DD2434 Assignment 1A

Yonghong Huang, Vianey Darsel

December 14, 2022

Contents

Listings

1 Dependency in a Directed Graphical Model

Question 1.1.1

The paths $Z_{d,n} \leftarrow \Theta_d \rightarrow Z_{d,n+1}$ and $W_{d,n} \leftarrow \beta_{1:K} \rightarrow W_{d,n+1}$ are d-separated by $\{\Theta_d, \beta_{1:K}\}$. So, $W_{d,n}$ and $W_{d,n+1}$ are d-separated by $\{\Theta_d, \beta_{1:K}\}$. So, $W_{d,n} \perp W_{d,n+1} | \Theta_d, \beta_{1:K}$

Question 1.1.2

The path $\Theta_d \leftarrow \alpha \rightarrow \Theta_{d+1}$ is not d-separated by $\{Z_{d,1:N}\}$. So, Θ_d and Θ_{d+1} are not d-separated by $\{Z_{d,1:N}\}$. So, $\Theta_d \not\perp \Theta_{d+1} | Z_{d,1:N}$.

Question 1.1.3

The path $\Theta_k \leftarrow \alpha \rightarrow \Theta_j$ is d-separated by $\{\alpha; Z_{1:D,1:N}\}$. So, Θ_d and Θ_{d+1} are d-separated by $\{\alpha; Z_{1:D,1:N}\}$. So, $\Theta_d \perp \Theta_{d+1} | \alpha; Z_{1:D,1:N}$.

Question 1.1.4

$W_{d,n} \not\perp W_{d,n+1} | \Lambda_d, \beta_{1:K}$

Question 1.1.5

$\Theta_d \not\perp \Theta_{d+1} | Z_{d,1:N}, Z_{d+1,1:N}$

Question 1.1.6

$\Lambda_d \not\perp \Lambda_{d+1} | \Phi, Z_{1:D,1:N}$

2 Likelihood of a Tree Graphical Mode

Question 1.2.7:

To use our code, you just need to run the file "run_file.py" in the 1A_2 folder. In this file, you can change the file to use by changing the value of the index of the list file. The function that we have implemented are object functions so they are in the file "Tree.py". The different methods added to the provided files are :

- Node.isLeaf() : return is the node is a leaf or not
- Node.isRoot() : return is the node is the root or not
- Node.Likelihood($\beta_{1:L}$) : return an array with the likelihood of the observation given the value of the parent node (if any) for all the possible values of the parent. Otherwise, it returns the global likelihood.
- Tree.Likelihood($\beta_{1:L}$) : return the likelihood of an observation (theta is included in the tree)

To implement the algorithm, we have used the following relation :

$$\mathbb{P}(\beta_{1:L}) = \sum_{k \in [K]} \mathbb{P}(\beta_{1:L} | X_{root} = i) \mathbb{P}(X_{root} = i) \tag{1}$$

We have reduced the size of the problem because $\mathbb{P}(\beta_{1:L}|X_{root}=i)$ is the same problem with one less depth. To start from the bottom. We can write :

$$\begin{aligned}\mathbb{P}(\beta_{1:L}|X_{root}=i) &= \sum_{k_i\in[K]^{|\{X_i\in CHILDREN_{X_{root}}\}|}} \mathbb{P}\left(\bigcap_{X_i\in CHILDREN_{X_{root}}} (X_i=k_i), \beta_{1:L}|X_{root}=i\right) \\ &= \sum_{k_i\in[K]^{|\{X_i\in CHILDREN_{X_{root}}\}|}} \mathbb{P}\left(\bigcap_{X_i\in CHILDREN_{X_{root}}} \left(X_i=k_i,\beta_{u(X_i)}\right)|X_{root}=i\right) \\ &= \sum_{k_i\in[K]^{|\{X_i\in CHILDREN_{X_{root}}\}|}} \prod_{X_i\in CHILDREN_{X_{root}}} \mathbb{P}(X_i=k_i,\beta_{u(X_i)}|X_{root}=i) \\ &= \sum_{k_i\in[K]^{|\{X_i\in CHILDREN_{X_{root}}\}|}} \prod_{X_i\in CHILDREN_{X_{root}}} \mathbb{P}(\beta_{u(X_i)}|X_i=k_i)\mathbb{P}(X_i=k_i|X_{root}=i) \\ &= \sum_{k_i\in[K]^{|\{X_i\in CHILDREN_{X_{root}}\}|}} \prod_{X_i\in CHILDREN_{X_{root}}} \alpha_i(X_i=k_i)\mathbb{P}(X_i=k_i|X_{root}=i)\end{aligned} \tag{2}$$

where :

- $\beta_{u(X_i)}$ are the values of the leaves of the tree generated by X_i.
- $\alpha_i(X_i=k_i)=\mathbb{P}(\beta_{u(X_i)}|X_i=k_i)$ and :
  $\alpha_i(X_i=x_i)=\sum_{\text{values of children of } X_i}\prod_{X_j\in textchildren}\alpha_j\mathbb{P}(X_j=x_j|X_i=k_i)$

Is is possible due to the structure of the tree. A β belongs to the tree generated by only one child. We have passed from depth d to depth $d-1$. Thus we can compute the likelihood by starting from the leaves to the root.

The implementation is given by the code below.

```python
class Node:
    def isLeaf(self):
        """return is the node is a leaf or not"""
        return (len(self.descendants) == 0)

    def isRoot(self):
        """return is the node is the root or not"""
        return (self.ancestor is None)

    def Likelihood(self, beta: list[float]):
        if self.isLeaf():
            res = np.zeros(len(self.cat))
            val = int(beta[int(self.name)])
            for i in range(len(self.cat)):
                res[i] = self.cat[i][val]
# if the parent node has i for value, the likelihood for (B_1,...B_n) does not depend
    on other variable
# the likelihood is P(X=val|par=i)
            return res
        else:
            likelihood_generated_tree = np.ones(len(self.cat))
            res = np.zeros(len(self.cat))
            for child in self.descendants:
                likelihood_generated_tree *= child.Likelihood(beta)
# P(children|parent=i) = P(child_1|parent)*P(child_2|parent)*...*P(child_N|parent)
    since children are independent given parent
            if (self.isRoot()):
                return np.sum(self.cat*likelihood_generated_tree)
            else:
                for j in range(len(self.cat)):
                    res[j] = np.sum(self.cat[j]*likelihood_generated_tree)
# P(children| grand-parent=j) = sum_i P(children|parent=i)*P(parent=i|grand-parent =j)
                return res
```

```


class Tree:
    def Likelihood(self, beta: list):
        if (len(beta)!=self.num_nodes):
            raise("Error: wrong size of Beta")
        return self.root.Likelihood(beta)
```
Listing 1: Question 2 implementation

Question 1.2.8:

The results we have obtained with the given data can be read in Table 1.

