Student Name: Essakine Amer
Student ID: amer.essakine@ens-paris-saclay.fr

Computational statistics
TP 2

Exercise 1

1. Given that we have a way to sample a uniform random variable U on $[0,1]$, we can generate a random variable X having the discrete distribution on $X = \{x_1, ..., x_n\}$ given by

$$\forall i \in \{1, ..., n\}, P(X = x_i) = p_i$$

   By inversing the cdf of X Giving U a uniform random variable on $[0,1]$, then :

$$X = \begin{cases} x_1 \text{ if } u \leq p_1 \\ x_k \text{ if } \sum_{i=1}^{k-1} p_i < u \leq \sum_{i=1}^{k} p_i \end{cases}$$

2. The following Python code provides an algorithm for generating random variables with a specified discrete probability distribution. It includes an example of generating a sequence of random numbers using the algorithm. Additionally, the script compares the experimental distribution to the theoretical distribution :

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

def generating_discrete(support,probabilities,N_samples) :
    samples = []
    cdf = np.cumsum(probabilities)
    for i in range(N_samples) :
        u = np.random.uniform(0,1)
        for i,cdf_value in enumerate(cdf) :
            if u <= cdf_value :
                samples.append(support[i])
                break
    return np.array(samples)
N = 10
probabilities = np.random.rand(N)
support = np.arange(N)
probabilities = probabilities / np.sum(probabilities)
samples = generating_discrete(support,probabilities,N_samples=100000)
hist, bin_edges = np.histogram(samples, bins=range(11), density=True)
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2 - 0.5
plt.bar(support, probabilities, width=0.4, label='Theoretical Distribution', color='black')
plt.bar(bin_centers, hist, width=0.4, alpha=0.5, label='Empirical Distribution', color='red')
plt.legend()
plt.show()
```

3. Here is the visualization given by the previous code for 100000 samples: As shown in the figure, the algorithm demonstrates high accuracy in generating a finite discrete distribution.

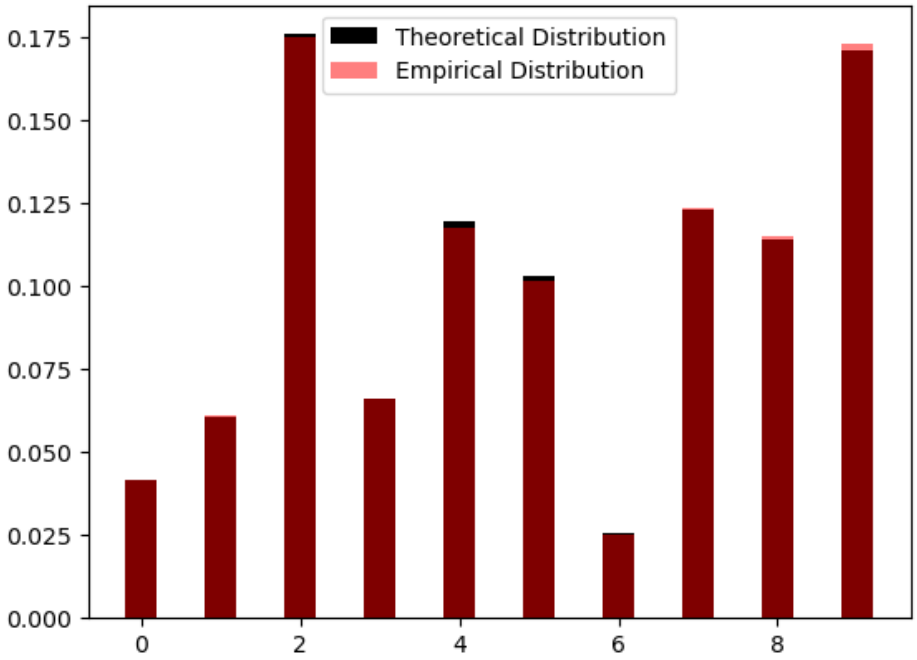

Figure 1: comparaison between the experimental distribution and the theoretical distribution of a finite random variable

Exercice 2

1. For $j = 1, \ldots, m$ denote by p_j the density function of the normal distribution $\mathcal{N}(\mu_j, \Sigma_j)$, then for all i:

$$\begin{aligned} f_\theta(x_i) &= \sum_{j=1}^{m} f_\theta(X_i|\theta, Z_j = i)P(Z_j = i) \\ &= \sum_{j=1}^{m} \pi_j p_j(x_i) \end{aligned}$$

   Hench the likehood of θ is given by :

$$\mathcal{L}(x_1, \ldots x_n; \theta) = \prod_{i=1}^{n}(\sum_{j=1}^{m} \pi_j p_j(x_i))$$

2. The following code sample a set of observation of 1000 points according to a gaussian mixture law and visualize the set of points :

