Final Exam

Louis Bensard

May 9, 2018

Problem 1:

(a)

First, to make the derivation easier, let $X_j = x_j - \bar{x}$ and $\mu_{ij} = \alpha_i + \beta_i X_j$. We have $Y_{ij} \sim N(\mu_{ij}, \sigma^2)$ thus:

$$f(y_{ij}) = \frac{1}{\sqrt{2\pi}\sigma} exp\{\frac{-1}{2\sigma^2}(y_{ij} - \mu_{ij})^2\}$$

Let $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_{30})$ and $\boldsymbol{\beta} = (\beta_1, ..., \beta_{30})$, therefore the likelihood is:

$$L(\boldsymbol{\alpha}, \boldsymbol{\beta}, \sigma^2 | y) = \prod_{i=1}^{n} \prod_{j=1}^{5} f(y_{ij})$$

Thus, taking the log, developing and expanding gives the following log-likelihood:

$$l(\boldsymbol{\alpha}, \boldsymbol{\beta}, \sigma^2) = -5n \cdot log(\sigma) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} \sum_{j=1}^{5} (y_{ij} - \mu_{ij})^2 + C$$

Taking the derivative of l w.r.t α_i and setting it equal to 0, we get the following MLE of α_i:

$$\hat{\alpha_i} = \frac{1}{5} \sum_{j=1}^{5} y_{ij} - \hat{\beta}_i X_j$$

Taking the derivative of l w.r.t β_i and setting it equal to 0, we get the following MLE of β_i:

$$\hat{\beta}_i = \frac{\sum_{j=1}^{5} (y_{ij} - \hat{\alpha_i}) X_j}{\sum_{j=1}^{5} X_j^2}$$

Plugging $\hat{\alpha_i}$ into $\hat{\beta}_i$ and we get:

$$\hat{\beta}_i = \frac{5}{4} \frac{\sum_{j=1}^{5} [y_{ij} - \frac{1}{5}(\sum_{j=1}^{5} y_{ij})] X_j}{\sum_{j=1}^{5} X_j^2}$$

Taking the derivative of l w.r.t σ^2 and setting it equal to 0, we get the following MLE of σ^2:

$$\hat{\sigma^2} = \frac{1}{5n} \sum_{i=1}^{n} \sum_{j=1}^{5} (y_{ij} - \hat{\alpha_i} - \hat{\beta}_i X_j)^2$$

Quick verification that the extrema found are actually maxima:

$$\frac{\partial^2 l}{\partial \alpha_i^2} = \frac{-5}{\sigma^2} < 0 \Rightarrow \hat{\alpha_i} \text{ is a maximum}$$
$$\frac{\partial^2 l}{\partial \beta_i^2} = \frac{-\sum X_j}{\sigma^2} < 0 \Rightarrow \hat{\beta_i} \text{ is a maximum}$$
$$\frac{\partial^2 l}{\partial (\sigma^2)^2}(\hat{\sigma^2}) = \frac{-(5n)^3}{2[\sum_i \sum_j (y_{ij} - \mu_{ij})^2]^2} < 0 \Rightarrow \hat{\sigma^2} \text{ is a maximum}$$

Let's now compute those MLE's :

```
data = read.csv('C:/Users/Louis/Documents/UPMC/M1/Spring 2018/MATH 534/Final exam/rats_data.csv',
                header=T)

n = length(data[,1])

xj = c(8,15,22,29,36)
Xj = xj - 22

y = data[,-1] #cleaning data

#initializing the vector of MLE's
alpha_hat = rep(0,n)
beta_hat = rep(0,n)
sig_sq_hat = 0

#we compute beta_hat first
for(i in 1:n){

    beta_hat[i] = (5/4)*sum((y[i,] - (1/5)*sum(y[i,]))*Xj) / sum(Xj^2)
}

#then we use beta_hat to compute alpha_hat
for(i in 1:n){

    alpha_hat[i] = (1/5)*sum(y[i,] - beta_hat[i]*Xj)
}

#finally we use alpha_hat and beta_hat to compute sigma_sd_hat
for(i in 1:n){

    sig_sq_hat = sig_sq_hat + sum((y[i,] - alpha_hat[i] - beta_hat[i]*Xj)^2)
}

sig_sq_hat = sig_sq_hat/(5*n)

print(alpha_hat)
```

```
##  [1] 239.8 248.0 252.8 232.2 231.2 250.0 228.2 248.6 284.8 218.4 258.8
## [12] 227.6 242.4 269.2 242.8 245.4 231.8 240.4 254.2 241.6 248.8 224.6
## [23] 228.0 245.2 234.2 254.4 254.8 243.0 217.0 241.4
```

```
print(beta_hat)
```

```
##  [1] 7.53571 9.14286 8.21429 6.35714 8.35714 7.71429 7.39286 8.10714
```

```
##  [9] 9.14286 7.17857 8.73214 7.62500 7.69643 8.55357 6.48214 7.30357
## [17] 7.87500 7.17857 8.08929 7.51786 8.08929 7.19643 7.01786 7.25000
## [25] 8.91071 8.32143 7.26786 7.17857 6.89286 7.64286
```

```
print(sig_sq_hat)
```

```
## [1] 258.047
```

(b)

(i)

Considering the given priors, the likelihood we computed in part (a) and the fact that we assume that α_i, β_i and τ are independents, the posterior distribution we want to generate values from is:

$$f(\boldsymbol{\alpha}, \boldsymbol{\beta}, \tau, \alpha, \beta | \boldsymbol{y}) \propto L(\boldsymbol{\alpha}, \boldsymbol{\beta}, \sigma^2 | y) \cdot \prod_{i=1}^{n} [f(\alpha_i|\alpha) \cdot f(\beta_i|\beta)] \cdot f(\tau) f(\alpha) f(\beta)$$

(Note: As mentioned before, $\boldsymbol{\alpha}$ is the vector of $\alpha_i's$ while α is the hyperprior, they are not the same parameter. Same distinction for $\boldsymbol{\beta}$ and β)

To implement Gibbs Sampler, we need the following conditionals :

$$\begin{array}{ll} f(\alpha_i|\alpha_{-i}, \boldsymbol{\beta}, \tau, \alpha, \beta, \boldsymbol{y}) & \forall i \in \{1, ..., 30\} \\ f(\beta_i|\beta_{-i}, \boldsymbol{\alpha}, \tau, \alpha, \beta, \boldsymbol{y}) & \forall i \in \{1, ..., 30\} \\ f(\tau|\boldsymbol{\alpha}, \boldsymbol{\beta}, \alpha, \beta, \boldsymbol{y}) & \\ f(\alpha|\boldsymbol{\alpha}, \boldsymbol{\beta}, \tau, \beta, \boldsymbol{y}) & \\ f(\beta|\boldsymbol{\alpha}, \boldsymbol{\beta}, \alpha, \tau, \boldsymbol{y}) & \end{array}$$

Let's find those conditional distributions:
Notation: $f(\alpha|\bullet)$ is the density of α given any other mentioned parameter (excluding alpha of course), $\bar{\boldsymbol{\alpha}} = \frac{1}{n}\sum_{i=1}^{n} \alpha_i$, $\bar{\boldsymbol{\beta}} = \frac{1}{n}\sum_{i=1}^{n} \beta_i$ and let L be the Likelihood.

1)

$$\begin{aligned} f(\tau|\bullet) &\propto L \cdot f(\tau) \\ &\propto gamma\left(\frac{5n}{2} + 1, \frac{1}{2}\sum_i \sum_j (y_{ij} - \mu_{ij})^2\right) \cdot gamma(10^{-3}, 10^{-3}) \\ &\propto gamma\left(\frac{5n}{2} + 10^{-3}, \frac{1}{2}\sum_i \sum_j (y_{ij} - \mu_{ij})^2 + 10^{-3}\right) \end{aligned}$$

2)

$$\begin{aligned} f(\alpha|\bullet) &\propto \prod_{i=1}^{n} f(\alpha_i|\alpha) \cdot f(\alpha) \\ &\propto N\left(\bar{\boldsymbol{\alpha}}, \frac{100}{n}\right) \cdot N(0, 10^4) \\ &\propto N\left(\frac{\bar{\boldsymbol{\alpha}} \cdot 10^4}{\frac{100}{n} + 10^4}, \frac{10^4}{1 + 100n}\right) \end{aligned}$$

3)

$$\begin{aligned} f(\beta|\bullet) &\propto \prod_{i=1}^{n} f(\beta_i|\beta) \cdot f(\beta) \\ &\propto N\left(\bar{\boldsymbol{\beta}}, \frac{1}{n}\right) \cdot N(0, 10^4) \\ &\propto N\left(\frac{\bar{\boldsymbol{\beta}} \cdot 10^4}{\frac{1}{n} + 10^4}, \frac{10^4}{1 + n \cdot 10^4}\right) \end{aligned}$$

4)

$$\begin{aligned} f(\alpha_i|\bullet) &\propto L \cdot \prod_{i=1}^{n} f(\alpha_i|\alpha) \\ &\propto N\left(\frac{\sum_j (y_{ij} - \beta_i X_j)}{5}, \frac{1}{5\tau}\right) \cdot N(\alpha, 100) \\ &\propto N\left(\frac{20\sum_j (y_{ij} - \beta_i X_j) + \frac{\alpha}{5\tau}}{100 + \frac{1}{5\tau}}, \frac{100}{500\tau + 1}\right) \end{aligned}$$

5)

$$\begin{aligned} f(\beta_i|\bullet) &\propto L \cdot \prod_{i=1}^{n} f(\beta_i|\beta) \\ &\propto N\left(\frac{\sum_j (y_{ij} - \alpha_i) X_j}{\sum_j X_j^2}, \frac{1}{\tau \sum_j X_j^2}\right) \cdot N(\beta, 1) \\ &\propto N\left(\frac{\sum_j (y_{ij} - \alpha_i) X_j + \frac{\beta}{\tau}}{\frac{1}{\tau} + \sum_j X_j^2}, \frac{1}{1 + \tau \sum_j X_j^2}\right) \end{aligned}$$

(ii)

Here is the Gibbs Sampling algorithm I am going to use:

Step 1: Start with an initial value $\theta^{(0)} = (\beta_1^{(0)}, ..., \beta_{30}^{(0)}, \tau^{(0)}, \alpha^{(0)}, \beta^{(0)})$ and set $t = 0$. We will use the MLE's computed in part (a) and the prior distributions of α and β to get those starting values.

