Machine Learning: Assignment 2

Pedro Pereira Sarmento

Queen Mary, University of London

December 03, 2019

This reports presents an application of unsupervised Gaussian Mixture Models to the Peterson and Barney's dataset of vowel formant frequencies for a phoneme, using the Expectation-Maximization algorithm. First, from the estimated parameters for the mixture model, a maximum likelihood estimation classifier is built, deciding if an observation is more likely under the parameters for phoneme 1 or phoneme 2, obtained by the EM algorithm. Second, the problem of singularity in the covariance matrix of a GMM is addressed, adding noise to yield a non-zero determinant.

1 Task 1

In Figure 1 a plot of F1 against F2 of our dataset is presented. Figure 5 displays a plot of F1 against F2 containing solely the phoneme 1.

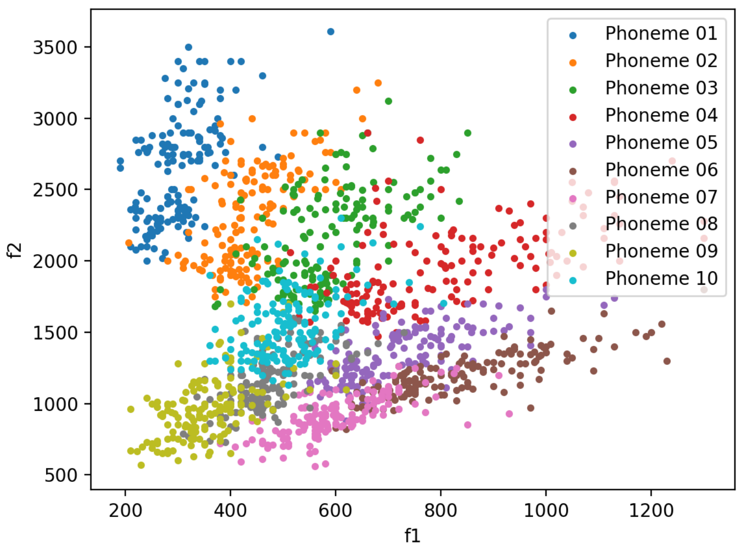


Fig. 1: Plot of F1 against F2, for all phonemes.

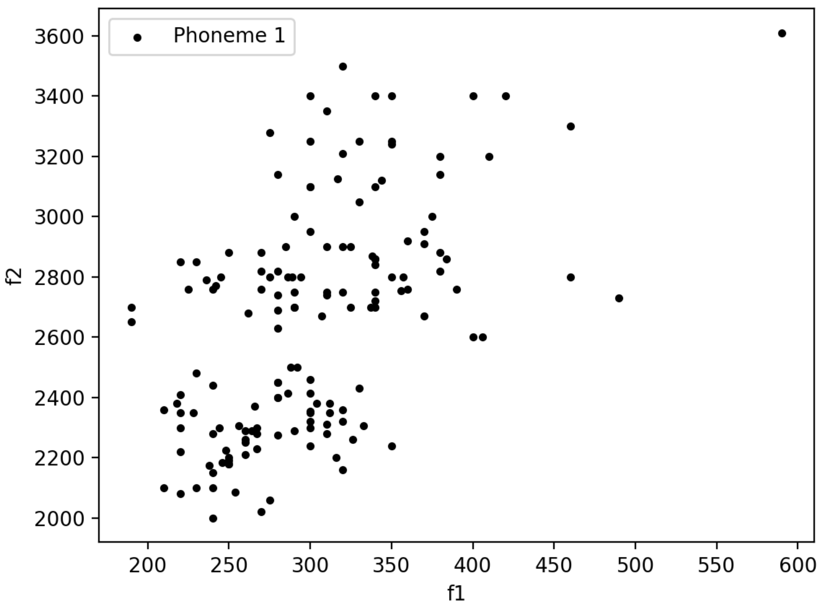


Fig. 2: Plot of F1 against F2, for all phoneme.

