DD2434 - Advanced Machine Learning - Assignment # 2

Altay Dikme

December 22, 2020

Contents

1 Dependencies in a Directed Graphical Model

2.1.1

No.

2.1.2

Yes.

2.1.3

$A = \{\mu_{r,c}, \mu_{r,c-1}, \mu_{r,c+1}, \mu_{r-1,c}, \mu_{r+1,c}\}$

2.1.4

No.

2.1.5

No.

2.1.6

$B = \{Z_m^n : n \in [N], m \in [M]\} \cup \{C^n : n \in [N]\}$

2 Likelihood of a Tree Graphical Model

2.2.7

We have T as a binary tree, with vertex set $V(T)$, and leaf set $L(T)$. For each vertex $v \in V(T)$ there is an associated random variable X_v that assumes values in $[K] = 0, 1, 2, 3, 4$. Moreover for each $v \in V(T)$ the CPD $\theta_v = p(X_v|X_{pa(v)})$ is a categorical distribution. We let $\beta = \{x_l : l \in L(T)\}$ be an assignment of values to all of the leaves of T.

We want to implement a dynamic programming algorithm which for a given T, θ, β determines $p(\beta|T, \theta)$. We know that the leafs of the tree are observed and thus we know the value of K which they will assume with probability 1. We can use the method which Jens explains in module 7. We begin by creating an S matrix which has dimensions $N \times K$ where N is the number of nodes in the tree and K is the number of values K which each node can take. In this case we have $K = 5$ and the number of nodes vary depending on if we are looking at the small, medium or large tree. The way to interpret the S matrix is that the i, jth entry represents the probability that node i takes on the jth value in $[K]$. So an example would be if we look at entry $S(4, 1)$ this will tell us the probability of node 4 having value $K = 1$.

Thus after we initialize S to be an $N \times K$ matrix we can start by filling it with the probabilities of each leaf node. We know what these values are so the row of each leaf node will only have zeros and a 1 in the position of the value K which the node has (Given to us by β).

Once we have done this we are left with an S matrix where the rows belonging to leaf nodes have been determined. Now we can start by looking at the "deepest" node, i.e the node with the largest index value and inspect its parent node. We set each value in the corresponding row of our S matrix to be 1 since we will be multiplying these with the probabilities later. Next we look for each value of K which the parent node can take on and determine the probability for this based on the probability of the children and their CPS's as described in module 7.6b. I.e that:

$$S(u, i) = P(X_{0 \cap \downarrow v}|X_u = i)P(X_{0 \cap \downarrow w}|X_u = i) \tag{1}$$

$$P(X_{0 \cap \downarrow v}|X_u = i) = \sum_j S(v, j)P(X_v = j|X_u = i) \tag{2}$$

Where v and w are the children of u. We can see that the two products in Eq. 1 are identical except for the node. Thus once we are at the parent node we calculate Eq. 2 for both of its children and multiply them together. This gives us $S(u, i)$ for the parent node (i.e filling the row corresponding to the parent node with probabilities that it takes upon values of K). This procedure is then repeated with the node with next largest index and so on until the probabilities of the parents of all the leafs have been determined. This procedure is then repeated until we reach the root node. In this way we have filled our S matrix.

Finally we want to look at the first row of the S matrix, corresponding to the probabilities of the root node and take the dot product between these and the CPD for the root in order to determine the likelihood.

To summarize we have:

- Initialize S matrix, dimensions $N \times K$.
- Fill in probabilities corresponding to the leaf nodes of tree.
- Starting from node with largest index (I.e deepest node), inspect parent node and determine the corresponding row in the S matrix using Eq. 1 and 2. We know that children of the parent will both be leafs since we start from the deepest node.
- Repeat until we reach the root node, meaning that we have filled the S matrix.
- Finally we can determine $p(\beta|T, \theta)$.

2.2.8

We want to apply this algorithm to the graphical model and data provided separately. We are given three trees, one small one medium and one large. We are also given 5 samples β for each tree. In total we have 15 samples (i.e observations on leaf nodes) and we want to determine 15 different likelihoods. These are shown in Table 1.