File	Small Tree	Medium Tree	Large Tree
Sample 0	0.0270	7.06e-18	2.04e-67
Sample 1	0.121	2.18e-18	3.12e-65
Sample 2	0.021	7.83e-19	4.50e-66
Sample 3	0.00763	1.99e-18	6.80e-68
Sample 4	0.00733	2.27e-17	2.06e-68

Table 1: Results of our algorithm with provided data

3 Simple Variational Inference

Question 1.3.9:

The code is implemented in the file VariationalInference_SupportFunction.py. We have chosen that all the variables converge to leave the loop. The update scheme is the following:

$$\begin{aligned}
a^* &= a + \frac{N+1}{2} \\
b^* &= b + \frac{1}{2}\mathbb{E}\left[\sum_{n=1}^{N}(X_n-\mu)^2 + \lambda(\mu-\mu')^2\right] \\
&= b + \frac{\sum_{n=1}^{N} X_n^2 + \lambda\mu'^2 - 2\mu^*\left(\sum_{n=1}^{N} X_n + \lambda\mu'\right) + (N+\lambda)\left(\frac{1}{\tau^*} + (\mu^*)^2\right)}{2} \\
\tau^* &= \mathbb{E}[\tau](N+\lambda) \\
&= \frac{a^*}{b^*}(N+\lambda) \\
\mu^* &= \frac{\lambda\mu' + \sum_{n=1}^{N} x_n}{\lambda+N}
\end{aligned} \tag{3}$$

Question 1.3.10:

$$p(\Theta|\mathcal{D}) = \frac{p(\mathcal{D},\Theta)}{p(\mathcal{D})} = \frac{p(\mathcal{D}|\Theta)p(\Theta)}{p(\mathcal{D})} \propto p(\mathcal{D}|\Theta)p(\Theta) \tag{4}$$

So we only need to compute $p(\mathcal{D}|\Theta)p(\Theta)$, where $\mathcal{D} = \mathbf{X}$ and $\Theta = \{\mu; \tau\}$. Since $\mu|\tau \hookrightarrow \mathcal{N}\left(\mu', \frac{1}{\lambda\tau}\right)$ and $\tau \hookrightarrow \mathcal{G}(a, b)$. So we expect to find $\mu|\tau \hookrightarrow \mathcal{N}\left(\mu^*, \frac{1}{\lambda^*\tau}\right)$ and $\tau \hookrightarrow \mathcal{G}(a^*, b^*)$.

$$\begin{aligned}
ln(p(\mu,\tau|\mathbb{X}) &\overset{\pm}{=} ln(p(\mathbb{X}|\mu,\tau) + ln(p(\mu,\tau)) \\
&= \sum_{n=1}^{N} ln(p(X_n|\mu,\tau) + ln(p(\mu|\tau)) + ln(p(\tau)) \\
&= \sum_{n=1}^{N} \left[\frac{1}{2}ln\left(\frac{\tau}{2\pi}\right) - \frac{\tau}{2}(X_n-\mu)^2\right] + \frac{1}{2}ln\left(\frac{\lambda\tau}{2\pi}\right) - \frac{\lambda\tau}{2}(\mu-\mu')^2 + ln\left(\frac{b^a}{\Gamma(a)}\right) + (a-1)ln(\tau) - b\tau \\
&\overset{\pm}{=} \left[\frac{N}{2}ln(\tau) - \sum_{n=1}^{N}\frac{\tau}{2}(X_n-\mu)^2\right] + \frac{1}{2}ln(\tau) - \frac{\lambda\tau}{2}(\mu^2 - 2\mu\mu' + \mu'^2) + (a-1)ln(\tau) - b\tau \\
&= ln(\tau)\left[\frac{N}{2} + a - 1\right] - \tau\left[\frac{\sum_{n=1}^{N} X_n^2}{2} + b + \frac{\lambda\mu'^2}{2}\right] - \frac{\sum_{n=1}^{N}\tau + \lambda\tau}{2}\mu^2 + \left[\tau\sum_{n=1}^{N} X_n + \lambda\tau\mu'\right]\mu \\
&= ln(\tau)\left[\frac{N}{2} + a - 1\right] - \tau\left[\frac{\sum_{n=1}^{N} X_n^2}{2} + b + \frac{\lambda\mu'^2}{2}\right] - \frac{N\tau + \lambda\tau}{2}\mu^2 + \left[\tau\sum_{n=1}^{N} X_n + \lambda\tau\mu'\right]\mu
\end{aligned} \tag{5}$$

If there is no data ($N = 0$), we only have $ln(p(\Theta))$. So,

$$ln(p(\Theta)) = ln(\tau)\left[a-1\right] - \tau\left(b + \frac{\lambda\mu'^2}{2}\right) - \frac{\lambda\tau}{2}\mu^2 + \lambda\tau\mu'\mu \tag{6}$$

Thus, we can identify the new parameter.

$$\mu|\tau \hookrightarrow \mathcal{N}\left(\mu^*, \frac{1}{\lambda^*\tau}\right)$$

$$\tau \hookrightarrow \mathcal{G}(a^*, b^*)$$

$$\begin{aligned}
\text{with } \lambda^* &= N + \lambda \\
\mu^* &= \frac{\sum_{n=1}^{N} X_n + \lambda\mu'}{N+\lambda} \\
a^* &= \frac{N}{2} + a \\
b^* &= \frac{\sum_{n=1}^{N} X_n^2}{2} + b + \frac{\lambda\mu'^2}{2} - \frac{\lambda^*(\mu^*)^2}{2}
\end{aligned}$$

Question 1.3.11:

We have computed the results from the Variational Inference algorithm and from the Exact Posterior for 3 different orders of magnitude:

- $\lambda >> N$
- $\lambda = N$
- $\lambda << N$

Figure 1 and 2 show that the Variational results are very close from the Posterior. The figures are also similar.

However, Figure 3 presents an even better similarity. The iso-lines are almost overlaid. With an increase of the number of observations, the loss of the Variational techniques is reduced.

Figure 4 shows the difference between the prior distribution and the distribution of the posterior one.

