DD2434 Advanced Machine Learning
2021-2022

Assignment 2: Probabilistic Graphical Models, Variational Inference and Expectation-Maximization Algorithms

---

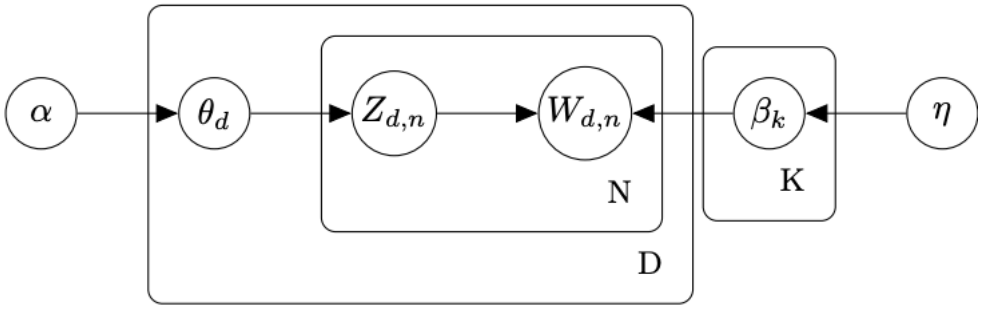
Martín Iglesias Goyanes
martinig@kth.se

School of Electrical Engineering and Computer Science
Stockholm, 20 November 2021

2.1 Dependencies in a Directed Graphical Model

Consider the graphical models shown in Figures 1 and 2. You merely have to answer "yes" or "no" to each question.

Figure 1: Graphical model of smoothed LDA



Question 2.1.1

In the graphical model of Figure 1, is $W_{d,n} \perp W_{d,n+1} \mid \theta_d, \beta_{1:K}$?

Yes, it holds that $W_{d,n} \perp W_{d,n+1} \mid \theta_d, \beta_{1:K}$

Question 2.1.2

In the graphical model of Figure 1, is $\theta_d \perp \theta_{d+1} \mid Z_{d,1:N}$?

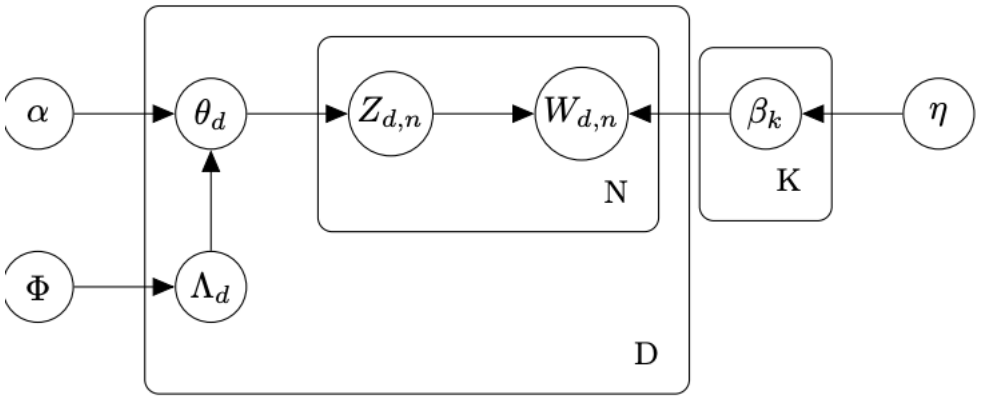
No, it holds that $\theta_d \not\perp \theta_{d+1} \mid Z_{d,1:N}$

Question 2.1.3

In the graphical model of Figure 1, is $\theta_d \perp \theta_{d+1} \mid \alpha, Z_{1:D,1:N}$?

Yes, it holds that $\theta_d \perp \theta_{d+1} \mid \alpha, Z_{1:D,1:N}$

Figure 2: Graphical model of Labeled LDA



Question 2.1.4

In the graphical model of Figure 2, is $W_{d,n} \perp W_{d,n+1} \mid \lambda_d, \beta_{1:K}$?

No, it holds that $W_{d,n} \not\perp W_{d,n+1} \mid \lambda_d, \beta_{1:K}$

Question 2.1.5

In the graphical model of Figure 2, is $\theta_d \perp \theta_{d+1} \mid Z_{d,1:N}, Z_{d+1,1:N}$?

No, it holds that $\theta_d \not\perp \theta_{d+1} \mid Z_{d,1:N}, Z_{d+1,1:N}$

Question 2.1.6

In the graphical model of Figure 2, is $\Lambda_d \perp \Lambda_{d+1} \mid \Phi, Z_{1:D,1:N}$?

No, it holds that $\Lambda_d \not\perp \Lambda_{d+1} \mid \Phi, Z_{1:D,1:N}$

2.2 Likelihood of a Tree Graphical Model

Let T be a rooted binary tree, with vertex set $V(T)$ and leaf set $L(T)$, and consider the graphical model T, Θ described as follows. For each vertex $v \in V(T)$ there is an associated random variable X_v that assumes values in $[K]$. Moreover, for each $v \in V(T)$, the CPD $\theta_v = p\left(X_v \mid x_{\mathrm{pa}(v)}\right)$ is a categorical distribution. Let $\beta = \{x_l : l \in L(T)\}$ be an assignment of values to all the leaves of T.

Question 2.2.7

Implement a dynamic programming algorithm that, for a given T, Θ and β, computes $p(\beta \mid T, \Theta)$.

To compute $p(\beta \mid T, \Theta)$ efficiently we need to find an expression of it that allows us to divide the computation in smaller sub-computations, i.e, dynamic programming. Using Bayes' Theorem we can marginalize in the following manner:

$$p(\beta \mid T, \Theta) = \sum_i p\left(X_{r \cap \beta}, X_r = i \mid T, \Theta\right) = \sum_i p\left(X_{r \cap \beta} \mid X_r = i, T, \Theta\right) p\left(X_r = i\right)$$

where X_r is the value of the root node. Now, let $s(v, i) = p\left(X_{O \cap \downarrow v} \mid X_{pa(v)} = i, T, \Theta\right)$ for any vertex v, where $\{O \cap \downarrow v\}$ are the observed values of the children of node v. Then we have that:

1. At the root layer, r:
$$p(\beta \mid T, \Theta) = \sum_i s(r, i) p\left(X_r = i\right)$$

2. We know that T is a binary tree, i.e, each node has 2 children: w and u. Then, at any node v between root and leaf nodes:
$$\begin{aligned} s(v, i) &= p\left(X_{O \cap \downarrow v} \mid X_v = i\right) = p\left(X_{O \cap \downarrow u} \mid X_v = i\right) p\left(X_{O \cap \downarrow w} \mid X_v = i\right) \\ &= \left(\sum_j s(u, j) p\left(X_u = j \mid x_v = i\right)\right)\left(\sum_j s(w, j) p\left(X_w = j \mid x_v = i\right)\right) \end{aligned}$$

3. Finally, we have as base case the leaf nodes l which are the observed ones:
$$s(l, i) = \begin{cases} 1, & X_l = i \\ 0, & \text{otherwise} \end{cases}$$

This manner of expressing $p(\beta \mid T, \Theta)$ is very convenient since we can calculate it in a dynamic programming fashion by working up the computations from the leaf nodes until the root of the tree. See code in Appendix 2: Tree Code.

Question 2.2.8

Report $p(\beta \mid T, \Theta)$ for each given data, separately. You should report 15 values in total; 5 for small tree, 5 for medium tree and 5 for large tree.

See results in tables 1, 2 and 3.

Sample	Likelihood
0	0.009786075668602368
1	0.015371111945909397
2	0.02429470256988136
3	0.005921848333806081
4	0.016186321212555956

Table 1: Small tree results.

Sample	Likelihood
0	1.7611947348905713e-18
1	2.996933026124685e-18
2	2.891411201505415e-18
3	4.6788419411270006e-18
4	5.664006737201378e-18

Table 2: Medium tree results.

Sample	Likelihood
0	3.63097513075208e-69
1	3.9405421986921234e-67
2	5.549061147187144e-67
3	9.89990102807915e-67
4	3.11420969368965e-72

Table 3: Large tree results.

2.3 Simple Variational Inference

Consider the model defined by Equation (10.21)-(10-23) in Bishop. We are here concerned with the VI algorithm for this model covered during the lectures and in the book.

Question 2.3.9

Implement the VI algorithm for the variational distribution in Equation (10.24) in Bishop.

We can follow the derivation explained in Bishop, [1]. Given a data set $D = \{x_1, \ldots, x_N\}$ of observed values x that are drawn from a Gaussian distribution, $p(D \mid \mu, \tau)$, we want to infer a posterior distribution, $p(\mu, \tau \mid D)$, using variational inference. The likelihood function is :

$$p(D \mid \mu, \tau) = \left(\frac{\tau}{2\pi}\right)^{N/2} \exp\left\{-\frac{\tau}{2}\sum_{n=1}^{N}(x_n - \mu)^2\right\}$$

The prior distributions for μ and τ are:

$$p(\mu \mid \tau) = \mathcal{N}\left(\mu; \mu_0, (\lambda_0 \tau)^{-1}\right)$$
$$p(\tau) = \Gamma\left(\tau; a_0, b_0\right)$$

where $\mu_0, \lambda_0, a_0, b_0$ are hyperparameters.
The variational approximation of the posterior is factorized and results in:

$$q(\mu, \tau) = q_\mu(\mu) q_\tau(\tau)$$

The factorized approximated distributions are computed in Bishop, [1], which result to be:

$$q_\mu(\mu) = \mathcal{N}\left(\mu; \mu_N, \lambda_N^{-1}\right)$$
$$q_\tau(\tau) = \Gamma\left(\tau; a_N, b_N\right)$$

These approximated distributions are updated and improved by the updates of their parameters, which are updated in the order below until convergence, i.e, change in the parameters update is below a desired threshold.

$$\mu_N = \frac{\lambda_0 \mu_0 + N\bar{x}}{\lambda_0 + N}$$
$$\lambda_N = (\lambda_0 + N)\, E_\tau[\tau] = (\lambda_0 + N)\, \frac{a_N}{b_N}$$
$$a_N = a_0 + \frac{N}{2}$$
$$b_N = b_0 + \frac{1}{2} E_\mu \left[\sum_{n=1}^{N} (x_n - \mu)^2 + \lambda_0 (\mu - \mu_0)^2\right]$$
$$= b_0 + \frac{1}{2}\left[(\sum_{n=1}^{N} x_n^2) + N E_\mu\left[\mu^2\right] - 2N\bar{x} E_\mu[\mu] + \lambda_0\left(E_\mu\left[\mu^2\right] + \mu_0^2 - 2\mu_0 E_\mu[\mu]\right)\right]$$
$$= \left\{E_\mu[\mu] = \mu_N, E_\mu\left[\mu^2\right] = V_\mu(\mu) + E_\mu[\mu]^2 = \lambda^{-1} + \mu_N^2\right\}$$
$$= b_0 + \frac{1}{2}\left[(\sum_{n=1}^{N} x_n^2) + (N + \lambda_0)\left(\lambda^{-1} + \mu_N^2\right) - 2\mu_N\left(N\bar{x} + \mu_0\lambda_0\right) + \lambda_0\mu_0^2\right]$$

See code in Appendix 3: Variational Inference Code.