	Small Tree	Medium Tree	Large Tree
Sample 0	0.016178983188381856	4.3359985610595595e-18	3.287622233372845e-69
Sample 1	0.015409920999590558	3.094115541974649e-20	1.1094518417121311e-66
Sample 2	0.011368474549717015	1.0500091205098358e-16	2.5224240937554147e-68
Sample 3	0.00864042722338585	6.585311240142626e-16	1.2423555476116803e-66
Sample 4	0.04091494618599329	1.4880108219386274e-18	3.535477501915256e-69

Table 1: $p(\beta|T,\theta)$ for the different samples β and the different trees given in the assignment

3 Simple Variational Inference

The likelihood function is given by:

$$P(D|\mu,\tau) = \left(\frac{\tau}{2\pi}\right)^{\frac{N}{2}} \exp\left\{-\frac{\tau}{2}\sum_{n=1}^{N}(x_n-\mu)^2\right\} \tag{3}$$

And the conjugate prior distributions for μ and τ are:

$$P(\mu|\tau) = N(\mu|\mu_0,(\lambda_0\tau)^{-1}) \tag{4}$$

$$P(\tau) = Gam(\tau|a_0,b_0) \tag{5}$$

2.3.9

Beginning with equation 10.24 in Bishop we have:

$$q(\mu,\tau) = q_\mu(\mu)q_\tau(\tau) \tag{6}$$

Which is the factorized variational approximation to the posterior distribution. It can be seen that $q_\mu(\mu)$ is a Gaussian $N(\mu|\mu_N,\lambda_N^{-1})$ with mean and precision given by:

$$\mu_N = \frac{\lambda_0\mu_0 + N\bar{x}}{\lambda_0+N} \tag{7}$$

$$\lambda_N = (\lambda_0+N)\mathbb{E}[\tau] \tag{8}$$

Similarly $q_\tau(\tau)$ is a Gamma distribution $Gam(\tau|a_N,b_N)$ with parameters:

$$a_N = a_0 + \frac{N}{2} \tag{9}$$

$$b_N = b_0 + \frac{1}{2}\mathbb{E}_\mu\left[\sum_{n=1}^{N}(x_n-\mu)^2 + \lambda_0(\mu-\mu_0)^2\right] \tag{10}$$

We can expand b_N and also rewrite λ_N since we know that $\mathbb{E}[\tau] = \frac{a_N}{b_N}$. Thus we have:

$$\mu_N = \frac{\lambda_0 \mu_0 + N\bar{x}}{\lambda_0 + N} \tag{11}$$

$$\lambda_N = (\lambda_0 + N)\frac{a_N}{b_N} \tag{12}$$

$$a_N = a_0 + \frac{N}{2} \tag{13}$$

$$b_N = b_0 + \frac{\lambda_0}{2}\left(\mathbb{E}[\mu^2] + \mu_0^2 - 2\mathbb{E}[\mu]\mu_0\right) + \frac{1}{2}\sum_{i=1}^{N}\left(x_i^2 + \mathbb{E}[\mu^2] - 2\mathbb{E}[\mu]x_i\right) \tag{14}$$

We can see that both μ_N and a_N remain constant and that λ_N and b_N change. Furthermore we know that:

$$\mathbb{E}[\mu] = \mu_N, \quad \mathbb{E}[\mu^2] = \frac{1}{\lambda_N} + \mu_N^2 \tag{15}$$

Thus λ_N is dependent on b_N and b_N is dependent on λ_N. We can solve these using an iterative process, where we begin by guessing λ_N, then computing b_N, and using this to compute λ_N. This process is repeated until the values of λ_N and b_N converge. This is usually a fairly quick process. Once λ_N and b_N have been determined we can calculate our factorized variational approximation of the posterior distribution by determining $q_\mu(\mu)$ and $q_\tau(\tau)$.

In the course book Bishop a closed form solution is presented for noninformative priors $\mu_0 = \lambda_0 = a_0 = b_0 = 0$, however that is not reproduced here since we want to use priors which are non-zero in the next subquestion.

