SHARIF UNIVERSITY OF TECHNOLOGY

INTRODUCTION TO MACHINE LEARNING

25737-2

Project Phase 1

By : Shabnam Fazliani 99106522
Mohammad Ghafourian 99106493

Professor: Dr.Amini

September 16, 2024

Theory Question 1

The MM algorithm, short for the Mixture Models, is a popular optimization method that can be used to solve a wide range of optimization problems. It is particularly useful when dealing with non-convex optimization objective functions.

The basic idea behind the MM algorithm is to reformulate a non-convex optimization problem as a sequence of simpler convex optimization sub-problems, which can be much more effortless to solve. This is achieved by introducing an additional variable, called the dual variable, and adding a penalty term to the original objective function. The penalty term is chosen in such a way that it forces the original objective function to satisfy certain constraints, which can be formulated as a set of convex functions.

To explain this in more detail, let's consider a simple non-convex optimization problem of the form:

$$\text{minimize} f(x)$$

where x is the optimization variable and $f(x)$ is a non-convex objective function. The goal of the MM algorithm is to transform this problem into a sequence of convex sub-problems of the form:

$$\text{minimize} \;\; f(x) + g(u)$$

$$\text{subject to} \;\; x - u = 0$$

where u is the dual variable and $g(u)$ is a convex function that penalizes deviations from the constraint $x - u = 0$.

The MM algorithm then proceeds by optimizing these sub-problems iteratively, alternating between updates of x and u. At each iteration, the value of x is updated by minimizing the current sub-problem with respect to x, while keeping u fixed. The value of u is then updated by applying a dual ascent step that moves u in the direction of the constraint $x - u = 0$.

So basically MM algorithm takes the following steps:

1. Start with an initial guess for the solution.
2. For each iteration, the algorithm updates the solution by optimizing an auxiliary function that approximates the original function at the current solution.
3. The new solution is then used to update the auxiliary function, and the process is repeated until convergence.

The key advantage of the MM algorithm is that the sub-problems are convex and can be efficiently solved using standard optimization techniques. Moreover, the MM algorithm has been shown to converge to a local optimum of the original non-convex optimization problem under certain conditions.

In summary, the MM algorithm can deal with non-convex optimization objective functions by reformulating them as a sequence of simpler convex subproblems, which can be easier to solve.One common way to do this is to use a Taylor series expansion to approximate the non-convex function around the current solution. This results in a series of convex functions that approximate the original function to different degrees of accuracy.

As the MM algorithm proceeds, the auxiliary functions become better and better approximations of the original non-convex function, and the algorithm converges to a solution that minimizes the original function.

By introducing an additional variable and penalty term, the original problem is transformed into a constrained convex optimization problem, which can be solved using standard optimization techniques.

Theory Question 2

Part 1

The formula for mixture models is given by:

$$p(y;\theta) = \sum_{k=1}^{K} p_Z(z_k;\theta)p_{Y|Z}(y \mid Z = z_k;\theta)$$

where $p(y;\theta)$ is the probability density function of the observed data y, $p_Z(z_k;\theta)$ is the prior probability of the k-th component, and $p_{Y|Z}(y|Z = z_k;\theta)$ is the conditional probability of the observed data given the k-th component.

Now, let us consider the log-likelihood function of the observed data, which is given by:

$$\ln(p_Y(y;\theta)) = \ln\left(\prod_{i=1}^{N} p(y^{(i)};\theta)\right) = \sum_{i=1}^{N} \ln(p(y^{(i)};\theta))$$

Substituting the formula for mixture models, yields

$$\ln(p_Y(y;\theta)) = \sum_{i=1}^{N} \ln(\sum_{k=1}^{K} p_Z(z_k;\theta)p_{Y|Z}(y^{(i)} \mid Z = z_k;\theta))$$

Notice that the inner sum in the above equation is over all possible values of the latent variable $Z^{(i)}$ for the i-th data point. Therefore, we can rewrite the equation as:

$$\ln(p_Y(y;\theta)) = \sum_{i=1}^{N} \ln(\sum_{k=1}^{K} p_{Y,Z}(y^{(i)}, Z^{(i)} = k;\theta))$$

where

$$p_{Y,Z}(y^{(i)}, Z^{(i)} = k;\theta) = p_Z(k;\theta)p_{Y|Z}(y^{(i)} \mid Z = k;\theta)$$

is the joint probability of the i-th data point and the k-th component.

Thus, we see that the formula for mixture models is the same as the sum over all possible values of $Z^{(i)}$ in the equation for the log-likelihood function. This shows that mixture models can be used to model complex probability distributions by assuming that the observed data is generated by a mixture of simpler subgroups or components, each with its own probability distribution.

Part 2

Mixture models are statistical models that assume that the observed data is generated by a mixture of several subgroups or components, each with its own probability distribution. The parameters of the mixture model include the mixing proportions (the probabilities of each component), and the parameters of each component distribution.

In the context of mixture models, it is often easier to optimize the joint distribution of the observed data and the latent variables (e.g., the component indicators or cluster assignments) than to optimize the distribution of the observed data alone. This is because the joint distribution of the observed data and the latent variables naturally decomposes into a product of conditional distributions, each of which is typically easier to optimize than the joint distribution.

Specifically, let us consider a mixture model with two subgroups or components, each with its own probability distribution, and let Y denote the observed data and Z denote the latent variables (i.e., the component indicators or cluster assignments). The joint distribution of the observed data and the latent variables is given by:

$$p_{Y,Z}(y_n, z_n; \theta) = p_Z(z_n; \theta) \cdot p_{Y|Z}(y_n|z_n; \theta)$$

where $p_Z(z_n; \theta)$ is the prior probability of the latent variable z_n, and $p_{Y|Z}(y_n|z_n; \theta)$ is the conditional probability of the observed data given the latent variable.

Now, optimizing the joint distribution $p_{Y,Z}(y_n, z_n; \theta)$ involves estimating the parameters of both the prior probabilities $p_Z(z_n; \theta)$ and the conditional probabilities $p_{Y|Z}(y_n|z_n; \theta)$. However, since the joint distribution decomposes into a product of conditional distributions, we can optimize each of these conditional distributions separately.

Specifically, we can optimize the prior probabilities $p_Z(z_n;\theta)$ by using a simple maximum likelihood estimate based on the observed frequencies of the latent variables. Then, given the estimates of the prior probabilities, we can optimize the conditional probabilities $p_{Y|Z}(y_n|z_n;\theta)$ by treating each subgroups separately and estimating the parameters of each subgroup's probability distribution using maximum likelihood.

In contrast, optimizing the distribution of the observed data alone $p_Y(y_n;\theta)$ involves estimating the parameters of each subgroup's probability distribution without knowledge of the underlying latent variables. This can be more difficult, especially when the subgroups are highly overlapping or when there are few observations per subgroup.

Therefore, in the context of mixture models, it is often easier to optimize the joint distribution of the observed data and the latent variables $p_{Y,Z}(y_n, z_n;\theta)$ than to optimize the distribution of the observed data alone $p_Y(y_n;\theta)$.