Question 2.3.10

What is the exact posterior?

The posterior we are after is $p(\mu, \tau \mid D)$. From Bayes' theorem we know that:

$$p(\mu, \tau \mid D) = \frac{p(D \mid \mu, \tau) p(\mu \mid \tau) p(\tau)}{p(D)} \propto p(D \mid \mu, \tau) p(\mu \mid \tau) p(\tau)$$

Both priors μ and τ are conjugate priors of the likelihood and therefore, will make the posterior

be Gaussian-Gamma distribution [2]:

$$
\begin{aligned}
p(\mu, \tau \mid D) &\propto p(D \mid \mu, \tau) p(\mu \mid \tau) p(\tau) \\
&\propto \tau^{\frac{N}{2}} \tau^{\frac{1}{2}} \tau^{a_0 - 1} e^{-b_0 \tau} \exp\left\{ -\frac{\tau}{2} \left[\sum_{n=1}^{N} (x_n - \mu)^2 + \lambda_0 (\mu - \mu_0)^2 \right] \right\} \\
&\propto \tau^{\frac{N}{2}} \tau^{\frac{1}{2}} \tau^{a_0 - 1} e^{-b_0 \tau} \exp\left\{ -\frac{1}{2} \tau (N + \lambda_0) \left(\mu - \frac{N\bar{x} + \lambda_0 \mu_0}{N\lambda_0} \right)^2 \right\} \times \\
&\exp\left\{ -\frac{\tau}{2} \left[-\frac{(N\bar{x} + \lambda_0 \mu_0)^2}{N + \lambda_0} + \sum_{n=1}^{N} x_n^2 + \lambda_0 \mu_0^2 \right] \right\} \\
&\propto \tau^{\frac{N}{2}} \tau^{a_0 - 1} e^{-b_0 \tau} \mathcal{N}\left(\mu; \frac{N\bar{x} + \lambda_0 \mu_0}{N\lambda_0}, \frac{1}{\tau (N + \lambda_0)} \right) \times \\
&\exp\left\{ -\frac{\tau}{2} \left[-\frac{(N\bar{x} + \lambda_0 \mu_0)^2}{N + \lambda_0} + \sum_{n=1}^{N} x_n^2 + \lambda_0 \mu_0^2 \right] \right\} \\
&\propto \mathcal{N}(\mu; \mu_{real}, \sigma_{real}) \times \Gamma(\tau; a_{real}, b_{real}))
\end{aligned}
$$

where:

$$
\begin{aligned}
\mu_{real} &= \frac{N\bar{x} + \lambda_0 \mu_0}{N + \lambda_0} \\
\sigma_{real} &= \frac{1}{\sqrt{\tau(N + \lambda_0)}} \\
a_{real} &= a_0 + \frac{N}{2} \\
b_{real} &= b_0 + \frac{1}{2}[(\sum_{n=1}^{N} x_n^2) + \lambda_0 \mu_0^2 - \frac{(N\bar{x} + \lambda_0 \mu_0)^2}{N + \lambda_0}]
\end{aligned}
$$

Question 2.3.11

Compare the inferred variational distribution with the exact posterior. Run the inference on data points drawn from i.i.d. Gaussians. Do this for three interesting cases and visualize the results. Describe the differences.

1. **Effects of data size:** In this case we try to approximate $N(0, 1)$ distribution, we leave all parameters set to 1 and vary the sample size to see the effect of the amount of data available on the approximation. We can see on figures 3, 4 and 5 how the more samples we have the better the inferred distribution fits the real distribution of the data. This was expected since its the usual case in machine learning. However, both real and inferred distributions fail to accurately model the data distribution $N(0, 1)$ in figure 3, in contrast to the other two cases, due to the extremely low number of points.

