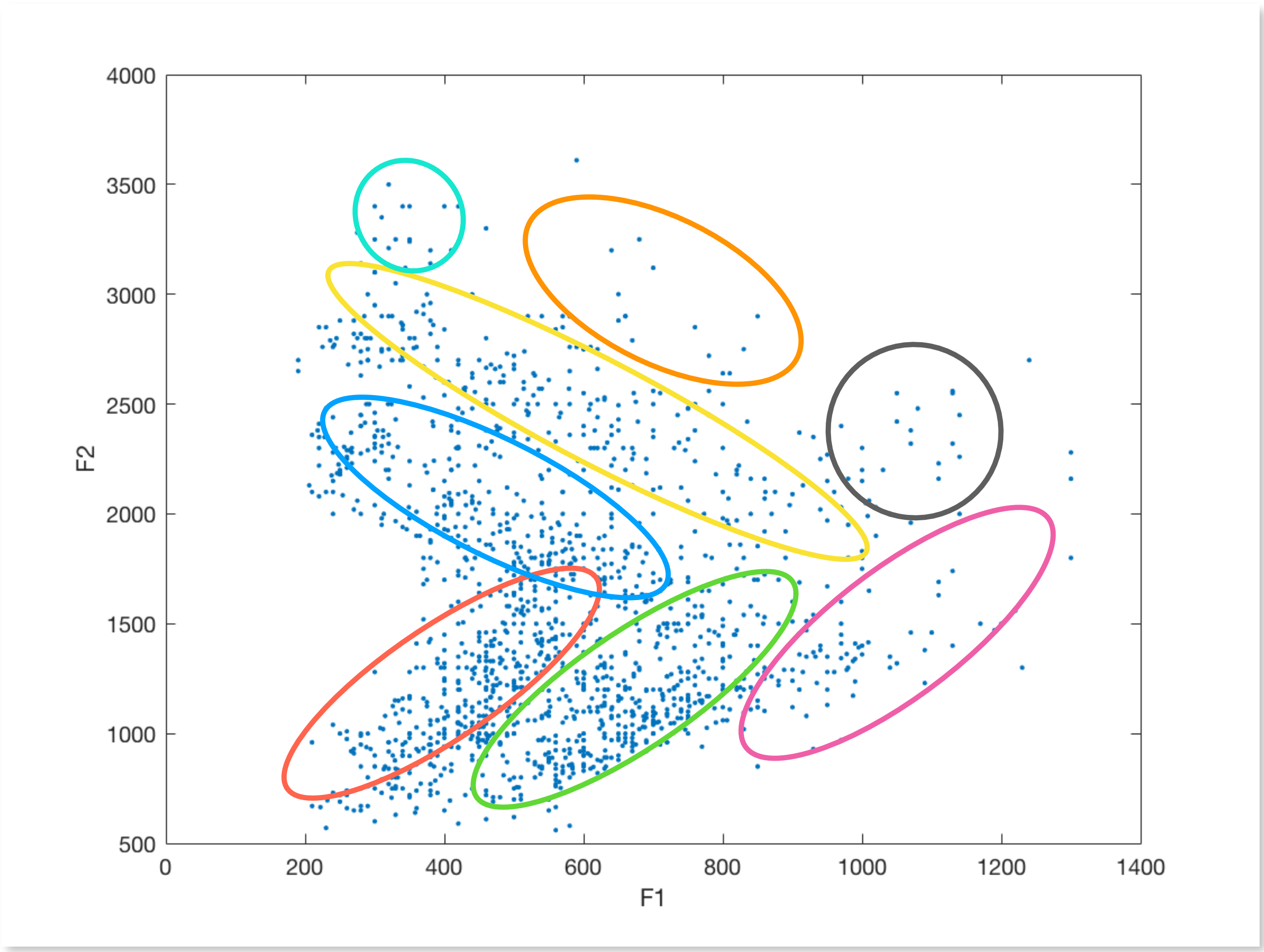
Clustering and MoG

MoG Modelling using the EM Algorithm

Task 1

task1_load_data.m

```
load('PB_data.mat');
J = [f1, f2];
figure; plot(J(:,1), J(:,2), '.'); xlabel('F1'); ylabel('F2');
```



Task 2

Phoneme 1

mog.m

```
load('PB_data.mat');
ph1F1 = f1(phno==1);
ph1F2 = f2(phno==1);
x = [ph1F1, ph1F2];
```

Phoneme 1, K = 3

Mean (mu) [Phoneme 1, K = 3]

