Queen Mary, University of London
School of Electronic Engineering and Computer Science

ECS708U: Machine Learning

Assignment 2: Clustering and Mixture of Gaussians (MoG)

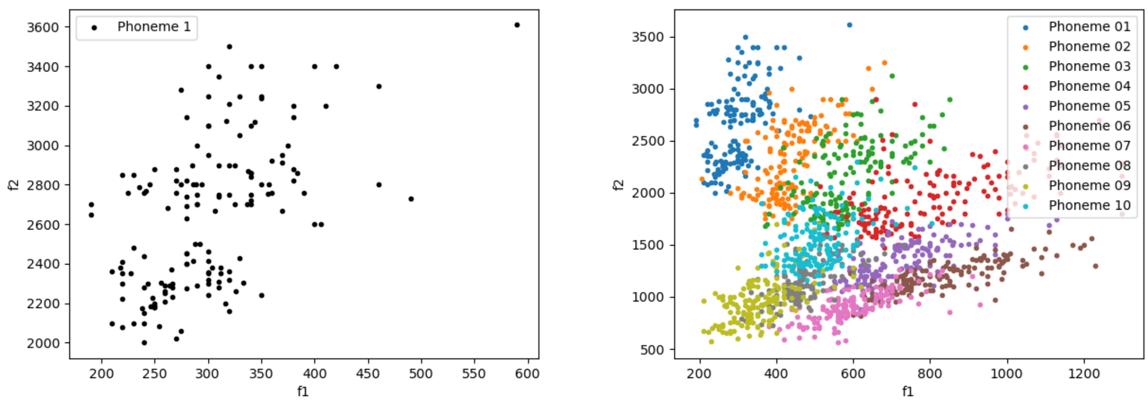
Mughees Asif | 180288337

Due: 12 October 2021

Task 1

```python
# 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 = X_full[phoneme_id == p_id, :]

```



(a) Phoneme 1

(b) All phonemes

Figure 1: Fundamental frequencies, $f1$ against $f2$

```
f1 statistics:
Min: 190.00 Mean: 563.30 Max: 1300.00 Std: 201.1881 | Shape: 1520
f2 statistics:
Min: 560.00 Mean: 1624.38 Max: 3610.00 Std: 636.8032 | Shape: 1520
```

Task 2

```
# 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 = X_full[phoneme_id == p_id, :]