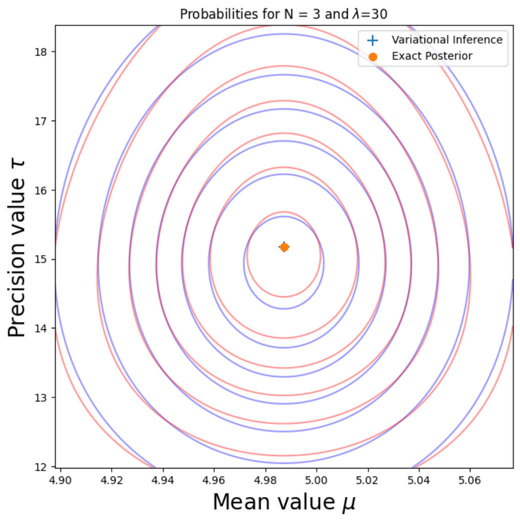

Figure 1: Comparison for $\lambda >> N$

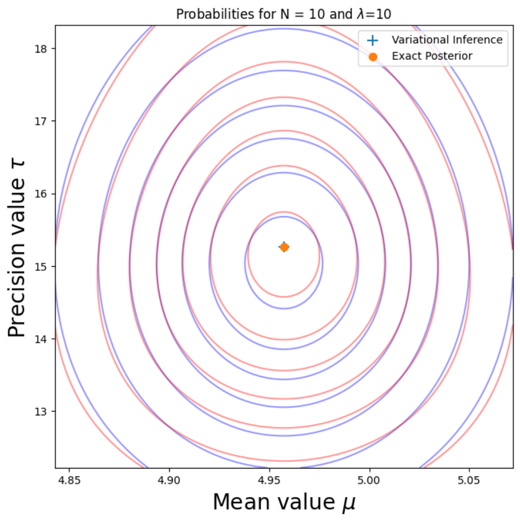

Figure 2: Comparison for $\lambda = N$

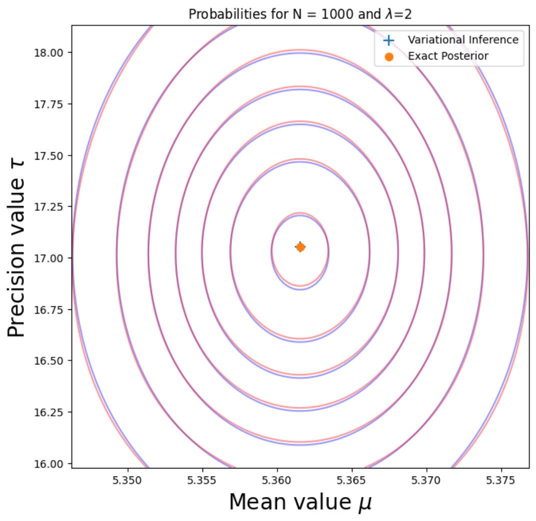

Figure 3: Comparison for $\lambda << N$

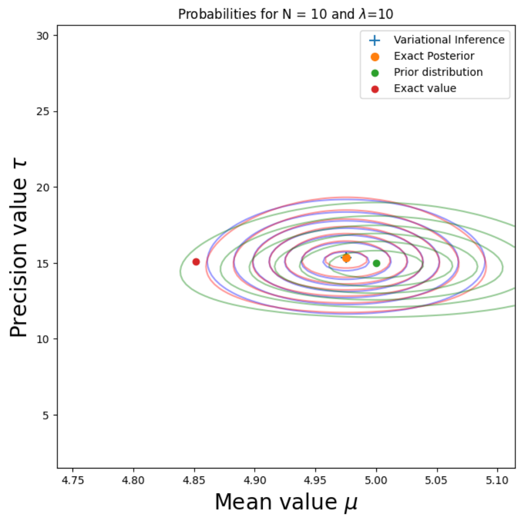

Figure 4: Comparison posterior and prior

The code is given below. Some useful functions to print the results are given in the appendix.

```python
import numpy as np
from math import *
import matplotlib.pyplot as plt
import scipy.stats as scs


def update_mu(mu_prime: float, lambd: float, a_0: float, b_0 : float, X: np.ndarray):
    mu = (lambd*mu_prime+np.sum(X))/(lambd+len(X))
    exeptation_tau = a_0/b_0
    tau = (lambd+len(X))*exeptation_tau
    return mu,tau

def update_tau(mu_prime: float, lambd: float , tau_0: float, mu_0: float, a: float, b : float, X: np.ndarray):
    N=len(X)
    a_1 = a + (N+1)/2
    b_1 = b + 1/2*(np.sum(np.power(X,2))+(np.power(mu_prime,2)*lambd)-2*mu_0*(np.sum(X)+lambd*mu_prime)+(len(X)+lambd)*(1/tau_0+mu_0**2))
    return a_1, b_1


def VI(X: np.ndarray,mu_prime : float,lambd : float,a : float,b : float, threshold: float):
    mu_0, tau_0, a_0, b_0 = 5, 5, 5, 5
    mu_1, tau_1, a_1, b_1 = 0, 1, 1, 1
    while(max(np.power(mu_0-mu_1,2),np.power(tau_0-tau_1,2),np.power(a_0-a_1,2),np.power(b_0-b_1,2))>threshold):
        mu_0, tau_0, a_0, b_0 = mu_1, tau_1, a_1, b_1
        mu_1,tau_1=update_mu(mu_prime,lambd,a_1,b_1,X)
        a_1,b_1 = update_tau(mu_prime,lambd,tau_1,mu_1,a,b,X)
    return mu_0, tau_0, a_0, b_0

def exactPosterior(X: np.ndarray,mu_prime : float,lambd : float,a : float,b : float):
    N=len(X)
    mu_star = (np.sum(X)+lambd*mu_prime)/(N+lambd)
    lambd_star = N+lambd
    a_star = N/2+a
    b_star = np.sum(np.power(X,2))/2+b+lambd*np.power(mu_prime,2)/2-lambd_star*np.power(mu_star,2)/2
    return mu_star,lambd_star,a_star,b_star
```

Listing 2: VI implementation

4 Super Epicentra - Expectation-Maximization

Question 1.4.12:

In this model, we have :

- $Z^n \hookrightarrow Cat(\pi)$
- $X_1^n|Z^n \hookrightarrow \mathcal{N}(\mu_{1,Z^n}, \frac{1}{\tau_{1,Z^n}})$
- $X_2^n|Z^n \hookrightarrow \mathcal{N}(\mu_{2,Z^n}, \frac{1}{\tau_{2,Z^n}})$
- $S^n|Z^n \hookrightarrow \mathcal{P}(\lambda_{Z^n})$

Thus, we can compute $p(X_1^n, X_2^n, S^n, Z^n|\mu, \tau, \pi)$ and $\mathbb{P}(Z^n|X_1^n, X_2^n, S^n, \mu, \tau, \pi)$. On the one hand,

$$\begin{aligned}
p(X_1, X_2, S, Z|\mu, \tau, \lambda, \pi) &= \prod_{n=1}^{N} p(X_1^n, X_2^n, S^n, Z^n|\mu, \tau, \lambda, \pi) \\
&= \prod_{n=1}^{N} p(X_1^n, X_2^n, S^n|Z^n, \mu, \tau, \lambda) p(Z^n|\pi) \\
&= \prod_{n=1}^{N} p(X_1^n|Z^n, \mu_1, \tau_1) p(X_2^n|Z^n, \mu_2, \tau_2) \mathbb{P}(S^n|Z^n, \lambda) p(Z^n|\pi) \qquad (7) \\
&= \prod_{n=1}^{N} \mathcal{N}(\mu_{Z^n,1}, \tau_{Z^n,1}) \mathcal{N}(\mu_{Z^n,2}, \tau_{Z^n,2}) \mathcal{P}(S^n|\lambda_{Z^n}) \pi_{Z^n} \\
&= \prod_{n=1}^{N} \prod_{k=1}^{K} [\mathcal{N}(\mu_{k,1}, \tau_{k,1}) \mathcal{N}(\mu_{k,2}, \tau_{k,2}) \mathcal{P}(S^n|\lambda_k) \pi_k]^{\mathbb{1}_{Z_n = k}}
\end{aligned}$$