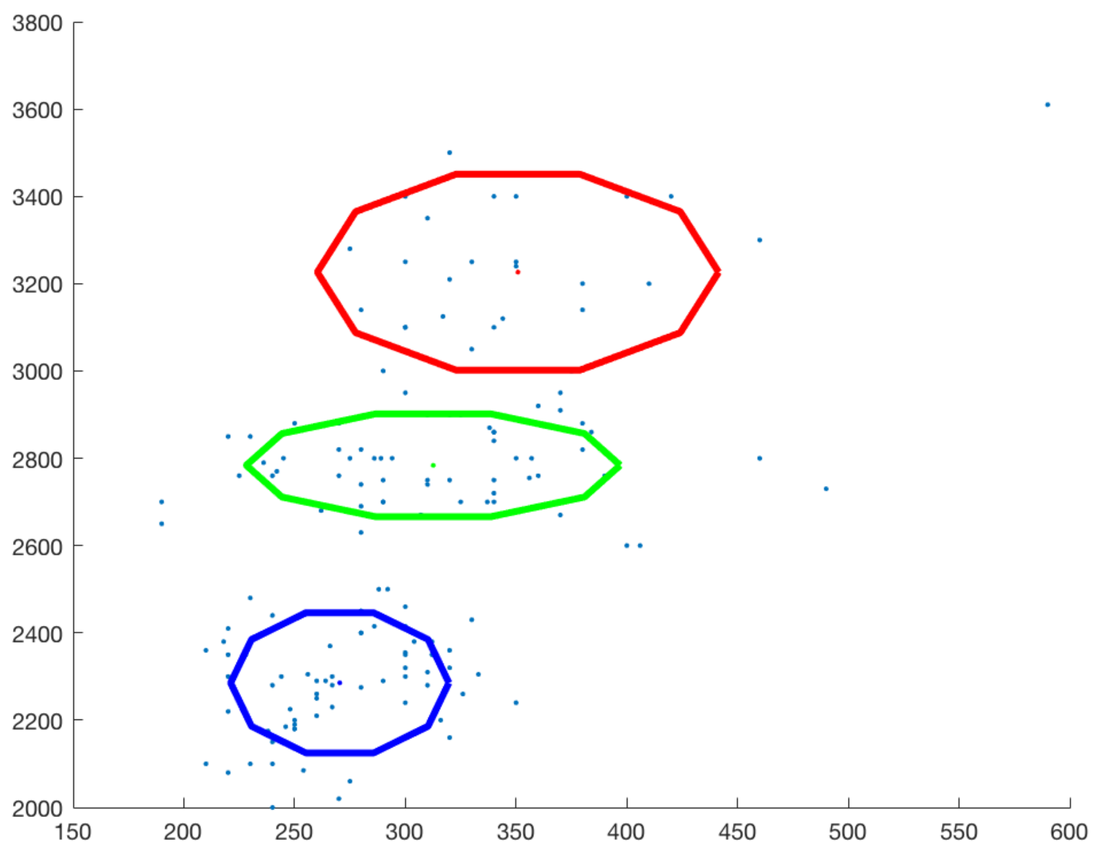
	1	2	3
F1	270.395	350.844	312.591
F2	2285.465	3226.339	2783.897

Covariance Matrices (s2) [Phoneme 1, K = 3]

1		2		3	
1213.738	0	4102.874	0	3562.597	0
0	14278.420	0	27829.557	0	7657.847

Mixing Proportions (p) [Phoneme 1, K = 3]

1	2	3
0.435	0.184	0.381



Phoneme 1, K = 6

Mean (mu) [Phoneme 1, K = 6]

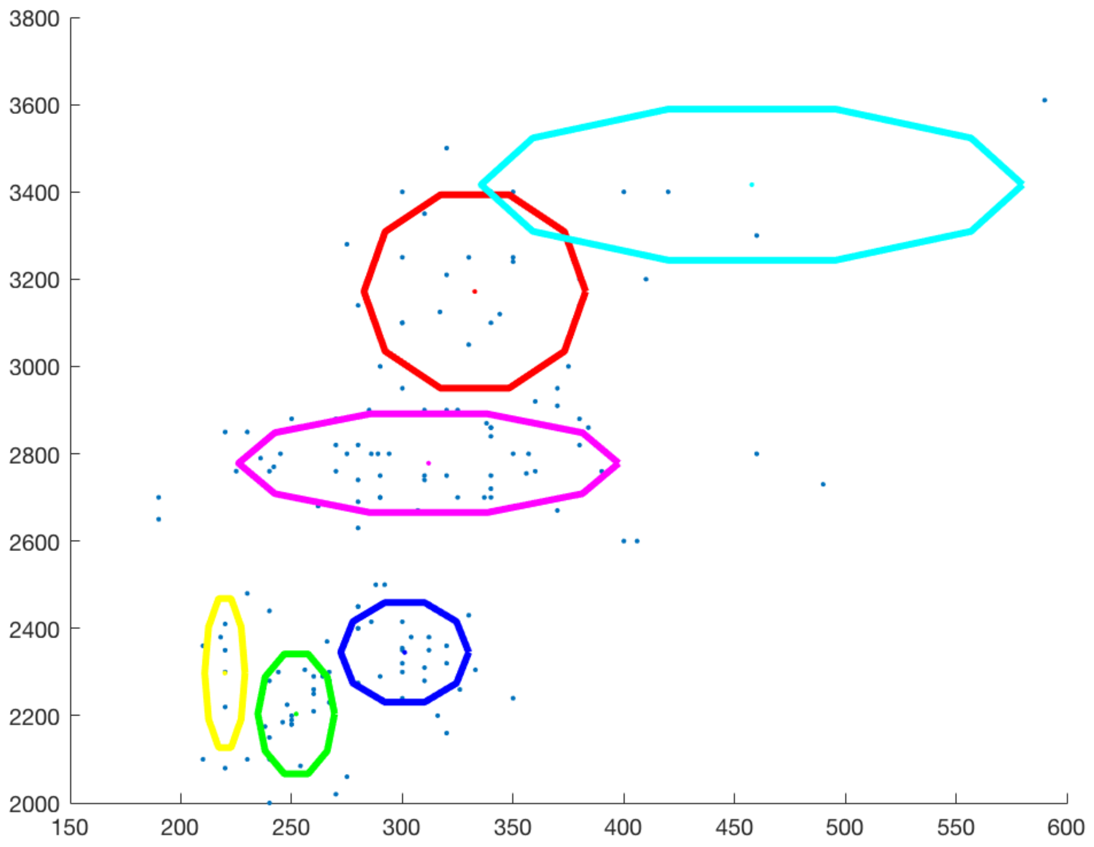
	1	2	3	4	5	6
F1	332.714	252.077	301.072	219.878	311.868	457.816
F2	3171.728	2203.973	2345.005	2297.635	2778.355	3416.416

Covariance Matrices (s2) Phoneme 1, K = 6]

1		2		3		4		5		6	
1251.7	0	150.1	0	417.6	0	41.3	0	3683.8	0	7474.5	0
0	27151.0	0	10443.9	0	7209.5	0	16137.4	0	7057.1	0	16556.7

Mixing Proportions (p) [Phoneme 1, K = 6]

1	2	3	4	5	6
0.1722	0.1642	0.2050	0.0649	0.3676	0.0261



Phoneme 2

mog.m

```
load('PB_data.mat');
ph2F1 = f1(phno==2);
ph2F2 = f2(phno==2);
x = [ph2F1, ph2F2];
```

