Assignment 2, TMA4300 Computer intenstive statistical methods

Kristin Benedicte Bakka and Pål Vegard Johnsen.

August 30, 2018

A MCMC for a toy example

B Ising model

This exercise concerns a 2D rectangular lattice with $m \times n$ nodes, where each node represents a stochastic variable x_{ij} that can take on the values 0 or 1. The probability of each state is

$$f(x) \propto \exp\{-\beta \sum_{(i,j)\sim(k,l)} I(x_{ij} \neq x_{kl})\} \ ,$$

where β is a parameters and the sum is over all pairs of nodes that are direct neighbors. Considering the proportionality constant to be unknown, we want to investigating some properties of the lattice.

These properties are

$$k_a(v) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} \ , \tag{1}$$

the proportion of nodes that have value 1, and

$$k_b(v) = \frac{1}{(m-1)n} \sum_{i=1}^{m-1} \sum_{j=1}^{n} I(x_{ij} = x_{i+1,j}) \ , \tag{2}$$

proportion vertical neighboring nodes that are equal. In R code this can be written

```
pb = sum((u[1:(rows-1),]-u[2:(rows),])==0)/((rows-1)*cols)
```

where u represent the grid and is a matrix with enries equal to zero or one. The mean and variances of k_a will be denoted μ_a and σ_a^2, and for k_b.

B.1

The task is to generate a Markov Chain that can be used to investigate the properties of the Ising model. When the previous state u is known, using the Metropolis-Hastings algorithm to find the next state y amounts to

1. Propose candidate $v \sim Q(v|u)$
2. Compute acceptance probability $\alpha(v|u)$
3. Accept $y = v$, or keep $y = u$ as next state

where $Q(v|u)$ is the proposal distribution. A reasonable candidate v is a lattice such that all nodes remain at the same values as in u, except for node (p_1, p_2), which gets the opposite value, i.e

$$v_{ij} = \begin{cases} 1 - v_{ij} & i = p_1, j = p_2 \\ v_{ij} & \text{otherwise .} \end{cases} \tag{3}$$

One option is to choose node (p_1, p_2) at random, another is to suggest nodes sequentially in a deterministic fashion such that all nodes get selected. Both these choises would make the proposal distribution irreducible and positive recurrent, as it would be possible to go to every state from every state in a finite number of steps. We choose to suggest nodes in the deterministic fashion because it is a waste of CPU time to draw a random number where a deterministic one might suffice, and it might make the convergence a little faster since the time until all nodes are proposed will decrease. In conclusion we have mn proposal kernels $Q_{(p_1,p_2)}$, which each are deterministic.

The formula for the acceptance probability is

$$\alpha = \min(1, \frac{f(v)}{f(u)} \frac{Q_(p_1, p_2)(u|v)}{Q_(p_1, p_2)(v|u)}). \tag{4}$$

State u is state v with the ith node changed, and state v is state u with the same node changed, so $Q_(p_1, p_2)(u|v) = Q_(p_1, p_2)(v|u) = 1$. The probabilities for the states are equal except for the contribution of the differences between node (p_1, p_2) and its neighbors. Therefore

$$\frac{f(v)}{f(u)} = e^{-\beta(d_v - d_u)} ,$$

where d_u is number of dissimilar neighboring nodes for node (p_1, p_2) in state u. If a the node has n neighbors, $d_v = n - d_u$ and the acceptance probability becomes

$$\alpha = \min(1, e^{\beta(2d_u - n)}) .$$

It follows that when the suggested change is to a state that is at least as likely, the candidate is always accepted, and when it is to a less likely state the candidate is sometimes accepted. Table 1 displayes the possible acceptance probabilities different from 1 for different βs, and when they occur. When β is large the acceptance probability gets low, and the convergence will also be slower, and it is possible it won't have time to converge at all.

Table 1: Table of acceptance probabilities for specific data with different βs.

n	d_u	$2d_u - n$	$\beta_1 = 0.5$	$\beta_2 = 0.87$	$\beta_3 = 1$
3	1	-1	0.61	0.42	0.37
4	1	-2	0.37	0.18	0.14
2	0	-2	0.37	0.18	0.14
3	0	-3	0.22	0.07	0.05
4	0	-4	0.14	0.03	0.02

The following function generates the markov chain with these properties, that is acceptance probabilities from equation (4) and deterministic state selection with states defined in equation (3).

```
isingModel <- function(rows=10,cols=6,runs=500,beta=0.5,
                       u=matrix(0, nrow = rows, ncol = cols),colsum=FALSE){
  #Generates a matrix where each column is a state of the Markov Chain
  #If colsum=true only retrun number of ones
  #Decalare Markov Chain vector
  if(!colsum){
    mc=matrix(NA,nrow=rows*cols,ncol = runs+1) #each column is an observation
```

```
  mc[,1]=as.vector(u) #plug in initial state
}else{
  numberOfones=matrix(NA,nrow=1,ncol = runs+1)
  numberOfones[1]=sum(u)
  k2_temp=matrix(NA,nrow=1,ncol = runs+1)
  k2_temp[1]= sum((u[1:(rows-1),]-u[2:(rows),])==0)
}
r=0 #run number
i=0 #index of considered node
case= matrix(9,nrow = rows ,ncol = cols)
#columns
case[,1]=5; case[,cols]=6
#rows
case[1,]=7; case[rows,]=8
#corners
case[1,1]=1; case[rows,1]=2;
case[1,cols]=3; case[rows,cols]=4

while(r < runs){
  # (1) PROPOSE CANDIDATE STATE
  r=r+1
  #next node to consider
  i= ifelse(i==cols*rows, 1, i+1)
  val=u[i]

  # (2) COMPUTE ACCEPTANCE PROBABILITY
  #which type of node is this?
  #compute 2d -n
  temp=switch(case[i],
              -2+2*abs(sum(u[c(i+1,i+rows)]-val)), #(1,1)
              -2+2*abs(sum(u[c(i-1,i+rows)]-val)), #(rows,1)
              -2+2*abs(sum(u[c(i+1,i-rows)]-val)), #(1,cols)
              -2+2*abs(sum(u[c(i-1,i-rows)]-val)), #(rows,cols)
              -3+2*abs(sum(u[c(i-1, i+1, i+rows)]-val)), #(:,1)
              -3+2*abs(sum(u[c(i-1, i+1, i-rows)]-val)), #(:,cols)
              -3+2*abs(sum(u[c(i+1, i-rows, i+rows)]-val)), #(1,:)
              -3+2*abs(sum(u[c(i-1, i-rows, i+rows)]-val)), #(rows,:)
              -4+2*abs(sum(u[c(i-1, i+1, i-rows, i+rows)]-val))) #interior
  #acceptance probability
  #accept = exp(beta*temp)
  accept = beta*temp


  # (3) ACCEPT OR REJECT CANDIDATE STATE
  random = log(runif(1))
  u[i]=ifelse(random<=accept,1-val,val)

  if(!colsum){
    mc[,r+1]=as.vector(u)
  }else{
    numberOfones[r+1]=sum(u)
    k2_temp[r+1]= sum((u[1:(rows-1),]-u[2:(rows),])==0)/((rows-1)*cols)
  }
```

```
  }
  if(!colsum){
    return(mc)
  }else{
    return(list("k1" = numberOfones/(rows*cols), "k2" = k2_temp)/((rows-1)*cols))
  }
}
```

The switch statement accounts for the complication that the nodes on the fringe does not have neighbors on all sides. The case numbering is as demonstrated in table 2, that is case 1 through 4 represent the corners, case 5 and 6 the columns, case 7 and 8 the rows, and case 9 the interior nodes.

Table 2: Case numbers for the different elements in a matrix.

1	7	7	7	3
5	9	9	9	6
5	9	9	9	6
5	9	9	9	6
5	9	9	9	6
2	8	8	8	4

The colsum parameter is included so that optionally only the funcion values and not the entire markov chain is kept. This vastly reduce the amount of memory required, and also makes the computation a tiny bit faster. On a 50×50 grid 100 000 iterations of 4 markov chains took 27.82 seconds to return the whole markov chain and compute the function values afterwards, while it took 27.69 seconds to when colsum was set to true.

When this algorithmn is run it yields a Markov Chain without the burn in period removed. Since the goal is to investigate properties of the limiting distribution, we need to disgard the states that comes before the Markov Chain has converged. To be able to do this we need to be able to recognize when it has converged.

We construct Markov Chains for different inital states, and plot the developement of some functions of the states. The logic is that for the chain to have converged all such functions need to behave in the same way, regardless of what initial state was used.

In the preliminary analysis we will use proportion of ones, that is k_a from equation (1), and later also k_b from equation (2).

1. All neighbors equal
   (a) $x_{ij} = 0 \forall i, j$
   (b) $x_{ij} = 1 \forall i, j$
2. All neighbors unequal
3. $x_{ij} \sim$Bernoulli(p)

To generate thes we employ the R code as follows

```
#all zeroes (1 a)
a0=matrix(0, nrow = rows1, ncol = cols1)

#all ones (1 b)
a1=matrix(1, nrow = rows1, ncol = cols1)

#random (3)
set.seed(42)
```

```
aR=matrix(rbinom(n=rows1*cols1, size=1,prob = 0.5), nrow = rows1, ncol = cols1)
#checkerboard (2)
if(rows1%%2==1)
{#odd number of rows
  aC=matrix(rep_len(c(1,0),length.out=rows1*cols1),nrow=rows1,ncol=cols1)
}else{
  ##even number of rows
  aC=matrix(rep_len(c(1,0),length.out=(rows1+1)*cols1),nrow=rows1+1,ncol=cols1)[1:rows1,1:cols1]
}
```

The computation will be quite heavy, so here is displayed a preliminary test run of 100000 iterations on a 6×8 grid when $\beta = 1$, function (1) evaluated on the states is displayed in figure 1. This plot is not so very informative, but it sort of looks like the markov chains from the different initial conditions behaves the same.

