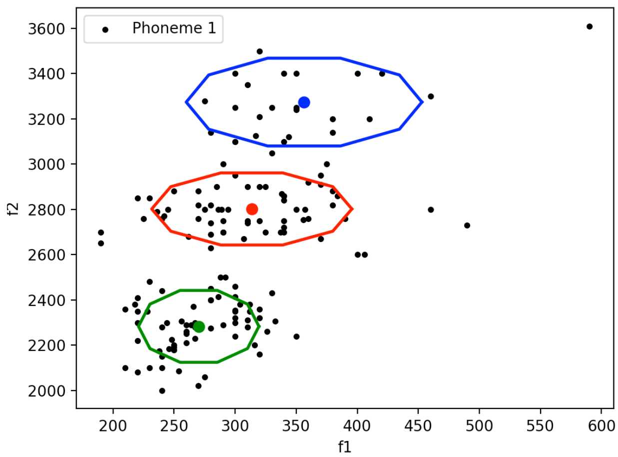
```
###########################################
# Write your code here
# Store f1 in the first column of X_full, and f2 in the second column of X_full
###########################################/
X_full[:,0] = f1
X_full[:,1] = f2
```

```
# 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 col
# 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_index = np.where(phoneme_id == p_id)
X_phoneme = np.take(X_full,X_phoneme_index,axis=0).reshape(X_phoneme_index[0].size,2)
```



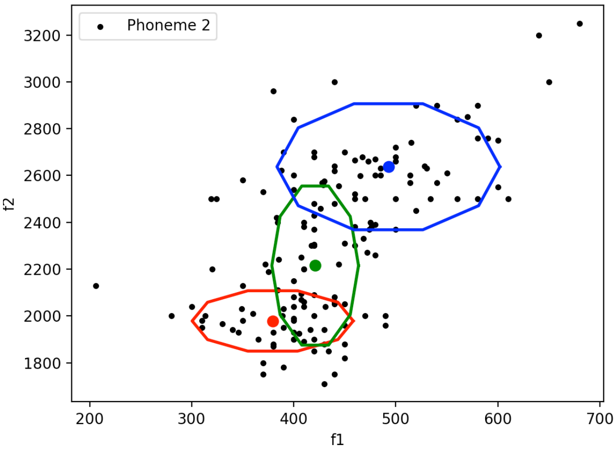

```
Implemented GMM | Mean values
[ 270.39523 2285.4656 ]
[ 312.58765 2783.882  ]
[ 350.84177 3226.2527 ]

Implemented GMM | Covariances
[[ 1213.73772361      0.        ]
 [     0.         14278.44415102]]
[[3562.65331818     0.        ]
 [    0.         7655.69925162]]
[[ 4102.44043808      0.        ]
 [     0.         27851.86932182]]

Implemented GMM | Weights
[0.43514475 0.38094492 0.18391033]
```

phoneme 1, with k=3. First run



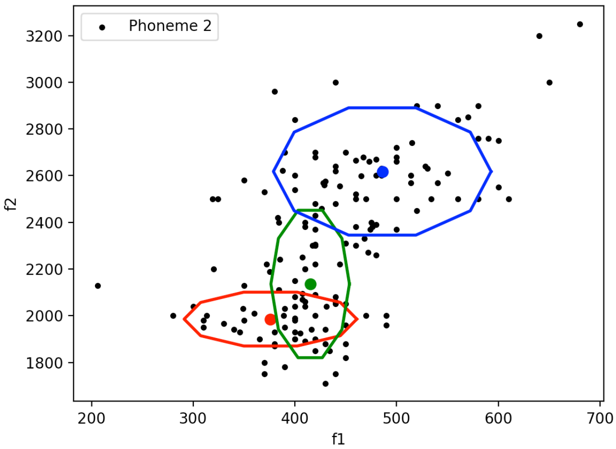

```
Implemented GMM | Mean values
[ 313.6613 2802.092 ]
[ 270.20688 2283.2168 ]
[ 356.45474 3274.358  ]

Implemented GMM | Covariances
[[ 3358.5245628       0.        ]
 [     0.         14020.64230588]]
[[ 1216.23366574      0.        ]
 [     0.         13954.79296709]]
[[ 4653.30763224      0.        ]
 [     0.         20805.47099489]]

Implemented GMM | Weights
[0.42275769 0.43015936 0.14708296]
```

