Computer lab 4: Markov Chain Monte Carlo

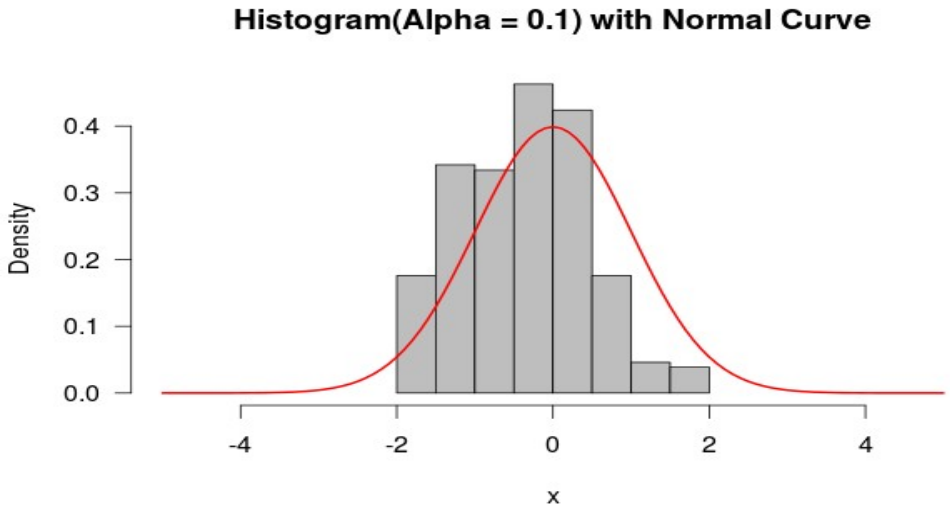
Yonglin Zhuo(yonzh457)

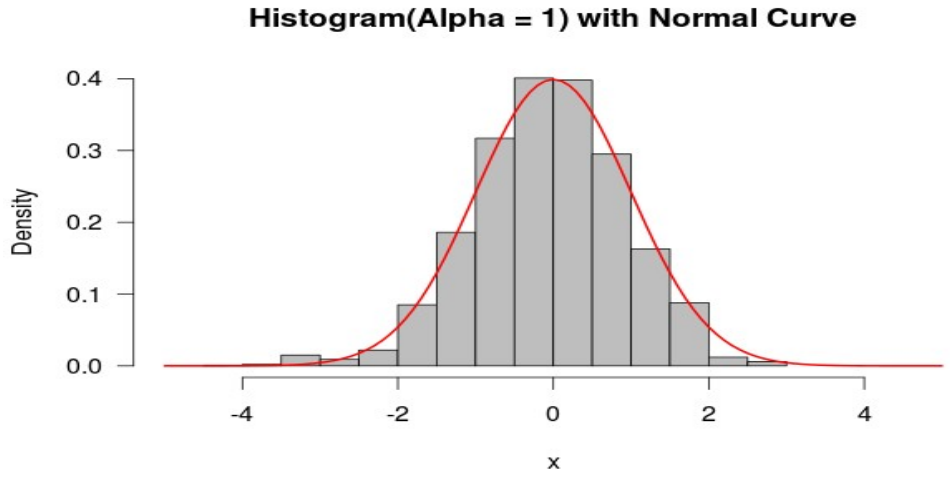
Assignment 1: Different versions of the Metropolis-Hastings algorithm