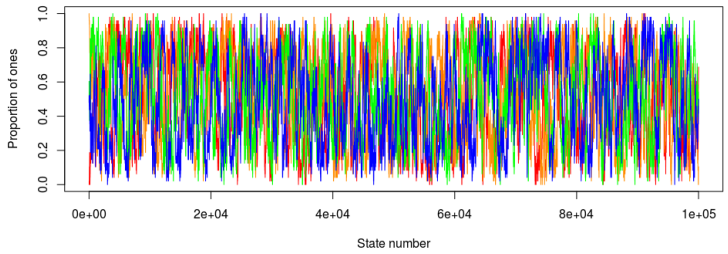


Figure 1: Proportion of ones in Markov Chains run from different initial condintions. Red is 1.(a) all zeros, orange is 1.(b) all ones, green is (2) all neighbors unequal, blue is (3) random. Grid is 6×8. $\beta = 0.87$.

To investigate the burn-in period we now plot the first 1/10th of the iterations and display the result in figure 2. It looks like all the functions have reached the same value for the first time after 2500 iterations. Now it is key to remember that this function is proportion of ones. We would like for every function to have been the lowest at one point and the highest at one point before we can conclude that the Markov Chain has converged. This is fulfilled after 7000 iterations where blue (checkerboard) is the largest. Just to be safe we can remove some more, and 10000 iterations seems to be a nice place from which to investigate the properties of the Markov Chain.

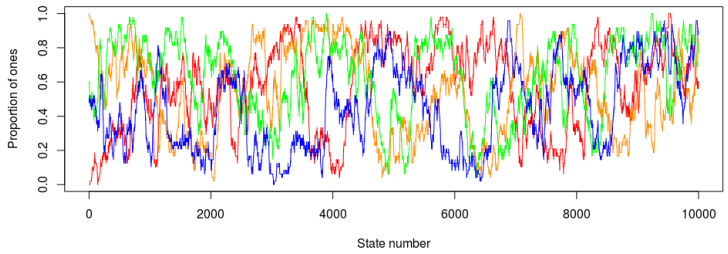


Figure 2: Proportion of ones in first 10 000 runs of Markov Chains from different initial condintions. Red is 1.(a) all zeros, orange is 1.(b) all ones, green is (2) all neighbors unequal, blue is (3) random. Grid is 6×8.

Writing $k_a \sim F(x)$, the approximation we obtained of the cummulative distribution $\hat{F}(x)$ is displayed in figure 3. This was obtained from the function values of the markov chain after the burn-in period was

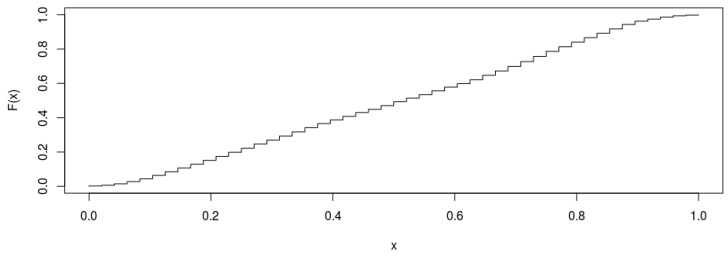


Figure 3: Cumulative distribution function of proportion of ones in Icing model on a 6×8 grid.

removed, and by the following code:

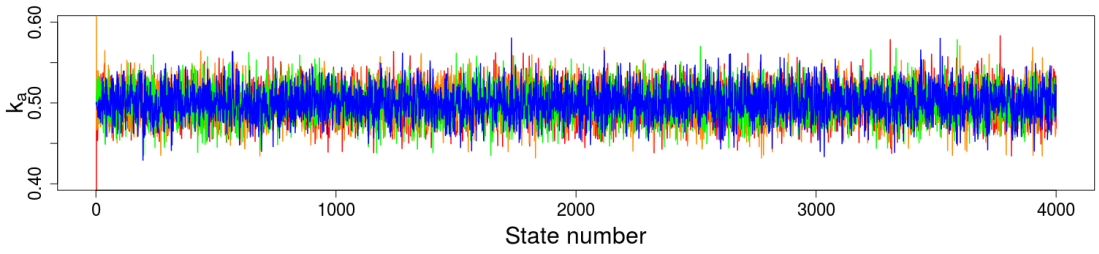
```
x_ka=sort(as.vector(ka))
F=(1:length(x_ka))/length(x_ka)
#plot(x=x_ka,y=F,type='l',ylim=c(0,1),xlab='x',ylab='F(x)')
```

If we are left with too few states after discarding the burn-in period, or if we did not discard enough states, this will not represent the target distribution. $\hat{F}(x)$ is not completely smooth, but looks like a step funcion. This might be because the probability space is discrete (each possible state v has a given probability, and maps to a specific state) or because there has not been produced a sufficient number of states, such that each possible probability is not represented. Since we evaluated the function for each choise of state, all states are represented, and the step form of F(x) is due to the first option. Saving the state will take both CPU time and memory, and to be able to gain as much information as possible with the available resources we modify the code to only save the state after it has gone through the lattice one time. Since saving a state represents such a high cost compared to iterating through the markov chain, we will want to change states many times between each save, such that the states we save are less correlated. This is the reason for increase of information. This is accoplished with the code below. We can also try to go through the grid multiple times before saving. Due to this change not all states are neccesarily represented in the markov chain, and it will be interesting to see what the cumulative function will look like.

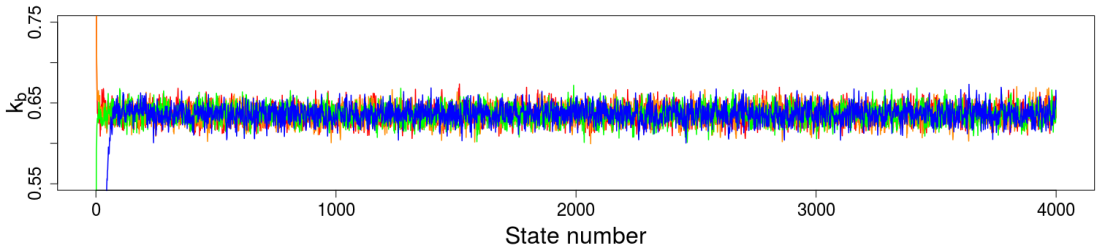
```
isingModel <- function(rows=50,cols=50,runsG=400000,beta=0.87,
                       u=matrix(0, nrow = rows, ncol = cols),colsum=TRUE){
  #Generates a matrix where each column is a state of the Markov Chain
  #If colsum=true only return function values
  #Decalare Markov Chain vector
  if(!colsum){
    mc=matrix(NA,nrow=rows*cols,ncol = runsG+1) #each column is an observation
    mc[,1]=as.vector(u) #plug in initial state
  }else{
    k1_temp=matrix(NA,nrow=1,ncol = runsG+1)
    k1_temp[1]=sum(u)
    k2_temp=matrix(NA,nrow=1,ncol = runsG+1)
    k2_temp[1]= sum((u[1:(rows-1),]-u[2:(rows),])==0)
  }
  case= matrix(9,nrow = rows ,ncol = cols)
  #columns
  case[,1]=5; case[,cols]=6
  #rows
  case[1,]=7; case[rows,]=8
  #corners
  case[1,1]=1; case[rows,1]=2;
```

```
  case[1,cols]=3; case[rows,cols]=4

  for(r in 1:runsG){
    for (k in 1:10){
      for(i in 1:(cols*rows)){ #fill grid 1 time
        # (1) PROPOSE CANDIDATE STATE
        #next node to consider
        val=u[i]

        # (2) COMPUTE ACCEPTANCE PROBABILITY
        #which type of node is this?
        #compute 2d -n
       temp=switch(case[i],
                   -2+2*abs(sum(u[c(i+1,i+rows)]-val)), #(1,1)
                   -2+2*abs(sum(u[c(i-1,i+rows)]-val)), #(rows,1)
                   -2+2*abs(sum(u[c(i+1,i-rows)]-val)), #(1,cols)
                   -2+2*abs(sum(u[c(i-1,i-rows)]-val)), #(rows,cols)
                   -3+2*abs(sum(u[c(i-1, i+1, i+rows)]-val)), #(:,1)
                   -3+2*abs(sum(u[c(i-1, i+1, i-rows)]-val)), #(:,cols)
                   -3+2*abs(sum(u[c(i+1, i-rows, i+rows)]-val)), #(1,:)
                   -3+2*abs(sum(u[c(i-1, i-rows, i+rows)]-val)), #(rows,:)
                   -4+2*abs(sum(u[c(i-1, i+1, i-rows, i+rows)]-val))) #interior
        #acceptance probability
        #accept = exp(beta*temp)
        accept = beta*temp

        # (3) ACCEPT OR REJECT CANDIDATE STATE
        random = log(runif(1))
        u[i]=ifelse(random<=accept,1-val,val)
      }

      if(!colsum){
        mc[,r+1]=as.vector(u)
      }else{
        k1_temp[r+1]=sum(u)
        k2_temp[r+1]= sum((u[1:(rows-1),]-u[2:(rows),])==0)
      }
    }
  }
  if(!colsum){
    return(mc)
  }else{
    return(list("k1" = k1_temp/(rows*cols), "k2" = k2_temp/((rows-1)*cols)))
  }
}
```

Now we are ready to run the markov chains for the large grids. We will start with $\beta_1 = 0.5$, then we will look at $\beta_2 = 0.87$, and last we will look at $\beta_3 = 1$. Figure 4a displays k_a evaluated at 10 000 000 iterations of the Markov Chain started from the different initial states, and figure 4b is corresponding for k_b. 10 000 000 iterations corresponds to setting runsG=4000. In the figures only the blue line, which was plotted last, is easily discernible after approximately state number 50. The behavior also seems to be the same for most of the run time, and so we conclude that the markov chain does in fact converge.