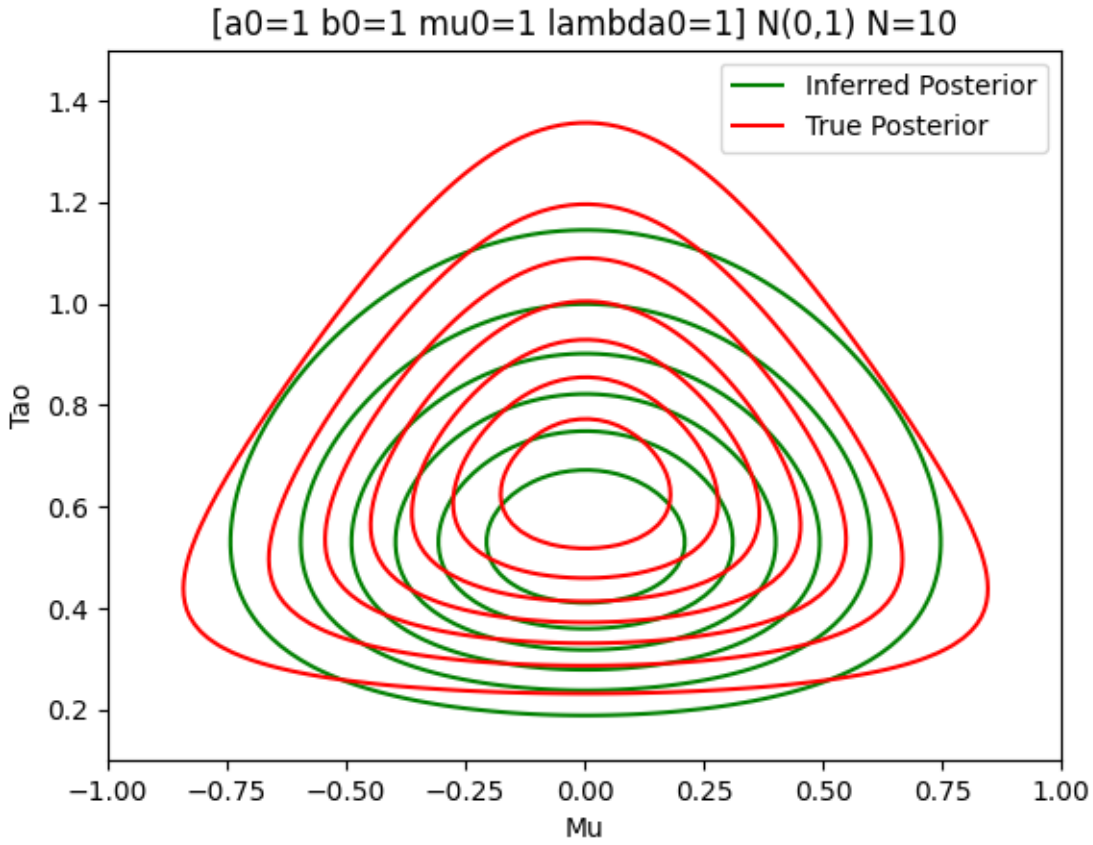


Figure 3: N(0,1) with 10 samples

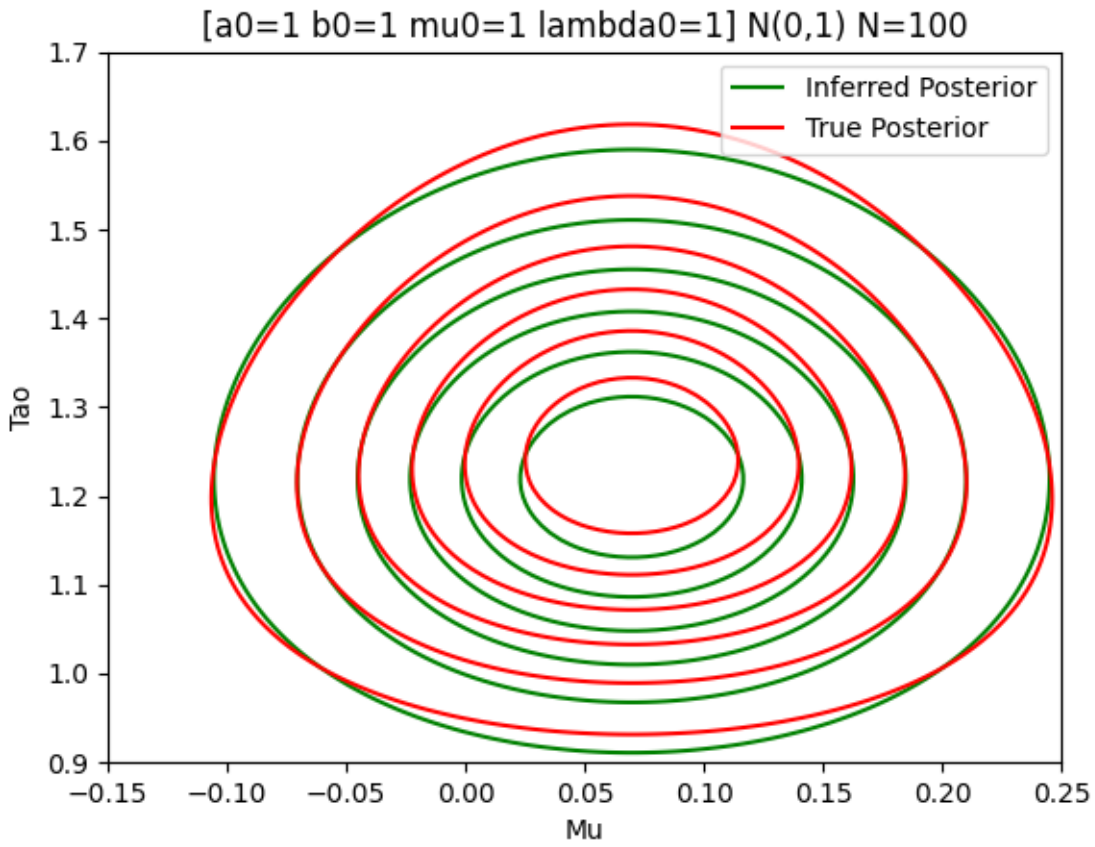


Figure 4: N(0,1) with 100 samples

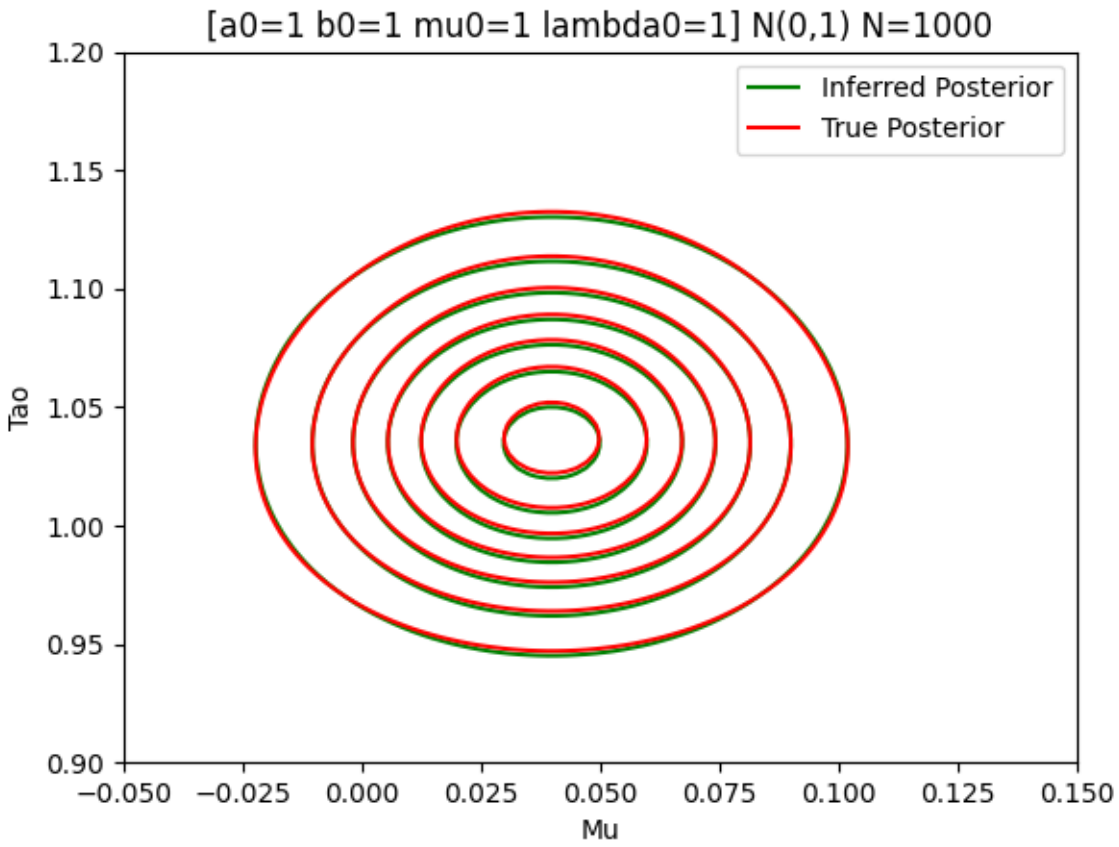


Figure 5: N(0,1) with 1000 samples

2. **Effects of model knowledge:** In this case we we try to approximate a $N(10, 1)$ distribution and we leave all parameters unchanged like in the previous section. In this section we will see how sometimes even with little data good models can be achieved if we have some knowledge about the model. It can be seen in figure 6 how this initialization of hyperparameters leeds to a good model on μ but a very poor one on τ.

   Now, let say that we learn from some external source that the underlying model has $\mu = 10$. Then we will change our initialization of μ_0 to 10. We can see on figure 7 how this really helped the model on both μ and τ.

   Other parameters like λ_0 can be initialized to different values but it seems like for this case is best to keep it unchanged. As we can see in figure 8 the model's τ just becomes worse.

