Machine Learning Programming
Assignment 4: Gaussian Mixture Models

Professors: Baptiste Busch, Aude Billard
Assistants: Laila El Hamamsy, Matthieu Dujany
and Victor Faraut
Contacts:
baptiste.busch@epfl.ch, aude.billard@epfl.ch
laila.elhamamsy@epfl.ch, matthieu.dujany@epfl.ch, victor.faraut@epfl.ch

Winter Semester 2018

Introduction

In this practical, you will code the Expectation-Maximization algorithm as well as model fitting procedures for learning Gaussian Mixture Models (GMM) for several 2D datasets.

Submission Instructions

Deadline: November 27, 2018 @ 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
my_gaussPDF.m	gmm_metrics
my_gmmLogLik.m	gmm_eval
my_covariance.m	--
my_gmmInit.m	--
expectation_step.m	--
maximization_step.m	--
my_gmmEM.m	--
test_gmm_em.m	test_gmm_fit.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. This can be done as follows in the MATLAB command window:

```
>> addpath(genpath('path_to_ML_toolbox'))
```

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. Your submission archive should contain ONLY the following:

TP4-GMM-Assignment-SCIPER
- Part1
- Part2

DO NOT upload ML_toolbox, check_utils_encr, utils or the Datasets directory.

Note: Due to the stochasticity of the initialization of the clusters, your results and graphs in this assignment might appear slightly different.

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 (we will cover these applications in TP5-GMM-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 priors $\alpha = \{\alpha_1, \ldots, \alpha_K\}$, means $\mu = \{\mu^1, \ldots, \mu^K\}$ and Covariance matrices $\mathbf{\Sigma} = \{\mathbf{\Sigma}^1, \ldots, \mathbf{\Sigma}^K\}$.
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.

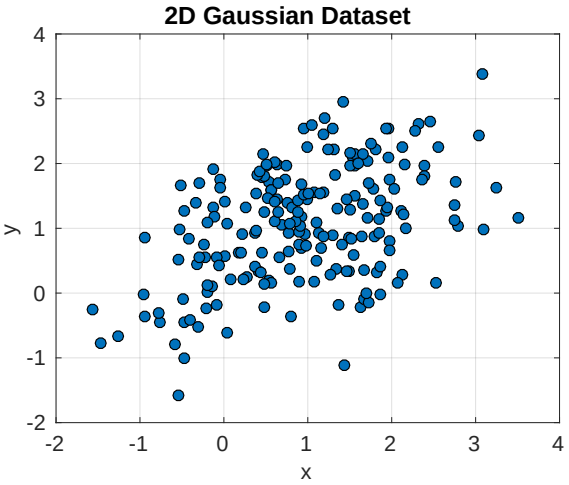

Figure 1: 2D Gaussian Dataset. Data points were sampled from $\mathcal{N}([1;1],[1,0.5;0.5,1])$.

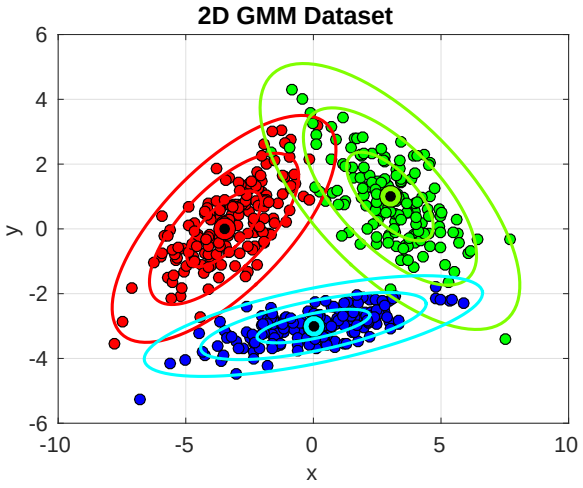

Figure 2: 2D GMM Dataset. Data points were sampled from a $K = 3$ GMM.

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` (2pts): 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` (2pts): The **log likelihood** of the parameters Θ of a learnt GMM for the given dataset $\mathbf{X}$; i.e. the log of Equation 3.

TASK 1: Implement my_gausspdf.m function (2pts)

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 encrypted `test_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.

```
--- Testing 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 (2pts)

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 [logl] =  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 logl       : (1 x 1) , loglikelihood
```

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

Test Implementation

To evaluate this function you can load the dataset from Figure1 or 2, by running code sub-block `(1a)` or `(1a)`, respectively. By running the **third** code block, the encrypted `test_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.

```
--- Testing 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 (3pts)
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 load any dataset with sub-blocks `(1a-b)`. By running the **fourth** code block, the encrypted `test_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.
```

Within this code-block we will also plot 3 figures corresponding to the contours of the first 3 standard deviations of the computed Covariance matrices, for dataset `(1a)` you should see Figure 3, 4 and 5.
Moreover for dataset `(1a)`, 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 $\alpha^{(0)}$, Centroids $\mu^{(0)}$ and Covariance Matrices $\mathbf{\Sigma}^{(0)}$

The EM algorithm for GMM begins with an initialization step. Here we shall initialize for iteration $t = 0$ the following:

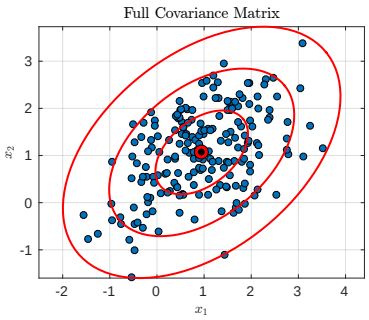

Figure 3: Full Covariance.

