Task 1

Code

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

In the image above, the code is setting the frequency values in X_full.

```
X_phoneme_1 = X_full[phoneme_id == p_id, :]
```

In the image above, the code fills the variable X_phoneme_1 with sample of the chosen phonemes.

Graphs

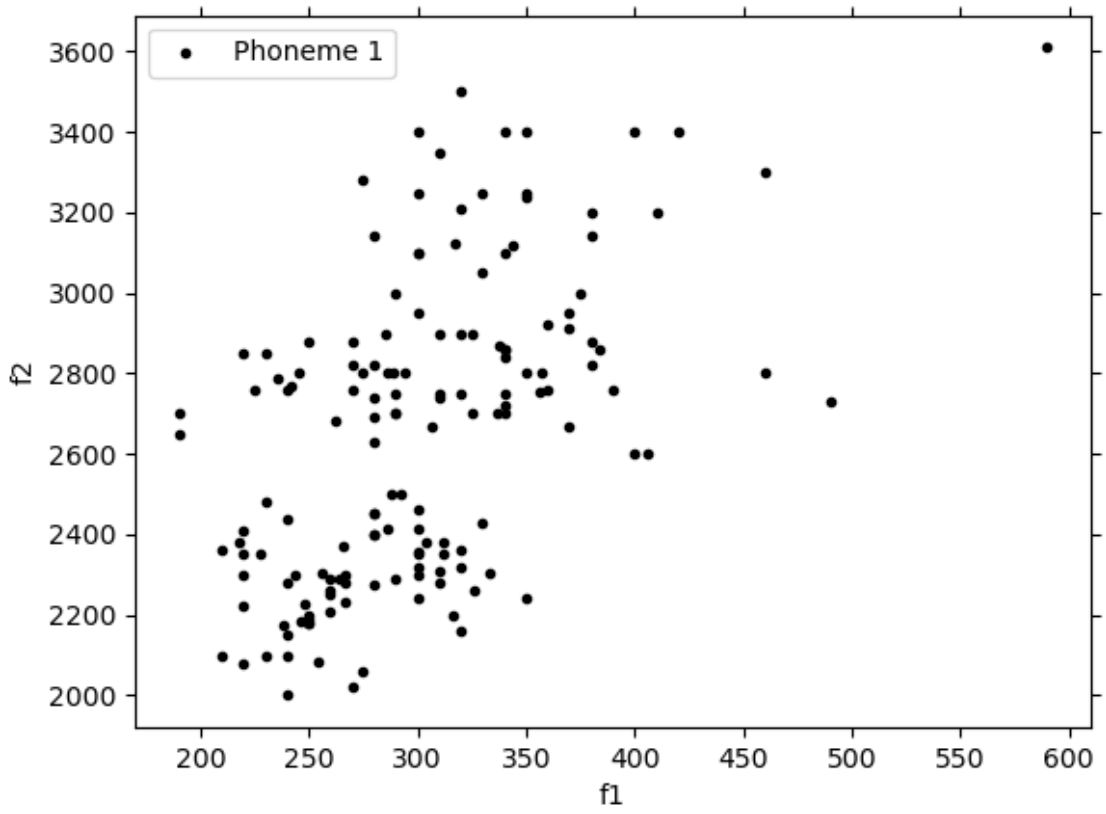


Figure 1 - Phoneme 1, f1 vs f2

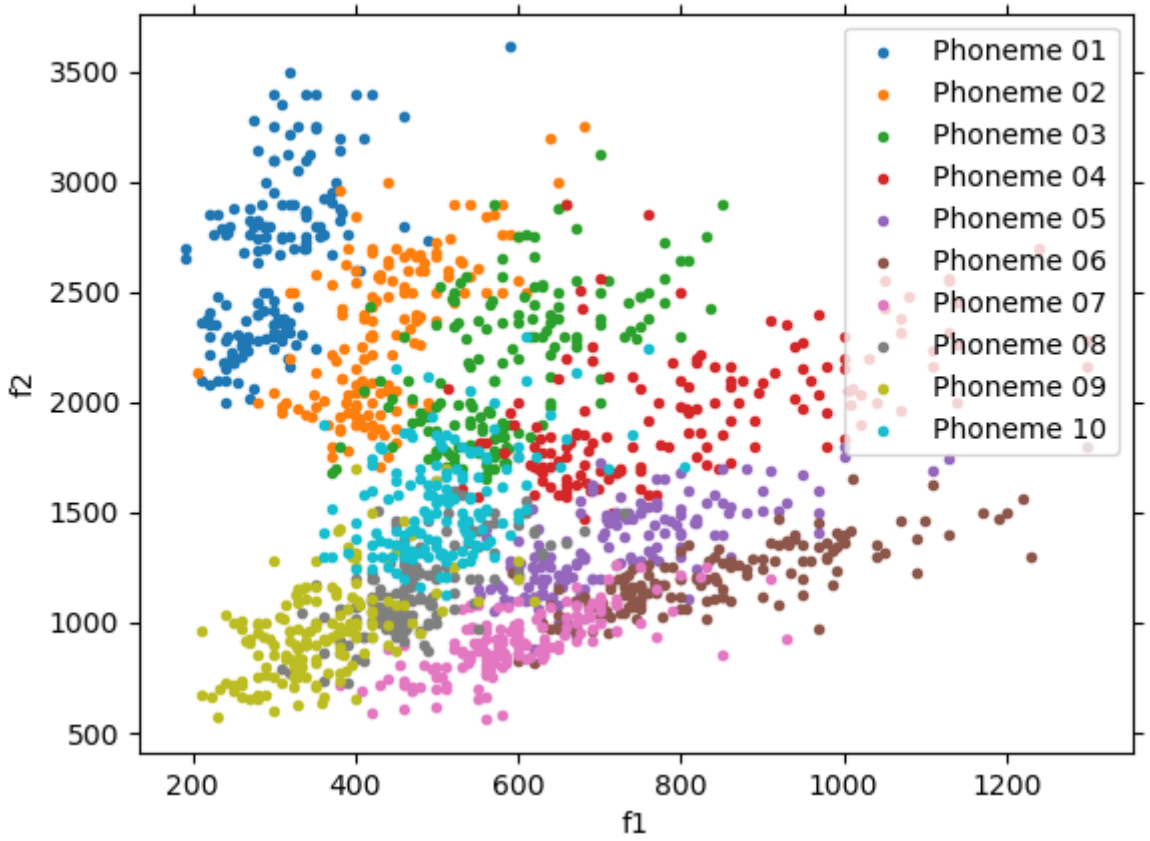


Figure 2 - All the phonemes, f1 vs f2

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

Observations

In figure 2, we can see all the phonemes from 1 to 10 and their frequencies.

Task 2

Code

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

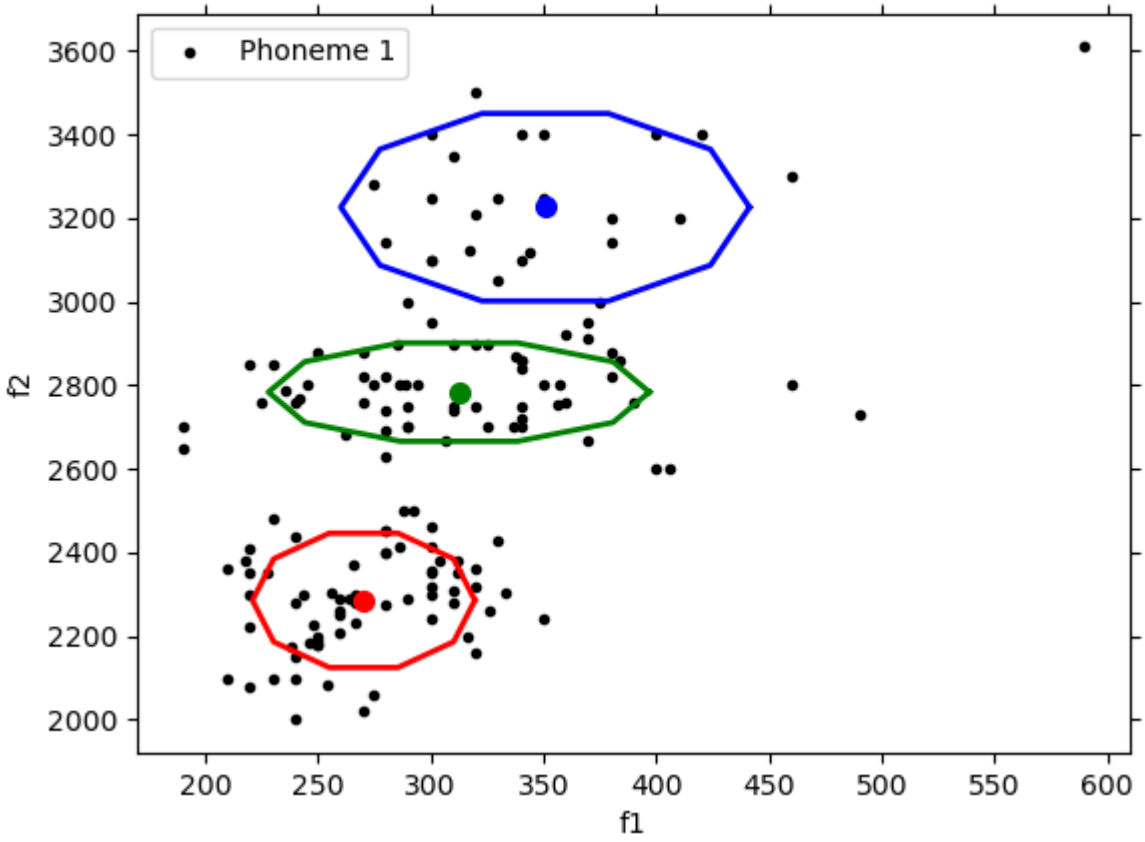
In the image above, the code is setting the frequency values in X_full.