We also have

$$\mathbb{P}(Z|X_1, X_2, S, \mu, \tau, \lambda, \pi) = \prod_{n=1}^{N} \mathbb{P}(Z^n|X_1^n, X_2^n, S^n, \mu, \tau, \lambda, \pi)$$

$$\mathbb{P}(Z^n = c|X_1^n, X_2^n, S^n, \mu, \tau, \lambda, \pi) = \frac{p(X_1^n, X_2^n, S^n|Z^n = c, \mu, \lambda, \tau) \mathbb{P}(Z^n = c|\pi)}{\sum_{k=1}^{K} p(X_1^n, X_2^n, S^n, Z^n = k|\mu, \tau, \lambda, \pi)}$$

From Equation 7, we can deduce that :

$$\begin{aligned}
\mathbb{P}(Z^n = c|X_1^n, X_2^n, S^n, \mu, \tau, \lambda, \pi) &= \frac{p(X_1^n, X_2^n, S^n|Z^n = c, \mu, \tau, \lambda) \mathbb{P}(Z^n = c|\pi)}{\sum_{k=1}^{K} p(X_1^n, X_2^n, S^n|Z^n = k, \mu, \tau) \mathbb{P}(Z^n = k|\pi)} \\
&= \frac{\mathcal{N}(X_1^n|\mu_{c,1}, \tau_{c,1}) \mathcal{N}(X_2^n|\mu_{c,2}, \tau_{c,2}) \mathcal{P}(S^n|\lambda_c) \pi_c}{\sum_{k=1}^{K} \mathcal{N}(X_1^n|\mu_{k,1}, \tau_{k,1}) \mathcal{N}(X_2^n|\mu_{k,2}, \tau_{k,2}) \mathcal{P}(S^n|\lambda_k) \pi_k} \qquad (8) \\
A_{n,c,\Theta^{old}} &= \frac{\mathcal{N}(X_1^n|\mu_{c,1}, \tau_{c,1}) \mathcal{N}(X_2^n|\mu_{c,2}, \tau_{c,2}) \mathcal{P}(S^n|\lambda_c) \pi_c}{\sum_{k=1}^{K} \mathcal{N}(X_1^n|\mu_{k,1}, \tau_{k,1}) \mathcal{N}(X_2^n|\mu_{k,2}, \tau_{k,2}) \mathcal{P}(S^n|\lambda_k) \pi_k}
\end{aligned}$$

From Equation 7 and 8, we can compute $\mathbb{E}_{\mathbb{P}(Z|X_1,X_2,S,\mu^{old},\tau^{old},\lambda^{old},\pi^{old})}\left[ln(p(X_1,X_2,S,Z|\mu,\tau,\lambda,\pi))\right]$.

$$\begin{aligned}
E &= \mathbb{E}_{\mathbb{P}(Z|X_1,X_2,S,\mu^{old},\tau^{old},\lambda^{old},\pi^{old})}\left[ln(p(X_1,X_2,S,Z|\mu,\tau,\lambda,\pi))\right] \\
&= \mathbb{E}_{\mathbb{P}(Z|X_1,X_2,S,\mu^{old},\tau^{old},\lambda^{old},\pi^{old})}\left[ln\left(\prod_{n=1}^{N}\mathcal{N}(\mu_{Z^n,1},\tau_{Z^n,1})\mathcal{N}(\mu_{Z^n,2},\tau_{Z^n,2})\mathcal{P}(S^n|\lambda_{Z^n})\pi_{Z^n}\right)\right] \\
&= \mathbb{E}_{\mathbb{P}(Z|X_1,X_2,S,\mu^{old},\tau^{old},\lambda^{old},\pi^{old})}\left[\sum_{n=1}^{N} ln\left(\mathcal{N}(\mu_{Z^n,1},\tau_{Z^n,1})\mathcal{N}(\mu_{Z^n,2},\tau_{Z^n,2})\mathcal{P}(S^n|\lambda_{Z^n})\pi_{Z^n}\right)\right] \\
&= \sum_{n=1}^{N}\mathbb{E}_{\mathbb{P}(Z|X_1,X_2,S,\mu^{old},\tau^{old},\lambda^{old},\pi^{old})} ln\left(\mathcal{N}(\mu_{Z^n,1},\tau_{Z^n,1})\mathcal{N}(\mu_{Z^n,2},\tau_{Z^n,2})\mathcal{P}(S^n|\lambda_{Z^n})\pi_{Z^n}\right) \\
&= \sum_{n=1}^{N}\mathbb{E}_{\mathbb{P}(Z^n|X_1^n,X_2^n,S^n,\mu^{old},\tau^{old},\lambda^{old},\pi^{old})} ln\left(\mathcal{N}(\mu_{Z^n,1},\tau_{Z^n,1})\mathcal{N}(\mu_{Z^n,2},\tau_{Z^n,2})\mathcal{P}(S^n|\lambda_{Z^n})\pi_{Z^n}\right) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K}\mathbb{P}(Z^n=k|X_1^n,X_2^n,S^n,\mu^{old},\tau^{old},\lambda^{old},\pi^{old}) ln\left(\mathcal{N}(\mu_{k,1},\tau_{k,1})\mathcal{N}(\mu_{k,2},\tau_{k,2})\mathcal{P}(S^n|\lambda_k)\pi_k\right) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K}A_{n,k,\Theta^{old}}\left[ln(\mathcal{N}(\mu_{k,1},\tau_{k,1}))+ln(\mathcal{N}(\mu_{k,2},\tau_{k,2}))+ln(\mathcal{P}(S^n|\lambda_k))+ln(\pi_k)\right] \\
&\overset{+}{=} \sum_{n=1}^{N}\sum_{k=1}^{K}A_{n,k,\Theta^{old}}\left[\frac{1}{2}ln(\tau_{k,1})-\frac{\tau_{k,1}}{2}(X_1^n-\mu_{k,1})^2+\frac{1}{2}ln(\tau_{k,2})-\frac{\tau_{k,2}}{2}(k_2-\mu_{k,2})^2\right. \\
&\left.-\lambda_k+S_n ln(\lambda_k)+ln(\pi_k)\right] \\
&= \sum_{k=1}^{K}\left[\frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{2}ln(\tau_{k,1})-\tau_{k,1}\frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{2}(X_1^n-\mu_{k,1})^2+\frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{2}ln(\tau_{k,2})\right. \\
&\left.-\tau_{k,2}\frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{2}(X_2^n-\mu_{k,2})^2-\sum_{n=1}^{N}A_{n,k,\Theta^{old}}\lambda_k+\sum_{n=1}^{N}A_{n,k,\Theta^{old}}S_n ln(\lambda_k)+\sum_{n=1}^{N}A_{n,k,\Theta^{old}}ln(\pi_k)\right]
\end{aligned} \tag{9}$$