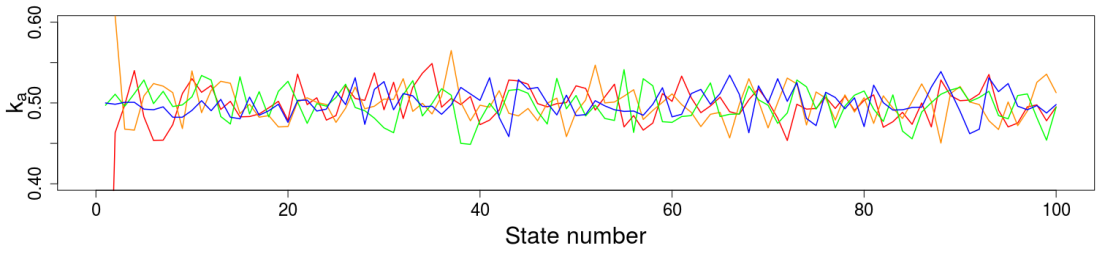
(a)



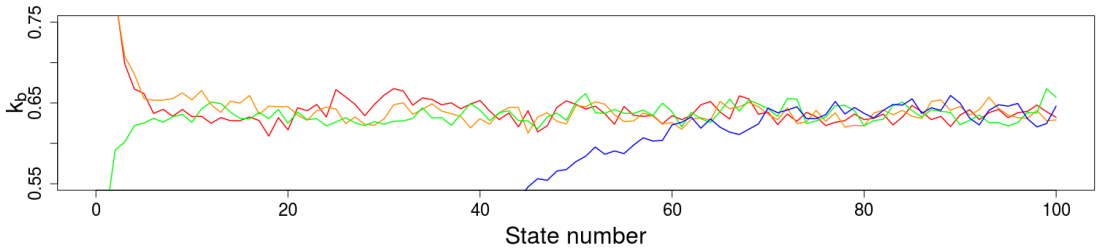
(b)

Figure 4: in Markov Chains run from different initial condintions. Red is 1.(a) all zeros, orange is 1.(b) all ones, green is (2) all neighbors unequal, blue is (3) random. Grid is 6×8. $\beta_1 = 0.5$.

To investigae the burn in period we plot only the first 100 states in figures 5a and 5b. From k_a it seems like the Markov Chain has converged after 40 states, since the behaviors seems to be the same, like previously discussed. However the blue line in figure 5b does not behave the same as the others before state number 80. This means that the markov chain has not converged before the 80th state. It might still be the case that the markov chain has not converged yet. To be safe we discard the first 180 states of each of the chains, and use the remainder of the chains to study the interesting entities, like we did in the preliminary analysis. If we were worried that this would not be enough to ascertain convergence, we could plot the chains of more initial states, or we could plot other functions of the state. One alternative when the burn-in period is determined is to run one markov chain for a long time, but we have already opted to run all the markov chains for semi-long time.



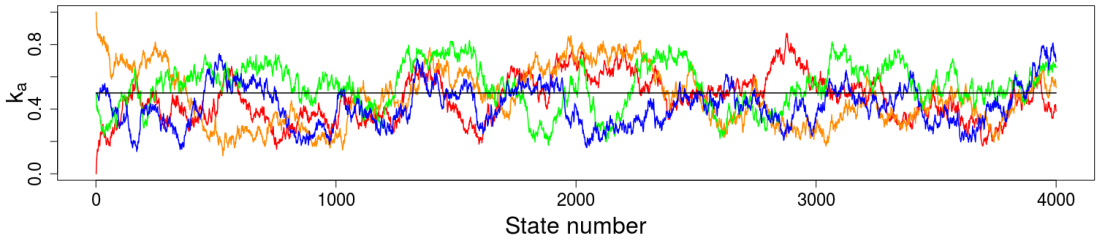
(a)



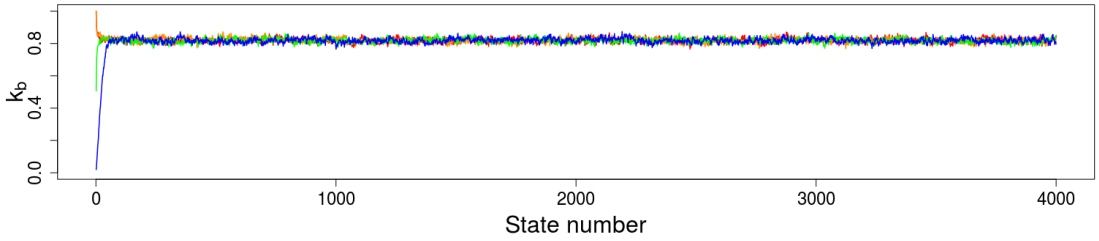
(b)

Figure 5: in Markov Chains run from different initial condintions. Red is 1.(a) all zeros, orange is 1.(b) all ones, green is (2) all neighbors unequal, blue is (3) random. Grid is 6×8. $\beta_1 = 0.5$.

Now we turn to the analysis for $\beta_2 = 0.87$. First the plots of the entire Markov Chains in figures 6a and 6b, must be considered. The plots for k_b are just as the plots for the first β_1, and is thus interpreted in the same way. They quickly localize to a small neighborhood of the mean value. The plots for k_a varies across evaluations between 0.15 and 0.85, and it is not as easy to determine when the behaviors of the functions are equal. It is of no consequece that the function values do not localize, this just implies that the distribution of k_a is not as localised, but has higher variance.



(a)



(b)

Figure 6: in Markov Chains run from different initial condintions. Red is 1.(a) all zeros, orange is 1.(b) all ones, green is (2) all neighbors unequal, blue is (3) random. Grid is 6×8. $\beta_1 = 0.5$.

Since the behavior of k_b implies the Markov Chain has not converged before approximately state number 100, which is comparatively low number, it will only be neccessary to analyze when the behavior of k_a permits the Markov Chain to have converged. Since there is nothing that distinguish the values 0 and 1, we know k_a, proportion of ones, will have a distribution symmetric about $\frac{1}{2}$, and to simplify the analysis we have indicated this mean in figure **??**.

Before state 250 the Markov Chain starting with all ones still dominates in size of k_a, and it is clear that the Markov Chain has not converged. At state 750 each of the different functions have been highest and lowest at some point. Being not so cautious we could choose to drop only the first 1000 evaluations. Being a bit more cautious it would worry us that the red curve, that represents all zeros as initial conditions, remain so low for such a long time. When the evaluation of k_a first reached the mean the state might be one with few probable adjacent states, such that some unprobable states are over represented. It would be difficult for ones to attain the upper hand. We want the state of the grid to act as if it is independent of the initial state, and we can not be assured this is the case when the red line behaves as it would if it was highly dependent on the initial state. This line represent starting at one of the two most likely boards. Apart from this there is nothing spectacular about this Markov Chain, and so when we won't conclude that the other Markov Chains have converged until all this has.

From state 1000 to 1500 the function evaluations are quite clustered, and the red curve has been highest once more. To be safe we could discard the 2000 first states. We might not have chosen to do this if this left us with too few states, but it is easy for us to simply elongate one of the Markov Chains by taking the last state of one of the converged Markov Chains and generate a longer Markov Chain. As the computational cost is not too high we choose to do this, and the rest of the chain is displayed in figure 7a and 7b.

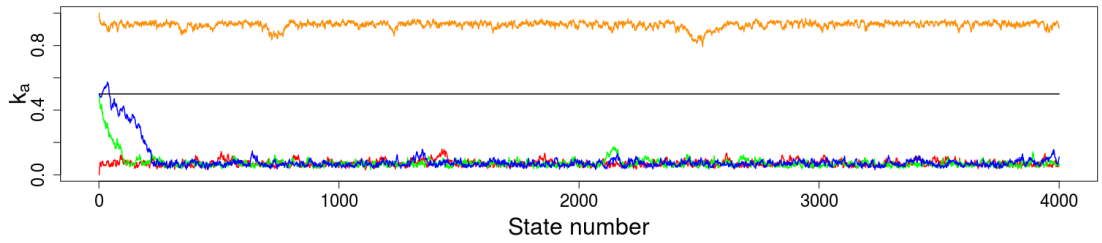
(a)

(b)

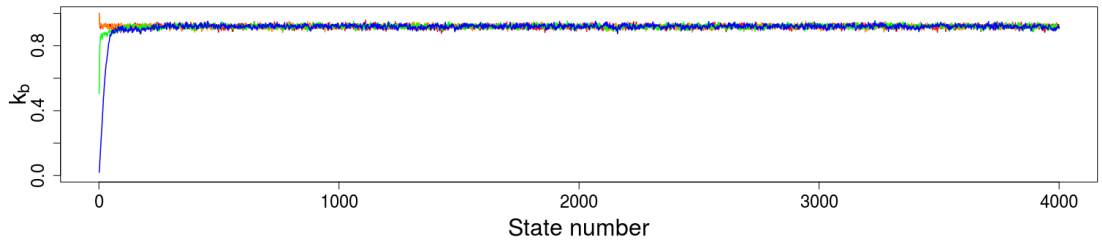
Figure 7: k_1 and k_2 evaluated at states in Markov Chains run from different initial condintions. Red is 1.(a) all zeros, orange is 1.(b) all ones, green is (2) all neighbors unequal, blue is (3) random. Grid is 6×8. $\beta_1 = 0.5$.