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

In the image above, the code fills the variable X_phoneme with sample of the chosen phonemes.

Phoneme 1

First Run (k=3)



Implemented GMM | Mean values

[270.3952 2285.4653]

[312.59125 2783.898]

[350.8446 3226.3394]

Implemented GMM | Covariances

[[1213.73843494 0.]

[0. 14278.42029945]]

[[3562.59743764 0.]

[0. 7657.84897274]]

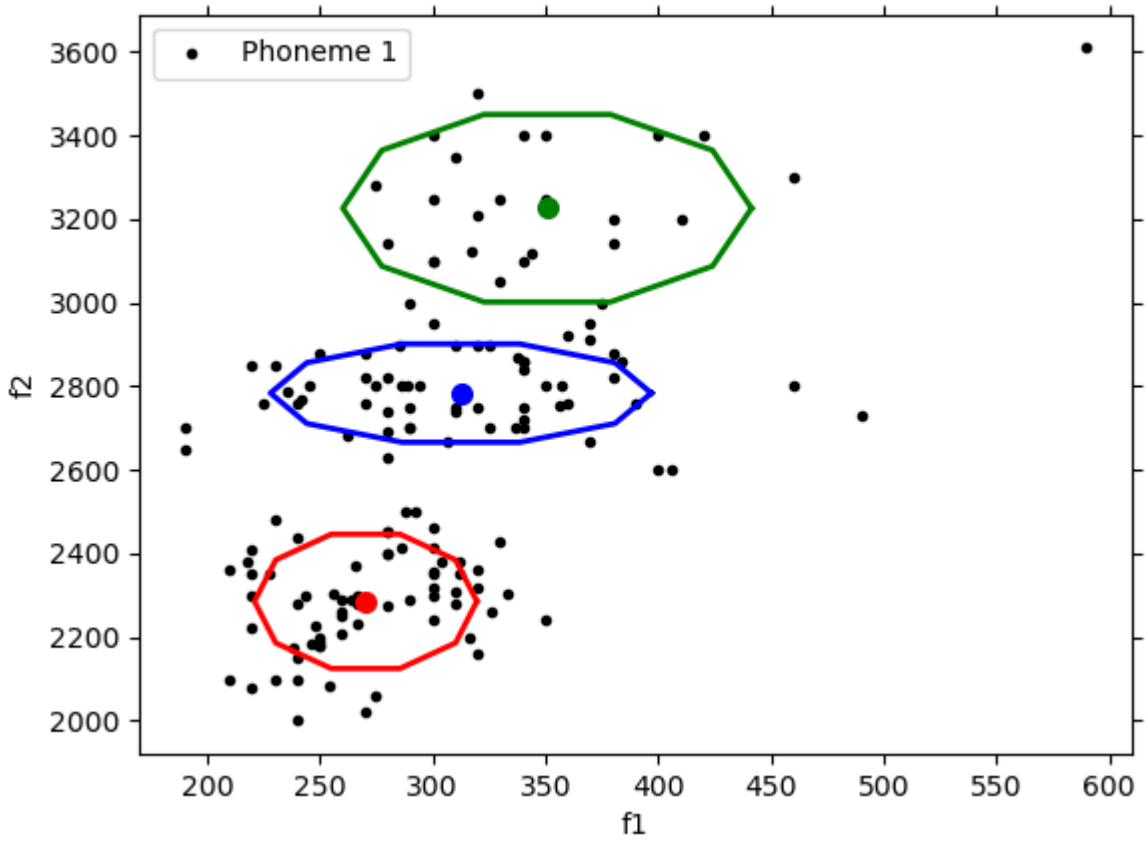
[[4102.87537505 0.]

[0. 27829.54221127]]

Implemented GMM | Weights

[0.43514434 0.38099528 0.18386038]

Second Run (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. 14278.4202995]]

[[4102.875375 0.]

[0. 27829.54221422]]

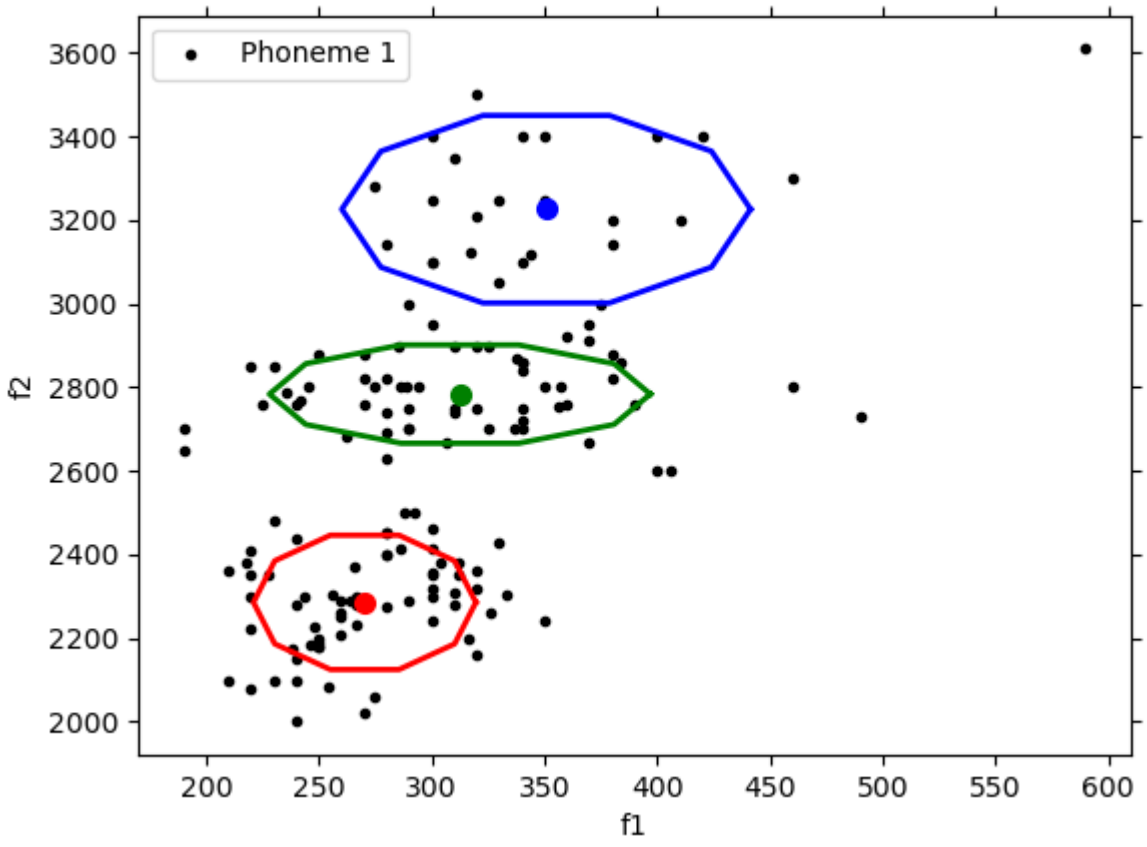
[[3562.59743765 0.]

[0. 7657.84897245]]

Implemented GMM | Weights

[0.43514434 0.18386038 0.38099528]

Third Run (K=3)



Implemented GMM | Mean values

[270.3952 2285.4653]

[312.59122 2783.8977]

[350.8446 3226.3389]

Implemented GMM | Covariances

[[1213.73843084 0.]

 [0. 14278.41959241]]

[[3562.59793301 0.]