We need to add a constraint on π_k since $\sum_{k=1}^{K}\pi_k=1$. So we finally have:

$$\begin{aligned}
E = \sum_{k=1}^{K}&\left[\frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{2}ln(\tau_{k,1})-\tau_{k,1}\frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{2}(X_1^n-\mu_{k,1})^2+\frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{2}ln(\tau_{k,2})\right. \\
&-\tau_{k,2}\frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{2}(X_2^n-\mu_{k,2})^2-\sum_{n=1}^{N}A_{n,k,\Theta^{old}}\lambda_k+\sum_{n=1}^{N}A_{n,k,\Theta^{old}}S_n ln(\lambda_k) \\
&\left.+\sum_{n=1}^{N}A_{n,k,\Theta^{old}}ln(\pi_k)\right]+\beta\left(\sum_{k=1}^{K}\pi_k=1\right)
\end{aligned} \tag{10}$$

Now we have to maximize E (Equation 10) with respect to each parameter.

$$\begin{aligned}
\frac{\partial E}{\partial \pi_k} &= 0 \\
&= \frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{\pi_k}+\beta \\
\Leftrightarrow \pi_k &= -\frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{\beta}
\end{aligned}$$

So $\pi_k \propto \sum_{n=1}^{N}A_{n,k,\Theta^{old}}$; so :

$$\pi_k = \frac{\sum_{n=1}^{N}A_{n,k,\Theta^{old}}}{\sum_{c=1}^{K}\sum_{n=1}^{N}A_{n,c,\Theta^{old}}} \tag{11}$$

$$\begin{aligned}
\frac{\partial E}{\partial \mu_{k,1}} &= 0 \\
&= \tau_{k,1} \sum_{n=1}^{N} A_{n,k,\Theta^{old}} \left(X_1^n - \mu_{k,1}\right) \\
\Leftrightarrow \sum_{n=1}^{N} A_{n,k,\Theta^{old}} \mu_{k,1} &= \sum_{n=1}^{N} A_{n,k,\Theta^{old}} X_1^n \\
\Leftrightarrow \qquad \mu_{k,1} &= \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} X_1^n}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial E}{\partial \tau_{k,1}} &= 0 \\
&= \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}}{2\tau_{k,1}} - \sum_{n=1}^{N} \frac{A_{n,k,\Theta^{old}}}{2} \left(X_1^n - \mu_{k,1}\right)^2 \\
\Leftrightarrow \tau_{k,1} \left[\sum_{n=1}^{N} A_{n,k,\Theta^{old}} \left(X_1^n - \mu_{k,1}\right)^2 \right] &= \sum_{n=1}^{N} A_{n,k,\Theta^{old}} \\
\Leftrightarrow \qquad \tau_{k,1} &= \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} \left(X_1^n - \mu_{k,1}\right)^2} \\
\Leftrightarrow \qquad \tau_{k,1} &= \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} \left(X_1^n - \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} X_1^n}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}}\right)^2}
\end{aligned}$$

With the symmetry of the roles we have:

$$\begin{aligned}
\mu_{k,1} &= \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} X_1^n}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}} \\
\tau_{k,1} &= \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} \left(X_1^n - \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} X_1^n}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}}\right)^2} \\
\mu_{k,2} &= \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} X_2^n}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}} \\
\tau_{k,2} &= \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} \left(X_2^n - \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} X_2^n}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}}\right)^2}
\end{aligned} \tag{12}$$

$$\begin{aligned}
\frac{\partial E}{\partial \lambda_k} &= 0 \\
&= -\sum_{n=1}^{N} A_{n,k,\Theta^{old}} + \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} S_n}{\lambda_k} \\
\Leftrightarrow \lambda_k \sum_{n=1}^{N} A_{n,k,\Theta^{old}} &= \sum_{n=1}^{N} A_{n,k,\Theta^{old}} S_n
\end{aligned}$$

So :

$$\lambda_k = \frac{\sum_{n=1}^{N} A_{n,k,\Theta^{old}} S_n}{\sum_{n=1}^{N} A_{n,k,\Theta^{old}}} \tag{13}$$

From Equation 11,12 and 13, we can compute the maximization step of the EM algorithm.

Question 1.4.13:

The algorithm is implemented in the functions below. Some useful functions that were used to produce the results of next question are given in appendix.

```python
import numpy as np
import scipy.stats as scs
from math import *
import matplotlib.pyplot as plt

def initiate(k: int, X_1:np.ndarray,X_2: np.ndarray,S: np.ndarray):
    '''Initiailisation by taking points as far as possible the one from the other'''
    ids = []
    ids.append(np.argmax(X_1))
    for j in range(k-1):
        ids.append(np.argmax([np.sum(np.array([(X_1[i]-X_1[id])**2+(X_2[i]-X_2[id])
    **2+(S[i]-S[id])**2 for id in ids])) for i in range(len(X_1)) if i not in ids]))
    mu_1 = np.array([X_1[id] for id in ids])
    mu_2 = np.array([X_2[id] for id in ids])
    tau_1 = np.ones(k)
    tau_2 = np.ones(k)
    lambd = np.array([S[id]+1 for id in ids])
    pi = np.ones(k)/k
    return mu_1,mu_2,tau_1,tau_2,lambd,pi


def E_step(K: int,X_1: np.ndarray,X_2: np.ndarray, S: np.ndarray, previous_mu_1: np.
    ndarray,previous_mu_2: np.ndarray,previous_tau_1: np.ndarray,previous_tau_2: np.
    ndarray,previous_lambda: np.ndarray,previous_pi: np.ndarray):
    '''
    E-step
    '''
    E = np.zeros((len(X_1),K))
    log_likelihood = 1
    for n in range(len(X_1)):
        for k in range(K):
            p_X_1 = scs.norm(previous_mu_1[k],np.sqrt(1/previous_tau_1[k])).pdf(X_1[n
    ])
            p_X_2 = scs.norm(previous_mu_2[k],np.sqrt(1/previous_tau_2[k])).pdf(X_2[n
    ])
            p_S = scs.poisson(previous_lambda[k]).pmf(S[n])
            p_Z = previous_pi[k]
            E[n][k]=p_X_1*p_X_2*p_S*p_Z
        log_likelihood += np.log(np.sum(E[n]))
        E[n] = E[n]/np.sum(E[n]) #normalize
    return E,log_likelihood

def M_step(K: int,X_1: np.ndarray,X_2: np.ndarray, S: np.ndarray, A : np.ndarray):
    mu_1 = np.zeros(K)
    mu_2 = np.zeros(K)
    tau_1 = np.zeros(K)
    tau_2 = np.zeros(K)
    lambd = np.zeros(K)
    pi = np.zeros(K)
    for k in range(K): #maximization / begin M-step
        if(np.sum(A[k])<1e-5):
            A[k] = A[k]/np.sum(A[k])
        pi[k]=np.sum(A[k])
        mu_1[k] = np.sum(A[k]*X_1)/np.sum(A[k])
        mu_2[k] = np.sum(A[k]*X_2)/np.sum(A[k])
        tau_1[k] = min(1e4,np.sum(A[k])/max(1e-8,np.sum(A[k]*np.power(X_1-mu_1[k],2)))
    )
        tau_2[k] = min(1e4,np.sum(A[k])/max(1e-8,np.sum(A[k]*np.power(X_2-mu_2[k],2)))
    )
        lambd[k] = np.sum(A[k]*S)/(np.sum(A[k]))
    pi = pi/np.sum(pi)
    return mu_1,mu_2,tau_1,tau_2,lambd,pi


def EM(threshold: float, K: int, X_1: np.ndarray,X_2: np.ndarray, S: np.ndarray):
    '''Full EM algorithm'''
    N= len(X_1)
    mu_1,mu_2,tau_1,tau_2,lambd,pi = initiate(K,X_1,X_2,S)
    # previous_mu_1,previous_mu_2,previous_tau_1,previous_tau_2,previous_lambda,
    previous_pi = mu_1+1,mu_2,tau_1*3,tau_2*3,lambd,pi*3
    # loop=1
    log_like = 0
    change = True
    while(change):
        # loop+=1
        # previous_mu_1,previous_mu_2,previous_tau_1,previous_tau_2,previous_lambda,
    previous_pi = mu_1.copy(),mu_2.copy(),tau_1.copy(),tau_2.copy(),lambd.copy(),pi.
    copy()
        A,new_log_like = E_step(K,X_1,X_2,S,mu_1,mu_2,tau_1,tau_2,lambd,pi)
        if (np.abs(new_log_like-log_like)<threshold):
            change=False
        log_like = new_log_like
        A = A.transpose()
        mu_1,mu_2,tau_1,tau_2,lambd,pi = M_step(K,X_1,X_2,S,A)
    return mu_1,mu_2,tau_1,tau_2,lambd,pi

```

Listing 3: EM implementation

Question 1.4.14:

The algorithm is very sensitive to the initial values and to extreme points. Thus, it happens that a "cluster" k contains only one value. That leads to an issue: the precision will be very high (close to infinity), so we could have computation errors. To avoid them, we have set a limit to the precision.

The observation of data in 3 dimensions from Figure 5 seems to show that 3 is a good number for the number of different "clusters" or for K.

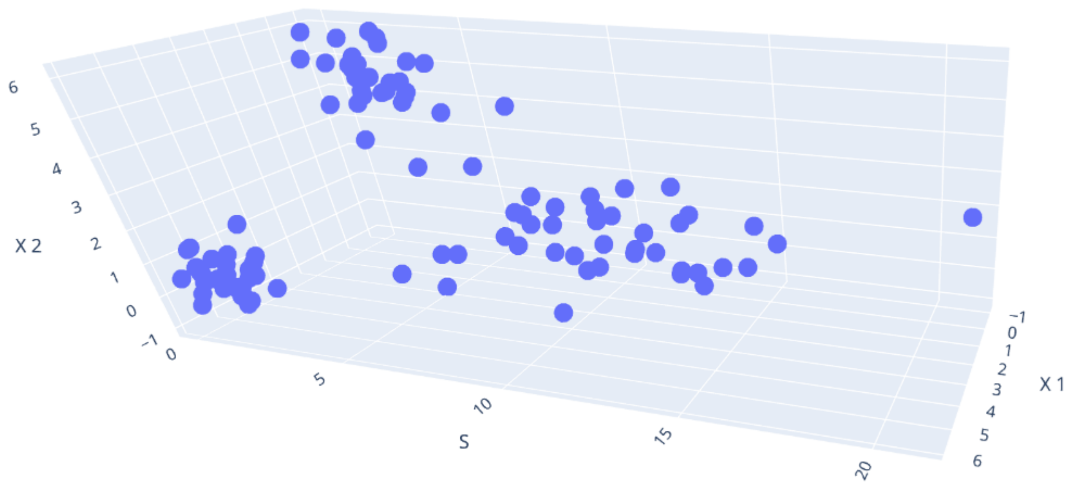

Figure 5: Original distribution of data

To validate our hypothesis, we have used the BIC criteria. We have computed the EM algorithm for different values of K and the best value is the one that minimizes the BIC value ($BIC = k_{parameters} ln(n) - 2ln(p(X|\Theta))$).

Figure 6 shows that our hypothesis is validated. The EM algorithm (initialized with maximal distance values) gives the output (Figure 7).