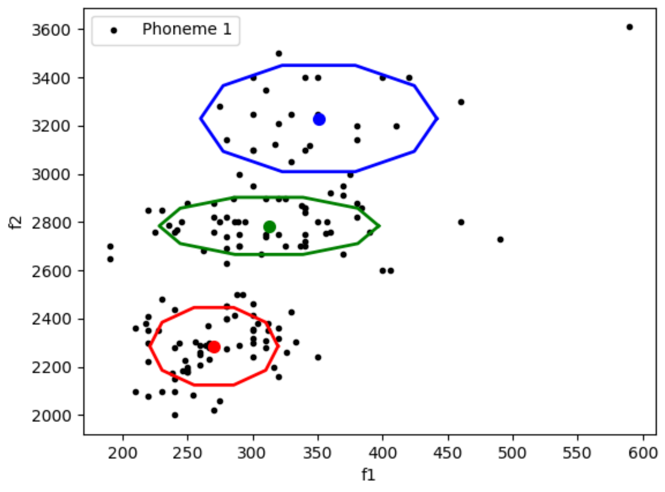
The snippet of code presented below corresponds to both plots previously displayed.

```
# 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))
j = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == p_id:
        X_phoneme_1[j,:] = X_full[i,:]
        j += 1
```

2 Task 2

Here we generate the matrices containing F1 and F2 for phonemes 1 and 2. Multiple runs of the model produce different clusters, as observed in Figure 3.



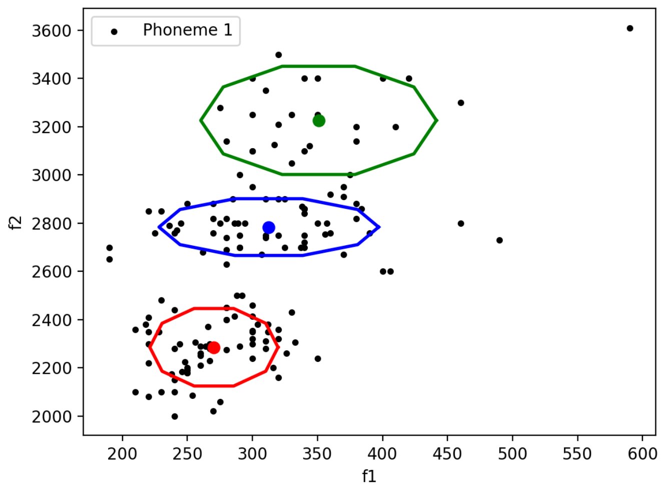


Fig. 3: Fist and second plot of F1 against F2, for phoneme 1, with k=3.

This is due to the random initialisation of its centroids to one of the observations in the dataset.

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

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

```python
X_phoneme = np.zeros((np.sum(phoneme_id==1), 2))
j = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == p_id:
        X_phoneme[j,:] = X_full[i,:]
        j += 1
k = 6
```

```python
X_phoneme = np.zeros((np.sum(phoneme_id==2), 2))
j = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == p_id:
        X_phoneme[j,:] = X_full[i,:]
        j += 1
k = 6
```

Presented above is the code contemplating all the combinations of phonemes and number of clusters requested.

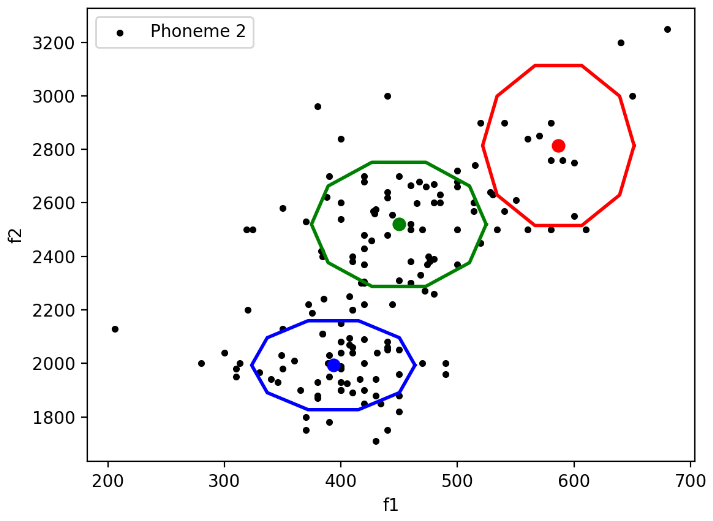


Fig. 4: Plot of F1 against F2, for phoneme 2, with k=3.