Comparing β_1 and β_2 the period we choose to discard is dramatically longer, and we might wonder whether we were right to evaluate the burn-in period to be so low as we did with β_1. It is possible that the effect of the reduced acceptance probability is indeed this high, also taking into account that the distribution of k_a is not as locallized. It is also possible that the Markov Chain has not had time to converge during the 4000 first states, in that case the behavior should change when we elongate the Markov Chain.

Then we evaluate for the last $\beta_3 = 1.0$. Advancing in the same way as before first we plot k_1 and k_2 evaluated at the whole Markov Chain in figures 8a and 8b. This is an example of why it is important to plot multiple functions of the Markov Chain. In the plot for k_b there is nothing that implies that the Markov Chain has not converged after 500 states, but the plot of k_a shows that the chain has not converged during the first 4000 states. This is reasonable since the acceptance probabilities for β_3 is so low. There is no reason to expect the convergence to happen in a feasible number of steps, and so we abandon the plan to compute k_a and k_b in this case.



(a)



(b)

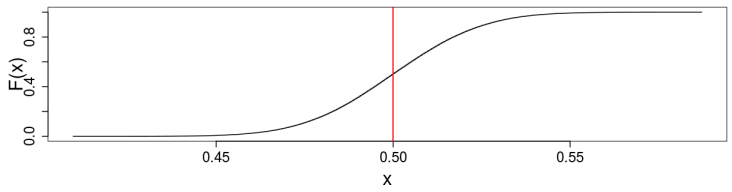
Figure 8: k_1 and k_2 evaluated at states in Markov Chains run from different initial condintions. Red is 1.(a) all zeros, orange is 1.(b) all ones, green is (2) all neighbors unequal, blue is (3) random. Grid is 6×8. $\beta_1 = 0.5$.

B.2 Interesting properties

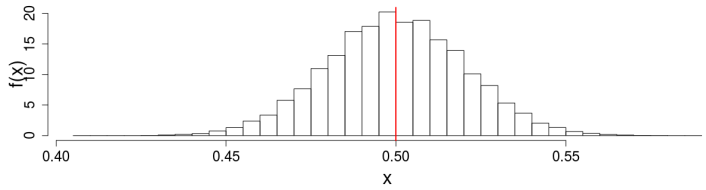
B.2 (a)

In this section the actual interesting properties are discussed. The first property is k_a which is displayed in equation (1), that is proportion of ones in the grid. Estimated cumulative plot is obtained in the same way as in the preliminary analysis, and is displayed in figure 9a. In addition we have estimated the probability density function with a histogram in figure 9b, which was obtained the same way as in the last project, namely with the following code:

```
#hist(xvals_k1, freq = FALSE,breaks = 100)
```



(a)

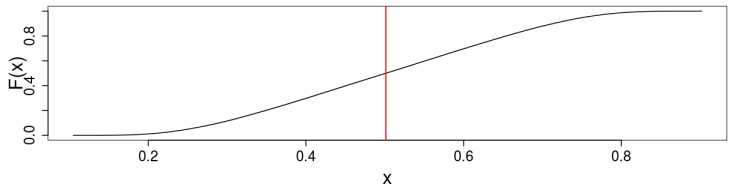


(b)

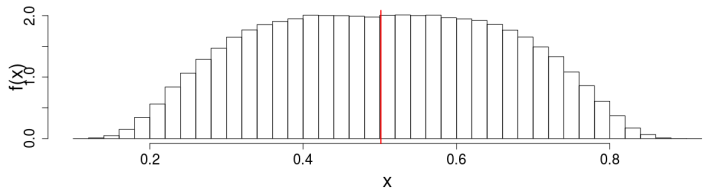
Figure 9: CDF and PDF of k_a with $\beta_1 = 0.5$.

Both of the plots in 9a and 9b look pretty nice in that they are approximately symmatric about the mean as we would assume. The distribution of k_1 seems to be unimodal. A specific state with overweight of ones or zeros is more likely to be a probable state as it is qualified to have more equal neigbors. On the other hand there are more states that evaluate around 0.5 than evaluate close to 1 or 0. A low β means a small penalty for neighbors to be different, and for the histogram to look like a normal distribution is not so strange. A tendency towards two modes close to each other would not be strange either.

The plots in figure 10a and 10b are not as easily interpreted. The cumulative distribution looks to have bulks, and the pdf is not symmetric at all. Have I plotted for enough states, and did I assess the length of the burn-in period correctly? To look at this we run the markov chain some more.



(a)



(b)

Figure 10: CDF and PDF of k_a with $\beta_2 = 0.87$.

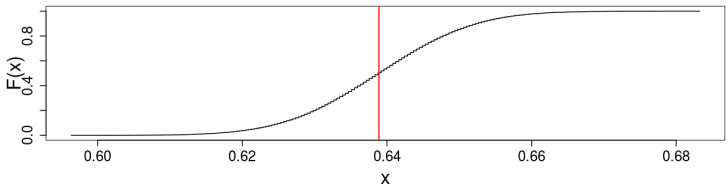
For β_1 and β_2 the estimates of the mean values $\hat{\mu}_{a,1}$ and $\hat{\mu}_{a,2}$ are displayed in table 3.

B.2 (b)

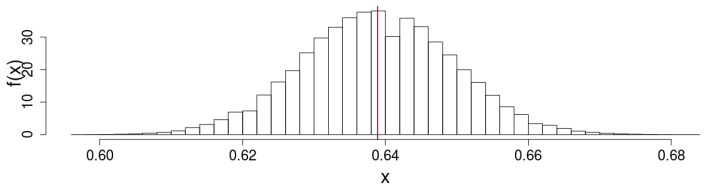
Estimated cumulative plot and pdfs for k_b are obtained in the same way as in B.2(a), and are displayed in figures 12a and 12b with $\beta_1 = 0.5$ and figures 12a and 12b with $\beta_2 = 0.87$. The plots show that even with this number of data some values are under represented. This is probably not a description of the true pdf, but rather implies that we could have run the markov chain even longer.

Proportion of vertical neighbors that are equal is not as easily imagined as proportion of ones in the matrix. It might be viewed as a simplification of the probability for a state. With more equal vertical neighbors the state is likely to have a higher probability, since it is likely to have less different neighbors in total.

States where either zeros or ones dominate the grid strongly will be expected to have a high number of equal vertical neigbors, but there are not many such states, so we would expect the pdf to tail off and not cut off. Likewise we imagine there are not so many states where there are almost no equal vertical neighbors. There is no apparent reason that the distribution should be symmetric in any way. We don't have a specific expectation for the behavior between the tails either. This is in fact why this Markov Chain is interesting to study.

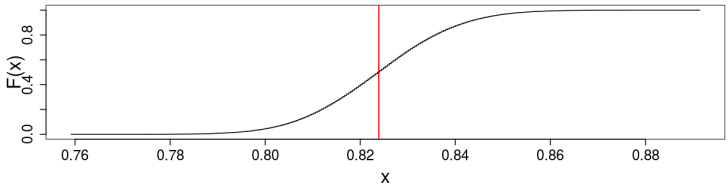


(a)

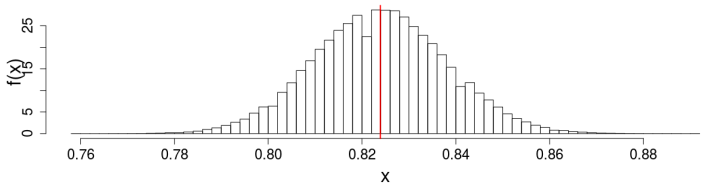


(b)

Figure 11: CDF and PDF of k_b with $\beta_1 = 0.5$.



(a)



(b)

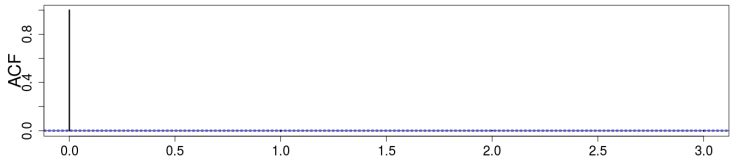
Figure 12: CDF and PDF of k_b with $\beta_2 = 0.87$.

For β_1 and β_2 the estimates of the mean values $\hat{\mu}_{b,1}$ and $\hat{\mu}_{b,2}$, are presented in table 3.

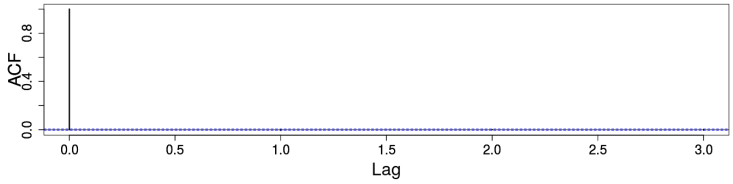
B.3

The central limits theorem states that a sum of n independently distributed variables x_i is approximately normally distributed when n is large, and so we want to use $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} k(u_i) \sim N(\hat{\mu}, \hat{\sigma})$, where n is number of Markov Chain states u. The problem is that the values $k(u_i)$ not neccessarily are independent. To assess this we first compute the autocorrelations (acfs) of the series using the R function acf from the stats library. The acfs of $k_a(u_i)$ and $k_b(u_i)$ when $\beta_1 = 0.5$ is used are displayed in figures 13a and 13b. They are uncorrelated, and we can proceed to compute the variance $\hat{\sigma^2} = \frac{1}{n-1}\sum_{i=1}^{n}(k(u_i) - \hat{\mu})^2$. Using that

$Pr(|X| \leq 1.96) = 0.95$ for a standard normal distribution, then $Pr(\hat{\mu} + 1.96\hat{\sigma} \leq \mu \leq \hat{\mu} + 1.96\hat{\sigma}) = 0.95$. These entities are displayed in table 3. The confidence intervals of μ_a and μ_b are displayed in the fourth and fifth columns of the table.