Phoneme 2, K = 3

Mean (mu) [Phoneme 2, K = 3]

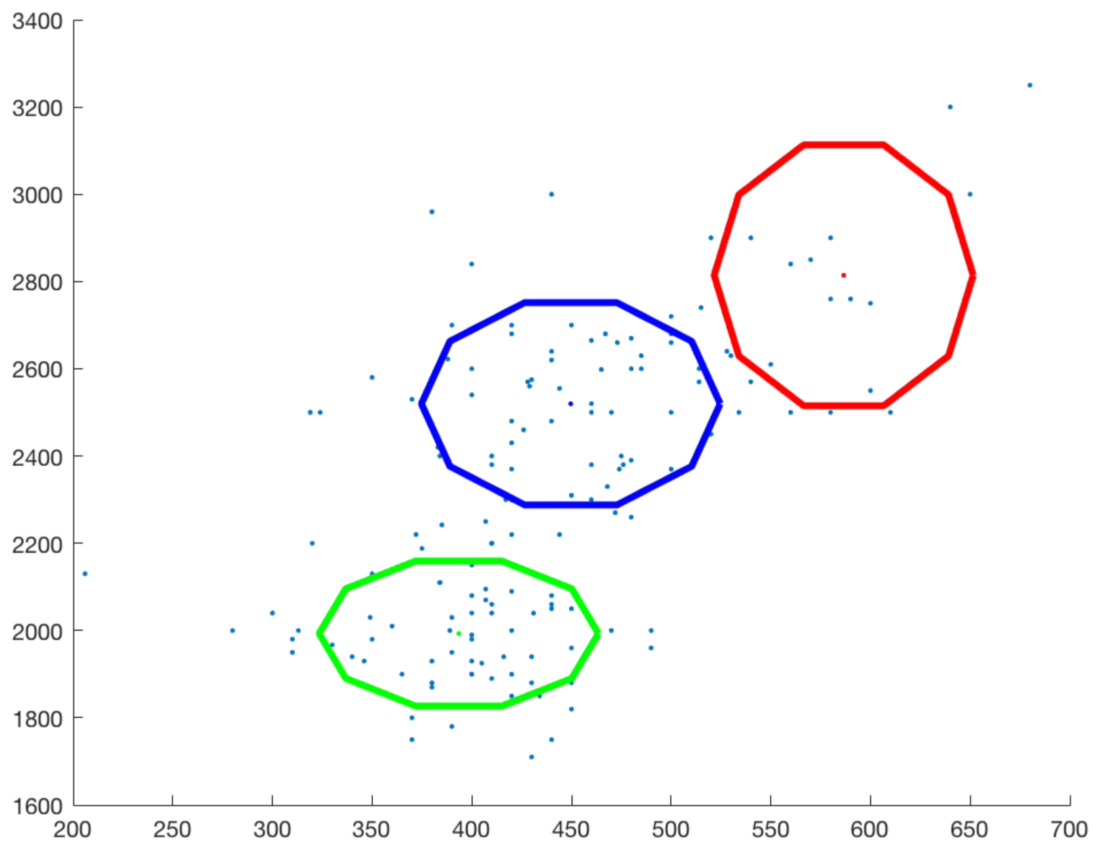
	1	2	3
F1	393.451	586.537	449.663
F2	1993.070	2814.080	2519.656

Covariance Matrices (s2) [Phoneme 2, K = 3]

1		2		3	
2454.973	0	2112.282	0	2808.432	0
0	15261.312	0	49424.481	0	29723.192

Mixing Proportions (p) [Phoneme 2, K = 3]

1	2	3
0.4351	0.0949	0.4699



Phoneme 2, K = 6

Mean (mu) [Phoneme 2, K = 6]

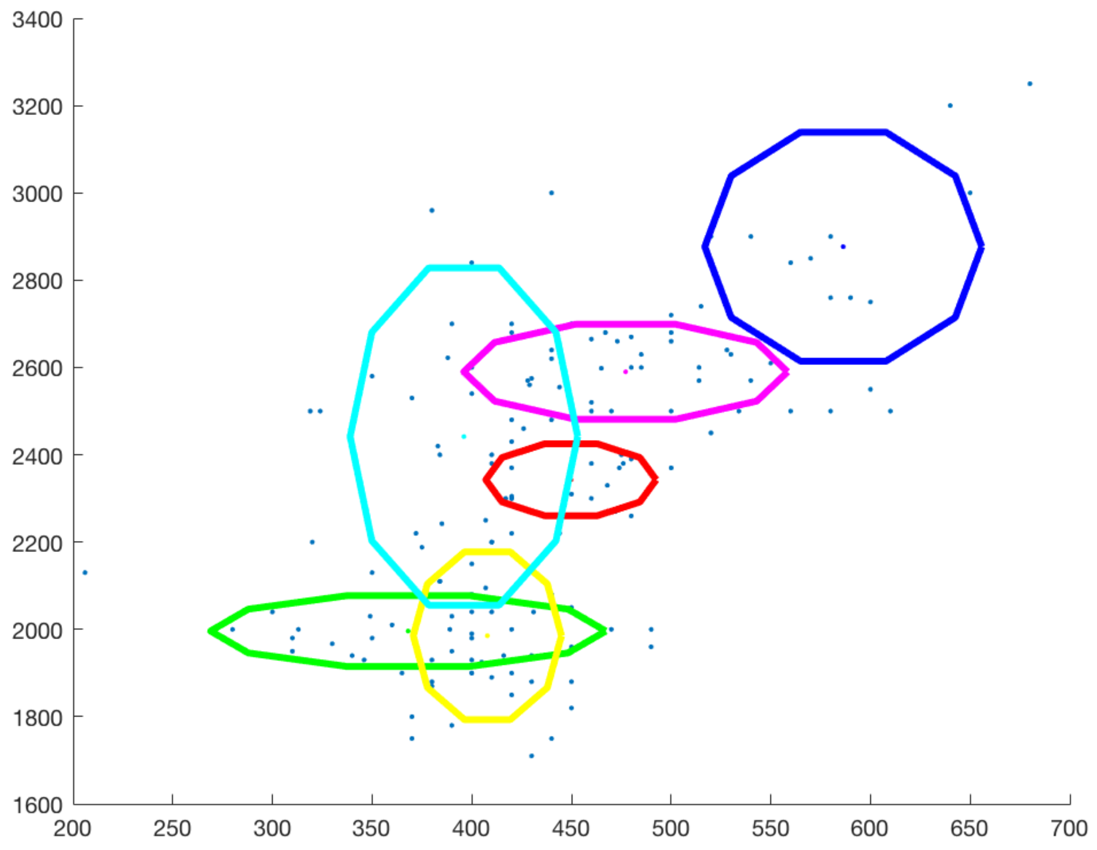
	1	2	3	4	5	6
F1	586.368	477.592	368.032	450.175	396.444	407.890
F2	2876.200	2590.233	1996.155	2342.355	2436.268	1984.226

