Machine Learning Programming
Assignment 4: Gaussian Mixture Models

Professor: Aude Billard
Assistants: Laura Cohen, Nadia Figueroa,
Murali Karnam and Denys Lamotte
contacts:
aude.billard@epfl.ch
laura.cohen@epfl.ch
nadia.figueroafernandez@epfl.ch

Winter Semester 2016

Introduction

This series of practicals focus on the development of machine learning algorithms introduced in the Applied Machine Learning course. In this practical, you will code the Expectation-Maximization algorithm for Gaussian Mixture Model learning in Matlab. You will then apply it to clustering applications and to classification using the GMM+Bayes method.

Submission Instructions

Deadline: December 7, 2016 @ 6pm. Assignments must be turned in by the deadline. 1pt will be removed for each day late. A day late starts one hour after the deadline.
Procedure: From the course Moodle webpage, the student should download and extract the .zip file named `TP4-GMM-Assignment.zip` which contains the following files:

Part 1 - EM Algorithm	Part 2 - Model Fitting + Clustering	Part 3 - Classification
my_gaussPDF.m	gmm_metrics	my_gmm_models.m
my_gmmLogLik.m	gmm_eval	my_gmm_classif
my_covariance.m	gmm_cluster	--
my_gmmInit.m	--	--
my_gmmEM.m	test_gmm_fit.m	--
test_gmm_2d.m	test_gmm_clustering_2d.m	test_gmm_classif_2d.m

As well as `TP4-GMM-Datasets.zip` which contains the datasets required to test your functions.

Assignment Instructions

The assignment consists on implementing the blue colored MATLAB functions from scratch. These functions can be tested with the `test_*.m`. These testing scripts depend on `ML_toolbox`, which must be downloaded from: `https://github.com/epfl-lasa/ML_toolbox`. Before proceeding make sure that all the sub-directories of the ML_toolbox have been added to your MATLAB search path. Once you have tested your functions, you can submit them as a .zip file with the name: `TP4-GMM-Assignment-SCIPER.zip` on the submission link in the Moodle webpage.

1 Part 1: Gaussian Mixture Models

A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of K Gaussian densities. GMMs are commonly used as a parametric model of the probability distribution of a dataset $\mathbf{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^M\}$ where $\mathbf{x}^i \in \mathbb{R}^N$. They are popular due to their capability of representing multi-modal sample distributions. The probability density function (pdf) of a K-component GMM is of the form,

$$p(\mathbf{x}|\Theta) = \sum_{k=1}^{K} \alpha_k p(\mathbf{x}|\mu^k, \mathbf{\Sigma}^k) \tag{1}$$

where $p(\mathbf{x}|\mu^k, \mathbf{\Sigma}^k)$ is the multivariate Gaussian pdf with mean μ^k and covariance $\mathbf{\Sigma}^k$

$$p(\mathbf{x}|\mu^k, \Sigma^k) = \frac{1}{(2\pi)^{N/2}|\Sigma^k|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu^k)^T(\Sigma^k)^{-1}(\mathbf{x}-\mu^k)\right\}. \tag{2}$$

$\Theta = \{\theta^1, \ldots, \theta^k\}$ is the complete set of parameters $\theta_k = \{\alpha_k, \mu^k, \Sigma^k\}$, where α_k are the priors (or mixing weights) of each Gaussian component, which satisfy the constraint $\sum_{k=1}^{K} \alpha_k = 1$.

GMMs are widely used in many areas of engineering, due to their modeling structure and flexibility they can be used for clustering, classification and regression purposes (in this assignment we will cover only the first two applications). The parameters $\mathbf{\Theta}$ can be estimated from training data using either a Maximum Likelihood parameter estimation approach through the iterative Expectation-Maximization (EM) algorithm or using Maximum A Posterior (MAP) estimation. In this assignment, we will implement the EM-algorithm for GMM parameter learning, following the steps covered in the Continuous Distributions slides 48-49 from the Applied Machine Learning course. For the derivation of the EM steps, refer to annexes provided in the course EM Annexes.

Maximum Likelihood (ML) Parameter Estimation for GMMs

The aim of ML estimation is to find the model parameters $\mathbf{\Theta}$ which maximize the likelihood of the GMM given the training dataset $\mathbf{X}$. For a dataset of M training data points, the GMM likelihood $\mathcal{L}(\Theta|\mathbf{X}) = p(\mathbf{X}|\Theta)$, assuming the data points are i.i.d (identically and independently distributed) is,

$$p(\mathbf{X}|\Theta) = \prod_{i=1}^{M} p(\mathbf{x}^i|\Theta) = \prod_{i=1}^{M} \sum_{k=1}^{K} \alpha_k p(\mathbf{x}^i|\mu^k, \mathbf{\Sigma}^k). \tag{3}$$

Unfortunately, this equation is a non-linear function of the parameters Θ and direct maximization is impossible. However, an ML estimate can be obtained iteratively using a special case of the expectation-maximization (EM) algorithm which tries to find the optimum of the likelihood, which is equivalent to finding the optimum of the log likelihood

$$\max_{\Theta} \quad \log \mathcal{L}(\Theta|\mathbf{X}) = \max_{\Theta} \quad \log p(\mathbf{X}|\Theta) \tag{4}$$

through the following steps:

1. **Initialization step:** Initialize the means $\mu = \{\mu^1, \ldots, \mu^k\}$ and priors $\alpha = \{\alpha_1, \ldots, \alpha_k\}$.
2. **Expectation Step:** For each Gaussian $k \in \{1, \ldots, K\}$, compute the probability that it is responsible for each point $\mathbf{x}^i$ in the dataset.
3. **Maximization Step:** Re-estimate the priors $\alpha = \{\alpha_1, \ldots, \alpha_K\}$, means $\mu = \{\mu^1, \ldots, \mu^K\}$ and Covariance matrices $\mathbf{\Sigma} = \{\mathbf{\Sigma}^1, \ldots, \mathbf{\Sigma}^K\}$
4. Go back to step 2 and repeat until the $\log \mathcal{L}(\Theta|\mathbf{X})$ stabilizes.

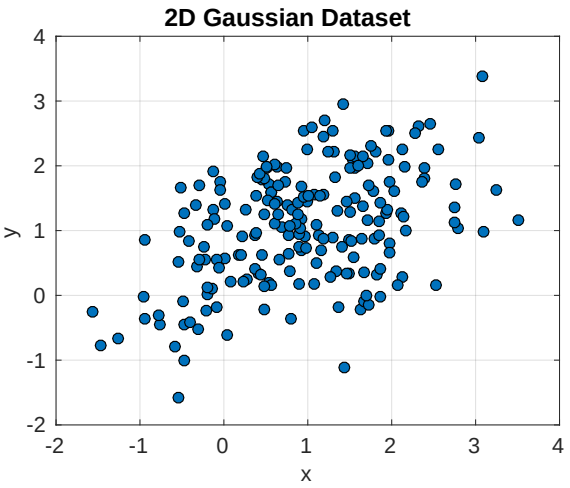

Figure 1: 2D Gaussian Dataset. Data points were sampled from $\mathcal{N}([1;1],[1,0.5;0.5,1])$.

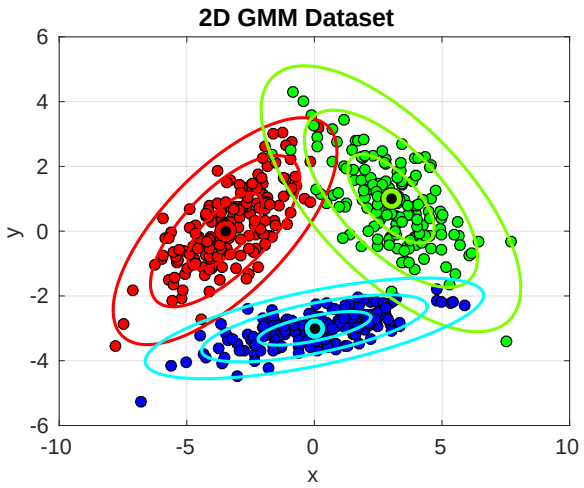

Figure 2: 2D GMM Dataset. Data points were sampled from a $K = 3$ GMM.

1.1 Gaussian PDF and GMM Likelihood

We will begin by implementing **two functions** that are used throughout the EM and model fitting algorithms, these are:

1. `my_gausspdf.m`: The **pdf** of a multivariate Gaussian function given a set of points $\{\mathbf{x}^1, \ldots, \mathbf{x}^D\}$, a mean μ and a Covariance matrix $\mathbf{\Sigma}$; i.e. Equation 2.
2. `my_gmmloglik.m`: The **log likelihood** of the parameters Θ of a learnt GMM for the given dataset X; i.e. Equation 3.

TASK 1: Implement my_gausspdf.m function

The first task is to implement the `my_gausspdf.m` function. The output should be a vector of probabilities with the length of the data points given. Each i-th element of `prob` is the probability of $\mathbf{x}^i$ being sampled from a Gaussian distribution parametrized by `Mu` and `Sigma`.

```
function [prob] =  my_gaussPDF(X, Mu, Sigma)
%MY_GAUSSPDF computes the Probability Density Function (PDF) of a
% multivariate Gaussian represented by a mean and covariance matrix.
%
% Inputs ----------------------------------------------------------------
%       o X     : (N x M), a data set with M samples each being of dimension N.
%                          each column corresponds to a datapoint
%       o Mu    : (N x 1), an Nx1 vector corresponding to the mean of the
%                           Gaussian function
%       o Sigma : (N x N), an NxN matrix representing the covariance matrix
%                          of the Gaussian function
%
% Outputs ---------------------------------------------------------------
%       o prob  : (1 x M),  a 1xM vector representing the probabilities for each
%                           M datapoints given Mu and Sigma
```

Implementation Hint: Useful functions `bsxfun()`, `repmat()`, `exp()`, `sqrt()`.