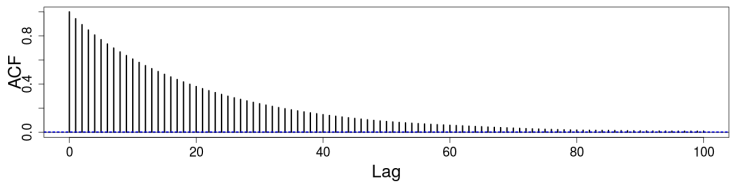


(a) Autocorrelation function for k_a with $\beta_1 = 0.5$.

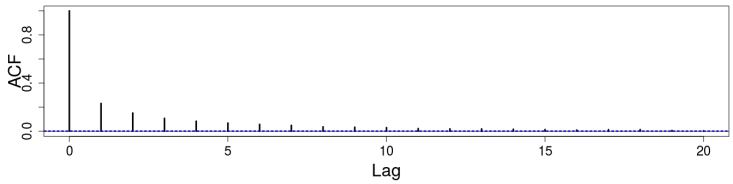


(b) Autocorrelation function for k_b with $\beta_1 = 0.5$.

Figure 13



(a) Autocorrelation function for k_a with $\beta_2 = 0.87$.



(b) Autocorrelation function for k_b with $\beta_2 = 0.87$.

Figure 14

Next we turn to the autocorrelations of $k_a(u_i)$ and $k_b(u_i)$ when $\beta_2 = 0.87$, which are displayed in figures 14a and 14b. Both of these have significant acfs, and we must use some other method. One option is to include for instance only every twentieth value in the plot. Since this would correspond to running the markov chain for a long time between each sample, which would make the samples less dependent. What we have chosen to do is divide our values into boxes, where we take the mean v_i inside box i, and then plot the autocorrelation of these means, like in the following R code.

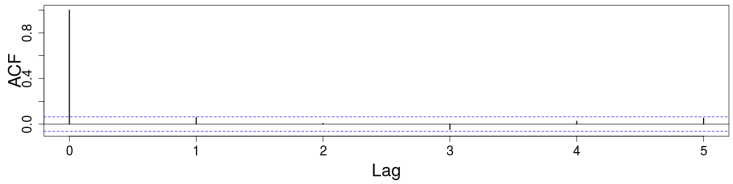
```
va_beta1=colSums(matrix(ka_beta2,nrow = 264))
```

To decide on a size s for each box, we plot the acf different values of s until we find a value that does not

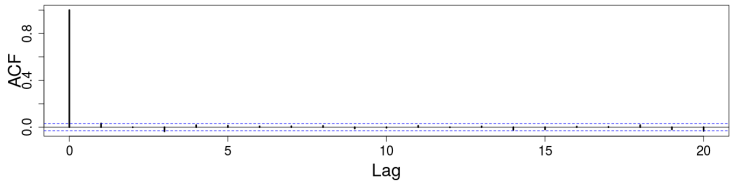
Table 3: The calculated values of the different parameters, along with the number of states the calculations are based on, and the box size.

	$\hat{\mu}$	$\hat{\sigma}^2$	$\hat{\mu} - 1.96\hat{\sigma}$	$\hat{\mu} + 1.96\hat{\sigma}$	States	box size
k_a, β_1	0.4999	$4.11 \cdot 10^{-4}$	0.4602	0.5397	121510	1
k_a, β_2	0.5014	0.0245	0.194	0.6596	250800	264
k_b, β_1	0.6389	$1.11 \cdot 10^{-4}$	0.6182	0.6595	121510	1
k_b, β_2	0.8239	$2.0 \cdot 10^{-4}$	0.7962	0.8516	250800	264

show correlation. For k_a and k_b we have chosen respectively 264 and 60, and the corresponding acfs are displayed in figures 15a and 15b. Then the variances can be computed with $\hat{\sigma^2} = \frac{1}{s-1}\sum_{i=1}^{s}(v_i - \hat{\mu})^2$, and the confidence intervals found as above. These too are displayed in table 3



(a) Autocorrelation function for $\mathbf{v}_a$ with $\beta_2 = 0.87$.



(b) Autocorrelation function for $\mathbf{v}_b$ with $\beta_2 = 0.87$.

Figure 15

C Microarray data

Given the independent data $x_{g1}, ..., x_{gS_1}, y_{g1}, ..., y_{gS_2}$, and the following relations for gene g:

$$x_{gi}|\nu_g, \Delta_g, \tau_g \sim N(\nu_g + \Delta_g, \frac{1}{\tau_g})$$

$$y_{gi}|\nu_g, \Delta_g, \tau_g \sim N(\nu_g - \Delta_g, \frac{1}{\tau_g})$$

Then, the joint distribution, $f(x_{g1}, ..., x_{gS_1}, y_{g1}, ..., y_{gS_2}|\nu_g, \Delta_g, \tau_g)$, is given by:

$$f(x_{g1}, ..., x_{gS_1}, y_{g1}, ..., y_{gS_2}|\nu_g, \Delta_g, \tau_g) = \frac{\tau_g}{2\pi}^{\frac{S_1+S_2}{2}} \prod_{i=1}^{S_1} exp(-\frac{\tau_g}{2}(x_{gi}-\nu_g-\Delta_g)^2) \prod_{i=1}^{S_2} exp(-\frac{\tau_g}{2}(y_{gi}-\nu_g+\Delta_g)^2)$$

C.1 Posterior distribution of ν_g

Assuming ν_g has a normal prior distribution, say, $\nu_g \sim N(\mu_\nu, \sigma_\nu^2)$, the posterior distribution of ν_g is given by:

$$\text{f}(\nu_g|x_{g1},...,x_{gS_1},y_{g1},....,y_{gS_2},\Delta_g,\tau_g) \propto f(x_{g1},...,x_{gS_1},y_{g1},....,y_{gS_2}|\nu_g,\Delta_g,\tau_g)f(\nu_g)$$

$$\propto \exp\left(-(\tfrac{\tau_g}{2}(S_1+S_2)+\tfrac{1}{2\sigma_\nu^2})\Big(\nu_g - \frac{\tau_g\Delta_g(S_2-S_1)+\tau_g(\sum_{i=1}^{S_1}x_{gi}+\sum_{i=1}^{S_2}y_{gi})+\frac{\mu_\nu}{\sigma_\nu^2}}{\tau_g(S_1+S_2)+\frac{1}{\sigma_\nu^2}}\Big)^2\right) \propto \exp\left(-(\tfrac{1}{2\sigma^2})(\nu_g-\mu)^2\right)$$

, which shows that the posterior distribution is also normal distributed since the distribution is proportional to $\exp(g(\nu_g|\mu,\sigma))$ where $g(\nu_g|\mu,\sigma)$ is a concave quadratic function of ν_g. Furthermore the mean of the posterior distribution is given by μ and the variance given by σ^2. Therefore the normal distribution is the (conditional) conjugate distribution for ν_g.

C.2 Posterior distribution of Δ_g and τ_g

Assuming Δ_g has a normal prior distribution, $\Delta_g \sim N(\mu_\Delta, \sigma_\Delta^2)$, one can prove in exactly the same way as above that the posterior distribution is given by:

$$\text{f}(\Delta_g|x_{g1},...,x_{gS_1},y_{g1},....,y_{gS_2},\nu_g,\tau_g) \propto f(x_{g1},...,x_{gS_1},y_{g1},....,y_{gS_2}|\nu_g,\tau_g)f(\Delta_g)$$

$$\propto \exp\left(-(\tfrac{\tau_g}{2}(S_1+S_2)+\tfrac{1}{2\sigma_\Delta^2})\Big(\Delta_g - \frac{\tau_g\nu_g(S_2-S_1)+\tau_g(\sum_{i=1}^{S_1}x_{gi}-\sum_{i=1}^{S_2}y_{gi})+\frac{\mu_\Delta}{\sigma_\Delta^2}}{\tau_g(S_1+S_2)+\frac{1}{\sigma_\Delta^2}}\Big)^2\right) \propto \exp\left(-(\tfrac{1}{2\tilde{\sigma^2}})(\Delta_g-\tilde{\mu})^2\right)$$

, which shows that the posterior distribution of Δ_g is also normal distributed with mean corresponding to $\tilde{\mu}$ and variance corresponding to $\tilde{\sigma^2}$ and therefore the normal distribution is the (conditional) conjugate distribution for Δ_g.

Assuming τ_g has a gamma prior distribution with shape k and scale θ, the posterior distribution is given by:

$$f(\tau_g|x_{g1},...,x_{gS_1},y_{g1},....,y_{gS_2},\nu_g,\Delta_g) \propto f(x_{g1},...,x_{gS_1},y_{g1},....,y_{gS_2}|\nu_g,\tau_g)f(\tau_g)$$

$$\propto \tau_g^{(k+\frac{S_1+S_2}{2})-1}\exp\left(-\tau_g\Big(\tfrac{1}{2}\textstyle\sum_{i=1}^{S_1}(x_{gi}-\nu_g-\Delta_g)^2+\tfrac{1}{2}\sum_{i=1}^{S_2}(y_{gi}-\nu_g-\Delta_g)^2+\tfrac{1}{\theta}\Big)\right) \propto \tau_g^{\tilde{k}-1}\exp\left(\frac{-\tau_g}{\tilde{\theta}}\right)$$

, which shows that the posterior distribution of τ_g is also gamma distributed with shape corresponding to $\tilde{k}$ and scale corresponding to $\tilde{\theta}$ and therefore the gamma distribution is the (conditional) conjugate distribution for τ_g.