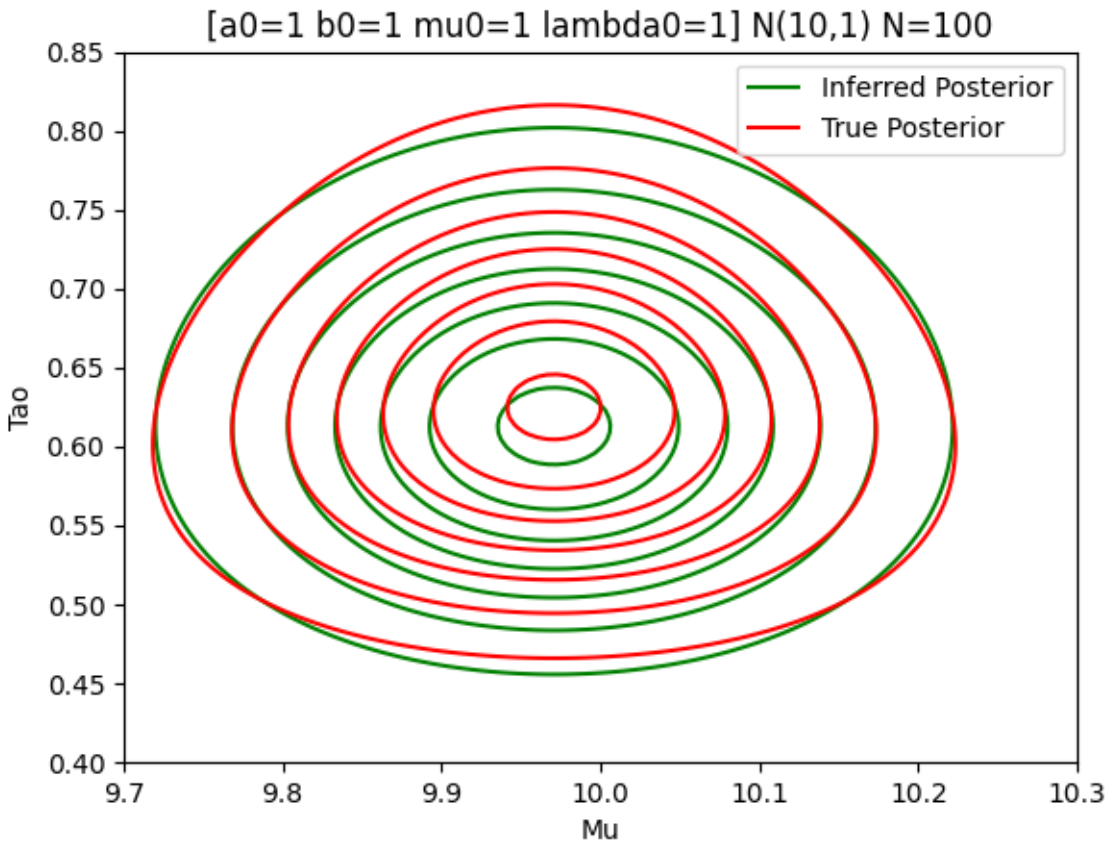


Figure 6: N(10,1) with 100 samples

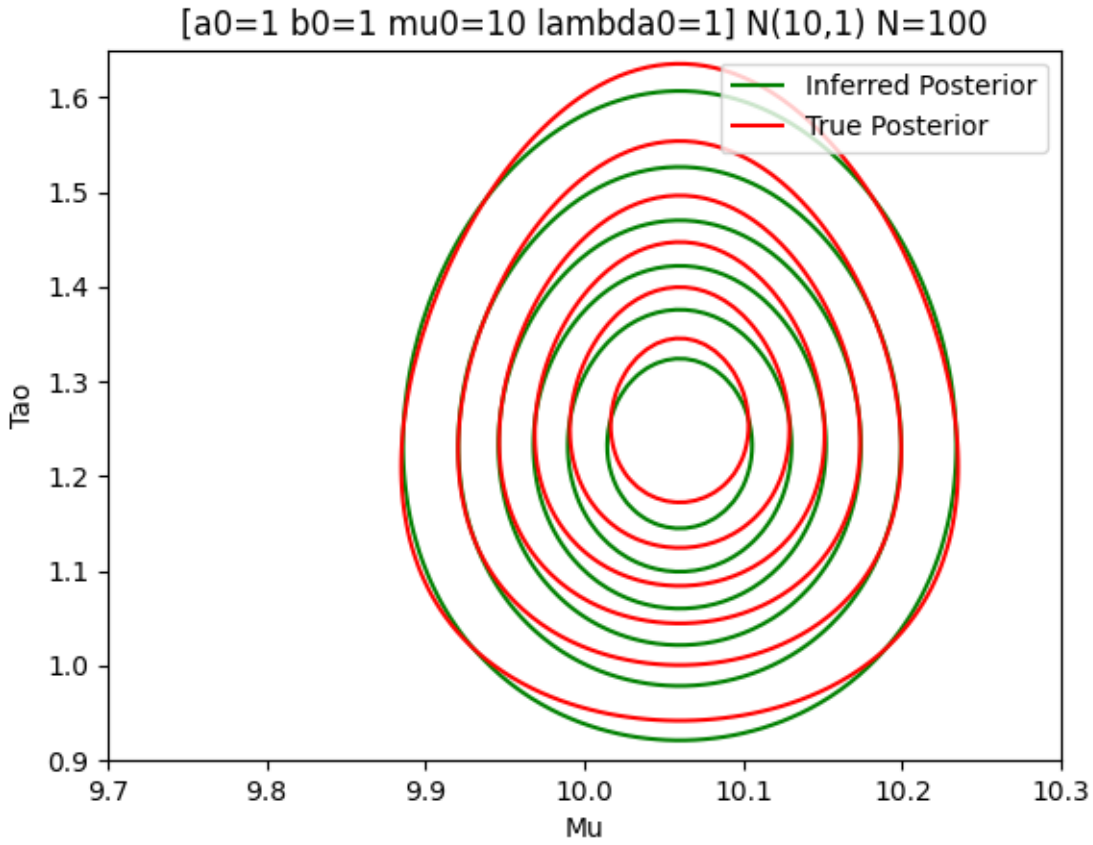


Figure 7: Effect of μ_0 on N(10,1) with 100 samples

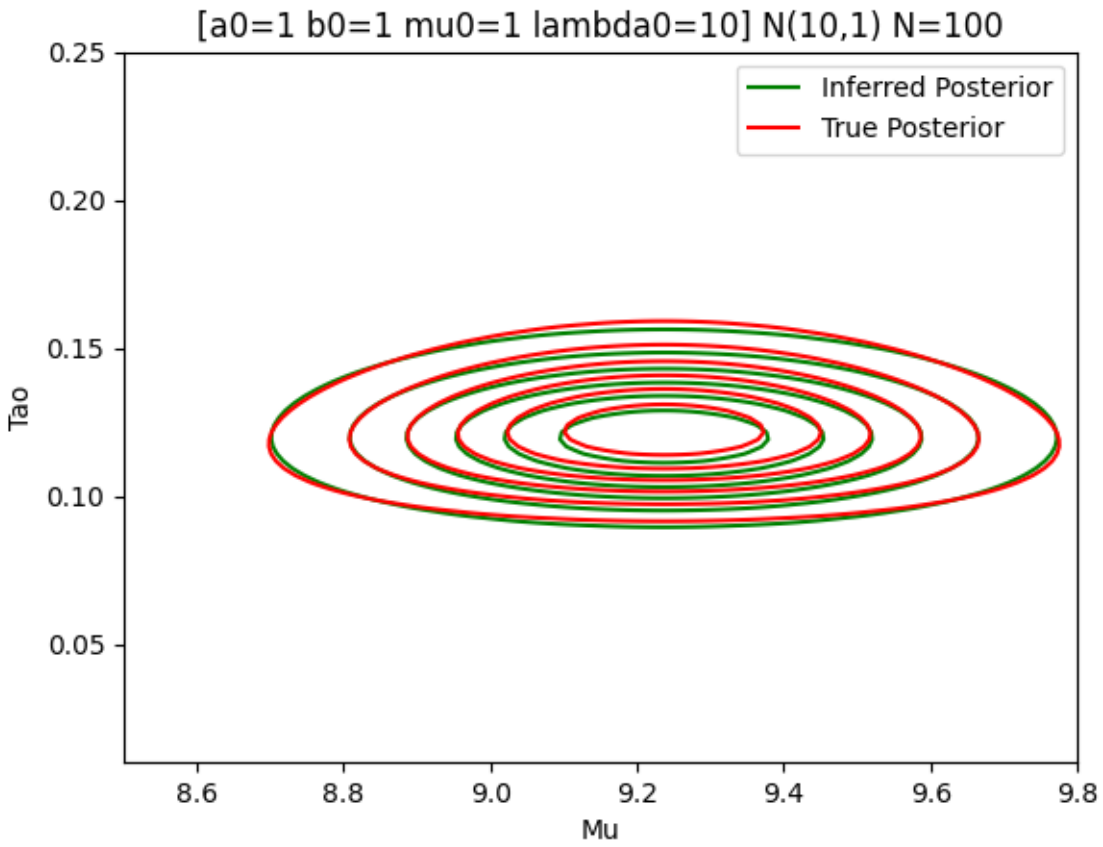


Figure 8: Effect of λ_0 on N(10,1) with 100 samples

3. **Other hyperparameter effects:** In this section we tune a_0 and b_0 and experiment with how their changes affect the distributions $N(0, 1)$. In order to do so we fix the data to have a 1000 samples and fix the other hyperparameters to be 0. As we can see in all of the cases (see figures 9, 10 and 11) the inferred posterior fits extremely well the true posterior due to the high number of data points. However, it can be seen in figure 9 how a high value of b_0 leads to an down shifted τ, in contrast, in figure 10 we see an up shifted τ do to the high value of the hyper parameter a_0. Finally, in figure 11 we can observe how, interestingly, when setting both a_0 and b_0 to the same value regardless of how high this value is, we obtain a almost perfect model for $N(0, 1)$.