Step 2: First, generate in parallel all the $\alpha_i^{(t+1)}$'s:

$$\alpha_i^{(t+1)} \sim f(\alpha_i|\beta_i^{(t)}, \alpha^{(t)}, \tau^{(t)}) \quad \forall i \in \{1, ..., 30\}$$

Then use those $\alpha_i^{(t+1)}$'s to generate all the $\beta_i^{(t+1)}$'s:

$$\beta_i^{(t+1)} \sim f(\beta_i|\alpha_i^{(t+1)}, \beta^{(t)}, \tau^{(t)}) \quad \forall i \in \{1, ..., 30\}$$

Then, generate :

$$\begin{aligned} \alpha^{(t+1)} &\sim f(\alpha|\alpha_1^{(t+1)}, ..., \alpha_{30}^{(t+1)}) \\ \beta^{(t+1)} &\sim f(\beta|\beta_1^{(t+1)}, ..., \beta_{30}^{(t+1)}) \\ \tau^{(t+1)} &\sim f(\tau|\alpha_1^{(t+1)}, ..., \alpha_{30}^{(t+1)}, \beta_1^{(t+1)}, ..., \beta_{30}^{(t+1)}) \end{aligned}$$

Step 3: Increment t and go to step 2 until reaching the number of simulations

(iii)

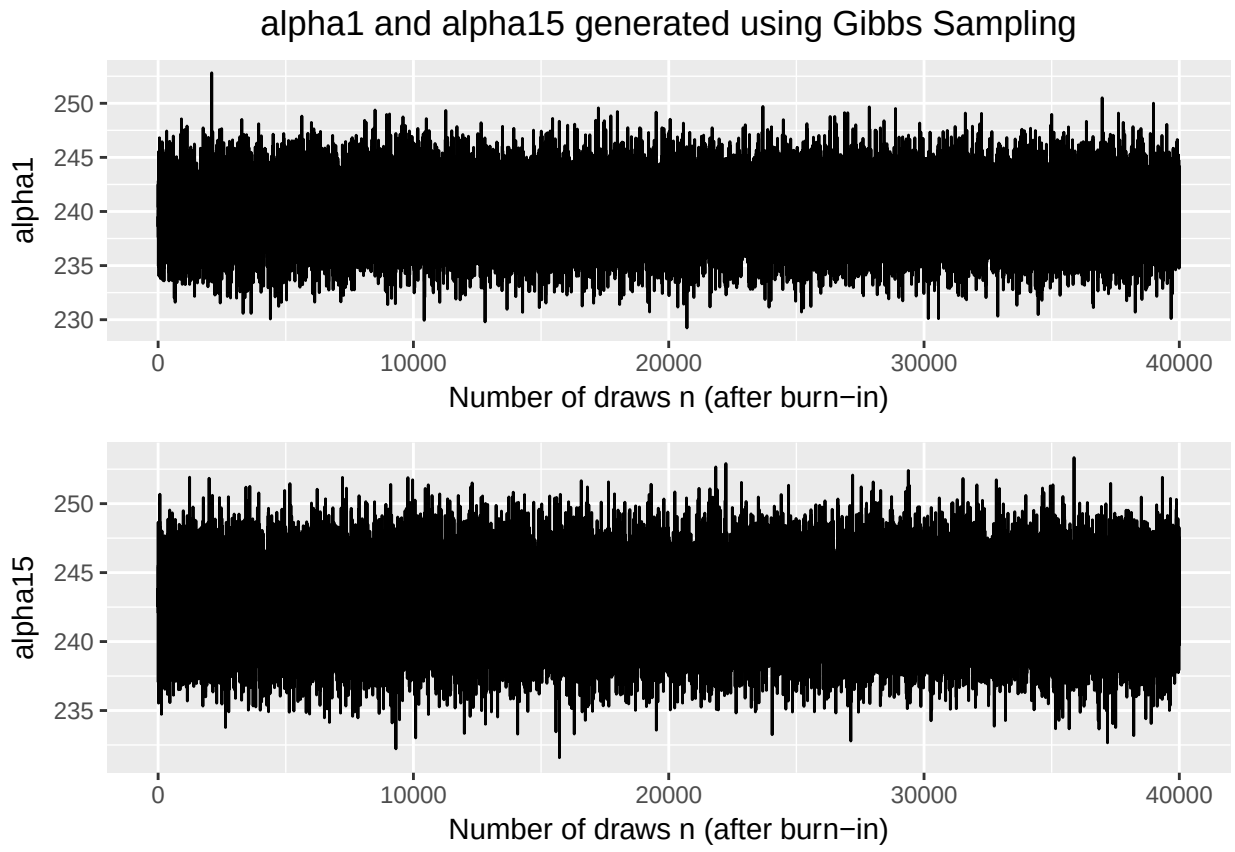
```
library(ggplot2)
library(gridExtra)

#functions to generate the required conditionals
gen_tau_given_all <- function(alpha_vect, beta_vect,n=30){

    shape = (5*n/2) + 0.001

    #we create the matrix of mu_ij
    mu_mat = beta_vect %*% t(Xj) + alpha_vect

    #we sum the 5 elements of each of the 30 rows together (we get a 30x1 vector)
    #and we sum the element of this vector together
    rate = (1/2)*sum(rowSums((y - mu_mat)^2)) + 0.001

    return(rgamma(1, shape=shape, rate=rate))
}

gen_alpha_given_all <- function(alpha_vect, n=30){

    mean = mean(alpha_vect)*(10^4) / ((100/n) + 10^4)
    var = (10^4) / (1+100*n)

    return(rnorm(1,mean=mean, sd=sqrt(var)))
}

gen_beta_given_all <- function(beta_vect, n=30){

    mean = mean(beta_vect)*(10^4) / ((1/n) + 10^4)
    var = (10^4) / (1+(10^4)*n)

    return(rnorm(1, mean=mean, sd=sqrt(var)))
}


gen_alpha_vect_given_all <- function(alpha, beta_vect, tau){

    beta_Xj = beta_vect %*% t(Xj)

    mean = (20*rowSums(y - beta_Xj) + (alpha / (5*tau))) / (100 + (1/(5*tau)) )
    var = 100 / (500*tau + 1)

    return(rnorm(30,mean=mean, sd=sqrt(var)))
}



gen_beta_vect_given_all <- function(beta, alpha_vect, tau){

    sum_Xj_sq = sum(Xj^2)
    y_alpha_Xj = t(t(y - alpha_vect)*Xj)
```

```
    mean = (rowSums(y_alpha_Xj) + (beta/tau)) / ((1/tau) + sum_Xj_sq)
    var = 1 / (1 + tau*sum_Xj_sq)

    return(rnorm(30, mean=mean, sd=sqrt(var)))
}


#Gibbs Sampling function
Gibbs <- function(nsim, theta_0){

    theta_Gibbs = matrix(NA,nsim,63)
    #each row is a vector theta partitionned the same way as described in the algorithm above.
    #the ith row of the matrix represents theta draws at t=i-1
    #theta_t = (alpha1,...,alpha30, beta1,...,beta30, tau, alpha, beta)

    #the first row is the initial values at t=0
    theta_Gibbs[1,(31:63)] = theta_0

    for(t in 1:(nsim-1)){

        beta_vect_t = theta_Gibbs[t, (31:60)]
        tau_t = theta_Gibbs[t,61]
        alpha_t = theta_Gibbs[t,62]
        beta_t = theta_Gibbs[t,63]

        #generating in parallel alpha_i's
        theta_Gibbs[(t+1), (1:30) ] = gen_alpha_vect_given_all(alpha_t, beta_vect_t, tau_t)
        alpha_vect_t1 = theta_Gibbs[(t+1), (1:30)]

        #using alpha'is at (t+1) to generate beta's at (t+1)
        theta_Gibbs[(t+1), (31:60) ] = gen_beta_vect_given_all(beta_t, alpha_vect_t1, tau_t)
        beta_vect_t1 = theta_Gibbs[(t+1), (31:60)]

        #generating tau at (t+1)
        theta_Gibbs[(t+1),61] = gen_tau_given_all(alpha_vect_t1, beta_vect_t1)

        #generating alpha at (t+1)
        theta_Gibbs[(t+1),62] = gen_alpha_given_all(alpha_vect_t1)

        #generating beta at (t+1)
        theta_Gibbs[(t+1),63] = gen_beta_given_all(beta_vect_t1)

    }

    return(theta_Gibbs)

}
```