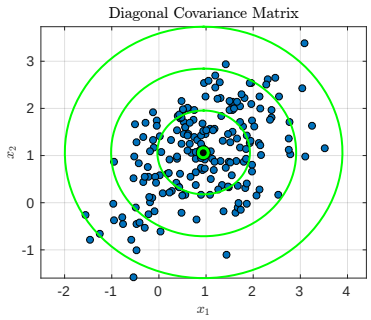

Figure 4: Diagonal Covariance.

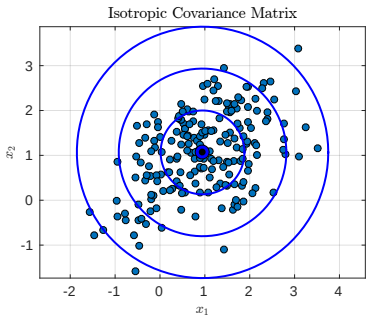

Figure 5: Isotropic Covariance.

- The set of priors $\alpha^{(0)} = \{\alpha_1^{(0)}, \ldots, \alpha_K^{(0)}\}$ to uniform probabilities; i.e. $\alpha_1^{(0)} = \cdots = \alpha_K^{(0)} = \frac{1}{K}$
- The means $\mu^{(0)} = \{\mu^{1(0)}, \ldots, \mu^{K(0)}\}$ shall be initialized with the K-Means algorithm.
- Given the means $\mu^{(0)}$ and labels computed from the previous step, one can initialize the set of corresponding Covariance matrices $\mathbf{\Sigma}^{(0)} = \{\mathbf{\Sigma}^{1(0)}, \ldots, \mathbf{\Sigma}^{K(0)}\}$ choosing the datapoints assigned to each $\mu^{k(0)}$ and using your `my_covariance.m` function.

TASK 4: Implement my_gmmInit.m function (2pts)
Now you will implement `my_gmmInit.m` function with input $\mathbf{X}$, and *params*, a Matlab structure that contains all the hyperparameters for the k-means initialization, i.e. k number of clusters, cov_type the type of covariance matrix among `full`, `iso`, and `diag`, d_type the metric function and max_iter_init the maximum number of iterations for the initialization. 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 [Priors0, Mu0, Sigma0, labels0] =  my_gmmInit(X, params)
%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 params : Structure containing the paramaters of the algorithm:
%           * cov_type: Type of the covariance matric among 'full', 'iso',
%           'diag'
%           * k: Number of clusters for the k-means initialization
%           * d_type: Distance metric for the k-means initialization
%           * init: Type of initialization for the k-means
%           * max_iter_init: Max number of iterations for the k-means
%   output ---------------------------------------------------------------
%       o Priors0   : (1 x K), the set of priors (or mixing weights) for each
%                           k-th Gaussian component
%       o Mu0       : (N x K), an NxK matrix corresponding to the centroids
%                           mu = {mu^1,...mu^K}
%       o Sigma0    : (N x N x K), an NxNxK matrix corresponding to the
%                         Covariance matrices  Sigma = {Sigma^1,...,Sigma^K}
%       o labels0   : (1 x M), a vector of labels \in {1,..,k}
%                           corresponding to the k-th Gaussian component
```

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

Test Implementation

To evaluate this function you load any dataset with sub-blocks `(1a-b)`. By running the **fifth** code block, the encrypted `test_mygmmInit.p` function will evaluate your `my_gmmInit.m` function with different values of K and `cov_type`. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Testing my_gmmInit.m with K=10 and cov_type = full---
[Test 1] Checking priors computation: Correct.
[Test 2] Checking Mu computation: Correct.
[Test 3] Checking Sigma computation: Correct.
--- Testing my_gmmInit.m with K=5 and cov_type = iso---
[Test 1] Checking priors computation: Correct.
[Test 2] Checking Mu computation: Correct.
[Test 3] Checking Sigma computation: Correct.
--- Testing my_gmmInit.m with K=8 and cov_type = diag---
[Test 1] Checking priors computation: Correct.
[Test 2] Checking Mu computation: Correct.
[Test 3] Checking Sigma computation: Correct.
```

Within the same code-block you can visually evaluate your initializations. For dataset `(1b)`, by modifying `K` and `cov_type` you can visualize the different initializations as in Figure 7-14.

