DD2434 Machine Learning, Advanced Course Assignment 2

Alice Karnsund

January 2018

In this assignment I exchange part 2.2 to part 2.6 for the E level. *Anyhow, I did parts of 2.2 and I left those results in here.*

I. Graphical Models

2.1 Dependence in a Directed Graphical Model

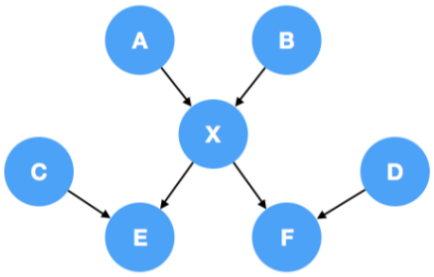

Figure 1: The DAG

Question 1: *Which pairs of variables, not including X, are dependent conditioned on X?*

Considering 3 nodes at a time (including X).
If we condition on X, the conditional distribution of A and B is

$$p(A, B|X) = \frac{p(A, B, X)}{p(X)} = \frac{p(A)p(B)p(X|A, B)}{p(X)}$$

which in general does not factorize into $p(A)p(B)$ and therefore A and B are dependent given X, observing X unblocks the path between A and B [1]. As for the nodes A and E, conditioned on X we get,

$$p(A, E|X) = \frac{p(A, E, X)}{p(X)} = \frac{p(A)p(X|A)p(E|X)}{p(X)} = p(A|X)p(E|X)$$

[1] Bishop, Pattern Recognition And Machine Learning, Springer, 2006, Page 376

which says that A and E are conditional independent given X. The same goes for the pairs A and F, B and F and B and E. For the nodes E and F conditioned on X we get,

$$p(E, F|X) = \frac{p(E, F, X)}{p(X)} = p(E|X)p(F|X)$$

which also is a conditional independence property given X. Considering the nodes F and D given X we obtain,

$$p(D, F|X) = \frac{p(D, F, X)}{p(X)} = \frac{p(D)p(X)p(F|X, D)}{p(X)} = p(D)p(F|X, D)$$

which says that F and D are dependent given X. The same argument goes for C and E.
Since the nodes A and E are independent given X, the nodes A and C are independent given X, the same goes for A and D, B and C and B and D. Also, because E and F are conditionally independent given X, this implies that E and D are also independent. A corresponding argument goes for F and C. Finally C and D are independent due to the fact that E and F are conditionally independent given X.

Summarizing, the only dependent pairs of variables that are dependent conditioned on X are A and B, F and D, C and E.

Question 2: *Which pairs of variables, not including X, are dependent, not conditioned on X?*

We consider 3 nodes at a time (including X).
First we consider A and B, the joint distribution of the 3 nodes is,

$$p(A, B, X) = p(A)p(B)p(X|A, B)$$

marginalizing both sides over X gives; $p(A, B) = p(A)p(B)$. So A and B are independent when X is not observed. The joint distribution for A, E and X can be written,

$$p(A, X, E) = p(A)p(X|A)p(E|X)$$

marginalizing over X we get, $p(A, E) = p(A)p(E|A)$ which implies that A and E are dependent when X is not observed. The same goes for A and F, B and E, and B and F. Now considering the nodes E and F,

$$p(E, F, X) = p(E|X)p(F|X)p(X) \rightarrow p(E, F) = \sum_X p(E|X)p(F|X)p(X)$$

This does not, in general, factor to $p(E)p(F)$, thus E and F are dependent when X is not observed.
The joint distribution for the nodes C and E is,

$$p(C, E, X) = p(C)p(E|C, X)p(X) \quad \rightarrow$$
$$p(C, E) = p(C) \sum_X p(E|C, X)p(X) = p(C)p(E|C)$$

and therefor E and C are dependent not conditioned on X. The same goes for the nodes D and F.
Turning to joint distributions including 4 nodes (not writing out the nodes, as before, for clarity), we examine the dependence between A and D,

$$\begin{aligned} p(A,X,F,D) &= p(A)p(X|A)p(F|X,D)p(D) \\ p(A,F,D) &= \sum_X p(A,X,F,D) = p(A)p(D)p(F|A,D) \\ p(A,D) &= \sum_F p(A,F,D) = p(A)p(D) \end{aligned}$$

From this we see that A and D are independent when not conditioned on X. The same result is obtained for B and C, A and C, B and D.
For nodes C and D,

$$\begin{aligned} p(E,X,F,D) =& p(E|X)p(X)p(F|X,D)p(D) \rightarrow \\ p(E,F,D) =& p(D)\sum_X p(E|X)p(F|X,D)p(X) \end{aligned}$$

This does not, in general, factor to $p(D)p(E)p(F|D)$, thus E and D are dependent when X is not observed. The same goes for F and C.
With the same argument when considering the joint distribution,

$$\begin{aligned} p(C,E,X,F,D) =& p(C)p(E|C,X)p(X)p(F|X,D)p(D) \rightarrow \\ p(C,E,F,D) =& p(C)p(D)\sum_X p(E|C,X)p(F|X,D)p(X) \end{aligned}$$

we will not in general get that this factor to $p(C)p(D)p(E|C)p(F|D)$, therefor C and D are dependent when X is not observed.

Summarizing, when not conditioning on X we get that; A is dependent of E and F. B is dependent of E and F. E and F are both dependent on all the other nodes (not considering X). C and D are dependent on E and F.

2.2 The Sum-HMM

Question 3: *Implement the Sum-HMM, i.e., write your own code for it.*

Z_k are the hidden (latent) variables in our standard HMM model. These variables can take on two different values depending on what group of tables they are in, value 0 if in group 1 or 1 if in group 2, since it is easier to use 0 and 1 when working with a binomial distribution. A player visit K tables and switching tables can be done according to,

$$\begin{aligned} Z_{k-1} = i \quad &\rightarrow \quad Z_k = i \;\; with\ prob: \frac{1}{4} \\ Z_{k-1} = i \quad &\rightarrow \quad Z_k = 1-i \;\; with\ prob: \frac{3}{4} \qquad i = 0,1 \end{aligned}$$

$X_k^n \sim p(X_k|Z_k = t_k^i$ is the outcome of the dice at table k and is observed with probability p and hidden with probability $1-p$.

A player n visits K tables and a hidden or not hidden value, outcome of each tables dice, is added to a sum $S^n = \sum_{k=1}^{K} X_k^n$. At the end of the round we return the sum and the sequence X^n of dice outcomes, where an element is shown with probability p and hidden with probability $1 - p$.