 [0. 7657.83387572]]

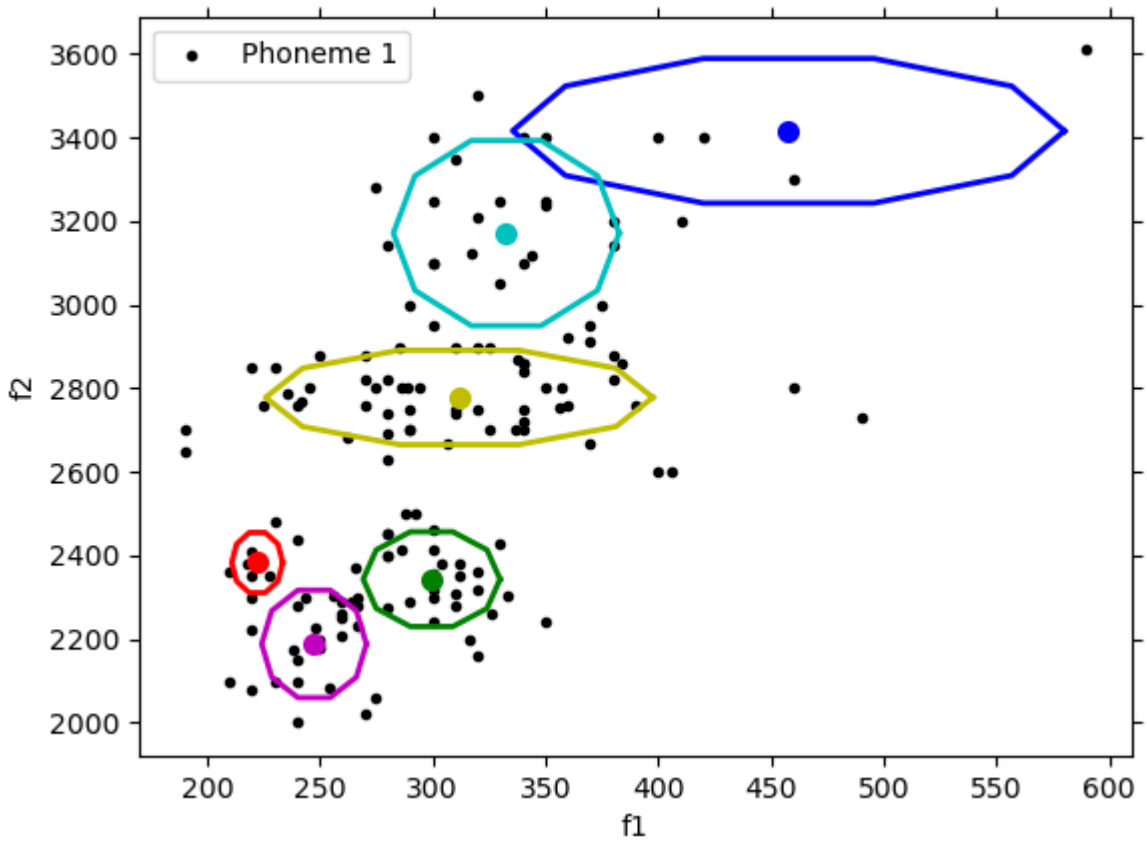
[[4102.87224614 0.]

 [0. 27829.68679975]]

Implemented GMM | Weights

[0.43514434 0.38099495 0.18386072]

First Run (K=6)



Implemented GMM | Mean values

[222.3631 2382.9187]

[299.63318 2343.3608]

[457.80066 3416.4023]

[332.7139 3171.6262]

[247.53249 2188.3271]

[311.8624 2778.3464]

Implemented GMM | Covariances

[[64.00201707 0.]

[0. 2870.80023051]]

[[460.06673632 0.]

[0. 7115.96079062]]

[[7474.82664225 0.]

[0. 16556.04287588]]

[[1251.67411284 0.]

[0. 27170.43924083]]

[[269.03937043 0.]

[0. 9147.83526928]]

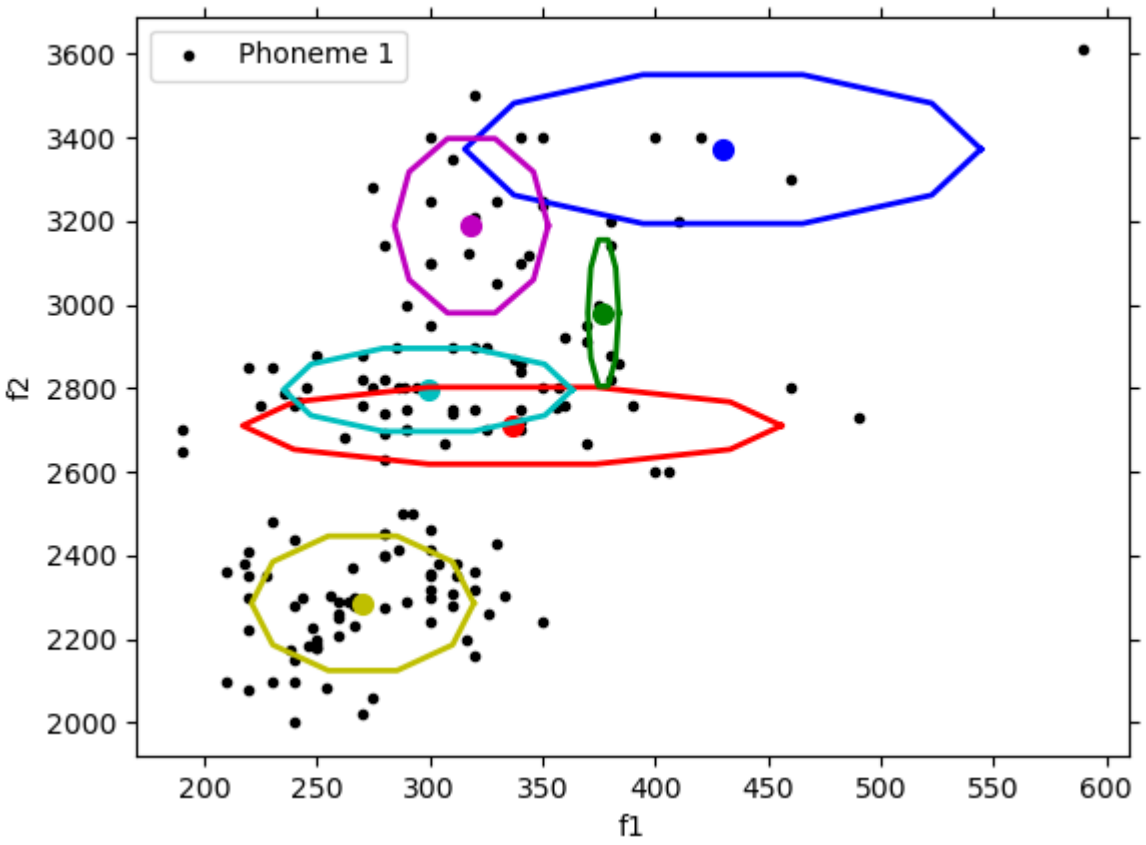
[[3684.2091033 0.]

[0. 7051.05230144]]

Implemented GMM | Weights

[0.04534776 0.21248999 0.02607958 0.172218 0.17626443 0.36760023]

Second Run (K=6)



Implemented GMM | Mean values

[336.59525 2710.763]

[377.0324 2980.4253]

[430.00836 3372.529]

[299.29004 2796.5022]

[318.5427 3189.148]

[270.4053 2285.6204]

Implemented GMM | Covariances

[[7129.66144859 0.]

[0. 4681.98856181]]

[[23.33746115 0.]

[0. 16909.43579972]]

[[6543.52671583 0.]

[0. 17538.12263736]]

[[2047.39972131 0.]

[0. 5458.39023789]]

[[581.21814699 0.]

[0. 24068.40836818]]

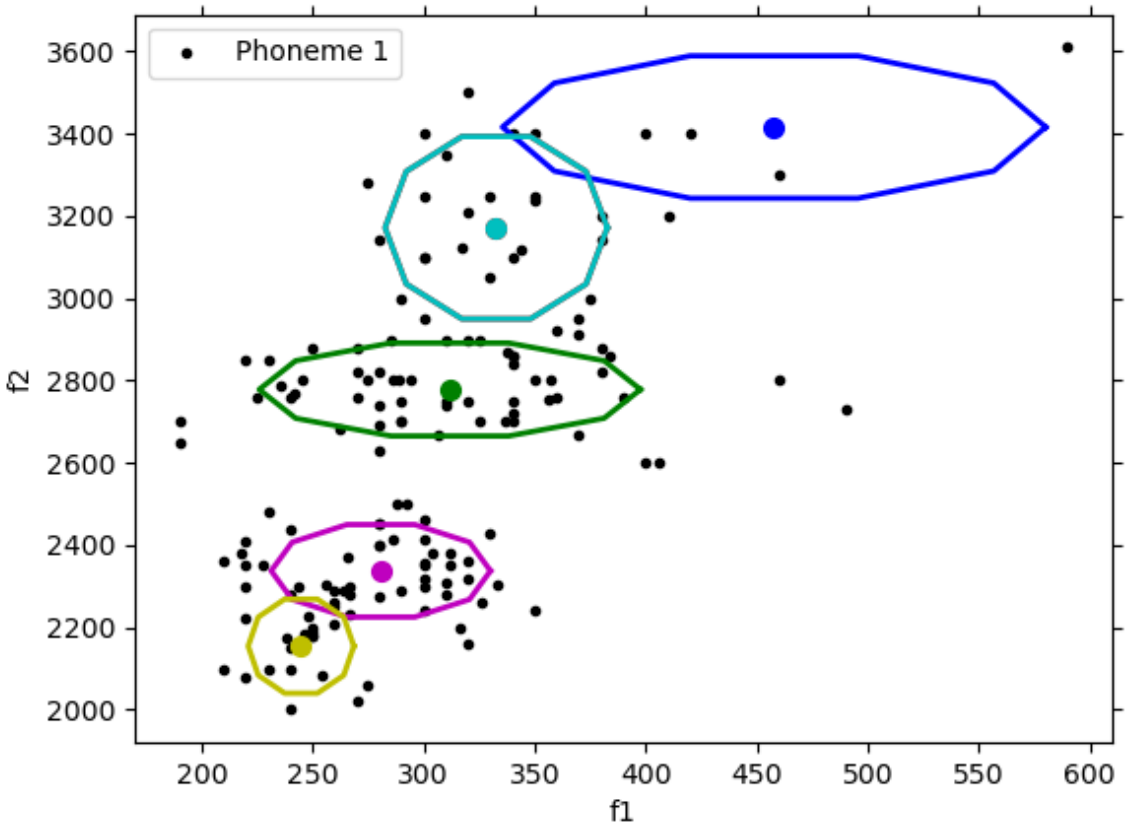
[[1213.13228135 0.]