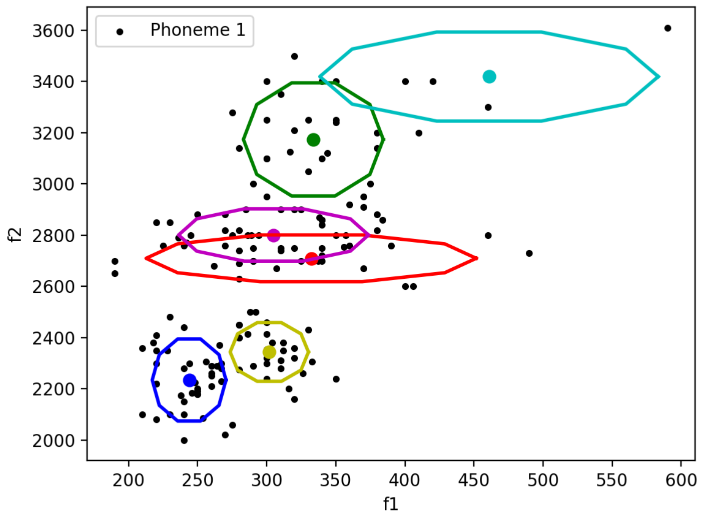


Fig. 5: Plot of F1 against F2, for phoneme 1, with k=6.

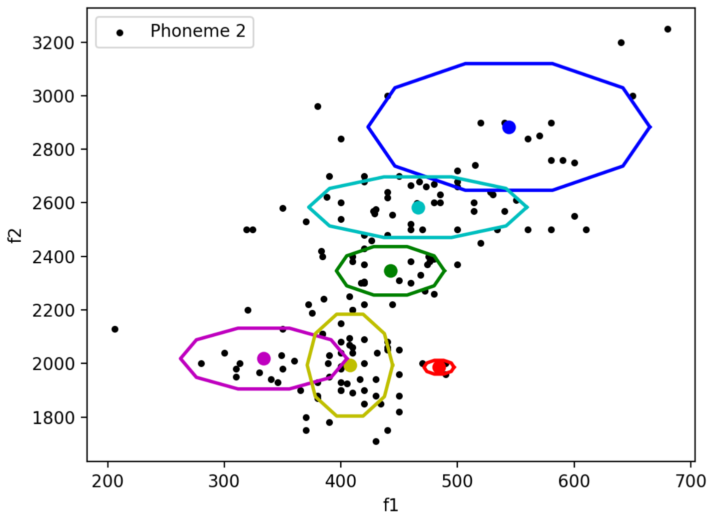


Fig. 6: Plot of F1 against F2, for phoneme 2, with k=6.