Code:

```
# Author: Alice Karnsund
import numpy as np


# Create a array of the K tables visited for player n
def tableRoute(K, startProb):
        tables = np.zeros(K)
        # Choosing first table group
        start = np.random.binomial(1, startProb, 1)
        tables[0] = start
        for i in range(tables.shape[0]-1):
                if tables[i] == 1:
                        following = np.random.binomial(1, 1/4, 1)
                else:
                        following = np.random.binomial(1, 3/4, 1)
                tables[i+1] = following

        # Checking when the player is at either table group
        k1 = np.where(tables == 0)[0]
        k2 = np.where(tables == 1)[0]
        return(tables, k1, k2)


# Create an array that returns the observations from all tables
def diceOutcome(tables, k1, k2, distr1, distr2):
        # Results from each table group
        results1 = np.zeros(len(k1))
        results2 = np.zeros(len(k2))
        # All results
        results = np.zeros(len(tables))
        i = 0
        j = 0
        for k in range(len(tables)):
                if tables[k] == 0:
                        # Throws each tables dice once
                        prob = np.random.multinomial(1, distr1[k,:])
                        results1[i] = np.where(prob == 1)[0]+1
                        i+=1
                else:
                        prob = np.random.multinomial(1, distr2[k,:])
                        results2[j] = np.where(prob == 1)[0]+1
                        j+=1
```

```
                results[k] = np.where(prob == 1)[0]+1
        return(results1, results2, results)


# different categorical distributions
def catDist(K):
        # Biased dice in a random way, different at each table
        catTables1 = np.zeros((K, 6))
        catTables2 = np.zeros((K, 6))
        for k in range(K):
                rand1 = np.random.rand(1,6)[0]
                s1 = sum(rand1)
                rand1 = rand1/s1
                catTables1[k]=rand1

                rand2 = np.random.rand(1,6)[0]
                s2 = sum(rand2)
                rand2 = rand2/s2
                catTables2[k]=rand2

        # Fair dice for every table in table group
        catEqual = np.ones((K,6))/6

        # Every other table in a group has a biased
        # and every other has a fair dice
        catMix = np.ones((K,6))/6
        for k in range(0,K,2):
                catMix[k] = catTables1[k]

        # Extremely biased dice for every table in table group.
        # Dice will only give 6
        cat6 = np.zeros((K,6))
        cat6[...,5] = np.ones(K)

        # Extremely biased dice for every table in table group.
        # Dice will only give 1
        cat1 = np.zeros((K,6))
        cat1[...,0] = np.ones(K)

        return(catTables1, catTables2, catEqual, catMix, cat6, cat1)



def observations(K, results, p):
        sum_n = sum(results)
        X_n = results
        observations = np.zeros(K)
        # prob = 1 indicates that the outcome is observed,
        # and prob = 0 means that it is hidden
```

```
        for k in range(K):
                prob = np.random.binomial(1, p, 1)
                obs = prob*X_n[k]
                observations[k] = obs
        return(observations, sum_n)


def allPlayers(N, K, p):
        # Array with the outcome sum for each player
        allSums = np.zeros(N)
        # Matrix with the observed outcome sequence
        # for each player (row)
        allSeq = np.zeros((N,K))
        # All "hidden" and "un-hidden" outcomes for
        # each player (row), just to check!
        allResults = np.zeros((N,K))
        for n in range(N):
                tables, k1, k2 = tableRoute(K, 0.5)
                distr1, distr2, catE, catM, cat6, cat1 = catDist(K)
                r1, r2, results = diceOutcome(tables, k1, k2, cat1, cat1)
                playerObs, playerSum = observations(K, results, p)
                allSums[n] = playerSum
                allSeq[n] = playerObs
                allResults[n] = results
        print()
        print('All sums')
        print(allSums)
        print('All hidden and unhidden outcomes')
        print(allResults)
        print('All observed outcomes')
        print(allSeq)
        return(allSums,allSeq)


# Choose how many players in Casio
N = 6
# Choose how many tables each player will visit
K = 10
# Choose probability for observing a dice outcome
p = 0.7
allPlayers(N, K, p)
```

Question 4: & Question 5: *Provide data generated using at least three different sets of categorical dice distributions that provide reasonable tests for the correctness of your program.*
Motivate your test and why the result of it indicates correctness.

In the code in Question 3 I provided six different sets of categorical dice distributions. Two are just randomly distributed, one is equally distributed (normal dices), one is a mix, and the last two are extremely biased and will give the same outcome every time (1 or 6). These distribution sets all are developed for one table group. To test the correctness of the program the last two distributions are beneficial. Because if we all the dices in both table groups are biased such that they can only give one, the sum for each player must be K. And in the same sense all players sums must be 6K if all the dices are biased in a way to only give 6. Thus the value of each player sum can only vary according to, $K \leq S^n \leq 6K$ since every player visits K tables. To support this with data we use the code just as it is above (exchange to cat6 second round), where we have 6 players, they visit 10 tables and the probability to observe an outcome of a dice is $p = 0.7$.

```
All sums
[ 10.  10.  10.  10.  10.  10.]
All hidden and unhidden outcomes
[[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]]
All observed outcomes
[[ 1.  1.  0.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  0.]
 [ 1.  1.  1.  1.  0.  1.  1.  0.  1.  1.]
 [ 0.  1.  1.  1.  1.  1.  1.  1.  0.  1.]
 [ 1.  0.  0.  1.  1.  0.  1.  1.  1.  0.]
 [ 1.  1.  1.  1.  1.  1.  1.  0.  1._ 1.]]
```

Figure 2: Outcomes for 6 players visiting 10 tables each and p=0.7. Here a biased distribution, only giving one, is used. Note that the first matrix is not given to us, only the second matrix ("All observed outcomes") and the sequence ("All sums"). The zeros in the second matrix are the censored unobserved outcomes.