2.3.10

We are looking for the exact posterior. We know what the normal and gamma distributions look like. Looking at our conjugate priors we have:

$$P(\mu|\tau)P(\tau) \propto \tau^{a_0 - \frac{1}{2}} \exp\left\{-\tau\left[b_0 + \frac{\lambda_0}{2}(\mu - \mu_0)^2\right]\right\} \tag{16}$$

$$= \tau^{a_0 - \frac{1}{2}} \exp\left\{-\tau\left[b_0 + \frac{\lambda_0}{2}(\mu^2 - 2\mu\mu_0 + \mu_0^2)\right]\right\} \tag{17}$$

$$= \tau^{a_0 - \frac{1}{2}} \exp\left\{-\tau\left[b_0 + \frac{\lambda_0\mu_0^2}{2} - \mu\lambda_0\mu_0 + \frac{\mu^2\lambda_0}{2}\right]\right\} \tag{18}$$

Which is a Normal-Gamma distribution. I.e we have $P(\mu|\tau)P(\tau) \sim \text{NormalGamma}(\mu_0, \lambda_0, a_0, b_0)$
Furthermore we know that the posterior is proportional to the prior times the likelihood, i.e that posterior $\propto$ prior $\times$ likelihood.

$$P(\mu, \tau|D) \propto \tau^{a_0 - \frac{1}{2}}\tau^{\frac{N}{2}} \exp\{-b_0\tau\} \exp\left\{-\frac{\lambda_0\tau}{2}(\mu - \mu_0)^2\right\} \exp\left\{-\frac{\tau}{2}\sum_{n=1}^{N}(x_n - \mu)^2\right\} \tag{19}$$

$$= \tau^{\left[a_0 - \frac{1}{2} + \frac{N}{2}\right]} \exp\left\{-b_0\tau - \frac{\lambda_0\tau}{2}(\mu - \mu_0)^2 - \frac{\tau}{2}\sum_{n=1}^{N}(x_n - \mu)^2\right\} \tag{20}$$

$$= \tau^{\left[a_0 - \frac{1}{2} + \frac{N}{2}\right]} \exp\left\{-b_0\tau - \frac{\lambda_0\tau}{2}(\mu^2 - 2\mu\mu_0 + \mu_0^2) - \frac{\tau}{2}\sum_{n=1}^{N}(x_n - \mu)^2\right\} \tag{21}$$

$$= \tau^{\left[a_0 - \frac{1}{2} + \frac{N}{2}\right]} \exp\left\{-\tau\left[b_0 + \frac{\lambda_0}{2}(\mu^2 - 2\mu\mu_0 + \mu_0^2) + \frac{1}{2}(N\bar{x^2} - 2N\bar{x}\mu + N\mu^2)\right]\right\} \tag{22}$$

$$= \tau^{\left[a_0 - \frac{1}{2} + \frac{N}{2}\right]} \exp\left\{-\tau\left[b_0 + \frac{(\lambda_0\mu_0^2 + N\bar{x^2})}{2} - \mu(\lambda_0\mu_0 + N\bar{x}) + \frac{\mu^2}{2}(\lambda_0 + N)\right]\right\} \tag{23}$$

Which is also NormalGamma distributed. Thus $P(\mu, \tau|D) \sim \text{NormalGamma}(\mu_N, \lambda_N, a_N, b_N)$. We can identify $\mu_N, \lambda_N, a_N, b_N$ by comparing terms since we can see that Eq. 23 is of the same form as Eq. 20. We see that:

$$a_N = a_0 + \frac{N}{2} \tag{24}$$

$$\lambda_N = \lambda_0 + N \tag{25}$$

$$\lambda_N \mu_N = \lambda_0 \mu_0 + N\bar{x} \tag{26}$$

$$b_N + \frac{\lambda_N \mu_N^2}{2} = b_0 + \frac{\lambda_0 \mu_0^2}{2} + \frac{N\bar{x^2}}{2} \tag{27}$$

We can solve for μ_N in Eq. 26 giving us:

$$\mu_N = \frac{\lambda_0 \mu_0 + N\bar{x}}{\lambda_0 + N} \tag{28}$$

Now we can use Equations 24-26 to determine b_N.