phoneme 1, with k=3. Second run



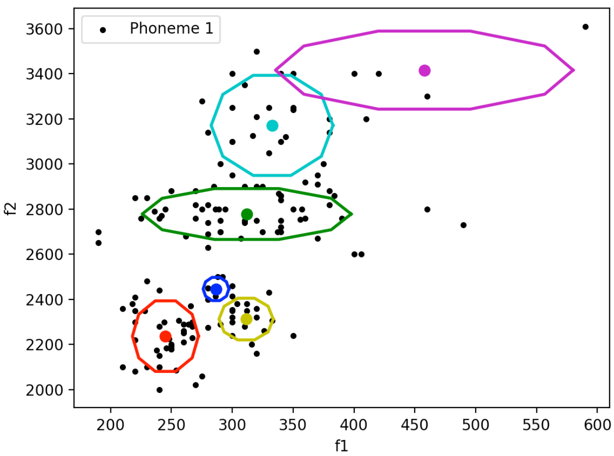

```
Implemented GMM | Mean values
[ 379.347   1978.5043]
[ 420.98712 2215.0947 ]
[ 492.76318 2637.243  ]

Implemented GMM | Covariances
[[3116.26532431     0.        ]
 [   0.          9197.58775315]]
[[  902.05333697      0.        ]
 [    0.          63855.32373409]]
[[ 5967.74629124      0.        ]
 [    0.          40074.34202426]]

Implemented GMM | Weights
[0.25869579 0.35144752 0.38985669]
```

phoneme 2, with k=3. First run



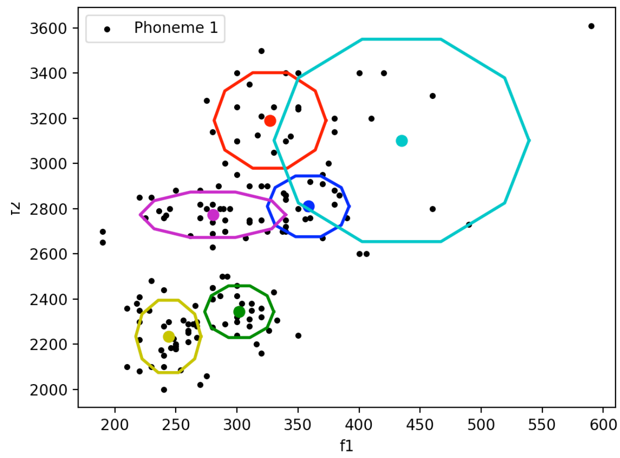

```
Implemented GMM | Mean values
[ 375.94388 1985.4246 ]
[ 414.9836 2135.8362]
[ 485.845 2617.785]

Implemented GMM | Covariances
[[3604.30752133     0.        ]
 [   0.          7347.69343598]]
[[  745.31541108      0.        ]
 [    0.          55142.44299427]]
[[ 5727.15320473      0.        ]
 [    0.          41095.89244664]]

Implemented GMM | Weights
[0.21499214 0.33896815 0.44603972]
```