This shows that all 6 players get the sum 10 when throwing a biased dice, only capable of giving 1, at each of the 10 tables they visit. Thus this supports the correctness of the program.
Next we have the upper bound, where the players throw a biased dice, only capable of giving 6 each time.

```
All sums
[ 60.  60.  60.  60.  60.  60.]
All hidden and unhidden outcomes
[[ 6.  6.  6.  6.  6.  6.  6.  6.  6.  6.]
 [ 6.  6.  6.  6.  6.  6.  6.  6.  6.  6.]
 [ 6.  6.  6.  6.  6.  6.  6.  6.  6.  6.]
 [ 6.  6.  6.  6.  6.  6.  6.  6.  6.  6.]
 [ 6.  6.  6.  6.  6.  6.  6.  6.  6.  6.]
 [ 6.  6.  6.  6.  6.  6.  6.  6.  6.  6.]]
All observed outcomes
[[ 6.  6.  0.  0.  0.  6.  6.  6.  6.  6.]
 [ 0.  6.  6.  6.  0.  6.  6.  0.  6.  6.]
 [ 0.  6.  0.  6.  6.  6.  0.  0.  0.  0.]
 [ 6.  6.  0.  6.  6.  6.  6.  0.  6.  6.]
 [ 6.  6.  6.  6.  6.  6.  6.  6.  6.  0.]
 [ 0.  0.  0.  6.  6.  6.  6.  6.  6.  6.]]
```

Figure 3: Outcomes for 6 players visiting 10 tables each and p=0.7. Here a biased distribution, only giving six, is used.Note that the first matrix is not given to us, only the second matrix ("All observed outcomes") and the sequence ("All sums"). The zeros in the second matrix are the censored unobserved outcomes.

Now this shows that all players will achieve a sum of 60 with this specific distribution, and again this supports the correctness of the program.
Considering another categorical distribution, like the mixed one which is a mixture between randomly biased dices and fair dices at all 2K tables we get for example,

```
All sums
[ 43.  24.  35.  33.  45.  36.]
All hidden and unhidden outcomes
[[ 5.  5.  6.  3.  4.  1.  5.  5.  3.  6.]
 [ 2.  4.  1.  3.  2.  5.  3.  2.  1.  1.]
 [ 2.  1.  4.  3.  6.  3.  3.  4.  6.  3.]
 [ 2.  6.  4.  4.  4.  1.  3.  4.  1.  4.]
 [ 6.  6.  6.  5.  6.  3.  4.  3.  2.  4.]
 [ 2.  5.  3.  3.  6.  6.  2.  2.  3.  4.]]
All observed outcomes
[[ 5.  5.  6.  0.  4.  1.  5.  5.  3.  0.]
 [ 2.  4.  1.  0.  2.  5.  3.  2.  0.  1.]
 [ 0.  1.  0.  3.  0.  0.  0.  4.  6.  3.]
 [ 0.  6.  0.  4.  4.  1.  0.  0.  1.  0.]
 [ 0.  6.  0.  0.  0.  0.  0.  0.  2.  0.]
 [ 2.  5.  3.  3.  0.  6.  2.  2.  3.  4.]]
```

Figure 4: Outcomes for 6 players visiting 10 tables each and p=0.7. Here a mixed distribution is used.Note that the first matrix is not given to us, only the second matrix ("All observed outcomes") and the sequence ("All sums"). The zeros in the second matrix are the censored unobserved outcomes.

The sums here can take on any value $K \leq S^n \leq 6K$, but as the number of tables visited K grows the sums will tend to a certain mean value due to the

central limit theorem. Therefore it will be unusual to see extreme values (close to K or 6K) in the sum sequence. To show this, consider the case of $K = 2$ and $K = 100$ and with the mixture distribution,

$$\begin{aligned} K = 2 &\sim [12.\ \ 12.\ \ 2.\ \ 8.\ \ 8.\ \ 10.] \\ K = 100 &\sim [339.\ \ 365.\ \ 341.\ \ 374.\ \ 351.\ \ 368.] \end{aligned}$$

From this we see that when $K = 2$ the sums S^n can taken on both the upper and lower bound values (i.e. 2 and 12), and this is pretty common and can be seen when running the program. When $K = 100$ the lower an upper bound values (i.e. 200 and 600) are never seen (or you are extremely lucky). All the S^n values tends to the values around the mean value $S^n_{mean} = 350$.
With this said and with the data given, the program fulfills the desired criteria for this casino game.

2.3 Simple VI

Question 8: *Implement the VI algorithm for the variational distribution in Equation (10.24) in Bishop.*

We are given the model,

$$\begin{aligned} Likelihood: \ p(D|\mu,\tau) &= (\frac{\tau}{2\pi})^{N/2} exp(-\frac{\tau}{2}\sum_{n=1}^{N}(x_n-\mu)^2) \\ Conjugate\ priors: \ p(\mu|\tau) &= \mathcal{N}(\mu|\mu_0,(\lambda_0\tau)^{-1}) \\ p(\tau) &= Gam(\tau|a_0,b_0) \end{aligned}$$

Our goal is the to approximate the true posterior distribution, using (eq. 10.24 in Bishop),

$$q(\mu,\tau) = q_\mu(\mu)q_\tau(\tau) \sim \mathcal{N}(\mu|\mu_N,\lambda_N^{-1})Gam(\tau|a_N,b_N) \tag{1}$$

This can be done with an iterative process where we update the parameters for q_μ and q_τ. Following the steps in one of the exercises and the equivalent example, to the one in Bishop, in [2], these parameters can be written,

$$\begin{aligned} \mu_N &= \frac{\lambda_0\mu_0 + N\overline{x}}{\lambda_0 + N} \\ \lambda_N &= (\lambda_0 + N)E_\tau[\tau] \\ a_N &= a_0 + \frac{N+1}{2} \\ b_N &= b_0 + \frac{1}{2}E_\mu[\lambda_0(\mu-\mu_0)^2 + \sum_{n=1}^{N}(x_n-\mu)^2] \\ &= b_0 - E_\mu[\mu](\lambda_0\mu_0 + \sum_{i=1}^{N}x_i) + \frac{1}{2}(E_\mu[\mu^2](\lambda_0+N) + \lambda_0\mu_0^2 + \sum_{i=1}^{N}x_i^2) \end{aligned}$$

[2] Murphy, Machine Learning: A Probabilistic Perspective, 2012, Chap 21.5

Where the moments are,

$$E_\mu[\mu] = \mu_N$$
$$E_\mu[\mu^2] = \frac{1}{\lambda_N} + \mu_N^2$$
$$E_\tau[\tau] = \frac{a_N}{b_N}$$