Theory Question 3

Variational inference and the Expectation-Maximization (EM) algorithm are two popular methods for approximating the posterior distribution of a latent variable model. While they share some similarities, they also have some important differences.
Variational Bayesian methods are a family of techniques for approximating intractable integrals arising in Bayesian inference and machine learning. They are typically used in complex statistical models consisting of observed variables (usually termed ”data”) as well as unknown parameters and latent variables, with various sorts of relationships among the three types of random variables, as might be described by a graphical model. As typical in Bayesian inference, the parameters and latent variables are grouped together as ”unobserved variables”. Variational Bayesian methods are primarily used for two purposes:
1.To provide an analytical approximation to the posterior probability of the unobserved variables, in order to do statistical inference over these variables. 2.To derive a lower bound for the marginal likelihood (sometimes called the evidence) of the observed data (i.e. the marginal probability of the data given the model, with marginalization performed over unobserved variables). This is typically used for performing model selection, the general idea being that a higher marginal likelihood for a given model indicates a better fit of the data by that model and hence a greater probability that the model in question was the one that generated the data. (See also the Bayes factor article.)
This is done by minimizing the Kullback-Leibler (KL) divergence between the true posterior and the approximation. The approximation is chosen to belong to a family of distributions that is easy to work with.
Variational Bayes can be seen as an extension of the expectation-maximization (EM) algorithm from maximum a posteriori estimation (MAP estimation) of the single most probable value of each parameter to fully Bayesian estimation which computes (an approximation to) the entire posterior distribution of the parameters and latent variables. As in EM, it finds a set of optimal parameter values, and it has the same alternating structure as does EM, based on a set of interlocked (mutually dependent) equations that cannot be solved analytically.
For more information and better description, you can visit below page:
https://mbernste.github.io/posts/variational_inference/
variational inference is used in situations in which we have a model that involves hidden random variables Z, observed data X, and some posited probabilistic model over the hidden and observed random variables P (Z, X).

In statistics, an expectation–maximization (EM) algorithm is an iterative method to find (local) maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.
The E-step computes the expected value of the latent variables given the observed data and the current estimate of the parameters.
The M-step updates the parameters to maximize the likelihood of the observed data given the expected values of the latent variables.
The EM algorithm gives the local optimization.
One key difference between the two methods is that while the EM algorithm focuses on finding the maximum likelihood estimate for the model parameters, the goal of variational inference is to find an approximate distribution that is closest to the target distribution. Additionally, variational inference is mostly used in Bayesian inference, while the EM algorithm is commonly used in Frequentist statistics.
In some situation, when we can't use EM algorithm we use variational inference like the models aren't simple, we use variational inference to simple them.

Theory Question 4

1. Determine model parameters and initialize them.

To fit a Gaussian Mixture Model (GMM), we need to determine the number of components or clusters in the data. Let's assume that we have decided on the number of components K. We then need to initialize the model parameters, which include the means, covariances, and mixing proportions of each component. So in summary our parameters are:

- **K:** Number of mixture components
- $\mu_{\mathbf{k}}$: The mean associated with each component k
- $\sigma_{\mathbf{k}}$: The standard deviation associated with each component k
- $\pi_{\mathbf{k}}$: The mixing proportions defined as Pr$(z = k)$

Initializing the parameters can be tricky. If you initialize all the components to the same parameters, they will never break apart because of symmetries in the model. If you choose a really bad initialization for one of the components, it may "die out" during E-M because the component doesn't assign high likelihood to any data points. There's no strategy that's guaranteed to work, but one good option is to initialize the different clusters to have random means and very broad standard deviation.
A common approach is to use k-means clustering to obtain initial estimates of the means and the mixing proportions, and to set the covariances to some initial value, such as the identity matrix.

Assuming that we have n data points and t features, here are our initial parameters:

- **K:** Number of mixture components (Given to us by the problem)
- $\mu_{\mathbf{k}}$: We can put the whole data set's mean (A matrix of $K \times t$)
- $\sigma_{\mathbf{k}}$: The Identity matrix of size $K \times t \times t$
- $\pi_{\mathbf{k}}$: A vector of size K and elements of $\frac{1}{K}$

2. Compute complete dataset likelihood.

We need to compute the likelihood of the observed data given the model parameters. This can be done using the formula for mixture models:

$$p(x_i;\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)$$

where $p(x_i;\theta)$ is the probability density function of the i-th data point, w_k is the mixing proportion of the k-th component, $\mathcal{N}(x_i|\mu_k, \Sigma_k)$ is the probability density function of the multivariate normal distribution with mean μ_k and covariance matrix Σ_k.

Assuming that all data samples are independent, the complete dataset likelihood is given by the product of the individual probabilities:

$$p(X,Z|\theta) = p(x_1, x_2, \cdots, x_N, z_1, z_2, \cdots, z_N|\theta) = \prod_{n=1}^{N} p(x_n, z_n|\mu, \Sigma) \prod_{n=1}^{N} p(z_n|\pi)$$

$$= \prod_{n=1}^{N}\prod_{k=1}^{K} \mathcal{N}(x_n|\mu_k, \Sigma_k)^{z_{nk}} \prod_{n=1}^{N}\prod_{k=1}^{K} \pi_k^{z_{nk}}$$

$$L(\theta) = (X, Z|\theta) = \ln\left(\prod_{n=1}^{N}\prod_{k=1}^{K} \mathcal{N}(x_n|\mu_k, \Sigma_k)^{z_{nk}} \prod_{n=1}^{N}\prod_{k=1}^{K} \pi_k^{z_{nk}}\right)$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}[\ln(\mathcal{N}(x_n|\mu_k, \Sigma_k)) + \ln(\pi_k)] \;\; (*)$$

3. Find closed-form solution for parameters using EM algorithm.

Expectation step:
To estimate the model parameters, we can use the Expectation-Maximization (EM) algorithm.
The EM algorithm consists of two steps: the E-step and the M-step.
In the E-step, we compute the posterior probabilities of the latent variables or cluster assignments given the data and the current estimates of the model parameters.

Now we apply the expected value operator to (*):

$$\mathbb{E}_{p(Z|X,\theta)}[\ln\{p(X,Z|\theta)\}]$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K} \mathbb{E}_{p(Z|X,\theta)}[z_{nk}]\left(\ln(\mathcal{N}(x_n|\mu_k,\Sigma_k)) + \ln(\pi_k)\right)$$

$$\mathbb{E}_{p(Z|X,\theta)}[z_{nk}] = \sum_{z_1}\sum_{z_2}\cdots\sum_{z_N} z_{nk}p(Z|X,\theta^{old})$$

$$= \sum_{n} z_{nk}p(z_n|x_n,\theta^{old})$$

$$= \sum_{n} z_{nk}\left(\frac{p(x_n,z_n|\theta^{old})}{p(x_n|\theta^{old})}\right) = \sum_{n} z_{nk}\left(\frac{p(x_n|z_n,\theta^{old})p(z_n|\theta^{old})}{p(x_n|\theta^{old})}\right)$$

$$= \sum_{n} z_{nk}\left(\frac{\prod_{k=1}^{K}\mathcal{N}(x_n|\mu_k,\Sigma_k)^{z_{nk}}\pi_k^{z_{nk}}}{\sum_{z_{nk}}\prod_{k=1}^{K}\mathcal{N}(x_n|\mu_k,\Sigma_k)^{z_{nk}}\pi_k^{z_{nk}}}\right)$$

$$= \frac{\pi_k\mathcal{N}(x_n|\mu_k,\Sigma_k)}{\sum_{k'=1}^{K}\pi_{k'}\mathcal{N}(x_n|\mu_{k'},\Sigma_{k'})}$$