```

$k = 3$

First run

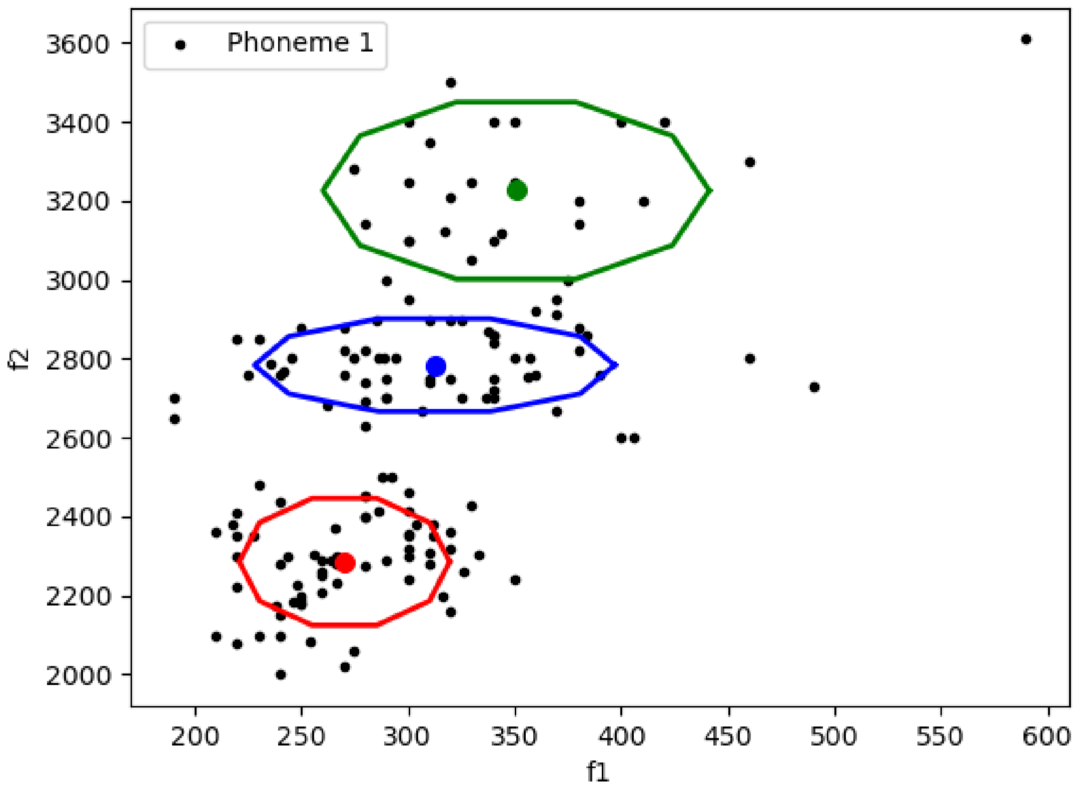


Figure 2: Gaussian phoneme, $k = 3$

```
Implemented GMM | Mean values
[ 270.3952  2285.4653]
[ 350.8446  3226.3394]
[ 312.59125 2783.898  ]

Implemented GMM | Covariances
[[ 1213.73843494     0.        ]
 [    0.         14278.42029955]]
[[ 4102.87537495     0.        ]
 [    0.         27829.54221723]]
[[3562.59743765    0.        ]
 [   0.         7657.84897216]]

Implemented GMM | Weights
[0.43514434 0.18386038 0.38099528]
```