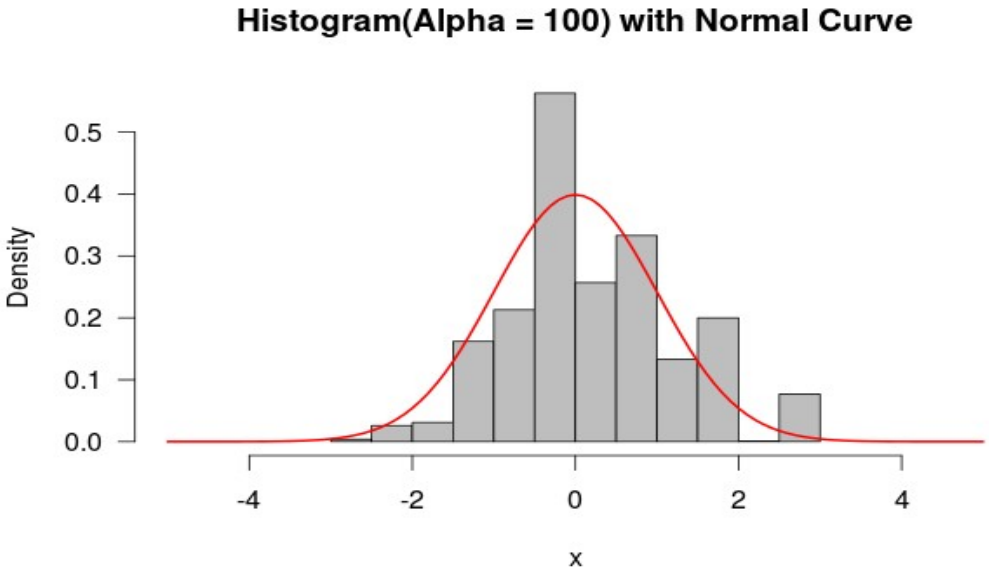
a) The function normm is simulating from a normal with zero mean and unit variance using a Metropolis algorithm with uniform proposal distribution. The exercise is to try the following alpha = 0.1,1 and 200 for 2000 iterations. Present trace plots of line type and histograms with interpretations. Tune the sampler by finding a value of alpha that gives an acceptance probability of 0.3. You have to modify the code in order to save the acceptance rate. Explain the code with comments.

Codes :

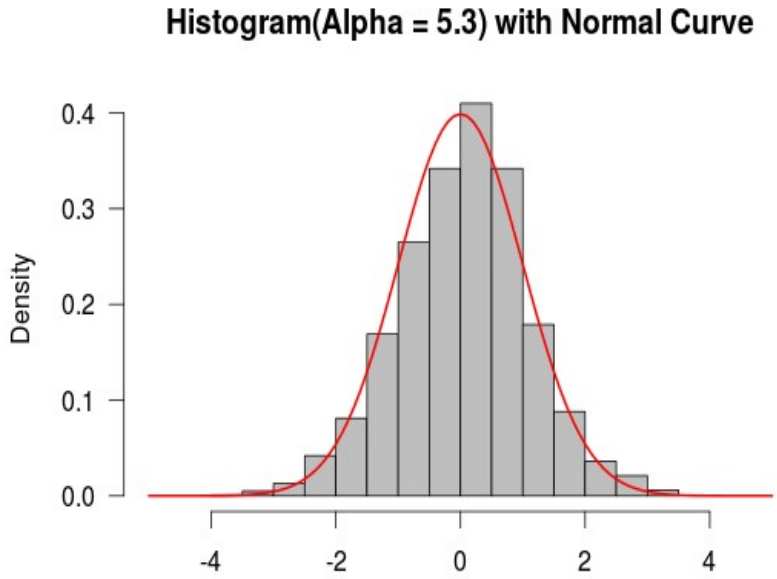
```
normm<-function (Nsim, a)
## Nism is the number if iterations and the a is
## if we choose a = 5.3, it seems the acceptance rate is
## around 0.3
## the parameter of the Proposal distribution g(.|x^t)
## here we choose uniform distribution
{
  vec <- vector("numeric", Nsim)
  ## the vector vec is used to store the simulation result
  x <- 0
  vec[1] <- x
  ## innitialzing
  accepts = 0
  for (i in 2:Nsim) {
    innov <- runif(1, -a, a)
    Xstar <- x + innov
    ## sample a candidate value Xstar
    ## from the proposal distribution
    aprob <- min(1, dnorm(Xstar)/dnorm(x))
    ## computing the Metropolis-Hastings ration
    ## here we using Metroplolis algorithm
    ## so g(x|y) = g(y|x)
    u <- runif(1)
    ## ramdom probability that accepts the Xstar
    if (u < aprob){
      x <- Xstar
      accepts <- accepts +1
    }
    vec[i] <- x
  }
  print(accepts/Nsim)
return (vec)
}

output <- function(iteration=2000,alpha=5.3){
  vec <- normm(iteration,alpha)
  h<-hist(vec, xlim = c(-5,5),freq=FALSE,las=1,col=8,xlab="x",
          main="Histogram(Alpha = ) with Normal Curve")
  curve(dnorm(x, mean=0, sd=1), add=TRUE, col=2, lwd=2)
}
```