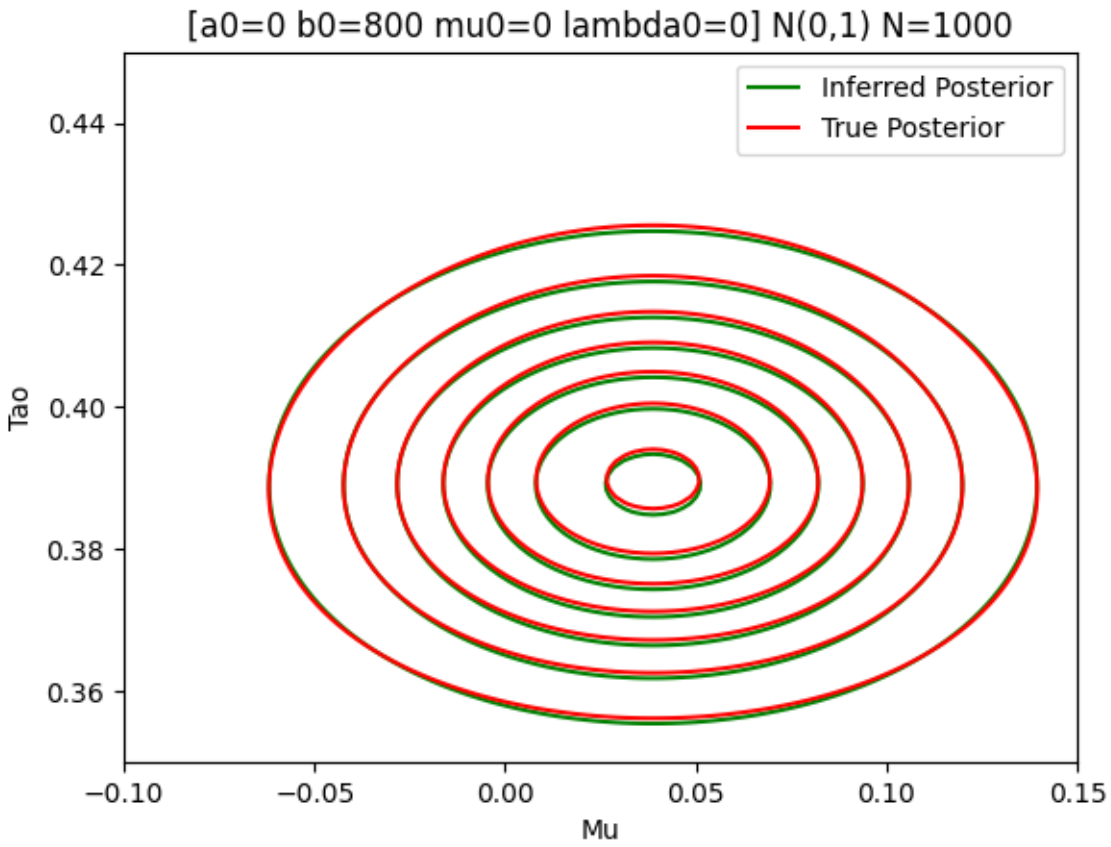


Figure 9: Effect of b_0 on N(0,1) with 1000 samples

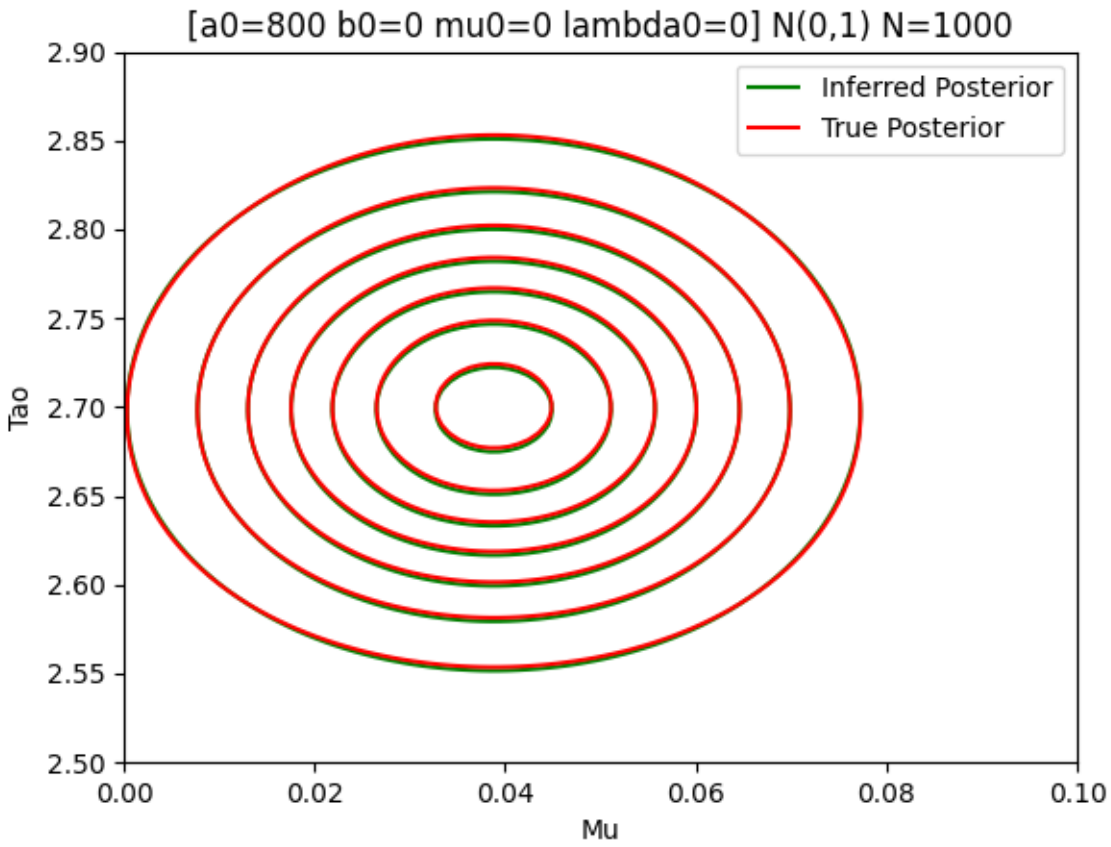


Figure 10: Effect of a_0 on N(0,1) with 1000 samples

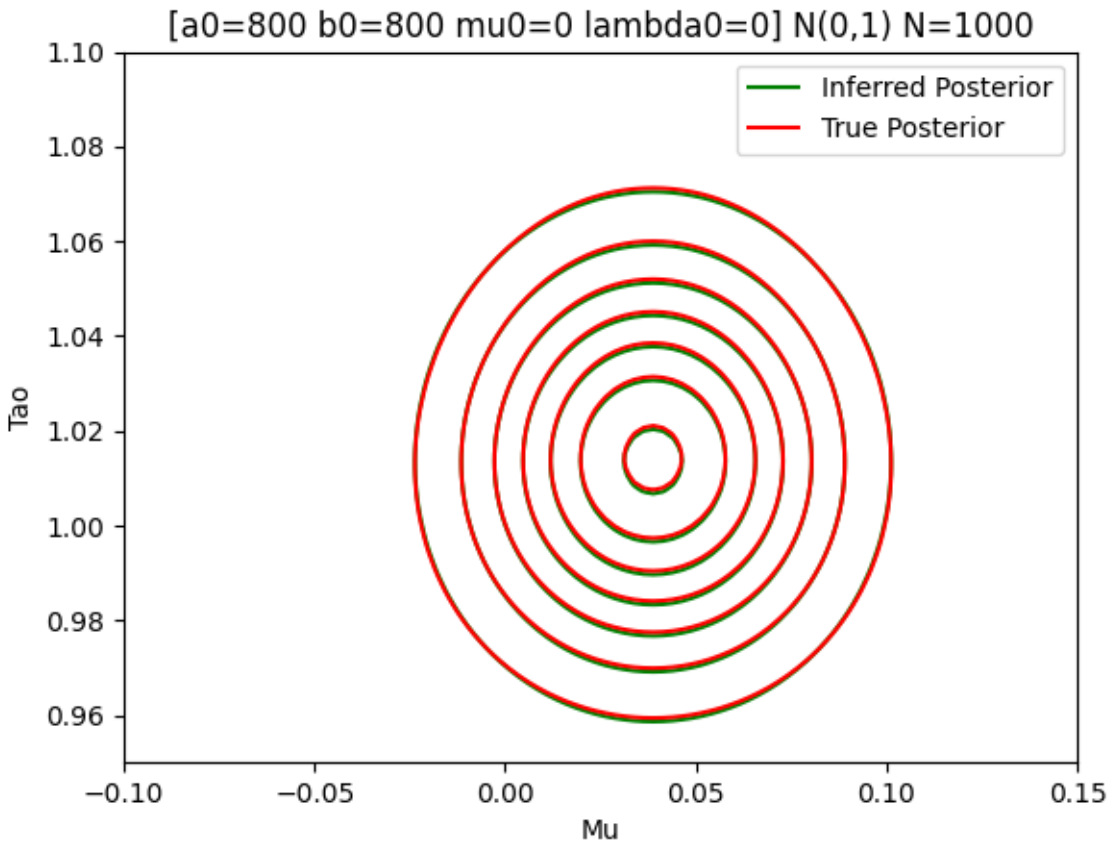
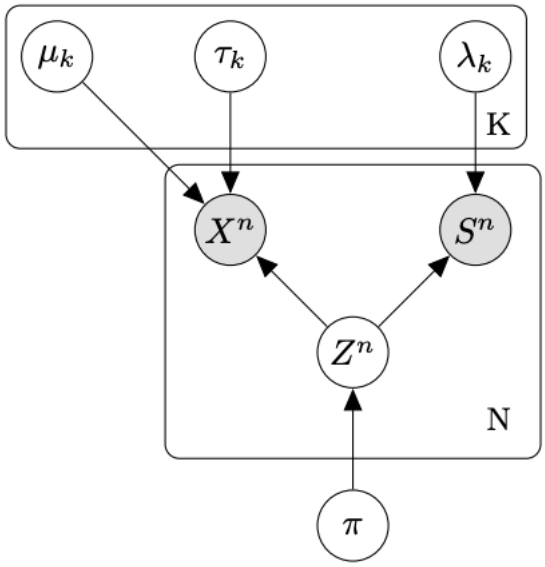


Figure 11: Effect of a_0, b_0 on N(0,1) with 1000 samples

2.4 Super Epicentra - Expectation-Maximization

We have seismographic from an area with frequent earthquakes emanating from K super epicentra. Each super epicentra is modeled by a 2-dimensional Gaussian determining the location of an earthquake and a Poisson distribution determining its strength. As shown in Figure 12 the entire model is a mixture of K such components. The variable Z^n is a class variable that follows a categorical distribution π and determines the super epicentra of the nth observation, i.e., the parameters of the Gaussian distribution that X^n is sampled from, $\mu_k = (\mu_{k,1}, \mu_{k,2})$ and $\tau_k = (\tau_{k,1}, \tau_{k,2})$ where $\tau_{k,1}$ and $\tau_{k,2}$ are diagonal elements of the diagonal precision matrix, as well as the intensity of the Poisson distribution that S^n is sampled from, λ_k.

Figure 12: Mixture of components modeling location and strengths of earthquakes associated with a super-epicentra. In the figure, $\mu_k = (\mu_{k,1}, \mu_{k,2})$ and $\tau_k = (\tau_{k,1}, \tau_{k,2})$



Question 2.4.12

Derive an EM algorithm for the model

The objective of EM algorithm is to maximize this likelihood [1]:

$$p(\boldsymbol{X} \mid \boldsymbol{\theta}) = \prod_{\boldsymbol{Z}} p(\boldsymbol{X}, \boldsymbol{Z} \mid \boldsymbol{\theta})$$

where $\boldsymbol{X}$ represents the observed variables X^n and S^n. X^n follows a $Normal(\mu_k, \tau_k)$ and S^n follows a $Poisson(\lambda_k)$. $\boldsymbol{Z}$ are the unobserved variables Z^n which follow a $Categorical(\pi_k)$. $\boldsymbol{\theta}$ concerns all parameters $\pi_k, \mu_k, \tau_k, \lambda_k$ which are what we actually need to find. We will define $\theta_k = (\mu_k, \tau_k, \lambda_k)$ and $\theta = (\pi, \theta_1, ..., \theta_k)$. Since it is more convenient for computations we will use the log-likelihood:

$$\ln(p\left(X^n, S^n \mid \boldsymbol{\theta}\right)) = \left\{ \sum_{Z} \ln(p\left(X^n, S^n, Z^n \mid \boldsymbol{\theta}\right)) \right\}$$

From the given graphical model, we can see X^n and S^n are conditionally independent given the latent variable Z^n and hence we can rewrite it as this:

$$P\left(X^n, S^n, Z^n \mid \theta\right) = p\left(Z^n \mid \theta\right) p\left(X^n \mid Z^n, \theta\right) p\left(S^n \mid Z^n, \theta\right)$$

Now, using all the information given, we can rewrite as:

$$p\left(X^n, S^n\right) = \sum_{Z} p\left(Z^n\right) p\left(X^n \mid Z^n\right) p\left(S^n \mid Z^n\right) = \sum_{k=1}^{K} \pi_k Normal\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\tau}_k^{-1} I\right) Poisson\left(S_n \mid \lambda_k\right)$$

Finally, from Bishop [1], the algorithm we must follow is:

1. Initalize $\pi_k, \mu_k, \tau_k, \lambda_k$
2. Compute Expectation [1]:

$$r_{nk} = \frac{\pi_k Normal\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\tau}_k^{-1} I\right) Poisson\left(\lambda_k\right)}{\sum_{k=1}^{K} \pi_k N\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\tau}_k^{-1} I\right) Po\left(\lambda_k\right)}$$

3. Now, we do the Maximization step in which we recompute the parameters using their distributions:

$$\boldsymbol{\mu}_k^* = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n$$

$$\boldsymbol{\tau}_k^* = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} \left(\boldsymbol{x}_n - \boldsymbol{\mu}_k^*\right) \left(x_n - \boldsymbol{\mu}_k^*\right)^T$$

$$\pi_k^* = \frac{N_k}{N}$$

$$\lambda_k^* = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} s_n$$

where $N_k = \sum_{n=1}^{N} r_{nk}$

4. Lastly, we evaluate the log-likelihood with the new parameters $(*)$. If the new log-likelihood has not changed by more than a user defined threshold, we stop. If it has changed more than the threshold, we go to 2.

Question 2.4.13

Implement your EM algorithm

See code in Appendix 4: Expectation-Maximization Code.

Question 2.4.14

Apply it to the data provided, give an account of the success, and provide visualizations for a couple of examples. Repeat it for a few different choices of K.

1. **Using 3 components:** In this section it is shown how the EM algorithm progresses through its first steps to the final result. It can be seen in 13 how the the clusters are initialized to $(0.3640, 6.7395)$ in the case of the red cluster, $(0.7429, 7.4138)$ for the green and $(5.7401, 3.7467)$ for the blue cluster. This is a very bad model of the data, however, it is just the inizialitation. After one iteration, in figure 14 we can see how the cluster means and rates make much more sense and describe the 3 clear groups in the 2-d data. In the next iteration (see figure 15) the model gets even better and the rates make much more sense. Finally, convergence its achieved in the 9th iteration of figure 16. All in all, the model obtained is very good and would generalize well.