Implementing this we get,
Code:

```
# Author: Alice Karnsund
import numpy as np
from math import pi, exp
import matplotlib.pyplot as plt
from scipy.special import gamma

# Our goal is to infer the posterior distribution for the
# mean mu and precision tau

# generate some data drawn independently from a Gaussian
def generateData(N):
        # goal values
        mu = 0
        sigma = 1          # tau = 1/sigma = 1
        D = np.random.normal(mu, sigma, N)
        return(D)


# Compute the moments
def moments(aN, bN, muN, lambdaN):
        E_mu = muN         # constant
        E_mu2 = (1/lambdaN) + muN**2
        E_tau = aN/bN
        return(E_mu, E_mu2, E_tau)


# Posterior approximation
def q(aN, bN, muN, lambdaN, mu, tau):
        q_mu = (1/gamma(aN))*(bN**aN * tau**(aN-1) * np.exp(-bN*tau))
        q_tau = (lambdaN/(2*pi))**(1/2)
        q_tau = q_tau * np.exp(-0.5*np.dot(lambdaN,((mu-muN)**2).T))
        q = q_mu*q_tau
        return(q)


# plot the contours for a distribution
def plotPost(mu, tau, approx, color, i):
```

```
        muGrid, tauGrid = np.meshgrid(mu, tau)
        plt.contour(muGrid, tauGrid, approx, colors = color)
        plt.title('Posterior approximation after '+str(i)+' iterations')
        plt.xlabel('$\mu$')
        plt.ylabel('tau')
        plt.show()


# VI step, update parameters
def updateParameters(N, D, a0, b0, mu0, lambda0, bN, lambdaN, mu, tau):
        # Some parameter values will be constant
        x_mean = (1/N)*sum(D)
        muN = (lambda0*mu0 + N*x_mean)/(lambda0 + N)
        aN = a0 + (N+1)/2
        iterations = 7
        for i in range(iterations):
                E_mu, E_mu2, E_tau = moments(aN, bN, muN, lambdaN)
                lambdaN = (lambda0+N)*E_tau
                bN = b0-E_mu*(lambda0*mu0+sum(D))
                bN = bN+0.5*(E_mu2*(lambda0+N)+lambda0*mu0**2+sum(D**2))

                # Calculate q for new values of the parameters
                approx = q(aN,bN,muN,lambdaN,mu[:,None],tau[:,None])

                # Plot the posterior approximation
                color = 'b'
                if i == iterations-1:
                        color = 'r'

                plotPost(mu, tau, approx, color, i)



# Generate some data points
N = 10
D = generateData(N)

# Choose parameter values for the conjugate prior
a0 = 3
b0 = 5
mu0 = 0.2
lambda0 = 15

# Start values
bN = 3
lambdaN = 15

# Generate some mu and tau values for the plot
mu = np.linspace(-0.5,1,100)
tau = np.linspace(0,2,100)
```

```
# Run VI
updateParameters(N, D, a0, b0, mu0, lambda0, bN, lambdaN, mu, tau)
```

Question 9: *Describe the exact posterior.*

Since the conjugate prior distribution is a Gaussian-gamma distribution and the likelihood is a Gaussian distribution the true posterior will take the form of a Gaussian-gamma distribution [3]. If the true Gaussian-gamma posterior is written $p(\mu, \tau|\mu_T, \lambda_T, a_T, b_T)$, where the index T stands for "True". Then our goal is to find the parameter values for $\mu_T, \lambda_T, a_T, b_T$. Making use of the steps in [4], we can reach the correct expressions for the parameters.
Our conjugate prior distributions are,

$$p(\mu|\tau) = \mathcal{N}(\mu|\mu_0, (\lambda_0\tau)^{-1})$$
$$p(\tau) = Gam(\tau|a_0, b_0)$$

Since the two expressions are coupled the total prior $p(\mu, \tau)$ has a Gaussian-gamma distribution that can be written as,

$$p(\mu, \tau|\mu_0, \lambda_0, a_0, b_0) = \frac{b_0^{a_0}\sqrt{\lambda_0}}{\Gamma(a_0)\sqrt{2\pi}}\tau^{a_0-1/2}e^{-b_0\tau}e^{-\frac{\lambda_0\tau(\mu-\mu_0)^2}{2}}$$

The observed data points $x_i, i = 1, ..., N$ are assumed to be drawn independently from a Gaussian distribution (Bishop 470). The likelihood can be written as,

$$\begin{aligned} p(D|\mu, \tau) &= \prod_{i=1}^{N}(\frac{\tau}{2\pi})^{1/2}e^{-\frac{\tau}{2}(x_i-\mu)^2} \\ &\propto \tau^{N/2}e^{-\frac{\tau}{2}\sum_{i=1}^{N}(x_i-\overline{x}+\overline{x}-\mu)^2} \\ &\propto \tau^{N/2}e^{-\frac{\tau}{2}\sum_{i=1}^{N}((x_i-\overline{x})^2+(\overline{x}-\mu)^2)} \\ &\propto \tau^{N/2}e^{-\frac{\tau}{2}(Ns+N(\overline{x}+N(\overline{x}-\mu)^2)} \end{aligned}$$

Where $\overline{x} = \frac{1}{N}\sum_{i=1}^{N}x_i$ and $s = \frac{1}{N}\sum_{i=1}^{N}(x_i-\overline{x})^2$. With this information at hand we can derive the posterior distribution over μ and τ using Bayes' theorem.

$$\begin{aligned} p(\mu, \tau|D) &\propto p(D|\mu, \tau)p(\mu|\tau)p(\tau) \\ &\propto \tau^{N/2}e^{-\frac{\tau}{2}(Ns+N(\overline{x}+N(\overline{x}-\mu)^2)}\tau^{a_0-1/2}e^{-b_0\tau}e^{-\frac{\lambda_0\tau(\mu-\mu_0)^2}{2}} \\ &= \tau^{\frac{N-1}{2}+a_0}e^{-\tau(\frac{1}{2}Ns+b_0)}e^{-\frac{\tau}{2}(\lambda_0(\mu-\mu_0)^2+N(\overline{x}-\mu)^2)} \end{aligned}$$

To be able to identify the parameters, we want an expression with μ minus something in the exponent, raised to the power of 2. By completing the square in the last term we get,

$$p(\mu, \tau|D) \propto \tau^{\frac{N-1}{2}+a_0}e^{-\tau(\frac{1}{2}Ns+b_0+\frac{\lambda_0\mu_0(\overline{x}-\mu_0)^2}{2(\lambda_0+N)})}e^{-\frac{\tau}{2}(\lambda_0+N)(\mu-\frac{\lambda_0\mu_0+N\overline{x}}{\lambda_0+N})^2}$$