Test Implementation

To evaluate this function you should run the first sub-block (1a) of the `first` code block. This will load the 2D Gaussian Dataset shown in Figure 1. By running the **second** code block, the hidden `check_mygaussPDF.p` function will evaluate your `my_gaussPDF.m` function. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Checking my_gaussPDF.m ---
[Test 1] Checking my_gausspdf against ML_toolbox: Correct.
[Test 2] Checking my_gausspdf against MATLAB function: Correct.
```

Now, to compute the log of Equation 3 we can reinterpret the log likelihood as:

$$\begin{aligned}\log p(\mathbf{X}|\Theta) &= \log\Big(\prod_{i=1}^{M} p(\mathbf{x}^i|\Theta)\Big) \\ &= \sum_{i=1}^{M} \log\Big(p(\mathbf{x}^i|\Theta)\Big) \\ &= \sum_{i=1}^{M} \log\Big(\sum_{k=1}^{K} \alpha^k p(\mathbf{x}^i|\mu^k, \mathbf{\Sigma}^k)\Big).\end{aligned} \tag{5}$$

TASK 2: Implement my_gmmLogLik.m function

To implement Equation 5 in the `my_gmmLogLik.m` function. You can begin by computing the likelihood of each point $\mathbf{x}^i$ for each set of parameters $\{\mu^k, \mathbf{\Sigma}^k\}$. Then compute the inner sum (.) considering the priors α_k and finally sum the log-ed probabilities.

```
function [ll] =  my_gmmLogLik(X, Priors, Mu, Sigma)
%MY_GMMLOGLIK Compute the likelihood of a set of parameters for a GMM
%given a dataset X
%
%   input----------------------------------------------------------------
%
%       o X      : (N x M), a data set with M samples each being of
%                           dimension N, each column corresponds to a datapoint
%       o Priors : (1 x K), the set of priors (or mixing weights) for each
%                           k-th Gaussian component
%       o Mu     : (N x K), an NxK matrix corresponding to the centroids
%                           mu = {mu^1,...mu^K}
%       o Sigma  : (N x N x K), an NxNxK matrix corresponding to the
%                         Covariance matrices  Sigma = {Sigma^1,...,Sigma^K}
%
%   output ----------------------------------------------------------------
%
%      o ll       : (1 x 1) , loglikelihood
```

Implementation Hint: Useful functions `my_gaussPDF.m`, `log()`, `sum()`.

Test Implementation

To evaluate this function you should initially run the second sub-block (1b) of the `first` code block. This will load the 2D GMM Dataset shown in Figure 2. By running the **third** code block, the hidden `check_mygmmLogLik.p` function will evaluate your `my_gmmLogLik.m` function. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Checking my_gmmLogLik.m ---
[Test] Checking my_gmmLogLik against ML_toolbox: Correct.
```

1.2 Type of Covariance Matrix

Apart from the number of clusters K, another flexible design decision in GMMs is the type of Covariance matrix $\mathbf{\Sigma}$ to use for each Gaussian component. Three types of $\mathbf{\Sigma}$ can be used:

1. **Full** Covariance matrix: The full Covariance matrix is computed as follows:

$$\mathbf{\Sigma}_{full} = \frac{1}{M-1}\mathbf{X}\mathbf{X}^T, \tag{6}$$

   where X is the zero-mean data ($X \rightarrow X - \bar{X}$), where $\bar{X}$ is the mean of the data. For a 2D dataset the form of $\mathbf{\Sigma}$ is:

$$\mathbf{\Sigma}_{full} = \begin{bmatrix} \sigma_x^2 & \sigma_x\sigma_y \\ \sigma_y\sigma_x & \sigma_y^2 \end{bmatrix}, \quad \mathbf{\Sigma} \in \mathbb{R}^{x+y}.$$

2. **Diagonal** Covariance matrix: A diagonal Covariance matrix of a dataset only considers the variance in the principal directions; i.e. disregarding the off-diagonal elements. For a 2D dataset it should have the following form:

$$\mathbf{\Sigma}_{diag} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}, \quad \mathbf{\Sigma} \in \mathbb{R}^{x+y}$$

   This can be computed from Equation 6 and extracting the diagonal values.

3. **Isotropic** Covariance matrix: Otherwise known as the circular Covariance matrix is solely dependent on the squared distance $|\mathbf{x} - \mathbf{x}'|^2$ between the original points (i.e. non-zero mean) and their mean μ. Hence, one must compute the isotropic variance as follows:

$$\sigma_{iso}^2 = \frac{1}{NM}\sum_{i=1}^{M} ||\mathbf{x}^i - \bar{\mathbf{x}}||^2. \tag{7}$$

   Then replicate it in a diagonal matrix, to have the following form:

$$\mathbf{\Sigma}_{iso} = \begin{bmatrix} \sigma_{iso}^2 & 0 \\ 0 & \sigma_{iso}^2 \end{bmatrix}, \quad \mathbf{\Sigma} \in \mathbb{R}^{x+y}.$$

TASK 3: Implement my_covariance.m function
Now you will implement `my_covariance.m` function with input $\mathbf{X}$, $\bar{\mathbf{X}}$ and `type = {'full','diag','iso'}`. Your output is $\mathbf{\Sigma} \in \mathbb{R}^{N \times N}$.

```
function [Sigma] =  my_covariance(X, X_bar, type)
%MY_COVARIANCE computes the covariance matrix of X given a covariance type
% and the mean of the dataset X (X_bar).
% Inputs ----------------------------------------------------------------
%       o X     : (N x M), a data set with M samples each being of dimension N.
%                          each column corresponds to a datapoint
%       o X_bar : (N x 1), an Nx1 matrix corresponding to mean of data X.
%       o type  : string , type={'full', 'diag', 'iso'} of Covariance matrix
%
% Outputs ---------------------------------------------------------------
%       o Sigma : (N x N), an NxN matrix representing the covariance matrix
%                          of the Gaussian function
```

Test Implementation

To evaluate this function you should initially run the first sub-block (1a) of the `first` code block. This will load the 2D Gaussian Dataset shown in Figure 1. By running the **fourth** code block, the hidden `check_mycovariance.p` function will evaluate your `my_covariance.m` function. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Checking my_covariance.m ---
[Test 1] Checking Full Covariance against ML_toolbox: Correct.
[Test 2] Checking Diagonal Covariance against ML_toolbox: Correct.
[Test 3] Checking Isotropic Covariance against ML_toolbox: Correct.
```

You can further evaluate your `my_covariance.m` function by running the `fifth` code block, which will plot 3 figures corresponding to the contours of the first 3 standard deviation of the computed Covariance matrices (Figure 3, 4 and 5).

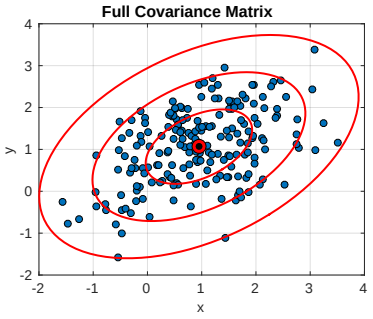

Figure 3: Full Covariance matrix.

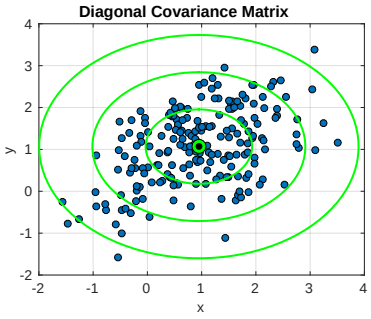

Figure 4: Diagonal Covariance matrix.

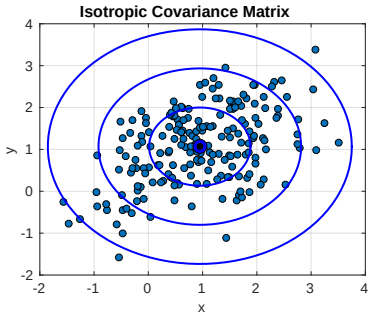

Figure 5: Isotropic Covariance matrix.

Moreover, the expected results are:

$$\mathbf{\Sigma}_{full} = \begin{bmatrix} 0.9631 & 0.3890 \\ 0.3890 & 0.7899 \end{bmatrix}, \quad \mathbf{\Sigma}_{diag} = \begin{bmatrix} 0.9631 & 0 \\ 0 & 0.7899 \end{bmatrix}, \quad \mathbf{\Sigma}_{iso} = \begin{bmatrix} 0.8721 & 0 \\ 0 & 0.8721 \end{bmatrix}$$

1.3 Expectation-Maximization for Estimating GMM Parameters

Step 1. Initialize Priors α^k Centroids μ_k

The EM algorithm for GMM begins with an initialization step. Here we shall initialize for iteration $t = 0$ the set of priors $\alpha^{(0)} = \{\alpha_1^{(0)}, \ldots, \alpha_K^{(0)}\}$ to uniform probabilities, the means $\mu^{(0)} = \{\mu^{1(0)}, \ldots, \mu^{K(0)}\}$ with your `my_kmeans.m` function from `TP2-KMeans-Assignment` and $\mathbf{\Sigma}^{(0)} = \{\mathbf{\Sigma}^{1(0)}, \ldots, \mathbf{\Sigma}^{K(0)}\}$ with your `my_covariance.m` function.

TASK 4: Implement my_gmmInit.m function

Now you will implement `my_gmmInit.m` function with input $\mathbf{X}$, K and `cov_type = {'full','diag','iso'}`. Your output should be $\alpha^{(0)} = \{\alpha_1^{(0)}, \ldots, \alpha_K^{(0)}\}$, $\mu^{(0)} = \{\mu^{1(0)}, \ldots, \mu^{K(0)}\}$ and $\mathbf{\Sigma}^{(0)} = \{\mathbf{\Sigma}^{1(0)}, \ldots, \mathbf{\Sigma}^{K(0)}\}$.