$$\begin{aligned}
b_N + \frac{\lambda_N \mu_N^2}{2} = b_0 + \frac{\lambda_0 \mu_0^2}{2} + \frac{N\bar{x^2}}{2} &\rightarrow 2b_N + \lambda_N \mu_N^2 = 2b_0 + \lambda_0 \mu_0^2 + N\bar{x^2} \\
&\rightarrow 2b_N = 2b_0 + \lambda_0 \mu_0^2 - \lambda_N \mu_N^2 + N\bar{x^2} \\
&= 2b_0 + \lambda_0 \mu_0^2 - (\lambda_0 + N)\left(\frac{\lambda_0 \mu_0 + N\bar{x}}{\lambda_0 + N}\right)^2 + N\bar{x^2} \\
&= 2b_0 + \lambda_0 \mu_0^2 + N\bar{x^2} - \frac{(\lambda_0 \mu_0 + N\bar{x})^2}{\lambda_0 + N} \\
&\Rightarrow b_N = b_0 + \frac{1}{2}\left[\lambda_0 \mu_0^2 + N\bar{x^2} - \frac{(\lambda_0 \mu_0 + N\bar{x})^2}{\lambda_0 + N}\right] \\
&\Rightarrow b_N = b_0 + \frac{1}{2}\left[\lambda_0 \mu_0^2 + \sum_{n=1}^{N} x_n^2 - \frac{(\lambda_0 \mu_0 + N\bar{x})^2}{\lambda_0 + N}\right]
\end{aligned}$$

Thus we have the exact posterior. It is NormalGamma distributed and we have

$$P(\mu, \tau|D) \sim \text{NormalGamma}(\mu_N, \lambda_N, a_N, b_N) = \tag{29}$$

$$= \text{NormalGamma}\left(\frac{\lambda_0 \mu_0 + N\bar{x}}{\lambda_0 + N}, \lambda_0 + N, a_0 + \frac{N}{2}, b_0 + \frac{1}{2}\left[\lambda_0 \mu_0^2 + \sum_{n=1}^{N} x_n^2 - \frac{(\lambda_0 \mu_0 + N\bar{x})^2}{\lambda_0 + N}\right]\right) \tag{30}$$

2.3.11

We begin by inspecting the case with broad, noninformative priors setting $\mu_0 = \lambda_0 = a_0 = b_0 = 0$, with dataset $D = [x_1, ...x_1 0]$ meaning that we have $N = 10$ datapoints which are iid Gaussians with $\mu = 0$ and $\sigma^2 = 1 \Rightarrow \tau = 1$.

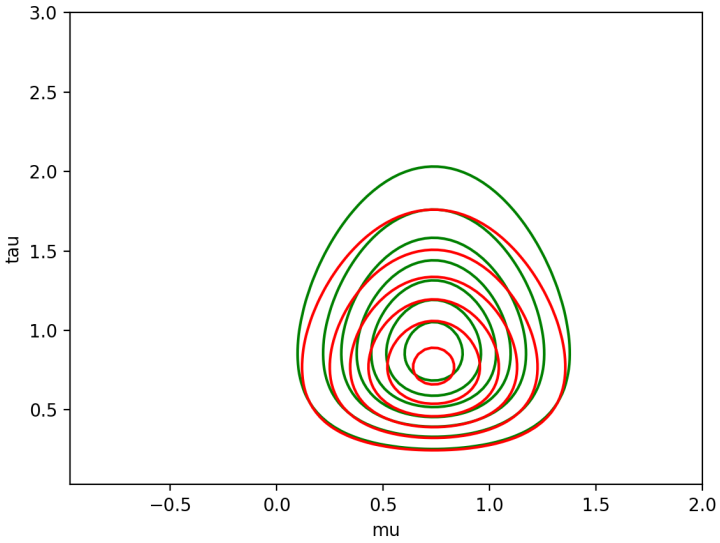


Figure 1: Exact posterior and inferred posterior plotted together. Green and red contours are exact and inferred posteriors respectively. For this iteration we had $N = 10$ datapoints drawn from iid Gaussians. Furthermore we have noninformative priors, i.e: $\mu_0 = \lambda_0 = a_0 = b_0 = 0$

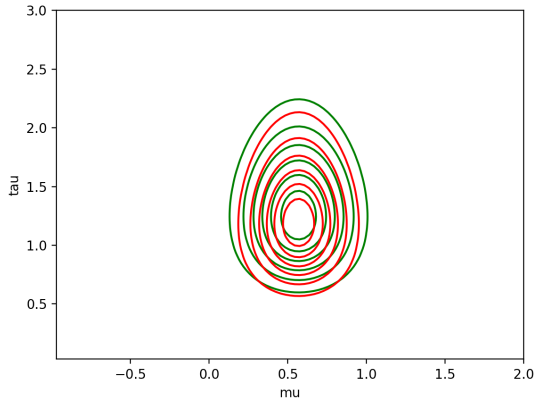
The results are shown in Fig. 1. When moving in the x-direction (μ) we can see that the exact posterior and the inferred posterior both look like a gaussian (they are symmetric) and in the y-direct (τ) it appears to be asymmetric (Similar to Figure 10.4 in Bishop).