From Figure 3 to Figure 6 a display of the plots of all the mentioned combinations is presented. Below are the results for each scenario, containing the mean values, the covariances and the weights.

```
# Phoneme 1, k = 3, as in Figure 3
Implemented GMM | Mean values
[ 270.39416997 2285.45534778]
[ 312.74477369 2784.61186694]
[ 350.97547317 3230.08617236]

Implemented GMM | Covariances
[[ 1213.7726742      0.        ]
 [    0.         14277.05090375]]
[[3559.54425171    0.        ]
 [   0.         7757.88408394]]
[[ 4122.82079382     0.        ]
 [    0.         26877.3530009 ]]

Implemented GMM | Weights
[0.43512309 0.38318759 0.18168932]
```

```
# Phoneme 2, k = 3 as in Figure 4
Implemented GMM | Mean values
[ 586.56778306 2814.19771488]
[ 449.67824897 2519.69165651]
[ 393.45309392 1993.08471613]

Implemented GMM | Covariances
[[ 2111.39749335     0.        ]
 [    0.         49424.39995763]]
[[ 2809.48745631     0.        ]
 [    0.         29719.80829062]]
[[ 2454.89801558     0.        ]
 [    0.         15264.11909168]]

Implemented GMM | Weights
[0.09486996 0.46995855 0.43517149]
```

```
# Phoneme 1, k = 6 as in Figure 5
Implemented GMM | Mean values
[ 332.16903135 2709.76234915]
[ 333.74736457 3173.88520451]
[ 243.85160537 2234.59565559]
[ 460.89128546 3419.34102306]
[ 304.93612892 2800.80775007]
[ 301.87892837 2344.00182145]

Implemented GMM | Covariances
[[7136.77308156    0.        ]
 [   0.         4606.33835129]]
[[ 1280.30268144     0.        ]
 [    0.         26969.50533185]]
[[  356.18754273     0.        ]
 [    0.         14197.26048374]]
[[ 7498.84082926     0.        ]
 [    0.         16705.12418145]]
[[2360.79954973    0.        ]
 [   0.         5764.79840649]]
[[ 401.29235044    0.        ]
 [   0.         7257.1985719 ]]

Implemented GMM | Weights
[0.08993948 0.17273776 0.23553351 0.02498189 0.27803873 0.19876862]
```

```
# Phoneme 2, k = 6 as in Figure 6
Implemented GMM | Mean values
[ 484.13309039 1985.89622909]
[ 442.44542389 2345.89756328]
[ 544.05299028 2883.26564295]
[ 465.903534   2583.52744679]
[ 333.8084811  2018.19514181]
[ 407.69115991 1993.68886941]

Implemented GMM | Covariances
[[ 83.38225157   0.        ]
 [  0.         365.25894732]]
[[1073.7864407     0.        ]
 [   0.         4524.52585992]]
[[ 7300.27881082     0.        ]
 [    0.         30934.31218996]]
[[4373.86718196    0.        ]
 [   0.         7086.66090075]]
[[2547.96557067    0.        ]
 [   0.         7114.83879147]]
[[  670.53460317     0.        ]
 [    0.         19994.90504574]]

Implemented GMM | Weights
[0.01830464 0.14182344 0.10345577 0.30580117 0.10463661 0.32597837]
```

3 Task 3

Below is presented the code for calculating the accuracy and misclassification error in Task 3.

```
# Import parameters learnt in Task 2
GMM_params_phoneme_01_k_03 =
    np.ndarray.tolist(np.load('data/GMM_params_phoneme_01_k_03.npy',
                      allow_pickle=True))
GMM_params_phoneme_01_k_06 =
    np.ndarray.tolist(np.load('data/GMM_params_phoneme_01_k_06.npy',
                      allow_pickle=True))
GMM_params_phoneme_02_k_03 =
    np.ndarray.tolist(np.load('data/GMM_params_phoneme_02_k_03.npy',
                      allow_pickle=True))
GMM_params_phoneme_02_k_06 =
    np.ndarray.tolist(np.load('data/GMM_params_phoneme_02_k_06.npy',
                      allow_pickle=True))

# Predictions for phoneme 1, with three clusters
pred_ph1_k3 = get_predictions(GMM_params_phoneme_01_k_03['mu'],
                              GMM_params_phoneme_01_k_03['s'],
                              GMM_params_phoneme_01_k_03['p'],
                              X_phonemes_1_2)
pred_ph1_k3 = np.sum(pred_ph1_k3, axis=1)

# Predictions for phoneme 2, with three clusters
pred_ph2_k3 = get_predictions(GMM_params_phoneme_02_k_03['mu'],
                              GMM_params_phoneme_02_k_03['s'],
                              GMM_params_phoneme_02_k_03['p'],
                              X_phonemes_1_2)
pred_ph2_k3 = np.sum(pred_ph2_k3, axis=1)

# Predictions for phoneme 1, with six clusters
pred_ph1_k6 = get_predictions(GMM_params_phoneme_01_k_06['mu'],
                              GMM_params_phoneme_01_k_06['s'],
                              GMM_params_phoneme_01_k_06['p'],
                              X_phonemes_1_2)
pred_ph1_k6 = np.sum(pred_ph1_k6, axis=1)

# Predictions for phoneme 2, with six clusters
pred_ph2_k6 = get_predictions(GMM_params_phoneme_02_k_06['mu'],
                              GMM_params_phoneme_02_k_06['s'],
                              GMM_params_phoneme_02_k_06['p'],
                              X_phonemes_1_2)
pred_ph2_k6 = np.sum(pred_ph2_k6, axis=1)


# Create classification array for k = 3
class_k3 = np.zeros((pred_ph1_k3.shape[0],1))
for i in range(len(pred_ph1_k3)):
    if pred_ph1_k3[i] > pred_ph2_k3[i]:
        class_k3[i] = 1
    else:
        class_k3[i] = 2
```

```
# Create classification array for k = 6
class_k6 = np.zeros((pred_ph1_k6.shape[0],1))
for i in range(len(pred_ph1_k6)):
    if pred_ph1_k6[i] > pred_ph2_k6[i]:
        class_k6[i] = 1
    else:
        class_k6[i] = 2

# Count missclassifications for k = 3
n_missclass_k3 = 0
for i in range(len(class_k3)):
    if class_k3[i] != phoneme_1_2[i]:
        n_missclass_k3 += 1

# Count missclassifications for k = 6
n_missclass_k6 = 0
for i in range(len(class_k6)):
    if class_k6[i] != phoneme_1_2[i]:
        n_missclass_k6 += 1

# Calculate accuracy of our models for both three and six clusters
accuracy_k3 = (1 - (n_missclass_k3 / len(pred_ph1_k3))) * 100
accuracy_k6 = (1 - (n_missclass_k6 / len(pred_ph1_k3))) * 100
```