```
function [Priors, Mu, Sigma] =  my_gmmInit(X, K, cov_type, plot_iter)
%MY_GMMINIT Computes initial estimates of the parameters of a GMM
% to be used for the EM algorithm
%   input----------------------------------------------------------------
%
%       o X         : (N x M), a data set with M samples each being of
%                           dimension N, each column corresponds to a datapoint.
%       o K         : (1 x 1) number K of GMM components.
%       o cov_type  : string ,{'full', 'diag', 'iso'} type of Covariance matrix
%       o plot_iter : (bool)  set to 1 of want to visalize initual Mu's and
%                           Sigma's, works only for N=2
%   output ----------------------------------------------------------------
%       o Priors : (1 x K), the set of priors (or mixing weights) for each
%                           k-th Gaussian component
%       o Mu     : (N x K), an NxK matrix corresponding to the centroids
%                           mu = {mu^1,...mu^K}
%       o Sigma  : (N x N x K), an NxNxK matrix corresponding to the
%                         Covariance matrices  Sigma = {Sigma^1,...,Sigma^K}
```

Implementation Hint: Use `my_covariance()` to compute initial Σ's.

Test Implementation

To evaluate this function you should initially run the second sub-block (1b) of the **`first`** code block. This will load the 2D GMM Dataset shown in Figure 2. By running the **sixth** code block you can visualize your initial μ's and $\mathbf{\Sigma}$'s as in Figure 6, by modifying `K` and `cov_type` to different parameters you can visualize the different initializations achieved as in Figure 7-14). If you achieve figures similar to these, you can move on to the next task. To disable plotting (which will only work for $N = 2$) set `plot=0`.

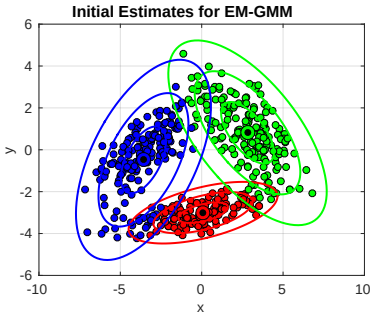

Figure 6: Initial parameters `K=3` and `cov_type='full'`

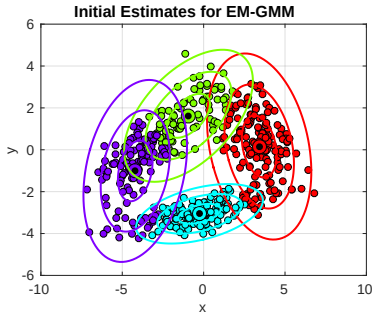

Figure 7: Initial parameters `K=4` and `cov_type='full'`

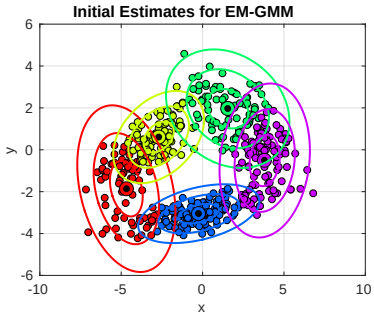

Figure 8: Initial parameters `K=5` and `cov_type='full'`

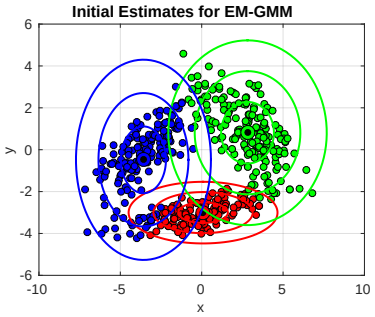

Figure 9: Initial parameters `K=3` and `cov_type='diag'`

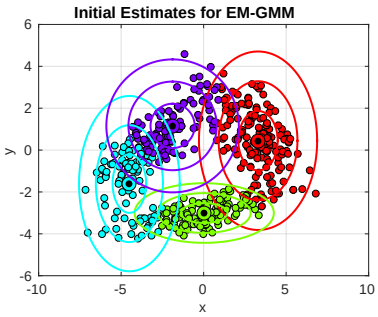

Figure 10: Initial parameters `K=4` and `cov_type='diag'`

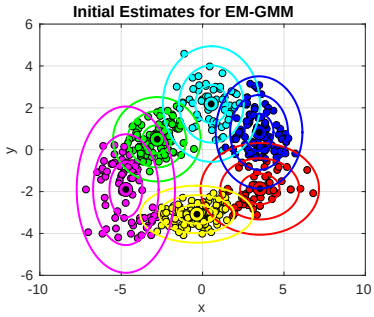

Figure 11: Initial parameters `K=6` and `cov_type='diag'`

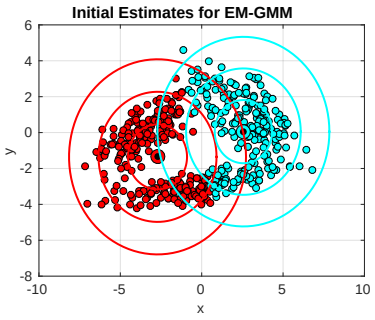

Figure 12: Initial parameters `K=2` and `cov_type='iso'`

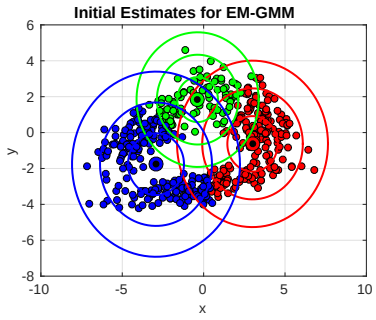

Figure 13: Initial parameters `K=3` and `cov_type='iso'`

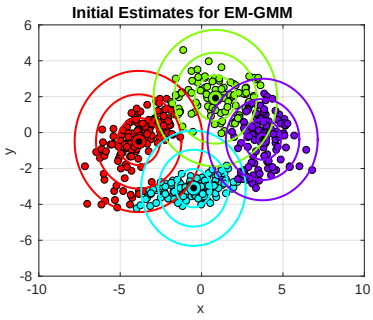

Figure 14: Initial parameters `K=4` and `cov_type='iso'`

Step 2. Expectation Step: Membership probabilities

At each iteration t, estimate, for each Gaussian k, the probability that this Gaussian is responsible for generating each point of the dataset. The *a posteriori* probability for a k-th component is given by

$$p(k|\mathbf{x}^i, \Theta^{(t)}) = \frac{\alpha_k^{(t)} p(\mathbf{x}^i|\mu^k, \mathbf{\Sigma}^k)}{\sum_{j=1}^{K} \alpha_j^{(t)} p(\mathbf{x}^i|\mu^{j(t)}, \Sigma^{j(t)})}. \tag{8}$$

These probabilities are the output of the Expectation Step. They must be computed for each $k \in \{1, \ldots, K\}$ for all datapoints $i \in \{1, \ldots, M\}$.

Step 3. Maximization Step: Update Priors α, Means μ and Covariances $\mathbf{\Sigma}$

In order to maximize the log-likelihood of the current estimate we update the priors $\alpha^{(t+1)} = \{\alpha_1^{(t+1)}, \ldots, \alpha_K^{(t+1)}\}$ with the following equation:

$$\alpha_k^{(t+1)} = \frac{1}{M} \sum_{i=1}^{M} p(k|\mathbf{x}^i, \Theta^{(t)}) \tag{9}$$

where $p(k|\mathbf{x}^i, \Theta^{(t)})$ is given by Eq. 8. The means are updated by the following equation:

$$\mu^{k(t+1)} = \frac{\sum_{i=1}^{M} p(k|\mathbf{x}^i, \Theta^{(t)}) \mathbf{x}^i}{\sum_{i=1}^{M} p(k|\mathbf{x}^i, \Theta^{(t)})}. \tag{10}$$

Finally, the Covariance matrices for each k-th component are computed with the following equation:

$$\mathbf{\Sigma}^{k(t+1)} = \frac{\sum_{i=1}^{M} p(k|\mathbf{x}^i, \Theta^{(t)})(\mathbf{x}^i - \mu^{k(t+1)})(\mathbf{x}^i - \mu^{k(t+1)})^T}{\sum_{i=1}^{M} p(k|\mathbf{x}^i, \Theta^{(t)})}. \tag{11}$$

For `full` Covariance matrices Eq. 11 can be used directly to update the $\mathbf{\Sigma}^{(t+1)}$. For `diagonal` Covariance matrices, you can compute Eq. 11 and then simply extract the diagonal elements.

Now, for `isotropic` matrices, one must use the following update equation:

$$\mathbf{\Sigma}_{iso}^{k(t+1)} = \frac{\sum_{i=1}^{M} p(k|\mathbf{x}^i, \Theta^{(t)})||\mathbf{x}^i - \mu^{k(t+1)}||^2}{N \sum_{i=1}^{M} p(k|\mathbf{x}^i, \Theta^{(t)})}. \tag{12}$$

Every so often, during EM iterations, certain Covariance matrices can become ill-conditioned, i.e. the matrix tends to be a singular matrix. In the likelihood sense, this means that the likelihood is converging towards infinity. This happens for various reasons; namely (i) there might be more parameters than datapoints or (ii) the variables are highly correlated. Hence, after estimating the Σ's we must add a tiny variance to avoid this numerical instability, this can be done by adding a very very small number $\epsilon = 1e^{-5}$ to the diagonal values of each Σ^k .

TASK 4: Implement my_gmmEM.m function

Your task is now to implement Step 1 (Initialization) and iterate through Step 2 (E-Step) and Step 3 (M-Step) until the log likelihood of your current parameters Θ (Eq. 5) is stabilized. This iterative loop should be implemented in `my_gmmEM.m` function with input $\mathbf{X}$, K and `cov_type = {'full','diag','iso'}`. Your output should be $\alpha = \{\alpha_1, \ldots, \alpha_K\}$, $\mu = \{\mu^1, \ldots, \mu^K\}$ and $\mathbf{\Sigma} = \{\mathbf{\Sigma}^1, \ldots, \mathbf{\Sigma}^K\}$.