Covariance Matrices (s2) Phoneme 2, K = 6]

1		2		3		4		5		6	
2408.5	0	3268.8	0	4913.6	0	903.3	0	1617.1	0	688.7	0
0	38140.2	0	6542.5	0	3629.9	0	3693.1	0	82063.2	0	20242.7

Mixing Proportions (p) [Phoneme 2, K = 6]

1	2	3	4	5	6
0.0816	0.2591	0.1374	0.1118	0.1385	0.2713



Task 3

task3.m
```
% load the dataset
load('PB_data.mat');

% get phoneme 1, 2 data from the first and second formant frequencies
ph1F1 = f1(phno==1); ph1F2 = f2(phno==1);
ph2F1 = f1(phno==2); ph2F2 = f2(phno==2);
ph1Data = [ph1F1, ph1F2]; ph2Data = [ph2F1, ph2F2];

% load the arguments (p, mu, s2) of MoG model built in task2
% ph1Mu, ph1S2, ph1P for phoneme 1 and ph2Mu, ph2S2, ph2P for phoneme 2
K = 3; % K = 6
load('ph1K3M.mat'); % or load('ph1K6M.mat');
load('ph2K3M.mat'); % or load('ph2K6M.mat');

% set the test data and information
testData = [ph1Data; ph2Data];
[n, D] = size(testData);

% use a classifier to discriminate test data between phonemes 1 and 2
[ph1Likelihood, ph2Likelihood, phClass] = classifier(testData, K, ph1Mu, ph1S2, ph1P, ph2Mu,
ph2S2, ph2P);
% distinguish the predicted results into two phonemes for showing
ph1Vector = testData(((ph2Likelihood > ph1Likelihood)+1)==1, :);
ph2Vector = testData(((ph2Likelihood > ph1Likelihood)+1)==2, :);

% label the test data with the phonemes
labels = ones(length(testData), 1);
labels(length(ph1Data)+1:end, 1) = 2;

% calculate the confusion matrix and get the miss classification error
[confM, missErr] = confusionMatrix(labels, ph1Likelihood, ph2Likelihood);

% show the discriminated results
figure; hold on;
plot(ph1Vector(:,1), ph1Vector(:,2), '.b');
plot(ph2Vector(:,1), ph2Vector(:,2), '.r');
xlabel('F1'); ylabel('F2'); hold off;
```

MoGs

$$p(x) = \sum_{k=1}^{K} \frac{p(c_k)}{(2\pi)^{\frac{D}{2}} det(\Sigma_k)^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k))$$

classifier.m
```
function [ph1Likelihood, ph2Likelihood, phClass] = classifier(testData, K, ph1Mu, ph1S2, ph1P,
ph2Mu, ph2S2, ph2P)
    % initialise likelihood vector for the two MoG models
    [n, D] = size(testData);
    ph1Likelihood = zeros(n, K);
    ph2Likelihood = zeros(n, K);
    % use the hypothesis to calculate the probability between two MoG models
    for ph1K = 1:K
        ph1Likelihood(:,ph1K) = ph1P(ph1K) * (2*pi)^(-D/2) * det(ph1S2(:,:,K))^(-1/2) * ...
            exp(-1/2*sum((testData'-repmat(ph1Mu(:,ph1K),1,n))'*inv(ph1S2(:,:,ph1K)).*...
            (testData'-repmat(ph1Mu(:,ph1K),1,n))', 2));
    end
    for ph2K = 1:K
        ph2Likelihood(:,ph2K) = ph2P(ph2K) * (2*pi)^(-D/2) * det(ph2S2(:,:,K))^(-1/2) * ...
            exp(-1/2*sum((testData'-repmat(ph2Mu(:,ph2K),1,n))'*inv(ph2S2(:,:,ph2K)).*...
            (testData'-repmat(ph2Mu(:,ph2K),1,n))', 2));
    end

    % sum the probability from each Gaussian model to express a MoGs
    ph1Likelihood = sum(ph1Likelihood, 2);
    ph2Likelihood = sum(ph2Likelihood, 2);

    % classify phoneme between 1 and 2
    phClass = (sum(ph2Likelihood) > sum(ph1Likelihood))+1;
end
```

confusionMatrix.m

```
function [confM, missErr] = confusionMatrix(labels, ph1Likelihood, ph2Likelihood)
    prep = [ph1Likelihood, ph2Likelihood]; % col1, col2: likelihood of phoneme 1 and phoneme 2
    prep(:,3)=(ph2Likelihood > ph1Likelihood)+1; % col3: the predicted class
    prep(:,4) = labels; % col4: actual class

    % calculate a confusion matrix by comparing predicted and actual classes
    confM = zeros(2,2);
    confM(1,1) = sum(sum((prep(:,3:4)==[1,1]), 2)==2);
    confM(1,2) = sum(sum((prep(:,3:4)==[1,2]), 2)==2);
    confM(2,1) = sum(sum((prep(:,3:4)==[2,1]), 2)==2);
    confM(2,2) = sum(sum((prep(:,3:4)==[2,2]), 2)==2);

    % calculate the miss-classification error by the confusion matrix
    missErr = sum(sum(confM - confM.*eye(2)))/sum(confM(1:end));
end
```

