MTH422A_assigment_3_Solution

Vijay Soren (211163)

2024-03-11

1

(a)

```
County <- 1:10
Approve <- c(12,90,80,5,63,15,67,22,56,33)
Disapprove <- c(50,150,63,10,63,8,56,19,63,19)
data <- data.frame(County, Approve,Disapprove)
samples_pro <- data$Approve/(data$Approve + data$Disapprove)
mean_samples_pro <- mean(samples_pro)
var_samples_pro <- var(samples_pro)

###a
Y <- data$Approve
n_is <- data$Approve + data$Disapprove
L <- qbeta(c(0.025),shape1 = Y + 0.5, shape2 = n_is - Y + 0.5)
U <- qbeta(c(0.975),shape1 = Y + 0.5, shape2 = n_is - Y + 0.5)
CI95 <- cbind(L,U)
CI95_data <- data.frame(data$County,CI95)
print("95% credibles interval")
```

```
## [1] "95% credibles interval"
```

```
print(CI95_data)
```

```
##    data.County         L         U
## 1            1 0.1103935 0.3045307
## 2            2 0.3155448 0.4374498
## 3            3 0.4775597 0.6389423
## 4            4 0.1402526 0.5841638
## 5            5 0.4135270 0.5864730
## 6            6 0.4488766 0.8197773
## 7            7 0.4565230 0.6308309
## 8            8 0.3858303 0.6823739
## 9            9 0.3825161 0.5600718
## 10          10 0.4993247 0.7553816
```

(b)

I have found $a = 5.5104$ and $b = 5.9696$

(c)

```
c <- 0.480
c_star <- c/(1-c)

d <- (c_star + 1)^2
e <- 0.02/c_star
f <- 1/(e*d) - 1
b <- f/(c_star + 1)
a <- c_star*b



###part c
Y <- data$Approve
n_is <- data$Approve + data$Disapprove
L <- qbeta(c(0.025),shape1 = Y + a, shape2 = n_is - Y + b)
U <- qbeta(c(0.975),shape1 = Y + a, shape2 = n_is - Y + b)
CI95_EB <- cbind(L,U)
CI95_data_EB <- data.frame(data$County,CI95_EB)
print("95% credibles interval when beta prior")
```

```
## [1] "95% credibles interval when beta prior"
```

```
print(CI95_data_EB)
```

```
##    data.County         L         U
## 1            1 0.1487391 0.3413448
## 2            2 0.3208890 0.4405087
## 3            3 0.4748497 0.6309109
## 4            4 0.2230410 0.5855504
## 5            5 0.4152060 0.5814999
## 6            6 0.4292059 0.7500640
## 7            7 0.4548387 0.6224387
## 8            8 0.3900460 0.6566197
## 9            9 0.3866469 0.5570160
## 10          10 0.4846772 0.7222625
```

(d)

both part results from (a) and (d) quite similar and all samples proportion's lies within their 95% credibles interval

2

(a)

$\mu_{map} = \sum_{i=1}^{n} \frac{Y_i}{\sigma_i^2} / \sum_{i=1}^{n} \frac{1}{\sigma_i^2}$

(b)

```
n <- 3

Map_est <- function(Y,sigma_s)
{
  upper <- sum(Y/sigma_s^2)
  lower <- sum(1/sigma_s^2)
  mu_hat <- upper/lower
  return(mu_hat)

}


###part b
n = 3
Y <- c(12,10,22)    # y1 = 12, y2 = 10, y3 = 22

sigma_s <- c(3,3,10)
mu_hat_map <- Map_est(Y = Y, sigma_s = sigma_s)
print((mu_hat_map))
```

```
## [1] 11.47368
```

(c)

```
set.seed(2)
n <- 1000
mu <- 5
sigma.s <- sample(1:40,n, replace = TRUE) ## random genration of sigma_s bw 1 :40 of length n
Y <- rnorm(n, mean = mu, sd = sigma.s)
mu_samples_pos <- rnorm(n, mean =  sum(Y/sigma.s^2)/(sum(1/sigma.s^2)), sd = sqrt(1/sum(1/sigma.s^2)))
 ## posterior mean
mean(mu_samples_pos)
```

```
## [1] 4.847362
```

```
print("mean of posterior")
```

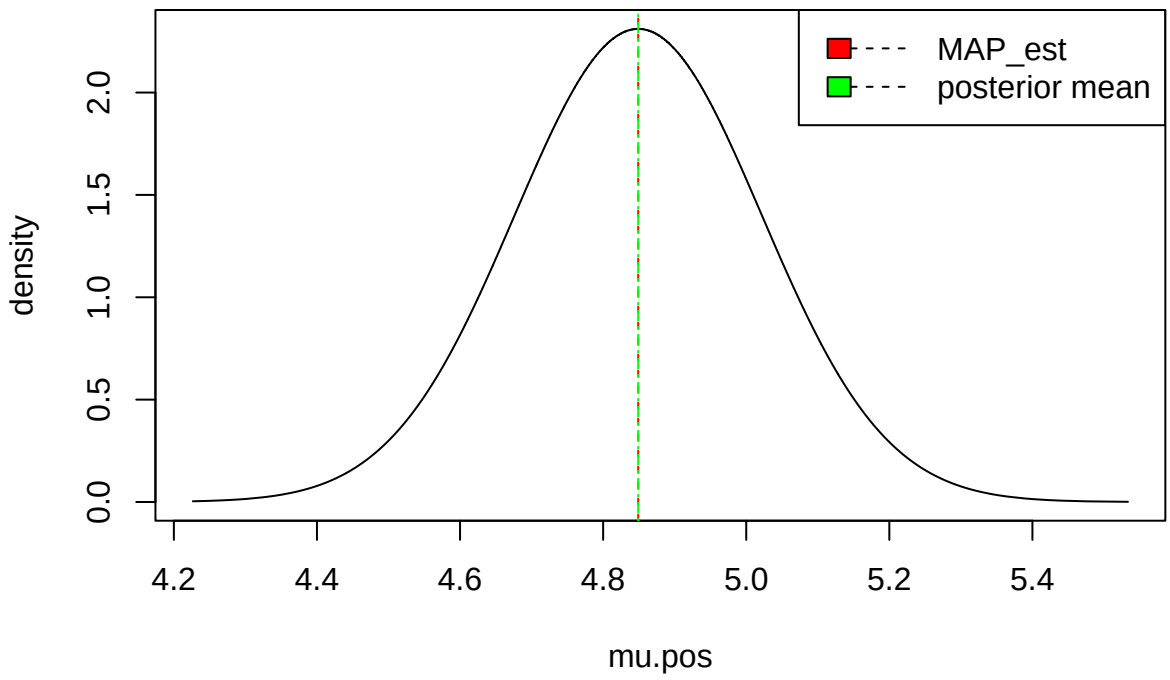
```
## [1] "mean of posterior"
```

```
print(mean(mu_samples_pos))
```

```
## [1] 4.847362
```

(d)

```r
set.seed(2)
n <- 1000
mu <- 5
sigma.s <- sample(1:40,n, replace = TRUE) ## random genration of sigma_s bw 1 :40 of length n
Y <- rnorm(n, mean = mu, sd = sigma.s)

## posterior distribution
mu.pos.var <- 1/(sum(1/sigma.s^2))
mu.pos.mean <- sum(Y/sigma.s^2)/sum(1/sigma.s^2)

MAP_mu <- sum(Y/sigma.s^2)/sum(1/sigma.s^2)

mu.pos.samples <- rnorm(1e4, mean = mu.pos.mean, sd = sqrt(mu.pos.var))
mu.pos <- sort(mu.pos.samples)
density <- dnorm(mu.pos, mean = mu.pos.mean, sd = sqrt(mu.pos.var))
plot(mu.pos, density, type = "l")
abline(v = MAP_mu , col = "red", lty = 2)
abline(v = mu.pos.mean , col = "green", lty = 6)
legend("topright", c("MAP_est","posterior mean"), fill = c("red","green"), lty = 2)
```



3

(a)

The full conditional posterior distributions for σ_1^2 is $Inverse - Gamma(a + 0.5, Y_1^2/2 + b)$

The full conditional posterior distributions for b is $Gamma(1, 1 + \sum_{i=1}^{n} \frac{1}{\sigma_i^2})$

(b) psudocode

```
set.seed(1)
library(invgamma)
n <- 10
a <- 10
b <- rgamma(1,shape = 1,rate = 1)
sigmaSq <- rinvgamma(n,shape = a, rate = b)

Y <- rnorm(n, mean = 0, sd = sqrt(sigmaSq))



b.update  <- function(sigmaSq)
{
  b.pos <- rgamma(1,shape = 1, rate = 1 + sum(1/sigmaSq))
  return(b.pos)
}

sigmaSq.update <- function(Y, a, b,n)
{
  sigmaSq<- rinvgamma(n, shape = a + 1/2, rate = b + 0.5*sum(Y^2))

  return(sigmaSq)
}

MCMC <- function(Y, b.init, sigmaSq.init, iters){

  # chain initiation
  b <- b.init
  n <- length(Y)
  sigmaSq <- rep(sigmaSq.init, n)

  # define chains
  b.chain <- rep(NA, iters)
  sigmaSq.chain <- matrix(NA, iters, n)

  # start MCMC
  for(i in 1:iters){
    sigmaSq <- sigmaSq.update(Y, a, b,n)
    b <- b.update(sigmaSq)


    b.chain[i] <- b
```

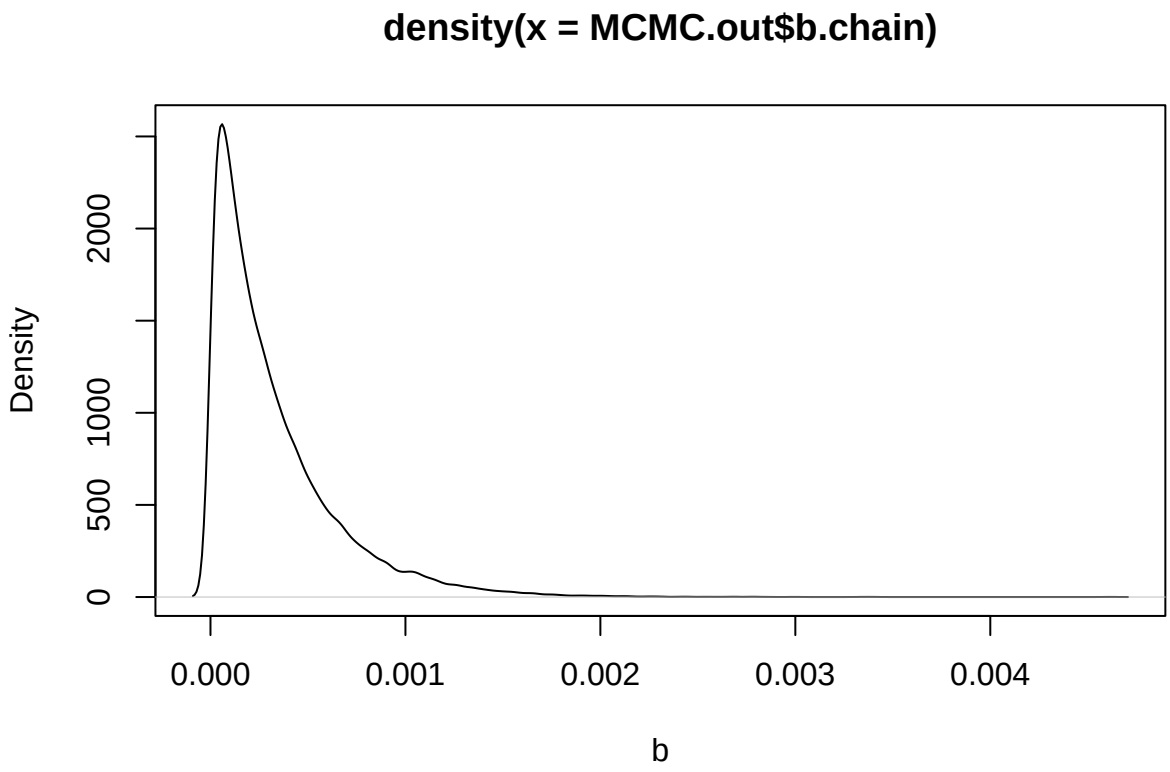
```
    sigmaSq.chain[i,] <- sigmaSq

  }

  # return chains
  out <- list(b.chain = b.chain,
              sigmaSq.chain = sigmaSq.chain)
  return(out)}
```

(c)

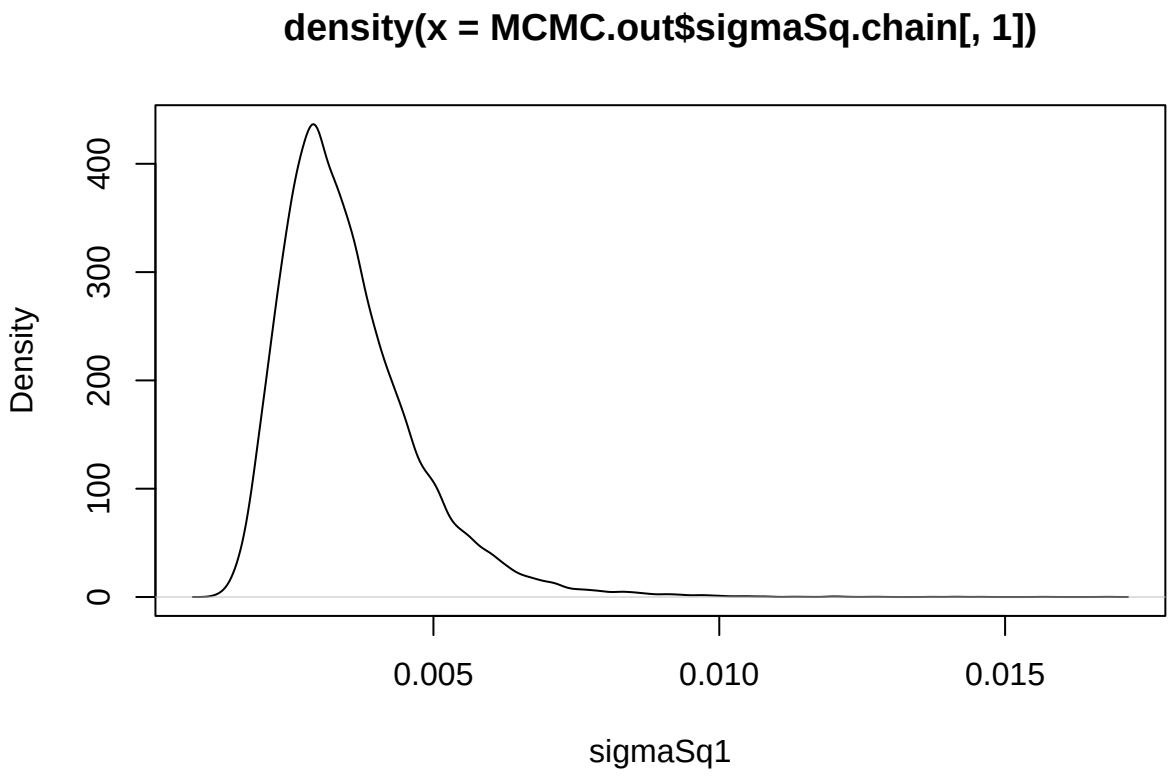
a = 10

```
set.seed(1)
library(invgamma)
n <- 10
a <- 10
b <- rgamma(1,shape = 1,rate = 1)
sigmaSq <- rinvgamma(n,shape = a, rate = b)

Y <- rnorm(n, mean = 0, sd = sqrt(sigmaSq))



b.update  <- function(sigmaSq)
{
  b.pos <- rgamma(1,shape = 1, rate = 1 + sum(1/sigmaSq))
  return(b.pos)
}

sigmaSq.update <- function(Y, a, b,n)
{
  sigmaSq<- rinvgamma(n, shape = a + 1/2, rate = b + 0.5*sum(Y^2))

  return(sigmaSq)
}

MCMC <- function(Y, b.init, sigmaSq.init, iters){

  # chain initiation
  b <- b.init
  n <- length(Y)
  sigmaSq <- rep(sigmaSq.init, n)

  # define chains
  b.chain <- rep(NA, iters)
  sigmaSq.chain <- matrix(NA, iters, n)

  # start MCMC
  for(i in 1:iters){
    sigmaSq <- sigmaSq.update(Y, a, b,n)
    b <- b.update(sigmaSq)
```