```
function [Priors, Mu, Sigma] =  my_gmmEM(X, K, cov_type, plot_iter)
%%%%%% STEP 1: Initialization of Priors, Means and Covariances %%%%%%
...
while true
    %%%%%% STEP 2: Expectation Step: Membership probabilities %%%%%%
    %1) %%% Compute probabilities p(x^i|k)
        ..
    %2) %%% Compute posterior probabilities p(k|x)
        ..
    %%%%%% STEP 3: Maximization Step: Update Priors, Means and Sigmas %%%%%%
    %1) %%% Update Priors
        ..
    %2) %%% Update Means and Covariance Matrices
        for k=1:K
            ...
            ...
            % Add a tiny variance to avoid numerical instability
            ..
        end
    %%%%%% Stopping criterion %%%%%%
    ..
end
```

Implementation Hint: DO NOT use `my_covariance.m` for this function, you should implement Equations 11 and 12.

Test Implementation

To evaluate this function you should initially run the second sub-block (1b) of the `first` code block. This will load the 2D GMM Dataset shown in Figure 2. By running the **seventh** code block you can visualize your initial μ's and $\mathbf{\Sigma}$'s as in Figure 15, your final α's, μ's and $\mathbf{\Sigma}$'(as in Figure 16) and the a visual representation of the pdf of the learnt parameters; i.e. Eq. 1 (Figure 17). By modifying `K` and `cov_type` to different parameters you can visualize the different results as in Figure 18-26). If you achieve figures similar to these, you can move on to the next task. To disable plotting of the initial and final parameters (which will only work for $N = 2$) set `plot_iter=0`.

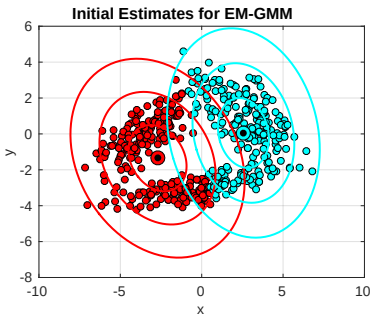

Figure 15: Initial parameters `K=2` and `cov_type='full'`

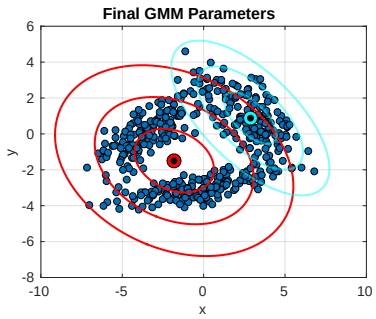

Figure 16: Final parameters `K=2` and `cov_type='full'`

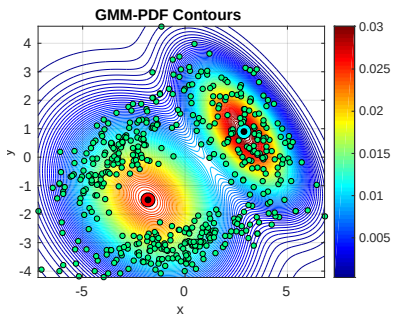

Figure 17: PDF of GMM parameters `K=2` and `cov_type='full'`

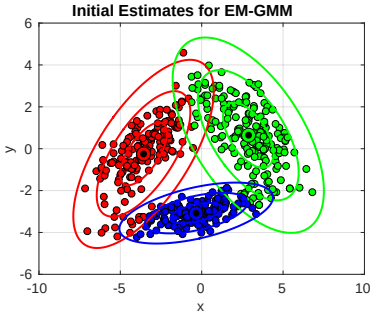

Figure 18: Initial parameters `K=3` and `cov_type='full'`

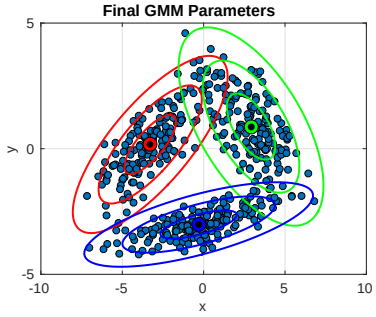

Figure 19: Final parameters `K=3` and `cov_type='full'`

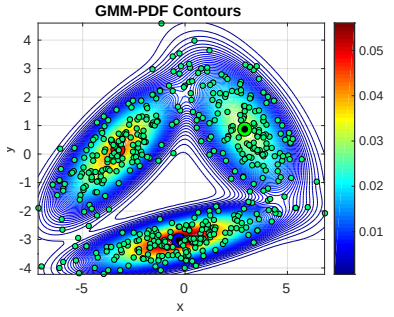

Figure 20: PDF of GMM parameters `K=3` and `cov_type='full'`

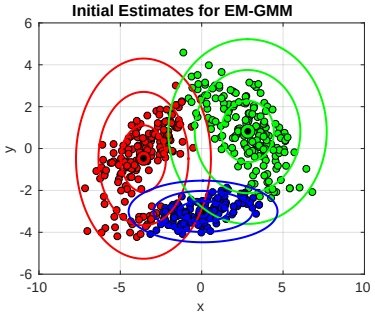

Figure 21: Initial parameters `K=3` and `cov_type='diag'`

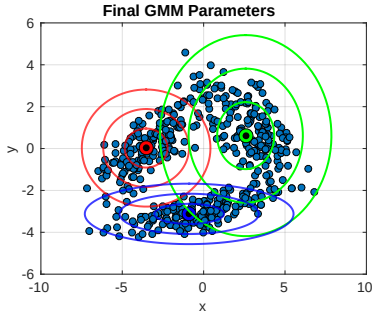

Figure 22: Final parameters `K=3` and `cov_type='diag'`

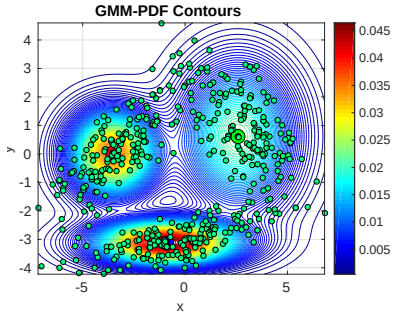

Figure 23: PDF of GMM parameters `K=3` and `cov_type='diag'`

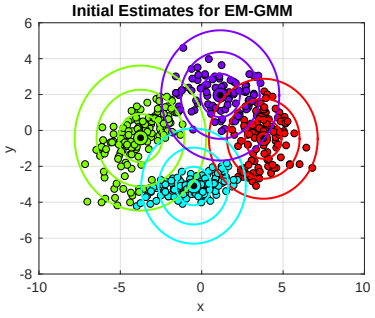

Figure 24: Initial parameters `K=4` and `cov_type='iso'`

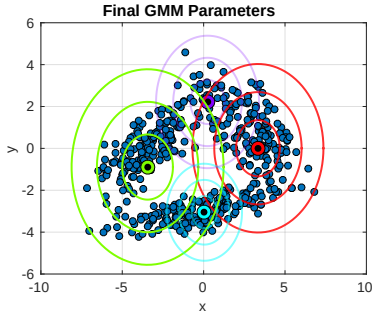

Figure 25: Final parameters `K=4` and `cov_type='iso'`

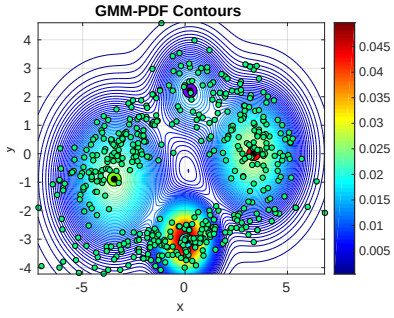

Figure 26: PDF of GMM parameters `K=4` and `cov_type='iso'`

2 Part 2: Model Fitting and Clustering with GMM

2.1 Fitting Gaussian Mixture Models

From Equation 1, one can observe that the set of model parameters are $\{K, (\alpha^k, \mu^k, \mathbf{\Sigma}_k)_{k=1}^K\}$. As in k-means, we can use the AIC and BIC metrics for model selection. To recall:

- **AIC:** The AIC metric is a maximum-likelihood measure that penalizes for model complexity as follows:
$$AIC = -2\ln\mathcal{L} + 2B \tag{13}$$
where $\mathcal{L}$ the likelihood of the model and B the total number of model parameters.

- **BIC:** The BIC metric goes even further and penalizes for number of datapoints as well with the following equation as follows:
$$BIC = -2\ln\mathcal{L} + \ln(M)B \tag{14}$$
where M is the total number of datapoints.

For GMMs the computation of the total model parameters B is more involved than k-means. For a dataset with $\mathbf{x} \in \mathbb{R}^N$, the total number of parameters to learn for a K-component GMM with **full** Covariance matrix is

$$B_{full} = K \times (1 + N + N \times (N-1)/2) - 1 \tag{15}$$

with -1 corresponding to the priors constraint $\sum_{k=1}^K \alpha^k = 1$. For the case of **diagonal** Covariance matrices,

$$B_{diag} = K \times (1 + N + N) - 1 \tag{16}$$

and for **isotropic** Covariance matrices the number of parameters is computed as follows:

$$B_{iso} = K \times (1 + N + 1) - 1 \tag{17}$$

TASK 5: Implement gmm_metrics.m function

```
function [AIC, BIC] =  gmm_metrics(X , Priors, Mu, Sigma, cov_type)
%GMM_METRICS Computes the metrics (AIC, BIC) for model fitting
%
%   input ----------------------------------------------------------------
%
%       o X        : (N x M), a data set with M samples each being of
%                            dimension N each column corresponds to a datapoint
%       o Priors : (1 x K), the set of priors (or mixing weights) for each
%                            k-th Gaussian component
%       o Mu     : (N x K), an NxK matrix corresponding to the centroids
%                            mu = {mu^1,...mu^K}
%       o Sigma  : (N x N x K), an NxNxK matrix corresponding to the
%                           Covariance matrices  Sigma = {Sigma^1,...,Sigma^K}
%       o cov_type : string ,{'full', 'diag', 'iso'} type of Covariance matrix
%
%   output ---------------------------------------------------------------
%
%       o AIC      : (1 x 1), Akaike Information Criterion
%       o BIC      : (1 x 1), Bayesian Information Criteria
```

Test Implementation

To evaluate your implementation, you will use the testing script `test_gmm_fit.m` in Part2 of your assignment directory. By running the code block **1a)**, the 2D GMM testing dataset will be loaded (Fig. 2). Now by running the **second** code block, the hidden `check_gmmMetrics.p` function will evaluate your `gmm_metrics.m` function. This will evaluate your AIC/BIC implementations for different K values and different types of Covariance matrices.If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Checking gmm_metrics.m ---
[Test 1] Checking AIC/BIC for K=1 w/full Covariance against ML_toolbox: Correct.
[Test 2] Checking AIC/BIC for K=1 w/diag Covariance against ML_toolbox: Correct.
[Test 3] Checking AIC/BIC for K=1 w/iso  Covariance against ML_toolbox: Correct.
[Test 4] Checking AIC/BIC for K=4 w/full Covariance against ML_toolbox: Correct.
...
...
```

Choosing the optimal K

For choosing the optimal K that best describes our dataset with a GMM, we follow the same procedure as in K-means. We estimate the GMM parameters for a range of K values, 10 times each. For each K we select the best run, which in *likelihood* terms, means the run with the **maximum** likelihood. We then plot the values and select the K which yields the best tradeoff between *likelihood* and *model complexity*.

TASK 6: Implement gmm_eval.m function

```
function [] =  gmm_eval(X , K_range, cov_type)
%GMM_EVAL Implementation of the GMM Model Fitting with AIC/BIC metrics.
%
%   input ------------------------------------------------------------------
%
%       o X        : (N x M), a data set with M samples each being of
%                    dimension N, each column corresponds to a datapoint
%       o repeats  : (1 X 1), # times to repeat k-means
%       o K_range  : (1 X K), Range of k-values to evaluate
%       o cov_type : string ,{'full', 'diag', 'iso'} type of Covariance matrix
%
%   output -----------------------------------------------------------------
```

Test Implementation

By running the **third** code block, your `gmm_eval.m` function will be evaluated. You should obtain a plot as in Figure 27. By modifying `cov_type` to `'diag'` and `'iso'`, you should obtain plots similar to Figure 28 and Figure 29. By simply looking at Figure 27, one can quickly infer that $K = 3$ with `cov_type='full'` is the best parametrization for our data. However, by taking a deeper look at Figure 28 and 29, it can be seen that the AIC/BIC values are much higher than for $K = 3$ with full covariance. This is due to the fact that depending on the choice of Covariance matrix one can achieve the same or even better likelihood fits, but with less complex models. In our case, we know that our dataset was sampled from s 3D full Covariance GMM, if we were going to use this model for clustering, this would be the best choice (see Figure 30). Nevertheless, GMMs can be used for classification and regression and in these cases, matching the original model might not be the priority, rather having a model with high likelihood or low model complexity (see Figure 31 and 32).