[3] Bishop, Pattern Recognition And Machine Learning, Springer, 2006, Page 470
[4] Wikipedia, Normal-gamma distribution, July 4, 2017, https://en.wikipedia.org/wiki/Normal-gamma_distribution

This expression is exactly the one of a Gaussian-gamma distribution, from which we can identify our true distribution parameters.

$$\mu_T = \frac{\lambda_0\mu_0 + N\overline{x}}{\lambda_0 + N}$$
$$\lambda_T = \lambda_0 + N$$
$$a_T = a_0 + \frac{N}{2}$$
$$b_T = b_0 + \frac{1}{2}\sum_{i=1}^{N}(x_i - \overline{x})^2 + \frac{\lambda_0 N(\overline{x} - \mu_0)^2}{2(\lambda_0 + N)}$$

With these parameter values the exact posterior will be,

$$p(\mu, \tau|D) = \frac{b_T^{a_T}\sqrt{\lambda_T}}{\Gamma(a_T)\sqrt{2\pi}}\tau^{a_T - 1/2}e^{-b_T\tau}e^{-\frac{\lambda_T\tau(\mu-\mu_T)^2}{2}} \tag{2}$$

Question 10: *Compare the variational distribution with the exact posterior. Run the inference for a couple of interesting cases and describe the difference.*

With addition of a few functions to the previous code we are able to compare the true posterior to the VI approximation.
Code:

```
# Author: Alice Karnsund
import numpy as np
from math import pi, exp, sqrt
import matplotlib.pyplot as plt
from scipy.special import gamma

# Our goal is to infer the posterior distribution for the
# mean mu and precision tau

# generate some data drawn independently from a Gaussian
def generateData(N):
        # Goal values
        mu = 0
        sigma = 1       # tau = 1/sigma = 1
        D = np.random.normal(mu, sigma, N)
        return(D)


# Compute the moments
def moments(aN, bN, muN, lambdaN):
        E_mu = muN          # constant
        E_mu2 = (1/lambdaN) + muN**2
        E_tau = aN/bN
        return(E_mu, E_mu2, E_tau)
```

```
# Posterior approximation
def q(aN, bN, muN, lambdaN, mu, tau):
        q_mu = (1/gamma(aN))*(bN**aN * tau**(aN-1) * np.exp(-bN*tau))
        q_tau = (lambdaN/(2*pi))**(1/2)
        q_tau = q_tau * np.exp(-0.5*np.dot(lambdaN,((mu-muN)**2).T))
        q = q_mu*q_tau
        return(q)


# plot the contours for a distribution
def plotPost(mu, tau, approx, color, i):
        muGrid, tauGrid = np.meshgrid(mu, tau)
        plt.contour(muGrid, tauGrid, approx, colors = color)
        plt.title('Posterior approximation after '+str(i)+' iterations')
        plt.axis([-1.,1.,0,2])  # not always necessary
        plt.xlabel('$\mu$')
        plt.ylabel('tau')
        plt.show()


# Compute the true posterior parameters
def trueParameters(N, D, a0, b0, mu0, lambda0):
        x_mean = (1/N)*sum(D)
        muT = (lambda0*mu0 + N*x_mean)/(lambda0 + N)
        lambdaT = lambda0 + N
        aT = a0 + N/2
        bT = b0 + 0.5*sum((D-x_mean)**2)
        bT = bT + (lambda0*N*(x_mean-mu0)**2)/(2*(lambda0+N))
        return(aT, bT, muT, lambdaT)


# Compute the true posterior distribution
def truePost(aT, bT, muT, lambdaT, mu, tau):
        post = ((bT**aT)*sqrt(lambdaT))/(gamma(aT)*sqrt(2*pi))
        post = post*(tau**(aT-0.5))*np.exp(-bT*tau)
        post = post*np.exp(-0.5*(lambdaT*np.dot(tau,((mu-muT)**2).T)))
        return(post)


# Plot the true posterior distribution
def plotTrue(mu, tau, true):
        muGrid, tauGrid = np.meshgrid(mu, tau)
        plt.contour(muGrid, tauGrid, true)


# VI step, update parameters
def updateParameters(N, D, a0, b0, mu0, lambda0, bN, lambdaN, mu, tau):
        # Some parameter values will be constant
```

```
    x_mean = (1/N)*sum(D)
    muN = (lambda0*mu0 + N*x_mean)/(lambda0 + N)
    aN = a0 + (N+1)/2
    iterations = 6

    # True posterior parameters
    aT, bT, muT, lambdaT = trueParameters(N,D,a0,b0,mu0,lambda0)

    for i in range(iterations):
        E_mu, E_mu2, E_tau = moments(aN, bN, muN, lambdaN)
        lambdaN = (lambda0+N)*E_tau
        bN = b0-E_mu*(lambda0*mu0+sum(D))
        bN = bN+0.5*(E_mu2*(lambda0+N)+lambda0*mu0**2+sum(D**2))

        # Calculate q for new values of the parameters
        approx = q(aN,bN,muN,lambdaN,mu[:,None],tau[:,None])

        # Calculate the true posterior
        post = truePost(aT,bT,muT,lambdaT,mu[:,None],tau[:,None])

        # Plot the posterior approximation and the true
        plotTrue(mu, tau, post)
        color = 'b'
        if i == iterations-1:
            color = 'r'

        plotPost(mu, tau, approx, color, i)



# Generate some data points
N = 10
D = generateData(N)

# Choose parameter values for the conjugate prior
a0 = 3 #0
b0 = 5 #0
mu0 = 0.2 #0
lambda0 = 15 #0

# Start values
bN = 3 #0.1
lambdaN = 15 #0.2

# Generate some mu and tau values for the plot
mu = np.linspace(-2,2,100)
tau = np.linspace(0,4,100)

# Run VI
updateParameters(N, D, a0, b0, mu0, lambda0, bN, lambdaN, mu, tau)
```

Here I will compare the VI approximation of the posterior to the true posterior for 4 different cases. Note that in this case our goal values for μ and τ are 0 and 1 respectively.

The first interesting case is where we set all the parameters of the prior distributions equal to zero, i.e. $a_0 = b_0 = \mu_0 = \lambda_0 = 0$. This implies broad and non-informative priors. Thus we do not include any prior beliefs in the distribution of μ and τ. Also the start values for b_N and λ_N are set very low, 0.1 and 0.2 respectively.