phoneme 2, with k=3. Second run



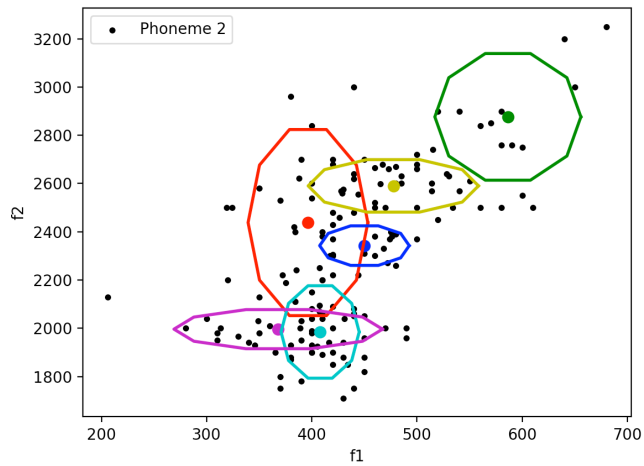

```
Implemented GMM | Mean values
[ 245.11943 2237.088  ]
[ 311.85422 2778.3416 ]
[ 286.65182 2446.0862 ]
[ 332.7125 3171.2952]
[ 457.7642 3416.3743]
[ 311.38663 2312.6072 ]

Implemented GMM | Covariances
[[  367.83554726      0.        ]
 [    0.          13514.10391681]]
[[3685.5965512     0.        ]
 [   0.         7020.9017499]]
[[  56.12213657    0.        ]
 [   0.         1460.31467806]]
[[ 1251.41263485      0.        ]
 [    0.          27228.96662456]]
[[ 7475.38939881      0.        ]
 [    0.          16553.54681457]]
[[ 248.32471022    0.        ]
 [   0.         4718.65417407]]

Implemented GMM | Weights
[0.24979599 0.36734425 0.05004984 0.17241233 0.02609128 0.1343063 ]
```

phoneme 1, with k=6. First run



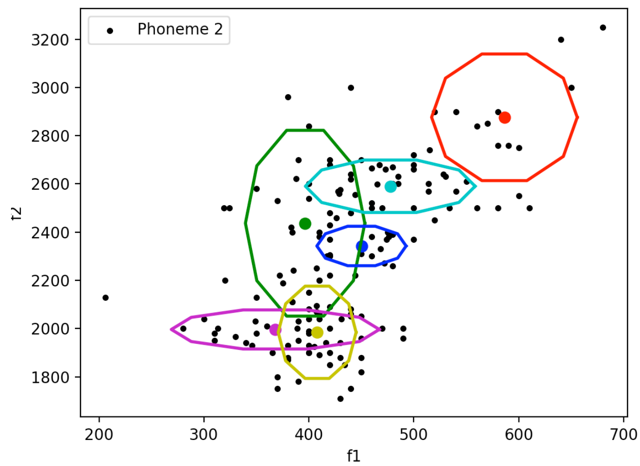

```
Implemented GMM | Mean values
[ 327.1001 3190.7595]
[ 301.86517 2344.0437 ]
[ 358.36786 2810.5403 ]
[ 434.77734 3102.2827 ]
[ 280.33453 2773.238  ]
[ 243.85497 2234.6045 ]

Implemented GMM | Covariances
[[ 1048.55520064      0.        ]
 [     0.         24715.59004468]]
[[ 399.91666179    0.        ]
 [    0.        7251.95683769]]
[[  559.94308088      0.        ]
 [     0.         10001.40645958]]
[[  5449.05418755      0.        ]
 [     0.         110963.36738974]]
[[1772.7220904     0.        ]
 [   0.         5582.1419498 ]]
[[  356.26941445      0.        ]
 [     0.         14193.3589094 ]]

Implemented GMM | Weights
[0.15161511 0.19844494 0.11485233 0.05955146 0.23995247 0.2355837 ]
```