Maximization step:

In the M-step, we update the estimates of the model parameters by maximizing the expected complete log-likelihood of the data given the posterior probabilities of the latent variables.

$$G(\theta) = \mathbb{E}_{p(\mathbf{Z}|\mathbf{X},\theta)}[\ln p(\mathbf{Z}|\mathbf{X},\theta)]$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K} \mathbb{E}_{p(\mathbf{Z}|\mathbf{X},\theta)}[z_{nk}]\{\ln (x_n|_k, \Sigma) + \ln \pi_k\}$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{nk})\{-\frac{1}{2}(x_n - \mu_k)^T\Sigma^{-1}(x_n - \mu_k) - \frac{1}{2}(\ln 2\pi + \ln|\Sigma_k|) + \ln \pi_k\}$$

$$\frac{\partial G(\theta)}{\partial \mu_k} = 0 \quad , \frac{\partial G(\theta)}{\partial \Sigma_k = 0}$$

$$\frac{\partial G(\theta)}{\partial \mu_k} = \sum_{n=1}^{N} \gamma(z_{nk})\Sigma_k^{-1}(x_n - \mu_k) = 0$$

$$\rightarrow \mu_k = \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}}$$

$$\frac{(\theta)}{\partial \Sigma_k = 0} = \sum_{n=1}^{N} \gamma(z_{nk})\{(x_n - \mu_k)(x_n - \mu_k)^T - \Sigma_k = 0\}$$

$$\rightarrow \Sigma_k = \frac{\sum_{n=1}^{N} \gamma_{nk}(x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma_{nk}}$$

$$G(\theta) = \sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{nk})\{-\frac{1}{2}(x_n - \mu_k)^T\Sigma^{-1}(x_n - \mu_k) - \frac{1}{2}(\ln 2\pi + \ln|\Sigma_k|) + \ln \pi_k\}$$

Subject to $\sum_{k=1}^{K}$, we have:

$$L(\theta) = G(\theta) - \lambda(\sum_{k=1}^{K} \pi_k - 1)$$

$$\frac{\partial L(\theta)}{\partial \pi_k} = 0 \rightarrow \pi_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

To sum up we have:

Expectation step:

$$\gamma(z_{nk})^{(t+1)} = \frac{\pi_k^{(t)} N(y_i; \theta_k^{(t)})}{\sum_{j=1}^{k} \pi_j^{(t)} N(y_i; \theta_j^{(t)})}$$

Maximization step:

$$\pi_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})}{N}, \quad \mu_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) x_n}{\sum_{n=1}^{N} \gamma(z_{nk})}$$

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})}$$

We then repeat the E-step and M-step until convergence, which is typically determined by the change in the complete dataset likelihood.

Theory Question 5

1. Determine model parameters and initialize them.

The model parameters for a categorical mixture mode describes as follows:
K = The number of mixture component
N = The number of observation
J = The number of categorical data

$$\pi = \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_k \end{bmatrix}$$

$\theta_{i=1...k}$ = parameter of distribution of observation associated with component i
$\phi_{i=1...k}$ = mixture weight, i.e.,prior probability associated with component i
ϕ = *k*-dimensional vector composed of all the individual $\phi_{1...k}$; must sum to 1
$z_{i=1..N}$ = component of observation i
$x_{i=1..N}$ = observation i
$F(x|\theta)$ = probability distribution of an observation, parameterized on θ
$z_{i=1..N}$ $\sim$ $Categorical(\phi)$
$x_{i=1..N}|z_{i=1...N}$ $\sim$ $F(\theta_{z_i})$
V : Dimension of categorical observations, e.g., size of word vocabulary
$\theta_{i=1...K,j=1...V}$: probability for component **i** of observing item **j**
$\boldsymbol{\theta}_{i=1...K}$: vector of dimension V, composed of $\theta_{i,1...V}$; must sum to 1
the random variables:
$z_{i=1..N}$ $\sim$ $Categorical(\phi)$
$x_{i=1..N}$ $\sim$ $Categorical(\theta_{z_i})$

2. Compute complete data-set likelihood.

we need to compute log likelihood, which is the log-likelihood of the observed data and the latent variables.
Let χ be an arbitrary measurable space and let $x = x_1, ..., x_n$ be n independent vectors in χ such that each x_i arises from a probability distribution with density (a mixture model)

$$f(x_i|\theta) = \sum_{k=1}^{K} \pi_k h(x_i|\lambda_i, \alpha)$$

where the π_k's are the mixing proportions $(0 < \pi_k < 1$ for all $k = 1, ..., K$ and $\pi_1 + ... + \pi_K = 1)$,$h(|\lambda_k, \alpha)$denotes a d-dimensional distribution parameterized by λ_k and α. The parameters α are not dependent from k and are common to all the components of the mixture. The vector parameter to be estimated is $\theta = (\pi_1, ..., \pi_K, \lambda_1, ..., \lambda_K, \alpha)$ and is chosen to maximize the observed log-likelihood.

$$L(\theta|x_1, ..., x_n) = \sum_{i=1}^{N} \ln \sum_{k=1}^{K} \pi_k h(x_i|\lambda_i, \alpha)$$

now our probability is:
A Categorical probability distribution on a finite space $\chi = 1, ..., L$ is a probability distribution of the form:

$$P(x = l) = p_l \qquad p_l > 0, l \in \chi$$

were $p_1 + p_2 + ... + p_L = 1$
A joint Categorical probability distribution on χ^d is a probability distribution of the form:

$$P(x = (x_1, ..., x_d)) = \prod_{j=1}^{d} p_{x_j}^{j}$$

It is well known that for a mixture distribution, a sample of indicator vectors or labels $z = z1, ..., zn$, with $z_i = (z_{i1}, ..., z_{iK}), z_{ik} = 1$ or 0, according to the fact that x_i is arising from the kth mixture component or not, is associated to the observed data x.