C.3 Gibbs sampling

Looking at the posterior distribution one sees that all the parameters ν_g, Δ_g and τ_g are dependent of each others. One may visualise the hierarchical bayesian model in terms of a graphical model of the stochastic parameters, with nodes corresponding to stochastic parameters and arrows from node to node visualizing the dependency between the stochastic parameters or in some sort how the information flows from the hyperparameters to the parameters:

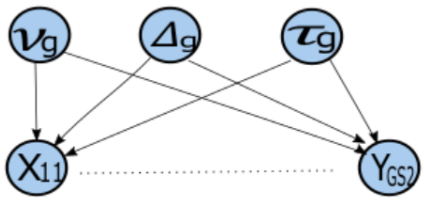


Figure 16: Graphical model of the Bayesian Hierarchy in task C.

As the posterior distributions for each unknown parameter is found, and since these distributions are easy to sample from, one may use the Gibbs algorithm. After many iterations, the samples produced by the Gibbs algorithm will, under mild conditions, converge to the joint distribution $f(\nu_g, \Delta_g, \tau_g)$. In fact the sample corresponding to a given parameter will convege to the marginal distribution ($f(\nu_g)$, etc.). The algorithm for this spesific problem is given below:

```
microarrayanalysis = function(nu_mean, nu_variance, delta_mean, delta_variance,
tau_shape, tau_scale, genedatax, genedatay, N){
# Gibbs sample function to make ONE markov chain for each parameter of the microarray-data.
# nu_mean, nu_variance, delta_mean, delta_variance, tau_shape, tau_scale start values for
 # the parameters of the distribution of mu (normal), delta(normal) and tau(gamma)
# genedatax and genedatay are data for the gene for the two different groups x and y
# N number of samples for each parameter wanted

s_1 = length(genedatax) # s_1 number of data from gene for group x
s_2 = length(genedatay) #s_2 number of data from gene for grup y

nu_samples = vector(mode = "numeric", length = N) # samples of nu
delta_samples = vector(mode = "numeric", length = N) # samples of delta
tau_samples = vector(mode = "numeric", length = N) # samples of tau

# Generate random start values for nu, delta and tau
nu_samples[1] = rnorm(1,nu_mean,sqrt(nu_variance))
delta_samples[1] = rnorm(1,delta_mean,sqrt(delta_variance))
tau_samples[1] = rgamma(1, shape = tau_shape, scale = tau_scale)
# Generate Gibbs samples:
for(i in 2:N){

# sample for nu:
nu_samples[i] = rnorm(1, (tau_samples[i-1]*delta_samples[i-1]*(s_2-s_1)
+ tau_samples[i-1]*(sum(genedatax)+sum(genedatay)) + nu_mean/nu_variance)/
(tau_samples[i-1]*s_1 + tau_samples[i-1]*s_2 + 1/nu_variance ),
sqrt(1/(tau_samples[i-1]*s_1 + tau_samples[i-1]*s_2 + 1/nu_variance )))

# sample for delta
delta_samples[i] = rnorm(1,(tau_samples[i-1]*nu_samples[i]*(s_2-s_1)
```

```
+ tau_samples[i-1]*(sum(genedatax)-sum(genedatay)) + delta_mean/delta_variance)/
(tau_samples[i-1]*s_1 + tau_samples[i-1]*s_2 + 1/delta_variance )
,  sqrt(1/(tau_samples[i-1]*s_1 + tau_samples[i-1]*s_2 + 1/delta_variance )))
#sample for tau
tau_samples[i] = rgamma(1, shape = (tau_shape + (s_1+s_2)/2),
scale = 1/(0.5*(sum((genedatax - nu_samples[i] - delta_samples[i])^2) +
sum((genedatay - nu_samples[i] + delta_samples[i])^2)) + 1/tau_scale))


}
# Create list containing sample from each parameter
l = list(nu_samples,delta_samples,tau_samples)
return(l)
}
```

The result when investigating gene 1,2 and 3 by making 10 000 samples from each parameter is shown in figure 17. As seen from the figures, the samples converge fast to the conditional distributions. One can now do inference analysis for each of the three parameters. One must be careful though, making sure that the samples have converged. This can be handled by deleting the samples from the first iterations were the samples seem to not converge to the wanted distributions. There are conventions on how to find out how many samples to delete from the analysis. In our analysis, it is clearly OK to drop the first, say, 1000 samples. From here on, all three figures shows that the samples fluxtuates around a spesific area.

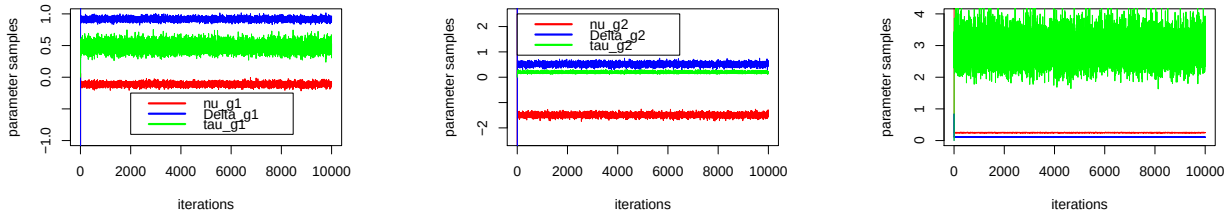


Figure 17: Generated Gibbs samples of the parameters when investigating gene number 1,2 and 3

One common way to estimate the parameters is to compute the empirical mean of the parameter which is unbiased and converges in probability to the true mean of the parameter. Based on the square loss function, this is the best we can do. Three estimates of the parameters ν_g, Δ_g and τ_g based on for instance gene 1 are then $\tilde{\nu_1} = -0.11$, $\tilde{\Delta_1} = 0.92$ and $\tilde{\tau_1} = 0.48$. Notice that these estimates fit well with the mean of the datasets of gene 1 for group x and y.

In order to decide whether any of these three genes are differentially expressed, one may look at the marginal distribution of Δ_g for each gene (1,2 and 3). As described earlier, the samples generated for each parameter will converge to the marginal distribution. The approximated marginal distributions for Δ_i for i = 1,2,3 are given below including the estimated Δ_i for each gene i and a mark of where 0 is. In this case 100 000 iterations are done just to make sure that the distributions look nice, and one may recognize if they look like a known distribution.

From the figures above one may conclude that gene 1 is differential expressed, while for gene 2 and gene 3 the computed differences are to small in order to say that these are differential expressed.

One may also recognize that the density distributions for Δ_i look very like the Gaussian distribution for all

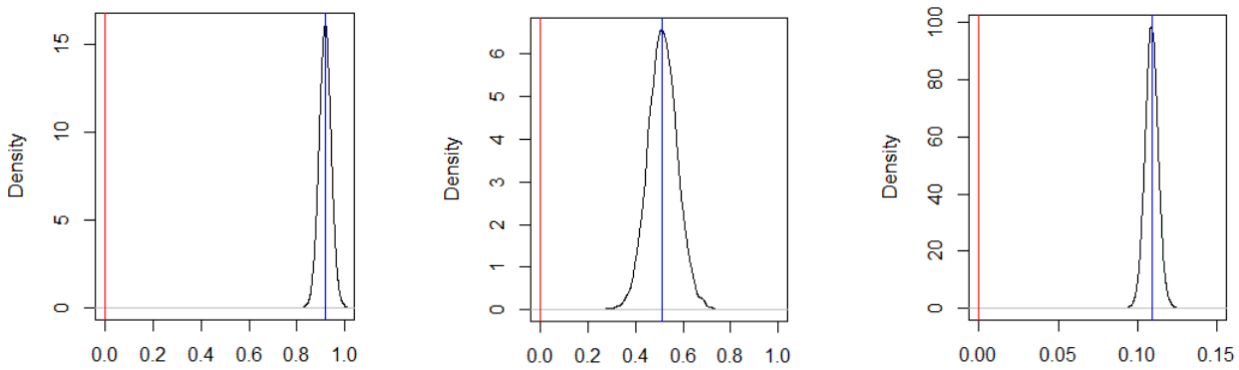


Figure 18: Estimated density plots for Δ_1, Δ_2 and Δ_3 from left to right. Red vertical line represents where zero is located, and blue vertical line represents estimation of Delta. The built-in R-function density() where used in order to estimate the densities.

three genes.

D More on microarray data

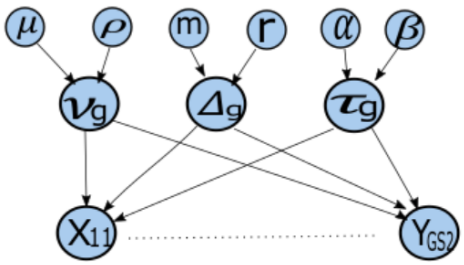


Figure 19: Graphical model of the Bayesian Hierarchy in task D

D.1

Now, consider the case where we dont know the hyperparameters of ν_g, Δ_g and τ_g in task C for g = 1,..,G. With the assumptions of the distribution of the hyperpriors given in task D, the full conditional posterior distribution is then given by

$$f(\nu_1, \Delta_1, \tau_1, \ldots, \nu_G, \Delta_g, \tau_g \mu, \rho, m, r, \alpha, \beta | x_{11}, \ldots, y_{G,S_2})$$
$$\propto f(\mu) f(\rho) f(m) f(r) f(\alpha) f(\beta)\ f(x_{11}, \ldots, y_{G,S_2} | \nu_1, \Delta_1, \ldots) \prod_{i=1}^{G} f(\nu_i | \mu, \rho) \prod_{j=1}^{G} f(\Delta_i | m, r) \prod_{k=1}^{G} f(\tau_k | m, r)$$