```python
def gaussian_mixture(mu,sigma,pi,N_samples) :
    C = mu.shape[0]
    support = np.arange(C)
    discrete_samples = generating_discrete(support,pi,N_samples)
    samples = []
    labels = []
    for i in discrete_samples :
        mean = mu[i]
        std_dev = sigma[i]
        sample = np.random.multivariate_normal(mean,std_dev)
        samples.append(sample)
        labels.append(i)
    return np.array(samples),np.array(labels)
mu = np.array([[0, 0], [3, 3], [-3, 3]])
sigma_sqrt = [
[[1, 0.5], [0.5, 1]],
[[1, -0.7], [-0.7, 1]],
[[0.5, 0], [0, 0.5]]
]
pi = [0.4, 0.4, 0.2]
samples,labels = gaussian_mixture(mu, sigma_sqrt, pi, N_samples=10000)
colors = ['red', 'blue', 'green']
for i, color in enumerate(colors):
    class_samples = samples[labels == i]
    plt.scatter(class_samples[:, 0], class_samples[:, 1], c=color, label=f"Class {i}",
                                                          alpha=0.6)
plt.legend()
plt.show()
```

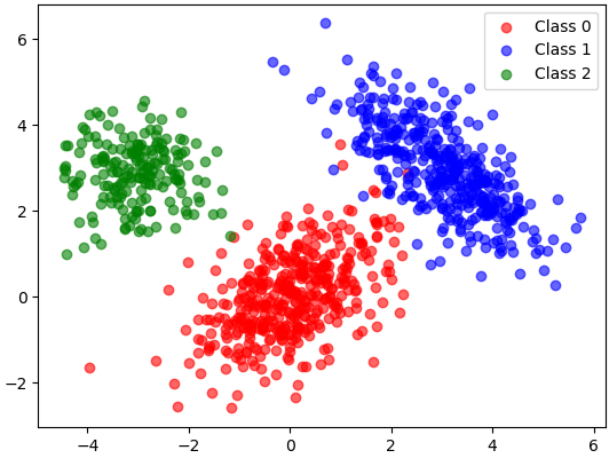

Figure 2: GMM samples

3. The code for the EM algorithm in Python is given :

```python
def EM_algorithm(observation, k, n_iter):
    n, d = np.shape(observation)
    # Initialization
    np.random.seed(42)
    mu = np.random.rand(k, d)
    sigma = np.array([np.eye(d) for _ in range(k)])
    pi = np.ones(k) / k
    log_likelihoods = []
    p = np.zeros((n, k))

    for iter in range(n_iter):
        # Expectation step
        for i in range(n):
            for j in range(k):
                diff = observation[i] - mu[j]
                exponent = -0.5 * diff.T @ np.linalg.inv(sigma[j]) @ diff
                norm_const = (2 * np.pi) ** (d / 2) * np.sqrt(np.linalg.det(sigma[j]))
                p[i, j] = pi[j] * np.exp(exponent) / norm_const
            p[i, :] /= np.sum(p[i, :])

        # Maximization step
        for j in range(k):
            resp_sum = np.sum(p[:, j])
            pi[j] = resp_sum / n
            mu[j] = np.sum(p[:, j][:, None] * observation, axis=0) / resp_sum
            sigma[j] = np.zeros((d, d))
            for i in range(n):
                diff = (observation[i] - mu[j]).reshape(-1, 1)
                sigma[j] += p[i, j] * (diff @ diff.T)
            sigma[j] /= resp_sum
            sigma[j] += 1e-5 * np.eye(d)