[0. 14308.02185058]]

Implemented GMM | Weights

[0.09029317 0.04210081 0.04162607 0.26237542 0.12817427 0.43543027]

Third Run (K=6)



Implemented GMM | Mean values

[332.71445 3171.7598]

[311.86218 2778.334]

[457.8119 3416.4106]

[332.71445 3171.7598]

[280.78967 2337.4602]

[244.86034 2154.2788]

Implemented GMM | Covariances

[[1251.75820144 0.]

[0. 27147.43200861]]

[[3682.97336474 0.]

[0. 7065.94810501]]

[[7474.67542332 0.]

[0. 16557.02366305]]

[[1251.75820144 0.]

[0. 27147.43200861]]

[[1218.40647337 0.]

[0. 6988.64202995]]

[[282.97668573 0.]

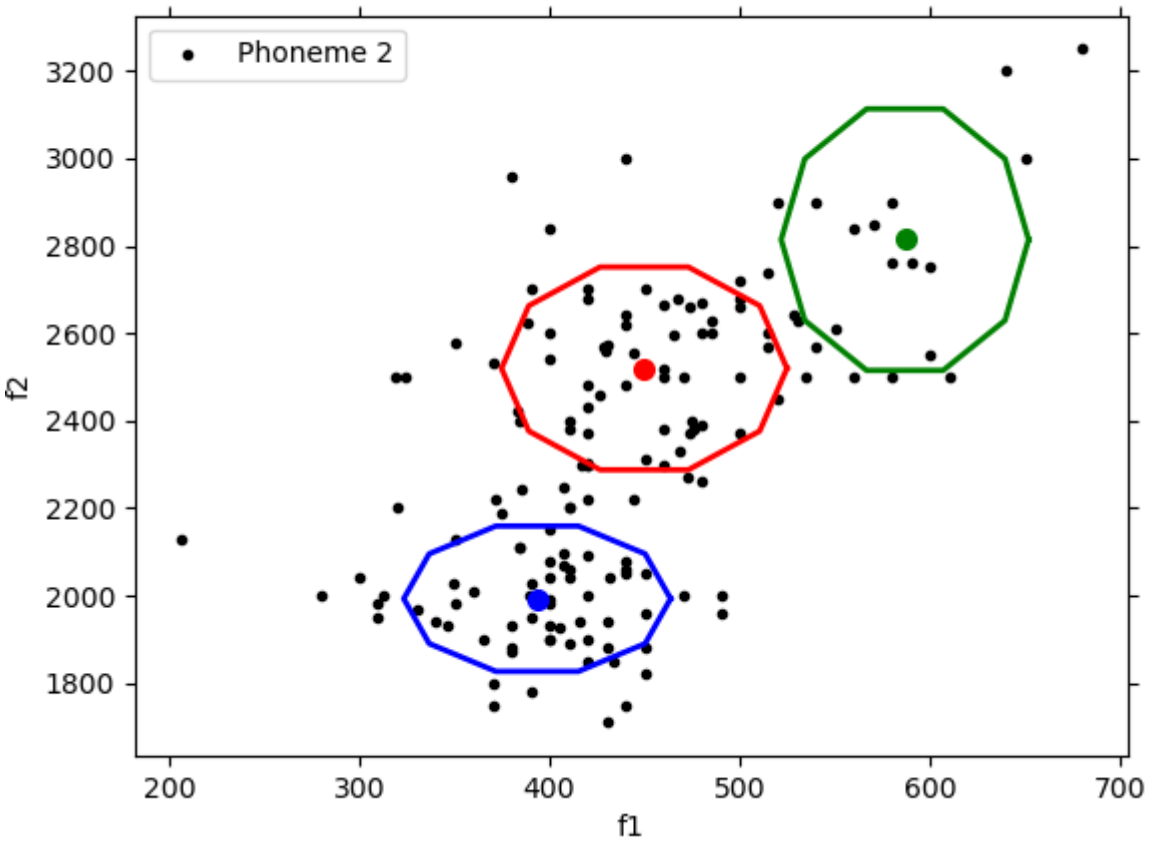
[0. 7242.63785257]]

Implemented GMM | Weights

[0.08606835 0.36773948 0.02607593 0.08606835 0.30859929 0.1254486]

Phoneme 2

First Run (K=3)



Implemented GMM | Mean values

[449.6744 2519.6824]

[586.55994 2814.1675]

[393.45273 1993.0809]

Implemented GMM | Covariances

[[2809.21535176 0.]

[0. 29720.67765194]]

[[2111.62543433 0.]

[0. 49424.42548053]]

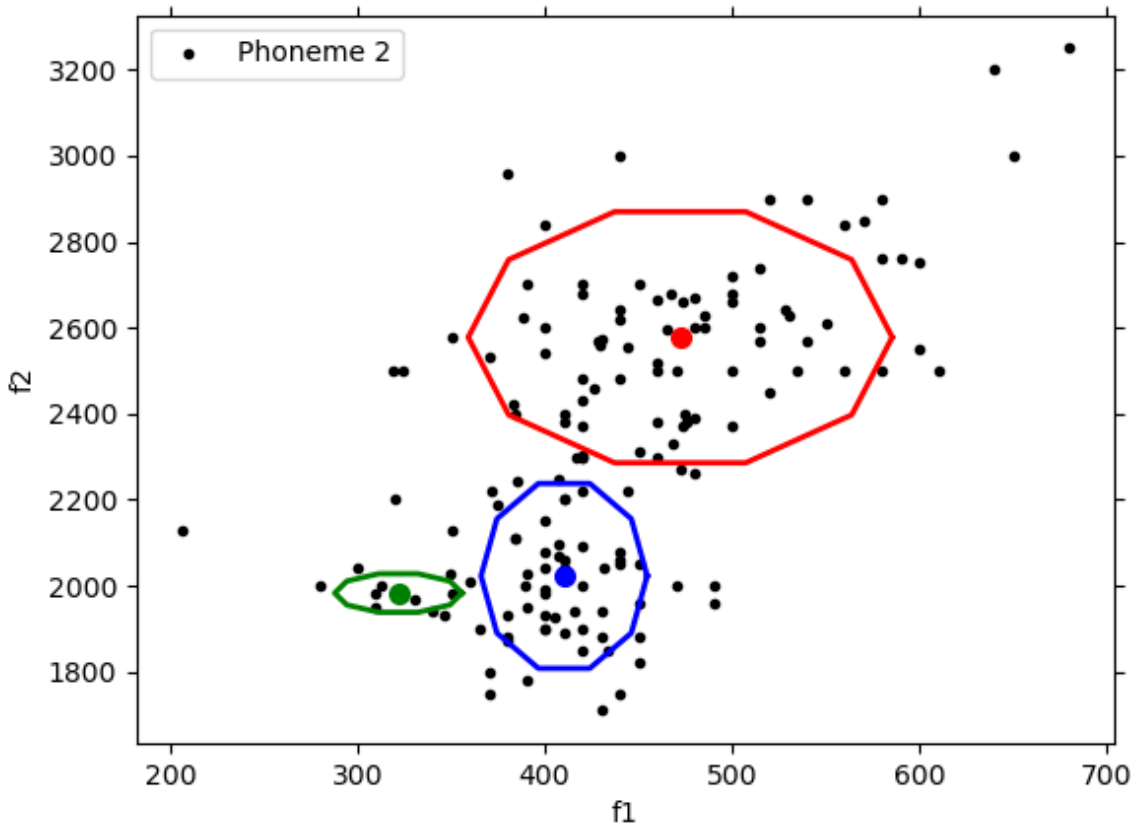
[[2454.91752678 0.]

[0. 15263.39779685]]

Implemented GMM | Weights

[0.46994901 0.09488679 0.4351642]

Second Run (k=3)



Implemented GMM | Mean values

[472.19562 2577.9177]

[321.85602 1983.2834]

[410.2541 2023.0538]

Implemented GMM | Covariances

[[6396.65171545 0.]

 [0. 47113.86597221]]

[[577.3285337 0.]