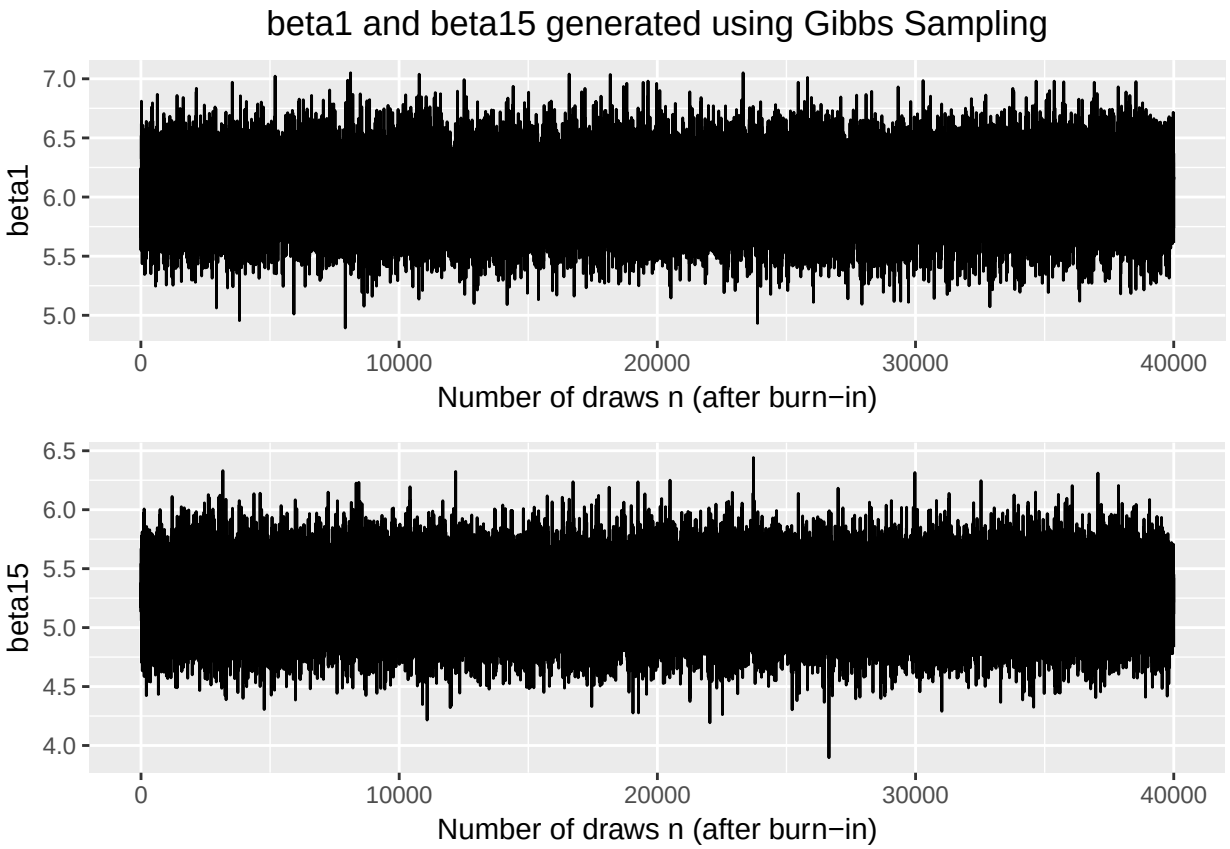
(iv)

```
#generating starting values
alpha_0 = rnorm(1,mean=0,sd = 100); beta_0 = rnorm(1,mean=0,sd = 100)
```

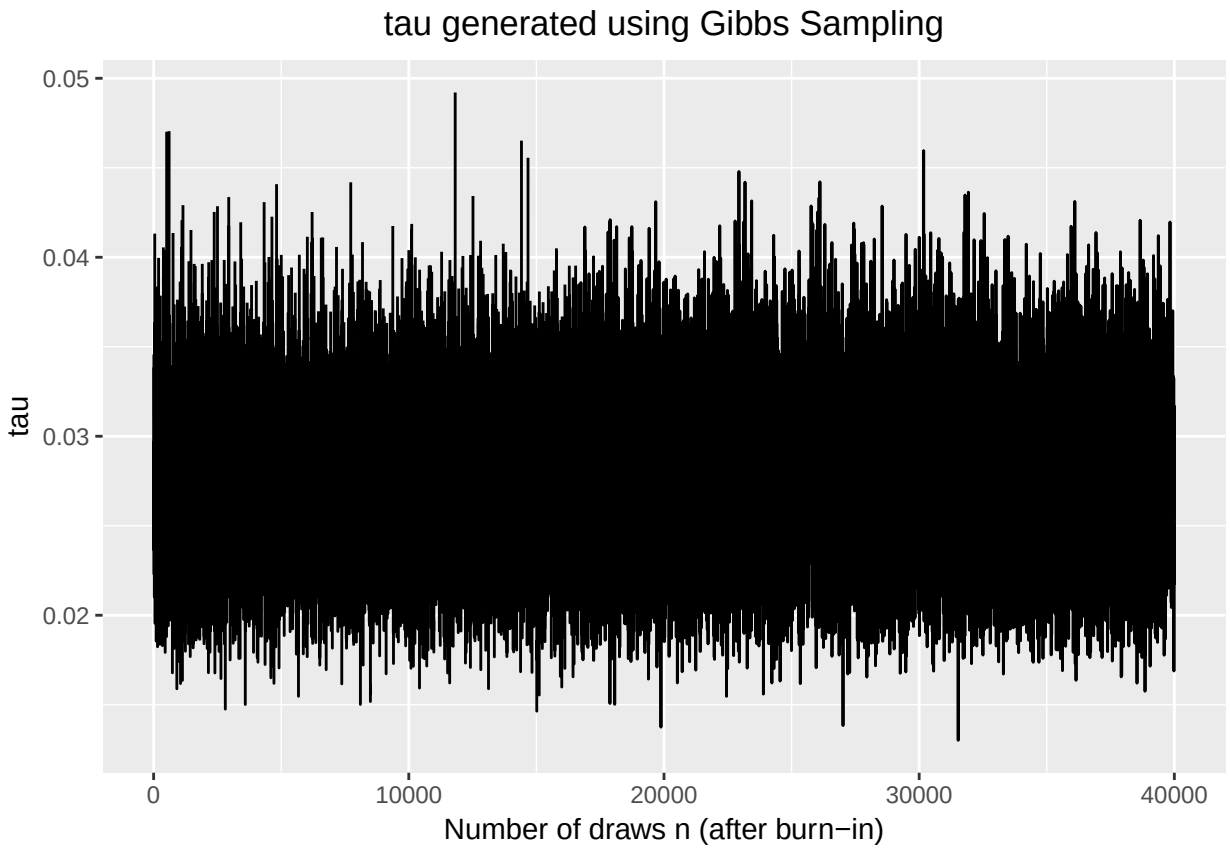
```r
#we use the MLE's computed in part a) for the other staring values
theta_0 = c(beta_hat, 1/sig_sq_hat, alpha_0, beta_0)

#number of simulations
nsim = 50000

#all the values generated are stored in a matrix
Gibbs_mat = Gibbs(nsim, theta_0)

#we take out the generated parameter values that we need
alpha_vect_1 = Gibbs_mat[,1]
alpha_vect_15 = Gibbs_mat[,15]
beta_vect_1 = Gibbs_mat[,31]
beta_vect_15 = Gibbs_mat[,45]
tau_vect = Gibbs_mat[,61]

#burn-in
burn = 10000
alpha_vect_1 = alpha_vect_1[(burn+1):nsim]
alpha_vect_15 = alpha_vect_15[(burn+1):nsim]
beta_vect_1 = beta_vect_1[(burn+1):nsim]
beta_vect_15 = beta_vect_15[(burn+1):nsim]
tau_vect = tau_vect[(burn+1):nsim]


df = data.frame(v = seq(1,nsim-burn), alpha_vect_1, alpha_vect_15)
p1 <- ggplot(data=df, aes(x=v, y=alpha_vect_1))+
  geom_line()+
  labs(x="Number of draws n (after burn-in)",
       title = "alpha1 and alpha15 generated using Gibbs Sampling" , y='alpha1')+
  theme(plot.title = element_text(hjust = 0.5))

p2 <- ggplot(data=df, aes(x=v, y=alpha_vect_15))+
  geom_line()+
  labs(x="Number of draws n (after burn-in)", y='alpha15')

grid.arrange(p1,p2, nrow=2)
```




```
df = data.frame(v = seq(1,nsim-burn), beta_vect_1, beta_vect_15)
p3 <- ggplot(data=df, aes(x=v, y=beta_vect_1))+
  geom_line()+
  labs(x="Number of draws n (after burn-in)",
       title = "beta1 and beta15 generated using Gibbs Sampling", y='beta1')+
  theme(plot.title = element_text(hjust = 0.5))

p4 <- ggplot(data=df, aes(x=v, y=beta_vect_15))+
  geom_line()+
  labs(x="Number of draws n (after burn-in)", y='beta15')

grid.arrange(p3,p4, nrow=2)
```