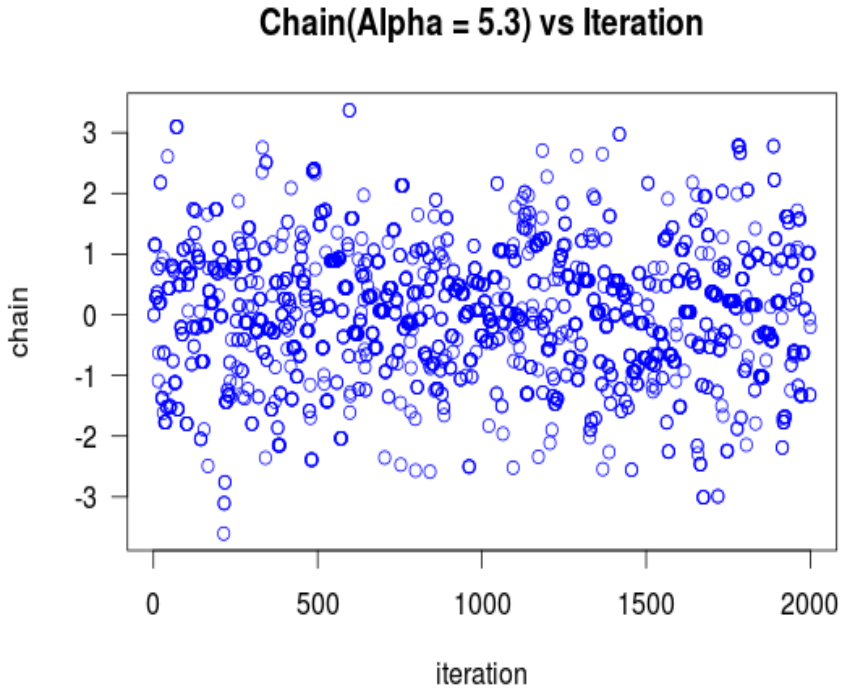

Histogram(Alpha = 0.1) with Normal Curve
Density
0.0
0.1
0.2
0.3
0.4
-4
-2
0
2
4
x


Histogram(Alpha = 1) with Normal Curve
Density
0.0
0.1
0.2
0.3
0.4
-4
-2
0
2
4
x


Histogram(Alpha = 100) with Normal Curve
Density
0.0
0.1
0.2
0.3
0.4
0.5
-4
-2
0
2
4
x

The accept rates for alpha = 0.1, 1, and 100 are 0.9805, 0.7965, and 0.015, respectively. After trying some values of alpha, it is easy to find that the accept value will be around 0.3, if we choose alpha=5.3. As you can see in the following figure that the chain tends to be stationary with the increasing of the iteration. It seems that the simulation is somewhat good.