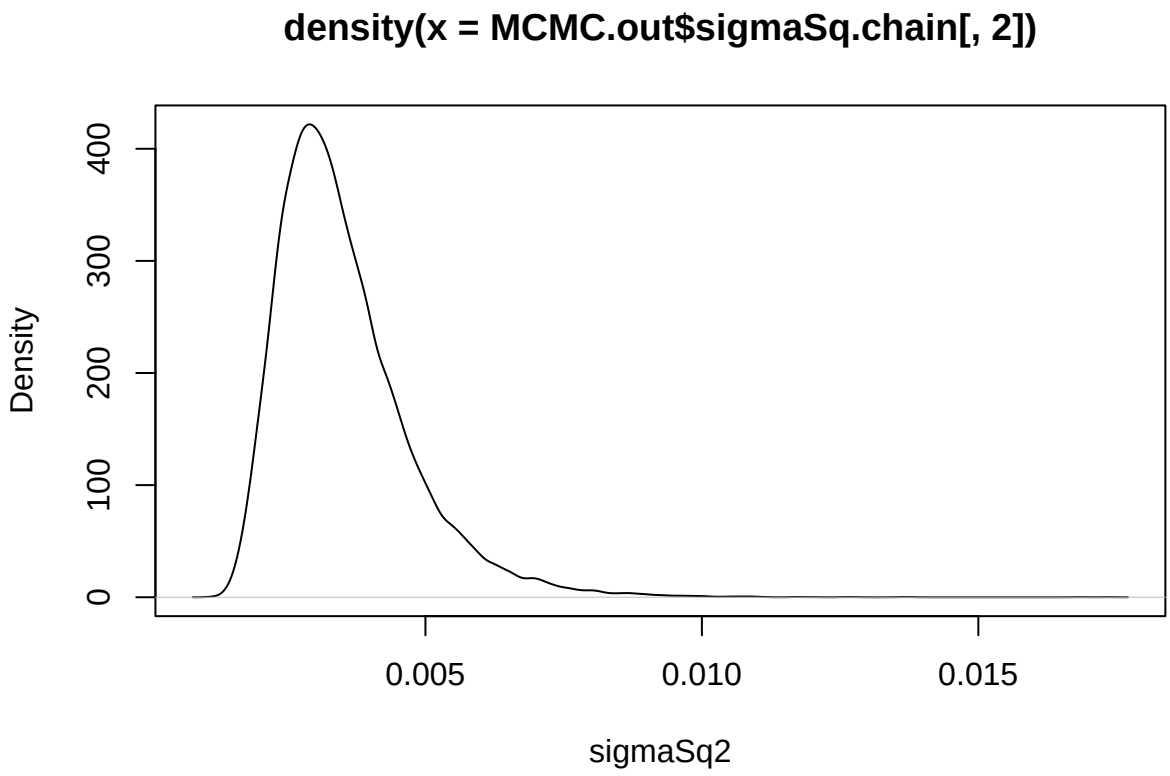
```
    b.chain[i] <- b
    sigmaSq.chain[i,] <- sigmaSq

  }

  # return chains
  out <- list(b.chain = b.chain,
              sigmaSq.chain = sigmaSq.chain)
  return(out)}


MCMC.out <- MCMC(Y = Y,
                 b.init = 0,
                 sigmaSq.init = .04,
                 iters = 30000)


# marginal plot
plot(density(MCMC.out$b.chain), xlab = expression(b))
```



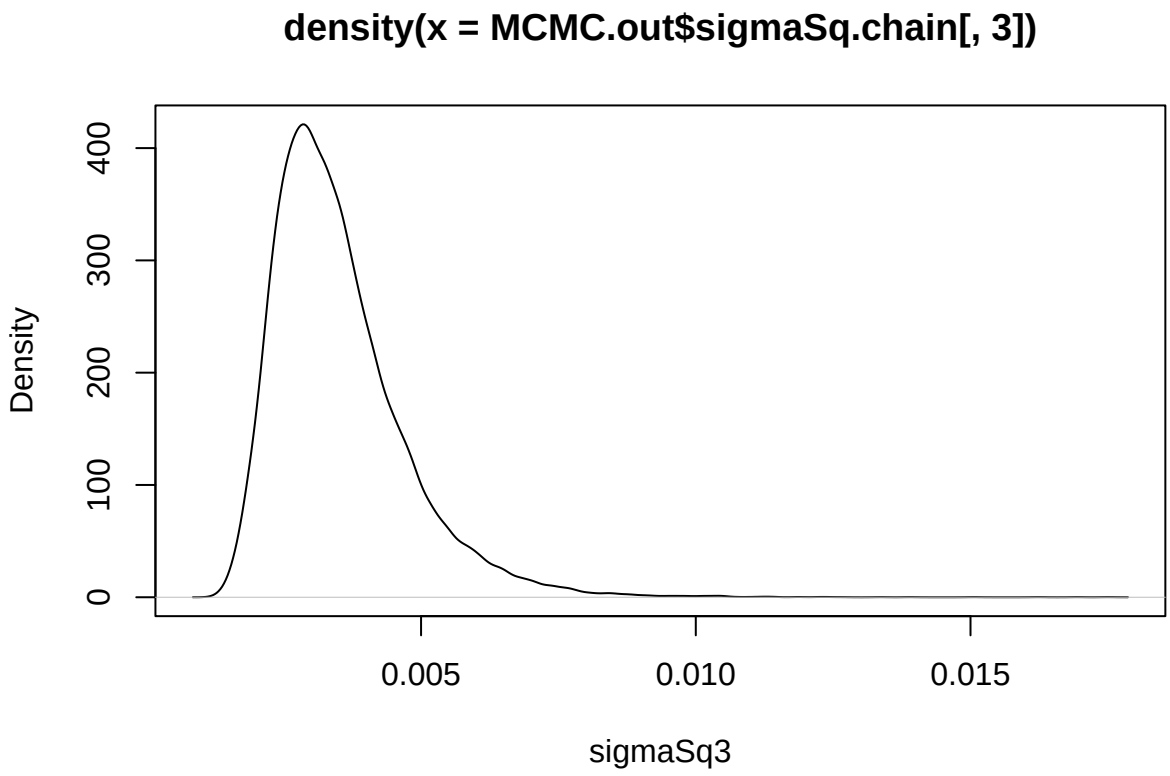
```
plot(density(MCMC.out$sigmaSq.chain[,1]),xlab = "sigmaSq1")
```



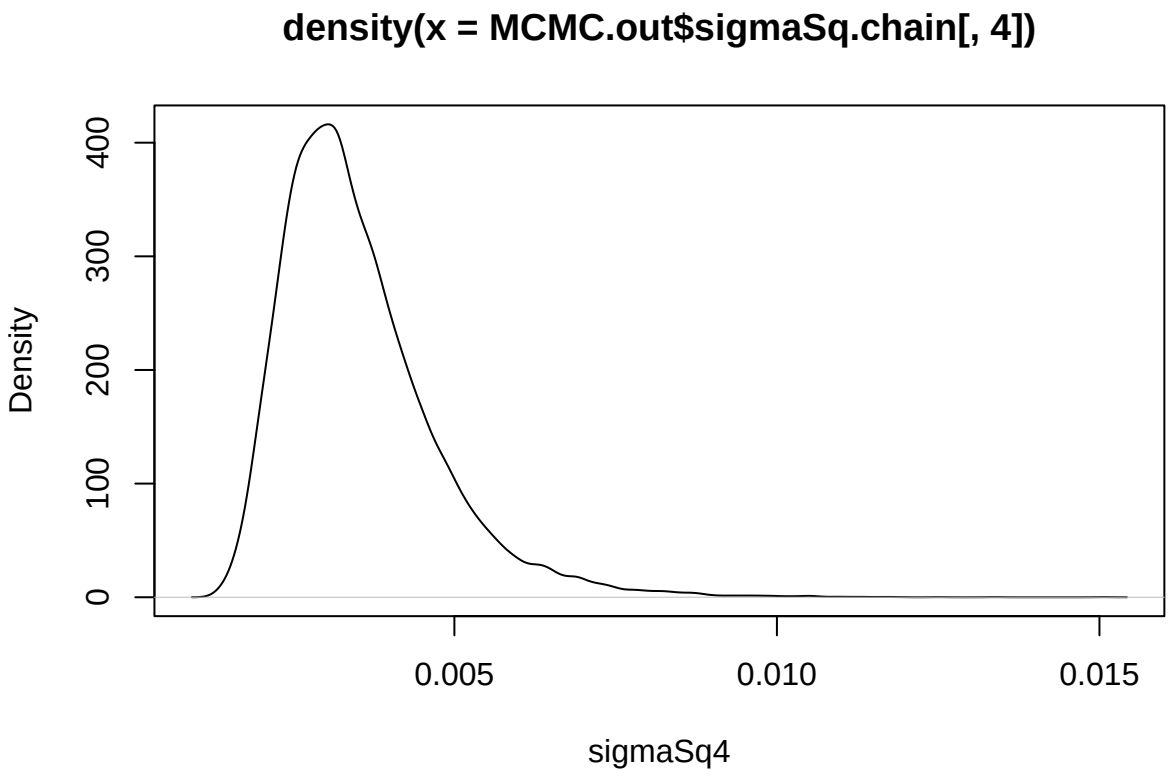

```
plot(density(MCMC.out$sigmaSq.chain[,2]),xlab = "sigmaSq2")
```



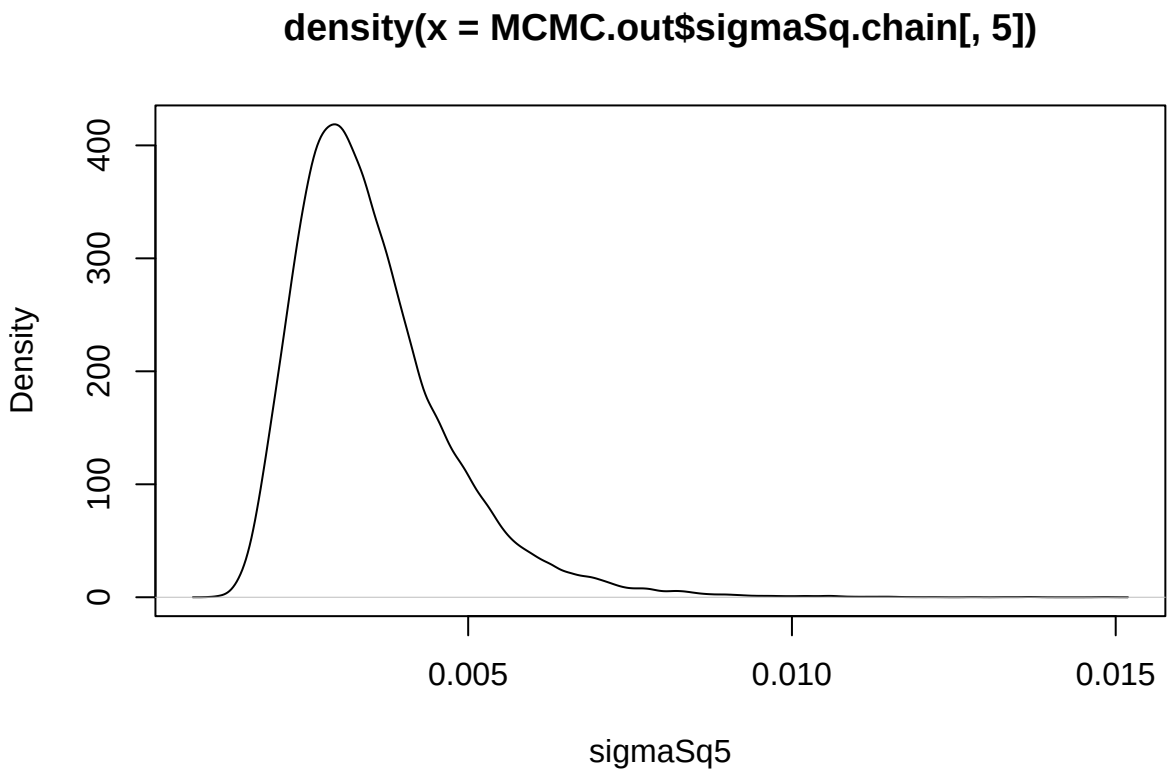

```
plot(density(MCMC.out$sigmaSq.chain[,3]),xlab = "sigmaSq3")
```



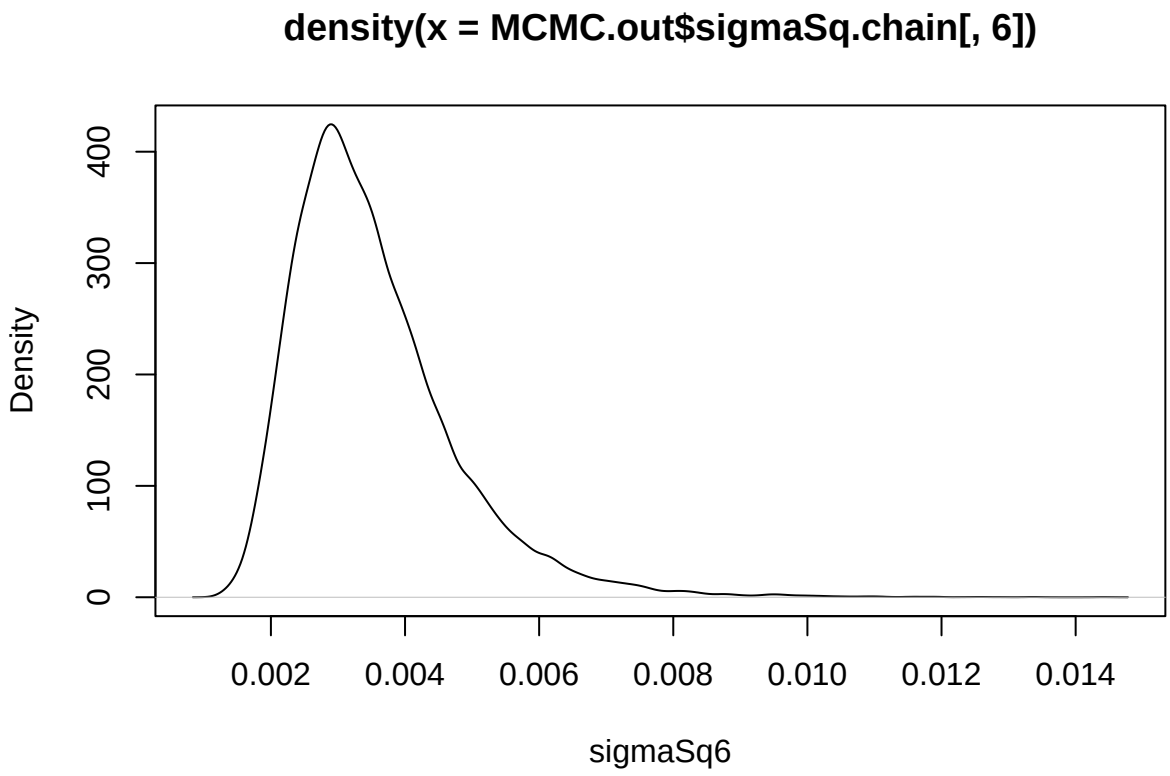

```
plot(density(MCMC.out$sigmaSq.chain[,4]),xlab = "sigmaSq4")
```