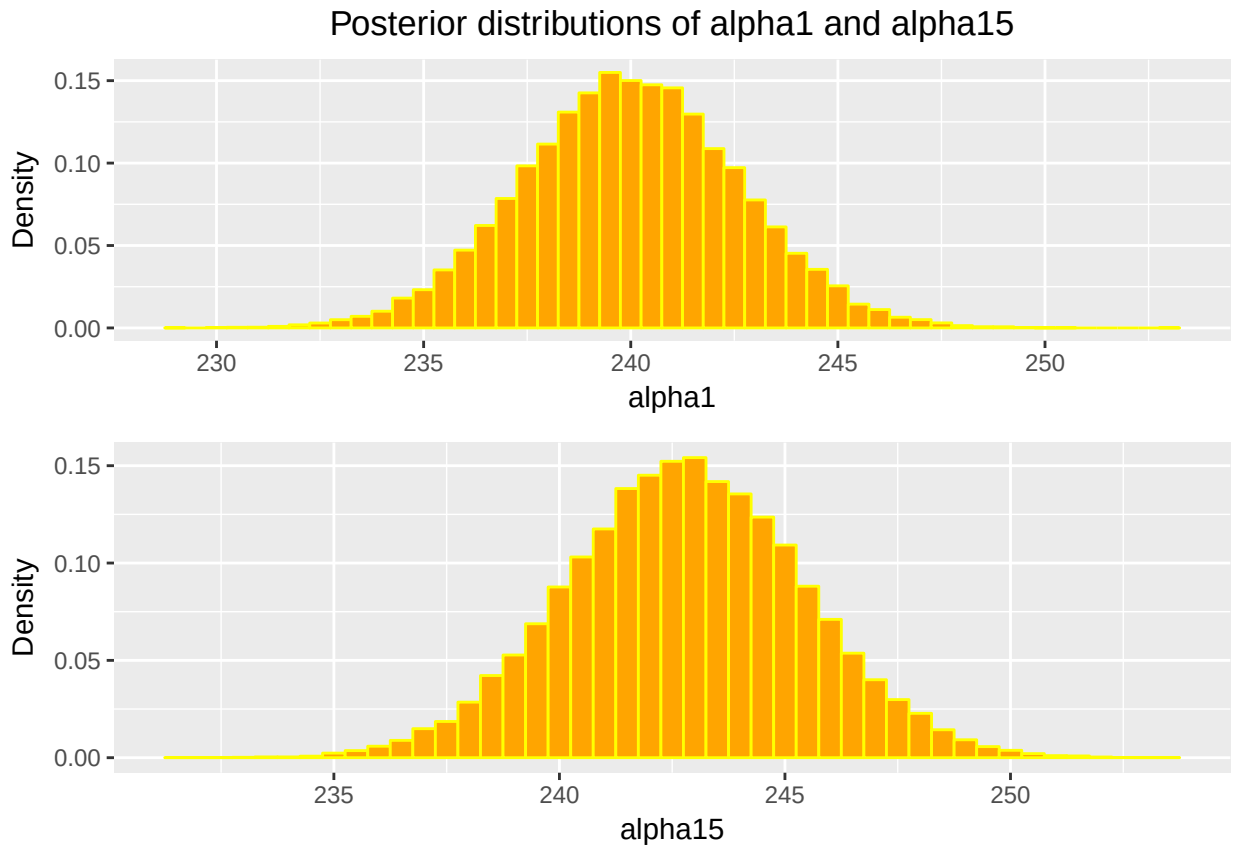

```
df = data.frame(v = seq(1,nsim-burn), tau_vect)
ggplot(data=df, aes(x=v, y=tau_vect))+
  geom_line()+
  labs(x="Number of draws n (after burn-in)",
       title = "tau generated using Gibbs Sampling", y='tau')+
  theme(plot.title = element_text(hjust = 0.5))
```



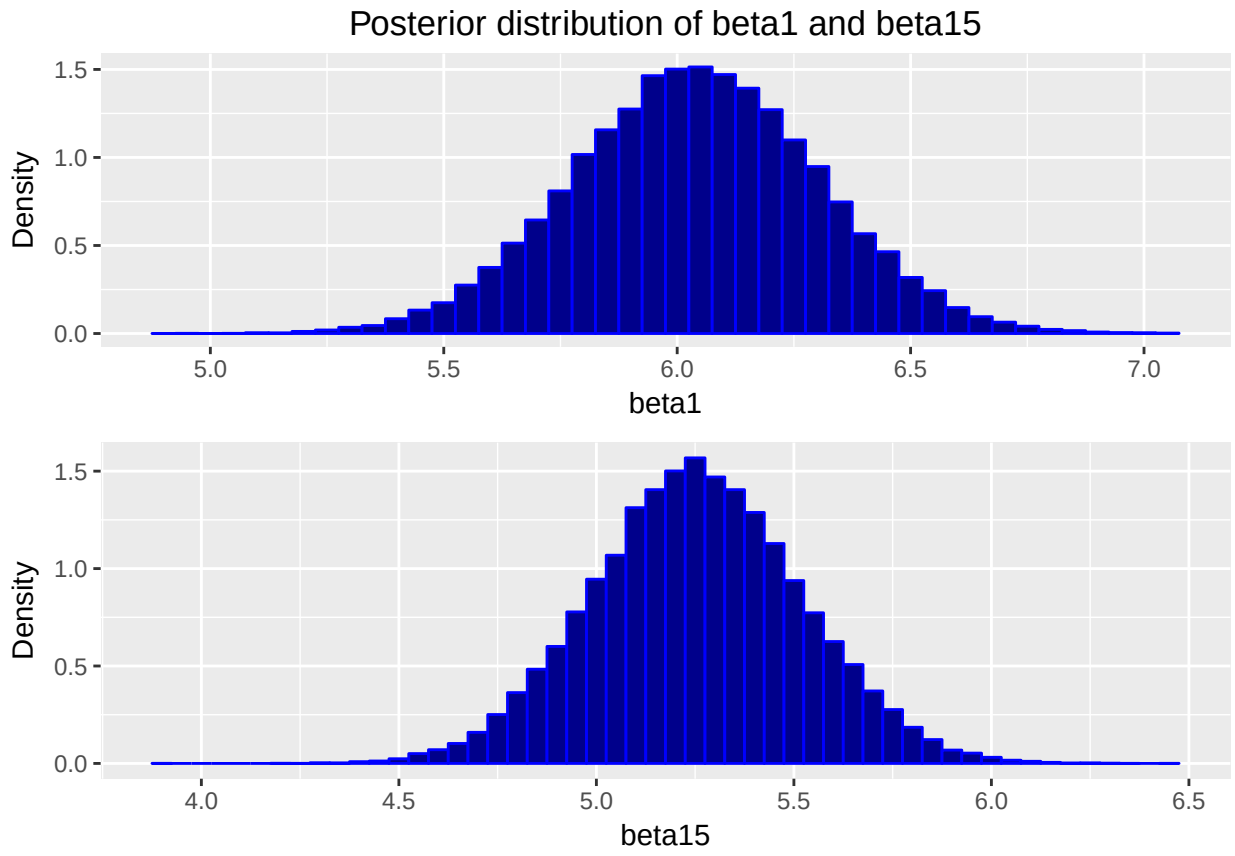
The mixing of the chains of α_1, α_{15}, β_1, β_{15} and τ look really good. We can see that each chain visits every point of the parameter space, they do not get stuck anywhere. It is a sign of good mixing.

(v)

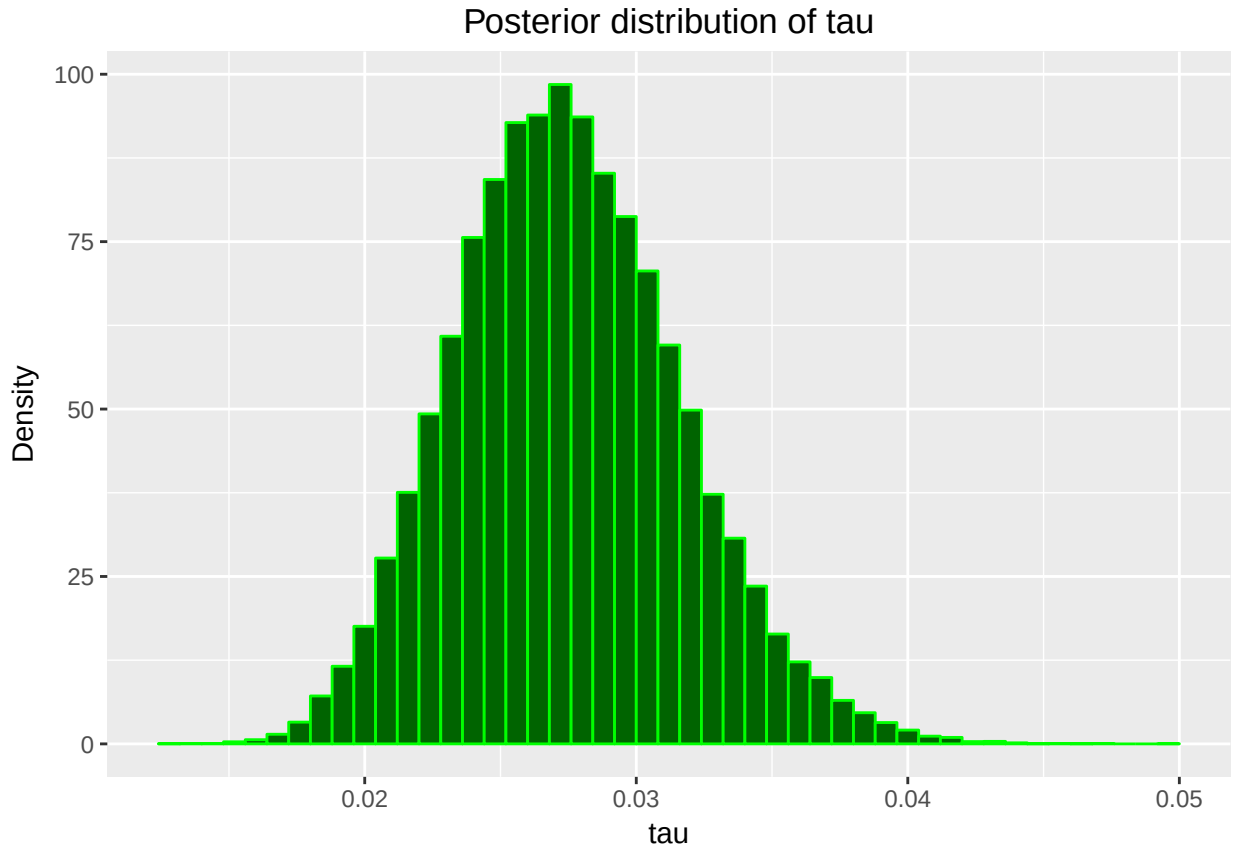
```
df = data.frame(alpha_vect_1, alpha_vect_15, beta_vect_1, beta_vect_15, tau_vect)
p1 <- ggplot(df, aes(x=alpha_vect_1))+
    geom_histogram(aes(y=..density..), fill="orange", color="yellow", binwidth=.5)+
    labs(y = "Density", x = "alpha1", title="Posterior distributions of alpha1 and alpha15")+
    theme(plot.title = element_text(hjust = 0.5))

p2 <- ggplot(df, aes(x=alpha_vect_15))+
    geom_histogram(aes(y=..density..), fill="orange", color="yellow", binwidth=.5)+
    labs(y = "Density", x = "alpha15")

grid.arrange(p1,p2, nrow=2)
```




```
p3 <- ggplot(df, aes(x=beta_vect_1))+
    geom_histogram(aes(y=..density..), fill="darkblue", color="blue", binwidth=.05)+
    labs(y = "Density", x = "beta1", title="Posterior distribution of beta1 and beta15")+
    theme(plot.title = element_text(hjust = 0.5))

p4 <- ggplot(df, aes(x=beta_vect_15))+
    geom_histogram(aes(y=..density..), fill="darkblue", color="blue", binwidth=.05)+
    labs(y = "Density", x = "beta15")

grid.arrange(p3,p4, nrow=2)
```




```
ggplot(df, aes(x=tau_vect))+
    geom_histogram(aes(y=..density..), fill="darkgreen", color="green", binwidth=.0008)+
    labs(y = "Density", x = "tau", title="Posterior distribution of tau")+
    theme(plot.title = element_text(hjust = 0.5))
```