3. Find closed-form solution for parameters using EM algorithm.

Starting from an initial arbitrary parameter θ, the mth iteration of the EM algorithm consists of repeating the following E and M steps.

$$P(y_i|\theta) = \sum_{k=1}^{K} p(y_i; \theta_k)$$

E step:computing the expected complete log-likelihood (E-step)
γ_k can be viewed as the responsibility

$$\gamma_{k,i}^{(t+1)} = p(z = k|x_i) = \frac{p(z = k)p(x_i|z = k)}{p(x)} =$$

$$\frac{p(z = k)p(x_i|z = k)}{\sum_{j=1}^{k} p(z = j)p(x_i|z = j)} =$$

$$\frac{\pi_k^{(t)} p(y_i; \theta_k^{(t)})}{\sum_{j=1}^{k} \pi_j^{(t)} p(y_i; \theta_j^{(t)})}$$

M step: maximizing the expected complete log-likelihood with respect to the model parameters
log-likelihood

$$\sum_{i=1}^{N} \ln \sum_{k=1}^{K} \pi_k p(y_i; \theta_k)$$

we should compute:

$$\frac{\partial p(y_i; \theta_k^{(t)})}{\partial \theta_k^{(t)}} = 0$$

$$\pi_k^{(t+1)} = \frac{1}{N} \sum_{i=1}^{N} \gamma_{k,i}^{(t)}$$

Simulation Question 1

Here's the code for simulation question 1:

```python
import numpy as np
import matplotlib.pyplot as plt


#Simulation Question 1
# Load the data from Image1.csv or Image2.csv
data = np.loadtxt('Image.csv', delimiter=',')

# plot the data points with three different colors
plt.scatter(data[1:201, 1], data[1:201, 2], color='red', label='
    Background')
plt.scatter(data[201:401, 1], data[201:401, 2], color='green',
    label='Normal Brain Tissue')
plt.scatter(data[401:, 1], data[401:, 2], color='blue', label='
    Tumor Tissue')
plt.legend()
plt.show()
```

Listing 1: Simulation Question 1

In the code above, we plotted the three distributions using the scatter function in matplotlib library.

And here are the results:

For Image1.csv:

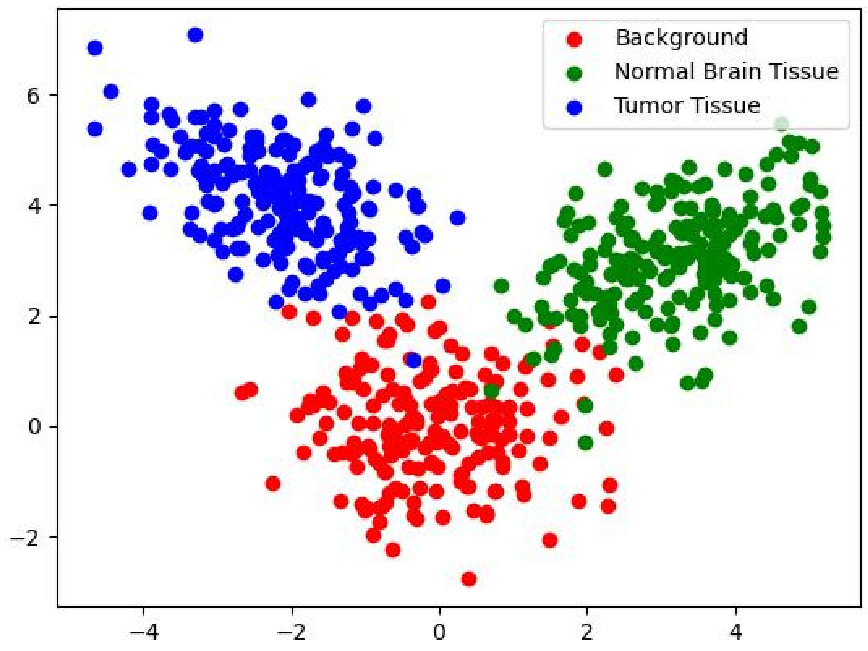


Figure 1: Image1 real data distribution

For Image2.csv:

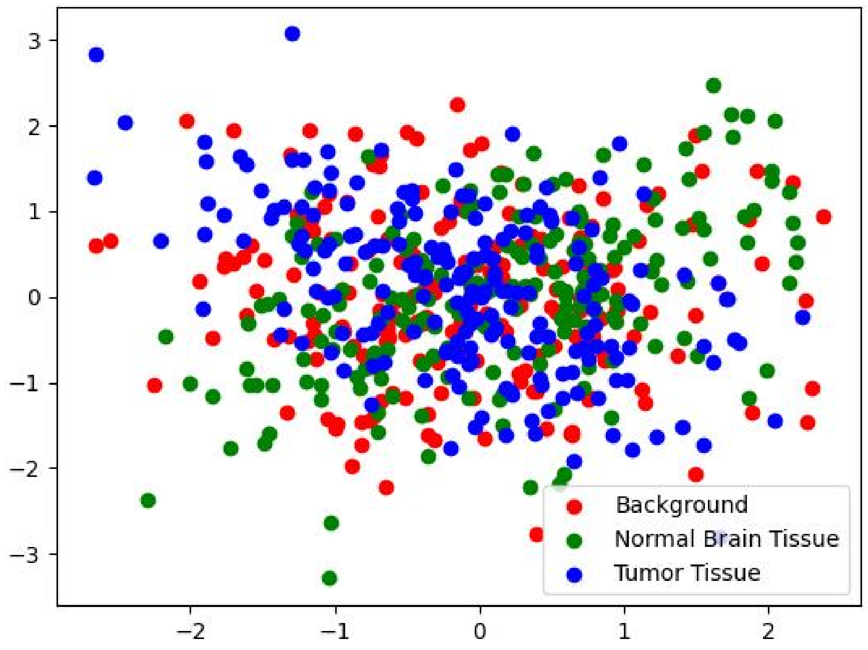


Figure 2: Image2 real data distribution

Simulation Question 2

Here's the code for simulation question 2:

```python
import numpy as np
def e_step(X, mu, var, pi):
    """
    Perform the E-step of the EM algorithm.
    Inputs:
        X: (N, 2) array representing the data points
        mu: (3, 2) array representing the means of the three Gaussian distributions
        var: (3, 2, 2) array representing the covariance matrices of the three Gaussian distributions
        pi: (3,) array representing the mixing coefficients of the three Gaussian distributions
    Returns:
        R: (3, N) array representing the probability distribution of each data point across the three Gaussian distributions
    """
    N = X.shape[0]
    K = mu.shape[0]
    R = np.zeros((K, N))

    for k in range(K):
        cov_inv = np.linalg.inv(var[k])
        X_mu = X - mu[k]
        exponent = np.sum(X_mu @ cov_inv * X_mu, axis=1)
        coef = 1 / np.sqrt((2 * np.pi) ** 2 * np.linalg.det(var[k]))
        R[k] = pi[k] * coef * np.exp(-0.5 * exponent)
    # Normalize the probabilities
    R /= np.sum(R, axis=0)

    return R
```

Listing 2: Simulation Question 2

In this part, we code the expectation step in the EM algorithm. The output of this function is the R matrix which determines to which distribution each data point belongs.
The method to determine the R matrix has been thoroughly explained in the theory question 4. Here, we simply bring the closed form of R:

$$r_k^{(i)} = \frac{\pi_k \mathcal{N}(x^{(i)}; \mu_k, \Sigma_k)}{\sum_{l=1}^{K} \pi_l \mathcal{N}(x^{(i)}; \mu_l, \Sigma_l)}$$

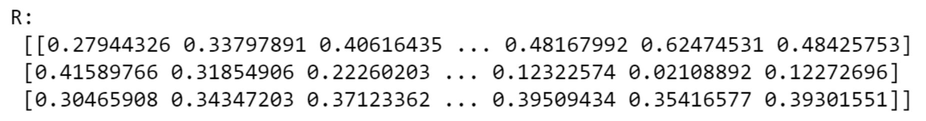
Using the formula for multivariate normal distribution and multipling each probability to the assigned weight, in line 17 to 21 of the code, we tend to compute the numerator of $R[i]$ (in other words the responsibility of the i-th data point):

$$f_X(x \mid \mu, \Sigma) = \frac{1}{2\pi\sqrt{|\Sigma|}} \text{exp}\left(\frac{-1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

At last, in line 24 we normalize the responsibilities. In other words, we are dividing the numerators that we computed in the for loop, by the denominator which is the sum of all probabilities of that data.