        # Compute log-likelihood
        log_likelihood = 0
        for i in range(n):
            temp = 0
            for j in range(k):
                diff = observation[i] - mu[j]
                exponent = -0.5 * diff.T @ np.linalg.inv(sigma[j]) @ diff
                norm_const = (2 * np.pi) ** (d / 2) * np.sqrt(np.linalg.det(sigma[j]))
                temp += pi[j] * np.exp(exponent) / norm_const
            log_likelihood += np.log(temp)
        log_likelihoods.append(log_likelihood)

    return pi, mu, sigma, log_likelihoods

p=3
pi, mu, sigma,log_likehoods = EM_algorithm(samples,p,25)
```

And the table comparing the estimated and real values of the parameters is :

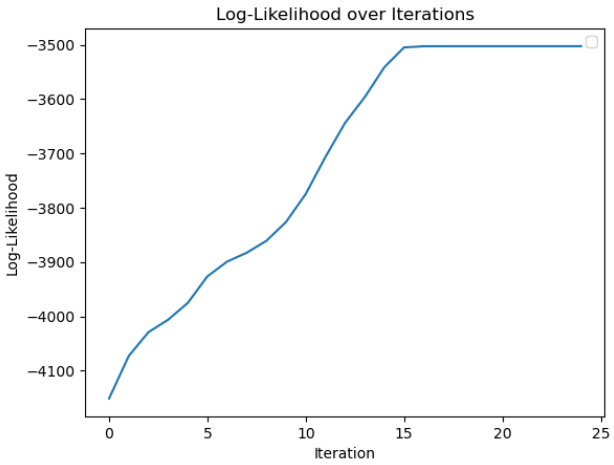

Figure 3: Caption

Param.	**Estimated Values**	**Real Values**
π	$\begin{bmatrix}0.1874\\0.3973\\0.4153\end{bmatrix}$	$\begin{bmatrix}0.4\\0.4\\0.2\end{bmatrix}$
μ	$\begin{bmatrix}-3.0501 & 2.8823\\3.0125 & 2.9671\\0.0328 & 0.0270\end{bmatrix}$	$\begin{bmatrix}0 & 0\\3 & 3\\-3 & 3\end{bmatrix}$
σ	$\begin{bmatrix}0.4920 & -0.0112\\-0.0112 & 0.5165\end{bmatrix}, \begin{bmatrix}1.0093 & -0.7312\\-0.7312 & 1.0364\end{bmatrix}, \begin{bmatrix}0.9344 & 0.4612\\0.4612 & 0.8905\end{bmatrix}$	$\begin{bmatrix}1 & 0.5\\0.5 & 1\end{bmatrix}, \begin{bmatrix}1 & -0.7\\-0.7 & 1\end{bmatrix}, \begin{bmatrix}0.5 & 0\\0 & 0.5\end{bmatrix}$

Table 1: Comparison of Estimated and Real Values for π, μ, and σ.

We can observe that the EM algorithm provides accurate estimates, as the dataset is well-suited to the algorithm. This is because the number of clusters is known, and there is no overlap between the clusters.

4. Here is the data given by the Crude Birth/Death Rate data This dataset appears less suited to the EM algorithm compared to the previous observations, as the exact number of clusters is unknown, which introduces additional uncertainty as we also don't know how much the clusters overlap.

5. We run the tests for values of m from 1 to 5 and compute the BIC for each, and finally draw the PDF for the best one : [h!] We see that the minimum BIC is for 5 classes, hence the result

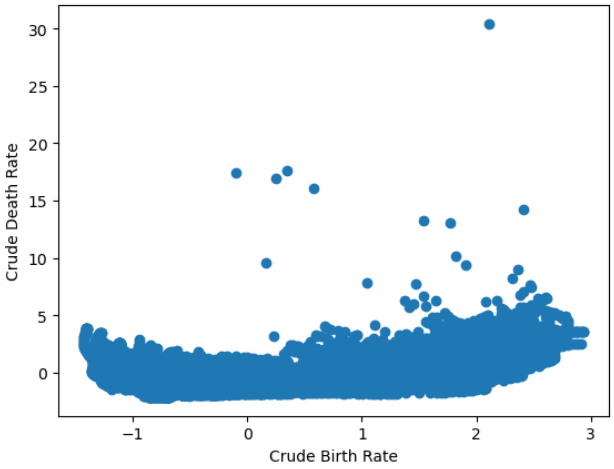

Figure 4: the Crude Birth/Death Rate data

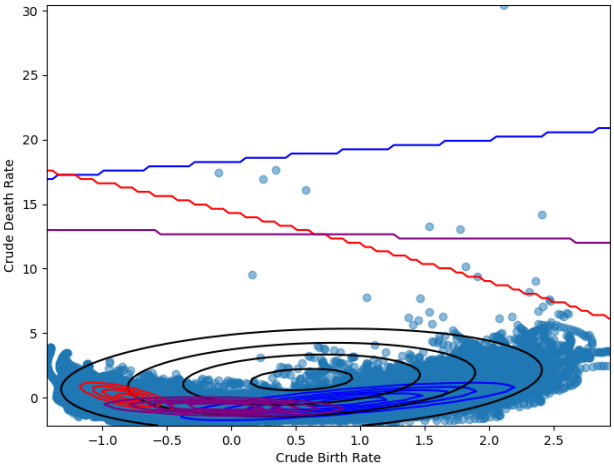

Figure 5: The pdf of GMM

Exercise 3

1. The following code does a simple importance sampling procedure