As a point estimate for each parameter, we will provide their mean:

```
ma1 = mean(alpha_vect_1); ma15 = mean(alpha_vect_15)
mb1= mean(beta_vect_1); mb15 = mean(beta_vect_15)
mt = mean(tau_vect)
```

$\bar{\alpha_1} = 239.983218$, $\bar{\alpha_{15}} = 242.774296$, $\bar{\beta_1} = 6.039517$, $\bar{\beta_{15}} = 5.254102$ and $\bar{\tau} = 0.027496$. I am happy about those mean values, they are of the same order of what I would expect knowing the MLE's and the priors.

(c)

```
#lower and upper bounds of the CI of alpha_1
L_a1 = sort(alpha_vect_1)[0.025*(nsim - burn)]
U_a1 = sort(alpha_vect_1)[0.975*(nsim - burn)]

#lower and upper bounds of the CI of beta_1
L_b1 = sort(beta_vect_1)[0.025*(nsim - burn)]
U_b1 = sort(beta_vect_1)[0.975*(nsim - burn)]
```

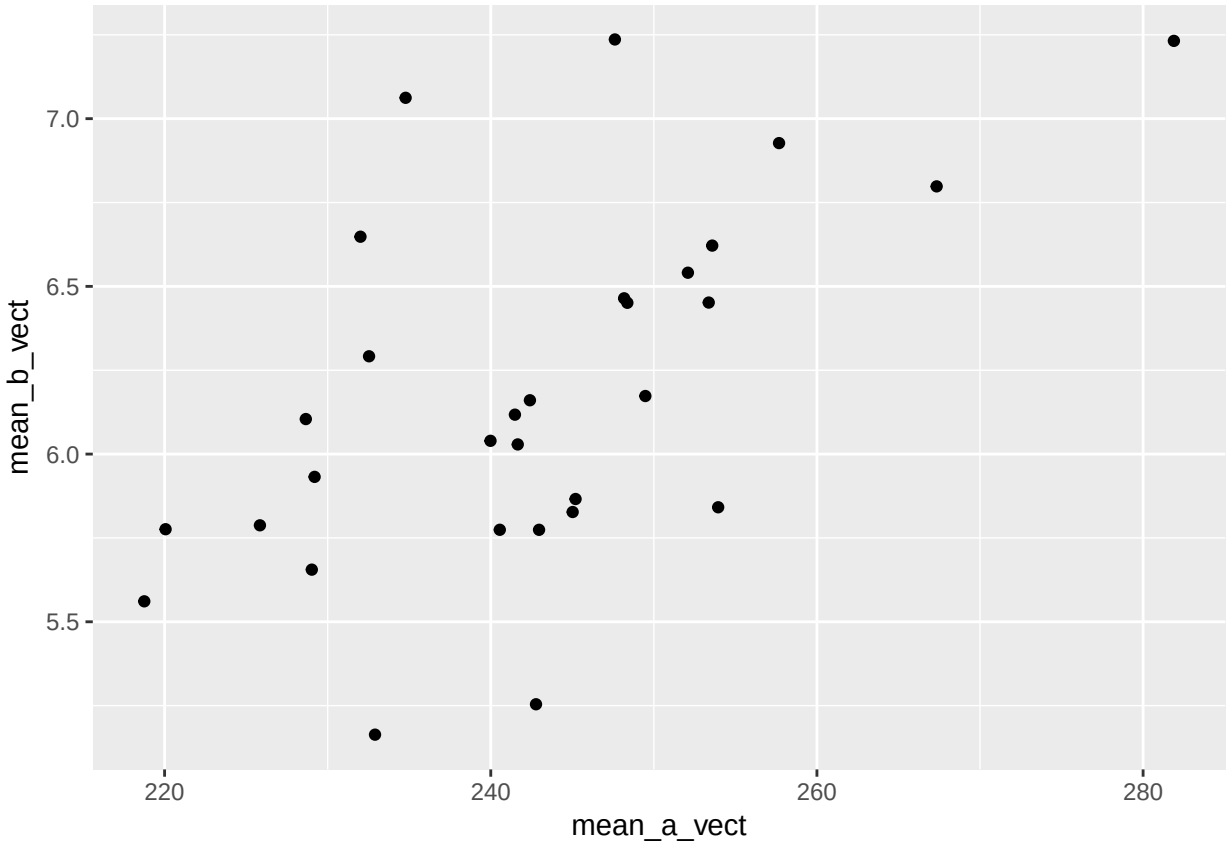
95% Credible Interval of α_1 : [234.812776, 245.110895], thus given the observed data, there is 95% chance that the true value of α_1 falls within this interval.
95% Credible Interval of β_1 : [5.521216, 6.555398], thus given the observed data, there is 95% chance that the true value of β_1 falls within this interval.

(d)

First I wanted to say that given the very limited amount of data, the independence of the prior distributions should lead to an independence of the posterior.

Then I decided to take the mean of each vector of α_i and the mean of each vector of β_i generated and plot them against each other to perhaps prove or disrpove the indenpence assumption.

```
mean_a_vect = colMeans(Gibbs_mat[((burn+1):50000), (1:30)])

mean_b_vect = colMeans(Gibbs_mat[((burn+1):50000), (31:60)])

df = data.frame(mean_a_vect, mean_b_vect)
ggplot(df, aes(x = mean_a_vect, y=mean_b_vect))+
    geom_point()
```



We can observe somewhat of a positive relationship between the means, not a very exchaustive one but still. We can see that for small values of mean(α_i), we have small values of mean(β_i). Similar remark for high values of mean(α_i) and mean(β_i). So based on the plot, I will conclude that the posterior does not support the independence assumption.

Problem 2:

(a)

We want to estimate $\theta = E(X^2)$ with X having a density proportional to $q(x) = e^{-|x^3|/3}$, $-\infty < x < \infty$. Let $h(x) = x^2$, we are going to draw from $g(x) \sim N(0,1)$ and use Importance Sampling to estimate θ. Here is the algorithm:

Step 1: Draw $X_1, ..., X_n \sim g(x)$

Step 2: For each value generated, compute the standardized weights:

$$w(X_i) = \frac{q(X_i)/g(X_i)}{\sum_{i=1}^{n} q(X_i)/g(X_i)}$$

Step 3: Use those weights to compute the estimated value $\hat{\theta}_{IS}$ of θ:

$$\hat{\theta}_{IS} = \sum_{i=1}^{n} h(X_i) \cdot w(X_i)$$

(b)

```
q <- function(x) return(exp(-abs(x^3)/3))

h <- function(x) return(x^2)

g <- function(x) return(dnorm(x,0,1))

w_star <- function(x) return(q(x)/g(x))

#step 1
n = 10000
X = rnorm(n,0,1)

#step 2

w_star = w_star(X)
w = w_star / sum(w_star)

#step 3
theta_IS = sum(h(X)*w)

print(theta_IS)
```

```
## [1] 0.774172
```

Thus an estimate for θ is $\hat{\theta}_{IS} = 0.774172$.