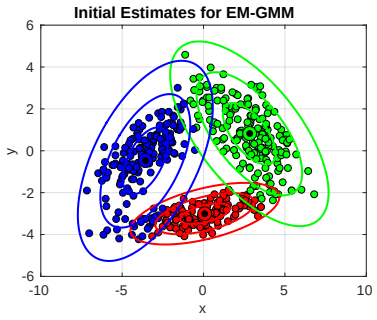

Figure 6: Initial parameters `K=3` and `cov_type='full'`

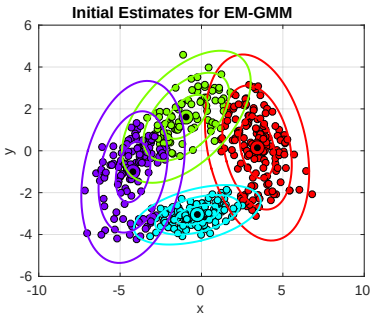

Figure 7: Initial parameters `K=4` and `cov_type='full'`

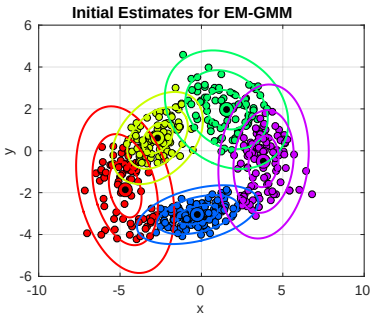

Figure 8: Initial parameters `K=5` and `cov_type='full'`

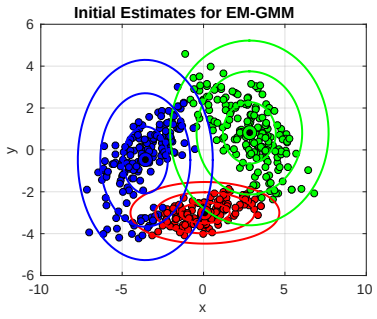

Figure 9: Initial parameters `K=3` and `cov_type='diag'`

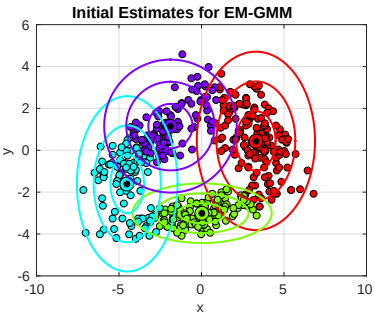

Figure 10: Initial parameters `K=4` and `cov_type='diag'`

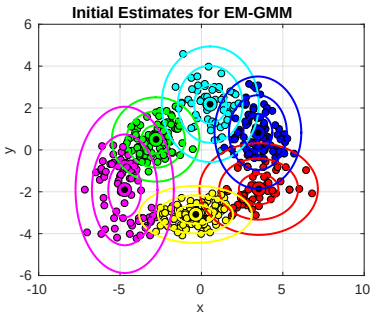

Figure 11: Initial parameters `K=6` and `cov_type='diag'`

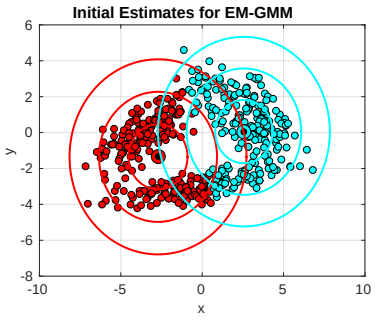

Figure 12: Initial parameters K=2 and cov_type='iso'

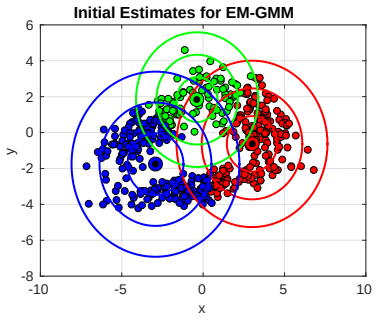

Figure 13: Initial parameters K=3 and cov_type='iso'

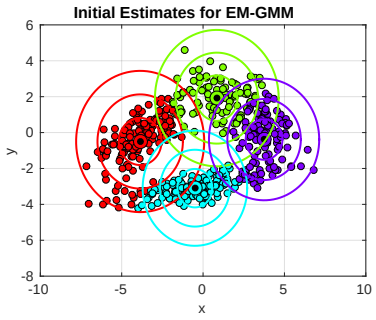

Figure 14: Initial parameters K=4 and cov_type='iso'

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 data points $i \in \{1, \ldots, M\}$.