Second run

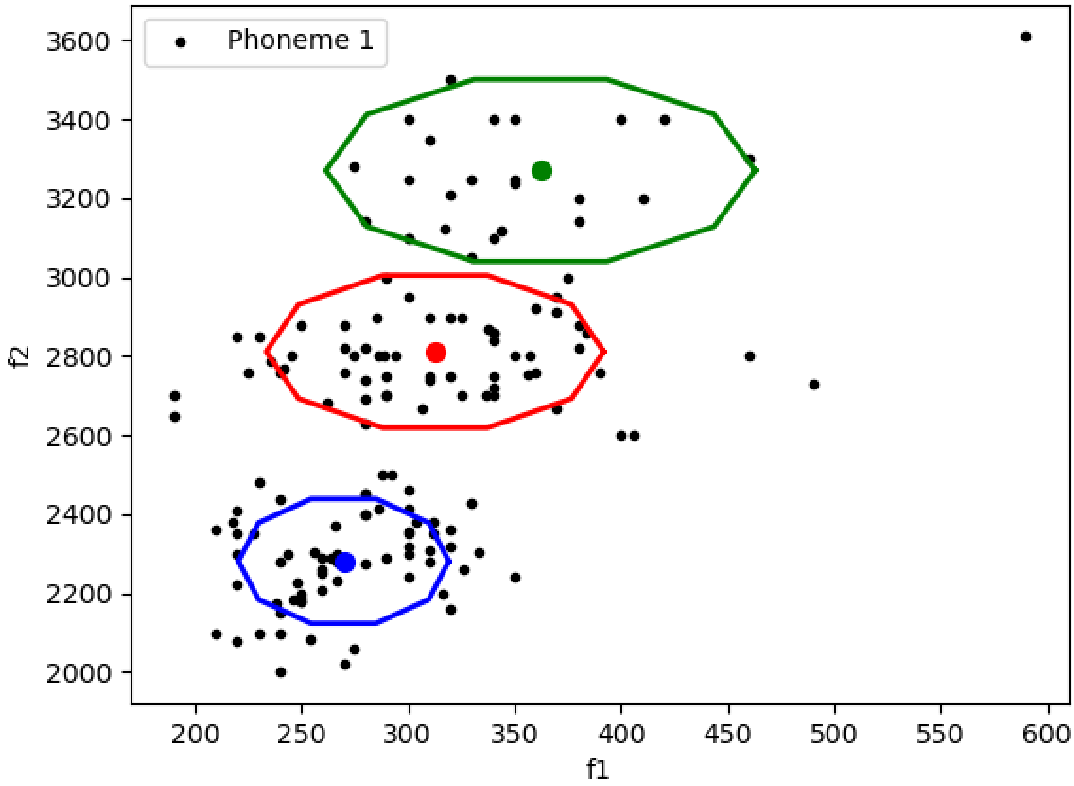


Figure 3: Gaussian phoneme, $k = 3$

```
Implemented GMM | Mean values
[ 312.7931 2811.8486]
[ 362.2751 3270.6072]
[ 269.92792 2281.0254 ]

Implemented GMM | Covariances
[[ 3133.01971418     0.        ]
 [    0.         20592.59093668]]
[[ 5064.81166948     0.        ]
 [    0.         29330.22701612]]
[[ 1215.82904825     0.        ]
 [    0.         13740.86780726]]

Implemented GMM | Weights
[0.44128011 0.13448926 0.42423063]
```

$k = 6$

First run

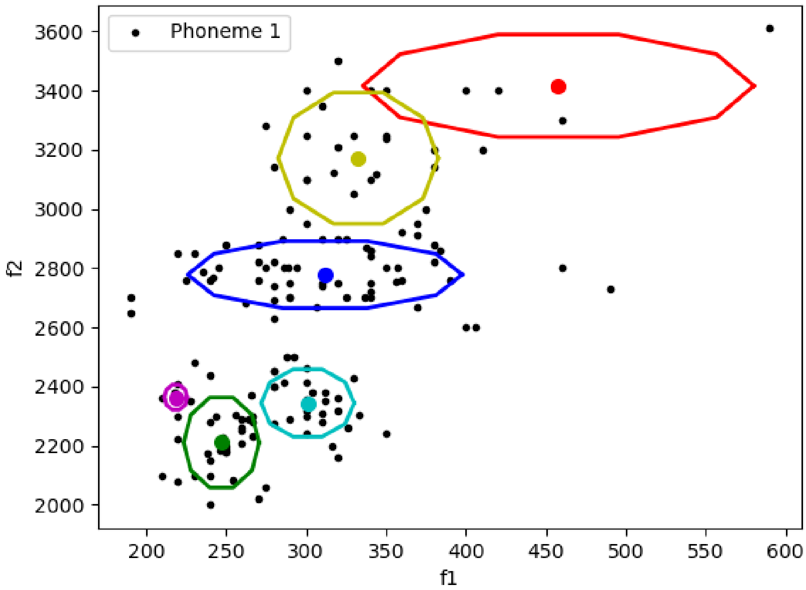


Figure 4: Gaussian phoneme, $k = 6$

```
Implemented GMM | Mean values
[ 457.8011 3416.4023]
[ 247.18047 2210.3774 ]
[ 311.85373 2778.3267 ]
[ 301.0684 2344.1323]
[ 218.6318 2363.589 ]
[ 332.71387 3171.762  ]

Implemented GMM | Covariances
[[ 7474.90630756      0.        ]
 [     0.         16556.63598171]]
[[  279.05520316      0.        ]
 [     0.         12839.83287966]]
[[3684.31355542     0.        ]
 [    0.          7070.6920406 ]]
[[ 421.65474188     0.        ]
 [    0.          7209.39225671]]
[[ 25.51633687    0.        ]
 [   0.          994.57566331]]
[[ 1251.69581206      0.        ]
 [     0.         27146.11552164]]

Implemented GMM | Weights
[0.02607926 0.19808257 0.36774705 0.20403745 0.03192522 0.17212846]
```