phoneme 1, with k=6. Second run




```
Implemented GMM | Mean values
[ 396.3096 2438.028 ]
[ 586.36005 2876.3152 ]
[ 450.03137 2342.4766 ]
[ 407.86667 1984.5808 ]
[ 368.04446 1996.1493 ]
[ 477.4643 2590.2024]

Implemented GMM | Covariances
[[ 1621.30608048      0.        ]
 [     0.         82225.52523401]]
[[ 2409.91192801      0.        ]
 [     0.         38111.96544857]]
[[ 906.70218219     0.        ]
 [    0.         3714.44838083]]
[[  689.50857788      0.        ]
 [     0.         20297.75742914]]
[[4913.99653129     0.        ]
 [   0.         3632.60336596]]
[[3278.10137124     0.        ]
 [   0.         6543.68482816]]

Implemented GMM | Weights
[0.13674383 0.08167359 0.11240297 0.27206812 0.13749416 0.25961734]
```

phoneme 2, with k=6. First run

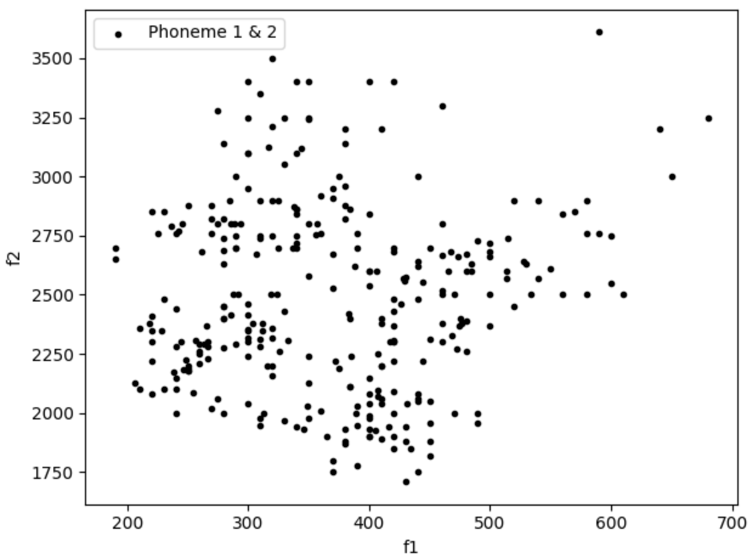



```
Implemented GMM | Mean values
[ 586.3655 2876.2258]
[ 396.40115 2436.8215 ]
[ 450.1299 2342.3948]
[ 477.5531 2590.226 ]
[ 368.03586 1996.1536 ]
[ 407.8833 1984.3352]

Implemented GMM | Covariances
[[ 2408.91698142      0.        ]
 [     0.         38133.83428523]]
[[ 1618.51367544      0.        ]
 [     0.         82110.62809793]]
[[ 904.3900702     0.        ]
 [    0.         3700.04149644]]
[[3271.49319893     0.        ]
 [   0.         6542.69850282]]
[[4913.81823007     0.        ]
 [   0.         3630.73664982]]
[[  688.98065874      0.        ]
 [     0.         20259.5316736 ]]

Implemented GMM | Weights
[0.08169255 0.13800839 0.112015   0.25927359 0.13746843 0.27154204]
```

phoneme 2, with k=6. Second run

Conclusion: We can get different MoG value when the algorithm run multiple times.

Task 3.




```
#########################################
# 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
```

```
# Create an array named "X_phonemes_1_2", containing only samples that belong to phoneme 1 and samples that belon
# The shape of X_phonemes_1_2 will be two-dimensional. Each row will represent a sample of the dataset, and each
# 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.zeros((np.sum(phoneme_id==1) + np.sum(phoneme_id==2), 2))
index = 0
pairs_label = np.zeros((np.sum(phoneme_id==1) + np.sum(phoneme_id==2), 1))
groundtruth_index = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == 1 or phoneme_id[i] == 2:
        X_phonemes_1_2[index] = X_full[i]
        index+=1
        if phoneme_id[i] == 1:
            pairs_label[groundtruth_index] = 1
        elif phoneme_id[i] == 2:
            pairs_label[groundtruth_index] = 2
        groundtruth_index+=1
```