TASK 5.1: Implement expectation_step.m function (2pts)

You will implement the expectation step in a separate function in `expectation_step.m`. This function takes as input $\mathbf{X}$, the current priors $\alpha = \{\alpha_1, \ldots, \alpha_K\}$, centroids $\mu = \{\mu^1, \ldots, \mu^K\}$, and covariance matrices $\mathbf{\Sigma} = \{\mathbf{\Sigma}^1, \ldots, \mathbf{\Sigma}^K\}$, and the hyper-parameter structure `params`. It outputs the posterior probability $p(k|\mathbf{x}^i, \Theta^{(t)})$ of each Gaussian k.

```
function [Pk_x] = expectation_step(X, K, Mu, Sigma)
%EXPECTATION_STEP Computes the expection step of the EM algorihtm
%   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 updated priors (or mixing weights) ...
    for each
%                           k-th Gaussian component
%       o Mu        : (N x K), an NxK matrix corresponding to the CURRENT
%                     centroids mu^(0) = {mu^1,...mu^K}
%       o Sigma     : (N x N x K), an NxNxK matrix corresponding to the CURRENT
%                     Covariance matrices Sigma^(0) = {Sigma^1,...,Sigma^K}
%       o params    : The hyperparameters structure that contains k, the ...
    number of Gaussians
% output---------------------------------------------------------------------
%       o Pk_x      : (K, M) a KxM matrix containing the posterior ...
    probabilty that a
%                     k Gaussian is responsible for generating a point m in ...
    the dataset
```

Implementation Hint: Use your implementations of my_gausspdf.m, repmat()

Step 3. Maximization Step: Update Priors α, Means μ and Covariances Σ

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 $\mathbf{\Sigma}$'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 $\mathbf{\Sigma}^k$.

TASK 5.2: Implement maximization_step.m function (2pts)
You will implement the maximization step in a separate function in `maximization_step.m`. This function takes as input $\mathbf{X}$, previously calculated posterior probability $p(k|\mathbf{x}^i, \Theta^{(t)})$ and the hyper-parameter structure *params*. It outputs the updated priors $\alpha = \{\alpha_1, \ldots, \alpha_K\}$, centroids $\mu = \{\mu^1, \ldots, \mu^K\}$, and covariance matrices $\mathbf{\Sigma} = \{\mathbf{\Sigma}^1, \ldots, \mathbf{\Sigma}^K\}$.

```
function [Priors,Mu,Sigma] = maximization_step(X, Pk_x, params)
%MAXIMISATION_STEP Compute the maximization step of the EM algorithm
%   input----------------------------------------------------------------
%       o X         : (N x M), a data set with M samples each being of
%       o Pk_x      : (K, M) a KxM matrix containing the posterior probabilty
%                     that a k Gaussian is responsible for generating a point
%                     m in the dataset, output of the expectation step
%       o params    : The hyperparameters structure that contains k, the ...
    number of Gaussians
%                     and cov_type the coviariance type
%   output ----------------------------------------------------------------
%       o Priors    : (1 x K), the set of updated priors (or mixing weights) ...
    for each
%                           k-th Gaussian component
%       o Mu        : (N x K), an NxK matrix corresponding to the updated ...
    centroids
%                           mu = {mu^1,...mu^K}
%       o Sigma     : (N x N x K), an NxNxK matrix corresponding to the
%                     updated Covariance matrices  Sigma = {Sigma^1,...,Sigma^K}
```

Implementation Hint: DO NOT use my_covariance.m for the maximization step of the Covariance matrices, you should implement Equations 11 and 12, repmat(), bsxfun()

TASK 5.3: Implement my_gmmEM.m function (2pts)

Your task is now to implement a function that iterates over Step 2 (E-Step) and Step 3 (M-Step) until the log likelihood of your current parameters Θ (Eq. 5) is stabilized, given some initial $\Theta^{(0)}$, $\alpha^{(0)}$, $\mu^{(0)}$, and $\mathbf{\Sigma}^{(0)}$, computed with your `my_gmmInit.m` function. This iterative loop should be implemented in `my_gmmEM.m` function with inputs: $\mathbf{X}$, and the hyper-parameters structure `params`. The outputs of `my_gmmEM.m` should be the final priors $\alpha = \{\alpha_1, \ldots, \alpha_K\}$, centroids $\mu = \{\mu^1, \ldots, \mu^K\}$, Covariance matrices $\mathbf{\Sigma} = \{\mathbf{\Sigma}^1, \ldots, \mathbf{\Sigma}^K\}$ and `iter`, which is the number of iterations your implementation took to converge to the given solution.

```
function [  Priors, Mu, Sigma, iter ] = my_gmmEM(X, params)
%MY_GMMEM Computes maximum likelihood estimate of the parameters for the
% given GMM using the EM algorithm and initial parameters
%   input----------------------------------------------------------------
%       o X         : (N x M), a data set with M samples each being of
%                           dimension N, each column corresponds to a datapoint.
%       o params : Structure containing the hyperparamaters of the algorithm:
%           * cov_type: Type of the covariance matric among 'full', 'iso',
%           'diag'
%           * k: Number of gaussians
%           * max_iter: Max number of iterations
%           * d_type: Distance metric for the k-means initialization
%           * init: Type of initialization for the k-means
%           * max_iter_init: Max number of iterations for the k-means
% output----------------------------------------------------------------
%       o Priors    : (1 x K), the set of FINAL priors (or mixing weights)
%                     for each k-th Gaussian component
%       o Mu        : (N x K), an NxK matrix corresponding to the FINAL
%                     centroids mu = {mu^1,...mu^K}
%       o Sigma     : (N x N x K), an NxNxK matrix corresponding to the
%                     FINAL Covariance matrices  Sigma = {Sigma^1,...,Sigma^K}
%       o iter      : (1 x 1) number of iterations it took to converge
```

Implementation Hint: Use your implementations of expectation_step.m, maximization_step.m, and my_gmmLogLik.m

Test Implementation

To evaluate this function you should initially load the `(1b)` dataset in 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 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 18-23).