Second run

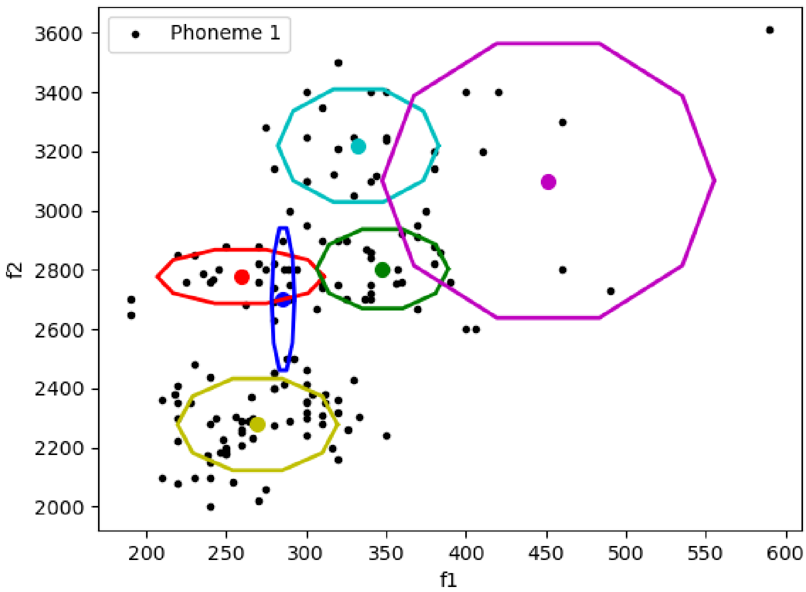


Figure 5: Gaussian phoneme, $k = 6$

```
Implemented GMM | Mean values
[ 259.19806 2777.2854 ]
[ 347.8378  2803.3938]
[ 285.69077 2701.1602 ]
[ 332.73532 3219.3318 ]
[ 451.44778 3101.4407 ]
[ 269.71927 2277.7375 ]

Implemented GMM | Covariances
[[1360.26534051    0.        ]
 [   0.         4540.09031381]]
[[ 845.34786211    0.        ]
 [   0.         9868.5388874 ]]
[[2.60575697e+01 0.00000000e+00]
 [0.00000000e+00 3.17874345e+04]]
[[ 1264.84075501     0.        ]
 [    0.         20058.38684489]]
[[  5387.36721704      0.        ]
 [     0.         118675.67982747]]
[[ 1252.55905729     0.        ]
 [    0.         13264.79104545]]

Implemented GMM | Weights
[0.13982013 0.18456919 0.06574364 0.14693026 0.04560488 0.41733191]
```

Task 3

```
# 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 containing only samples that belong to phoneme 1 and samples that belong to phoneme 2
X_phonemes_1_2 = X_full[np.logical_or(phoneme_id == 1, phoneme_id == 2), :]
...
# Get predictions on samples from a GMM with k components, pretrained on phoneme 1
phoneme_model_1 = 'data/GMM_params_phoneme_{:02}_k_{:02}.npy'.format(1, k)
params_1 = np.load(phoneme_model_1, allow_pickle=True).item()
copy = X_phonemes_1_2.copy()
Z_1 = get_predictions(
    params_1['mu'],
    params_1['s'],
    params_1['p'],
    copy
)
pred_1 = Z_1.sum(axis=1)

# Get predictions on phoneme 2
phoneme_model_2 = 'data/GMM_params_phoneme_{:02}_k_{:02}.npy'.format(2, k)
params_2 = np.load(phoneme_model_2, allow_pickle=True).item()
Z_2 = get_predictions(
    params_2['mu'],
    params_2['s'],
    params_2['p'],
    copy
)
pred_2 = Z_2.sum(axis=1)

# Compare these predictions for each sample of the dataset, and calculate the accuracy, and store it in a scalar variable named "accuracy"
predictions = np.ones(len(copy)) * 2
predictions[pred_1 >= pred_2] = 1
labels = phoneme_id[np.logical_or(phoneme_id == 1, phoneme_id == 2)]
accuracy = np.sum(predictions == labels) / copy.shape[0] * 100

```

Results

```
Accuracy using GMMs with 3 components: 96.38%
Accuracy using GMMs with 6 components: 95.72%
```

Observations

The above code is structured as:

1. **Line 2–4**: Copy the relevant frequency values.
2. **Line 6**: Fill `X_phonemes_1_2` with the samples of `X_full` that belong to the chosen phonemes.
3. **Line 9–10**: Load the pretrained GMM model for $k = 3$ & 6.
4. **Line 11**: Make a copy of the `X_phonemes_1_2` to get the samples for `p_id`.
5. **Line 12–17**: Get the predictions (likelihood) for both phonemes. Sum up the predictions and save in `pred_i`, $i = 1 \mid 2$.
6. **Line 21–29**: Repeat steps 3–5 for the second phoneme.
7. **Line 33–36**: Initialise the `predictions` array that holds the cumulative likelihood from the individually summed up predictions from step 5. Use the information to calculate the accuracy %.

The GMMs trained on 3 parameters were found to train faster and exhibit increased accuracy. The GMMs with 6 parameters were found to exhibit a reduction in accuracy due to overfitting and took longer to train.