Load data for phoneme1 & phoneme 2

```
# Load data for phoneme1
data_phoneme1 = np.load(phoneme1_path, allow_pickle=True)
data_phoneme1 = np.ndarray.tolist(data_phoneme1)
means1 = data_phoneme1['mu']
weights1 = data_phoneme1['p']
covariance1 = data_phoneme1['s']
predictions1 = get_predictions(means1, covariance1, weights1, X)

# load data for phoneme2
data_phoneme2 = np.load(phoneme2_path, allow_pickle=True)
data_phoneme2 = np.ndarray.tolist(data_phoneme2)
means2 = data_phoneme2['mu']
weights2 = data_phoneme2['p']
covariance2 = data_phoneme2['s']
predictions2 = get_predictions(means2, covariance2, weights2, X)
```

Get the sum value for each row

```
# get sum value from each row for predictions 1 & 2
sumValue_predictions1 = np.zeros((np.sum(phoneme_id == 1) + np.sum(phoneme_id == 2), 1))
sumValue_predictions2 = np.zeros((np.sum(phoneme_id == 1) + np.sum(phoneme_id == 2), 1))
for i in range(len(X)):
    sumValue_predictions1[i] = np.sum(predictions1[i])
    sumValue_predictions2[i] = np.sum(predictions2[i])
print(sumValue_predictions1)

# compare sum values from each array
predicted_label = np.zeros((np.sum(phoneme_id == 1) + np.sum(phoneme_id == 2), 1))
predicted_index = 0
for i in range(len(X)):
    if sumValue_predictions1[i] > sumValue_predictions2[i]:
        predicted_label[predicted_index] = 1
    elif sumValue_predictions1[i] < sumValue_predictions2[i]:
        predicted_label[predicted_index] = 2
    predicted_index += 1
```

```
# calculate accuracy
correct_samples = 0
total_samples = 0
for i in range(len(predicted_label)):
    if predicted_label[i] == phoneme_label[i]:
        correct_samples += 1
    total_samples += 1

accuracy = correct_samples/total_samples
print('Accuracy using GMMs with {} components: {:.2f}%'.format(k, accuracy))
```

Accuracy using GMMs with 3 components: 0.95%

Accuracy using GMMs with 6 components: 0.96%

Conclusion : Accuracy using GMMs with 6 components is higher than accuracy using GMMs with 3 components

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
```

```
# Create an array named "X_phonemes_1_2", containing only samples that belong to phoneme 1
# The shape of X_phonemes_1_2 will be two-dimensional. Each row will represent a sample of
# 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 e
# X_phonemes_1_2 = ...
########################################/
phonemes1_index = np.where(phoneme_id == 1)
phonemes1_index_size = phonemes1_index[0].size
phonemes_1 = np.take(X_full,phonemes1_index,axis=0).reshape(phonemes1_index_size,2)
print(len(phonemes_1))

phonemes2_index = np.where(phoneme_id == 2)
phonemes2_index_size = phonemes2_index[0].size
phonemes_2 = np.take(X_full,phonemes2_index,axis=0).reshape(phonemes2_index_size,2)
print(len(phonemes_2))
# print(phonemes_2)

X_phonemes_1_2 = np.vstack((phonemes_1,phonemes_2))
print(len(X_phonemes_1_2))
# print(X_phonemes_1_2)