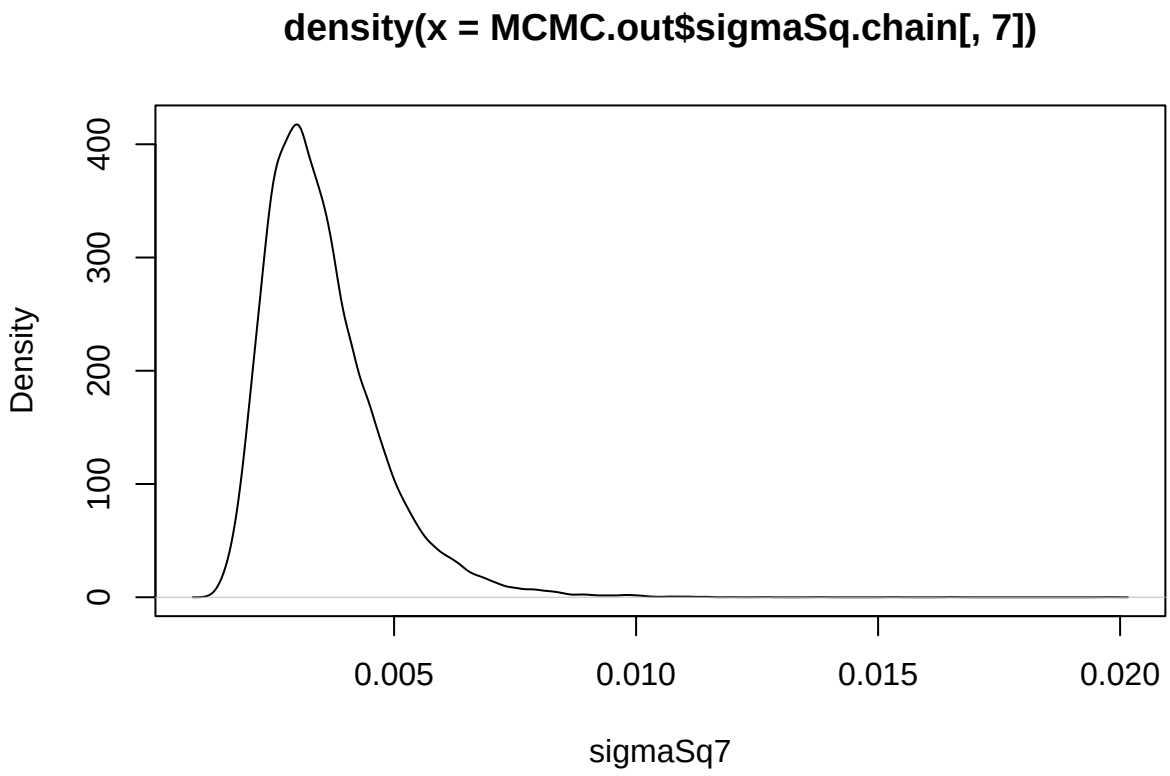

```
plot(density(MCMC.out$sigmaSq.chain[,5]),xlab = "sigmaSq5")
```



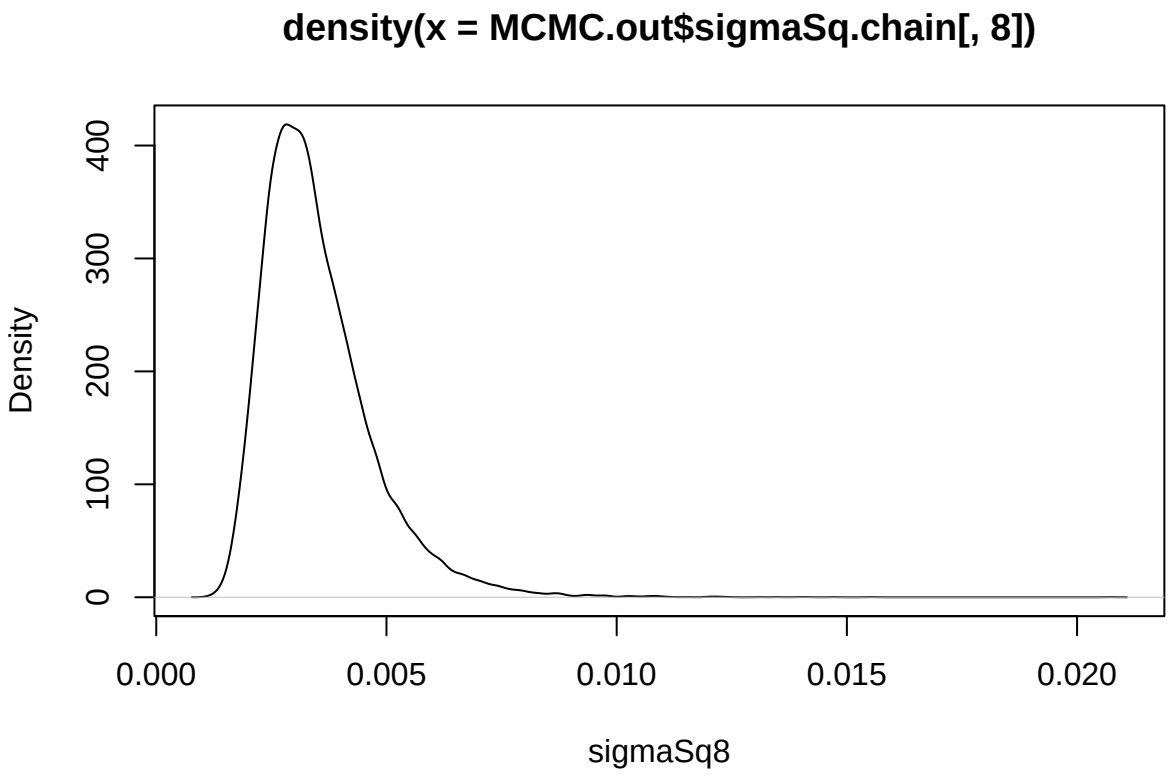

```
plot(density(MCMC.out$sigmaSq.chain[,6]),xlab = "sigmaSq6")
```



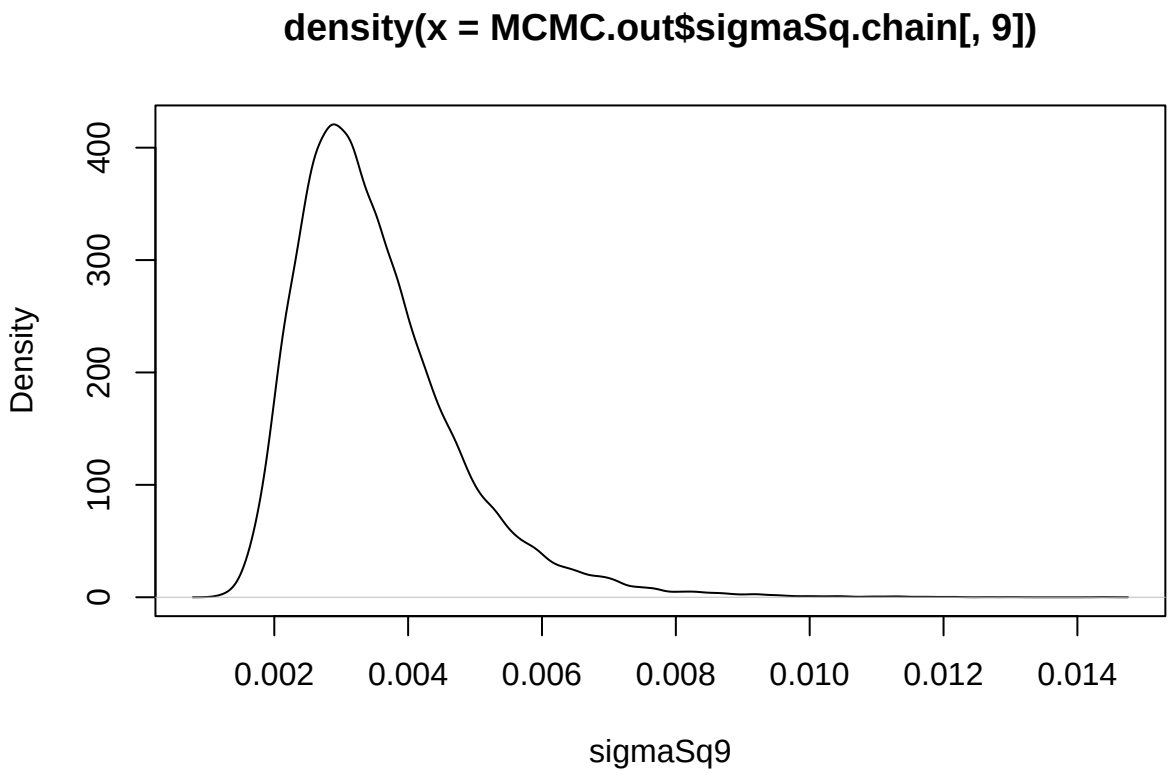

```
plot(density(MCMC.out$sigmaSq.chain[,7]),xlab = "sigmaSq7")
```



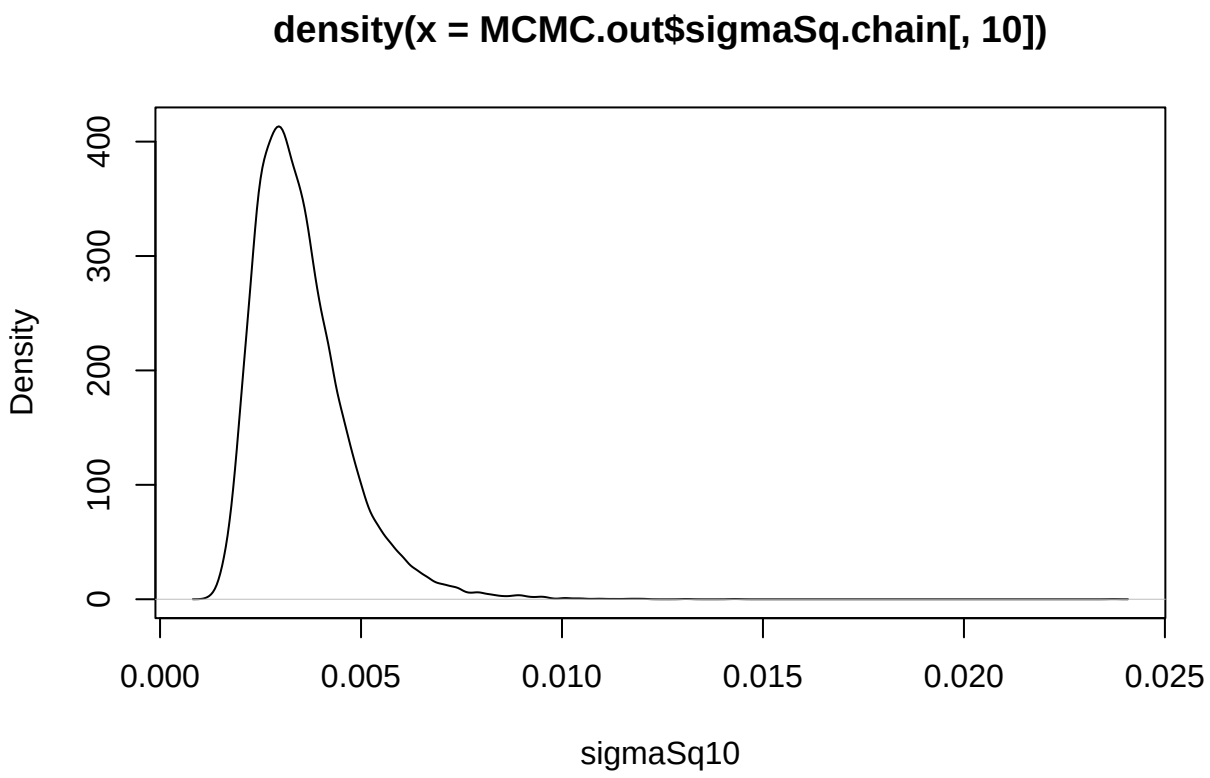

```
plot(density(MCMC.out$sigmaSq.chain[,8]),xlab = "sigmaSq8")
```




```
plot(density(MCMC.out$sigmaSq.chain[,9]),xlab = "sigmaSq9")
```




```
plot(density(MCMC.out$sigmaSq.chain[,10]),xlab = "sigmaSq10")
```