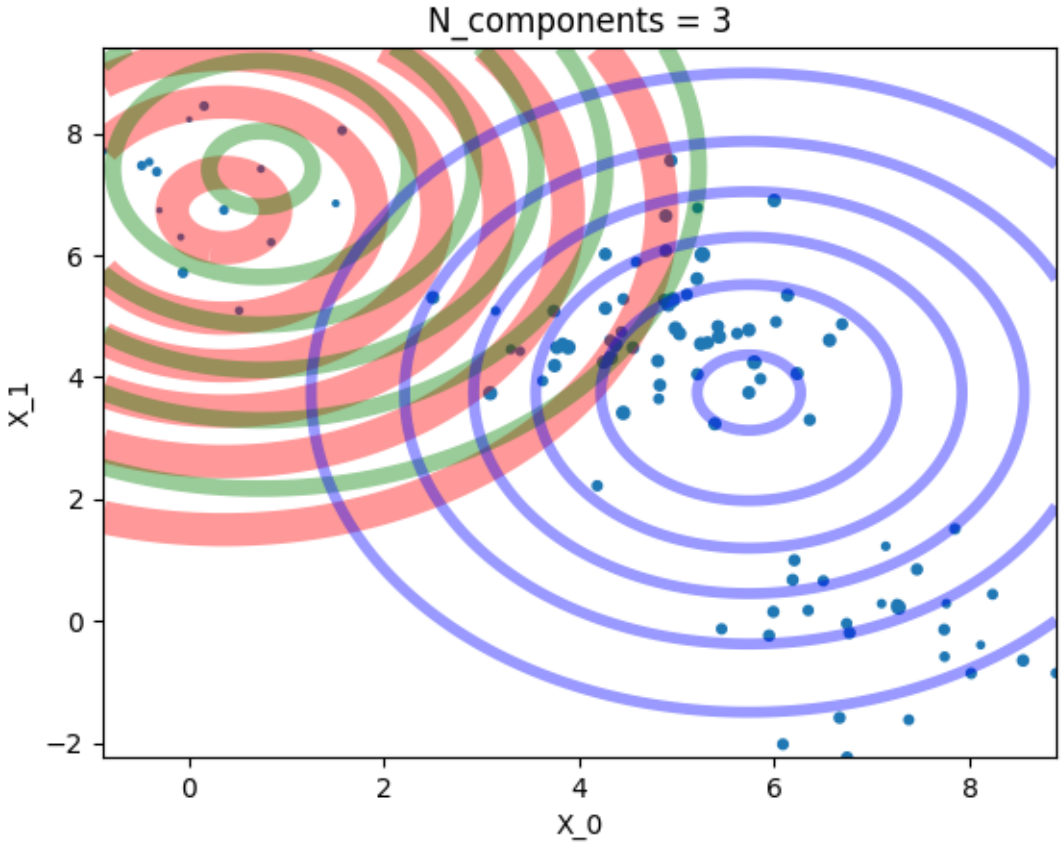


Figure 13: Initialization of EM algorithm

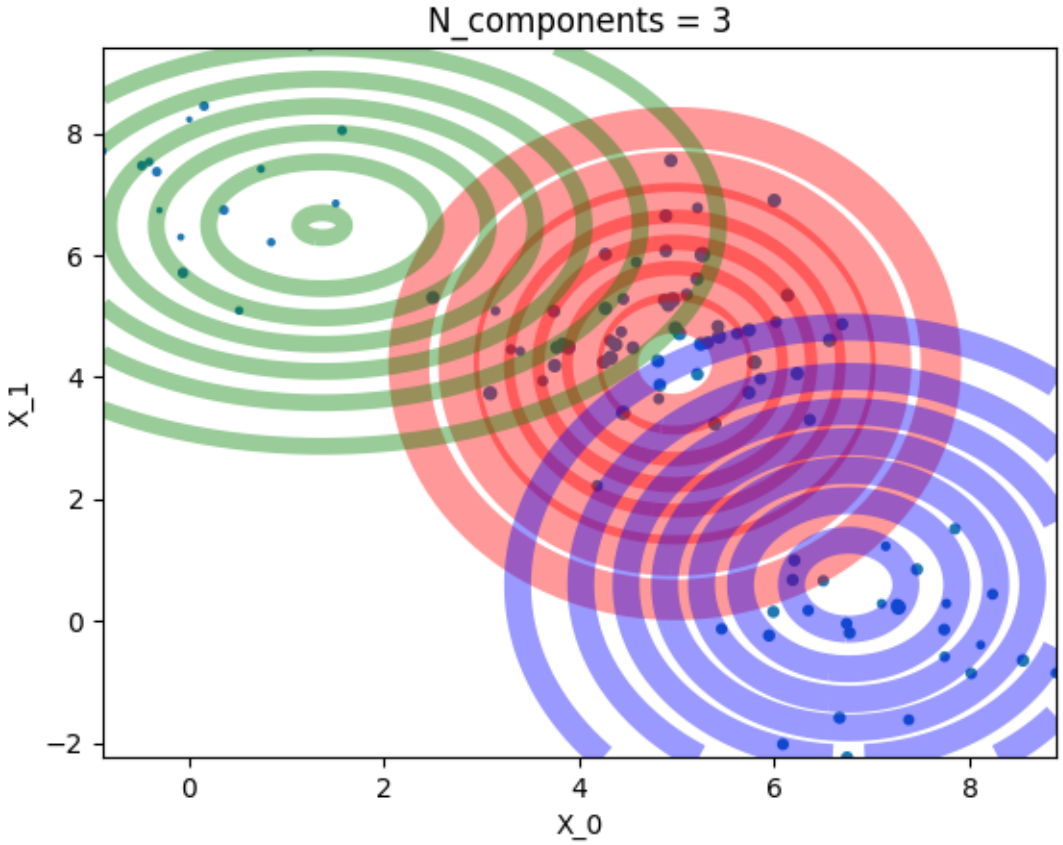


Figure 14: First iteration of EM algorithm

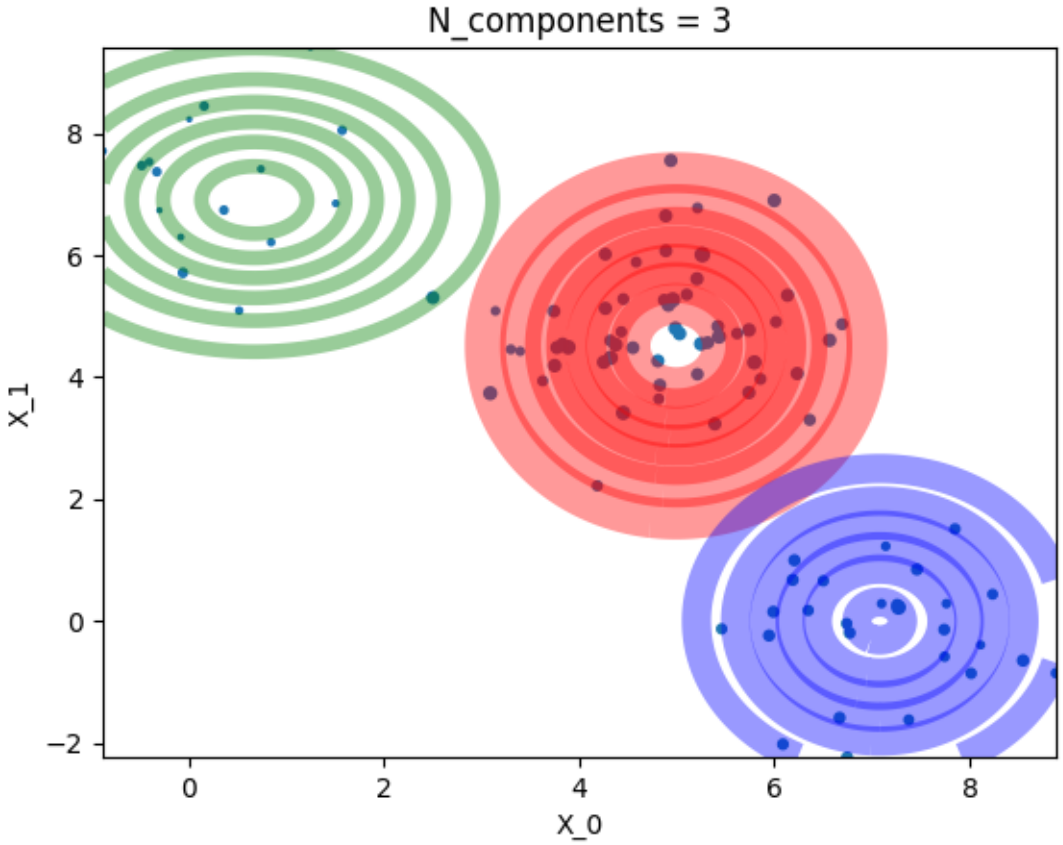


Figure 15: Second iteration of EM algorithm

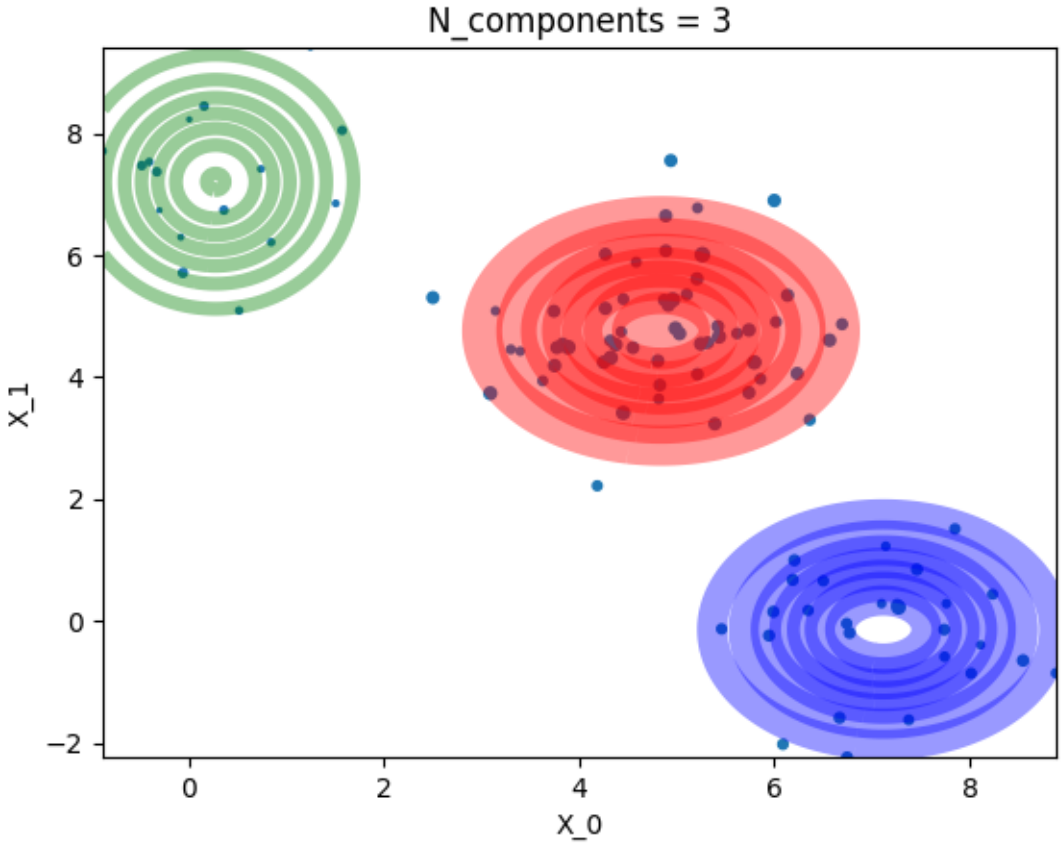


Figure 16: 9th iteration of EM algorithm - Convergence achieved

2. **Using < 3 components:** In this case we show in figure 17 how a bad election of clusters can lead to bad models. Even though convergence was achieved for a threshold value of $1e-300$ the model lacks 1 more cluster to represent the data. This is a clear example of a model which generalizes too much, i.e, a high bias but a very low variance model.