Problem 3:

(a)

Our goal is to generate values from the bivariate normal distribution with parameters $\mu = (1,2)^T$ and $\Sigma = \begin{pmatrix} 1 & 0.9 \\ 0.9 & 1 \end{pmatrix}$, using the proposal density $g(\cdot|x^{(t)}) = N(x^{(t)}, D)$ for a given value of D. Let $f(x)$ be density of the $N(\mu, \Sigma)$ distribution, here is the Metropolis-Hasting algorithm to achieve our goal:

Step 1: Start with an initial value $x^{(0)}$ and set $t = 0$

Step 2: Select a proposal density. Here we are going to use $g(\cdot|x^{(t)}) = N(x^{(t)}, D)$ with $D = \begin{pmatrix} \delta_1 & 0 \\ 0 & \delta_2 \end{pmatrix}$

Step 3: Generate a candidate value x^* from the proposal density $g(\cdot|x^{(t)})$

Step 4: Compute the Metropolis-Hastings ratio. Note that the proposal density is symmetric therefore the ratio is:

$$R = \frac{f(x^*)}{f(x^{(t)})}$$

Step 5: Set $x^{(t+1)} = \begin{cases} x^*, \text{ with probability } min(R, 1) \\ x^{(t)} \text{ otherwise} \end{cases}$

Step 6: Increment t and go to step 3 until reaching the number n of iteration

(b)

```
library(MASS)
library(emdbook)


f <- function(x){

    mu = c(1,2)
    Sigma = matrix(c(1,0.9,0.9,1),2,2)
    return(dmvnorm(x, mu, Sigma))
}

random_walk <- function(n, x0, D){

    x = matrix(0,nrow=n,ncol=2)
    x[1,] = x0
    accept = 0

    for(i in 2:n){

        xt = x[(i-1),]

        #step 3
        x_star = mvrnorm(1, xt, D)

        #step 4
        R = f(x_star) / f(xt)
```

```
        #step 5
        if (R >= 1 || (R < 1 & runif(1) < R)){
          x[i,] = x_star
          accept = accept+1

        }else x[i,] = xt
    }

    ratio = accept/n
    return(list(X=x, ratio=ratio))
}
```

(c)

```
n = 10000

delta1 = 0.001; delta2 = 0.001
D = matrix(c(delta1,0,0,delta2),2,2)
x0 = c(0,0)

list = random_walk(n, x0, D)
X = list$X
ratio = list$ratio
```

(i)

```
print(list$ratio)
```

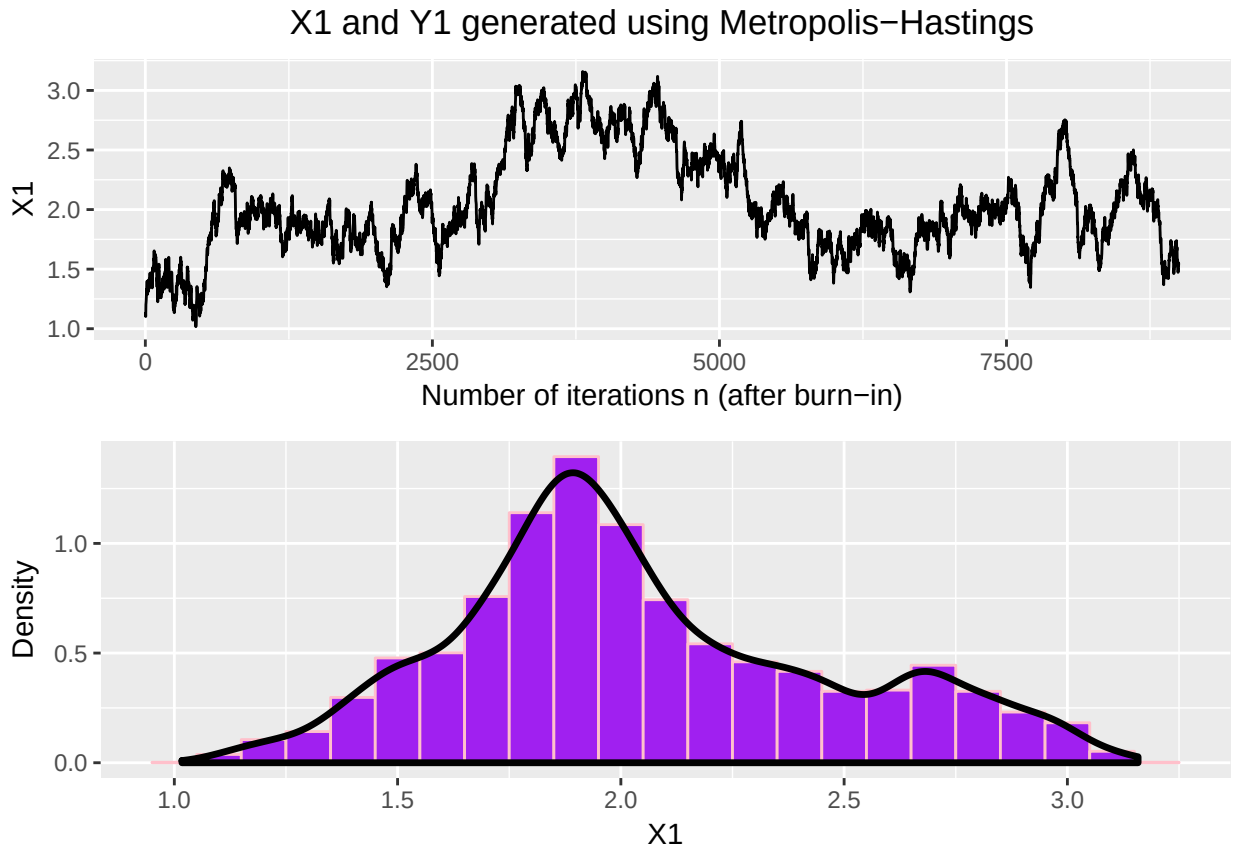
```
## [1] 0.9682
```

The acceptance rate is about 97% which is way too high, we should be accepting roughly between 25% and 50% of the values. The algorithm is accepting almost all of them so it moves by "baby steps" very slowly and probably won't get to the desired distribution (it might eventually get there but very slowly). However, let's investigate different plots of the generated values before coming to any definitive conclusion.

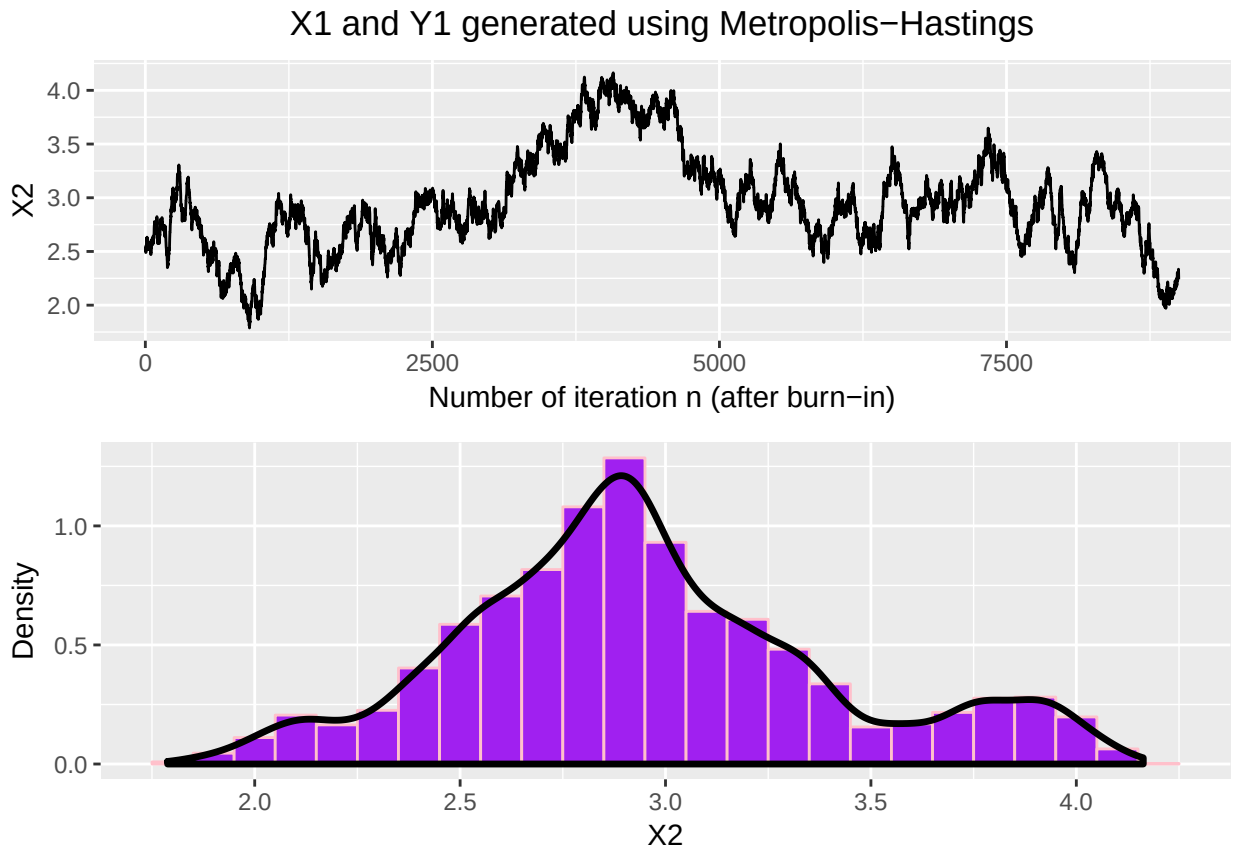
(ii)

```
#burn-in
burn = 1000
X1 = X[((burn+1):n),1]
X2 = X[((burn+1):n),2]

df = data.frame(v = seq(1,n-burn), X1, X2)
p1 <- ggplot(data=df, aes(x=v, y=X1))+
    geom_line()+
    labs(x="Number of iterations n (after burn-in)",
         title = "X1 and Y1 generated using Metropolis-Hastings")+
    theme(plot.title = element_text(hjust = 0.5))

p2 <- ggplot(data=df, aes(x=v, y=X2))+
```

```
    geom_line()+
    labs(x="Number of iteration n (after burn-in)",
         title = "X1 and Y1 generated using Metropolis-Hastings")+
    theme(plot.title = element_text(hjust = 0.5))

p3 <- ggplot(data=df, aes(x=X1))+
    geom_histogram(aes(y = ..density..), fill = "purple", color="pink", binwidth=.1)+
    geom_density(lwd = 1.2)+
    labs(y="Density")

p4 <- ggplot(data=df, aes(x=X2))+
    geom_histogram(aes(y = ..density..), fill = "purple", color="pink", binwidth=.1)+
    geom_density(lwd = 1.2)+
    labs(y="Density")

grid.arrange(p1,p3, nrow=2)
```