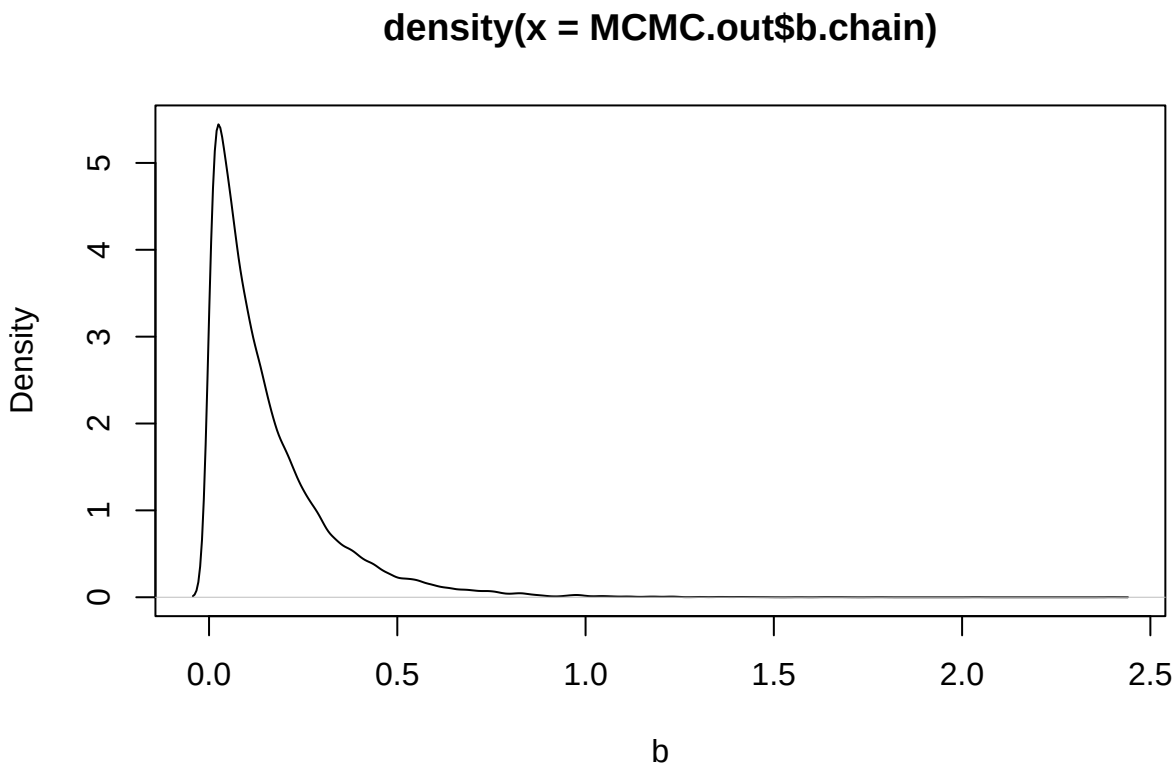


(d)

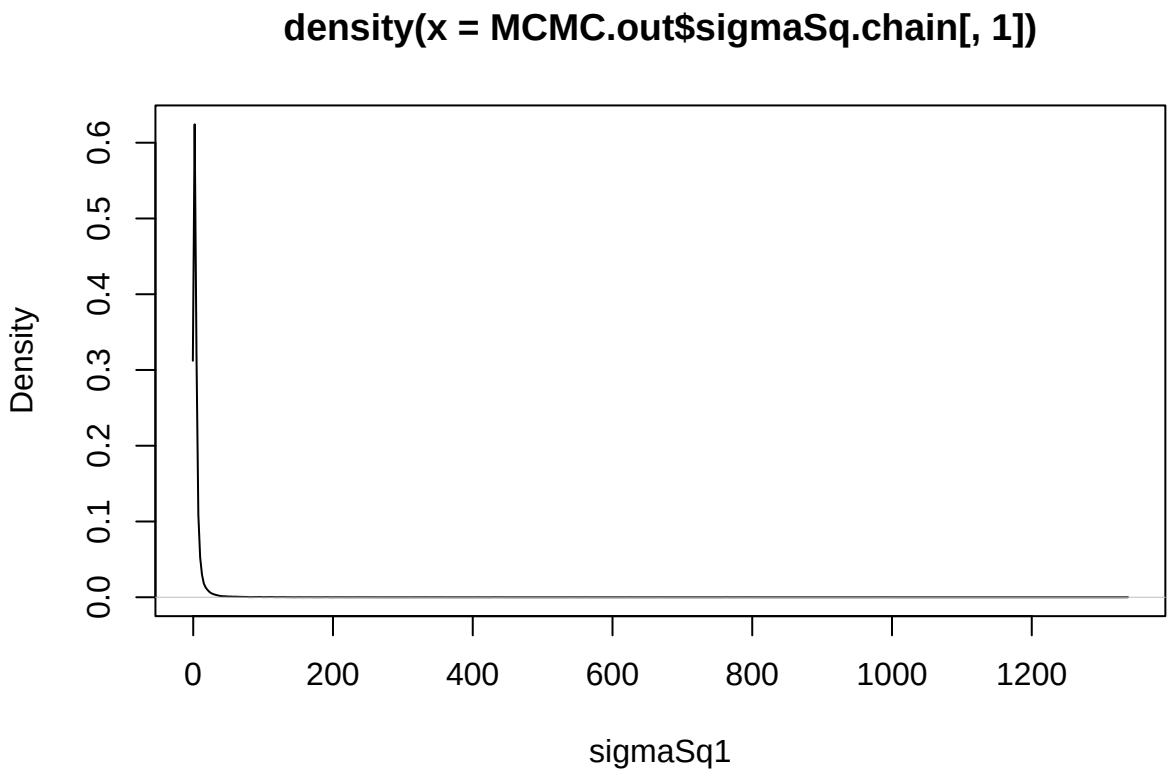
a =1

```
set.seed(1)
library(invgamma)
n <- 10
a <- 1
b <- rgamma(1,shape = 1,rate = 1)
sigmaSq <- rinvgamma(n,shape = a, rate = b)

Y <- rnorm(n, mean = 0, sd = sqrt(sigmaSq))



b.update  <- function(sigmaSq)
{
  b.pos <- rgamma(1,shape = 1, rate = 1 + sum(1/sigmaSq))
  return(b.pos)
}

sigmaSq.update <- function(Y, a, b,n)
{
  sigmaSq<- rinvgamma(n, shape = a + 1/2, rate = b + 0.5*sum(Y^2))
```

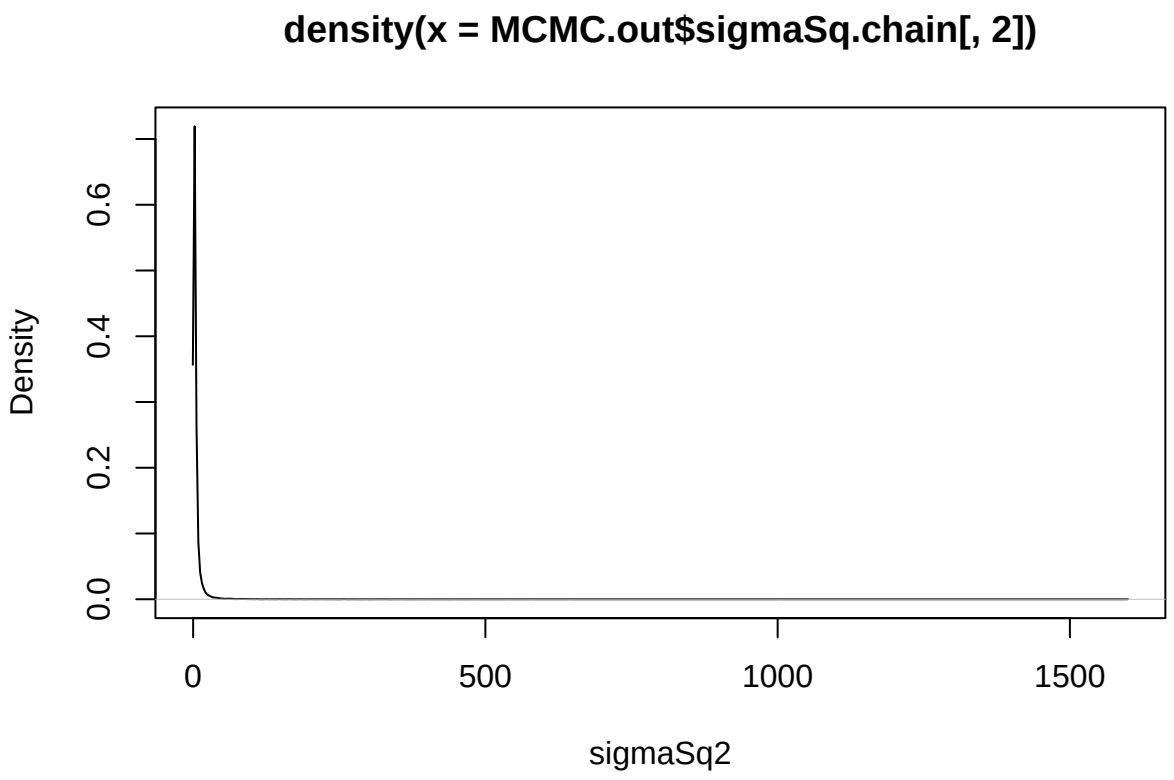
```
  return(sigmaSq)
}

MCMC <- function(Y, b.init, sigmaSq.init, iters){

  # chain initiation
  b <- b.init
  n <- length(Y)
  sigmaSq <- rep(sigmaSq.init, n)

  # define chains
  b.chain <- rep(NA, iters)
  sigmaSq.chain <- matrix(NA, iters, n)

  # start MCMC
  for(i in 1:iters){
    sigmaSq <- sigmaSq.update(Y, a, b,n)
    b <- b.update(sigmaSq)


    b.chain[i] <- b
    sigmaSq.chain[i,] <- sigmaSq

  }

  # return chains
  out <- list(b.chain = b.chain,
              sigmaSq.chain = sigmaSq.chain)
  return(out)}


MCMC.out <- MCMC(Y = Y,
                 b.init = 0,
                 sigmaSq.init = .04,
                 iters = 30000)


# marginal plot
plot(density(MCMC.out$b.chain), xlab = expression(b))
```



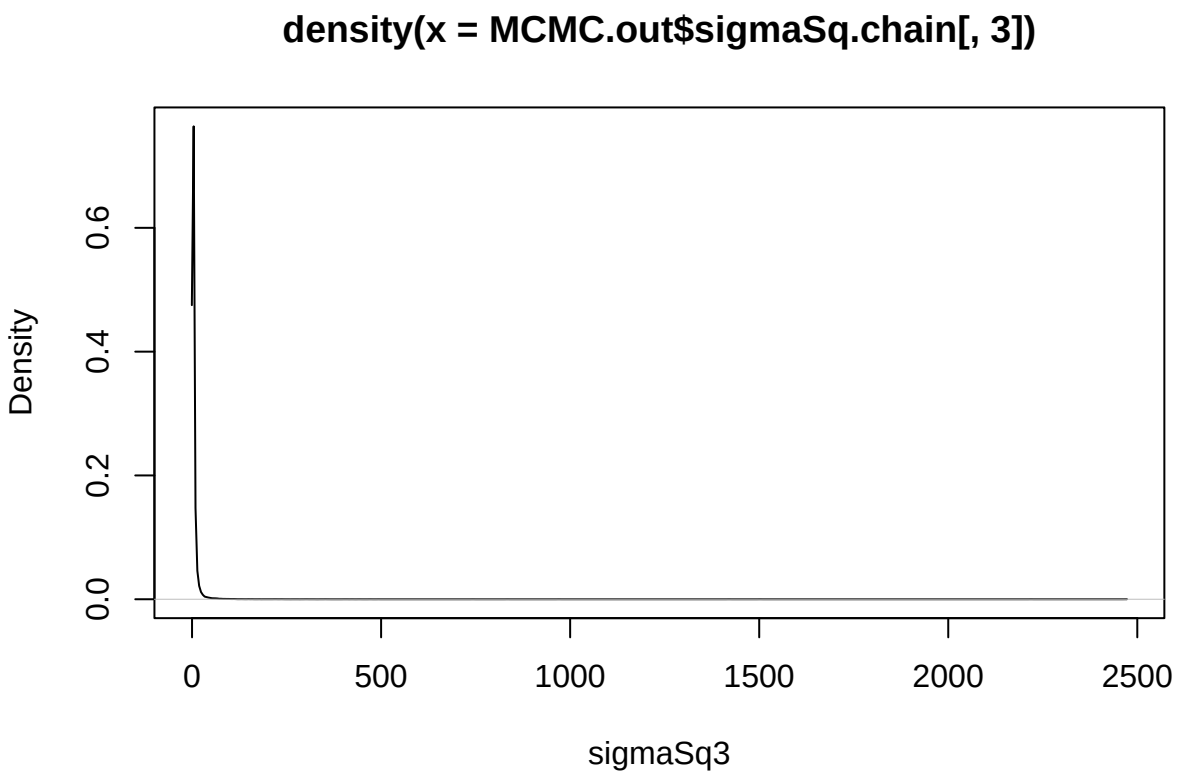

```
plot(density(MCMC.out$sigmaSq.chain[,1]),xlab = "sigmaSq1")
```



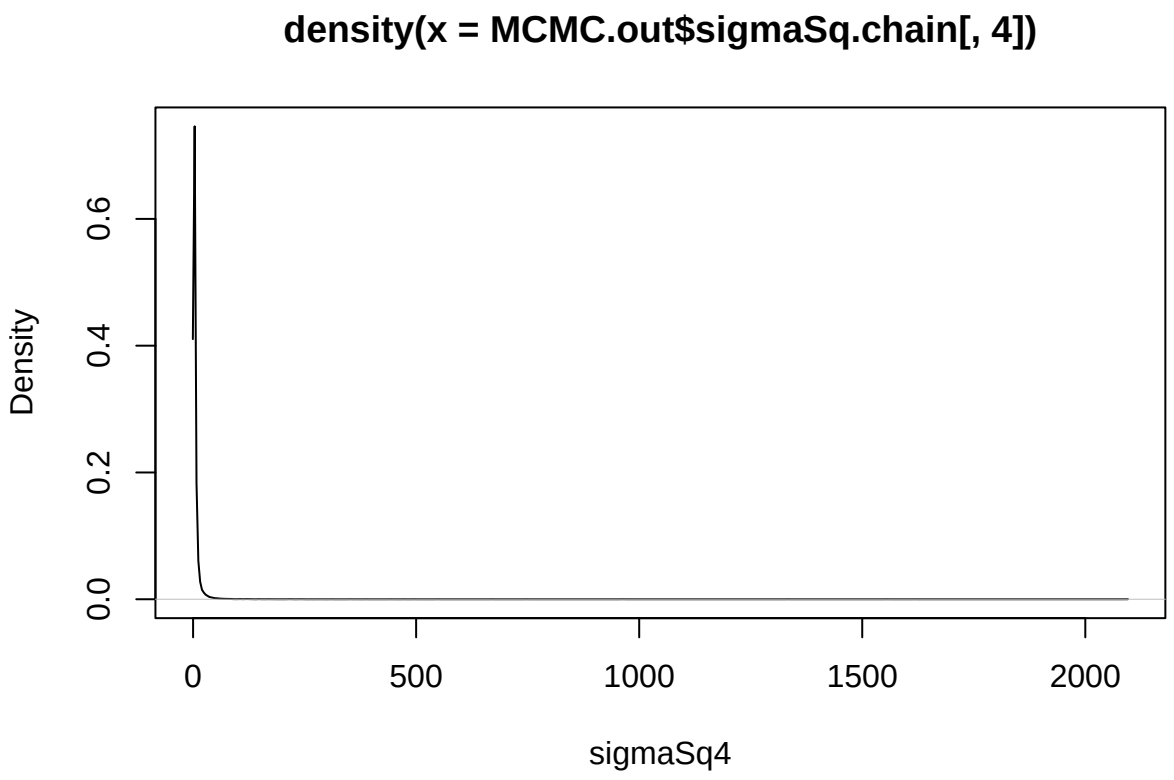

```
plot(density(MCMC.out$sigmaSq.chain[,2]),xlab = "sigmaSq2")
```