Furthermore we might be interested in increasing the number of data points N and seeing what happens to our exact and inferred posteriors. We can see this shown in Fig. 2 where N is increased in increments of 10, i.e we have $N = 20, 30, 40, 50$. We see that as we increase the number of data points the contour plot the posterior appears to behave as a normal distribution in both the μ and τ axis.

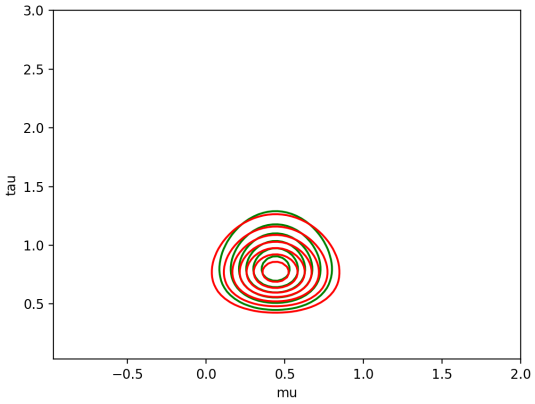
Next we may wish to play around with the priors. One thing we can do is to increase the scale factor of the prior a_0, as we can see this will also increase the scale factor of the posterior a_N. For large scale factors one can expect the gamma distribution to behave like a normal distribution with mean $\mu = a_N/b_N$ and variance $\sigma^2 = a_N/b_N^2$ or re-writing this in terms of precision we have $\tau = b_N^2/a_N$ (Important to note however that these are not the same μ and τ which we are plotting in our figures). Thus setting for example $a_0 = 10$ will result in $a_N = 15$ (Given that we still have $N = 10$). Leaving all other parameters unchanged, i.e $\mu_0 = \lambda_0 = b_0 = 0$ and setting $a_0 = 10$ results in the contour plot in Fig. 3. We can see in this case that the contours of the exact posterior (Green) looks more like a bivariate normal distribution. In this case we also have $b_N = 4.67621100250142$ thus $a_N/b_N \approx 3.2$ which we can see is approximately where the center of our posterior happens to be in the τ axis, which is what we thought would happen as we increase the scale factor.

One interesting thing to see in this case is that our VI algorithm does not give us a very accurate posterior as seen by the red contours. This is because the guess for λ_N given by our algorithm is incorrect. We can see that the exact value is $\lambda_N = 10$ and that our inferred value is $\lambda_{N,i} = 31.00801052870277$. As we increase N this value comes closer to the exact value, as expected (At $N = 50$ we get $\lambda_N = 50$ and $\lambda_{N,i} \approx 54$ which is a lot closer than earlier. (Note: This is not plotted).

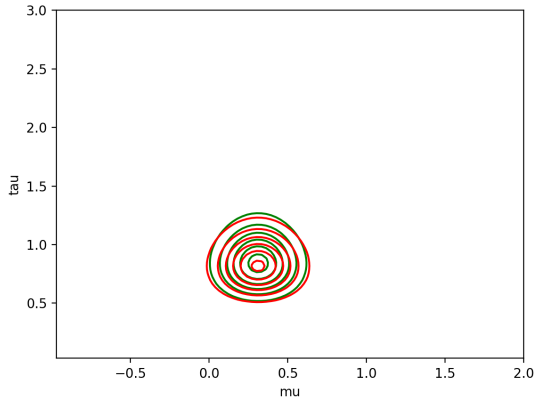
As we just noted the inferred posterior was not that great when we had a large value for a_0. We might now ask ourselves what happens if we instead have a large value for b_0? We try setting $\mu_0 = \lambda_0 = a_0 = 0$ and $b_0 = 10$, with $N = 10$ data points. The resulting contour plot is shown in Fig. 4. Note that the axes are