Task 4

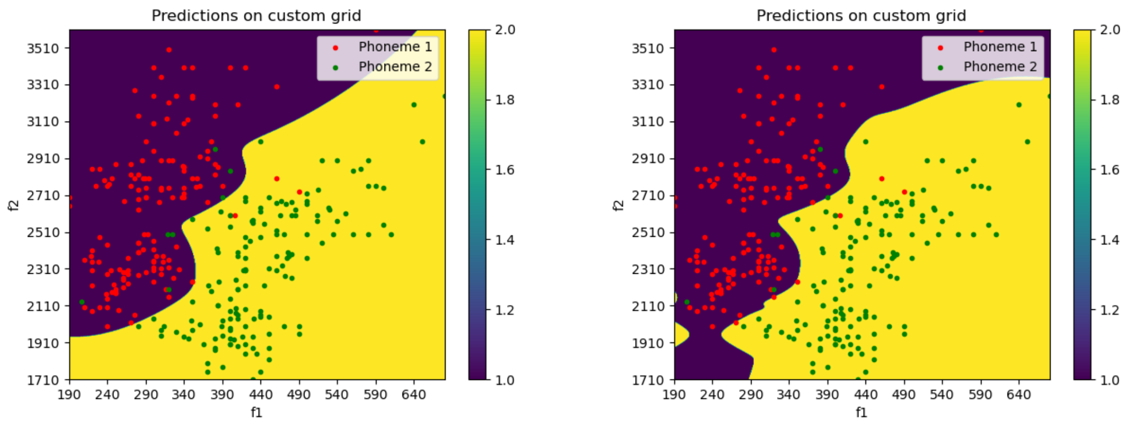
```
# Store f1 in the first column of X_full, and f2 in the second column of X_full
X_full[:, 0] = f1
X_full[:, 1] = f2
...
# Create an array containing only samples that belong to phoneme 1 and samples that belong to phoneme 2
X_phonemes_1_2 = X_full[np.logical_or(phoneme_id == 1, phoneme_id == 2), :]
...
# Generate grid
ax_f1 = np.linspace(min_f1, max_f1, N_f1)
ax_f2 = np.linspace(min_f2, max_f2, N_f2)
x_axis, y_axis = np.meshgrid(ax_f1, ax_f2)
samples = np.stack((x_axis.flatten(), y_axis.flatten())).transpose()


# Phoneme model no. 1
phoneme_model_1 = 'data/GMM_params_phoneme_{:02}_k_{:02}.npy'.format(1, k)
params_1 = np.load(phoneme_model_1, allow_pickle=True)
params_1 = np.ndarray.tolist(params_1)
copy = samples.copy()
Z_1 = get_predictions(
    params_1['mu'],
    params_1['s'],
    params_1['p'],
    copy
)

# Phoneme model no. 2
phoneme_model_2 = 'data/GMM_params_phoneme_{:02}_k_{:02}.npy'.format(2, k)
params_2 = np.load(phoneme_model_2, allow_pickle=True)
params_2 = np.ndarray.tolist(params_2)
Z_2 = get_predictions(
    params_2['mu'],
    params_2['s'],
    params_2['p'],
    copy
)

# Get the predictions
pred_1 = Z_1.sum(axis=1)
pred_2 = Z_2.sum(axis=1)
predictions = np.ones(len(copy)) * 2
predictions[pred_1 >= pred_2] = 1
```

```
44      # M
45      M = predictions.reshape(N_f2, N_f1)
46      ...
47      # Print confusion matrix
48      ids = phoneme_id[np.isin(phoneme_id, [1, 2])]
49      X1 = X[ids == 1]
50      X2 = X[ids == 2]
51      plt.scatter(X1[:, 0] - min_f1, X1[:, 1] - min_f2, marker='.',
        ↪  color='red', label='Phoneme 1')
52      plt.scatter(X2[:, 0] - min_f1, X2[:, 1] - min_f2, marker='.',
        ↪  color='green', label='Phoneme 2')
53
```



(a) $k = 3$

(b) $k = 6$

Figure 6: Confusion matrix for k components

The above code is structured as:

1. **Line 2–4**: Copy the relevant frequency values.
2. **Line 6**: Fill `X_phonemes_1_2` with the samples of `X_full` that belong to the chosen phonemes.
3. **Line 9–16**: Create the grid using linearly spaced vectors.
4. **Line 19–45**: Same as explained in Task 3: Observations.
5. **Line 45**: Create `M` array containing 0 in the points that belong to phoneme 1, and 1 in the points that belong to phoneme 2. `M` forms the background for the confusion matrices highlighted in Fig. 6.
6. **Line 51–55**: Print the confusion matrix after sorting the values for the colour assignment.