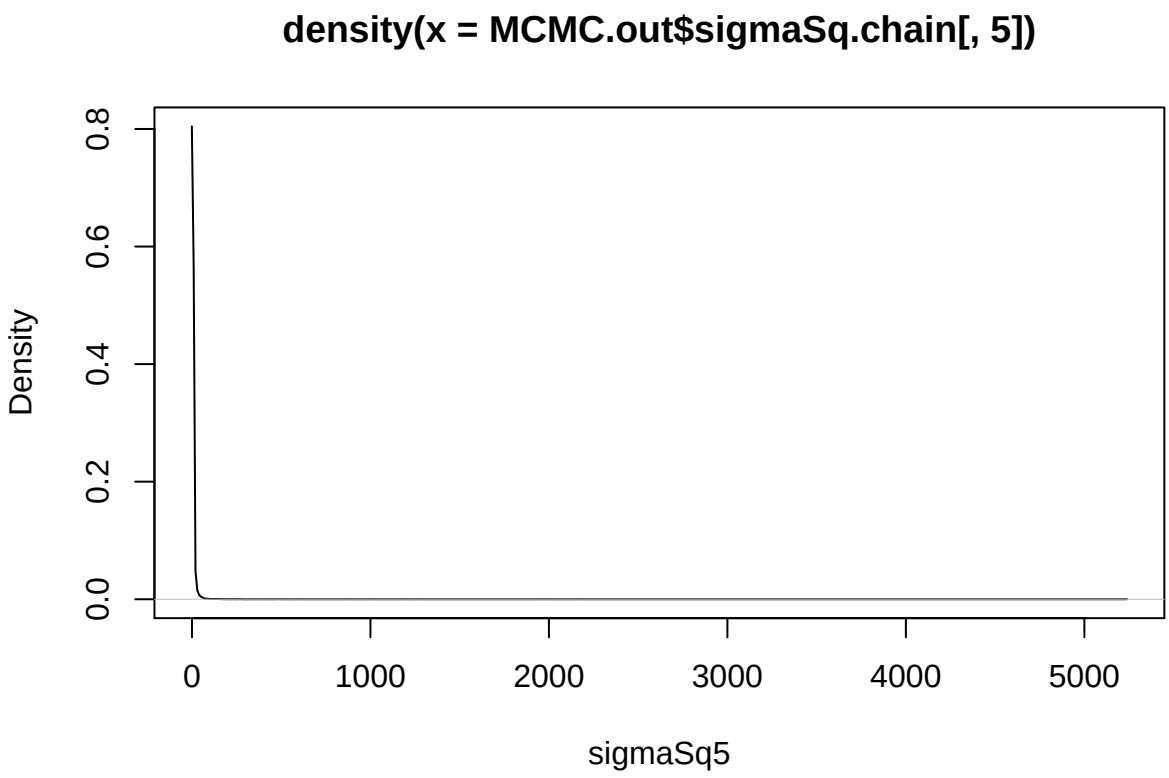

```
plot(density(MCMC.out$sigmaSq.chain[,3]),xlab = "sigmaSq3")
```



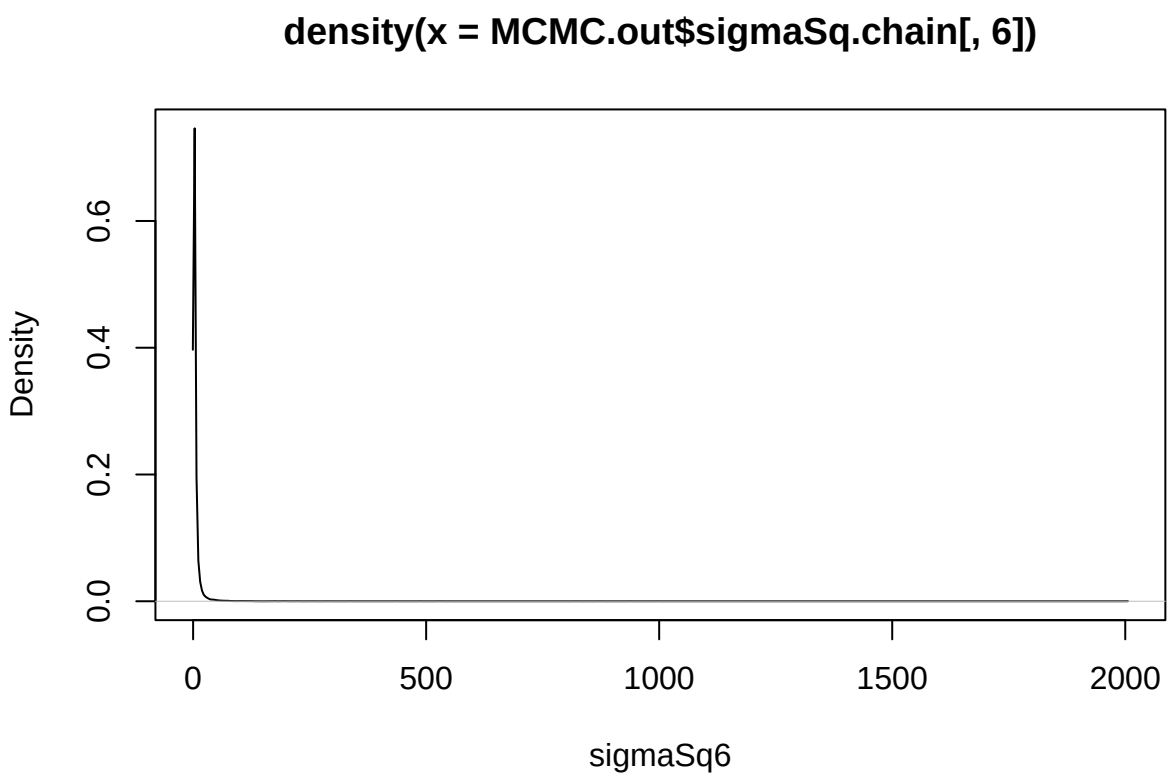

```
plot(density(MCMC.out$sigmaSq.chain[,4]),xlab = "sigmaSq4")
```



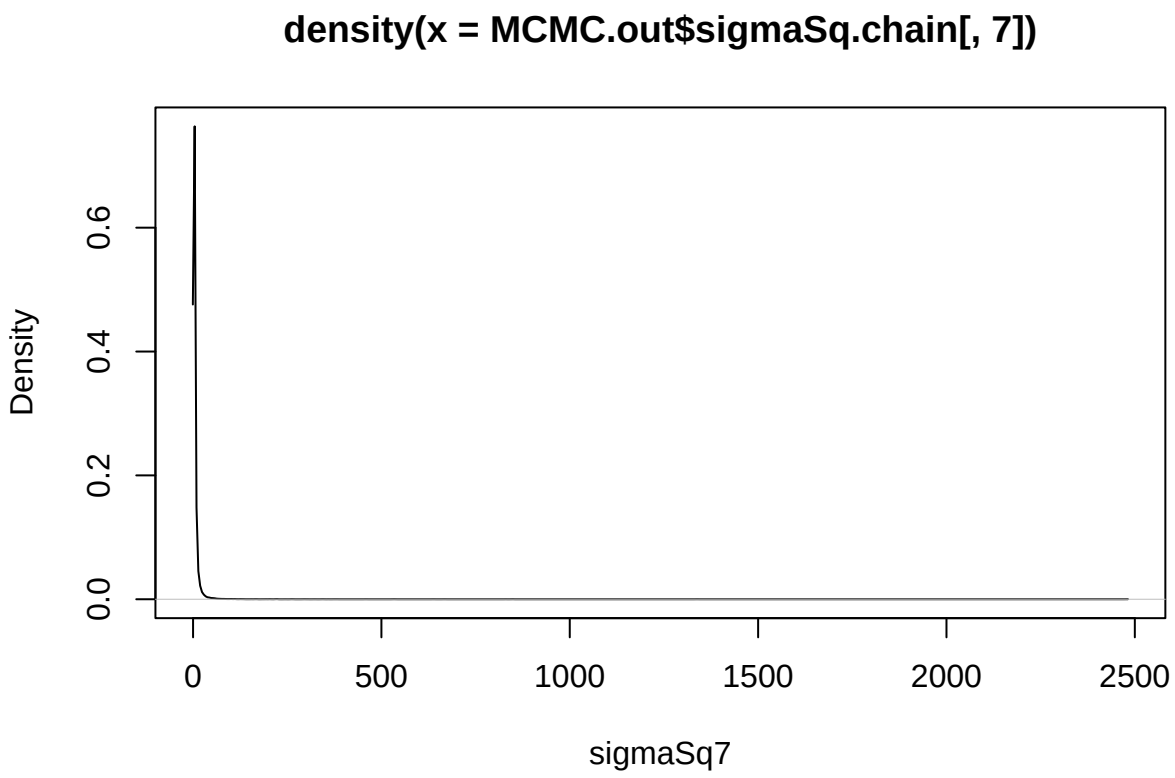

```
plot(density(MCMC.out$sigmaSq.chain[,5]),xlab = "sigmaSq5")
```



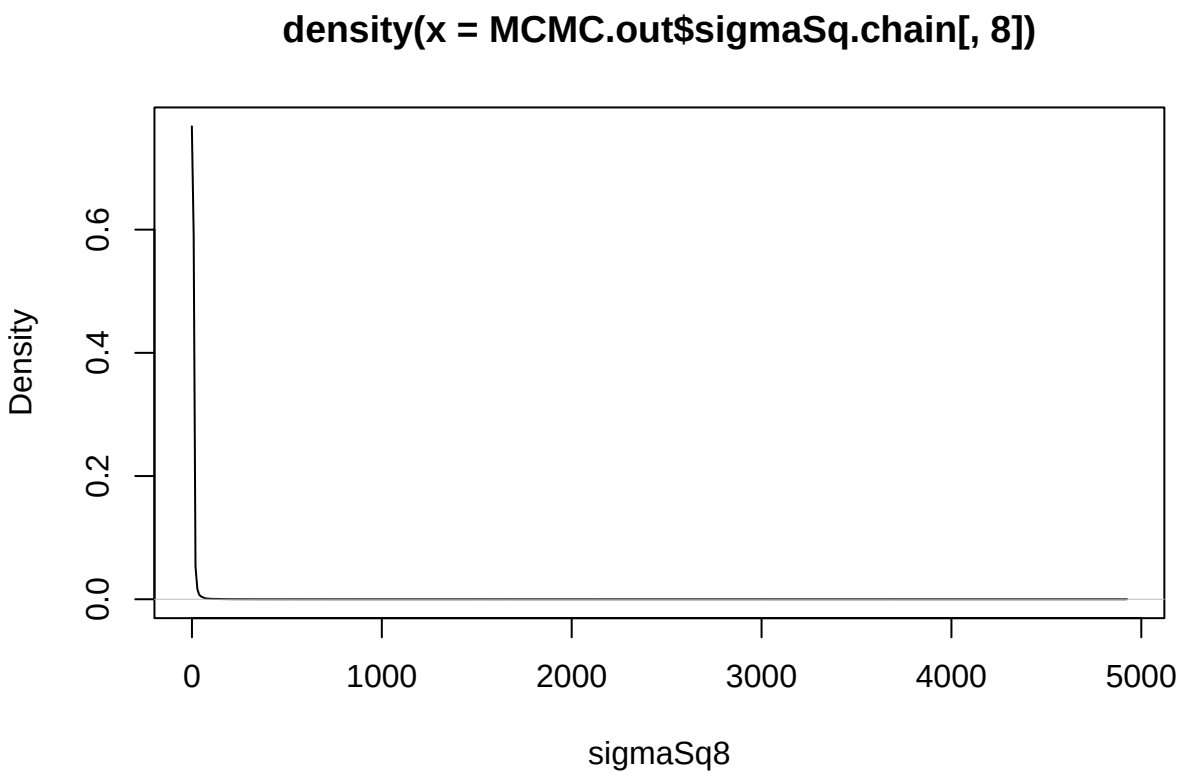

```
plot(density(MCMC.out$sigmaSq.chain[,6]),xlab = "sigmaSq6")
```



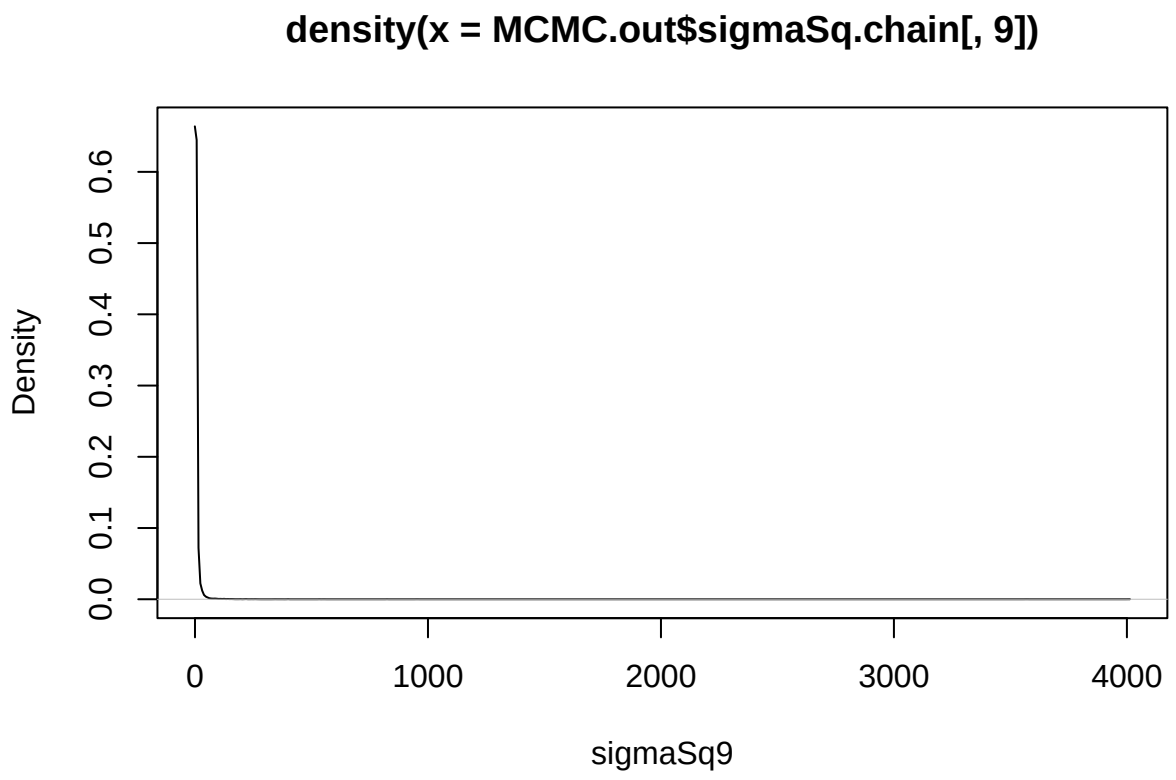

```
plot(density(MCMC.out$sigmaSq.chain[,7]),xlab = "sigmaSq7")
```