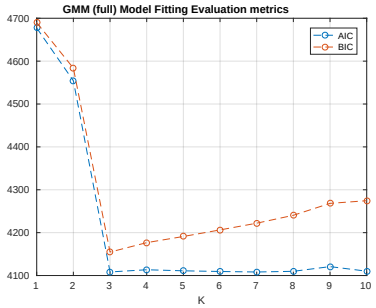

Figure 27: GMM Fitting metrics for `K_range=1:10` and `cov_type='full'`

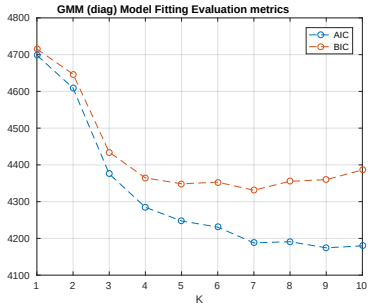

Figure 28: GMM Fitting metrics for `K_range=1:10` and `cov_type='diag'`

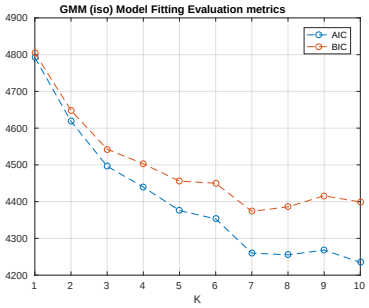

Figure 29: GMM Fitting metrics for `K_range=1:10` and `cov_type='iso'`

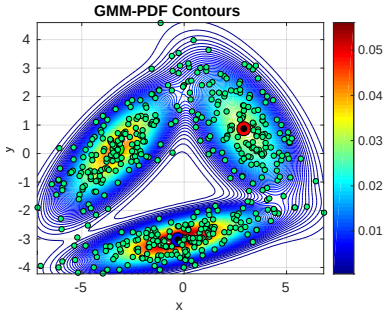

Figure 30: Optimal GMM `K=3`, `cov_type='full'`

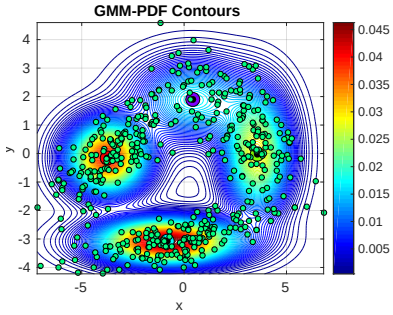

Figure 31: Optimal GMM `K=1`, `cov_type='diag'`

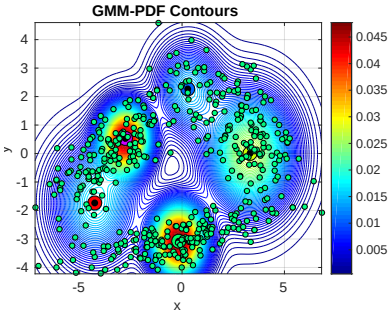

Figure 32: Optimal GMM `K=1`, `cov_type='iso'`

The previously analyzed dataset was somewhat ideal for model selection, as it was generated from a GMM itself. However, these metrics tend to yield different results for data that is not clearly described by a mixture of Gaussians or is not multi-modal. Take for example the data in Figure 33 (which can be loaded by running code block 1b)) in the MATLAB testing script `test_gmm_fit.m`. By running the **third** code block and testing for the three different types of Covariance matrix. As can be seen from Figures 34,35 and 36 there is no clear elbow or "optimal" point from the range of $K = 1 : 10$. The AIC metric always tends to the maximum

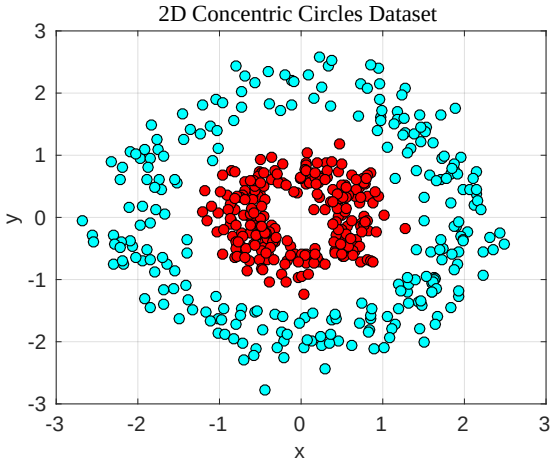

Figure 33: 2D Concentric Dataset

number of clusters in these cases however, we can see that the BIC does penalize for model complexity. By analyzing the BIC curve for the `diagonal` and `isotropic` type of GMM, we

can see that, in fact, the "optimal" K is 1, which would be the most suitable representation of this particular dataset without considering the labels, as can be seen in Figure 39 and 38. If we had solely done model selection with a `full` Covariance matrix, one might have chosen $K = 5$ or $K = 8$, as in Figure 37. Another way of evaluating your models when the AIC/BIC curves are not that informative, one can monitor the increase in the Likelihood of the model and select the "optimal" K once the likelihood stabilizes.

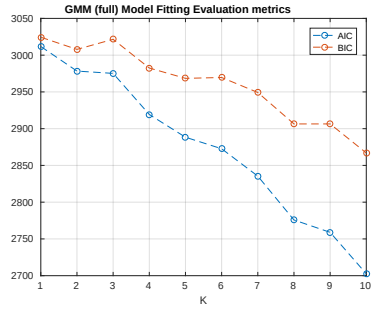

Figure 34: GMM Fitting metrics for `K_range=1:10` and `cov_type='full'`

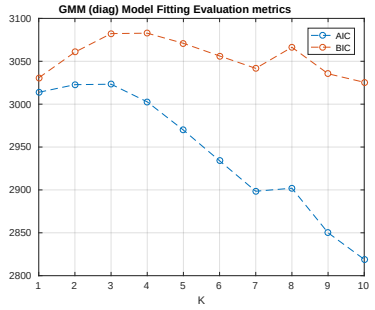

Figure 35: GMM Fitting metrics for `K_range=1:10` and `cov_type='diag'`

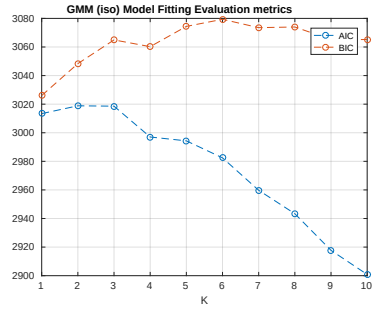

Figure 36: GMM Fitting metrics for `K_range=1:10` and `cov_type='iso'`

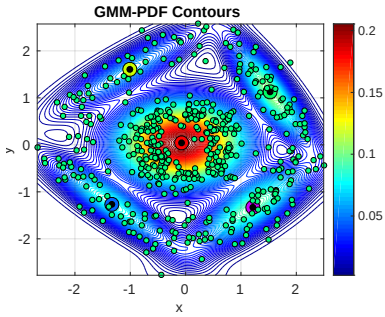

Figure 37: Optimal GMM `K=5`, `cov_type='full'`

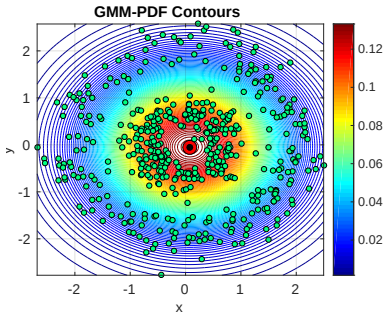

Figure 38: Optimal GMM `K=1`, `cov_type='diag'`

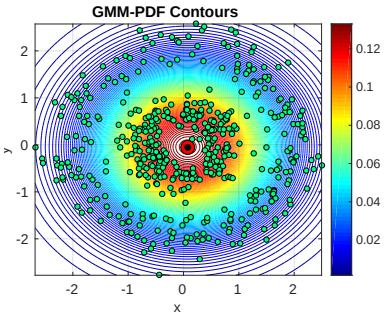

Figure 39: Optimal GMM `K=1`, `cov_type='iso'`

2.2 Clustering with GMM

In this part we will use GMM to cluster data and compare it with the K-Means algorithm on a simple 2-D dataset. To read about clustering using GMM, refer to the slides 45-56 in Continuous Distributions from the Applied Machine Learning course. You will work with the files, `test_gmm_clustering-2d.m` and `gmm_cluster.m` in this part of the assignment.

We begin by loading the data set, by running the first block of the code in the script, `test_gmm_clustering-2d.m`. The next task is to fit a GMM to this dataset, as elaborated in Section 2.1. By running the **second block** of the code in the script you can visualize the metrics and decide on the desired number of clusters, K.

Finding the clusters

Next, we assign each datapoint to a cluster (one of the K Gaussian components used to fit the data) based on the posterior probability $p(k|x^i, \Theta^k)$, which is found using Equation 8, used in the Expectation Step while finding the GMM. This will be a $K \times M$ matrix, with the probability for each cluster for each datapoint.

We will implement two ways in which a datapoint is assigned to a cluster – hard and soft clustering.

- In hard clustering, the cluster which has the highest probability is assigned to the data point.
- In soft clustering, a datapoint is assigned not labeled (given cluster 0), if the confidence for clustering is low. For this implementation we shall use the following criteria to ascertain that the confidence is low:

$$t_{\min} < \max_k(p(k|x^i, \Theta^k)) \quad \text{and any other } p(k|x^i, \Theta^k) \quad \forall \quad k < t_{\max}. \tag{18}$$

  That is, if the highest probability is in a range specified by a threshold (given by $t_{\min}$ and $t_{\max}$), along with another cluster, it is unlabeled. The confidence to classify these datapoints is low since more than one cluster have similar probability as specified by the threshold. Otherwise, the it is assigned the cluster as in hard clustering.