Task 5

```python
    # Store f1 in the first column, f2 in the second column, and (f1 + f2) in
    ↪ the third column of X_full
        X_full[:, 0] = f1
        X_full[:, 1] = f2
        X_full[:, 2] = f1 + f2
    ...
    # Create array containing only samples that belong to phoneme 1
    X_phoneme_1 = X_full[phoneme_id == p_id, :]
    ...
    # Add regularisation to offset singularity
    s[1,:,:] += 0.001 * np.identity(D)
```

Errors

```
Iteration 110/150
Iteration 111/150
Iteration 112/150
C:\Users\user\lib\site-packages\matplotlib\cbook\__init__.py:1333:
    ComplexWarning: Casting complex values to real discards the
    imaginary part
return np.asarray(x, float)
C:\Users\user\lib\site-packages\matplotlib\cbook\__init__.py:1333:
    \linebreak ComplexWarning: Casting complex values to real discards
    the imaginary part
return np.asarray(x, float)
C:\Users\user\lib\site-packages\matplotlib\cbook\__init__.py:1333:
    ComplexWarning: Casting complex values to real discards the
    imaginary part
return np.asarray(x, float)
C:\Users\user\lib\site-packages\matplotlib\cbook\__init__.py:1333:
    ComplexWarning: Casting complex values to real discards the
    imaginary part
return np.asarray(x, float)
Iteration 113/150
Iteration 114/150
```

$k = 3$

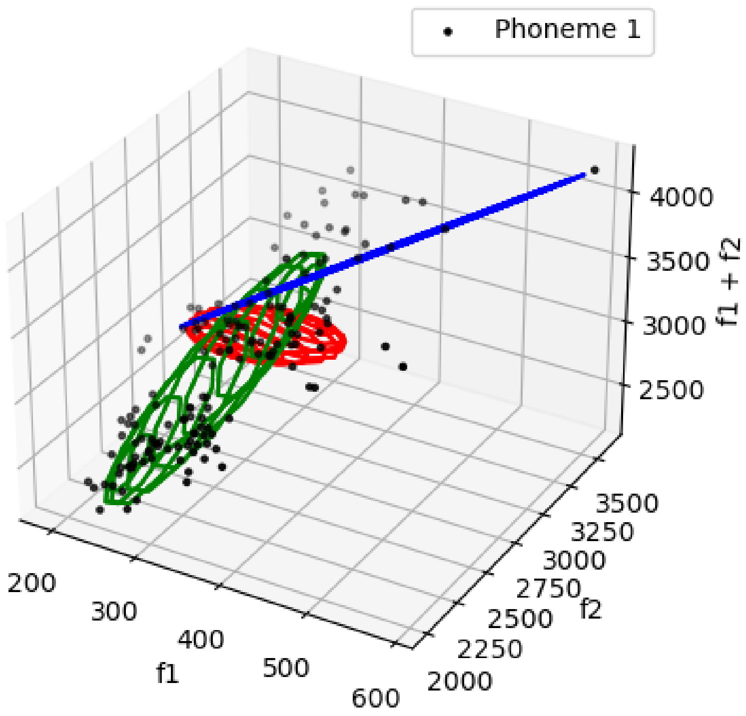


Figure 7: Full covariance matrices fit to new data

```
Implemented GMM | Mean values
[ 320.95685 2779.7197   3100.6765 ]
[ 290.86166 2585.0154   2875.8772 ]
[ 400.65002 3176.3762   3577.0261 ]

Implemented GMM | Covariances
[[ 4614.61035882   -445.20477564   4169.40558318]
 [ -445.20477564   6560.32199013   6115.11721449]
 [ 4169.40558318   6115.11721449 10284.52279768]]
[[   1996.20052277   10892.01222129   12888.21274406]
 [ 10892.01222129 161071.73613414 171963.74835543]
 [ 12888.21274406 171963.74835543 184851.96109951]]
[[ 16314.08843417   37052.91712156   53367.00555573]
 [ 37052.91712156   84205.97468187 121258.89180342]
 [ 53367.00555573 121258.89180342 174625.89735916]]

Implemented GMM | Weights
[0.22433881 0.74241566 0.03324552]
```