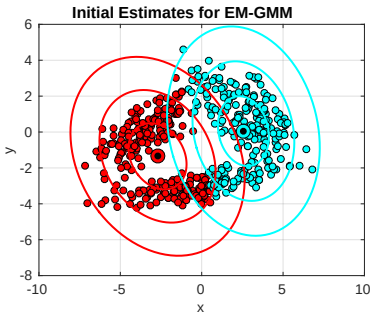

Figure 15: Initial parameters `K=2` and `cov_type='full'`

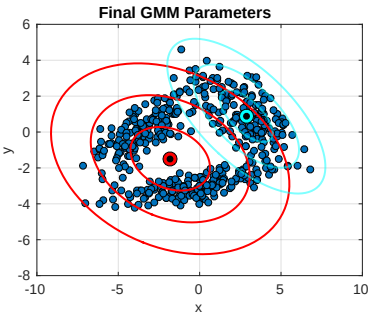

Figure 16: Final parameters `K=2` and `cov_type='full'`

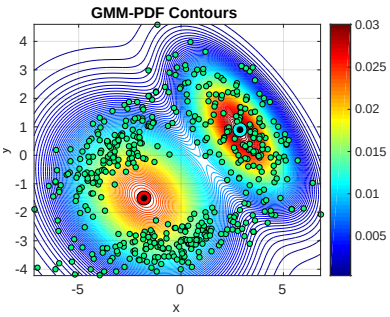

Figure 17: PDF of GMM parameters `K=2` and `cov_type='full'`

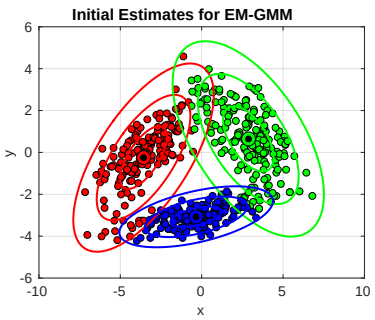

Figure 18: Initial parameters `K=3` and `cov_type='full'`

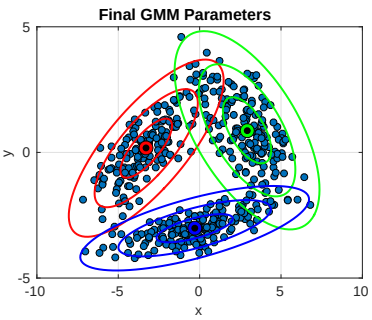

Figure 19: Final parameters `K=3` and `cov_type='full'`

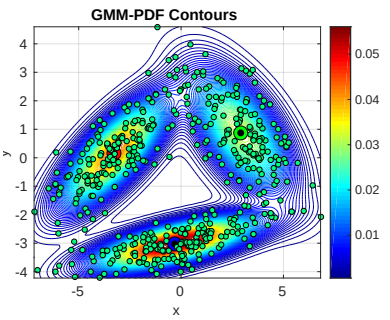

Figure 20: PDF of GMM parameters `K=3` and `cov_type='full'`

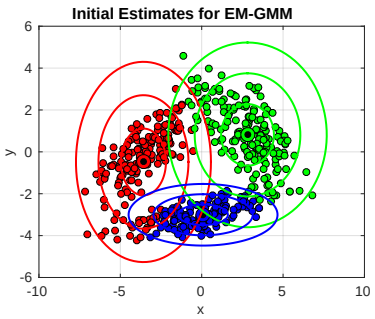

Figure 21: Initial parameters `K=3` and `cov_type='diag'`

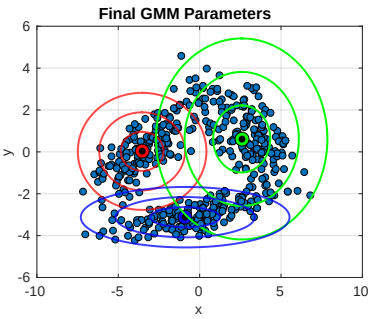

Figure 22: Final parameters `K=3` and `cov_type='diag'`

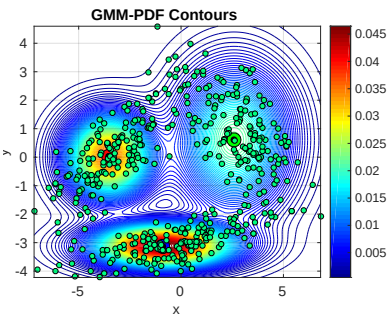

Figure 23: PDF of GMM parameters `K=3` and `cov_type='diag'`

Further, you can test your GMM-EM algorithm on your own 2D Datasets, which you can draw with the `ml_generate_mouse_data.m` function from ML_toolbox. By loading the `(1c)` dataset you can load the drawing GUI shown in Figure 24. After drawing your data, you should click on the `Store Data` button, this will store a data array in your MATLAB workspace. The data and labels will be stored in two different arrays: $\mathbf{X} \in \mathbb{R}^{2\times M}$ and `labels` $\in \mathbb{R}^{1\times M}$, which you can then use to visualize and manipulate in MATLAB (Figure 25).