 [0. 1108.73626161]]

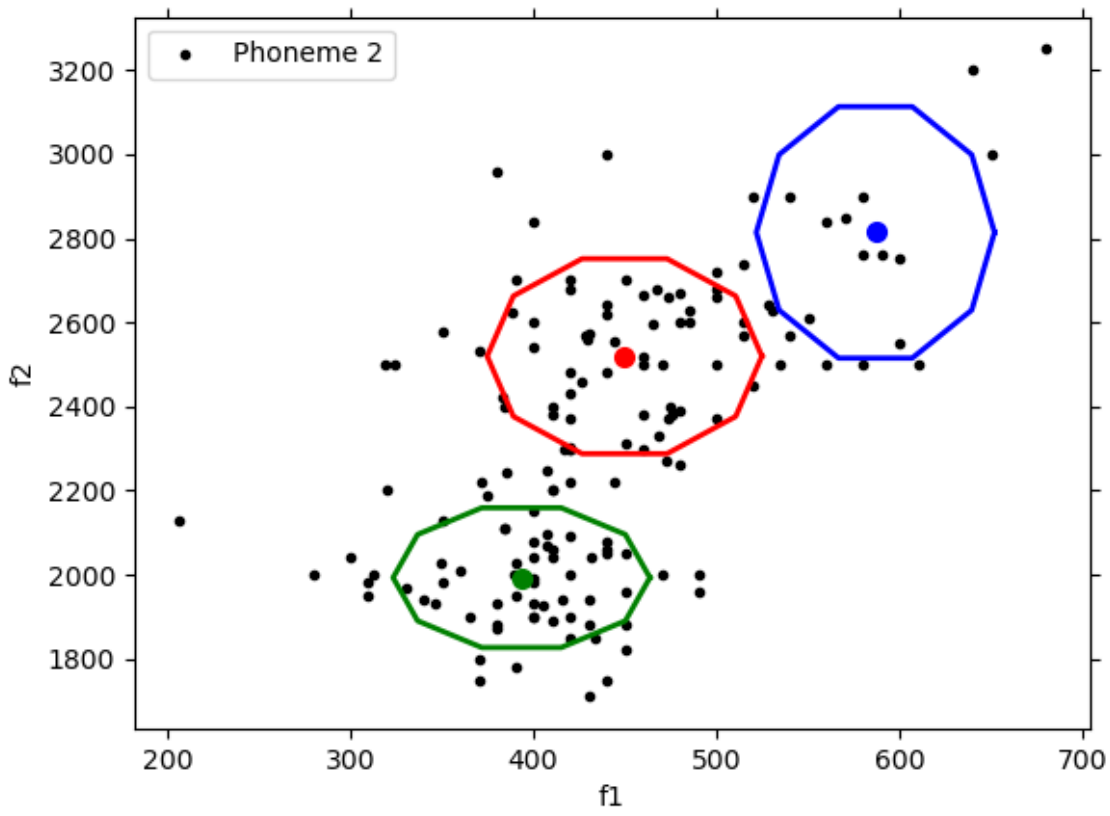
[[981.49617843 0.]

 [0. 25535.19724216]]

Implemented GMM | Weights

[0.53670486 0.05996791 0.40332722]

Third Run (k=3)



Implemented GMM | Mean values

[449.67908 2519.6936]

[393.4532 1993.0856]

[586.56946 2814.204]

Implemented GMM | Covariances

[[2809.54481645 0.]

[0. 29719.62019399]]

[[2454.89378408 0.]

[0. 15264.27547495]]

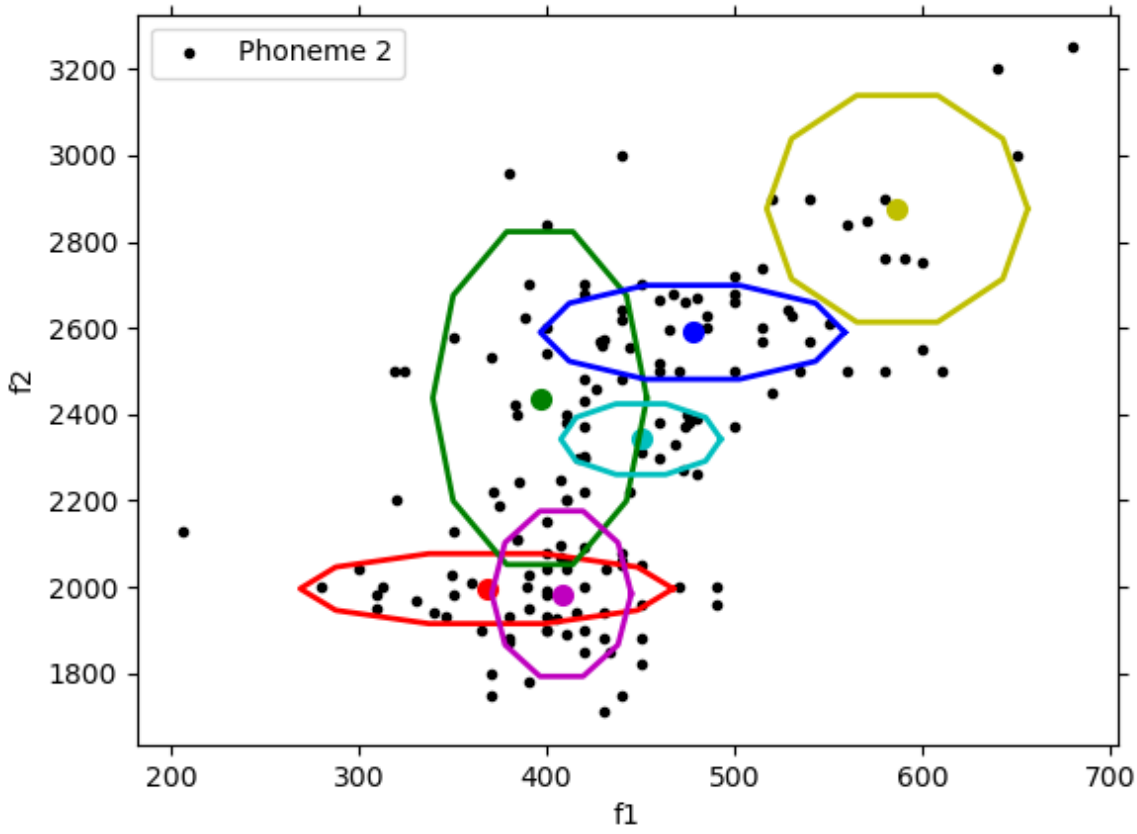
[[2111.34948199 0.]

[0. 49424.3941955]]

Implemented GMM | Weights

[0.46996052 0.43517307 0.09486641]

First Run (K=6)



Implemented GMM | Mean values

[368.04147 1996.1503]

[396.34482 2437.572]

[477.49707 2590.2095]

[450.0691 2342.4448]

[407.87292 1984.4917]

[586.36224 2876.2893]

Implemented GMM | Covariances

[[4913.90784654 0.]

[0. 3631.93760091]]

[[1620.23876669 0.]

[0. 82183.8653292]]

[[3275.78355302 0.]

[0. 6543.4813716]]

[[905.81578668 0.]

[0. 3708.78411331]]

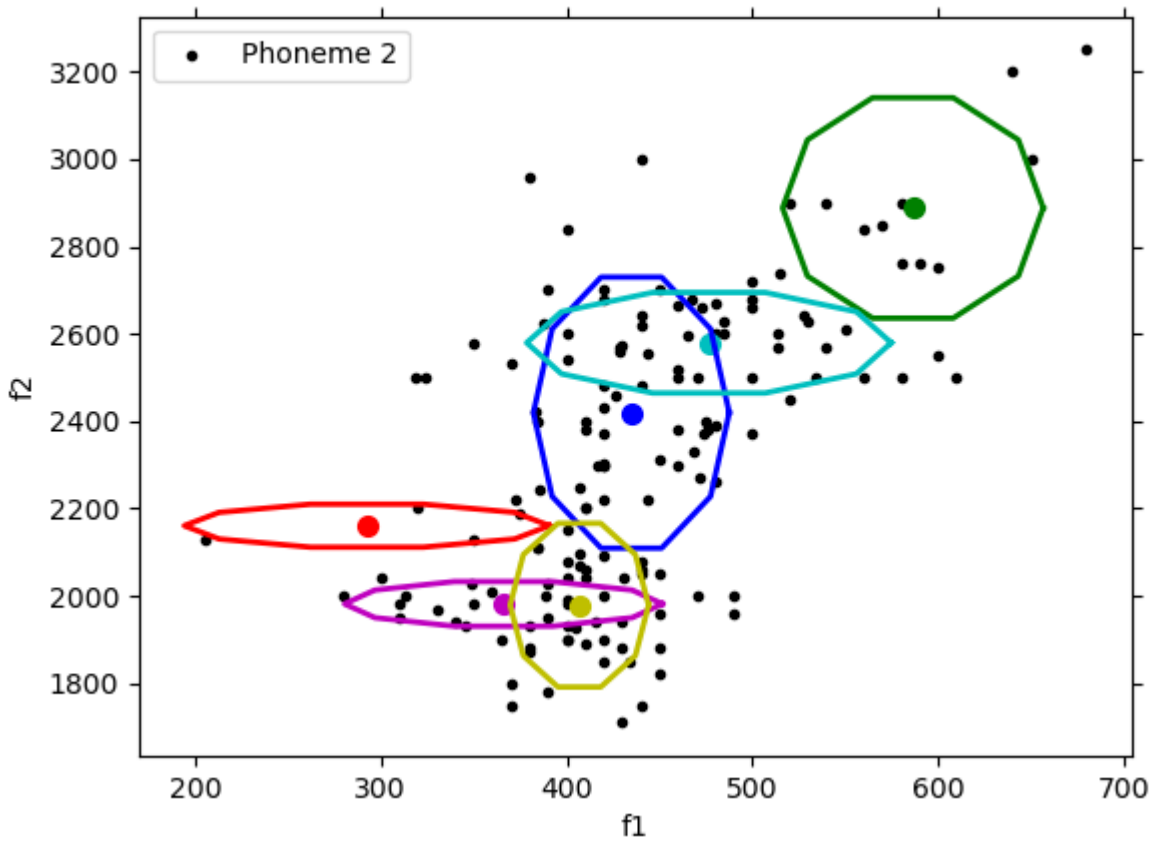
[[689.30401167 0.]