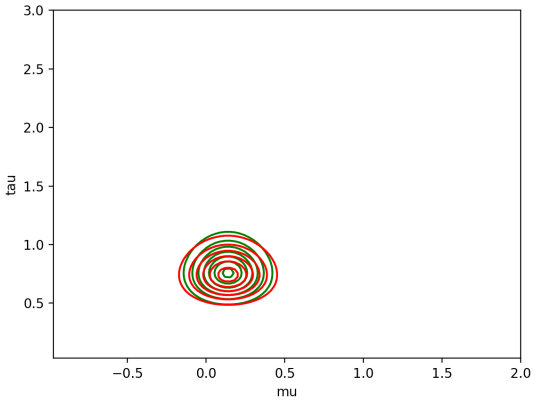
(a) $N = 20$



(b) $N = 30$



(c) $N = 40$



(d) $N = 50$

Figure 2: Contours of the exact and variationally inferred posterior (Green and red resp) behaviour as the number of data points N is increased. Notice that the axes are not rescaled. We have noninformative priors, i.e $\mu_0 = \lambda_0 = a_0 = b_0 = 0$.

different than in the other figures. We notice here that the inferred posterior is also not too good and instead of undershooting on the μ axis as was the case when we set $a_0 = 10$ it overshoots. In this case we see that the inferred value of λ_N is smaller than the exact value, i.e the opposite of what we had when we set $a_0 = 10$.

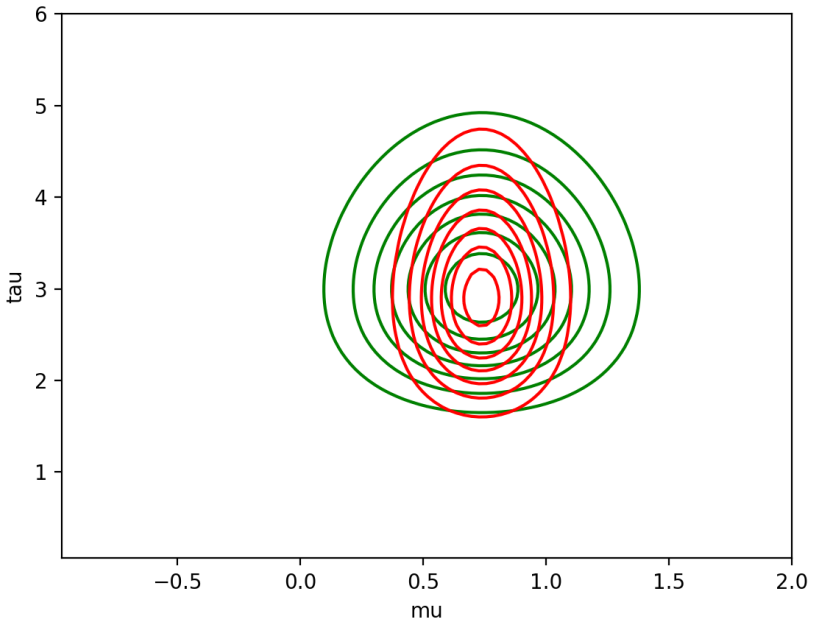


Figure 3: Exact posterior and inferred posterior plotted together. Green and red contours are exact and inferred posteriors respectively. For this iteration we had $N = 10$ datapoints drawn from iid Gaussians, and set $a_0 = 10$, with remaining parameters zero.