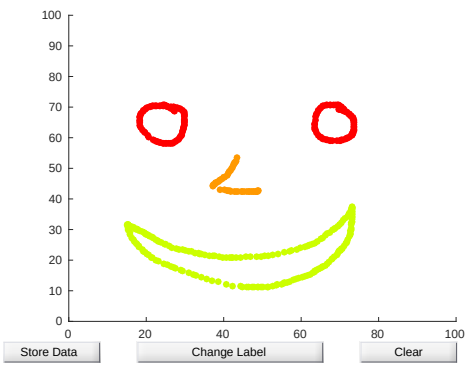

Figure 24: ML_toolbox 2D Data Drawing GUI.

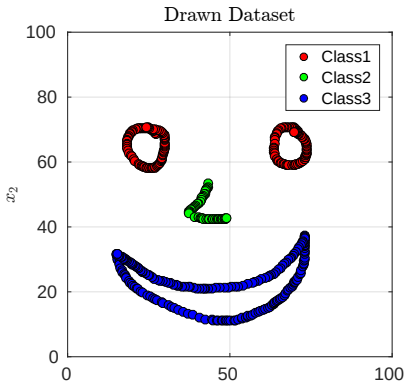

Figure 25: Data stored in MATLAB workspace.

After drawing your own dataset, you can test the **sixth** code block with different parameters for K and `cov_type` and learn a GMM for it, which you can then visualize as in Figure 26, 27 and 28.

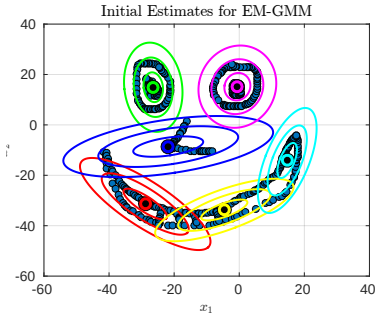

Figure 26: Initial parameters `K=6` and `cov_type='full'`

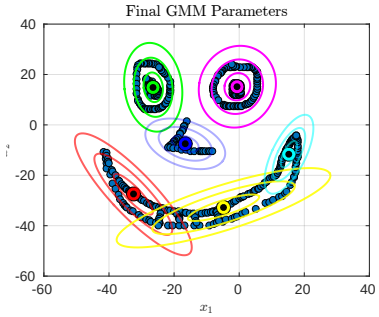

Figure 27: Final parameters `K=6` and `cov_type='full'`

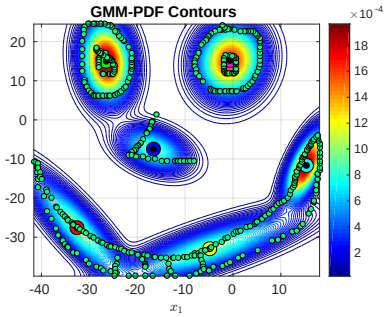

Figure 28: PDF of GMM parameters `K=6` and `cov_type='full'`

To numerically evaluate this function you can load any dataset `(1a-b-c)`. By running the **sixth** code block, the encrypted `test_mygmmEM.p` function will evaluate your `expectation_step.m`, `maximization_step.m`, and `my_gmmEM.m` function with different values of K and `cov_type`. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Testing my_gmmEM.m with K=10 and cov_type = full---
[Test 1] Checking priors computation: Correct.
[Test 2] Checking Mu computation: Correct.
[Test 3] Checking Sigma computation: Correct.
--- Testing my_gmmEM.m with K=10 and cov_type = iso---
[Test 1] Checking priors computation: Correct.
[Test 2] Checking Mu computation: Correct.
[Test 3] Checking Sigma computation: Correct.
--- Testing my_gmmEM.m with K=10 and cov_type = diag---
[Test 1] Checking priors computation: Correct.
[Test 2] Checking Mu computation: Correct.
[Test 3] Checking Sigma computation: Correct.
```

2 Part 2: 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

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

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 6: Implement gmm_metrics.m function (3pts)