[0. 20284.07826212]]

[[2409.57390077 0.]

[0. 38118.4717854]]

Second Run (K=6)



Implemented GMM | Mean values

[292.5281 2160.8699]

[586.40515 2888.2856]

[434.86816 2419.6685]

[476.62473 2579.6494]

[366.04477 1981.877]

[406.88116 1978.9423]

Implemented GMM | Covariances

[[4830.37693909 0.]

[0. 1339.4581564]]

[[2460.49080092 0.]

[0. 35195.84540257]]

[[1388.76532002 0.]

[0. 53189.35506248]]

[[4811.20255947 0.   ]

 [0.   7324.40309705]]

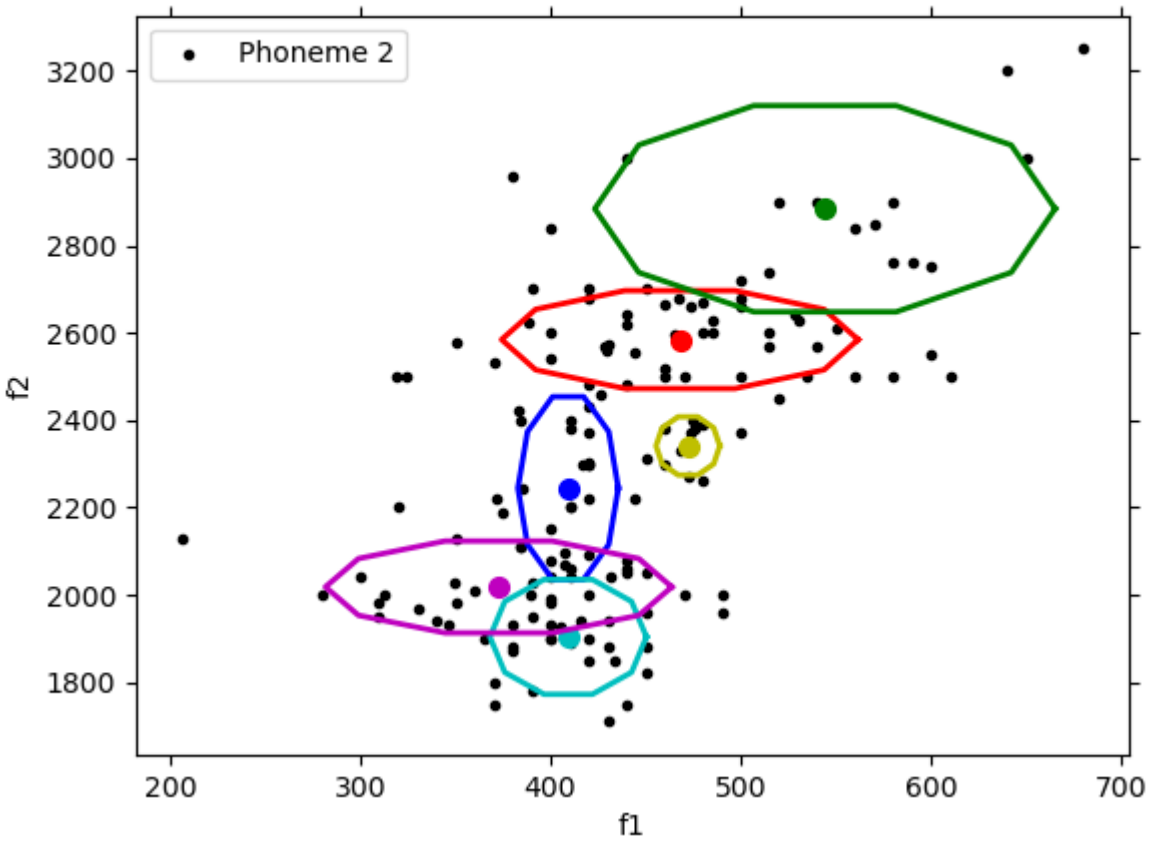
[[3660.01679839 0.   ]

 [0.   1443.92947921]]

[[694.39904103 0.   ]

 [0.   19402.30922563]]

Third Run (K=6)



Implemented GMM | Mean values

[467.8516 2584.6438]

[544.064 2884.1245]

[409.17392 2244.709]

[409.2006 1904.0365]

[372.67194 2018.3407]

[472.10065 2341.2456]

Implemented GMM | Covariances

[[4382.32282227 0.]

[0. 6943.2347047]]

[[7307.3748548 0.]

[0. 30798.54839362]]

[[347.64802414 0.]

[0. 24189.9196599]]

[[845.4895841 0.]

[0. 9542.26971978]]

[[4136.97986138 0.]

[0. 6113.6144999]]

[[140.25644954 0.]

[0. 2453.84168937]]

Observations

In both phenome 1 and 2, upon multiple runs on both k=3 and k=6, the position of the clusters seems to change. With K=6, the some of the clusters overlap each other. Some clusters in k=6, contain very few phonemes and mostly are empty space.

Task 3

Code

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

In the image above, the code is setting the frequency values in X_full.

```
X_phonemes_1_2 = X_full[np.logical_or(phoneme_id ==1, phoneme_id ==2), :]
```

In the image above, the code fills the variable X_phonemes_1_2 with sample of the chosen phonemes.

```
# Phoneme 1
p1_model = "data/GMM_params_phoneme_{:02}_k_{:02}.npy".format(1, k)
param_p1 = np.load(p1_model, allow_pickle=True).item()
copy = X_phonemes_1_2.copy()
Z_p1 = get_predictions(param_p1['mu'], param_p1['s'], param_p1['p'], copy)

p1_pred = Z_p1.sum(axis=1)

# phenome 2
p2_model = "data/GMM_params_phoneme_{:02}_k_{:02}.npy".format(2, k)
param_p2 = np.load(p2_model, allow_pickle=True).item()
copy = X_phonemes_1_2.copy()
Z_p2 = get_predictions(param_p2['mu'], param_p2['s'], param_p2['p'], copy)

p2_pred = Z_p2.sum(axis=1)
```

The code in the image above loads the phoneme model into variables as a dictionary. The code also gets the predictions for the phonemes. The predictions are summed and stored in variables p1_pred and p2_pred.

```
# Calculate Accuracy

preds = np.ones(len(copy)) * 2
preds[p1_pred >= p2_pred] = 1
labels = phoneme_id[np.logical_or(phoneme_id ==1, phoneme_id ==2)]
accuracy = np.sum(preds == labels / copy.shape[0] * 100)
```

The code in the image above finds the accuracy of our models.

Accuracy

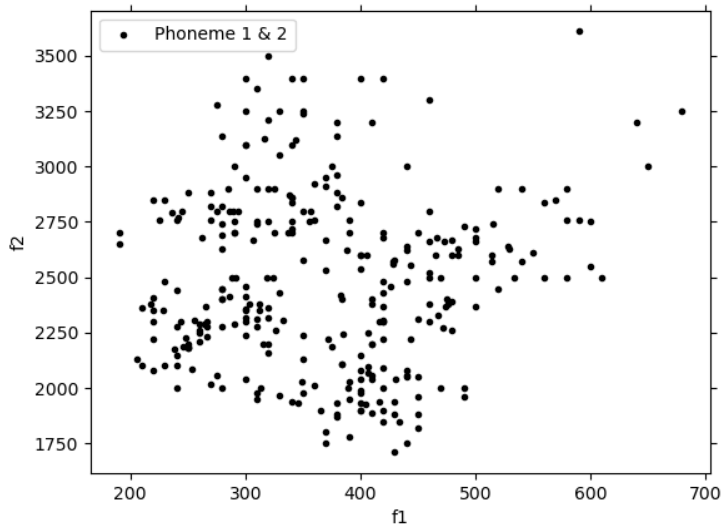


Figure 3 - GNN (K=3)

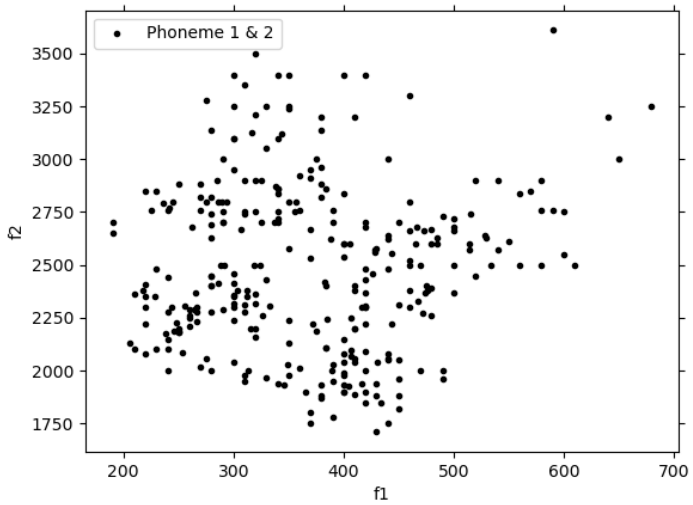


Figure 4 - GMM (K=6)

Accuracy using GMMs with 3 components: 95.07%

Accuracy using GMMs with 6 components: 95.72%

Observations

K= 6 gives a slightly higher accuracy value but the training was slower while K=3 gives lower accuracy, but training was faster.