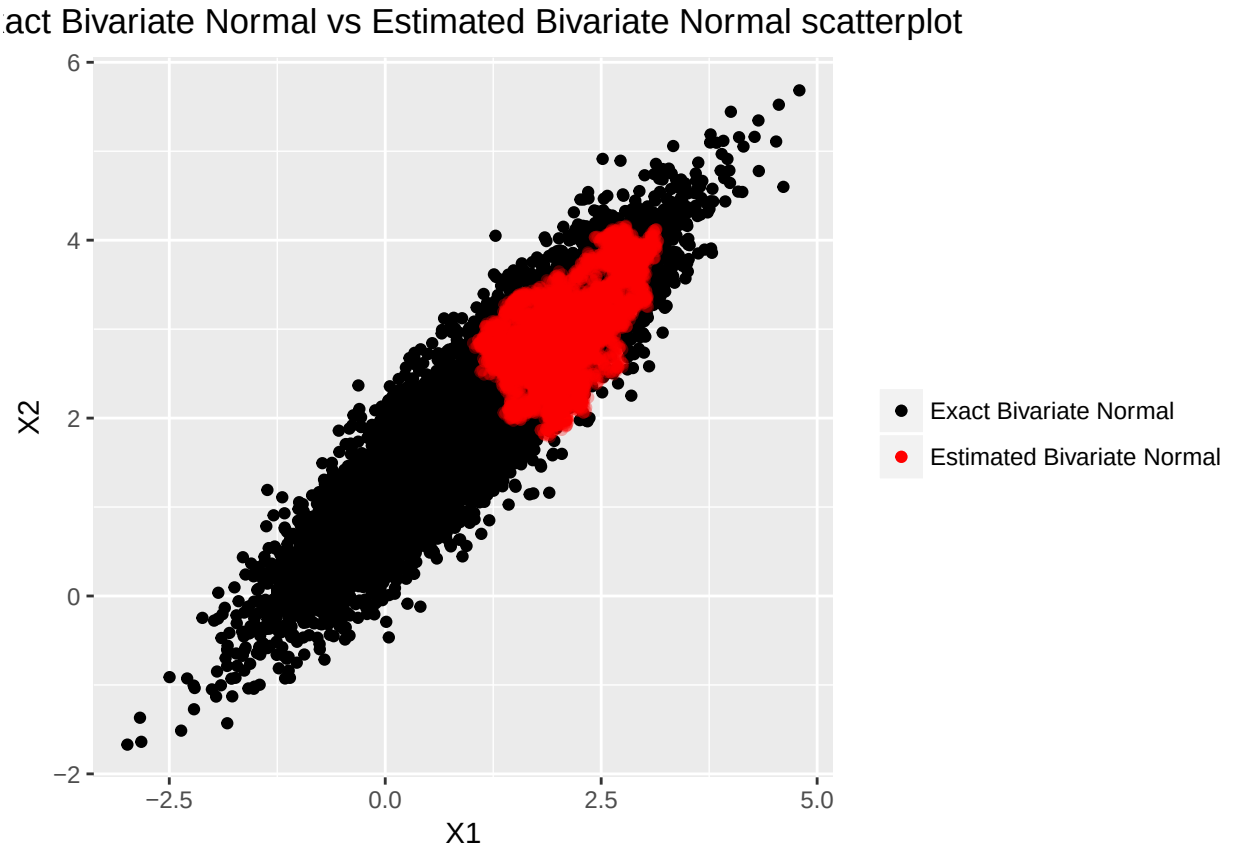

```
grid.arrange(p2,p4, nrow=2)
```



We can see in that in both traceplots, the chain gets stuck in certain areas or the parameter space, which indicates bad mixing. Moreover, both density plots do not look normal at all. We are aiming for a bivariate normal density so both generated X_1 and X_2 should look normal so it looks like we missed the target distribution.

(iii)

```
mu = c(1,2)
Sigma = matrix(c(1,0.9,0.9,1),2,2)
Y = mvrnorm(n-burn, mu, Sigma)

df = data.frame(Y, X1, X2)
 ggplot(data=df, aes(x=Y[,1], y=Y[,2]))+
    geom_point(aes(color = "a"))+
    geom_point(aes(x=X1, y=X2, color="b"), alpha=0.3)+
    labs(title = "Exact Bivariate Normal vs Estimated Bivariate Normal scatterplot",
         x="X1", y="X2")+
    theme(plot.title = element_text(hjust = 0.5))+
    scale_color_manual("", values = c("black", "red"),
                       labels = c("Exact Bivariate Normal", "Estimated Bivariate Normal"))
```



We can clearly see that the estimated distribution (red points) don't match the target distribution (black points). This is because δ_1 and δ_2 are too small. Indeed, those two parameters control the spread of the values generated, and those paramters being small makes this spread small. So during the algorithm, x^* is never going to be any far from $x^{(t)}$ wich will make R big (because $f(x^*)$ and $f(x^{(t)})$ will be very close). Thus the algorithm accepts too many of those x^* values.

(d)

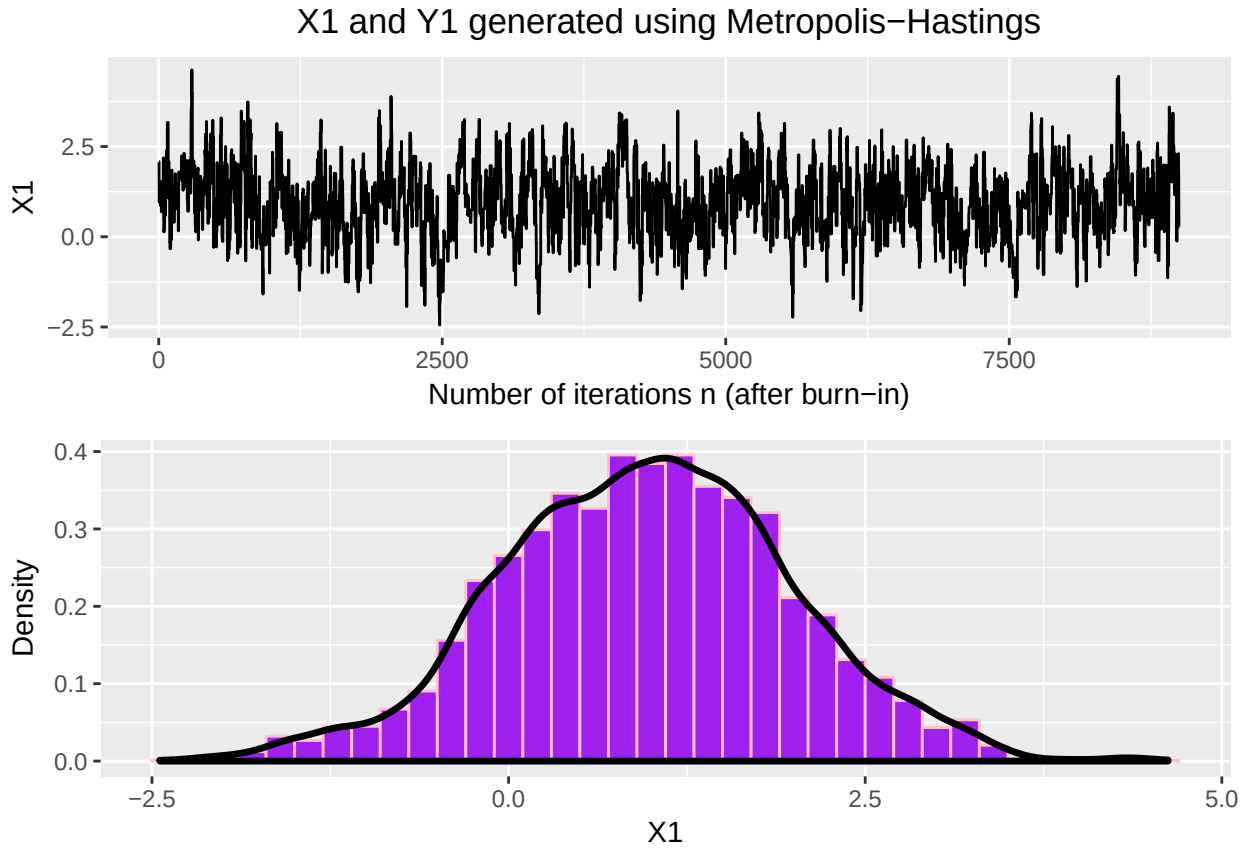
We choose $\delta_1 = \delta_2 = 1$ because the target distribution as a similar spread ($\Sigma_{11} = \Sigma 22 = 1$).

```
delta1 = 1; delta2 = 1
D = matrix(c(delta1,0,0,delta2),2,2)
x0 = c(0,0)
n = 10000

list = random_walk(n, x0, D)
X = list$X
ratio = list$ratio
```