```
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 encrypted `test_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.

```
--- Testing gmm_metrics.m ---
[Test 1] Checking AIC/BIC for K=3  with full Covariance: Correct.
[Test 2] Checking AIC/BIC for K=3  with diag Covariance: Correct.
[Test 3] Checking AIC/BIC for K=3  with iso Covariance: Correct.
[Test 4] Checking AIC/BIC for K=1  with full Covariance: Correct.
[Test 5] Checking AIC/BIC for K=1  with diag Covariance: Correct.
[Test 6] Checking AIC/BIC for K=1  with iso Covariance: 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; in other words, we choose the **minimum** AIC/BIC value from the 10 runs. We then plot the values and select the K which yields the best tradeoff between *likelihood* and *model complexity*.

TASK 7: Implement gmm_eval.m function (2pts)

```
function [AIC_curve, BIC_curve] =  gmm_eval(X, K_range, repeats, params)
%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 K_range   : (1 X K), Range of k-values to evaluate
%       o repeats   : (1 X 1), # times to repeat k-means
%       o params : Structure containing the paramaters of the algorithm:
%           * cov_type: Type of the covariance matric among 'full', 'iso',
%           'diag'
%           * d_type: Distance metric for the k-means initialization
%           * init: Type of initialization for the k-means
%           * max_iter_init: Max number of iterations for the k-means
%           * max_iter: Max number of iterations for EM algorithm
%
%   output -----------------------------------------------------------------
%       o AIC_curve  : (1 X K), vector of max AIC values for K-range
%       o BIC_curve  : (1 X K), vector of max BIC values for K-range
```

Test Implementation

By running the **third** code block, the encrypted `test_gmmeval.p` function will evaluate your `gmm_eval.m` function. This will evaluate your AIC and BIC curves for a range of K and different types of Covariance matrices. If you get the following messages on your MATLAB command

window, you have succesfully implemented this function.