Here are the results after one iteration:

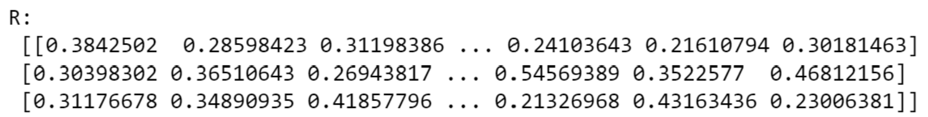
For Image1.csv:



```
R:
 [[0.27944326 0.33797891 0.40616435 ... 0.48167992 0.62474531 0.48425753]
 [0.41589766 0.31854906 0.22260203 ... 0.12322574 0.02108892 0.12272696]
 [0.30465908 0.34347203 0.37123362 ... 0.39509434 0.35416577 0.39301551]]
```

Figure 3: E-step one iteration result (Image1)

For Image2.csv:



```
R:
 [[0.3842502  0.28598423 0.31198386 ... 0.24103643 0.21610794 0.30181463]
 [0.30398302 0.36510643 0.26943817 ... 0.54569389 0.3522577  0.46812156]
 [0.31176678 0.34890935 0.41857796 ... 0.21326968 0.43163436 0.23006381]]
```

Figure 4: E-step one iteration result (Image2)

Simulation Question 3

Here's the code for simulation question 3:

```python
import numpy as np
def m_step(X, R):
    """
    Perform the M-step of the EM algorithm.
    Inputs:
        X: (N, 2) array representing the data points
        R: (3, N) array representing the probability distribution of each data point across the three Gaussian distributions
    Returns:
        mu_new: (3, 2) array representing the updated means of the three Gaussian distributions
        var_new: (3, 2, 2) array representing the updated covariance matrices of the three Gaussian distributions
        pi_new: (3,) array representing the updated mixing coefficients of the three Gaussian distributions
    """
    # N: the number of all data point
    N = X.shape[0]
    # K: the number of distribution
    K = R.shape[0]

    #Update the mixing coefficients
    pi_new = np.sum(R, axis=1) / N

    # Update the means
    mu_new = np.zeros((K, 2))
    for k in range(K):
        mu_new[k] = np.sum(R[k].reshape(-1, 1) * X, axis=0) / np.sum(R[k])

    # Update the covariances
    var_new = np.zeros((K, 2, 2))
    for k in range(K):
        X_mu = X - mu_new[k]
        R_diag = np.diag(R[k])
        var_new[k] = (X_mu.T @ R_diag @ X_mu) / np.sum(R[k]) + 1e-6 * np.eye(2)

    return mu_new, var_new, pi_new
```
Listing 3: Simulation Question 3

In line 19 we are computing new π_k with below formula

$$\pi_k = \frac{\sum_{n=1}^{N} R_{nk}}{N}$$

In line 22 we are calculating our $t+1$ step means from Responsibility matrix with below terms:

$$\mu_k = \frac{\sum_{n=1}^{N} R_{nk} x_n}{\sum_{n=1}^{N} R_{nk}}$$

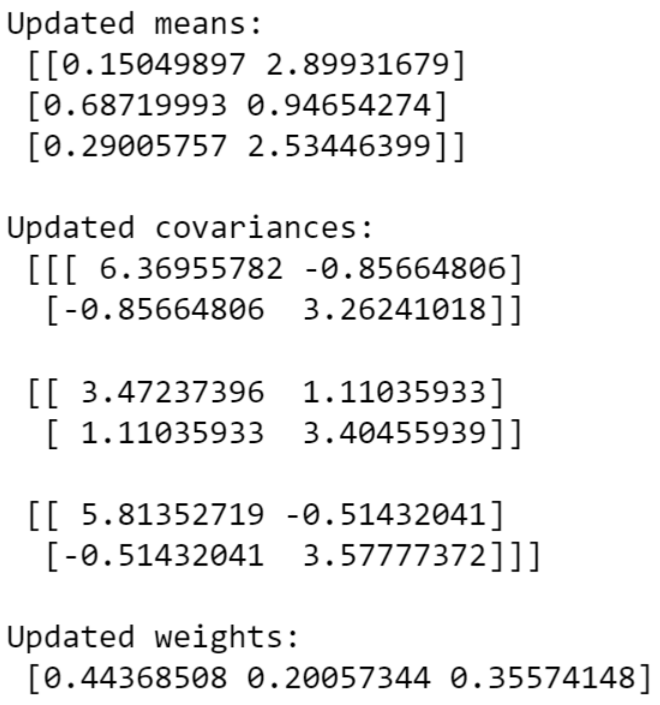
And finally in line 28 we compute the covariance matrix:

$$\Sigma_k = \frac{\sum_{n=1}^{N} R_{nk}(x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^{N} R_{nk}}$$

As you can see, in line 31, we've added a small value to the covariance matrix. After multiple iteration of the code, the covariance matrix becomes singular, and hence it cannot be inverted. In order to prevent this, we add a very small value for instance (10^{-6}) to the diagonal of the covariance matrix. In this way, not only we make sure that the matrix stays invertible, but also our results don't change much.

Here are the results after one iteration:

For Image1.csv:



```
Updated means:
 [[0.15049897 2.89931679]
 [0.68719993 0.94654274]
 [0.29005757 2.53446399]]

Updated covariances:
 [[[ 6.36955782 -0.85664806]
  [-0.85664806  3.26241018]]

 [[ 3.47237396  1.11035933]
  [ 1.11035933  3.40455939]]

 [[ 5.81352719 -0.51432041]
  [-0.51432041  3.57777372]]]

Updated weights:
 [0.44368508 0.20057344 0.35574148]
```

Figure 5: M-step one iteration result (Image1)

For Image2.csv:

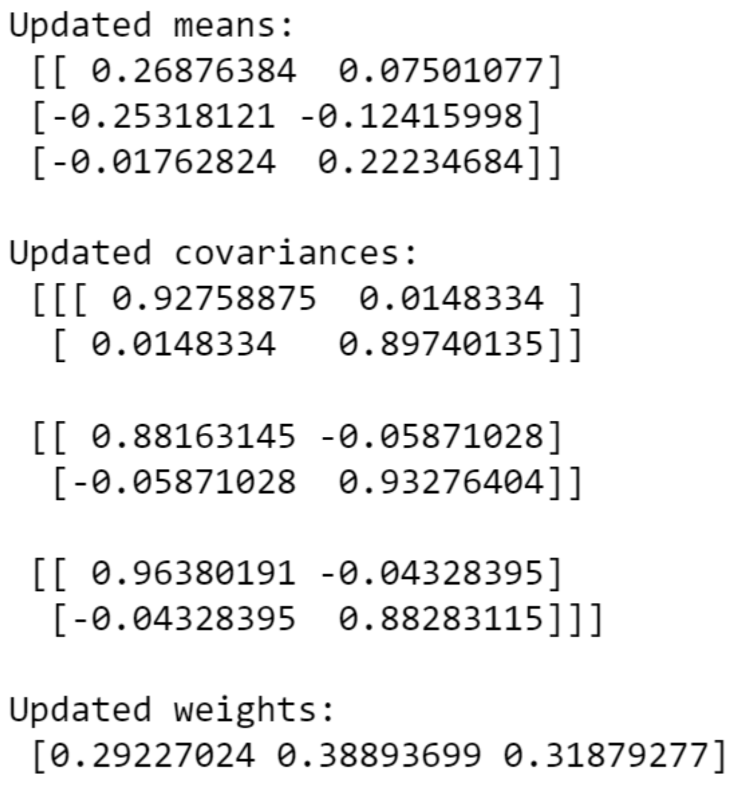

```
Updated means:
 [[ 0.26876384  0.07501077]
 [-0.25318121 -0.12415998]
 [-0.01762824  0.22234684]]

Updated covariances:
 [[[ 0.92758875  0.0148334 ]
  [ 0.0148334   0.89740135]]

 [[ 0.88163145 -0.05871028]
  [-0.05871028  0.93276404]]

 [[ 0.96380191 -0.04328395]
  [-0.04328395  0.88283115]]]

Updated weights:
 [0.29227024 0.38893699 0.31879277]
```

Figure 6: M-step one iteration result (Image2)

Simulation Question 4

Here's the code for simulation question 4:

```python
def Compute_log_likelihood(X, mu, var, pi):
    """
    Compute the log likelihood of the data given the current parameters of the mixture model.
    Inputs:
        X: (N, 2) array representing the data points
        mu: (3, 2) array representing the means of the three Gaussian distributions
        var: (3, 2, 2) array representing the covariance matrices of the three Gaussian distributions
        pi: (3,) array representing the mixing coefficients of the three Gaussian distributions
    Returns:
        ll: float representing the log likelihood of the data
    """
    N = X.shape[0] # Number of data points
    K = mu.shape[0] # Number of distributions
    loglike = 0

    for n in range(N):
        p_x = 0
        for k in range(K):
            cov_inv = np.linalg.inv(var[k])
            X_mu = X[n] - mu[k]
            exponent = X_mu @ cov_inv @ X_mu.T
            coef = 1 / np.sqrt((2 * np.pi) ** 2 * np.linalg.det(var[k]))
            p_x += pi[k] * coef * np.exp(-0.5 * exponent)
        loglike += np.log(p_x)

    return loglike
```

Listing 4: Simulation Question 4

```python
def EM(X, mu, var, pi, max_iter=200, tol=1e-4):
    """
    Run the EM algorithm to estimate the parameters of the Gaussian mixture model.
    Inputs:
        X: (N, 2) array representing the data points
        mu: (3, 2) array representing the initial means of the three Gaussian distributions
        var: (3, 2, 2) array representing the initial covariance matrices of the three Gaussian distributions
        pi: (3,) array representing the initial mixing coefficients of the three Gaussian distributions
        max_iter: int representing the maximum number of iterations to run the algorithm
        tol: float representing the tolerance level for convergence
    Returns:
        mu: (3, 2) array representing the estimated means of the three Gaussian distributions
        var: (3, 2, 2) array representing the estimated covariance matrices of the three Gaussian distributions
        pi: (3,) array representing the estimated mixing coefficients of the three Gaussian distributions
        log_likelihoods: list of log-likelihoods at each iteration
    """
    log_likelihoods = []
    for i in range(max_iter):
        # E-step
        R = e_step(X, mu, var, pi)

        # M-step
        mu_new, var_new, pi_new = m_step(X, R)

        # Compute log-likelihood
        log_likelihood = Compute_log_likelihood(X, mu_new, var_new, pi_new)

        # Check for convergence
        if i > 0 and abs(log_likelihood - log_likelihoods[-1]) < tol:
            break

        # Update parameters
        mu = mu_new
        var = var_new
        pi = pi_new

        # Add log-likelihood to the final list
        log_likelihoods.append(log_likelihood)


     # Compute the argmax of the responsibilities to assign each
    data point to a distribution
    argmax_R = np.argmax(R, axis=0)

    # Convert argmax to binary matrix R
    R = np.zeros((3, 600))
    for i in range(len(argmax_R)):
        R[argmax_R[i], i] = 1


    # Plot the results
    labels = np.argmax(R, axis=0)
    colors = ['deeppink', 'limegreen', 'deepskyblue']
    for j in range(3):
        plt.scatter(X[labels == j, 0], X[labels == j, 1], c=
colors[j], label=f'Distribution {j + 1}')
        plt.scatter(mu[j, 0], mu[j, 1], marker='x', s=100, c='
black')
    plt.title('Estimated Distibutions')
    plt.legend()
    plt.savefig('Estimated Distibutions.jpg')
    plt.show()
    plt.close()

    return mu, var, pi, R, log_likelihoods
```
Listing 5: Simulation Question 4

The inputs and outputs of this function are explained in the code.
Firstly, we initiate the number of maximum iterations and a tolerance in order to break the iteration loop whenever convergence is achieved.
Convergence is determined by the difference between the log likelihood of the data with the last element of all the log-likelihoods. We have implemented a separate function for computing log-likelihood in which we simply take the $\log_{10}$ of the sum of the normal distributions (property of log function).

And here are the results:

For Image1.csv:

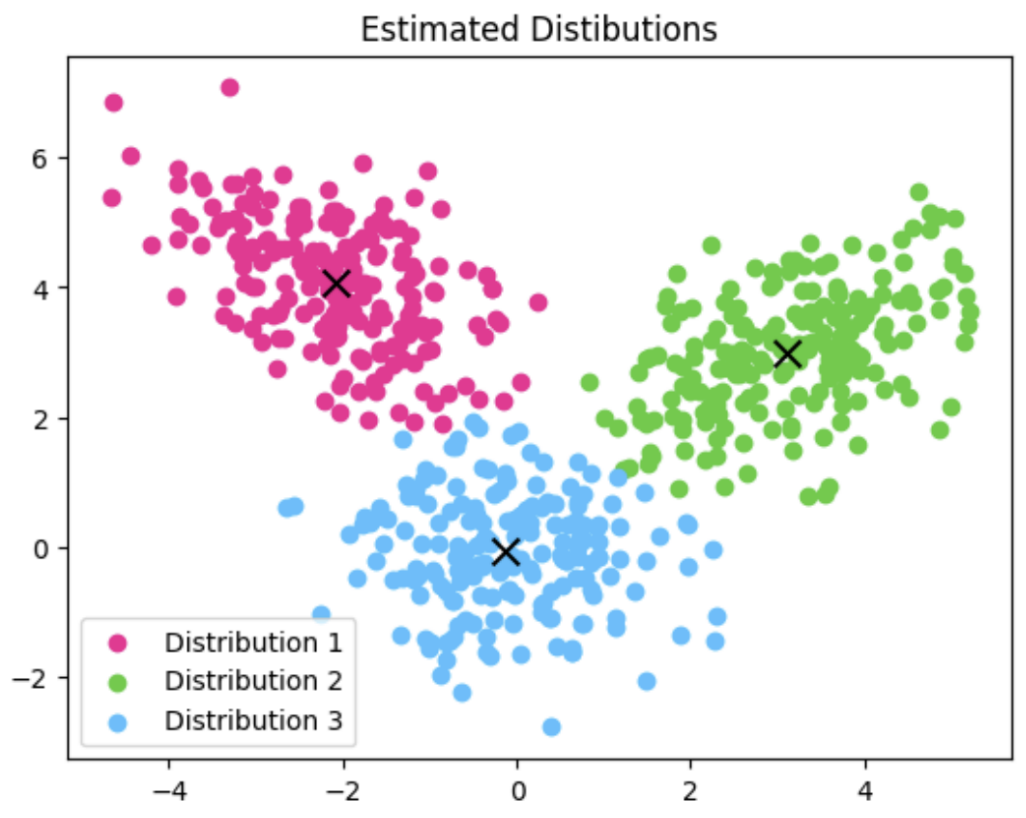


Figure 7: Estimated Distributions Plot (Image1)