In order to determine the mis-classification error, first we import from Task 2 the parameters learnt for the two MoGs. Then, using the Maximum Likelihood criterion, we calculate the likelihood of a data vector (here including phonemes 1 and 2) for each of the two learnt MoGs. A vector (*class_k*) is thus created to store the classifications of each scenario (k=3 and k=6) by comparing the predicted values for both phonemes 1 and 2. The number of mis-classifications is stored and finally it is used to calculate the accuracy by dividing by the total number of samples in our input data vector. For three clusters an accuracy of 95.07% was found (mis-classification error of 4.93%), and of 95.72% for six clusters (mis-classification error of 4.28%).

4 Task 4

Differently from the process used to create the data vector *X_phonemes_1_2* in previous tasks, here it is important that the vector is composed of first all the phoneme 1 data points and then those of phoneme 2, as represented in the code below, due to the structure of the scatter plot process in lines 176 and 177 of *task_4.py*.

```
X_phonemes_1_2 = np.zeros((np.sum(phoneme_id==1) +
                           np.sum(phoneme_id==2), 2))

X_phoneme_1 = np.zeros((np.sum(phoneme_id==1), 2))
X_phoneme_2 = np.zeros((np.sum(phoneme_id==1), 2))
j1 = 0
j2 = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == 1:
        X_phoneme_1[j1,:] = X_full[i,:]
        j1 += 1
    if phoneme_id[i] == 2:
        X_phoneme_2[j2,:] = X_full[i,:]
        j2 += 1

X_phonemes_1_2 = np.vstack((X_phoneme_1,X_phoneme_2))
```

Furthermore, below is presented the code that generates the classification matrix, displayed if Figure 7.

```
# Import parameters learnt in Task 2
GMM_params_phoneme_01_k_03 =
    np.ndarray.tolist(np.load('data/GMM_params_phoneme_01_k_03.npy',
                      allow_pickle=True))
GMM_params_phoneme_02_k_03 =
    np.ndarray.tolist(np.load('data/GMM_params_phoneme_02_k_03.npy',
                      allow_pickle=True))

# Instantiate coordinates vector
x_1 = np.arange(min_f1, max_f1)
x_2 = np.arange(min_f2, max_f2)
x_1_2 = np.empty((1,2))

# Instantiate classification matrix
M = np.zeros((N_f2, N_f1))

# Calculate predictions for each point in the coordinate vector
# Fill the classification matrix
for i in range(len(x_1)):
    for j in range(len(x_2)):

        x_1_2[0,1] = x_1[i]
        x_1_2[0,0] = x_2[j]
```

```
# Predictions for phoneme 1, with three clusters
pred_ph1_k3 = get_predictions(GMM_params_phoneme_01_k_03['mu'],
                              GMM_params_phoneme_01_k_03['s'],
                              GMM_params_phoneme_01_k_03['p'],
                              X_phonemes_1_2)
pred_ph1_k3 = np.sum(pred_ph1_k3, axis=1)

# Predictions for phoneme 2, with three clusters
pred_ph2_k3 = get_predictions(GMM_params_phoneme_02_k_03['mu'],
                              GMM_params_phoneme_02_k_03['s'],
                              GMM_params_phoneme_02_k_03['p'],
                              X_phonemes_1_2)
pred_ph2_k3 = np.sum(pred_ph2_k3, axis=1)

# If phoneme 1, assign 0
# If phoneme 2, assign 1
if pred_ph1_k3 > pred_ph2_k3:
    M[j,i] = 0.0
else:
    M[j,i] = 1.0
```