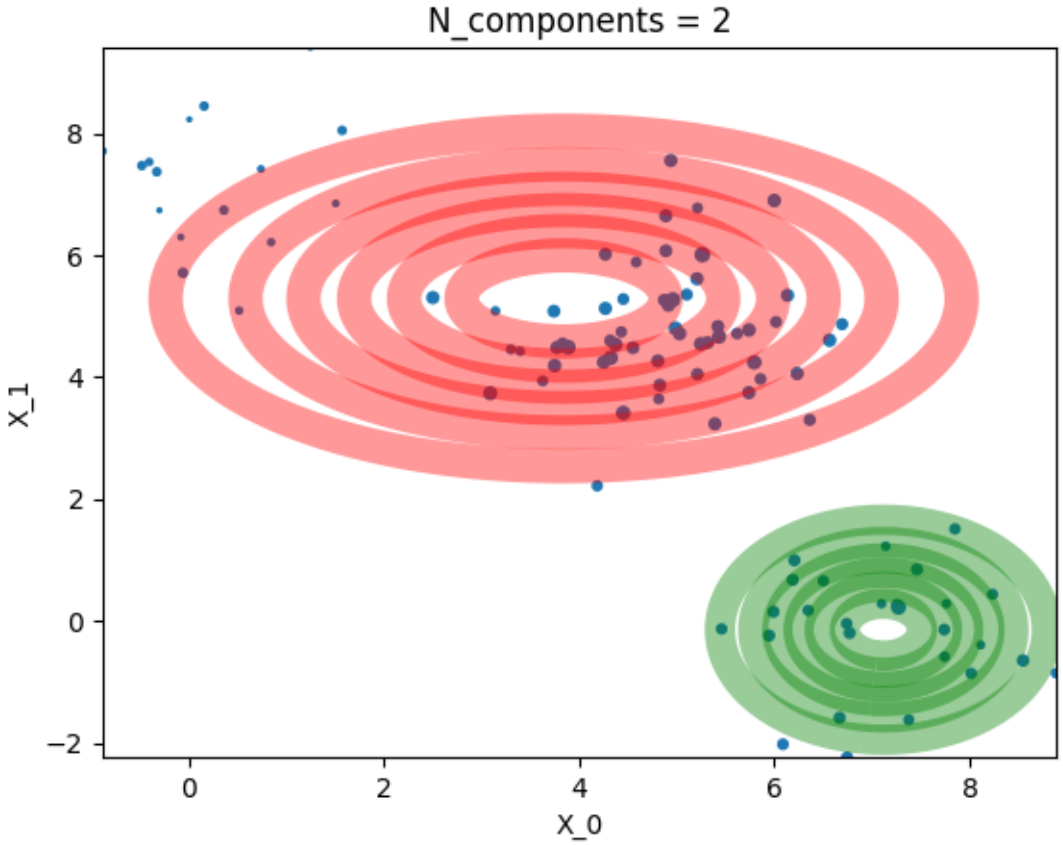


Figure 17: Bad Model - Convergence achieved

3. **Using > 3 components:** In the case of 18 we show the exact opposite of the previous point. The model overfits the data and tries to create too many clusters which are too specialized. Some of the Poisson rates lead to very complex relationships. This is a clear example of a low bias but high variance model.

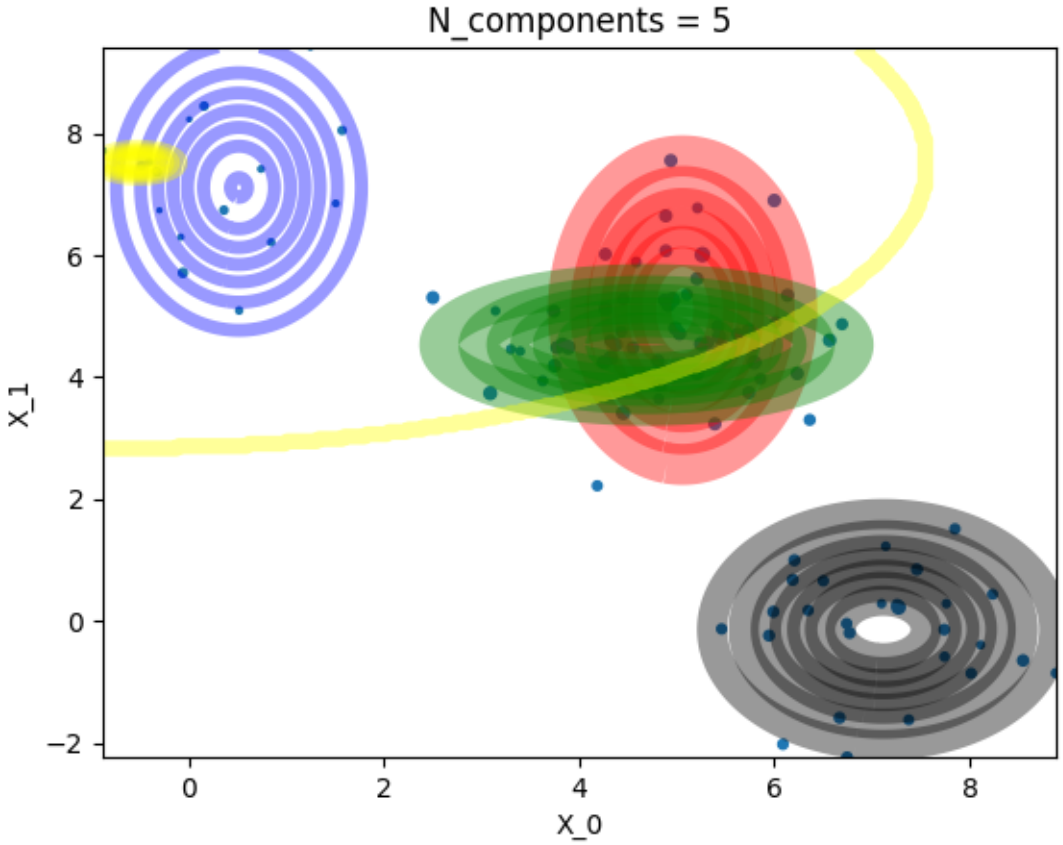


Figure 18: Bad Model - Convergence achieved

List of Figures

References

[1] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2016.

[2] *Normal-gamma distribution*, May 2021. [Online]. Available: https://en.wikipedia.org/wiki/Normal-gamma_distribution.

Appendix 2: Tree Code

```
""" This file is created as a suggested solution template for question 2.2 in

    We encourage you to keep the function templates.
    However this is not a "must" and you can code however you like.
    You can write helper functions etc however you want.

    If you want, you can use the class structures provided to you (Node and Tr
    file), and modify them as needed. In addition to the data files given to y
    test your algorithm with your own simulated data for various cases and and

    For those who do not want to use the provided structures, we also saved th
    format. Let us know if you face any problems.

    Also, we are aware that the file names and their extensions are not well-f
    (i.e example_tree_mixture.pkl_samples.txt). We wanted to keep the template
    You can change the file names however you want.

    For this task, we gave you three different trees (q2_2_small_tree, q2_2_me
    Each tree has 5 samples (the inner nodes' values are masked with np.nan).
    We want you to calculate the likelihoods of each given sample and report i

    Note:   The alphabet "K" is K={0,1,2,3,4}.

    Note:   A VERY COMMON MISTAKE is to use incorrect order of nodes' values i
            theta is a list of lists, whose shape is approximately (num_nodes,
            For instance, if node "v" has a parent "u", then p(v=Zv | u=Zu) =

            If you ever doubt your useage of theta, you can double check this
            \sum_{k=1}^K p(v = k | u=Zu) = 1
"""

import numpy as np
from Tree import Tree
from Tree import Node


def calculate_likelihood(tree_topology, theta, beta):
    """
    This function calculates the likelihood of a sample of leaves.
    :param: tree_topology: A tree topology. Type: numpy array. Dimensions: (nu
    :param: theta: CPD of the tree. Type: list of numpy arrays. Dimensions (ap
    :param: beta: A list of node assignments. Type: numpy array. Dimensions: (
                Note: Inner nodes are assigned to np.nan. The leaves have valu
```

```
    :return: likelihood: The likelihood of beta. Type: float.

    This is a suggested template. You don't have to use it.
    """

    print("\tCalculating the likelihood...")
    likelihood = 0
    K = len(theta[0])
    num_nodes = tree_topology.size
    s_matrix = np.zeros(shape=(num_nodes, K))

    for node, value in reversed(list(enumerate(beta))):
        if not np.isnan(value):  # leaf node
            s_matrix[node][int(value)] = 1

        else:  # inner node
            children = np.argwhere(tree_topology == node).flatten()
            for k in range(K):
                s_matrix[node][k] = 1
                for child in children:
                    s_matrix[node][k] = s_matrix[node][k] * \
                        np.dot(s_matrix[child], theta[child][k])
    # End: Example Code Segment
    likelihood = np.dot(theta[0], s_matrix[0])
    return likelihood


def main():
    print("Hello World!")
    print("This file is the solution template for question 2.2.")

    print("\n1. Load tree data from file and print it\n")

    filenames = ["2_2/data/q2_2_small_tree.pkl",
                 "2_2/data/q2_2_medium_tree.pkl", "2_2/data/q2_2_large_tree.pk
    for f in filenames:
        print('#'*70)
        print("##########\tFilename: ", f, '   ##########')
        print('#'*70)

        t = Tree()
        t.load_tree(f)
        # t.print()
        print("K of the tree: ", t.k, "\talphabet: ", np.arange(t.k))

        print("\n2. Calculate likelihood of each FILTERED sample\n")
        # These filtered samples already available in the tree object.
        # Alternatively, if you want, you can load them from corresponding .tx

        for sample_idx in range(t.num_samples):
            beta = t.filtered_samples[sample_idx]
            # print("\n\tSample: ", sample_idx, "\tBeta: ", beta)
```

```
            sample_likelihood = calculate_likelihood(
                t.get_topology_array(), t.get_theta_array(), beta)
            print("\tLikelihood: ", sample_likelihood)


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

Appendix 3: Variational Inference Code