Task 4

Code

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

In the image above, the code is setting the frequency values in X_full.

```
X_phonemes_1_2 = X_full[np.logical_or(phoneme_id ==1, phoneme_id ==2), :]
```

In the image above, the code fills the variable X_phonemes_1_2 with sample of the chosen phonemes.

```
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()
```

In the image above, the code creates a grid.

```
# phenome 1

p1_model = "data/GMM_params_phoneme_{:02}_k_{:02}.npy".format(1, k)
param_p1 = np.load(p1_model, allow_pickle=True).item()
param_p1 = np.ndarray.tolist(param_p1)
copy = samples.copy()
Z_p1 = get_predictions(param_p1['mu'], param_p1['s'], param_p1['p'], copy)

p1_pred = Z_p1.sum(axis=1)

# phenome 2
p2_model = "data/GMM_params_phoneme_{:02}_k_{:02}.npy".format(2, k)
param_p2 = np.load(p2_model, allow_pickle=True).item()
param_p2 = np.ndarray.tolist(param_p2)
copy = samples.copy()
Z_p2 = get_predictions(param_p2['mu'], param_p2['s'], param_p2['p'], copy)

p2_pred = Z_p2.sum(axis=1)
```

The code in the image above loads the phoneme model into variables. The code also gets the predictions for the phonemes. The predictions are summed and stored in variables p1_pred and p2_pred.

```
preds = np.ones(len(copy)) * 2
preds[p1_pred >= p2_pred] = 1

M = preds.reshape(N_f2, N_f1)
```

```
ids = phoneme_id[np.isin(phoneme_id, [1,2])]
X1 = X[ids ==1]
X2 = X[ids ==2]
plt.scatter(X1[:, 0] - min_f1, X1[:, 1] - min_f2, marker='.', color='red', label='Phoneme 1')
plt.scatter(X2[:, 0] - min_f1, X1[:, 1] - min_f2, marker='.', color='green', label='Phoneme 2')
```

In the image above, the code creates the M array. The plot data colours are set to red and green to differentiate them and are appropriately labelled.

Classification Matrix (K=3)

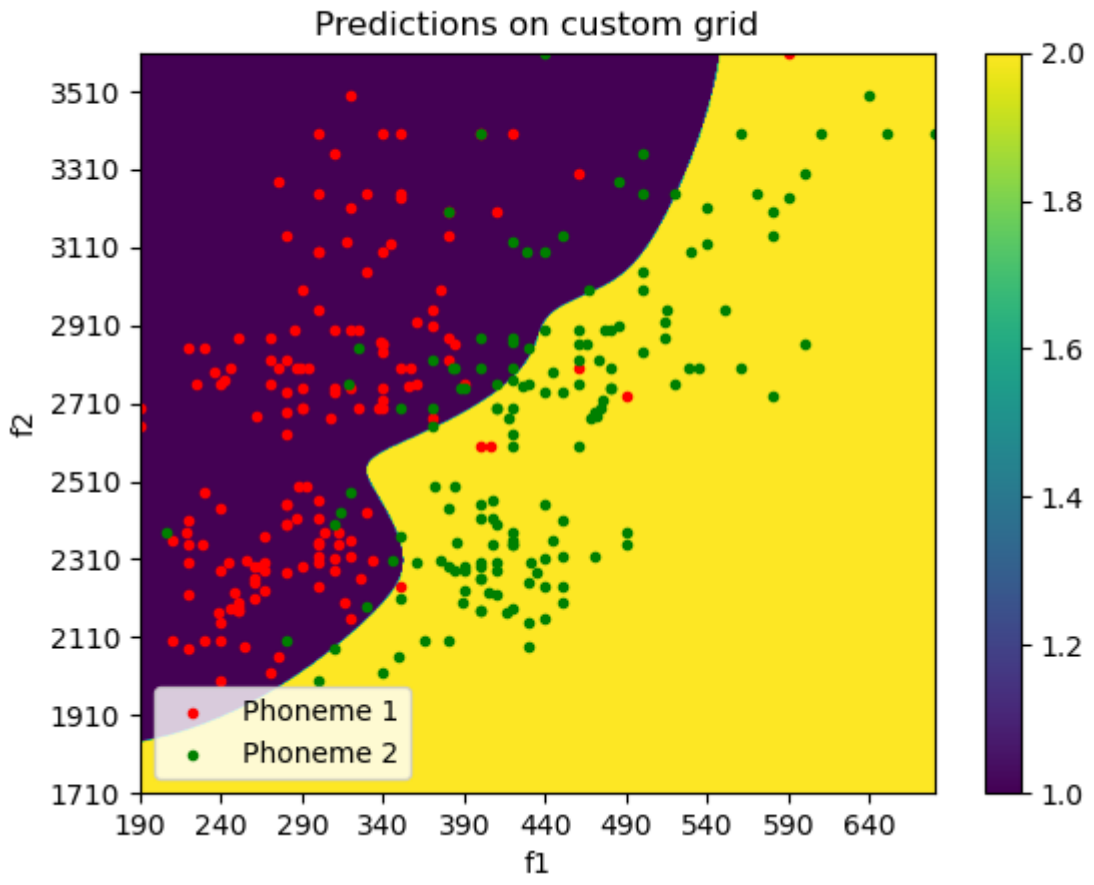


Figure 5 - Classification Matrix (K=3)

f1 range: 190-680 | 490 points

f2 range: 1710-3610 | 1900 points

Classification Matrix (K=6)

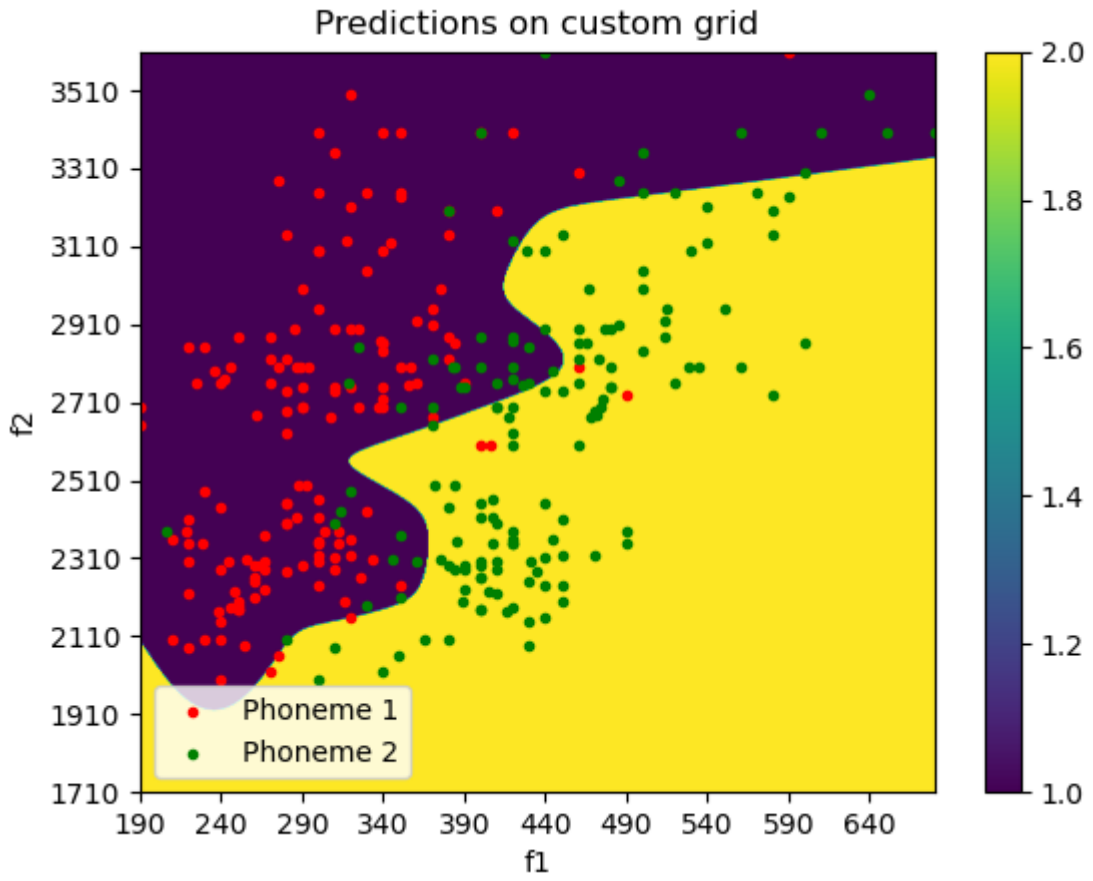


Figure 6 - Classification Matrix (K=6)

f1 range: 190-680 | 490 points

f2 range: 1710-3610 | 1900 points

Observations

With more clusters (K=6), the decision boundary seems to have been better from the middle however the edges seem to have become worse. It is possible that with more clusters that the boundary would be even better at deciding its edges.