# as dataset X, we will use only the samples of phoneme 1 and 2
X = X_phonemes_1_2.copy()
```

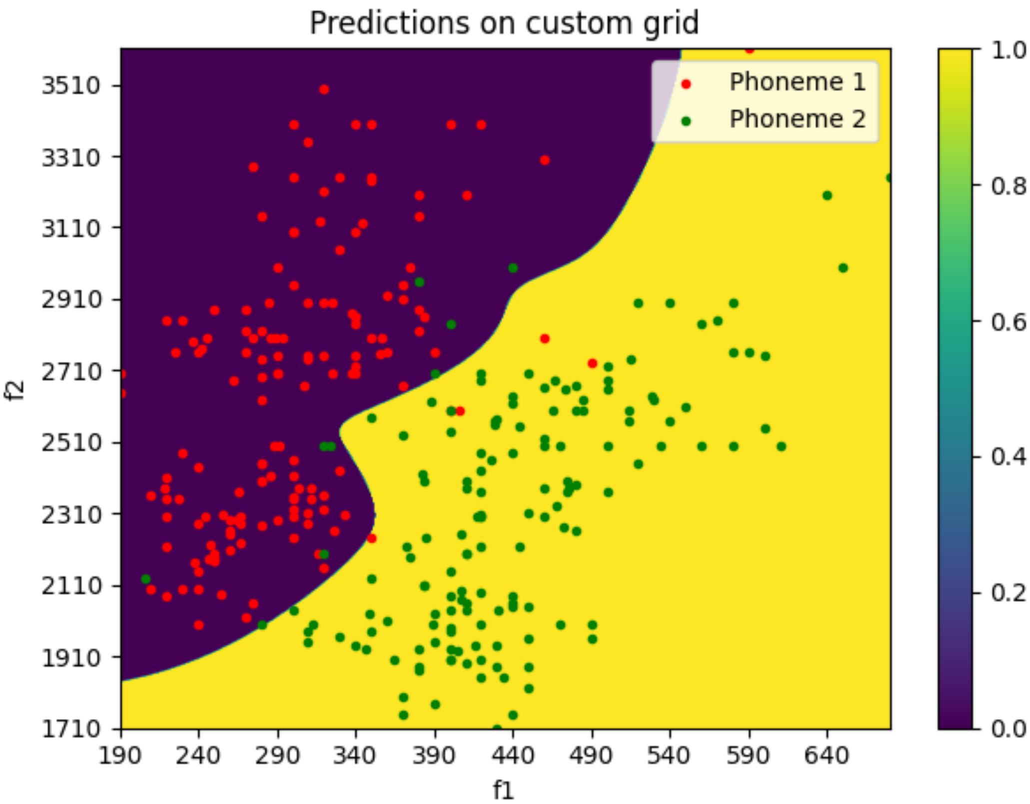
```
# Store these prediction in a 2D numpy array named "M", of shape N_f2 x N_f1 (the first dime
# M should contain "0.0" in the points that belong to phoneme 1 and "1.0" in the points that
########################################/

#get f1 values between [minf1, maxf1]
f1_values = np.linspace(min_f1,max_f1,num=N_f1)
#get f2 values between [minf2, maxf2]
f2_values = np.linspace(min_f2,max_f2,num=N_f2)
#get every possible combination of values
newf1,newf2=np.meshgrid(f2_values,f1_values)

# print(newf1.shape)
# print(newf2.shape)

# generate the 2 arrays for every possible combination between f1 and f2
custom_grid = np.array([newf2,newf1]).T
```

```
# File path of phoneme 1 & phoneme 2
if k == 3:
    phoneme1_path = 'data/GMM_params_phoneme_01_k_03.npy'
    phoneme2_path = 'data/GMM_params_phoneme_02_k_03.npy'
else:
    phoneme1_path = 'data/GMM_params_phoneme_01_k_06.npy'
    phoneme2_path = 'data/GMM_params_phoneme_02_k_06.npy'

# Load data for phoneme1
data_phoneme1 = np.load(phoneme1_path, allow_pickle=True)
data_phoneme1 = np.ndarray.tolist(data_phoneme1)
# load data for phoneme2
data_phoneme2 = np.load(phoneme2_path, allow_pickle=True)
data_phoneme2 = np.ndarray.tolist(data_phoneme2)