$k = 6$

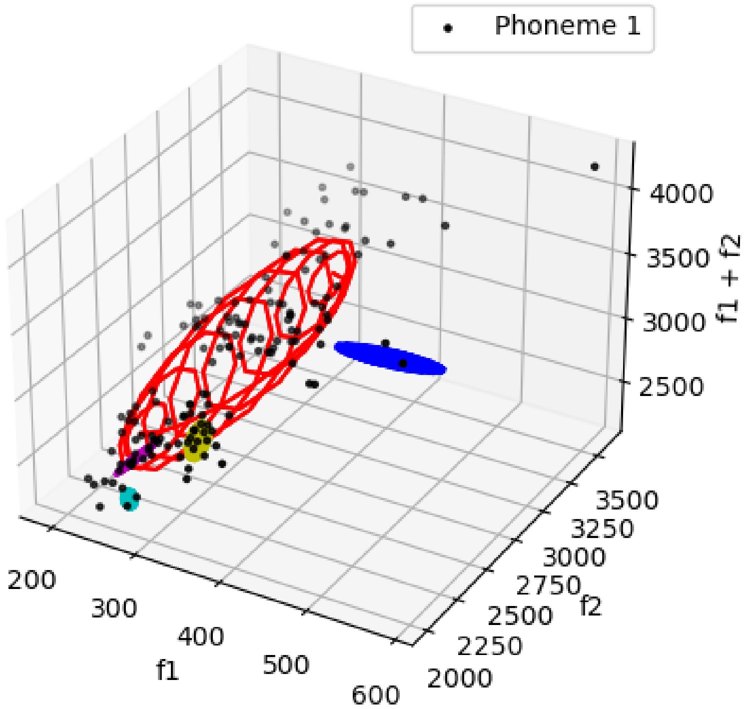


Figure 8: Full covariance matrices fit to new data

```
Implemented GMM | Mean values

[ 303.02448 2678.7605   2981.785   ]
[ 280.  2400.  2680.]
[ 472.704   2742.6934  3215.3975]
[ 268.2653  2047.8208  2316.0862]
[ 251.20546 2205.8057   2457.0112 ]
[ 310.13544 2312.8076   2622.943   ]

Implemented GMM | Covariances
[[   3588.25985186   11809.41486645   15397.67471831]
 [  11809.41486645  137364.33602465  149173.75089109]
 [  15397.67471831  149173.75089109  164571.42560939]]
[[0.005 0.    0.   ]
 [0.    0.005 0.   ]
 [0.    0.    0.005]]
[[2228.82332945  -682.77261291  1546.05071654]
 [-682.77261291   750.88929805    68.11668514]
 [1546.05071654    68.11668514  1614.16740169]]
[[   63.45874286  -110.63331313   -47.17457027]
 [-110.63331313   694.31867819   583.68536506]
 [  -47.17457027   583.68536506   536.5107948  ]]
[[   92.83688005   459.37117444   552.20805449]
 [  459.37117444  2730.52641071  3189.89758515]
```

```
 [ 552.20805449 3189.89758515 3742.10563965]]
[[ 106.54427401 -223.97845157 -117.43417756]
 [-223.97845157 3181.81198929 2957.83353772]
 [-117.43417756 2957.83353772 2840.39936016]]

Implemented GMM | Weights
[0.92474453 0.01137863 0.00368719 0.00426987 0.0402883  0.01563148]
```

Observations

The errors displayed in Errors section display the singularity problem. The issue arises from the covariance matrices that were previously using zero as non-diagonal entries. A singular matrix e.g., A does not have an inverse, where $\det(A) = 0$. The program tries to use the inverse of the covariance matrices that can not exist, and therefore kept timing out.

The problem was fixed by implementing regularisation to the covariance matrices highlighted in Line **10** of the code sample for this task. A diagonal matrix with the equal dimensions is represented as an identity matrix and multiplied with a value of 0.001 to force the diagonal entries to never reach 0.