For Image2.csv:

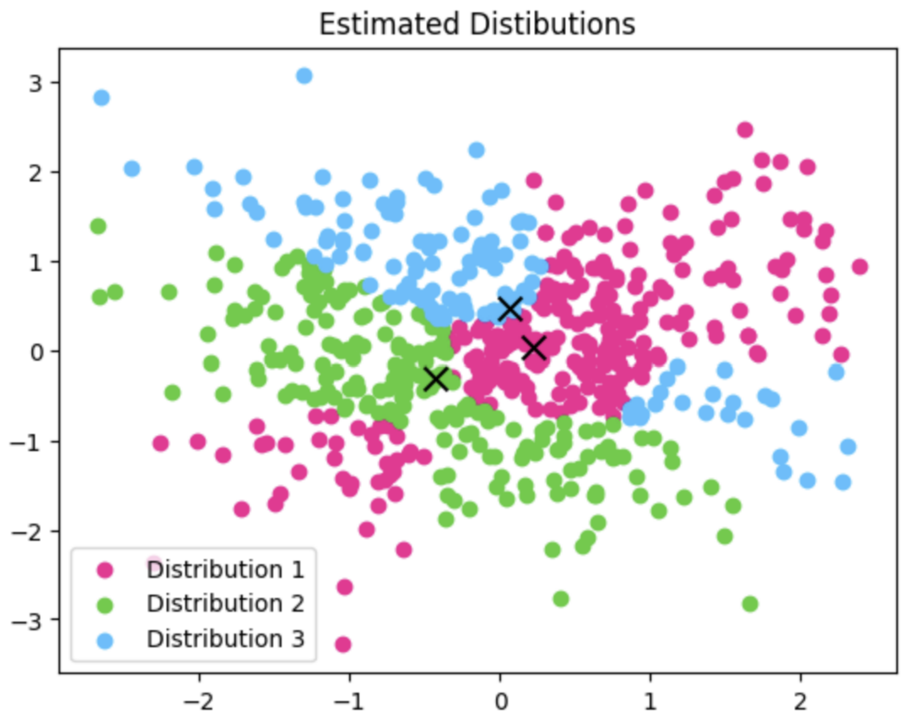


Figure 8: Estimated Distributions Plot (Image2)

Simulation Question 5

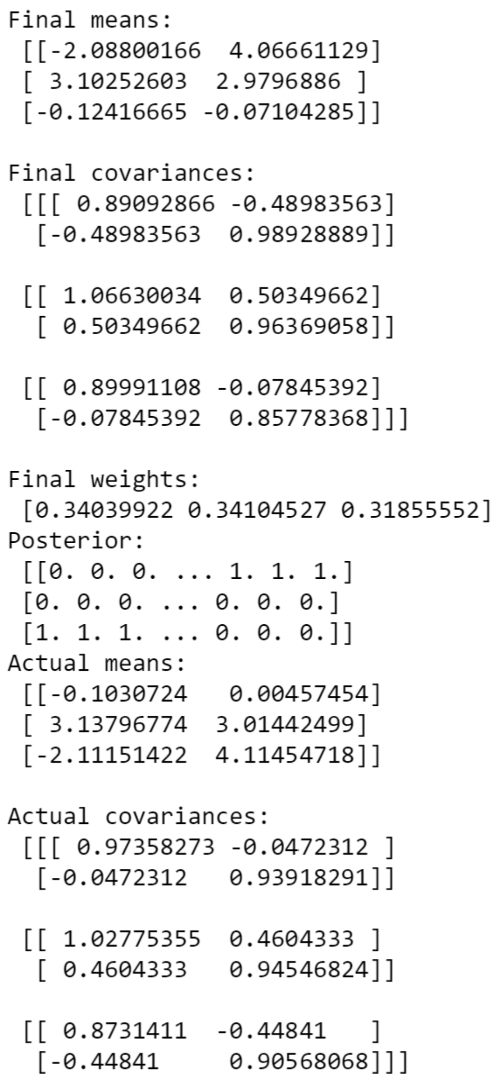
In this section, we also implemented a code to get the accuracy percentage of our results:

Here's the code:

```python
#Computing Accuracy Percentage
asignr=np.argmax(assignments,axis=0)
es_point = np.zeros((3,3))
for i in range (3):
    for j in range(3):
        for k in range(200):
            if asignr[j*200+k]==i:
                es_point[i,j]+=1
print("\nNumber of correctly classified:\n", es_point)
max_point=(np.amax(es_point,axis=0)/200)*100
print("\nAccuracy percentage:\n", max_point)
```

Listing 6: Simulation Question 5

Here are the real data and estimated parameters for Image1.csv:



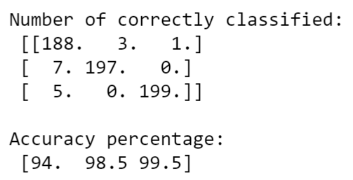
```
Final means:
 [[-2.08800166  4.06661129]
 [ 3.10252603  2.9796886 ]
 [-0.12416665 -0.07104285]]

Final covariances:
 [[[ 0.89092866 -0.48983563]
  [-0.48983563  0.98928889]]

 [[ 1.06630034  0.50349662]
  [ 0.50349662  0.96369058]]

 [[ 0.89991108 -0.07845392]
  [-0.07845392  0.85778368]]]

Final weights:
 [0.34039922 0.34104527 0.31855552]
Posterior:
 [[0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 0. 0. 0.]
 [1. 1. 1. ... 0. 0. 0.]]
Actual means:
 [[-0.1030724   0.00457454]
 [ 3.13796774  3.01442499]
 [-2.11151422  4.11454718]]

Actual covariances:
 [[[ 0.97358273 -0.0472312 ]
  [-0.0472312   0.93918291]]

 [[ 1.02775355  0.4604333 ]
  [ 0.4604333   0.94546824]]

 [[ 0.8731411  -0.44841   ]
  [-0.44841     0.90568068]]]
```

Figure 9: Estimated Parameters (Image1)

At first, we should note that in the estimated data the order of the distributions may change. Here we can see that the estimated means and covariances are very close to the real data's means and covariances. And the posterior shows that the code has correctly distinguished the 3 distributions.