Confusion Matrix

		Actual Class	
		Phoneme 1	**Phoneme 2**
Predicted Class	**Phoneme 1**	True Positive	False Positive
	Phoneme 2	False Negative	True Negative

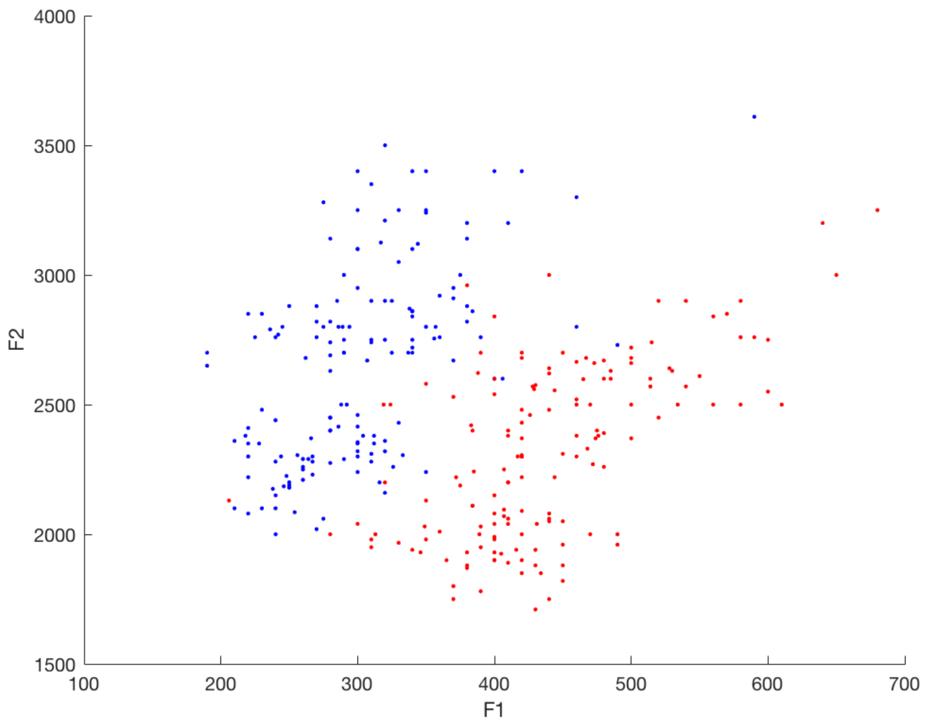
The miss-classification error is the number of test data where the classifier has made an error divided by the total number of test data.
So that calculating the miss-classification error by confusion matrix below:

$$\frac{FP + FN}{N}$$

Where FP is the number of False Positive, FN is the number of False Negative and N is the total number of examples. Therefore, the miss-classification error is the sum of the off-diagonal elements in the confusion matrix.

Actual Class

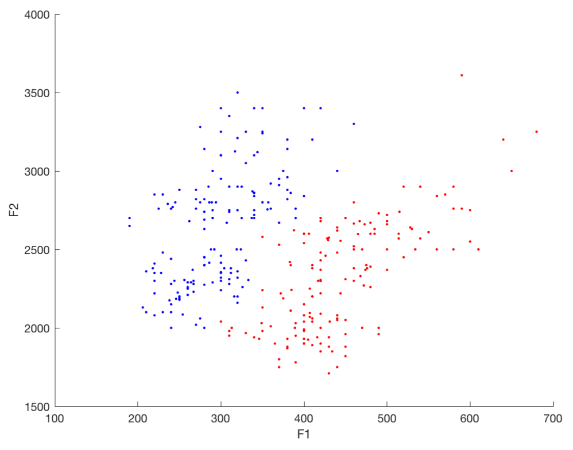
Blue: Phoneme 1 Red: Phoneme 2

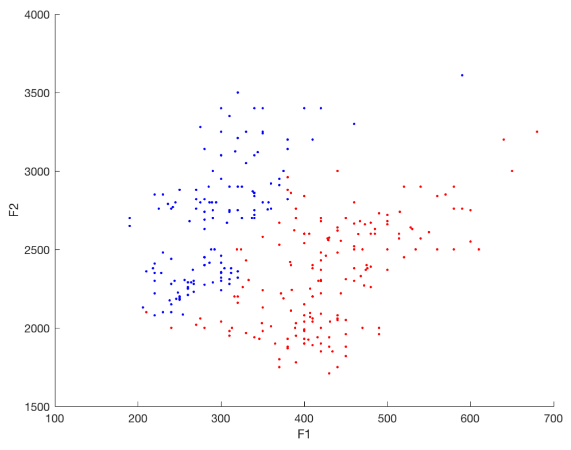


Predicted Class

	K = 3	K = 6
Testing data	all F1,F2 of phoneme 1and phoneme 2	
Miss classification error	0.0493	0.0625

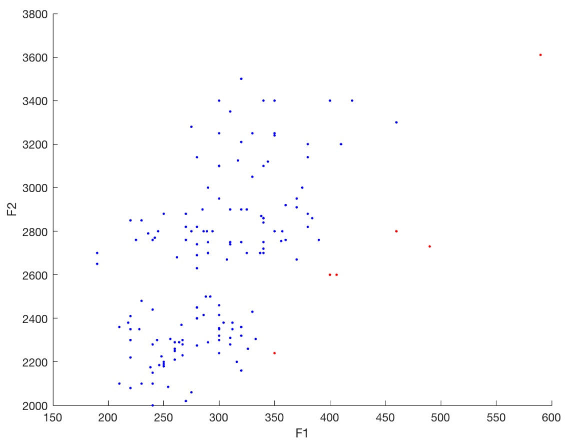
Blue: Phoneme 1 Red: Phoneme 2

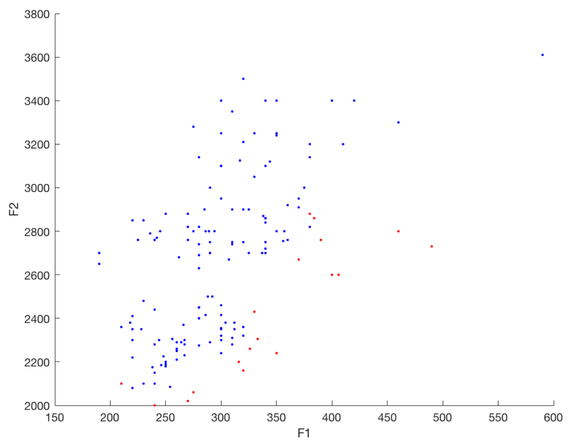




	K = 3	K = 6
Testing data	all F1,F2 of phoneme 1	
Miss classification error	0.0395	0.1184