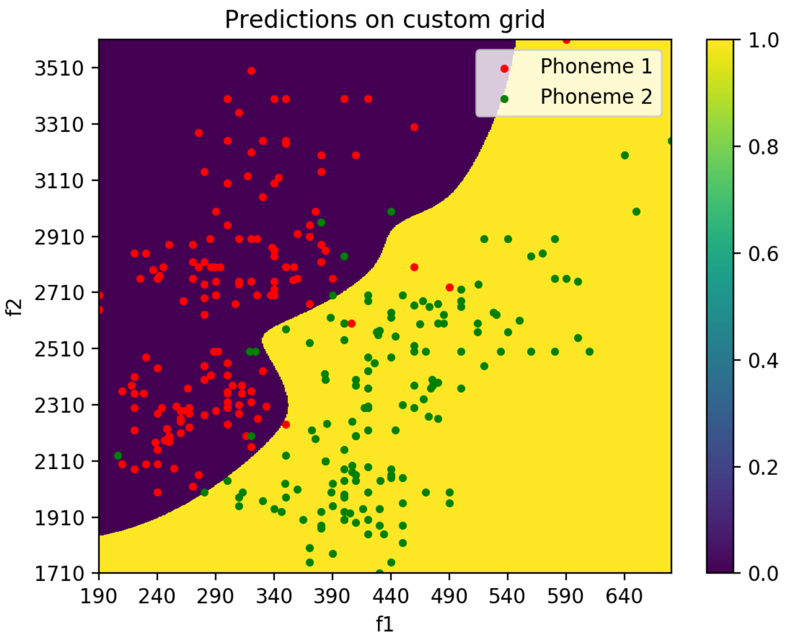


Fig. 7: Classification matrix.

5 Task 5

When running the model with the general gaussians form for our covariance matrix, there is a singular matrix error for the covariance matrix.

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

This means that its inverse cannot be calculated when computing the likelihood under the gaussian distribution's probability density function, because its determinant will be zero. This is caused by the third column F 1 + F 2 and the fact that it is composed of a linear combination of the first two columns.

```
X_full[:,0] = f1
X_full[:,1] = f2
X_full[:,2] = f1+f2
```

One possible solution to overcome this problem is demonstrated in the snippet of code bewlo, where some noise is added to the diagonal of the covariance matrix that contains the variance of each variable within the gaussian distribution itself.

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

This is done using an identity matrix multiplied by an epsilon, then added to the covariance matrix, thus generating a non-zero determinant. By doing so we are changing the eigenvalues of our matrix *s*. Increasing the variances has neglible impact on the likelihoods.
The results of the new MoG models, for both k=3 and k=6, are displayed in Figure 8 and Figure 9.