TASK 7: Implement gmm_cluster.m function You will now implement the above steps to cluster the dataset and assign a label to each point in`gmm_cluster.m`. Your output should be the cluster labels for the complete dataset.

```
function [labels] =  gmm_cluster(X, Priors, Mu, Sigma, type, softThresholds)
%GMM_CLUSTER Computes the cluster labels for the data points given the GMM
%
%   input ------------------------------------------------------------------
%
%       o X      : (N x M), a data set with M samples each being of ...
    dimension N.
%                           each column corresponds to a datapoint
%       o Priors : (1 x K), the set of priors (or mixing weights) for each
%                           k-th Gaussian component
%       o Mu     : (N x K), an NxK matrix corresponding to the centroids
%                           mu = {mu^1,...mu^K}
%       o Sigma  : (N x N x K), an NxNxK matrix corresponding to the
%                           Covariance matrices  Sigma = {Sigma^1,...,Sigma^K}
%       o type   : string ,{'hard', 'soft'} type of clustering
%
%       o softThresholds: (2 x 1), a vecor for the minimum and maximum of
%                           the threshold for soft clustering in that order
%
%   output -----------------------------------------------------------------
%
%       o labels   : (1 x M), a M dimensional vector with the label of the
%                             cluster for each datapoint
%                             - For hard clustering, the label is the
%                             cluster number.
%                             - For soft clustering, the label is 0 for
%                             data points which do not have high confidence
%                             in cluster assignment
```

Test Implementation

By running the **third** block of the script, `test_gmm_clustering-2d.m`, the dataset will be clustered and visualized. You should obtain a plot as in Figure 40 and Fig 41. By modifying the type of clustering between `type='soft'`, you should obtain plots similar to Figure 42 and Fig 43.

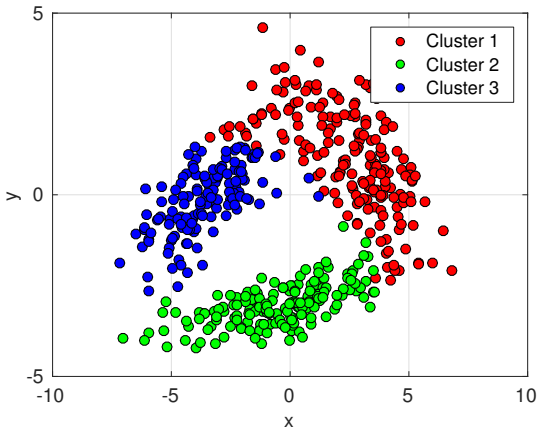

Figure 40: Output of hard clustering the 2-D dataset with GMM, `K=3`, `cov_type='full'`.

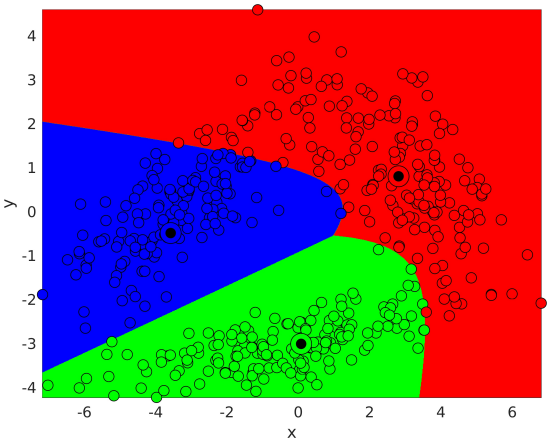

Figure 41: The cluster boundaries for the GMM trained using the 2-D dataset with `K=3`, `cov_type='full'`.

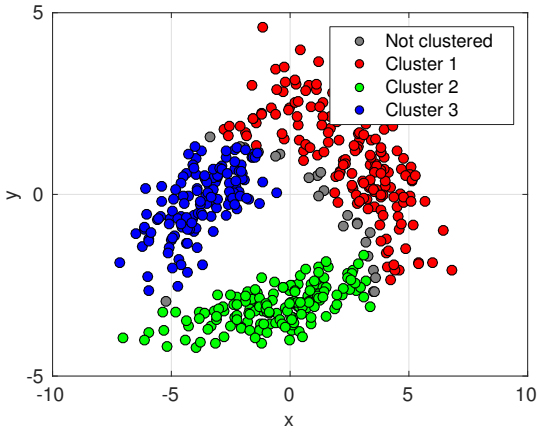

Figure 42: Output of soft clustering the 2-D dataset with GMM, `K=3`, `cov_type='full'`, and the soft thresholds set to [0.3, 0.7]. Here the points in gray do not have confidence to be clustered completely.

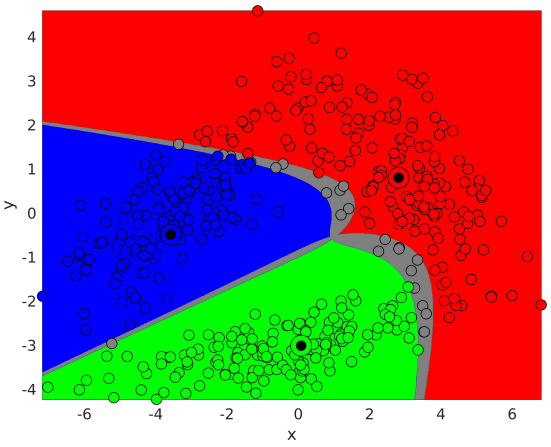

Figure 43: The cluster boundaries for the GMM trained using the 2-D dataset with `K=3`, `cov_type='full'`, and the soft thresholds set to [0.3, 0.7]. The region in gray is where the confidence to cluster is low.

Compare with K-Means

We can now compare the clustering with K-Means. Update the **fourth** block of the script, `test_gmm_clustering-2d.m`, to find the optimal K for K-Means and plot the decision boundaries for this dataset. Upon running this block of the code, you should visualize the K-Means result as shown in Figure 44.

Finally, you can use the F1-measure to compare the clustering quantitatively. For this dataset, subject to the initialization you should observe a higher F1-measure for the clustering using GMM ($\approx$0.9780) as compared with K-Means ($\approx$0.9126).

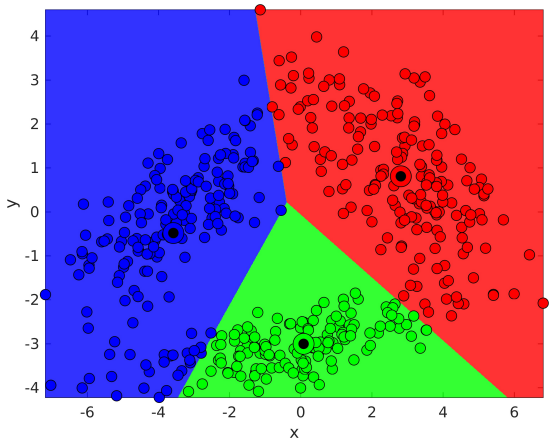

Figure 44: The decision boundary of clustering the 2-D dataset with K-Means, with `K=2`.

Testing another 2d dataset

Now we apply the same clustering to another two dimensional dataset, which can be loaded using the **block 1.b** of the `test_gmm_clustering-2d.m` script. After that, you should be able to run all the sections of the code with the appropriate K for GMM and K-means. The results should look as in the Figure 45 to Figure 53.

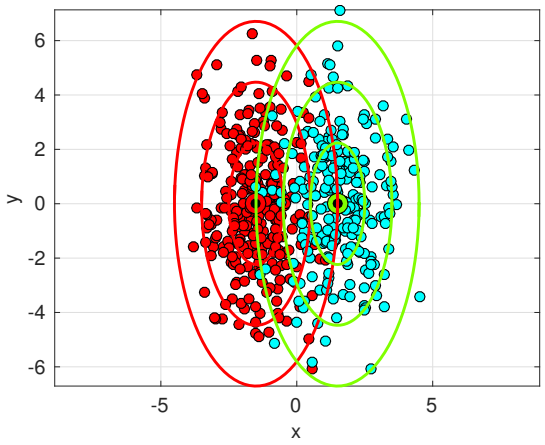

Figure 45: The second 2-D dataset generated from mixture of two Gaussians to be used for clustering.

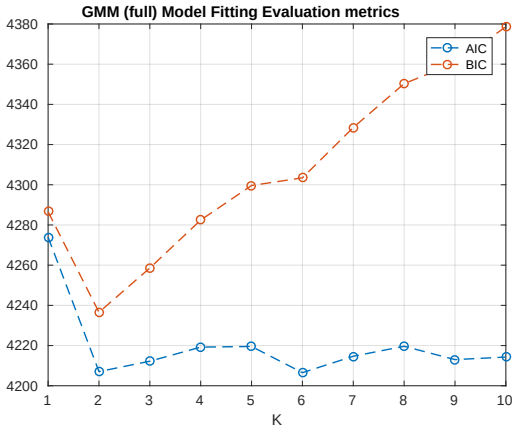

Figure 46: The AIC and BIC curves for the second dataset to fit it with a GMM, for K range from 1 to 10. From this it is clear that $K = 2$ is the best, which is the same as the ground truth.

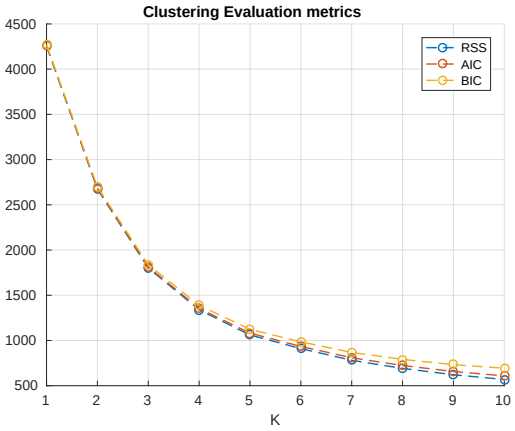

Figure 47: The RSS, AIC, and BIC curves for the second dataset for K-Means, for K range from 1 to 10. From this it is hard to identity the best K.

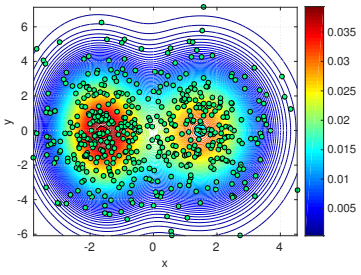

Figure 48: The contour plot of the GMM fit to the second 2-D dataset, `K=2`, `cov_type='full'`.

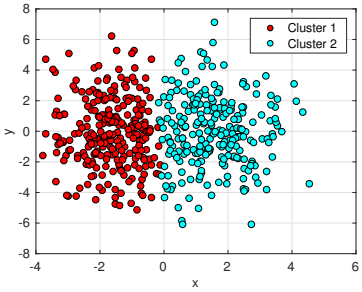

Figure 49: Output of hard clustering on the second 2-D dataset with GMM, `K=2`, `cov_type='full'`.