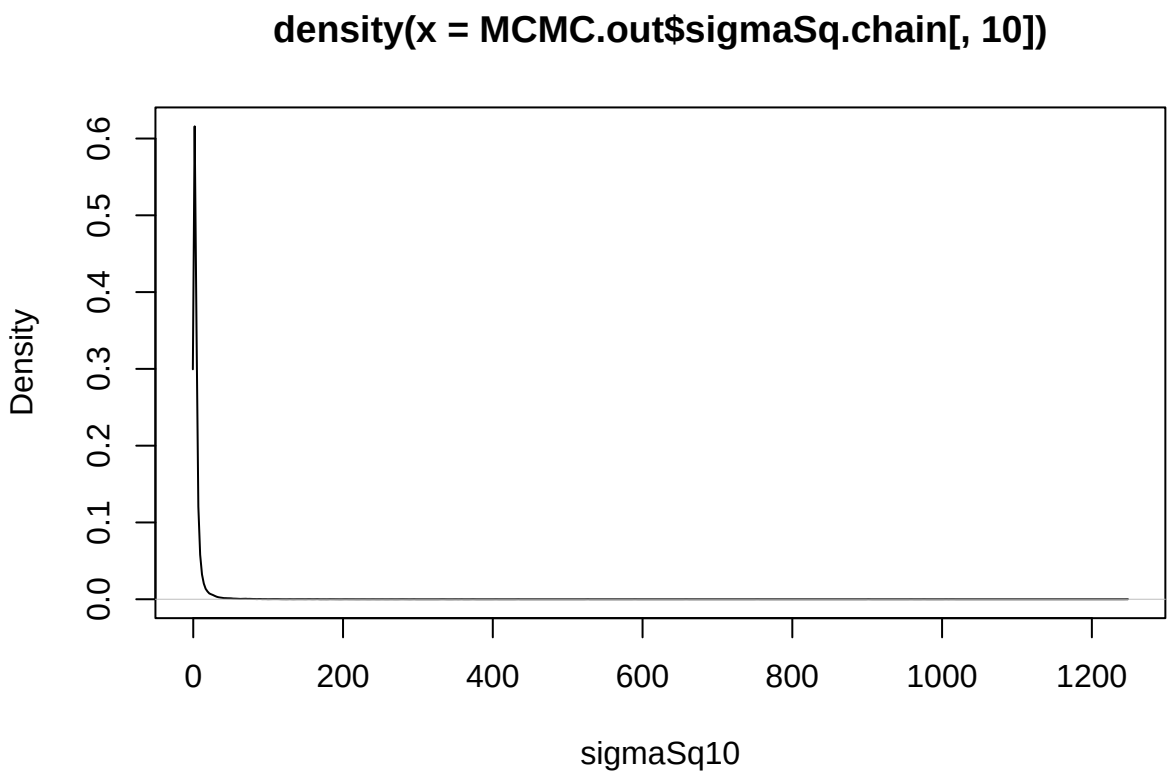

```
plot(density(MCMC.out$sigmaSq.chain[,8]),xlab = "sigmaSq8")
```




```
plot(density(MCMC.out$sigmaSq.chain[,9]),xlab = "sigmaSq9")
```




```
plot(density(MCMC.out$sigmaSq.chain[,10]),xlab = "sigmaSq10")
```



(e)

jags part

```
library(rjags)
```

```
## Warning: package 'rjags' was built under R version 4.3.3
```

```
## Loading required package: coda
```

```
## Warning: package 'coda' was built under R version 4.3.3
```

```
## Linked to JAGS 4.3.1
```

```
## Loaded modules: basemod,bugs
```

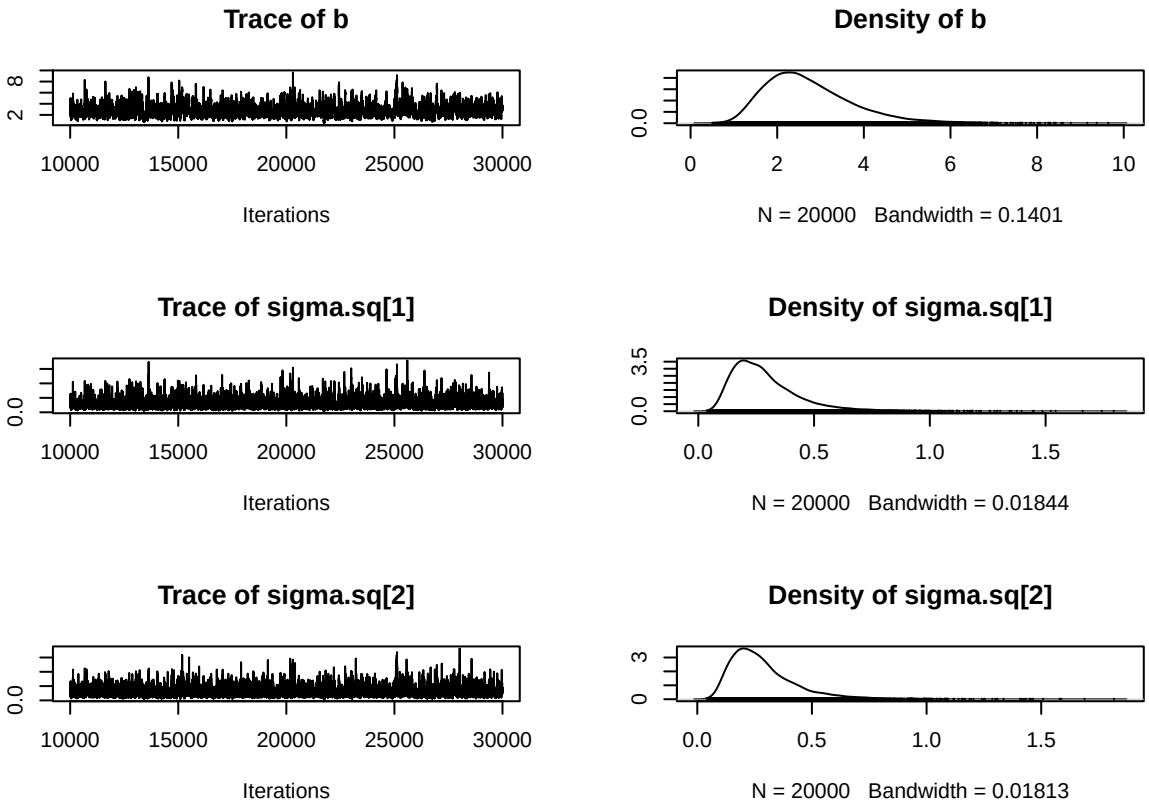
```
## a=10
a <- 10

data <- list(Y = Y, n = length(Y), a = a)
model_string <- textConnection("model{
# Likelihood (dnorm uses a precision, not variance)
```

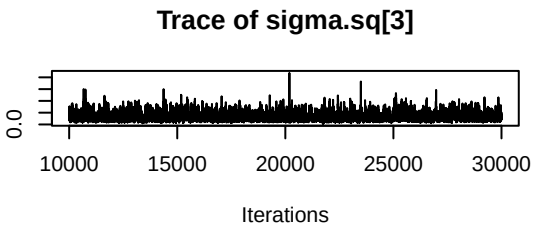
```
for(i in 1:n){
Y[i] ~ dnorm(0,tau[i]) #tau = 1/sigma.sq
}

#prior
for(i in 1:n){
  tau[i] ~ dgamma(a, b)
  sigma.sq[i] <- 1/tau[i]
}

b ~ dgamma(1,1)

}")
tau_init <- rep(0.04, n)
inits <- list(tau = tau_init, b = 0.02)
model <- jags.model(model_string, data = data, inits = inits, quiet = TRUE)
update(model, 10000, progress.bar = "none")
params <- c("b", "sigma.sq")
samples <- coda.samples(model,
                        variable.names = params,
                        n.iter = 20000, progress.bar = "none")

summary(samples)
```

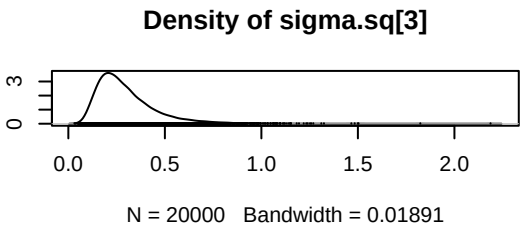
```
##
## Iterations = 10001:30000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean     SD Naive SE Time-series SE
## b            2.7352 1.0277 0.007267       0.036457
## sigma.sq[1]  0.2880 0.1515 0.001071       0.004009
## sigma.sq[2]  0.2870 0.1512 0.001069       0.003997
## sigma.sq[3]  0.2970 0.1526 0.001079       0.003906
## sigma.sq[4]  0.2926 0.1517 0.001073       0.003991
## sigma.sq[5]  0.2909 0.1519 0.001074       0.003893
## sigma.sq[6]  0.2957 0.1521 0.001076       0.003909
## sigma.sq[7]  0.3032 0.1524 0.001078       0.003789
## sigma.sq[8]  0.2894 0.1515 0.001071       0.004005
## sigma.sq[9]  0.4919 0.2008 0.001420       0.003963
## sigma.sq[10] 0.2972 0.1514 0.001070       0.003926
##
## 2. Quantiles for each variable:
##
##                2.5%    25%    50%    75%  97.5%
## b            1.2455 2.0019 2.5582 3.2852 5.2238
## sigma.sq[1]  0.1014 0.1841 0.2559 0.3531 0.6733
## sigma.sq[2]  0.1008 0.1841 0.2538 0.3503 0.6728
## sigma.sq[3]  0.1074 0.1910 0.2618 0.3643 0.6941
```

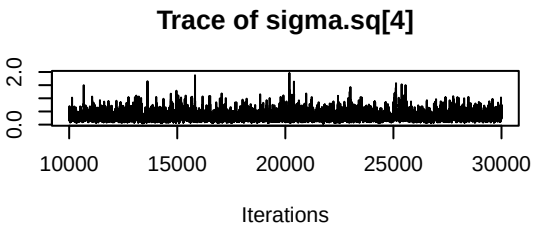
```
## sigma.sq[4]  0.1037 0.1892 0.2579 0.3561 0.6792
## sigma.sq[5]  0.1023 0.1866 0.2573 0.3552 0.6725
## sigma.sq[6]  0.1060 0.1907 0.2624 0.3624 0.6835
## sigma.sq[7]  0.1125 0.1966 0.2687 0.3725 0.6856
## sigma.sq[8]  0.1020 0.1849 0.2556 0.3544 0.6755
## sigma.sq[9]  0.2301 0.3520 0.4511 0.5840 0.9954
## sigma.sq[10] 0.1079 0.1927 0.2643 0.3615 0.6831
```