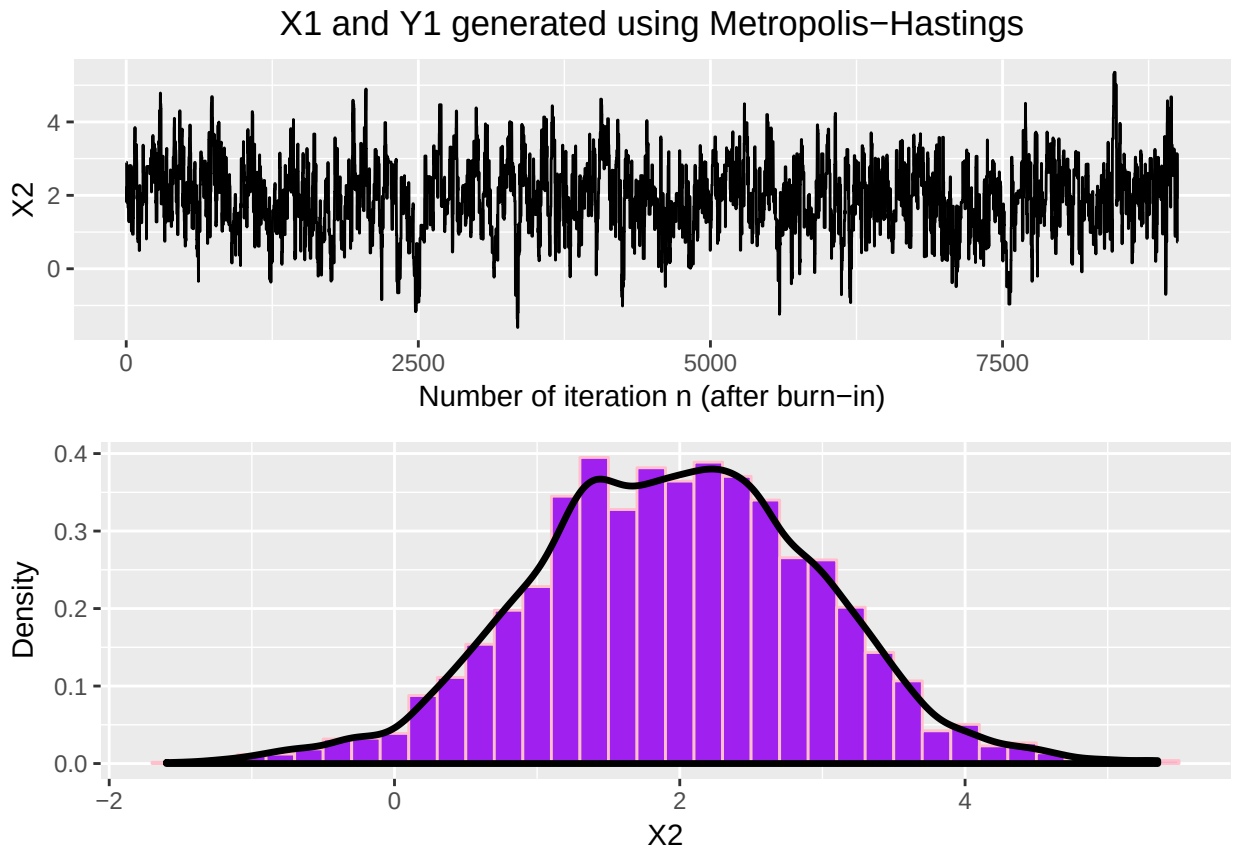
(ii)

```
#burn-in
burn = 1000
X1 = X[((burn+1):n),1]
X2 = X[((burn+1):n),2]
```

```r
df = data.frame(v = seq(1,n-burn), X1, X2)

p1 <- ggplot(data=df, aes(x=v, y=X1))+
    geom_line()+
    labs(x="Number of iterations n (after burn-in)",
         title = "X1 and Y1 generated using Metropolis-Hastings")+
    theme(plot.title = element_text(hjust = 0.5))

p2 <- ggplot(data=df, aes(x=v, y=X2))+
    geom_line()+
    labs(x="Number of iteration n (after burn-in)",
         title = "X1 and Y1 generated using Metropolis-Hastings")+
    theme(plot.title = element_text(hjust = 0.5))

p3 <- ggplot(data=df, aes(x=X1))+
    geom_histogram(aes(y = ..density..), fill = "purple", color="pink", binwidth=.2)+
    geom_density(lwd = 1.2)+
    labs(y="Density")

p4 <- ggplot(data=df, aes(x=X2))+
    geom_histogram(aes(y = ..density..), fill = "purple", color="pink", binwidth=.2)+
    geom_density(lwd = 1.2)+
    labs(y="Density")

grid.arrange(p1,p3, nrow=2)
```



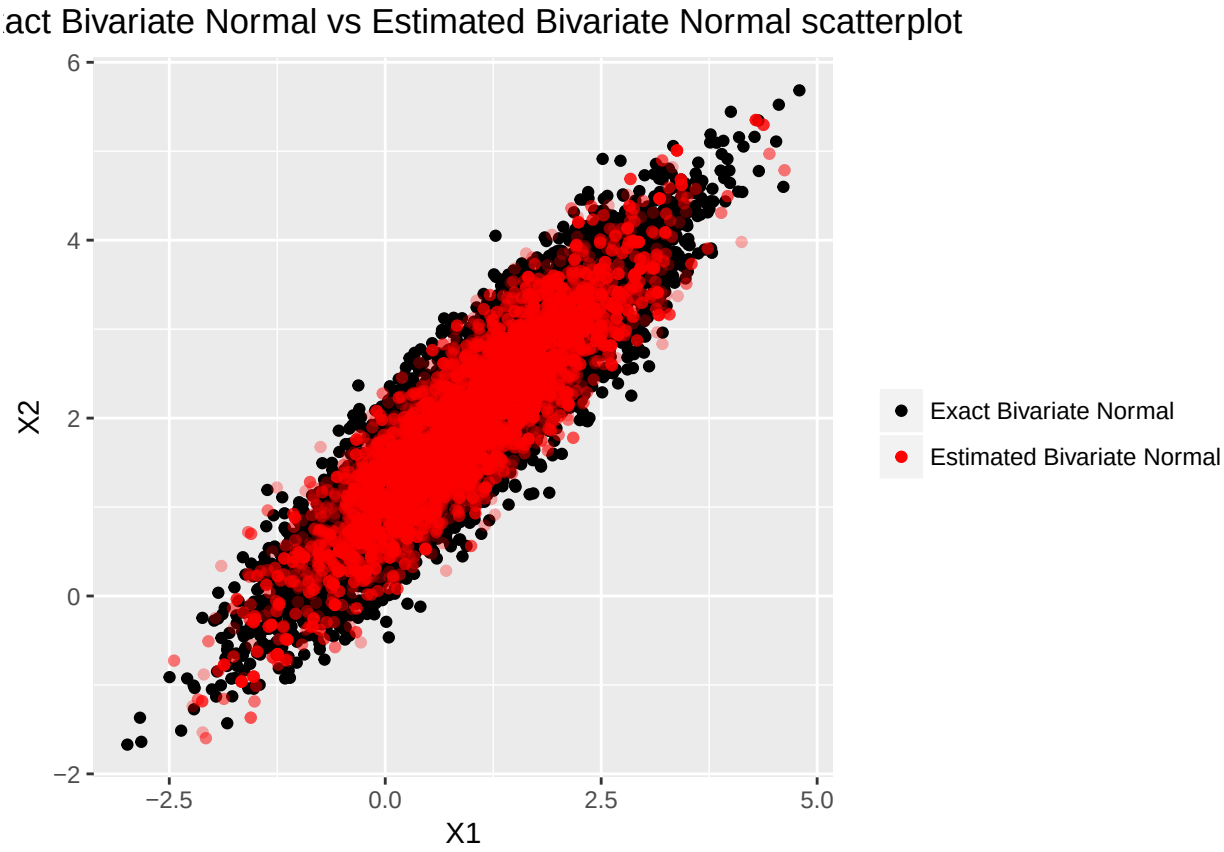
```
grid.arrange(p2,p4, nrow=2)
```



In both traceplots, we can see that our chains visit every point of the parameter space, they do not get stuck anywhere. This is a sign of good mixing. Moreover, both density plots look normal, which is what we are aiming for, so those plots are good as well.

(iii)

```
df = data.frame(Y, X1, X2)
 ggplot(data=df, aes(x=Y[,1], y=Y[,2]))+
    geom_point(aes(color = "a"))+
    geom_point(aes(x=X1, y=X2, color="b"), alpha=0.3)+
    labs(title = "Exact Bivariate Normal vs Estimated Bivariate Normal scatterplot",
         x="X1", y="X2")+
    theme(plot.title = element_text(hjust = 0.5))+
    scale_color_manual("", values = c("black", "red"),
                       labels = c("Exact Bivariate Normal", "Estimated Bivariate Normal"))
```



We now clearly meet all the target requirement, the distribution is the same as the target.

(e)

If I choose δ_1 and δ_2 to be very large, I expect the acceptance rate to be very small. As explained before, as those parameter control the spread the the values generated a each iteration, the generated value x^* can end up very far off the heavy part of the density and thus $f(x^*)$ would be way smaller than $f(x^{(t)})$ and thus R would be very small in most cases and thus we would accept generated values (with probability $min(R, 1)$) rarely.