```
    def f(x):
        return 2 * np.sin((np.pi / 1.5) * x)

    def p(x):
        return x**(1.65 - 1) * np.exp(-x**2 / 2)

    def q(x, mu=0.8, sigma=1.5):
        return (2 / (np.sqrt(2 * np.pi * sigma))) * np.exp(-((x - mu)**2) / (2 * sigma))

def importance_sampling(N_samples) :
    for i in range(N) :
        samp = np.random.normal(0.8,1.5)
        while samp < 0 :
            samp = np.random.normal(0.8,1.5)
    weights_normalized = p(samp)/q(samp) / np.sum(p(samp)/q(samp))
```

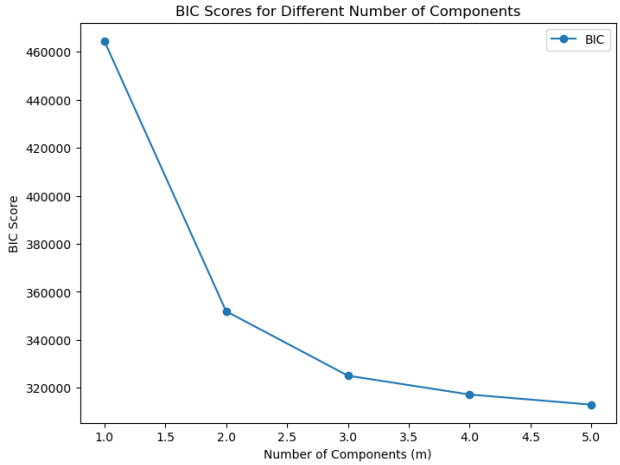

Figure 6: Best BIC

```
    esperance = f(samp) * weights_normalized
    return np.mean(esperance),np.var(esperance)
esperance,variance = importance_sampling(N_samples = 1000)
print("Estimation de E_p[f(X)]:", esperance)
```

2. The results are given in the table :

Number of Samples (N_{samples})	**Estimation of $E_p[f(X)]$**	**Variance of the Estimation**
10	0.0696	0.0332
100	0.0113	0.0004
1000	0.0010	3.75×10^{-6}
10000	0.0001	3.71×10^{-8}

Table 2: Estimation and variance of $E_p[f(X)]$ for different sample sizes for $\mu = 0.8$.

3. When we shift the mean by 6, we get the following : We can see that the variance gets

Number of Samples (N_{samples})	**Estimation of $E_p[f(X)]$**	**Variance of the Estimation**
10	0.1939	0.2338
100	-0.0036	0.0026
1000	0.0010	0.0010
10000	0.0002	0.0003

Table 3: Estimation and variance of $E_p[f(X)]$ for different sample sizes for $\mu = 6$.

much higher (almost 10 times) when we introduce a shift which shows that a poor choice of q significantly affects the quality of importance sampling.

4. The EM algorithm allows to parametrize the maximum log like hood given in the step of (iii) by a succession of expectation and maximization step using the following updates :

- **Update for α_j:**

$$\alpha_j = \frac{\sum_{i=1}^{n} \omega_i^{(0)} r_{ij}}{\sum_{i=1}^{n} \omega_i^{(0)}}.$$

- **Update for μ_j:**

$$\mu_j = \frac{\sum_{i=1}^{n} \omega_i^{(0)} r_{ij} X_i^{(0)}}{\sum_{i=1}^{n} \omega_i^{(0)} r_{ij}}.$$

- **Update for Σ_j:**

$$\Sigma_j = \frac{\sum_{i=1}^{n} \omega_i^{(0)} r_{ij} \left(X_i^{(0)} - \mu_j\right) \left(X_i^{(0)} - \mu_j\right)^\top}{\sum_{i=1}^{n} \omega_i^{(0)} r_{ij}}.$$