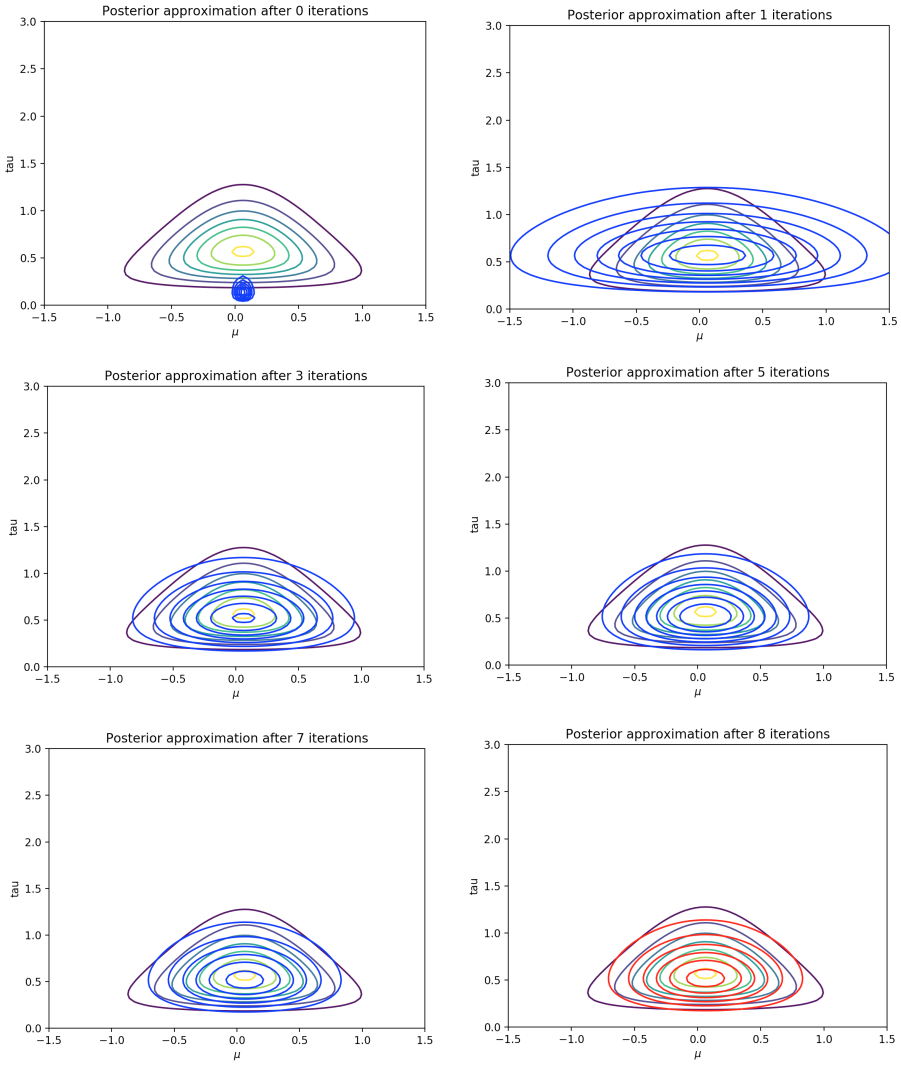

Figure 5: Case where $a_0 = b_0 = \mu_0 = \lambda_0 = 0$, i.e. very broad and non-informative priors. After 8 iterations the VI approximation do not chnage much and is representing the true posterior pretty well. The true posterior is in mixed colors and the approximation is in clear blue. Start values for b_N and λ_N were set to 0.1 and 0.2 respectively. And $N = 10$ data points were observed.

This example tells us that even though we are dealing with priors that does not hold much of information, we are still able to find a good approximation with the VI algorithm. Already after 3 iterations the approximate posterior obtains a good solution, where the approximate mean reaches an even better value than at 8 iterations. But at 8 iterations there is no more clear changes to the approximate posterior and thus we can say that we have reached convergence.

The second case is a more general case, where we set $a_0, b_0, \mu_0, \lambda_0$ to arbitrary values. This gives us a feeling for a general case, in which we might have some prior belief in the distribution of μ and τ.

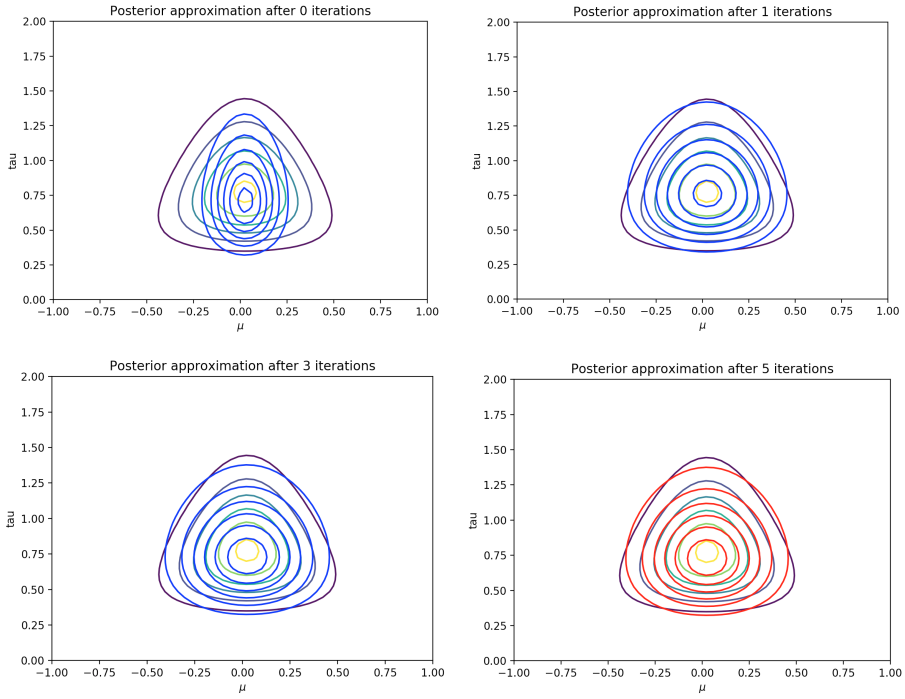

Figure 6: Case where $a_0 = 3, b_0 = 5, \mu_0 = 0.2, \lambda_0 = 15$. After 3 iterations the VI approximation do not change much and is representing the true posterior pretty well. The true posterior is in mixed colors and the approximation is in clear blue. Start values for b_N and λ_N were set to 3 and 15 respectively. And $N = 10$ data points were observed.