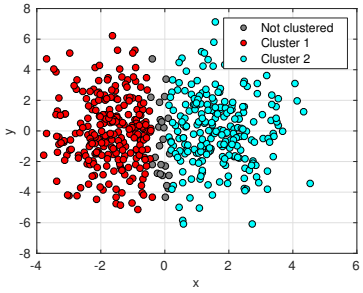

Figure 50: Output of soft clustering on the second 2-D dataset with GMM, `K=2`, `cov_type='full'`, and the soft thresholds set to [0.3, 0.7]. Here the points in gray do not have confidence to be clustered completely.

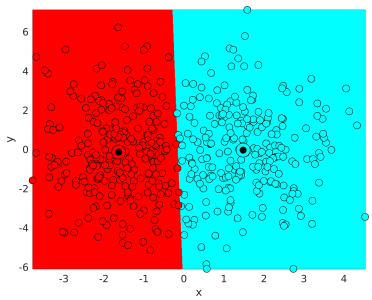

Figure 51: The clustering boundary of hard clustering on the second 2-D dataset with GMM, `K=2`, `cov_type='full'`. It can be seen that GMM clusters the datapoints properly, even though the datapoints are mixed.

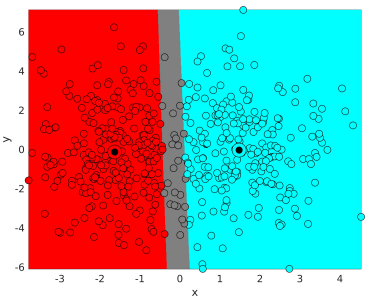

Figure 52: The clustering boundary for the GMM trained using the second 2-D dataset with `K=2`, `cov_type='full'`, and the soft thresholds set to [0.3, 0.7]. The region in gray is where the confidence to cluster is low.

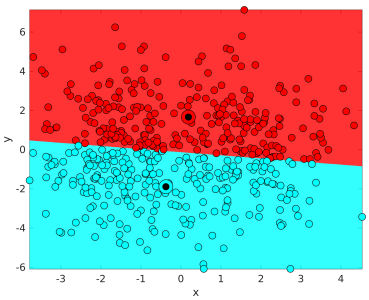

Figure 53: The cluster boundaries for the K-Means on the second 2-D dataset with `K=2`. Here it can be seen that the algorithm fails to recognize the clusters correctly since the points of the two clusters are very close to eachother.

3 Part 3: Classification with GMM+Bayes

In the previous part, we used GMM to cluster datapoints without knowing their labels. In this part, we want to decide to which class each datapoint belongs. To perform classification, we need to create a Train dataset (i.e. a labeled dataset to build our model) and a Test dataset to evaluate the performances of the algorithm. To perform classification with GMM, the first step is to create a GMM based on the Train datapoints to model each of the classes. This will result in a GMM per class. We will then implement a Gaussian Likelihood Discriminant Rule to classify the Test datapoints.

3.1 Learning a GMM for each class

Load the `2D Concentric Circles Dataset` by running the **first block** of code in the function `test_gmm_classif_2D.m`. To split the data in Train and Test datasets, we will use the function `split_data.m` from assignment 3 - KNN. For each class, we need to learn a GMM.

TASK 8: Implement my_gmm_models.m function
Using the previously developed function `my_gmmEM.m`, learn a GMM for each of the classes on the Train dataset.

```
function [models] = my_gmm_models(X_train, y_train, K, cov_type, plot_iter)
%MY_GMM_MODELS Computes maximum likelihood estimate of the parameters for the
% given GMM using the EM algorithm and initial parameters for each class of
% the dataset X_train
%   input----------------------------------------------------------------
%
%       o X_train   : (N x M_train), a data set with M_train samples each ...
    being of
%                           dimension N, each column corresponds to a datapoint.
%       o y_train   : (1 x M_train), a vector with labels y corresponding to ...
    X_train.
%       o K         : (1 x 1) number K of GMM components.
%       o cov_type  : string ,{'full', 'diag', 'iso'} type of Covariance matrix
%       o plot_iter : (bool)  set to 1 of want to visalize initual Mu's and
%                         Sigma's, works only for N=2
%
%   output ----------------------------------------------------------------
%       o models    :  (1 x N_classes) struct array with fields:
%                   | o Priors : (1 x K), the set of priors (or mixing
%                   |            weights) for each k-th Gaussian component
%                   | o Mu     : (N x K), an NxK matrix corresponding to
%                   |            the centroids mu = {mu^1,...mu^K}
%                   | o Sigma  : (N x N x K), an NxNxK matrix
%                   |            corresponding to the Covariance matrices
%                   |            Sigma = {Sigma^1,...,Sigma^K}
%
%%
```

Implementation Hint: `Useful functions struct().`

Test Implementation
By running block 2 of `test_gmm_classif_2D.m`, you should obtain a figure similar to Figure 54 and Figure 55.

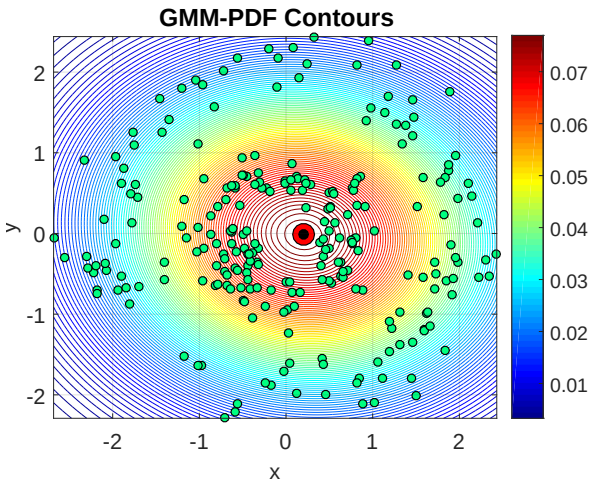

Figure 54: PDF of GMM parameters `K=1` and `cov_type='full'` for **class 1**.

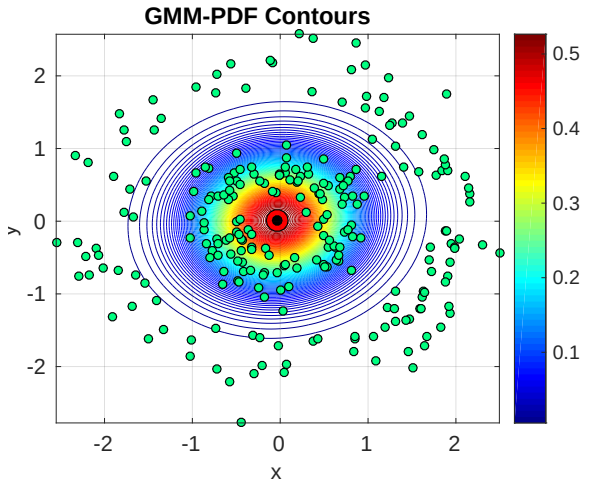

Figure 55: PDF of GMM parameters `K=1` and `cov_type='full'` for **class 2**

3.2 Gaussian Maximum Likelihood Discriminant Rule

A maximum likelihood classifier chooses the class label that is the most likely. For a binary classification problem, a new datapoint $\mathbf{x}'$ belongs to class 1 if the corresponding likelihood is superior to the likelihood of class 2,

$$p(y = 1|\mathbf{x}') > p(y = 2|\mathbf{x}'). \tag{19}$$

By Bayes, we have

$$p(y = i|\mathbf{x}') = \frac{p(\mathbf{x}'|y = i)p(y = i)}{p(\mathbf{x}')}, \text{with class } i = 1, 2. \tag{20}$$

Replacing this term in the Maximum Likelihood (ML) Discriminant Rule for $\mathbf{x}'$ in class 1 (equation 19), we obtain

$$p(\mathbf{x}'|y = 1)p(y = 1) > p(\mathbf{x}'|y = 2)p(y = 2). \tag{21}$$

In general, one can assume equal class distribution, i.e. $p(y = 1) = p(y = 2)$. By replacing in the previous equation, the discriminant rule becomes

$$p(\mathbf{x}'|y = 1) > p(\mathbf{x}'|y = 2). \tag{22}$$

For a GMM composed of K^i multivariate Gaussian functions, the conditional densities to belong to class i is similar to Equation 1,

$$p(\mathbf{x}'|y = i) = \sum_{k=1}^{K^i} \alpha_k p(\mathbf{x}'|\mu^k, \mathbf{\Sigma}^k), \tag{23}$$

with $p(\mathbf{x}'|\mu^k, \mathbf{\Sigma}^k)$ defined according to Equation 2.

In the case of a multi-class problem with $i = 1...I$ classes, the ML Discriminant Rule is the minimum of the log-likelihood (i.e., equivalent to maximizing the likelihood). **Assuming equal class distribution**, the ML Discriminant Rule is:

$$c_i(\mathbf{x}') = \underset{i}{\text{argmin}} \left\{ -\log(p(\mathbf{x}'|y = i)) \right\}. \tag{24}$$

In the case that one **does not make the assumption of equal class distribution**, according to Equation 22, the ML Discriminant Rule is:

$$c_i(\mathbf{x}') = \underset{i}{\operatorname{argmin}} \left\{ -\log(p(\mathbf{x}'|y=i)p(y=i)) \right\}. \tag{25}$$

TASK 9: Implement my_gmm_classif.m function
Implement ML Discriminant Rule to estimate labels `y_est` for each datapoints in `X_test` from Equation 24.

```
function [y_est] = my_gmm_classif(X_test, models, labels, K, P_class)
%MY_GMM_CLASSIF Classifies datapoints of X_test using ML Discriminant Rule
%   input----------------------------------------------------------------
%       o X_test    : (N x M_test), a data set with M_test samples each
%                     being of dimension N, each column corresponds to a
%                     datapoint.
%       o models    : (1 x N_classes) struct array with fields:
%                   | o Priors : (1 x K), the set of priors (or mixing
%                   |            weights) for each k-th Gaussian component
%                   | o Mu     : (N x K), an NxK matrix corresponding to
%                   |            the centroids mu = {mu^1,...mu^K}
%                   | o Sigma  : (N x N x K), an NxNxK matrix
%                   |            corresponding to the Covariance matrices
%                   |            Sigma = {Sigma^1,...,Sigma^K}
%
%       o labels    : (1 x N_classes) unique labels of X_test.
%       o K         : (1 x 1) number K of GMM components.
%   output ---------------------------------------------------------------
%       o y_est   :  (1 x M_test), a vector with estimated labels y
%                    \in {0,...,N_classes} corresponding to X_test.
%%
```

`Implementation Hint:` `Useful functions my_gaussPDF().`

Test Implementation

By running **block 3** of `test_gmm_classif_2D.m`, you should obtain Figure 56 for $K = 1$, Figure 57 for $K = 5$ and Figure 58 for $K = 10$.