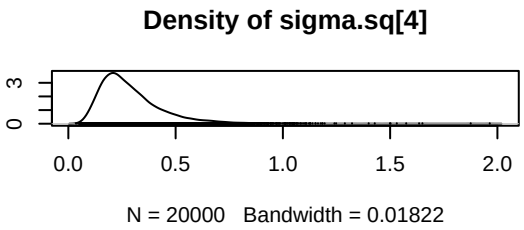
```
plot(samples)
```

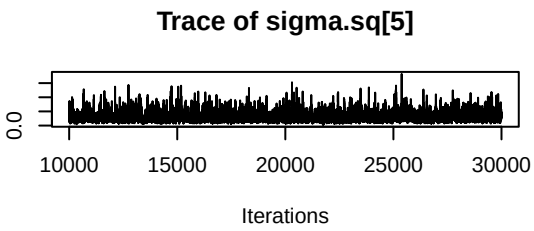


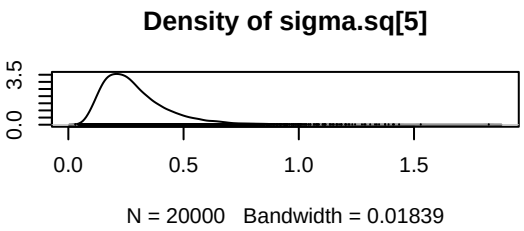

Trace of sigma.sq[3]
0.0
10000
15000
20000
25000
30000
Iterations

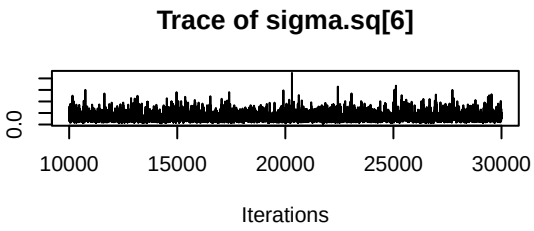

Density of sigma.sq[3]
0
3
0.0
0.5
1.0
1.5
2.0
N = 20000 Bandwidth = 0.01891

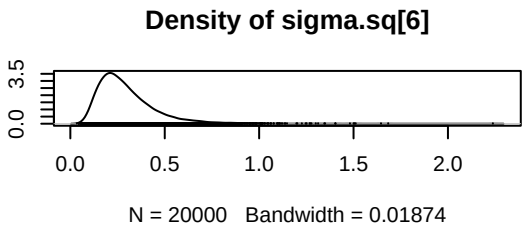

Trace of sigma.sq[4]
0.0
2.0
10000
15000
20000
25000
30000
Iterations

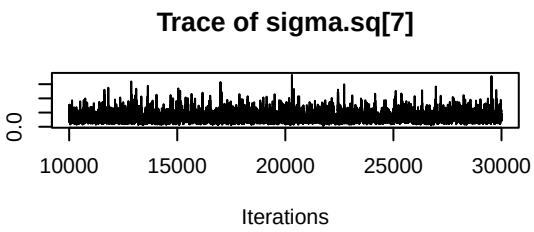

Density of sigma.sq[4]
0
3
0.0
0.5
1.0
1.5
2.0
N = 20000 Bandwidth = 0.01822

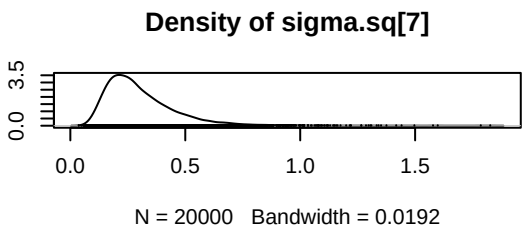

Trace of sigma.sq[5]
0.0
10000
15000
20000
25000
30000
Iterations

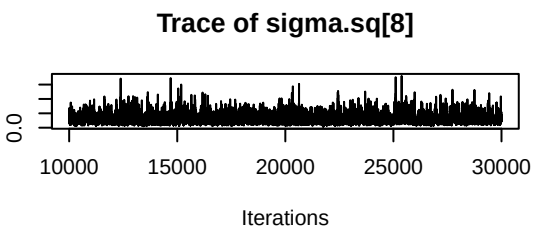

Density of sigma.sq[5]
0.0
3.5
0.0
0.5
1.0
1.5
N = 20000 Bandwidth = 0.01839

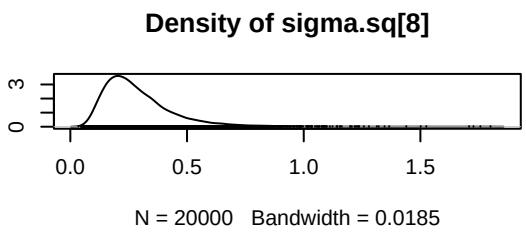

Trace of sigma.sq[6]
0.0
10000
15000
20000
25000
30000
Iterations

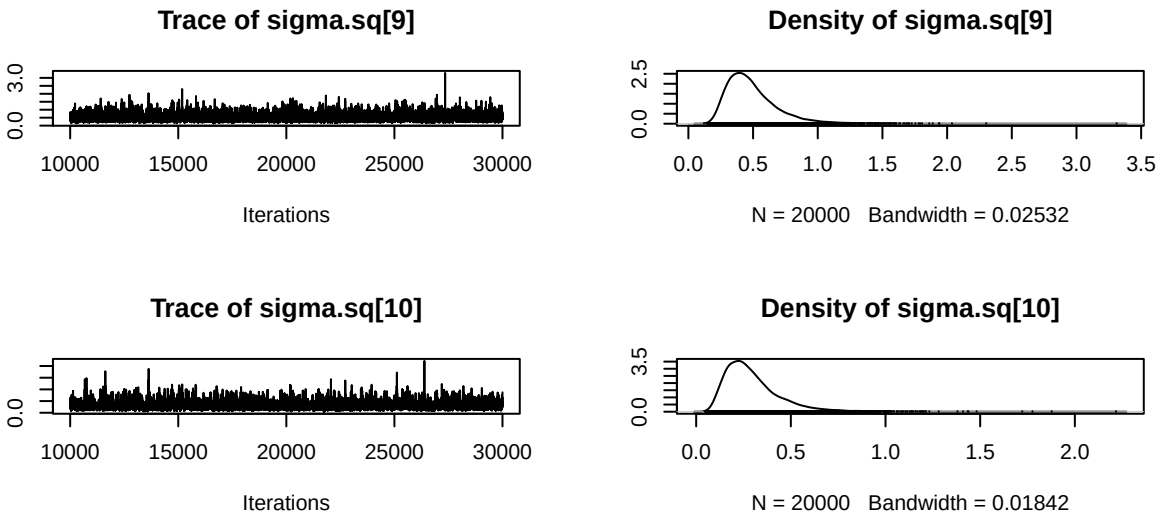

Density of sigma.sq[6]
0.0
3.5
0.0
0.5
1.0
1.5
2.0
N = 20000 Bandwidth = 0.01874


Trace of sigma.sq[7]
0.0
10000
15000
20000
25000
30000
Iterations


Density of sigma.sq[7]
0.0
3.5
0.0
0.5
1.0
1.5
N = 20000 Bandwidth = 0.0192


Trace of sigma.sq[8]
0.0
10000
15000
20000
25000
30000
Iterations


Density of sigma.sq[8]
0
3
0.0
0.5
1.0
1.5
N = 20000 Bandwidth = 0.0185



both have similar results

4

(a)

because when calculate mean of θ_i prior then we get q_i which is belong to $(0, 1)$

(b)

(c)

The full conditional posterior for θ_1 is $Beta(Y_1 + q_1 * exp(m), n_1 - Y_1 + (1 - q_i) * exp(m))$

(d)

```
proportion <- c(0.845,0.847,0.880,0.674,0.909,0.898,0.770,0.801,0.802,0.875)
clucth_make <- c(64,72,55,27,75,24,28,66,40,13)
clucth_attm <- c(75,95,63,39,83,26,41,82,54,16)

player_nam <- c("Russell Westbrook ","James Harden ","Kawhi Leonard ","LeBron James",
                "Isaiah Thomas","Stephen Curry","Giannis Antetokounmpo",
```

```
                "John Wall","Anthony Davis","Kevin Durant")
data <- data.frame(player_nam,proportion,clucth_make,clucth_attm)
n <- 10
Y<- data$clucth_make
theta <- data$proportion
n_s <- data$clucth_attm
q <- runif(n)
m.update <- function(m,theta, q, sdev =4)
{
  m_j <- m
  m_c <- rnorm(1,mean = m,sd = sdev)
  Low <- (m^2)/20
  Upp <- (m_c^2)/20
  for (i in 1:n) {
    Low <- Low + (q[i]* exp(m) - 1)*log(theta[i]) + ((1- q[i])*exp(m) -1)*log(1-theta[i])
  }

  for (i in 1:n) {
    Upp <- Upp + (q[i]* exp(m_c) - 1)*log(theta[i]) + ((1- q[i])*exp(m_c) -1 )*log(1-theta[i])
  }
  logR <- Upp - Low
  if(logR == "NaN")
  {
    return(m_j)
  }
  U <- runif(1) # U(0,1)
  if(log(U) < logR){
    m_j <- m_c
  }

  return(m_j)
}


theta.update <- function(Y,q,n,m, n_s)
{
  theta.pos <- rbeta(n, shape1 = Y + q*exp(m), shape2 = n_s - Y +exp(1- q))
  return(theta.pos)
}


MCMC <- function(Y, m.init, theta.init, a, b, iters){

  # chain initiation
  m <- m.init
  n <- length(Y)
  theta <- rep(theta.init, n)

  # define chains
  m.chain <- rep(NA, iters)
  theta.chain <- matrix(NA, iters, n)

  # start MCMC
```

```
  for(i in 1:iters){
    theta <- theta.update(Y,q,n, m,n_s )
    m <- m.update(m,theta , q)


    m.chain[i] <- m
    theta.chain[i,] <- theta

  }

  # return chains
  out <- list(m.chain = m.chain,
              theta.chain = theta.chain)
  return(out)}

# RUN MCMC

MCMC.out <- MCMC(Y = Y,
                 m.init = 1,
                 theta.init = 0.4,
                 iters = 30000,n)


theta.pos.samples <- MCMC.out$theta.chain
theta1_95 <- quantile(theta.pos.samples[,1], c(0.025,0.975))
theta2_95 <- quantile(theta.pos.samples[,2], c(0.025,0.975))
theta3_95 <- quantile(theta.pos.samples[,3], c(0.025,0.975))
theta4_95 <- quantile(theta.pos.samples[,4], c(0.025,0.975))
theta5_95 <- quantile(theta.pos.samples[,5], c(0.025,0.975))
theta6_95 <- quantile(theta.pos.samples[,6], c(0.025,0.975))
theta7_95 <- quantile(theta.pos.samples[,7], c(0.025,0.975))
theta8_95 <- quantile(theta.pos.samples[,8], c(0.025,0.975))
theta9_95 <- quantile(theta.pos.samples[,9], c(0.025,0.975))
theta10_95 <- quantile(theta.pos.samples[,10], c(0.025,0.975))


m.pos.samples <- MCMC.out$m.chain
m.pos_95 <- quantile(m.pos.samples, c(0.025,0.975))

ci95 <- rbind(m.pos_95 ,theta1_95, theta2_95, theta3_95,
              theta4_95, theta5_95, theta6_95,theta7_95,
              theta8_95, theta9_95,theta10_95)
print("95% credibles interval")
```

```
## [1] "95% credibles interval"
```

```
print(ci95)
```

```
##                      2.5%         97.5%
## m.pos_95   -4.639222e+04 -1198.0257339
## theta1_95   7.507938e-01     0.9145465
## theta2_95   6.497186e-01     0.8225249
```

```
## theta3_95   7.502965e-01     0.9224594
## theta4_95   5.194313e-01     0.8052713
## theta5_95   8.193632e-01     0.9491796
## theta6_95   7.311259e-01     0.9687348
## theta7_95   5.169892e-01     0.7981694
## theta8_95   6.949099e-01     0.8679698
## theta9_95   5.952744e-01     0.8277774
## theta10_95  5.360523e-01     0.9214676
```

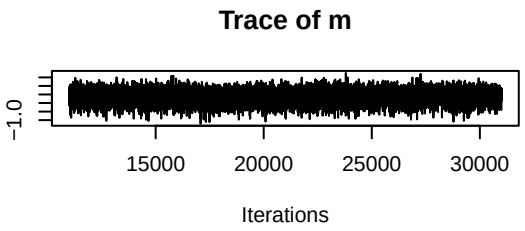
(e)

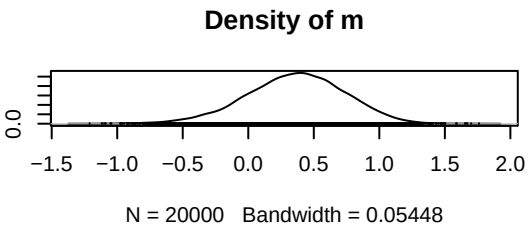
```
proportion <- c(0.845,0.847,0.880,0.674,0.909,0.898,0.770,0.801,0.802,0.875)
clucth_make <- c(64,72,55,27,75,24,28,66,40,13)
clucth_attm <- c(75,95,63,39,83,26,41,82,54,16)

player_nam <- c("Russell Westbrook ","James Harden ","Kawhi Leonard ","LeBron James",
                "Isaiah Thomas","Stephen Curry","Giannis Antetokounmpo",
                "John Wall","Anthony Davis","Kevin Durant")
data <- data.frame(player_nam,proportion,clucth_make,clucth_attm)
n <- 10
Y<- data$clucth_make
theta <- data$proportion
n_s <- data$clucth_attm
q <- runif(n)
data <- list(Y = Y, n = n, q=q, n_s = n_s)
library(rjags)
model_string <- textConnection("model{
# Likelihood
for(i in 1:n){
Y[i] ~ dbin(theta[i],n_s[i])
}

#prior
for(i in 1:n){
theta[i] ~ dbeta(q[i] * exp(m),exp(m)*(1-q[i]) )
}
m ~ dnorm(0,0.1)

}")
theta_inti <- rep(0.4,n)
inits <- list( theta = theta_inti,m = 3.3)
model <- jags.model(model_string, data = data, inits = inits, quiet = TRUE)
update(model, 10000, progress.bar = "none")
params <- c("m", "theta")
samples <- coda.samples(model,
                        variable.names = params,
                        n.iter = 20000, progress.bar = "none")

summary(samples)
```

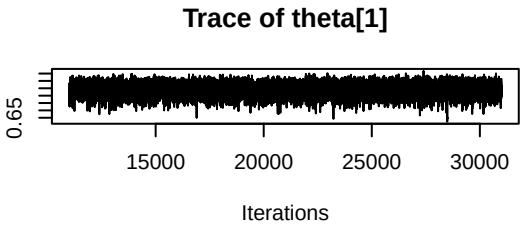
```
##
## Iterations = 11001:31000
```

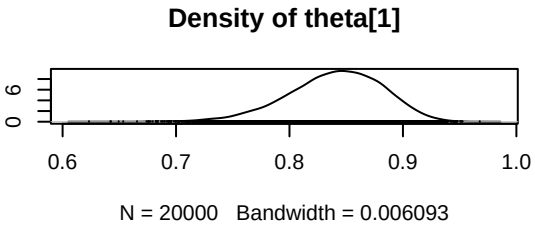
```
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean      SD  Naive SE Time-series SE
## m         0.3594 0.37466 0.0026493      0.0037126
## theta[1]  0.8402 0.04166 0.0002946      0.0003883
## theta[2]  0.7597 0.04328 0.0003060      0.0004027
## theta[3]  0.8582 0.04327 0.0003060      0.0004040
## theta[4]  0.6811 0.07189 0.0005083      0.0006414
## theta[5]  0.8964 0.03281 0.0002320      0.0003079
## theta[6]  0.8873 0.06087 0.0004304      0.0006603
## theta[7]  0.6679 0.07156 0.0005060      0.0006298
## theta[8]  0.8018 0.04294 0.0003036      0.0003934
## theta[9]  0.7358 0.05833 0.0004124      0.0005323
## theta[10] 0.7900 0.09420 0.0006661      0.0008967
##
## 2. Quantiles for each variable:
##
##              2.5%    25%    50%    75%  97.5%
## m         -0.4147 0.1164 0.3725 0.6156 1.0524
## theta[1]   0.7508 0.8140 0.8430 0.8701 0.9126
## theta[2]   0.6698 0.7317 0.7613 0.7895 0.8394
## theta[3]   0.7635 0.8311 0.8620 0.8897 0.9310
## theta[4]   0.5338 0.6337 0.6835 0.7310 0.8140
## theta[5]   0.8242 0.8758 0.8993 0.9202 0.9515
## theta[6]   0.7431 0.8524 0.8967 0.9330 0.9759
## theta[7]   0.5223 0.6199 0.6696 0.7183 0.8012
## theta[8]   0.7110 0.7744 0.8044 0.8320 0.8782
## theta[9]   0.6153 0.6982 0.7383 0.7766 0.8432
## theta[10]  0.5780 0.7317 0.8001 0.8603 0.9401
```

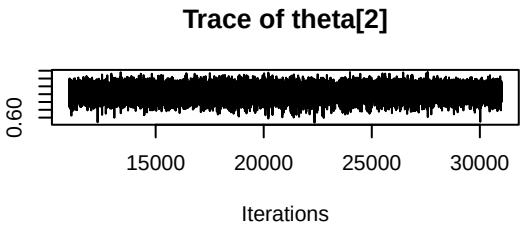
```
plot(samples)
```

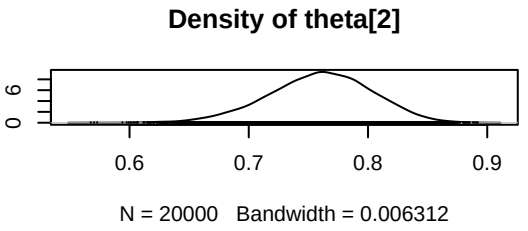

Trace of m
−1.0
15000
20000
25000
30000
Iterations

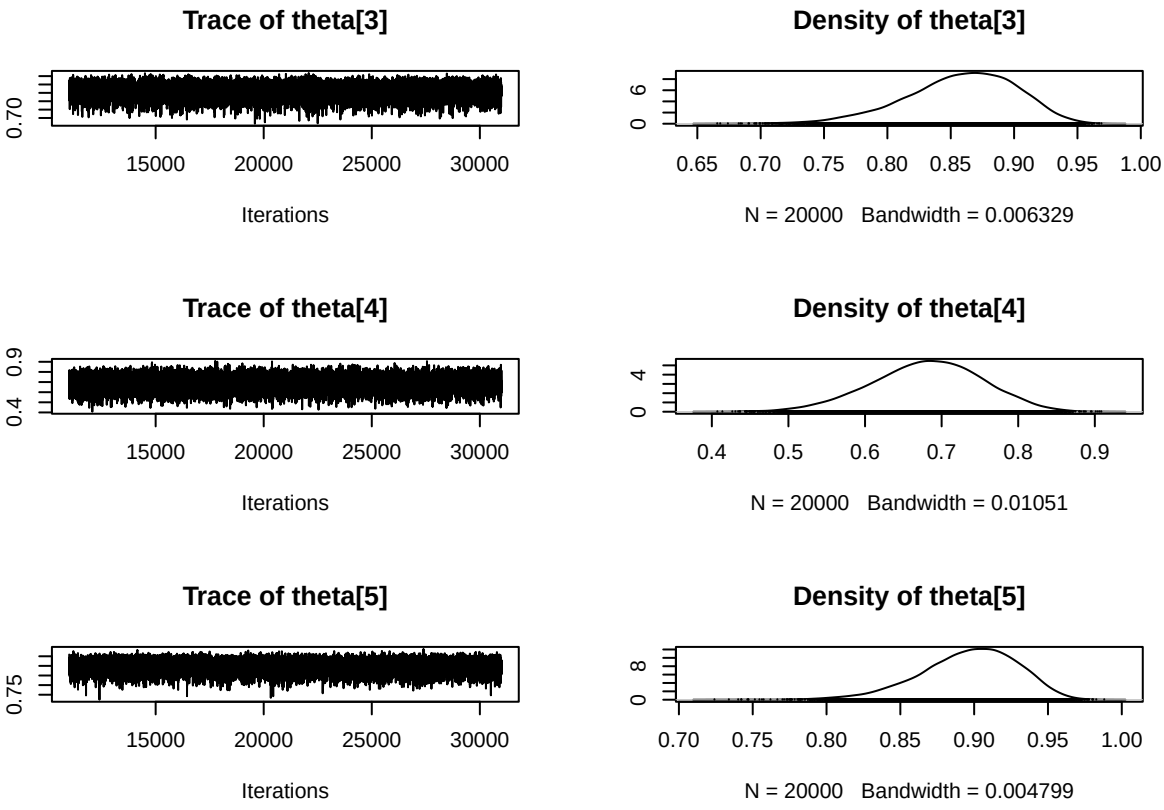

Density of m
0.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
N = 20000 Bandwidth = 0.05448