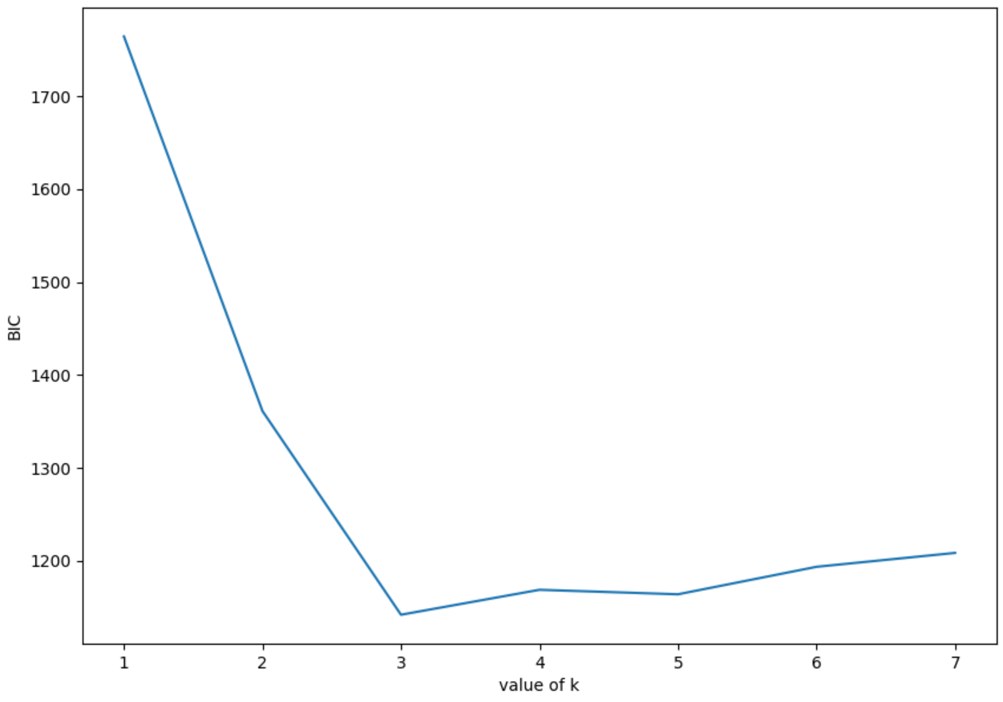

Figure 6: BIC according to the value of K

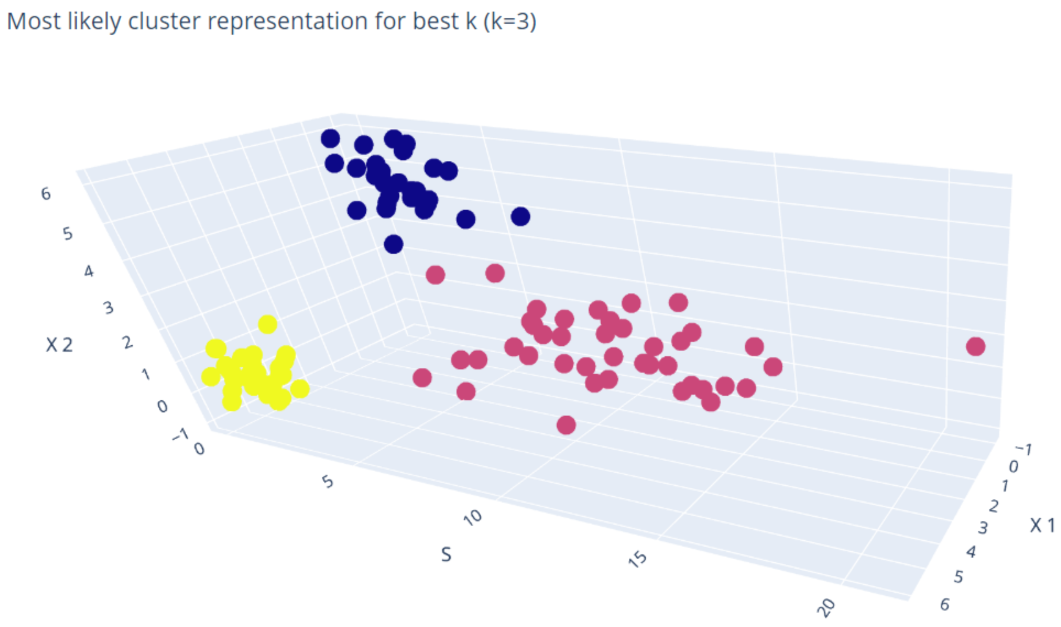

Figure 7: Distribution of the clusters for $k = 3$

However, with a larger number of clusters, it detects individual behaviors, as Figure 8.