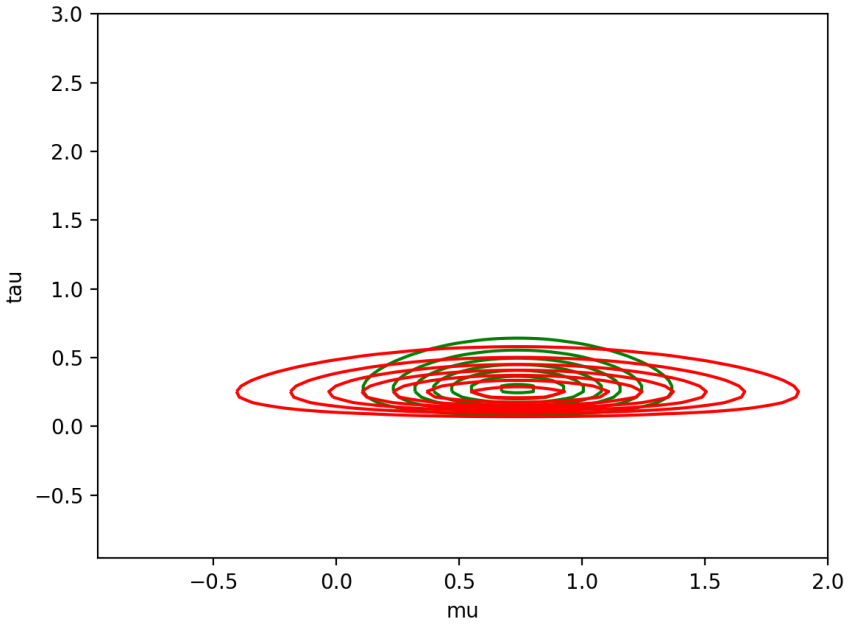


Figure 4: Exact posterior and inferred posterior plotted together. Green and red contours are exact and inferred posteriors respectively. For this iteration we had $N = 10$ datapoints drawn from iid Gaussians, and set $b_0 = 10$, with remaining parameters zero.

A Code : Likelihood of a Tree Graphical Model

```python
import numpy as np
from Tree import Tree
from Tree import Node

#This helper function takes the topology of a tree and returns a children_dictionary
#The dictionary has keys = node and the values is a list of its children
def children_dictionary(num_nodes,topology):
    children = dict()
    for node in range(num_nodes):
        tempdict = {node : list(np.where(topology == node)[0])}
        children.update(tempdict)

    children = {key: value for key, value in children.items() if value}
    return children


def calculate_likelihood(num_nodes, topology,K , theta, beta):
    # Initialize S(u,j). It is an num_nodes times K matrix.
    # Each row corresponds to a node and its probability of
    S = np.zeros((num_nodes,K))
    bookkeeping = []

    children = children_dictionary(num_nodes,topology)
    #We set the NaN values to -1 so that we can remake the objects
    # in the beta array as integers.
    beta[np.isnan(beta)] = -1
    beta = beta.astype(int)

    #This loop looks through beta, and for the leaf nodes we set the
    #S value to be 1 since it is observed.
    #We also append indicies of the leaf nodes into a list.
    #Ignores all the NaNs.
    for i, b in enumerate(beta):
        if b != -1:
            S[i,b] = 1
            #print(i,b)
            bookkeeping.append(i)
    #Go through the list of leaf nodes, from the very deepest node.
    #Sort at the start so we are sure we are working with the deepest node.
    #We add the parent node to the list so we can go through it on the next iteration.
    #Finally we determine S(u,i) for the parent node
    while len(bookkeeping) > 0:
        bookkeeping.sort()
        node = bookkeeping.pop()
        #If we come to the root node we break.
        if node == 0:
            break
        #Finding the index of the father of the current node
        parent = topology[node]

        bookkeeping.append(parent)
```

```
52
53          #We set the row equal to 1 since we will be multiplying with the probabilities later.
54          S[parent] = 1
55          #Loop over all possible values K which the parent node can assume.
56          for a in range(K):
57              #Looking at each child on the parent node
58              for child in children[parent]:
59                  S[parent, a] *= np.sum(S[child,:]*theta[child][a])
60      #Determining the likelihood which is what we want.
61      L = np.dot(S[0], theta[0])
62      return L
63
64
65  def main():
66      trees = ['small','medium','large'] #The three different trees
67      for tree in trees:  #Go through each tree
68          filename = "data/q2_2/q2_2_"+str(tree)+"_tree.pkl"
69
70          t = Tree()
71          t.load_tree(filename)
72
73          theta = t.get_theta_array()
74          num_nodes = t.num_nodes
75          topology = t.get_topology_array().astype(int)
76          K = t.k
77          #Calculating the likelihoods of each sample in each tree.
78          for sample_idx in range(t.num_samples):
79            beta = t.filtered_samples[sample_idx]
80            print("\n\tSample: ", sample_idx)#, "\tBeta: ",beta)
81            sample_likelihood = calculate_likelihood(num_nodes, topology,K, theta, beta)
82            print("\tLikelihood: ", sample_likelihood)
83
84  if __name__ == "__main__":
85      main()
```