In this case we see that a solution is reached much earlier. This is due to that fact that the conjugate priors contains more information now, and in this case the settings seems to be pretty good.

The third and most interesting case is when the start values for b_N and λ_N are set to the true values, i.e. $b_N = b_T$ and $\lambda_N = \lambda_T$, while $a_0 = b_0 = \mu_0 = \lambda_0 = 0$. This means that we are again dealing with broad non-informative priors, but at this time our start guesses for b_N and λ_N are extremely good and thus the priors will not affect the solution as much.

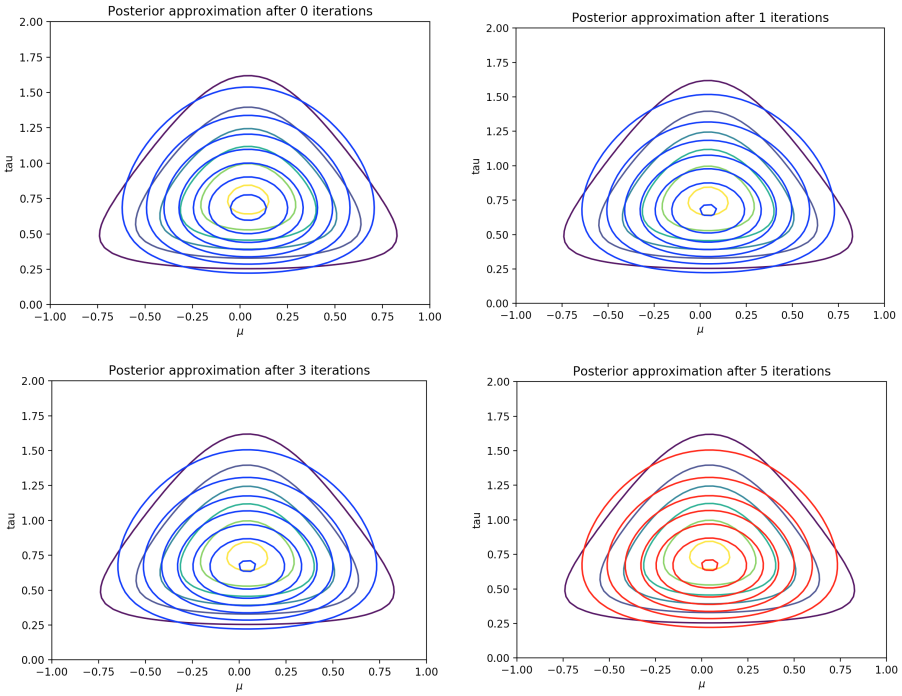

Figure 7: Case where $a_0 = b_0 = \mu_0 = \lambda_0 = 0$, and the start values were set to $b_N = b_T, \lambda_N = \lambda_T$ which in this case where ~ 6.09585 and 10 respectively. Here the VI approximation is extremely good even before any iterations. The true posterior is in mixed colors and the approximation is in clear blue. And $N = 10$ data points were observed.

This tells us that in this case, if we have enough information to make very good start guesses for the parameters b_N and λ_N the priors will have no effect on the result.

Note that in the 3 previous cases I used 10 observations through out. If we now consider the same settings as in case 1, but this time we observe $N = 200$ data points, we should expect a much more sharp solution. Because this time we have much more data and information at hand and thus should be much more confident in the posterior distribution.

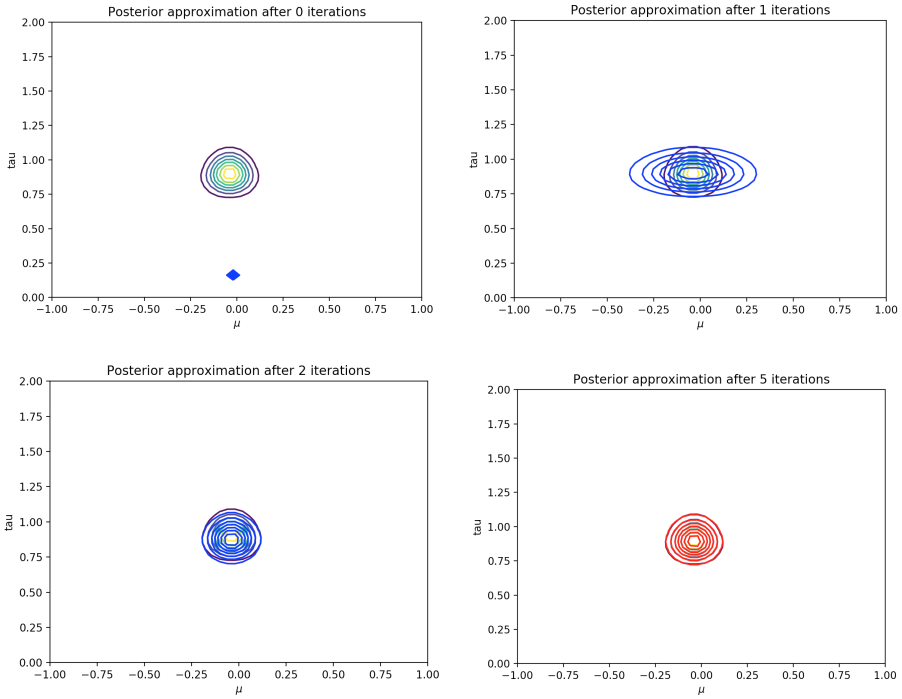

Figure 8: Case where $a_0 = b_0 = \mu_0 = \lambda_0 = 0$. Here we have observed $N = 200$ data points and thus we have much more information which makes the posterior more sharp. The true posterior is in mixed colors and the approximation is in clear blue. Start values for b_N and λ_N were set to 0.1 and 0.2 respectively.

As can be seen from Figure 8 the previous discussion is consistent with the behaviour of the true posterior and its approximation. Here again we are dealing with broad non-informative priors, but since so much data is available we are still able to quickly obtain a good solution.