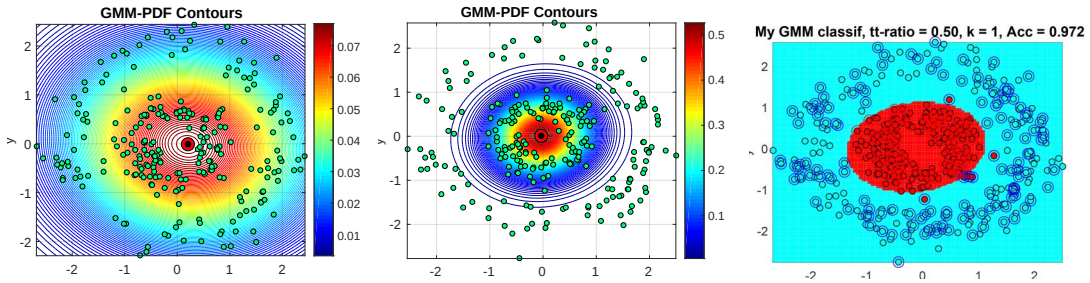

Figure 56: Estimated PDF and classification boundary for a model with `K=1`.

3.3 Unbalanced Classes without Equality of Class Distribution Assumption

Here, we will study the influence of unbalanced number of datapoints in each class without assuming an equality of class distribution. For visualization purpose, we will use a dataset composed of a Gaussian distribution per class with equal variances, which guaranties a linear boundary between classes. We also assume that both `Mu` have a 0 y-coordinate to ensure a vertical boundary line. These assumptions are already coded in `test_gmm_classif_2D.m`. Our

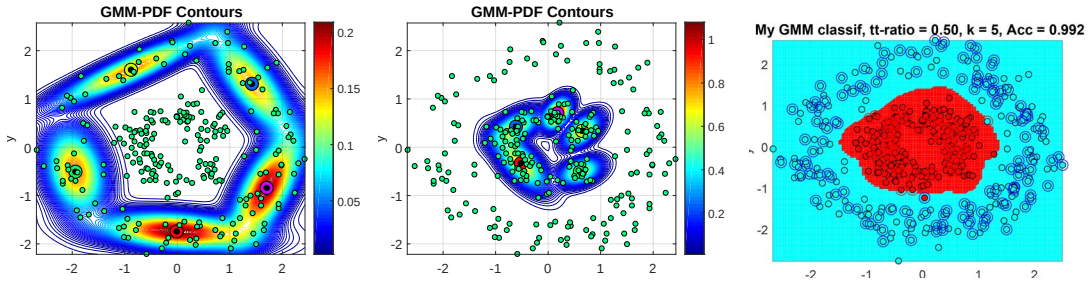

Figure 57: Estimated PDF and classification boundary for a model with `K=5`.

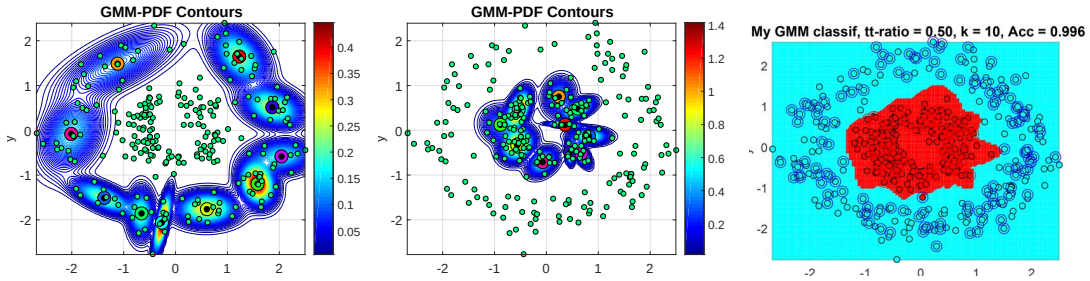

Figure 58: Estimated PDF and classification boundary for a model with `K=10`.

models will be learned with $K = 1$.

TASK 10: Modify my_gmm_classif.m function

Implement ML Discriminant Rule **without equal distribution of classes assumption** to estimate labels `y_est` for each datapoints in `X_test` from Equation 25.

```
function [y_est] = my_gmm_classif(X_test, models, labels, K, P_class)
%MY_GMM_CLASSIF Classifies datapoints of X_test using ML Discriminant Rule
%   input--------------------------------------------------------------------
%       o X_test    : (N x M_test), a data set with M_test samples each
%                       being of dimension N, each column corresponds to a
%                       datapoint.
%       o models    : (1 x N_classes) struct array with fields:
%                   | o Priors : (1 x K), the set of priors (or mixing
%                   |            weights) for each k-th Gaussian component
%                   | o Mu     : (N x K), an NxK matrix corresponding to
%                   |            the centroids mu = {mu^1,...mu^K}
%                   | o Sigma  : (N x N x K), an NxNxK matrix
%                   |            corresponding to the Covariance matrices
%                   |            Sigma = {Sigma^1,...,Sigma^K}
%       o labels    : (1 x N_classes) unique labels of X_test.
%       o K         : (1 x 1) number K of GMM components.
%   optional-----------------------------------------------------------------
%       o P_class   : (1 x N_classes), the vector of prior probabilities
%                      for each class i, p(y=i). If provided, equal class
%                      distribution assumption is no longer made.
%   output ------------------------------------------------------------------
%       o y_est  :  (1 x M_test), a vector with estimated labels y
%                   \in {0,...,N_classes} corresponding to X_test.
%%
```

Implementation Hint: Useful functions `my_gaussPDF()`, `nargin`.

Test Implementation

By running **block 4** and **block 5** of `test_gmm_classif_2D.m`, you should obtain a figure similar to the following one:

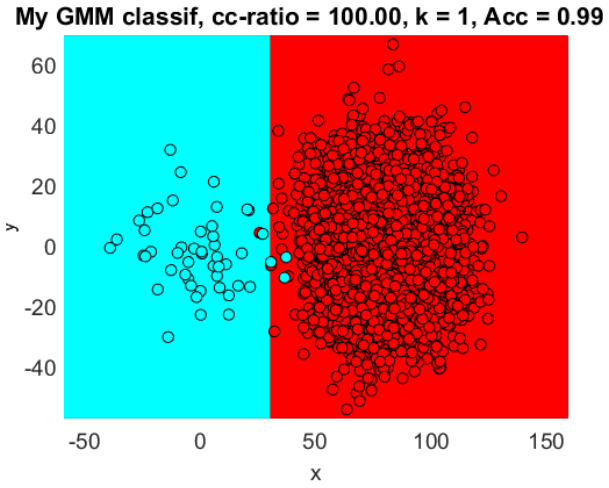

Figure 59: Classification boundary for a ratio of datapoints between classes `'cc-ratio'=100`.

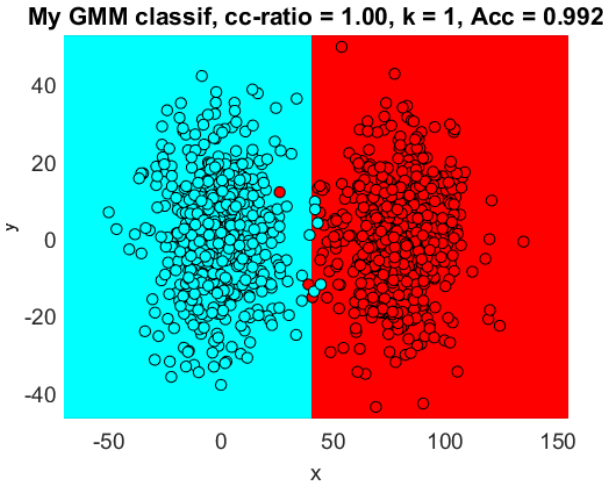

Figure 60: Classification boundary for a ratio of datapoints between classes `'cc-ratio'=1`.

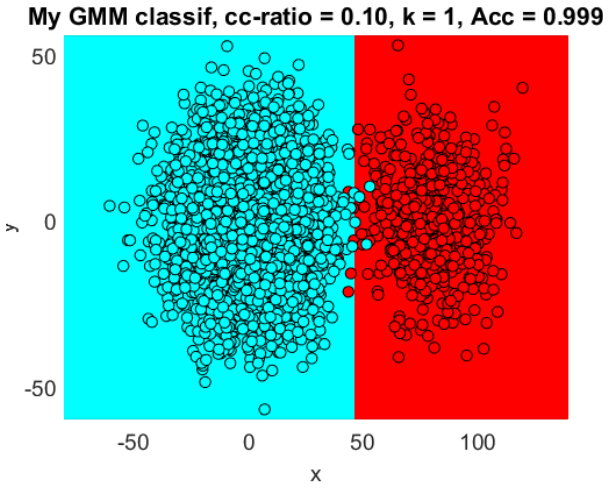

Figure 61: Classification boundary for a ratio of datapoints between classes `'cc-ratio'=0.1`.

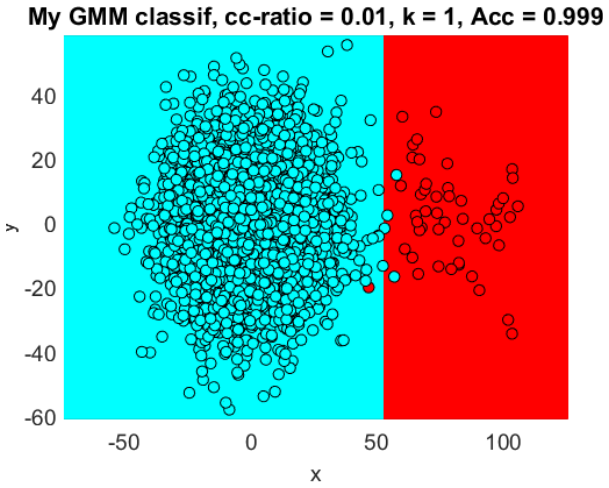

Figure 62: Classification boundary for a ratio of datapoints between classes `'cc-ratio'=0.01`.