```
import numpy as np
import math
from scipy import stats
import matplotlib.pyplot as plt
import sys
import os


class VI():
    def __init__(self, a0, b0, mu0, lambda0, Etao, threshold):
        self.a0 = a0
        self.b0 = b0
        self.mu0 = mu0
        self.lambda0 = lambda0
        self.Etao = Etao
        self.muN = 0
        self.lambdaN = 0
        self.aN = 0
        self.bN = 0
        self.threshold = threshold

    def fit(self, X):
        print("\tDoing Variational Inference....")
        while True:
            old_Etao = self.Etao
            self.q_mu_update(X)
            self.q_tao_update(X)
            self.Etao = self.aN/self.bN
            if abs(self.Etao - old_Etao) < self.threshold:
                break

    def q_mu_update(self, X):
        mean, N = np.mean(X), X.shape[0]

        self.muN = ((self.lambda0*self.mu0) + N*mean) / (self.lambda0 + N)
        self.lambdaN = (self.lambda0 + N)*self.Etao

    def q_tao_update(self, X):
        mean, N = np.mean(X), X.shape[0]

        self.aN = self.a0 + N/2
```

```
        self.bN = np.dot(X, X) + (N + self.lambda0) * \
            (1/self.lambdaN + math.pow(self.muN, 2))
        self.bN = self.bN - 2*self.muN*(N*mean + self.mu0*self.lambda0)
        self.bN = self.bN + self.lambda0*math.pow(self.mu0, 2)
        self.bN = self.b0 + 0.5*self.bN

    def get_posterior(self, mu_axis, tao_axis):
        mu_size, tao_size = len(mu_axis), len(tao_axis)
        q = np.zeros(shape=(mu_size, tao_size))

        print('\tCalculating VI posterior approximation ...')
        for i in range(mu_size):
            for j in range(tao_size):
                q_tao = stats.gamma.pdf(
                    tao_axis[j], self.aN, loc=0, scale=1/self.bN)
                q_mu = stats.norm.pdf(
                    mu_axis[i], self.muN, np.sqrt(1/self.lambdaN))
                q[j][i] = q_tao*q_mu
        return q


def generate_data(mu, tao, N, mu_axis, tao_axis, lambda0, mu0, a0, b0):
    sigma = np.sqrt(1/tao)
    X = np.random.normal(mu, sigma, N)
    true_posterior = get_true_posterior(
        mu_axis, tao_axis, X, lambda0, mu0, a0, b0)
    return X, true_posterior


def get_true_posterior(mu_axis, tao_axis, X, lambda0, mu0, a0, b0):
    mu_size, tao_size = len(mu_axis), len(tao_axis)
    p = np.zeros(shape=(mu_size, tao_size))
    mean, N = np.mean(X), X.shape[0]

    mu = (N*mean + lambda0*mu0)/(N + lambda0)
    a = a0 + N/2
    b = b0 + 0.5 * (np.dot(X, X) + lambda0*mu0**2 -
                    (N*mean + lambda0*mu0)**2/(N+lambda0))

    print('\tCalculating true posterior ...')
    for i in range(mu_size):
        for j in range(tao_size):
            tao = (tao_axis[j]*(N+lambda0)) + sys.float_info.epsilon
            sigma = np.sqrt(1/tao)

            p_tao = stats.gamma.pdf(tao_axis[j], a, loc=0, scale=1/b)
            p_mu = stats.norm.pdf(mu_axis[i], mu, sigma)
            p[j][i] = p_tao*p_mu

    return p
```

```
def plot_contours(mu_axis, tao_axis, VI_posterior, true_posterior, filename, t
    filename = f'2_3/plots/{filename}'

    plt.figure()
    CS_inferred = plt.contour(mu_axis, tao_axis, VI_posterior, colors='green')
    CS_true = plt.contour(mu_axis, tao_axis, true_posterior, colors='red')

    # Plot density of pdf on the contour line
    # plt.clabel(CS_inferred, fmt="%1.2f")
    # plt.clabel(CS_true, fmt="%1.2f")

    labels = ["Inferred Posterior", "True Posterior"]
    lines = [CS_inferred.collections[0], CS_true.collections[0]]
    plt.legend(lines, labels)
    plt.xlabel("Mu")
    plt.ylabel("Tao")

    plt.title(title)
    plt.savefig(f'{filename}.png')


def main():
    np.random.seed(1)
    # Init parameters
    a0, b0, mu0, lambda0 = 1, 1, 1, 1
    Etao = 1
    threshold = 1e-300
    # Generate data
    mu, tao, N = 0, 1, 1000
    mu_axis = np.linspace(-0.05, 0.15, 100)
    tao_axis = np.linspace(0.9, 1.2, 100)
    X, true_posterior = generate_data(
        mu, tao, N, mu_axis, tao_axis, lambda0, mu0, a0, b0)

    VI_model = VI(a0, b0, mu0, lambda0, Etao, threshold)
    VI_model.fit(X)
    VI_posterior = VI_model.get_posterior(mu_axis, tao_axis)

    # a0-b0-mu0-lambda0-Etao-thres-mu-tao-N
    filename = f'{a0}-{b0}-{mu0}-{lambda0}-{str(threshold)}-{mu}-{tao}-{N}'
    title = f'[a0={a0} b0={b0} mu0={mu0} lambda0={lambda0}] N({mu},{tao}) N={N}

    plot_contours(mu_axis, tao_axis, VI_posterior,
                  true_posterior, filename, title)


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

Appendix 4: Expectation-Maximization Code

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal, poisson


def load_data(X_file, S_file):
    X, S = [], []

    with open(f"2_4/data/{X_file}", 'r') as f:
        for line in f.readlines():
            X.append(line.split(' '))
        X = np.array(X).astype(float)

    with open(f"2_4/data/{S_file}", 'r') as f:
        for line in f.readlines():
            S.append(line.split()[0])
        S = np.array(S).astype(float)
    return X, S


def plot_contours(X, S, mu_k, covar_k, lambda_k):
    N_points = 300
    N_components = mu_k.shape[0]

    X0_min, X0_max = np.min(X[:, 0]), np.max(X[:, 0])
    X1_min, X1_max = np.min(X[:, 1]), np.max(X[:, 1])

    X0_axis = np.linspace(X0_min, X0_max, N_points)
    X1_axis = np.linspace(X1_min, X1_max, N_points)

    X0, X1 = np.meshgrid(X0_axis, X1_axis)
    plt.figure()
    # plot X points
    plt.scatter(X[:, 0], X[:, 1], s=S)

    # Draw Mixture of Gaussians
    colors = ['red', 'green', 'blue', 'yellow', 'black']
    for k in range(N_components):
        mu = mu_k[k]
        covar = covar_k[k]
        rv = multivariate_normal(mu, covar)
        Z = rv.pdf(np.dstack((X0, X1)))
        plt.contour(X0, X1, Z, colors=colors[k],
                    linewidths=lambda_k[k], alpha=0.4)

    plt.xlabel("X_0")
    plt.ylabel("X_1")
    plt.title(f"N_components = {N_components}")
    plt.savefig(f"2_4/plots/{N_components}.png")


class GaussianPoissonEM:
    def __init__(self, N_components, threshold):
```

```
        self.N_components = N_components
        self.threshold = threshold

    def fit(self, X, S):
        self.init_params(X, S)
        old_log_likelihood = 0
        iter = 0

        print('\tFitting Expectation Maximization...')
        while True:
            self.E_step(X, S)
            self.M_step(X, S)
            new_log_likelihood = self.log_likelihood(X, S)
            if abs(old_log_likelihood - new_log_likelihood) < self.threshold:
                break
            old_log_likelihood = new_log_likelihood
            iter = iter + 1

        self.log_likelihood = new_log_likelihood
        print(f'\tIterations= {iter}')

    def init_params(self, X, S):
        N_points, N_dim = X.shape

        self.pi_k = np.full(self.N_components, 1/self.N_components)
        self.mu_k = X[np.random.choice(
            N_points, self.N_components, replace=False)]
        self.covar_k = np.full(
            (self.N_components, N_dim, N_dim), np.cov(X, rowvar=False))
        self.lambda_k = np.full(self.N_components, np.mean(S))
        self.lambda_k = self.lambda_k / \
            np.array([i for i in range(1, self.N_components+1)])

        # Exercise assumes covariances diagonal so...
        covar_k = []
        for c in self.covar_k:
            covar_k = np.append(covar_k, np.diag(np.diag(c)))
        self.covar_k = np.array(covar_k).reshape(self.N_components, 2, 2)

    def get_params(self):
        return self.pi_k, self.mu_k, self.covar_k, self.lambda_k

    def E_step(self, X, S):
        N_points = X.shape[0]
        r_nk = np.zeros((N_points, self.N_components))

        for n in range(N_points):
            for k in range(self.N_components):
                r_nk[n][k] = self.pi_k[k] *\
                    multivariate_normal(self.mu_k[k], self.covar_k[k]).pdf(X[n
                    poisson(self.lambda_k[k]).pmf(S[n])
            r_nk[n] = r_nk[n]/r_nk[n].sum()
```

```
        # Set responsibility, i.e, E[I(Z_n=class_k)]
        self.r_nk = r_nk

    def M_step(self, X, S):
        N_k = self.r_nk.sum(axis=0)
        N_points, N_dim = X.shape

        for k in range(self.N_components):
            self.pi_k[k] = N_k[k]/N_points
            self.lambda_k[k] = np.sum(self.r_nk[:, k] * S[:])/N_k[k]
            for d in range(N_dim):
                self.mu_k[k][d] = np.sum(self.r_nk[:, k] * X[:, d])/N_k[k]
                self.covar_k[k][d][d] = np.sum(self.r_nk[:, k] *
                                               (X[:, d] - self.mu_k[k][d]) *
                                               np.transpose(X[:, d] - self.mu_

    def log_likelihood(self, X, S):
        log_likelihood = np.zeros(self.N_components)
        for k in range(self.N_components):
            log_likelihood[k] = sum(self.r_nk[:, k] * (self.pi_k[k] * multivar
                self.mu_k[k], self.covar_k[k]).pdf(X[::]) * poisson(self.lambd
        log_likelihood = np.sum(np.log(log_likelihood))

        return log_likelihood

    def print_params(self):
        for k in range(self.N_components):
            print(f'\tComponent {k}:')
            pi = self.pi_k[k]
            mu_0, mu_1 = self.mu_k[k][0], self.mu_k[k][1]
            c_0, c_1 = self.covar_k[k][0][0], self.covar_k[k][1][1]
            rate = self.lambda_k[k]
            print(
                f'\t\tpi=[{pi:.4f}]\n\t\tNormal([{mu_0:.4f}, {mu_1:.4f}],[{c_0


def main():
    np.random.seed(1)
    X_file, S_file = "X.txt", "S.txt"
    X, S = load_data(X_file, S_file)
    N_components = [2, 3, 5]
    threshold = 1e-10
    for k in N_components:
        print('#'*10, f'\t{k} Components\t', '#'*10)
        EM_model = GaussianPoissonEM(k, threshold)
        EM_model.fit(X, S)
        EM_model.print_params()

        pi_k, mu_k, covar_k, lambda_k = EM_model.get_params()
        plot_contours(X, S, mu_k, covar_k, lambda_k)
```

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