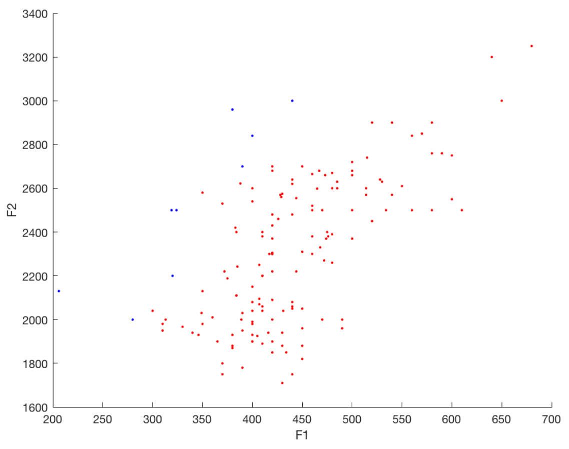
Blue: Phoneme 1 Red: Phoneme 2

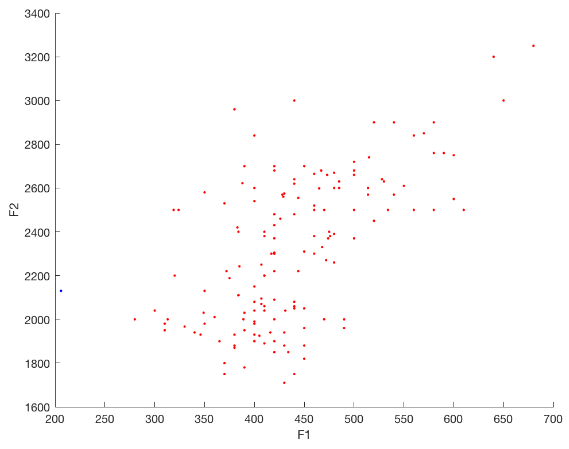




	K = 3	K = 6
Testing data	all F1,F2 of phoneme 2	
Miss classification error	0.0592	0.0066

Blue: Phoneme 1 Red: Phoneme 2





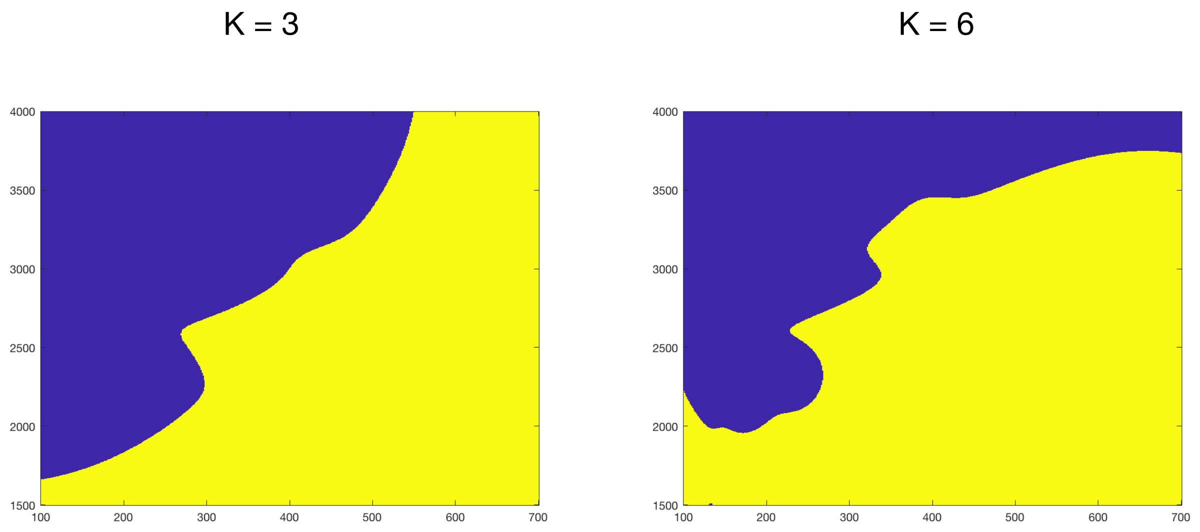
Intuitively, the number of MoGs components is higher, the miss-classification error is lower, because the MoGs model can fit more when K is bigger. However, I get the similar error numbers (0.0493 and 0.0625) when I adjust K from 3 to 6 for all phonemes, but it is an significant difference when data is only for one phoneme like 1 or 2, which are 0.0395 against 0.1184 for phoneme 1, and 0.0592 and 0.0066 for phoneme 2 respectively.