As before, the posterior distributions of ν_g, Δ_g and τ_g are the same as in task C.1 and C.2 with $\mu_\nu = \mu$, $\sigma_\nu^2 = 1/\rho$, $\mu_\Delta =$, $\sigma_\Delta^2 = 1/r$, and for the gamma distributed τ_g with mean α and variance β, shape equals α^2/β and scale equals β/α. With the proposed uniform improper priors for μ, ρ, m, r, α and β, the conditional posterior distributions are given by:

$$f(\mu|\cdot) \sim N\left(\frac{\sum_{i=1}^{G}\nu_g}{G}, \frac{1}{\rho G}\right)$$
$$f(\rho|\cdot) \sim Gamma\left(\frac{G}{2}, scale = \frac{2}{\sum_{g=1}^{G}(\nu_g - \mu)^2}\right)$$
$$f(m|\cdot) \sim N\left(\frac{\sum_{i=1}^{G}\Delta_g}{G}, \frac{1}{rG}\right)$$
$$f(r|\cdot) \sim Gamma\left(\frac{G}{2}, scale = \frac{2}{\sum_{g=1}^{G}(\Delta_g - \mu)^2}\right)$$
$$f(\alpha,\beta)|\cdot) \propto \left(\frac{1}{\Gamma(\frac{\alpha^2}{\beta})(\frac{\beta}{\alpha})^{\frac{\alpha^2}{\beta}}}\right)^G (\prod_{g=1}^{G}\tau_g)^{\frac{\alpha^2}{\beta}} \exp(-\frac{\alpha}{\beta}\sum_{g=1}^{G}\tau_g)$$

As one can see, many of the posterior distributions can easily be sampled, which favours the Gibbs sampling. But the posterior distribution of α and β are not known and is not easy to sample from. In addition their posterior distributions are higly linked together. The strategy is therefore to use a Metropolis Hastings step for the joint posterior distribution of α and β within a Gibbs step for the rest of the parameters.

For the Metropolis Hastings algorithm one must choose a proposal distribution. For simplicity a random walk proposal is suggested with the bivariate normal distribution, which is symmetric. The acceptance probability, π, is then just given by:

$$\pi = \min(\frac{f(\alpha*, \beta*|\cdot)}{f(\alpha,\beta|\cdot)}, 1)$$

, where $\alpha*$ and $\beta*$ are proposed values given from the bivariate normal distribution, $N(\mu = (\alpha,\beta), \Sigma)$, where (α,β) are the old values. Care must be taken when either $\alpha*$ or $\beta*$ is negative which is outside their domain. By definition $\pi = 0$ so the proposed values will then be rejected. When it comes to the chosen covariance Σ, this is a difficult step where the parameters (variances of alpha, beta) have to be tuned. Too small variances give more acceptance of proposed values, but converges slowly since it will cover the whole space slowly. Too high variances will reject too many proposed values and also produce slow convergence. One most find tuning values which make sure that the markov chains will converge not too slowly.

The pseudo-code is then given by:

```
hybridGibbssample = function(dataGroupx, dataGroupy, NrOfSamples){

    # Initialize values for the unknown parameters
    # and compute variables necessary to compute the posterior distributions

    # Begin Gibbs sample with one Metropolis-Hastings step within.
    # The order of what parameters to update in the loop are not crucial.

    # The important thing is to always use the newest updates from all parameters

    # Preallocate the samples of the data in appropriate data structures
    for(i in 2:NrOfSamples){

    # 1. Use Gibbs iterations for easy sampled posterior distributions
```

```
    # 2. Compute one Metropolis Hastings step for alpha and
    # beta (at the same time)
    # using random walk proposal with proposal distribution bivariate normal with
    # tuning variances for alpha and beta.
     }
        # return samples of data
    }
```

D.2 Analysis

The corresponding R-code is given below:

```
hybridGibbs2 = function(datax,datay, N){
 library(base) # In order to use rowSums function
 # datax contains the data of the genes from groupx
 # datay contains the data of the genes from groupy
 # N is the number of iterations
 G = nrow(datax) # = nrow(datay) is the number of genes to evaluate

 s_1 = ncol(datax) #number of subjects for each gene evaluated for group x
 s_2 = ncol(datay) #number of subjects for each gene evaluated for group y

 nu_samples = matrix(data = rep(0,G*N), G, N)
 delta_samples = matrix(data = rep(0,G*N), G,N)
 tau_samples = matrix(data = rep(0,G*N),G,N)
 mu_samples = vector(mode = "numeric", length = N)
 rho_samples = vector(mode = "numeric", length = N)
 m_samples = vector(mode = "numeric", length = N)
 r_samples = vector(mode = "numeric", length = N)
 alpha_samples = vector(mode = "numeric", length = N)
 beta_samples = vector(mode = "numeric", length = N)


 # PRODUCE START VALUES FOR nu_g, delta_g, tau_g for g = 1,...,G, mu, rho, m,
 #r, alpha and beta
 # tau_g, rho, r, alpha and beta must be positive:
 nu_0 = rnorm(G, mean = 0, sd = 1)
 nu_samples[,1] = nu_0
 delta_0 = rnorm(G, mean = 0, sd = 1)
 delta_samples[,1] = delta_0
 tau_samples[,1] = runif(G)
 mu_0 = rnorm(1, mean = 0, sd = 1)
 mu_samples[1] = mu_0
 m_0 = rnorm(1, mean = 0, sd = 1)
 m_samples[1] = m_0
 rho_samples[1] = runif(1)
 r_samples[1] = runif(1)
 alpha_samples[1] = runif(1)
 beta_samples[1] = runif(1)
 prob = vector(mode = "numeric", length = N) # save acc. probabilities in
 lnRvector = vector(mode = "numeric", length = N)
 acrate = vector(mode = "numeric", length = N)
```

```
# Generate Gibbs samples:
for(i in 2:N){

  # sample for nu for g = 1,...,G:

  nu_samples[,i] = rnorm(G, (tau_samples[,i-1]*delta_samples[,i-1]*(s_2-s_1)
  + tau_samples[,i-1]*(rowSums(datax)+rowSums(datay))
  + mu_samples[i-1]*rho_samples[i-1])/
  (tau_samples[,i-1]*(s_1+s_2) + rho_samples[i-1] ),
  sqrt(1/(tau_samples[,i-1]*(s_1+s_2) + rho_samples[i-1] )))

  # sample for delta for g = 1,.., G

  delta_samples[,i] = rnorm(G,(tau_samples[,i-1]*nu_samples[,i]*(s_2-s_1)
  + tau_samples[,i-1]*(rowSums(datax)-rowSums(datay))
  + m_samples[i-1]*r_samples[i-1])/
  (tau_samples[,i-1]*(s_1+s_2) + r_samples[i-1] ),
  sqrt(1/(tau_samples[,i-1]*(s_1+s_2) + r_samples[i-1] )))

    #sample for tau for g = 1,...,G

  tau_samples[,i] = rgamma(G, shape = (alpha_samples[i-1]^2/(beta_samples[i-1])
  + (s_1+s_2)/2),
  scale = 1/(0.5*(rowSums((datax - nu_samples[,i] - delta_samples[,i])^2) +
  rowSums((datay - nu_samples[,i] + delta_samples[,i])^2))
  + alpha_samples[i-1]/beta_samples[i-1]))

  mu_samples[i] = rnorm(1, sum(nu_samples[,i])/G, sqrt(1/(rho_samples[i-1]*G)))
  rho_samples[i] = rgamma(1, shape = G/2
  , scale = 2/(sum((nu_samples[,i]-mu_samples[i])^2)))
  m_samples[i] = rnorm(1, sum(delta_samples[,i])/G, sqrt(1/(r_samples[i-1]*G)))
  r_samples[i] = rgamma(1, shape = G/2,
  scale = 2/(sum((delta_samples[,i]-m_samples[i])^2)))

  # SAMPLE WITH USE OF METROPOLIS HASTINGS FOR THE DISTRIBUTION OF ALPHA AND BETA:
  #Using a random walk chain with proposal distribution bivariate normal
  # Propose new values for alpha and beta
  alpha_propose = rnorm(1, alpha_samples[i-1],sd = 0.4)
  beta_propose = rnorm(1, beta_samples[i-1],sd = 0.4)
  if(alpha_propose <0 || beta_propose <0){
    alpha_samples[i] = alpha_samples[i-1]
    beta_samples[i] = beta_samples[i-1]
    acrate[i] = 0
  }
  else{
    # Compute acceptance probability

     r = alpha_propose^2/beta_propose
     s = alpha_propose/beta_propose
     t = alpha_samples[i-1]^2/beta_samples[i-1]
     u = alpha_samples[i-1]/beta_samples[i-1]
     tau = tau_samples[,i]
```

```
    ln_new = G*(r*log(s)-lgamma(r))+sum( (r-1)*log(tau)- tau*s )
    ln_old = G*(t*log(u)-lgamma(t))+sum( (t-1)*log(tau)- tau*u )
    lnR = ln_new-ln_old
    lnRvector[i] = lnR
    R = exp(lnR)
    prob[i] = min(R, 1)
    u = runif(1)
    if(u <= prob[i]){
      alpha_samples[i] = alpha_propose
      beta_samples[i] = beta_propose
      acrate[i] = 1
    }
    else{
      alpha_samples[i] = alpha_samples[i-1]
      beta_samples[i] = beta_samples[i-1]
      acrate[i] = 0
    }

  }
}

l = list(nu_samples, delta_samples, tau_samples, mu_samples, rho_samples, m_samples,
r_samples, alpha_samples, beta_samples,prob, lnRvector, acrate)
return(l)
}
```

The corresponding trace plots for ν_1, δ_1, τ_1, μ, ρ, m, r, α and β are given below after 100 000 iterations for the whole datasets (x1 and y1).