B Code : Simple Variational Inference

```
1   import matplotlib.pyplot as plt
2   import numpy as np
3   from scipy.stats import norm
4   from scipy.stats import gamma
5
6
7   def q_mu(x,mu,lamb):
8       return norm.pdf(x, mu, np.sqrt(1 / lamb))
9
10  def q_tau(tau,a,b):
11      return gamma.pdf(tau, a, loc = 0, scale =1 / b)
12
13  def exp_mu_N_squared(lambda_N,mu_N):
14      return (1/lambda_N)+mu_N**2
15
```

```
def b_N(b_0,lambda_0,mu_0,exp_mu_N,exp_mu_N_squared,D):
    return b_0 + 0.5*lambda_0*(exp_mu_N_squared+mu_0**2-2*exp_mu_N*mu_0) + 0.5*np.sum(np.square(D)+exp_m

def lambda_N(lambda_0,N,a_N,b_N):
    return (lambda_0+N)*(a_N/b_N)

#setting a seed so we can get same results every time
np.random.seed(0)
#Setting our priors
[mu_0,lambda_0,a_0,b_0] = [0,0,0,10]

#Number of data points
N = 10
D = np.random.normal(0 , 1 ,N) #mean 0 sigma 1

#Looking at 100 different values of tau and mu
gridpoints = 100
x = np.linspace(-1,2,gridpoints)[1:] #tau
y = np.linspace(-1,3,gridpoints)[1:] #mu

#Creating a meshgrid
x_mu, y_tau = np.meshgrid(x,y,indexing ='ij')

#Initializing an empty matrix for our exact posterior
P = np.zeros((np.size(x),np.size(y)))

#Exact posterior derived in the assignment.
a_N_exact = a_0 + 0.5*N
b_N_exact = b_0 + 0.5*(lambda_0*mu_0**2 + np.sum(D**2)-((lambda_0*mu_0+N*np.mean(D))**2)/(lambda_0+N))
mu_N_exact = (lambda_0*mu_0+N*np.mean(D))/(lambda_0+N)
lambda_N_exact = lambda_0+N
print("------------")
print('b_N = ',b_N_exact)
print('lambda_N = ',lambda_N_exact)
print("------------")
for i,mu in enumerate(x):
    for j,tau in enumerate(y):
        P[i,j] = q_mu(mu,mu_N_exact,lambda_N_exact)*q_tau(tau,a_N_exact,b_N_exact)

#Plotting the exact posterior
b = plt.contour(x_mu,y_tau,P,colors='g')


"-------------------------------------------"
#Beginning the variational inference here.
# mu_N and a_N do not vary so they are created as constants here.
mu_N = (lambda_0 * mu_0 + np.sum(D)) / (lambda_0 + N)
a_N = a_0+0.5*N

#How many iterations we want for our VI algorithm.
#There is probably a better way of doing this.
#but I just did it visually. 10 iterations us usually more than enough.
iterations = 10

```

```
#Initial guess for lambda.
l_guess = 1
#First iteration
b = b_N(b_0,lambda_0,mu_0,mu_N,exp_mu_N_squared(l_guess,mu_N),D)
l = lambda_N(lambda_0,N,a_N,b)

#Remaining iterations
for i in range(iterations):
    b = b_N(b_0,lambda_0,mu_0,mu_N,exp_mu_N_squared(l,mu_N),D)
    l = lambda_N(lambda_0,N,a_N,b)
    print("------------")
    print('b guess',b)
    print('lambda guess',l)
    print("------------")


#Initializing the factorized variational approximation matrix
Q = np.zeros((np.size(x),np.size(y)))
for i,mu in enumerate(x):
    for j,tau in enumerate(y):
        Q[i,j] = q_mu(mu,mu_N,l)*q_tau(tau,a_N,b)

#Plotting the VI posterior
a = plt.contour(x_mu,y_tau,Q,colors='r')
#Fixing some labels
plt.xlabel("mu")
plt.ylabel("tau")
plt.show()
```