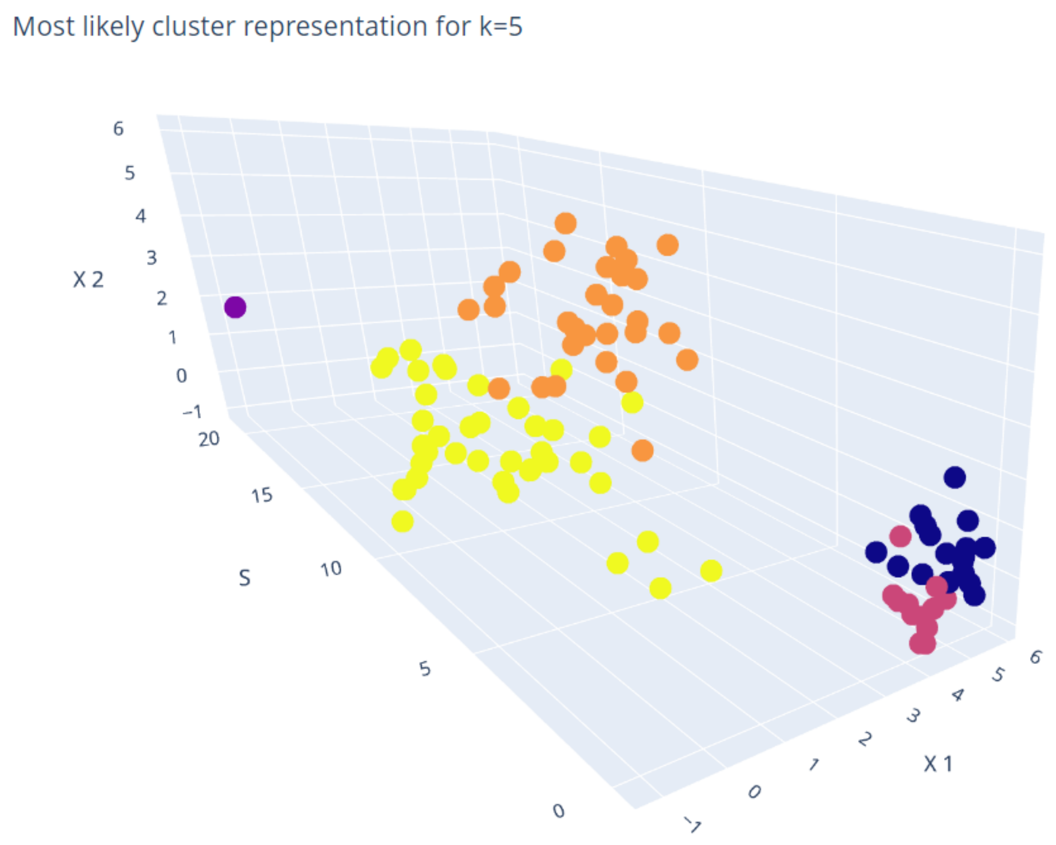

Figure 8: Distribution of the clusters for $k = 5$

Appendix

Useful functions for Question 1.3

```python
def generate_sample(N,mu_prime,lambd,a,b):
    T = np.random.gamma(a,1/b)
    m = np.random.normal(mu_prime,1/np.sqrt(lambd*T))
    X= np.random.normal(m,1/np.sqrt(T),N)
    return T,m,X


def print_comparison(val_VI: list, val_post: list, mu_prime : float,lambd : float,a : float,b : float,mu_val:float, tau_val:float, N: int, prior=False):
    x_mu = np.linspace(val_post[0]-2*np.sqrt(val_post[3]/(val_post[1]*val_post[2])), val_post[0]+2*np.sqrt(val_post[3]/(val_post[1]*val_post[2])),100)
    y_tau = np.linspace(max(val_post[2]/val_post[3]*(1-2/(np.sqrt(lambd*N))),0.01), val_post[2]/val_post[3]*(1+2/(np.sqrt(lambd*N))),100)
    if (prior):
        x_mu= np.linspace(min(val_post[0],mu_prime,mu_val)-2*np.sqrt(val_post[3]/(val_post[1]*val_post[2])),max(val_post[0],mu_prime,mu_val)+2*np.sqrt(val_post[3]/(val_post[1]*val_post[2])),100)
        y_tau = np.linspace(max(min(a/b/10,tau_val,val_post[2]/val_post[3]/5),0.05), max(a/b,val_post[2]/val_post[3]*2),100)
    XX, YY = np.meshgrid(x_mu,y_tau)

    q_tau = scs.gamma.pdf(x=y_tau,a=val_VI[2],scale=1/val_VI[3])
    q_mu  = scs.norm(val_VI[0],1/np.sqrt(val_VI[1])).pdf(x_mu)
    M = q_mu*np.transpose([q_tau])

    p_tau = scs.gamma.pdf(YY,a=val_post[2],scale=1/val_post[3])
    p_mu = scs.norm(val_post[0],np.sqrt(1/(val_post[1]*YY))).pdf(XX)
    M2 = p_tau*p_mu

    pp_tau = scs.gamma.pdf(YY,a=a,scale=1/b)
    pp_mu = scs.norm(mu_prime,1/np.sqrt(lambd*YY)).pdf(XX)
    M3 = pp_tau*pp_mu

    plt.figure(figsize=(10,8))
    plt.contour(XX,YY,M,colors='b',alpha=0.4)
    plt.contour(XX,YY,M2,colors='r',alpha=0.4)
    plt.scatter(val_VI[0],val_VI[2]/val_VI[3], marker="+",s=100, label="Variational Inference")
    plt.scatter(val_post[0],val_post[2]/val_post[3],s=50, label="Exact Posterior")
    if (prior):
        plt.contour(XX,YY,M3,colors='g',alpha=0.4)
        plt.scatter(mu_prime,a/b, label="Prior distribution")
        plt.scatter(mu_val,tau_val, label="Exact value")
    plt.legend(fontsize=10)
    plt.xlabel(r"Mean value $\mu$", fontsize=20)
    plt.ylabel(r"Precision value $\tau$", fontsize=20)
    plt.title(r"Probabilities for N = "+str(N)+" and $\lambda$="+str(round(lambd,2)))
    plt.xlim(x_mu[0],x_mu[-1])
    plt.ylim(y_tau[0],y_tau[-1])
    plt.show()
```

Useful functions for Question 1.4

```python
def read_files(X_file_name: str, S_file_name: str):
    X_file ="data\\"+ X_file_name
    S_file ="data\\"+ S_file_name
    with open(X_file) as file:
        X = file.readlines()
    X_1 = np.zeros(len(X))
    X_2 = np.zeros(len(X))
    for i,line in enumerate(X):
        X_1[i]=float(line.split(" ")[0])
        X_2[i]=float(line.split("\n")[0].split(" ")[1])
    with open(S_file) as file:
        S_temp = file.readlines()
    S = np.zeros(len(S_temp))
    for i,line in enumerate(S_temp):
        S[i]=float(line.split("\n")[0])
    return X_1, X_2, S

def proba_point(x_1,x_2,s,mu_1,mu_2,tau_1,tau_2,lambd,pi):
    k = len(mu_1)
    res = np.zeros(k)
    for i in range(k):
        res[i] = pi[i] * scs.norm(loc=mu_1[i],scale=np.sqrt(1/tau_1[i])).pdf(x_1) *
    scs.norm(loc=mu_2[i],scale=1/tau_2[i]).pdf(x_2) * scs.poisson(mu=lambd[i]).pmf(s)
    return res

def affectation (X_1 : np.ndarray, X_2 : np.ndarray, S : np.ndarray,
                    mu_1 : np.ndarray, mu_2 : np.ndarray, tau_1 : np.ndarray, tau_2 :
    np.ndarray,
                    lambd : np.ndarray, pi : np.ndarray):
    res = np.zeros(len(X_1))-1
    proba = np.zeros(len(X_1))-1
    for i in range(len(X_1)):
        temp = proba_point(X_1[i],X_2[i],S[i],mu_1,mu_2,tau_1,tau_2,lambd,pi)
        proba[i] = np.sum(temp)
        res[i] = np.argmax(temp)
    return res

def main():
    X_1,X_2,S = read_files("X.txt","S.txt")
    fig = go.Figure(data=go.Scatter(x=X_1,y=X_2,mode="markers", marker={"color":S,'
    coloraxis':'coloraxis'}),
                 layout={"xaxis_title":"X 1","yaxis_title":"X 2","legend_title":"S", "
    title":"Distribution", "coloraxis_colorbar_title_text":"S"})
    fig.write_html('example.html', auto_open=True)
    fig = go.Figure(data=go.Scatter3d(x=X_1,y=S,z=X_2,mode="markers"))
                # layout={"xaxis_title":,"yaxis_title":"S","zaxis_title":"X 2", "title
    ":"Distribution"})
    fig.update_layout(scene = dict(xaxis_title = "X 1",yaxis_title = "S",zaxis_title =
     "X 2"),title="Distribution")
    fig.write_html('example2.html', auto_open=True)
    max_val = 8
    val = np.arange(1,max_val)
    BIC= np.zeros(len(val))
    for k in val:
        mu_1,mu_2,tau_1,tau_2,lambd,pi = EM(0.1,k,X_1,X_2,S)
        proba = np.zeros(len(X_1))-1
        for i in range(len(X_1)):
            temp = proba_point(X_1[i],X_2[i],S[i],mu_1,mu_2,tau_1,tau_2,lambd,pi)
            proba[i] = np.sum(temp)
        BIC[k-val[0]] = (6*k)*np.log(len(X_1))- 2*np.sum(np.log(proba))
    plt.figure(figsize=(10,7))
    plt.plot(val,BIC)
    plt.xlabel("value of k")
    plt.ylabel("BIC")
    plt.show()
    k = np.argmin(BIC)+1
    mu_1,mu_2,tau_1,tau_2,lambd,pi = EM(0.05,k,X_1,X_2,S)
    aff = affectation(X_1,X_2,S,mu_1,mu_2,tau_1,tau_2,lambd,pi)
    fig = go.Figure(data=go.Scatter3d(x=X_1,y=S,z=X_2,mode="markers",marker={"color":
    aff, 'coloraxis':'coloraxis'} ),)
    fig.update_layout(scene = dict(xaxis_title = "X 1",yaxis_title = "S",zaxis_title =
     "X 2"),title="Most likely cluster representation for best k (k="+str(k)+")")
    fig.write_html('example.html', auto_open=True)

if __name__ == "__main__":
    main()
```