Task 5

Code

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

In the image above, the code is setting the frequency values in X_full.

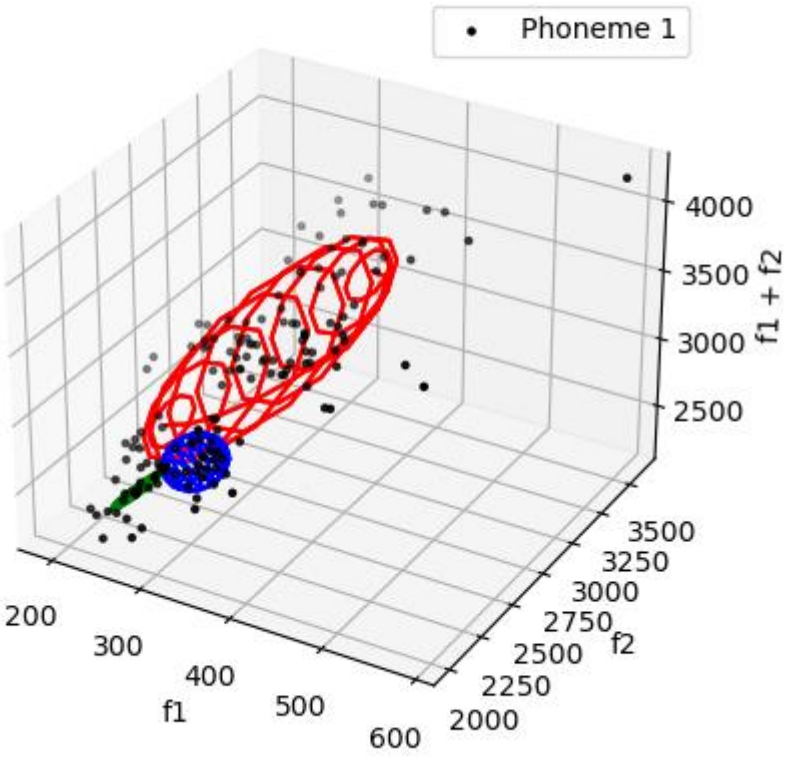
```
X_phoneme = X_full[phoneme_id == p_id, :]
```

In the image above, the code fills the variable X_phoneme with sample of the chosen phonemes.

```
s[i, :, :] += 0.001 * np.identity(D)
```

In the image above, the code offsets the singularity by making its determinant non-zero

Problem



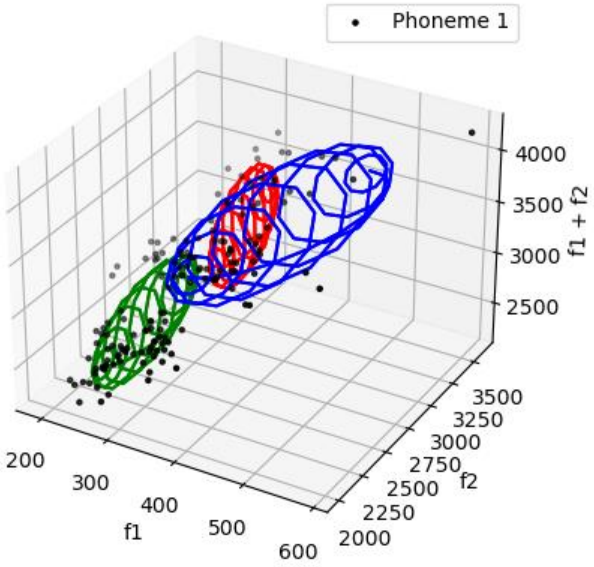
When running the program, the following error is given:

C:\Users\khuzi\anaconda3\lib\site-packages\numpy\core_asarray.py:102: ComplexWarning: Casting complex values to real discards the imaginary part

return array (a, dtype, copy=False, order=order)

Because of singularity problem, the above error is given.

K = 3



Implemented GMM | Mean values

[339.9106 2995.1353 3335.0457]

[271.79526 2446.1313 2717.9268]

[389.04092 3023.7139 3412.755]

Implemented GMM | Covariances

[[1018.29562342 -2919.73972488 -1901.44510146]

 [-2919.73972488 56663.51823295 53743.77750805]

 [-1901.44510146 53743.77750805 51842.33340661]]

[[1299.95673705 1414.03684539 2713.99258245]

 [1414.03684539 66818.74406288 68232.77990827]

 [2713.99258245 68232.77990827 70946.77349073]]

[[7949.5496478   14287.09169386 22236.64034166]

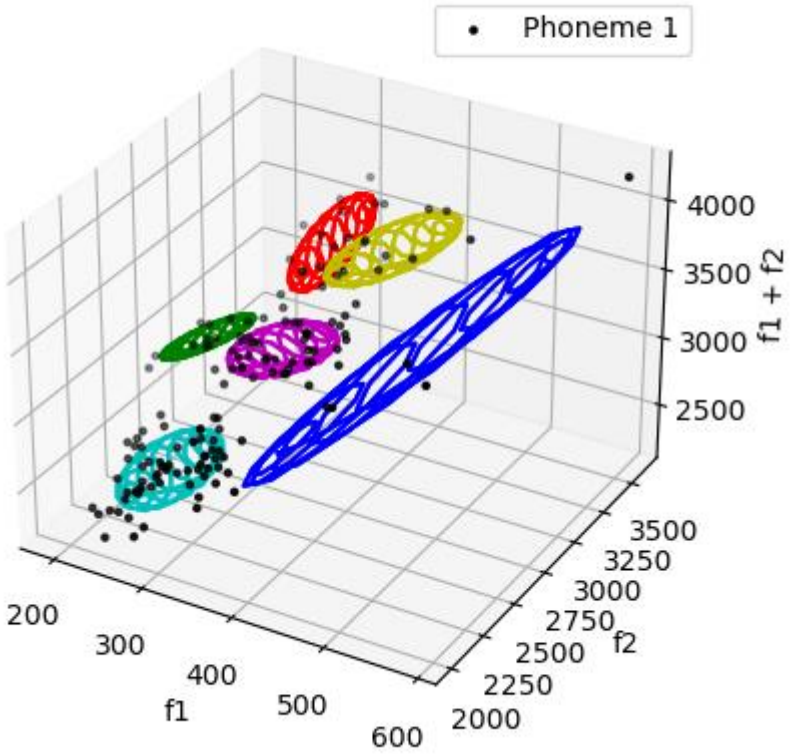
 [14287.09169386 110090.70538741 124377.79608126]

 [22236.64034166 124377.79608126 146614.43742292]]

Implemented GMM | Weights

[0.26731035 0.63665231 0.09603733]

K = 6



Implemented GMM | Mean values

[315.5175 3226.0444 3541.5618]

[235.70126 2799.2407 3034.9421]

[460.9221 2829.604 3290.5261]

[270.13962 2285.8438 2555.9834]

[322.29938 2792.4634 3114.7627]

[383.94305 3227.0059 3610.949]

Implemented GMM | Covariances

[[552.85491716   815.07580115 1367.92971831]

 [815.07580115 22156.17133534 22971.24613648]

 [1367.92971831 22971.24613648 24339.1768548]]

[[826.47541708 1836.29846402 2662.7728811]

 [1836.29846402 6154.83051321 7991.12797723]

[2662.7728811 7991.12797723 10653.90185834]]

[[5673.01636991 28598.04521208 34271.06058199]

[28598.04521208 161751.02813593 190349.07234801]

[34271.06058199 190349.07234801 224620.13392999]]

[[1197.70198714 1168.36284079 2366.06382793]

[1168.36284079 14338.98201665 15507.34385744]

[2366.06382793 15507.34385744 17873.40868538]]

[[1530.25436842 1054.16033602 2584.41370443]

[1054.16033602 7402.33298984 8456.49232586]

[2584.41370443 8456.49232586 11040.9070303]]

[[1789.01780175 3139.54504892 4928.56185067]

[3139.54504892 14606.34331977 17745.88736869]

[4928.56185067 17745.88736869 22674.45021936]]

Implemented GMM | Weights

[0.10955851 0.07144987 0.03458617 0.43292329 0.29176868 0.05971347]

Observations

First it is important to understand what a singular matrix is to solve the problem. In a singular matrix:

- Determinant is zero
- It is non-invertible; hence we cannot find its inverse.

However, the code tries to use inverse which cannot be applied to singular matrix.

The solution to this is to give the matrix the dimensions diagonal and multiply it by 0.001 which is added to the covariance split so that the determinant is not zero which would make the matrix non-inverse. By implementing this method, the error described in the Problem section disappeared.