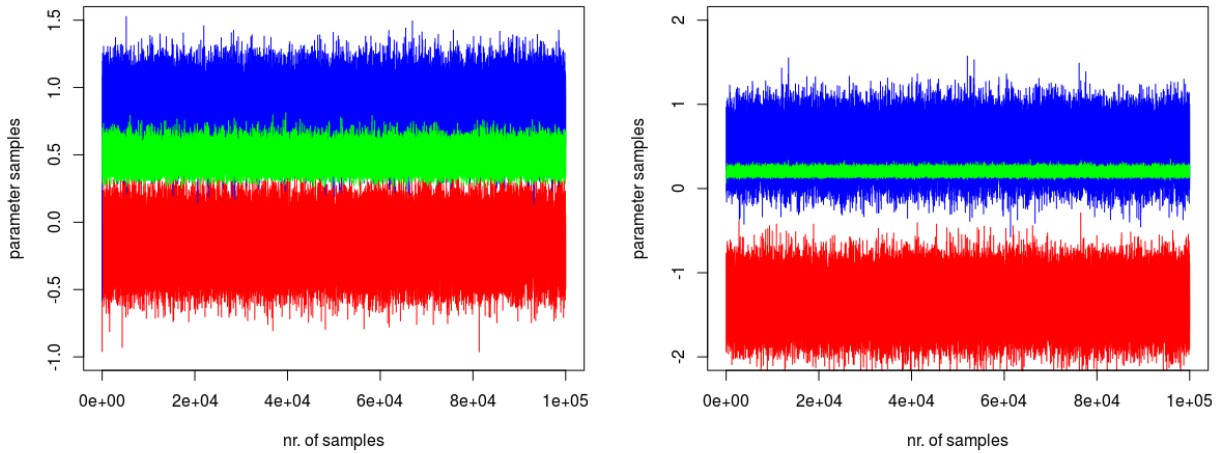


Figure 20: Trace plots for gene 1 and gene 2 from left to right respectively for ν_i (red), Δ_i (blue) and τ_i (green)

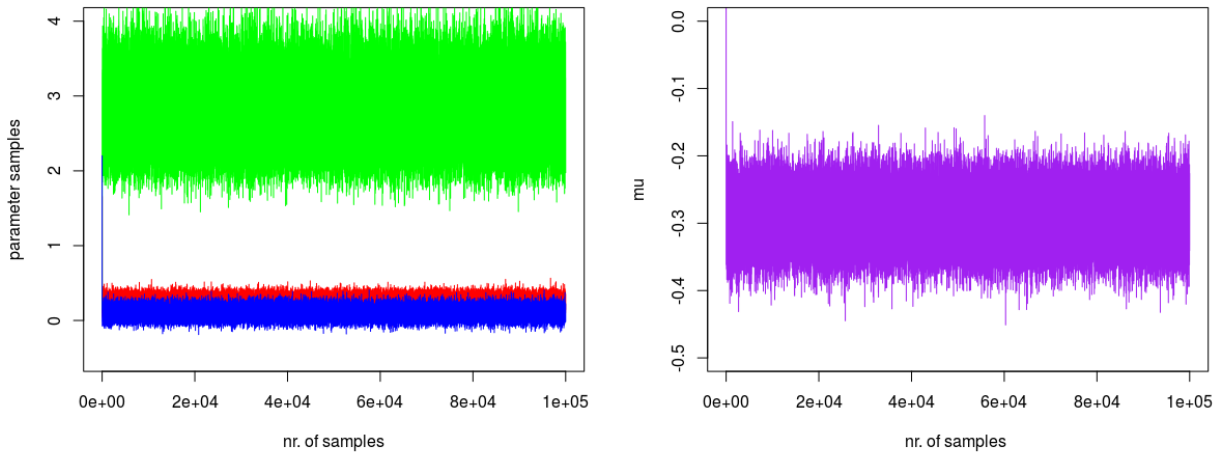


Figure 21: Trace plots for gene 3 and μ from left to right

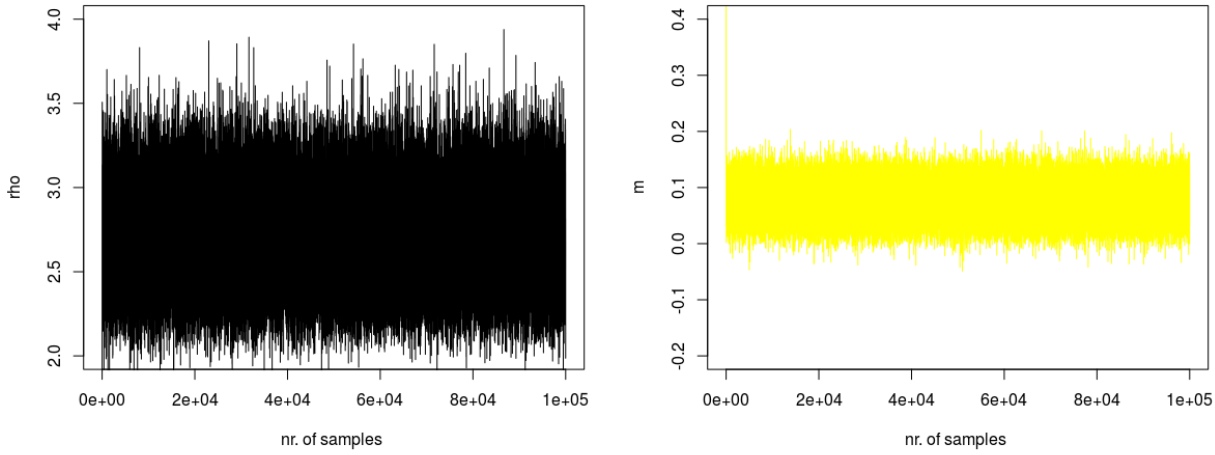


Figure 22: Trace plots for ρ and m from left to right

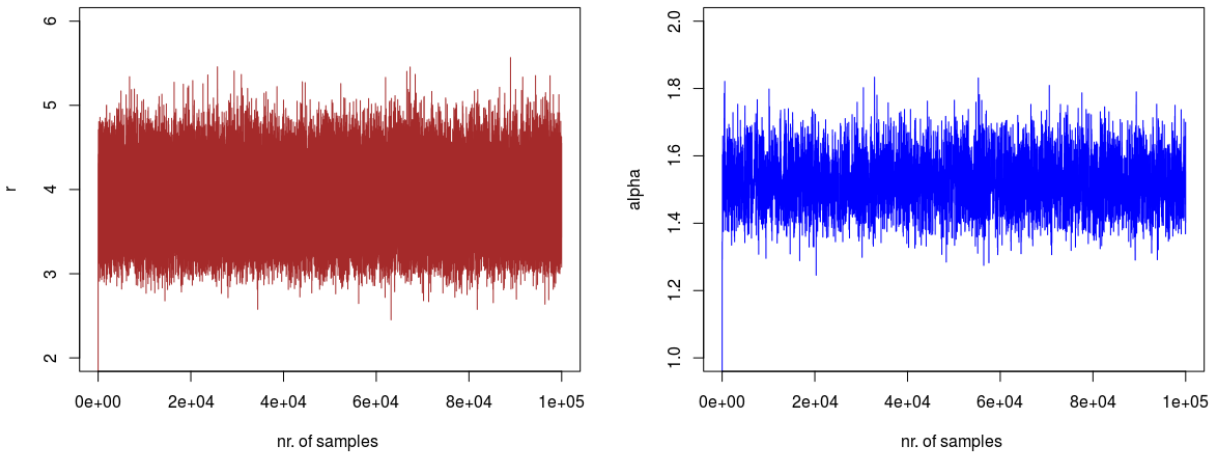


Figure 23: Trace plots for r and α from left to right

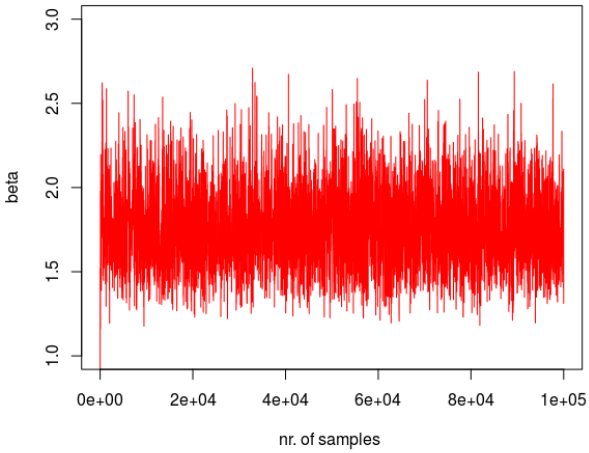


Figure 24: Trace plot of β

As all figures show, it appears as one obtains convergence pretty fast. Evaluating the samples after 50 000 iterations seems to be satisfactory. In fact, sampling for (much) more than 100 000 iterations one can conclude that this convergence is true. For instance, The estimates for gene 1 are now $\nu_1 = -0.16$, $\Delta_1 = 0.83$ and $\tau_1 = 0.48$. Comparing the estimated for all the three genes in task C and task D one sees that there are some small differences. In task D we have less information, uncertainty in the hyperparameters are considered. Therefore, Comparing gene 1,2 and 3 from task D in figure 20 and figure 21 with gene 1, 2 and 3 in figure 17, there are more uncertainties in the parameters ν_i, Δ_i and τ_i (higher variances) when there are uncertainties in the hyperparameters.

Using the subdata x2 and y2, the uncertainties are even bigger. The reason is that the algorithm only sees the data. The more data, the more information and therefore less unertainties. Plotting the estimated marginal densities for α and β:

Figure 25: Estimated marginal density plots for α and β

, one sees that the distributions are some kind of similar to a normal distribution centered around 1.5 for α and 1.75 for β.