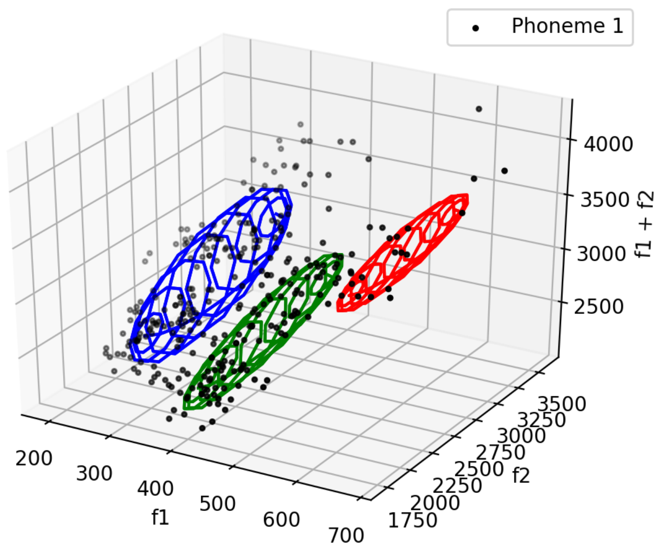


Fig. 8: MoG on J = [F1, F2, F1+F2], with k=3.

```
Implemented GMM | Mean values
[ 591.80332318 2759.11636054 3350.91968372]
[ 442.46019756 2273.83032624 2716.2905238 ]
[ 309.39516125 2586.51852351 2895.91368476]

Implemented GMM | Covariances
[[ 1751.50615321  7638.68373299  9389.8898862 ]
 [ 7638.68373299 62636.14428831 70274.52802129]
 [ 9389.8898862  70274.52802129 79664.71790749]]
[[  2338.09150605  11363.06304143  13700.95454747]
 [ 11363.06304143 112158.77259765 123521.63563908]
 [ 13700.95454747 123521.63563908 137222.79018656]]
[[  3376.03469139   7038.03006315  10413.96475454]
 [  7038.03006315 143402.83765101 150440.76771416]
 [ 10413.96475454 150440.76771416 160854.83246869]]

Implemented GMM | Weights
[0.04624768 0.35527273 0.59847959]
```

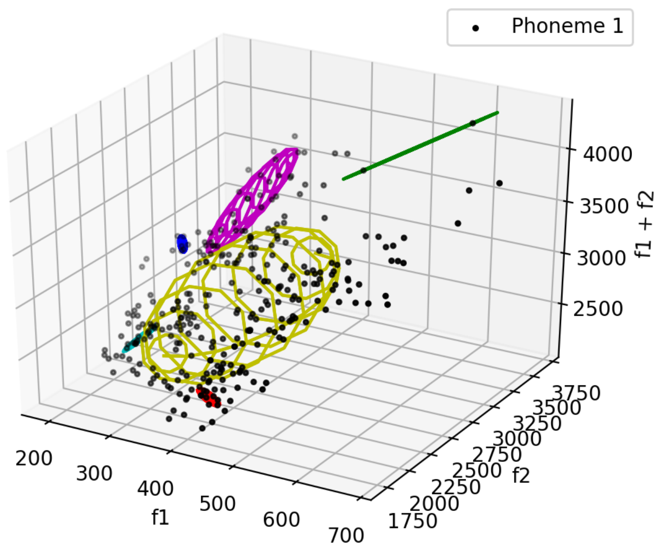


Fig. 9: MoG on J = [F1, F2, F1+F2], with k=6.

```
Implemented GMM | Mean values
[ 413.07262028 1892.73545182 2305.80807209]
[ 527.46378336 3460.87517563 3988.33895898]
[ 235.24964453 2802.54881124 3037.79845578]
[ 252.15904089 2213.10291751 2465.2619584 ]
[ 308.70172155 3089.96768791 3398.66940946]
[ 381.35598011 2451.13701232 2832.49299243]
```

```
Implemented GMM | Covariances
[[ 296.71457819 -520.0777729  -223.96319471]
 [-520.0777729  1206.24789328  685.57012038]
 [-223.96319471  685.57012038  462.20692567]]
[[ 4219.43109188 10060.52801372 14279.4591056 ]
 [10060.52801372 23990.98993988 34051.0179536 ]
 [14279.4591056  34051.0179536  48330.9770592 ]]
[[  68.22415416 -246.71085427 -178.88670012]
 [-246.71085427 1322.41403214 1075.30317787]
 [-178.88670012 1075.30317787  896.81647775]]
[[ 116.45852754  603.84616978  720.00469732]
 [ 603.84616978 3684.30712642 4287.8532962 ]
 [ 720.00469732 4287.8532962  5008.15799352]]
```

```
[[  659.429108    3185.40854394  3844.63765193]
 [ 3185.40854394 57606.51486717 60791.7234111 ]
 [ 3844.63765193 60791.7234111  64636.56106304]]
[[  9206.70180317   4644.7406631   13851.34246627]
 [  4644.7406631  122444.94691691 127089.58758001]
 [ 13851.34246627 127089.58758001 140941.03004628]]

Implemented GMM | Weights
[0.02612674 0.00632936 0.0155954  0.04556841 0.07175425 0.83462583]
```