Also, in the picture below, it is apparent that the code has identified the correct distribution for a large portion of the data. Of course we've had some misclassified cases, all of which were outliers.

Here's the accuracy percentage for Image1.csv:



```
Number of correctly classified:
 [[188.   3.   1.]
 [  7. 197.   0.]
 [  5.   0. 199.]]

Accuracy percentage:
 [94.  98.5 99.5]
```

Figure 10: Accuracy (Image1)

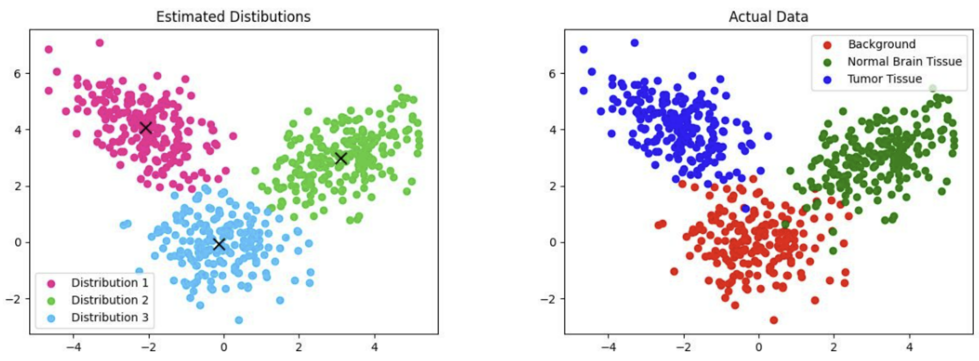
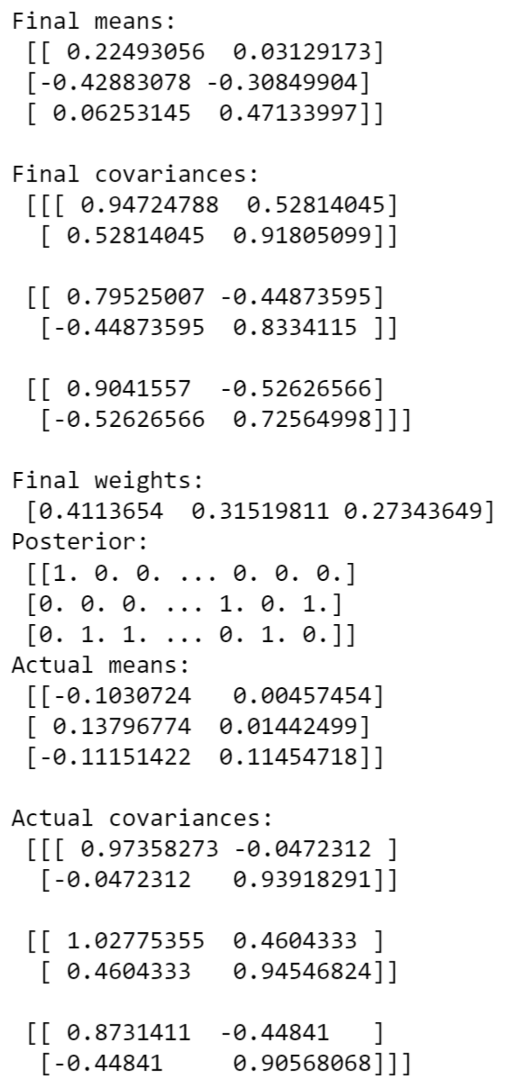


Figure 11: Comparison (Image1)

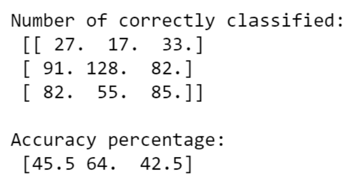
Here are the real data and estimated parameters for Image2.csv:



```
Final means:
 [[ 0.22493056  0.03129173]
 [-0.42883078 -0.30849904]
 [ 0.06253145  0.47133997]]

Final covariances:
 [[[ 0.94724788  0.52814045]
  [ 0.52814045  0.91805099]]

 [[ 0.79525007 -0.44873595]
  [-0.44873595  0.8334115 ]]

 [[ 0.9041557  -0.52626566]
  [-0.52626566  0.72564998]]]

Final weights:
 [0.4113654  0.31519811 0.27343649]
Posterior:
 [[1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 0. 1.]
 [0. 1. 1. ... 0. 1. 0.]]
Actual means:
 [[-0.1030724   0.00457454]
 [ 0.13796774  0.01442499]
 [-0.11151422  0.11454718]]

Actual covariances:
 [[[ 0.97358273 -0.0472312 ]
  [-0.0472312   0.93918291]]

 [[ 1.02775355  0.4604333 ]
  [ 0.4604333   0.94546824]]

 [[ 0.8731411  -0.44841   ]
  [-0.44841     0.90568068]]]
```

Figure 12: Estimated Parameters (Image2)

Here, as we can see the real data's distribution is all over the place. The distributions are meddled together and therefore their variances are much higher than the previous case and the data points are very spread out from the mean, and from one another. As in Image2.csv, the variances and means of these three distributions are quite close, it is apparent that our code won't have a much high accuracy in classifying this data set. As we can see in figure 12, the estimated means and variances are not as close to the real data parameters as we expect them to be. This is also due to the synchrony of the three distribution's parameters.
In comparison to the data set from Image1.csv file, here we don't have separate classes and the data for each distribution is widely spread and not concentrated around a value.
In computing the accuracy for Image2.csv file, note that the data assigned to some distributions went above 200 and as every time we ran the code, the estimated results changed, these percentages aren't stable.
As seen in the figure below, the classification accuracy is lower here than the case for Image1.csv:



```
Number of correctly classified:
 [[ 27.  17.  33.]
 [ 91. 128.  82.]
 [ 82.  55.  85.]]

Accuracy percentage:
 [45.5 64.  42.5]
```

Figure 13: Accuracy (Image2)

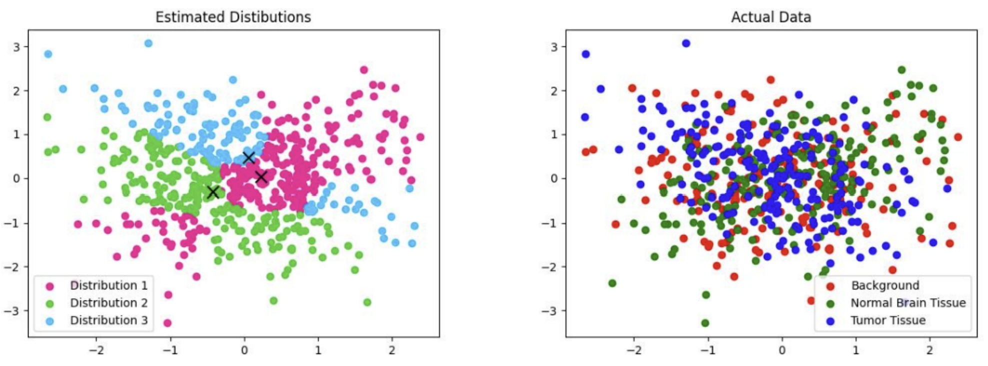


Figure 14: Comparison (Image2)