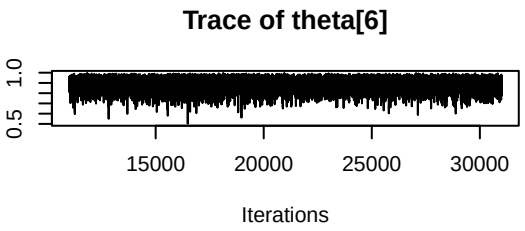

Trace of theta[1]
0.65
15000
20000
25000
30000
Iterations

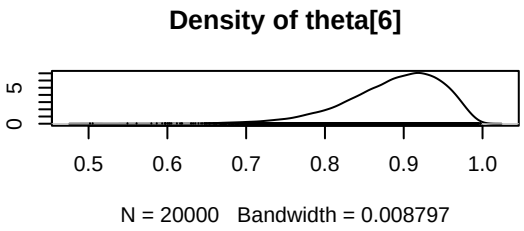

Density of theta[1]
0 6
0.6
0.7
0.8
0.9
1.0
N = 20000 Bandwidth = 0.006093

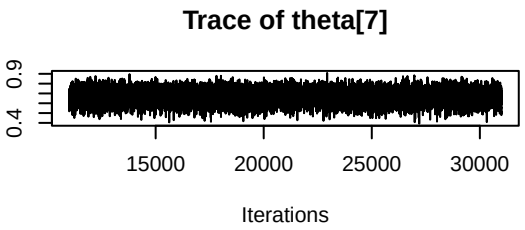

Trace of theta[2]
0.60
15000
20000
25000
30000
Iterations

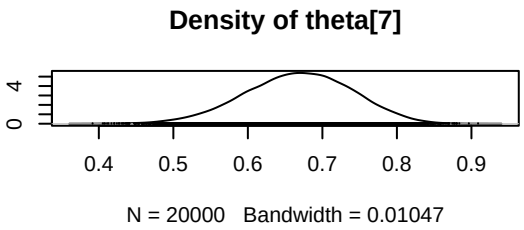

Density of theta[2]
0 6
0.6
0.7
0.8
0.9
N = 20000 Bandwidth = 0.006312

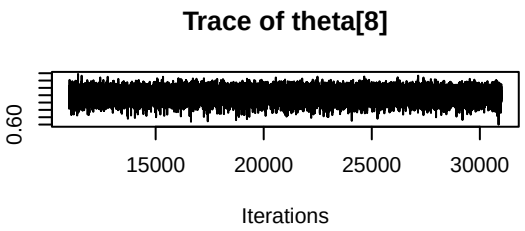

Trace of theta[3]
0.70
15000
20000
25000
30000
Iterations
Density of theta[3]
0 6
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00
N = 20000 Bandwidth = 0.006329
Trace of theta[4]
0.4 0.9
15000
20000
25000
30000
Iterations
Density of theta[4]
0 4
0.4
0.5
0.6
0.7
0.8
0.9
N = 20000 Bandwidth = 0.01051
Trace of theta[5]
0.75
15000
20000
25000
30000
Iterations
Density of theta[5]
0 8
0.70
0.75
0.80
0.85
0.90
0.95
1.00
N = 20000 Bandwidth = 0.004799

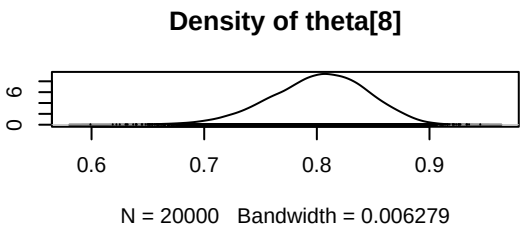

Trace of theta[6]
0.5 1.0
15000 20000 25000 30000
Iterations

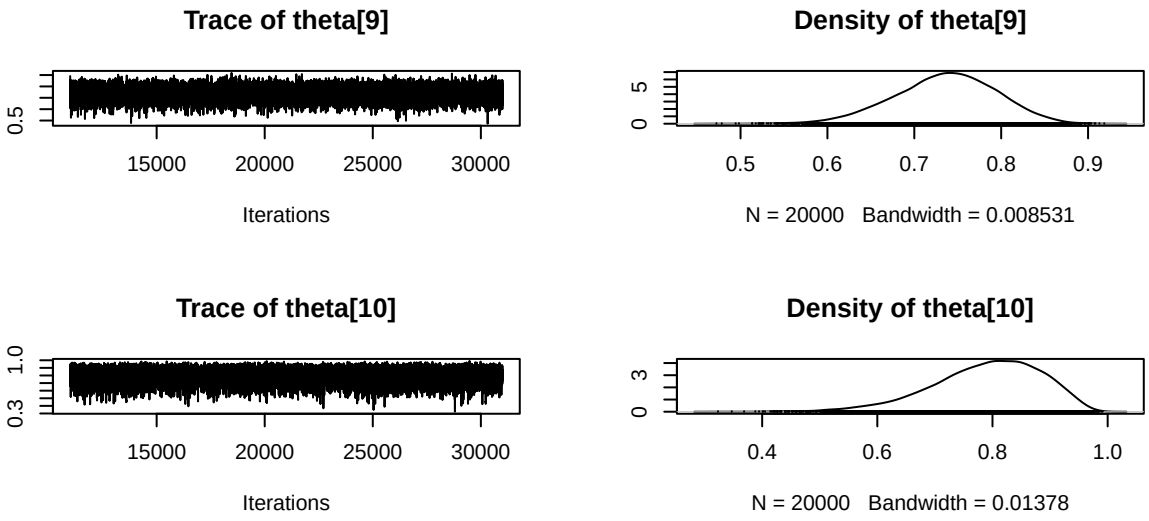

Density of theta[6]
0 5
0.5 0.6 0.7 0.8 0.9 1.0
N = 20000 Bandwidth = 0.008797


Trace of theta[7]
0.4 0.9
15000 20000 25000 30000
Iterations


Density of theta[7]
0 4
0.4 0.5 0.6 0.7 0.8 0.9
N = 20000 Bandwidth = 0.01047


Trace of theta[8]
0.60
15000 20000 25000 30000
Iterations


Density of theta[8]
0 6
0.6 0.7 0.8 0.9
N = 20000 Bandwidth = 0.006279



yes both code have very close answer of 95% credibles interval

(f)

JAGS are easy use here we don't have write all marginal posterior distribution

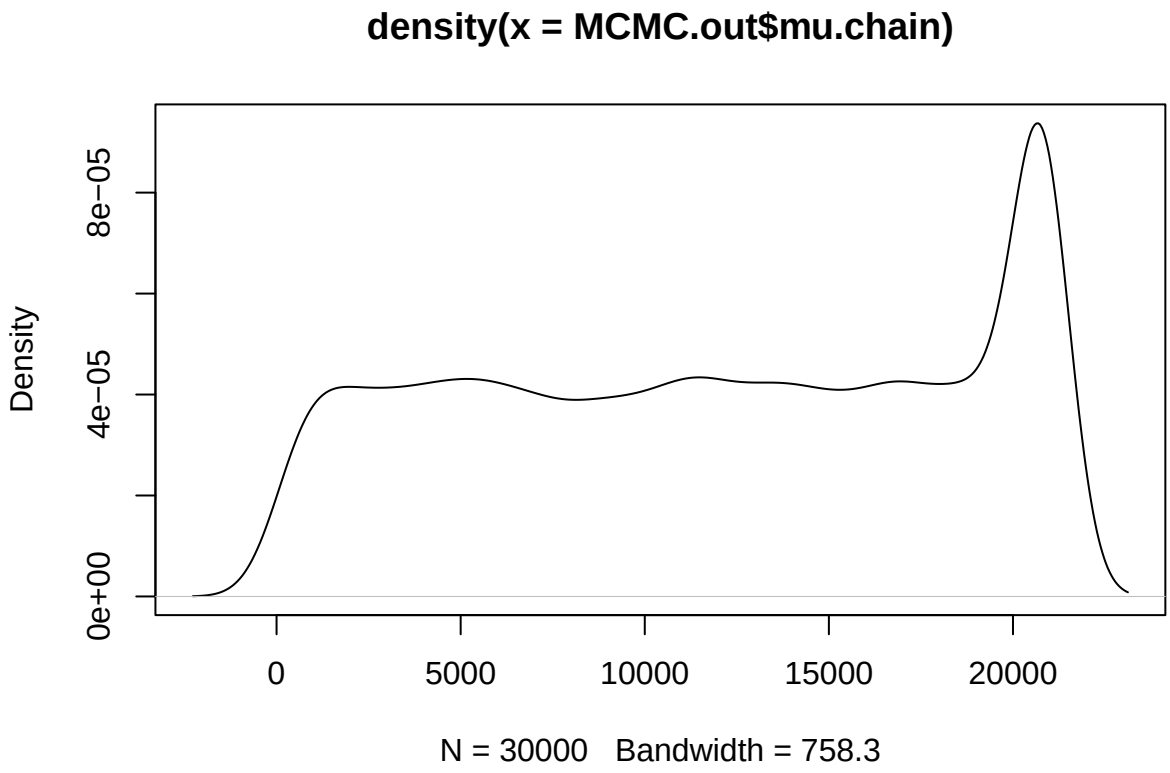
but writing own MCMC code we have find their marginal posterior distribution

8

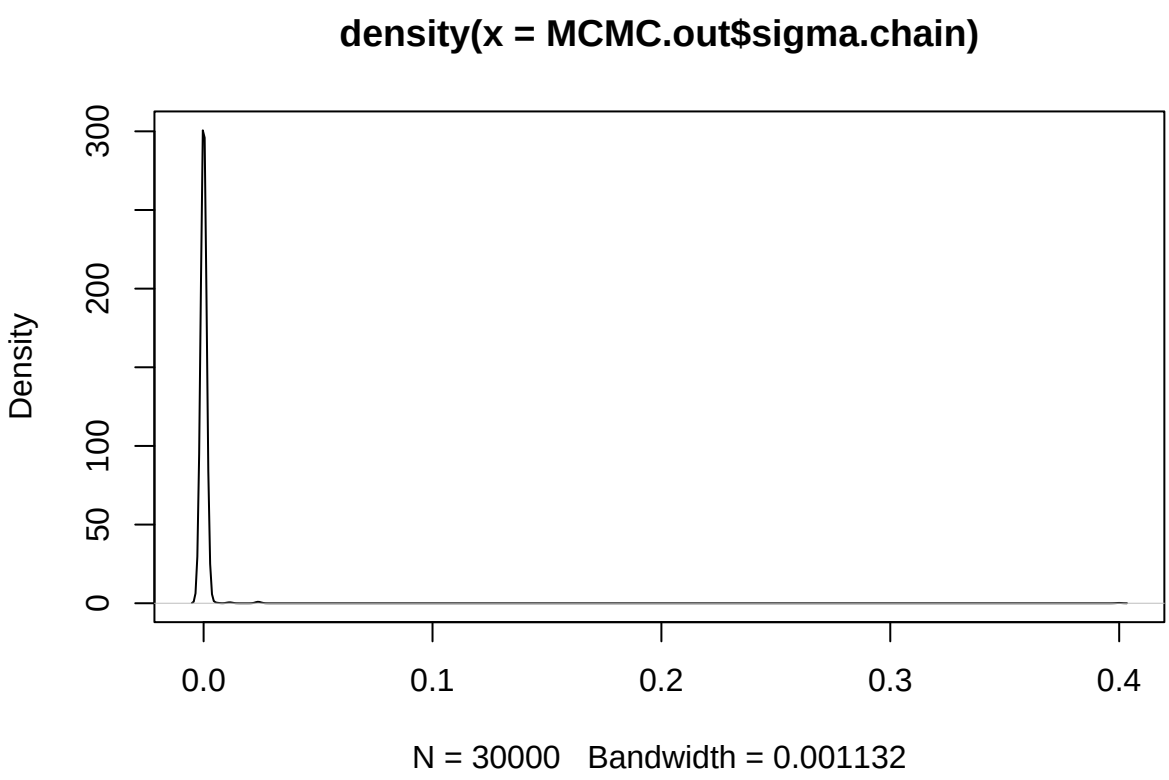
```
library(MASS)
data(galaxies)
Y <- galaxies


mu.update <- function(Y, mu, sigma)
{
  mu_j <- mu
  mu_c <- rnorm(1, mean = mu, sd = 2)
  R <- exp(sum(abs(Y - mu)) - sum(abs(Y - mu_c)))
  R <- min(1,R)
  U <- runif(1)
  if(U < R)
```

```
  {
    mu_j <- mu_c
  }
  return(mu_j)
}
sigma.update <- function(Y, mu, sigma)
{
  n <- length(Y)
  sigma_j <- sigma
  sigma_c <- rnorm(1, mean = sigma, sd = 2)
  R <- (sigma/(sigma_c))^n * exp(sum(Y -mu)*(1/sigma - 1/sigma_j))
  R <- min(1,R)
  if(runif(1) < R)
  {
    sigma_j <- sigma_c
  }
  return(sigma_j)
}

MCMC <- function(Y, mu.init, sigma.init, iters){

  # chain initiation
  mu <- mu.init
  n <- length(Y)
  sigma <- sigma.init

  # define chains
  mu.chain <- rep(NA, iters)
  sigma.chain <-rep(NA, iters)
  # start MCMC
  for(i in 1:iters){
    sigma <- sigma.update(Y, mu, sigma )
    mu <- mu.update(Y, mu, sigma)


    mu.chain[i] <- mu
    sigma.chain[i] <- sigma

  }

  # return chains
  out <- list(mu.chain = mu.chain,
              sigma.chain = sigma.chain)
  return(out)}

# RUN MCMC

MCMC.out <- MCMC(Y = Y,
                 mu.init = 1,
                 sigma.init = 0.4,
                 iters = 30000)
```

```r
# marginal plot
plot(density(MCMC.out$mu.chain))
```




```r
plot(density(MCMC.out$sigma.chain))
```


density(x = MCMC.out$sigma.chain)
Density
0
50
100
200
300
0.0
0.1
0.2
0.3
0.4
N = 30000 Bandwidth = 0.001132