Task 4

task4.m

```
% load the dataset
load('PB_data.mat');

% get phoneme 1, 2 data from the first and second formant frequencies
ph1F1 = f1(phno==1); ph1F2 = f2(phno==1);
ph2F1 = f1(phno==2); ph2F2 = f2(phno==2);

% load the arguments (p, mu, s2) of MoG model built in task2
% ph1Mu, ph1S2, ph1P for phoneme 1 and ph2Mu, ph2S2, ph2P for phoneme 2
K = 3; % or K = 6
load('ph1K3M.mat'); % or load('ph1K6M.mat');
load('ph2K3M.mat'); % or load('ph2K6M.mat');

% find the minimum and the maximum value of F1 and F2 for the first two phonemes
minF1 = min([ph1F1; ph2F1]); maxF1 = max([ph1F1; ph2F1]);
minF2 = min([ph1F2; ph2F2]); maxF2 = max([ph1F2; ph2F2]);
M = zeros(length(minF2:maxF2), length(minF1:maxF1));

% loop for filling classification matrix
for i = minF2:maxF2
   for j = minF1:maxF1
      [ph1Likelihood, ph2Likelihood, phClass] = classifier([j, i], K, ph1Mu, ph1S2, ph1P, ph2Mu,
ph2S2, ph2P);
       M(i-minF2+1, j-minF1+1) = phClass;
   end
end

% show the results
figure; imagesc([100 700], [1500 4000], M); set(gca, 'YDir', 'normal');
```

Classification Matrix



According to the images of classification matrix, it is obviously that I get an more adapted shape when K = 6. Further, it is clear that decision boundary on the classification matrix.

Task 5

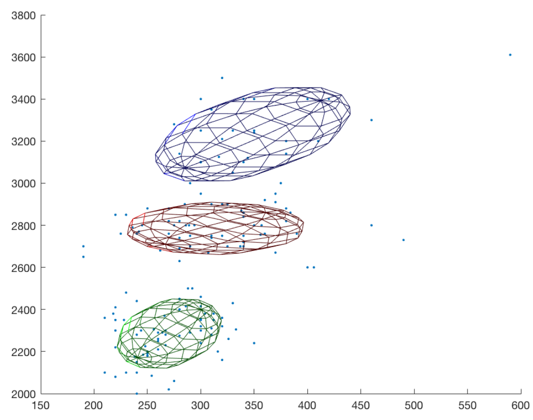
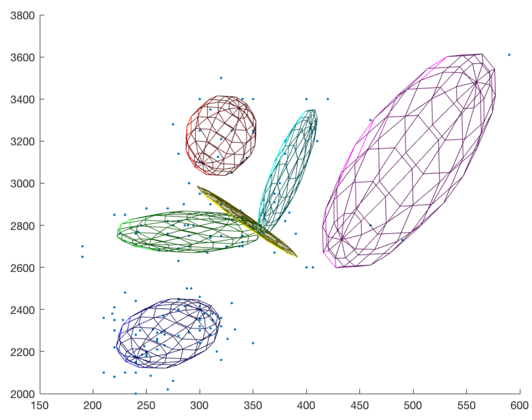
The aim of this task is to train a MoG model with a full covariance matrices from a new dataset in which each data is 3 dimensional vector. In the process of implementing, I face an error called "Numerical Error in Loop - Possibly Singular Matrix" which is companied a waring "Matrix is close to singular or badly scaled. Results may be inaccurate.". Therefore, I think the problem is that the data dimensionality is higher than the data samples causing the covariance matrix is singular, non invertible and zero determinant. Additionally, there are two way to solve this problem below:

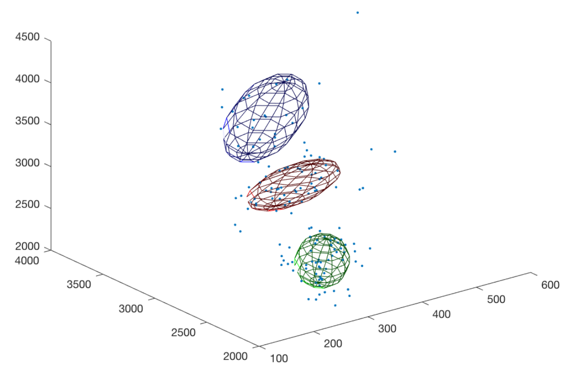
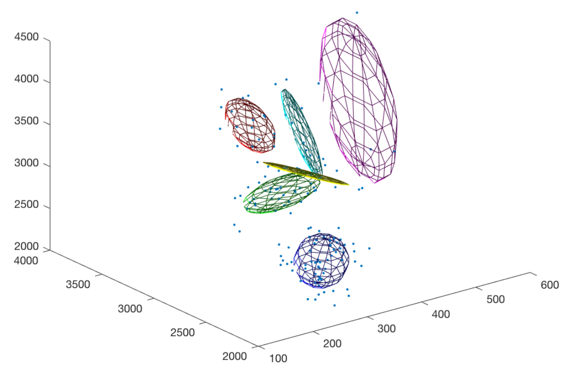
1. Put constraints in the form of the covariance matrix.
2. Regularise the covariance matrix by adding a diagonal matrix.

And I choose the second method to solve this problem by adding a diagonal matrix. Finally, it is worth noting that the clusters in the task1 scatter plot are a little similar with the results of "All F1 F2" 2D plot.