2.6 Variational Inference

Question 15: *Present the algorithm written down in a formal manner (using both text and mathematical notation, but not pseudo code).*

- R row distributions $\{\mathcal{N}(\mu_r, \lambda_r^{-1}) : 1 \leq r \leq R\}$ and $\mu_r \sim \mathcal{N}(\mu, \lambda^{-1})$
- C column distributions $\{\mathcal{N}(\xi_c, \tau_c^{-1}) : 1 \leq c \leq C\}$ and $\xi_c \sim \mathcal{N}(\xi, \tau^{-1})$
- Known parameters: $\lambda_r^{-1}, \mu, \lambda^{-1}, \tau_c^{-1}, \xi, \tau^{-1}$
- Matrix size: $R \times C$ with elements: $S_{rc} = X_r + Y_c$ where $X_r \sim \mathcal{N}(\mu_r, \lambda_r^{-1})$ and $Y_c \sim \mathcal{N}(\xi_c, \tau_c^{-1})$

We wish to find an expression for the variational distribution,

$$q(\mu_1, ..., \mu_R, \xi_1, ...\xi_C) = \prod_r q(\mu_r) \prod_c q(\xi_c) \tag{3}$$

which approximates $p(\mu_1, ..., \mu_R, \xi_1, ..., \xi_C|S)$.

The sum of two Gaussian distributions is a Gaussian distribution with the means and variances added together, i.e

$$S_{rc} \sim \mathcal{N}(\mu_r + \xi_c, \lambda_r^{-1} + \tau_c^{-1}) = \frac{1}{\sqrt{2\pi(\lambda_r^{-1} + \tau_c^{-1})}} e^{-\frac{(S_{rc}-(\mu_r+\xi_c))^2}{2(\lambda_r^{-1}+\tau_c^{-1})}} \tag{4}$$

The optimum factors $q^*(\mu_r)$ and $q^*(\xi_c)$ can be obtained from the general result [5],

$$lnq_j^*(\mathbf{Z}_j) = E[lnp(\mathbf{X}, \mathbf{Z})] + const \tag{5}$$

where the joint distribution can be written as, $p(\mathbf{X}|\mathbf{Z})p(\mathbf{Z})$. In our case we get, $p(S_{rc}, \mu_r, \xi_c) = p(S_{rc}|\mu_r, \xi_c)p(\mu_r, \xi_c) = p(S_{rc}|\mu_r, \xi_c)p(\mu_r)p(\xi_c)$, since μ_r and ξ_c are independent.

Making use of equation 5 we now begin with finding the optimal expression for $q(\mu_r)$,

$$\begin{aligned}
lnq^*(\mu_r) &= E_{\xi_c}[lnp(S, \mu_r, \xi_c)] + const \\
&= E_{\xi_c}[lnp(S|\mu_r, \xi_c)] + p(\mu_r) + const \\
&= \frac{-1}{2(\lambda_r^{-1} + \tau_c^{-1})} E_{\xi_c}[(S_{rc} - (\mu_r + \xi_c))^2] - \frac{\lambda}{2}(\mu_r - \mu)^2 + const \\
&= \frac{-1}{2(\lambda_r^{-1} + \tau_c^{-1})} (2\mu_r(E_{\xi_c}[\xi_c] - S_{rc}) + \mu_r^2) - \frac{\lambda}{2}(\mu_r - \mu)^2 + const \\
&= -\frac{1}{2}(\lambda + \frac{1}{\lambda_r^{-1} + \tau_c^{-1}})\mu_r^2 + (\lambda\mu - \frac{E_{\xi_c}[\xi_c] - S_{rc}}{\lambda_r^{-1} + \tau_c^{-1}})\mu_r + const
\end{aligned}$$

Comparing this expression with the logarithm of a general Gaussian distribution,

$$ln(\mathcal{N}(x|\mu, \lambda^{-1})) \propto -\frac{1}{2}\lambda x^2 + \lambda\mu x \tag{6}$$

we can conclude that $q(\mu_r)$ takes the form of a Gaussian $\mathcal{N}(\mu_r|\mu_N, \lambda_N^{-1})$. And this also means that $q(\xi_c)$ is a Gaussian distribution $\mathcal{N}(\xi_c|\xi_N, \tau_N^{-1})$, since the $q(\xi_c)$ is obtain in an exact corresponding manner. In these distributions we have,

$$\lambda_N = \lambda + \frac{1}{\lambda_r^{-1} + \tau_c^{-1}} \tag{7}$$

$$\tau_N = \tau + \frac{1}{\lambda_r^{-1} + \tau_c^{-1}} \tag{8}$$

$$\mu_N = E_{\mu_r}[\mu_r] = (\lambda\mu - \frac{E_{\xi_c}[\xi_c] - S_{rc}}{\lambda_r^{-1} + \tau_c^{-1}})\lambda_N^{-1} = \frac{\lambda\mu(\lambda_r^{-1} + \tau_r^{-1}) - \xi_N + S_{rc}}{\lambda(\lambda_r^{-1} + \tau_c^{-1}) + 1} \tag{9}$$

$$\xi_N = E_{\xi_c}[\xi_c] = (\tau\xi - \frac{E_{\mu_r}[\mu_r] - S_{rc}}{\lambda_r^{-1} + \tau_c^{-1}})\tau_N^{-1} = \frac{\tau\xi(\lambda_r^{-1} + \tau_r^{-1}) - \mu_N + S_{rc}}{\tau(\lambda_r^{-1} + \tau_c^{-1}) + 1} \tag{10}$$

In summary, the elements in equation 3 are,

$$q(\mu_r) = \mathcal{N}(\mu_r|\mu_N, \lambda_N^{-1}) \tag{11}$$

$$q(\xi_c) = \mathcal{N}(\xi_c|\xi_N, \tau_N^{-1}) \tag{12}$$

[5] Bishop, Pattern Recognition And Machine Learning, Springer, 2006, Page 466

where the parameters μ_N, λ_N, ξ_N and τ_N are given by equations 7-10. Since the parameters λ_r, μ, λ, τ_c, ξ and τ, are know, the parameters in 7 and 8 are constant. For the last two parameters, equations 9 and 10, optimal values can be find by making an initial guess for, say, ξ_N and use this to compute μ_N and $q(\mu_r)$. Then using this revised value to update ξ_N and $q(\xi_c)$. Then use these to compute μ_N and $q(\mu_r)$ again, and so on. This is done for all $1 \leq r \leq R$ and $1 \leq c \leq C$. These updates are done until, for example, a certain convergence criteria is fulfilled.