# predict each row
predict_row = lambda X, p: np.sum(get_predictions(p["mu"], p["s"], p["p"], X), axis=1)
predict_data_phoneme_1 = lambda X: predict_row(X, dict(data_phoneme1))
predict_data_phoneme_2 = lambda X: predict_row(X, dict(data_phoneme2))

P_1 = np.array([predict_data_phoneme_1(F_n) for F_n in custom_grid])
P_2 = np.array([predict_data_phoneme_2(F_n) for F_n in custom_grid])
```

f1 range: 190-680 | 490 points

f2 range: 1710-3610 | 1900 points

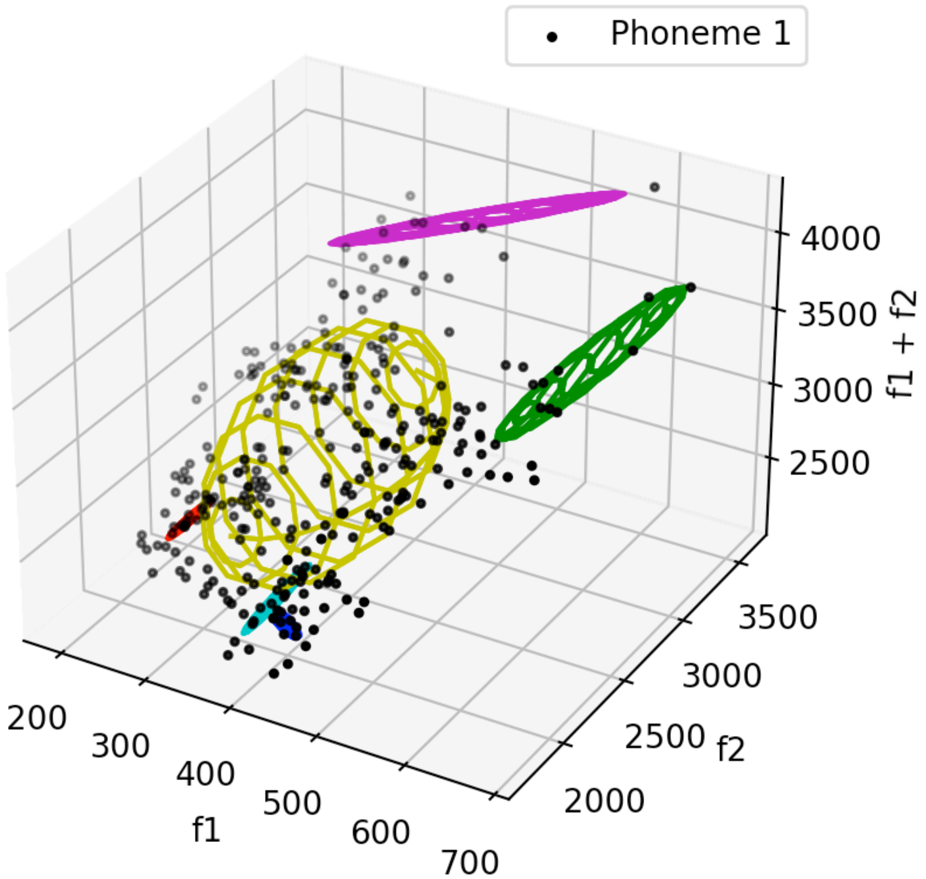
```
f1 range: 190-680 | 490 points
f2 range: 1710-3610 | 1900 points
[[1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 ...
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]]
```



Task 5.

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

```
# X_phoneme = ...
#########################################/
X_phoneme = np.zeros((np.sum(phoneme_id==1) + np.sum(phoneme_id==2), 3))
j = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == 1 or phoneme_id[i] == 2:
        X_phoneme[j,:] = X_full[i,:]
        j += 1
```



When I was test, I met the following error,

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

And the way to solve it is

```
#########################################
# Write your code here
# Suggest ways of overcoming the singularity
#########################################/
s = s + np.multiply(np.eye(D), 0.1)
```