```
--- Testing gmm_eval.m with cov_type=full ---
Evaluating students function...done
Evaluating optimal function...done
[Test 1] AIC Curve computation for cov_type=full is Correct.
[Test 2] BIC Curve computation for cov_type=full is Correct.
--- Testing gmm_eval.m with cov_type=diag ---
Evaluating students function...done
Evaluating optimal function...done
[Test 1] AIC Curve computation for cov_type=diag is Correct.
[Test 2] BIC Curve computation for cov_type=diag is Correct.
--- Testing gmm_eval.m with cov_type=iso ---
Evaluating students function...done
Evaluating optimal function...done
[Test 1] AIC Curve computation for cov_type=iso is Correct.
[Test 2] BIC Curve computation for cov_type=iso is Correct.
```

You can visually evaluate your function by modifying `cov_type` in the next code-block you should obtain plots similar to Figure 29, 30 and 31. By simply looking at Figure 29, 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 30 and 31, 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 a 3-Gaussian full Covariance GMM, if we were going to use this model for clustering, this would be the best choice (see Figure 32). 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 33 and 34).

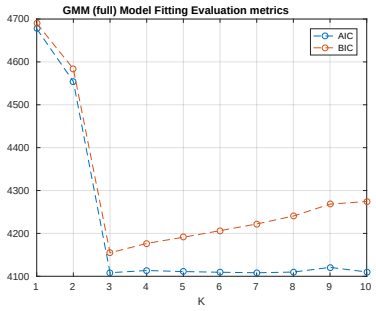

Figure 29: GMM Fitting metrics for `K_range=1:10` and `cov_type='full'`

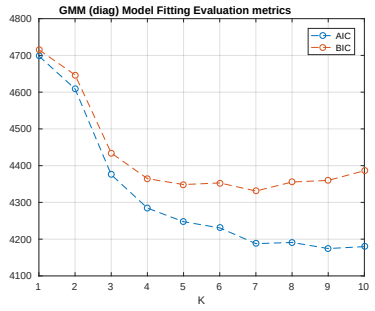

Figure 30: GMM Fitting metrics for `K_range=1:10` and `cov_type='diag'`

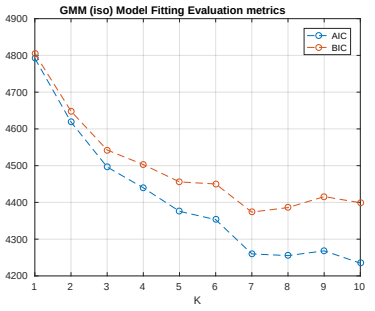

Figure 31: GMM Fitting metrics for `K_range=1:10` and `cov_type='iso'`

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 35 (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 36,37 and 38 there is no clear elbow or "optimal" point from the range of $K = 1 : 10$. The AIC metric always tends to the maximum 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 41 and 40. If we

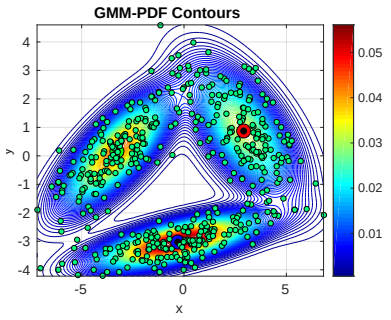

Figure 32: Optimal GMM `K=3`, `cov_type='full'`

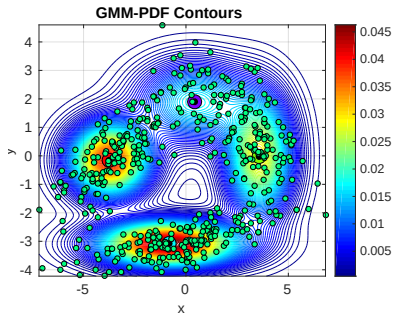

Figure 33: Optimal GMM `K=4`, `cov_type='diag'`

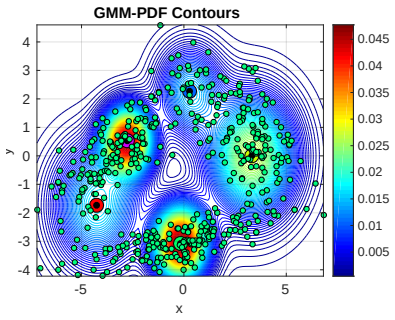

Figure 34: Optimal GMM `K=5`, `cov_type='iso'`

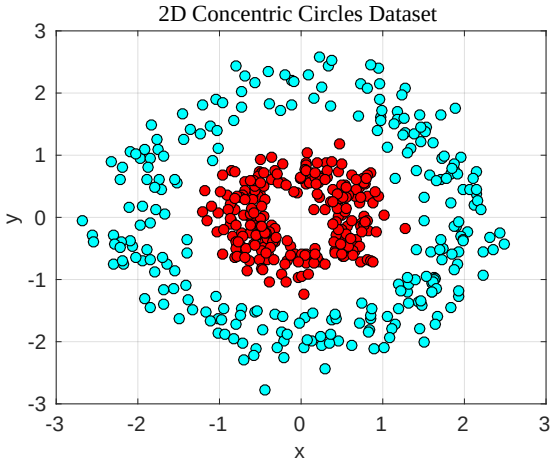

Figure 35: 2D Concentric Dataset

had solely done model selection with a `full` Covariance matrix, one might have chosen $K = 5$ or $K = 8$, as in Figure 39. 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.

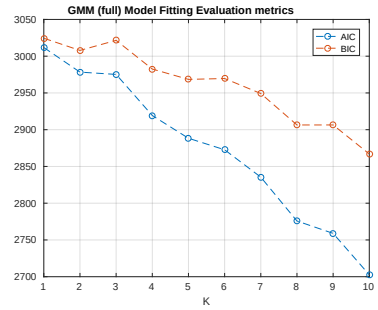

Figure 36: GMM Fitting metrics for `K_range=1:10` and `cov_type='full'`

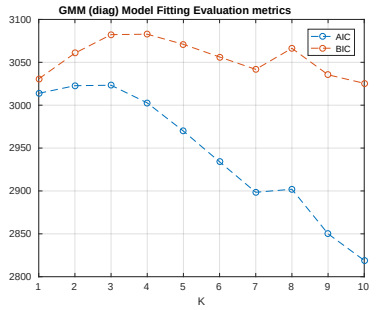

Figure 37: GMM Fitting metrics for `K_range=1:10` and `cov_type='diag'`

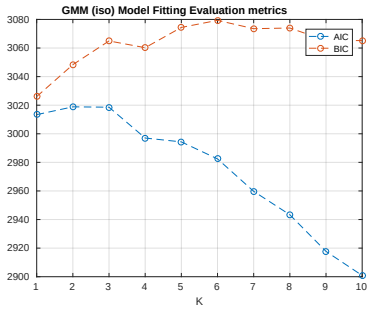

Figure 38: GMM Fitting metrics for `K_range=1:10` and `cov_type='iso'`

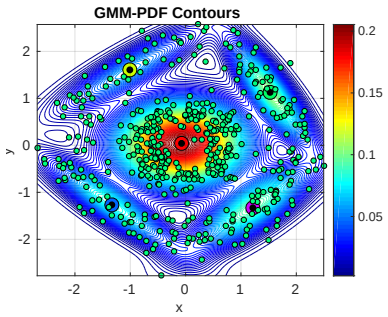

Figure 39: Optimal GMM `K=5`, `cov_type='full'`

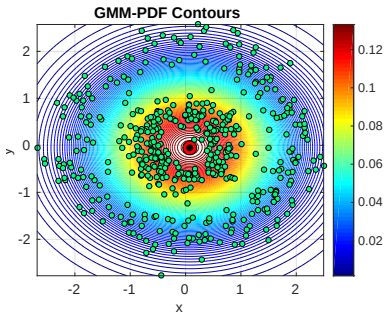

Figure 40: Optimal GMM `K=1`, `cov_type='diag'`

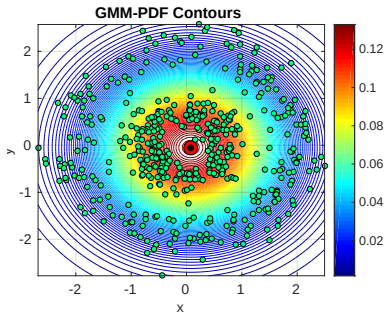

Figure 41: Optimal GMM `K=1`, `cov_type='iso'`