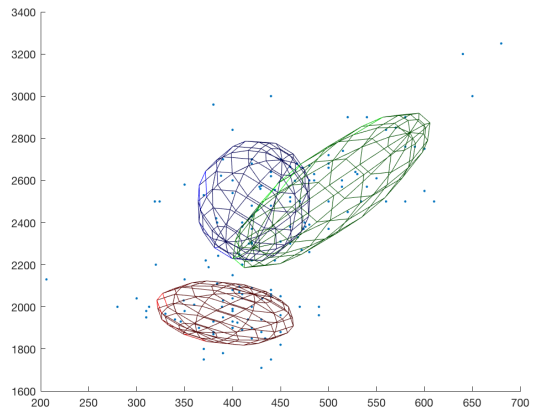
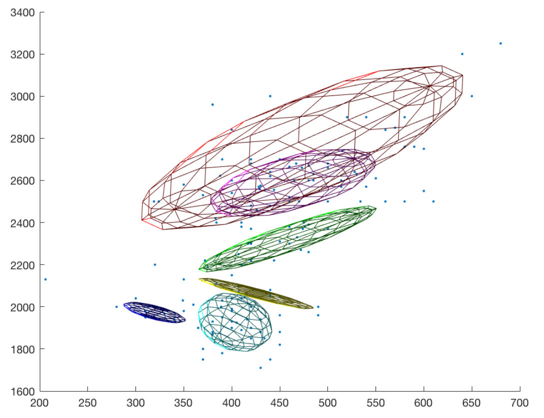
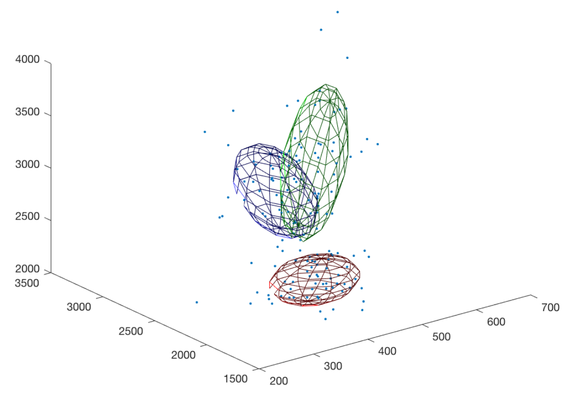
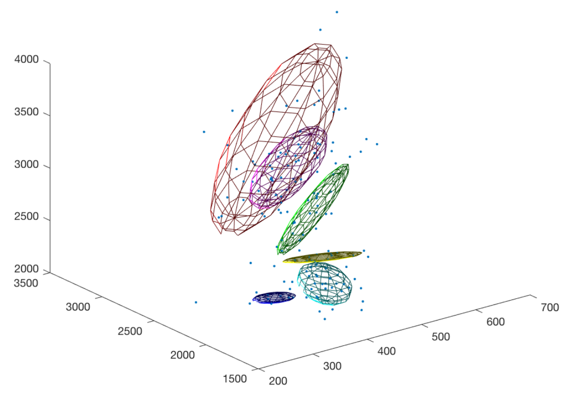
task5.m
```
% load the dataset
load('PB_data.mat');

% get phoneme 1, 2 data from the first and second formant frequencies
ph1F1 = f1(phno==1); ph1F2 = f2(phno==1);
ph2F1 = f1(phno==2); ph2F2 = f2(phno==2);
ph1Data = [ph1F1, ph1F2, ph1F1+ph1F2]; ph2Data = [ph2F1, ph2F2, ph2F1+ph2F2];

% the number of Gaussians used and test data
k = 3; % or k = 6
x = ph1Data; % or x = ph2Data

% code from mog.m...

% To fit general Gaussians use the line:
s2(:,:,i) = (x'-repmat(mu(:,i),1,n))*(repmat(Z(:,i),1,D).*(x'-repmat(mu(:,i),1,n))')./sum(Z(:,i)) + 2*eye(D);
```

Phoneme 1

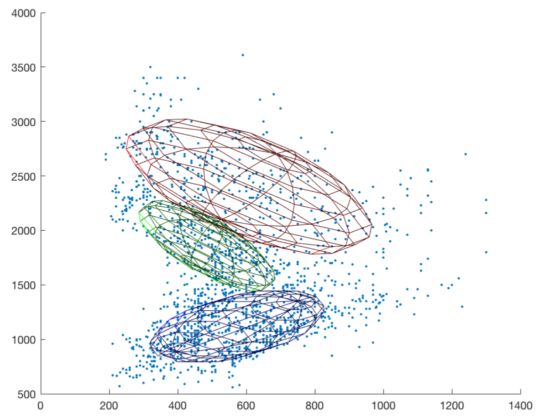
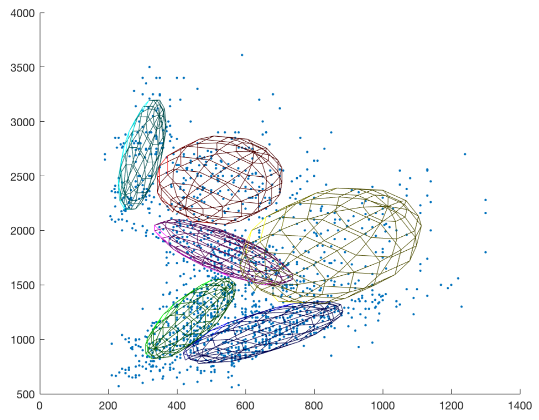
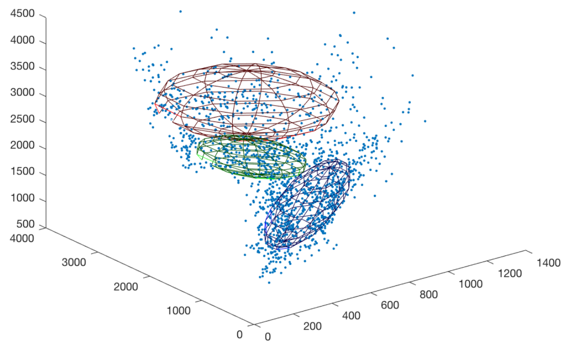
K = 3	K = 6
2D	
	

3D	
	

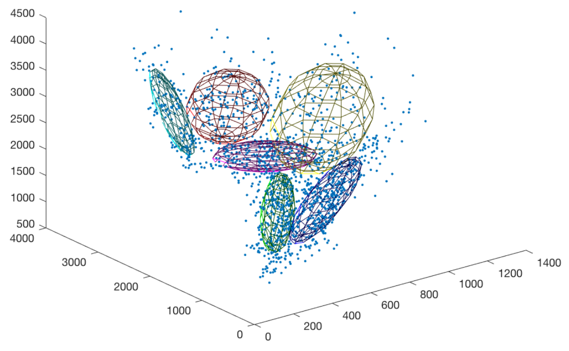
Phoneme 2

K = 3	K = 6
2D	
	
3D	
	

All F1 F2

K = 3	K = 6
2D	
	
3D	


4500 4000 3500 3000 2500 2000 1500 1000 500
4000 3000 2000 1000 0
0 200 400 600 800 1000 1200 1400

 | 

4500 4000 3500 3000 2500 2000 1500 1000 500
4000 3000 2000 1000 0
0 200 400 600 800 1000 1200 1400

 |