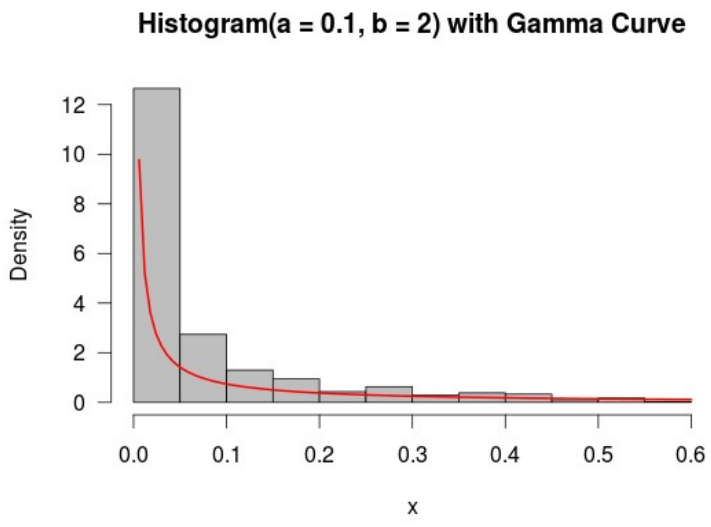


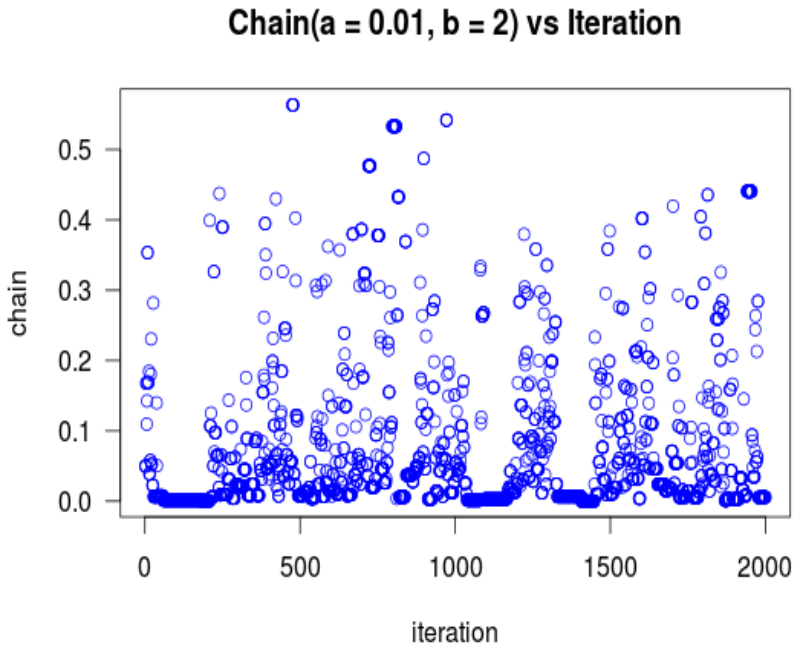
b.) Function gammh is aMetropolis-Hastings independence sampling algorithm with normal proposal distribution with the same mean and variance as the desired gamma. Try a = 0.1, 2 and b = 0.01, 2. Present trace plots and histograms with interpretations. Explain the code with comments.

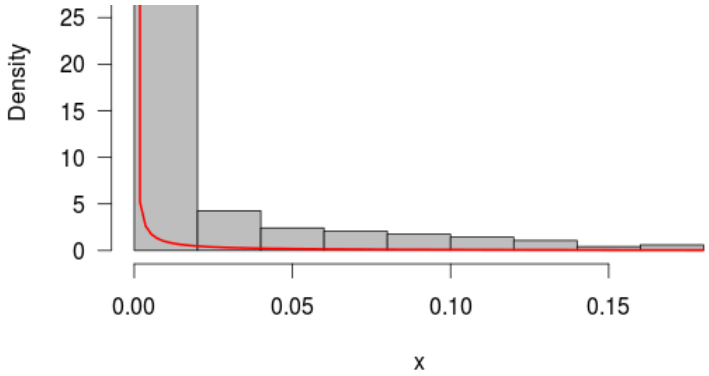
Codes:

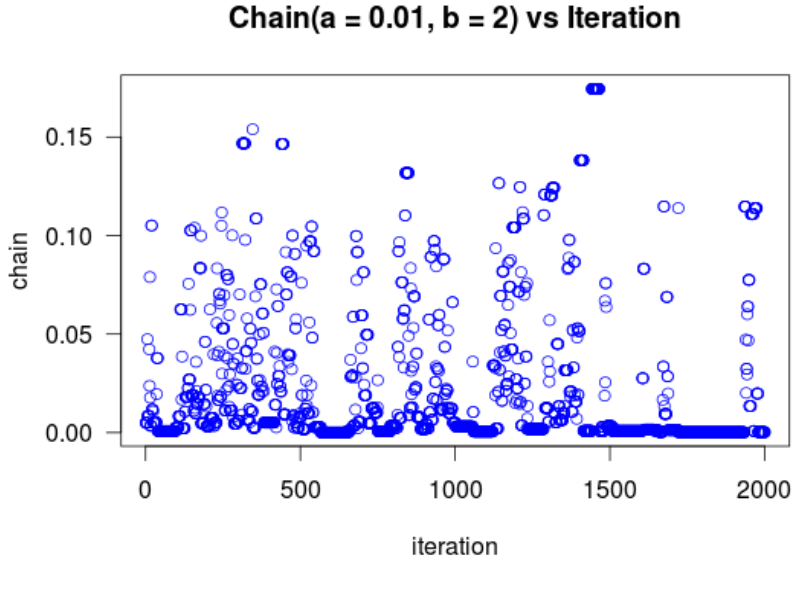
```
gammh<-function (Nsim, a, b)
## Nism is the number if iterations and a and b are the parameters of the
##Gamma distribution
{
  mu <- a/b
  ## the mean of the proposal distribution
  sig <- sqrt(a/(b * b))
  ## the standard deviation of the proposal distribution
  vec <- vector("numeric", Nsim)
  x <- a/b
  vec[1] <- x
  ## initializing
  accepts = 0
  for (i in 2:Nsim) {
    can <- rnorm(1, mu, sig)
    aprob <- min(1, (dgamma(can, a, b)/dgamma(x,
a, b))/(dnorm(can, mu, sig)/dnorm(x,mu, sig)))
    ## computing the Metropolis-Hastings ratio
    ## here we use Metropolis-Hastings independence sampling algorithm
    ## so g(x|y) = g(x)

    u <- runif(1)
    ## ramdom probability that accepts the Xstar
    if (u < aprob){
      x <- can
    }
    vec[i] <- x
    accepts <- accepts+1
  }
  accepts
  return(vec)
}
output <- function(iteration=2000,a,b){
  vec <- gammh(iteration,a,b)
  h<-hist(vec,freq=FALSE,las=1,col=8,xlab="x",
          main="Histogram(a = 0.1, b = 2) with Gamma Curve")
  curve(dgamma(x, shape=0.1, scale=2), add=TRUE, col=2, lwd=2)
  return (vec)
}
h = output(2000,0.1,2)
plot(h,las=1,col=4,xlab='iteration',ylab='chain',,
     main="Chain(a = 0.01, b = 2) vs Iteration")
```

The simulation seems not so bad. The chain tends to have a period of 250.




Chain(a = 0.01, b = 2) vs Iteration
chain
0.0
0.1
0.2
0.3
0.4
0.5
0
500
1000
1500
2000
iteration


Density
0
5
10
15
20
25
0.00
0.05
0.10
0.15
x


Chain(a = 0.01, b = 2) vs Iteration
chain
0.00
0.05
0.10
0.15
0
500
1000
1500
2000
iteration

Assignment 2: The Gibbs sampling algorithm for the one-way random effects model

a)Implement R-code for the normal one-way random effects model with k levels. Provide results in Tables with mean, standard deviations and 95% credible intervals for the mean, the random effects and all variances for both your own code and the MCMChregress() analysis

Codes:

```
library(SuppDists)
data_set <- c(1545,1440,1440,1520,1580,1540,1555,1490,1560,1495,
              1595,1550,1605,1510,1560,1445,1440,1595,1465,1545,
              1595,1630,1515,1635,1625,1520,1455,1450,1480,1445)
Y <- matrix(data_set,6,5)
#### initialize constants and parameters ####
N <- 10000
## lenght of chain
X <- matrix(0,N,10)
## the chain, a bivariate sample
Y_bar_1 <- mean(Y[1,])
Y_bar_2 <- mean(Y[2,])
Y_bar_3 <- mean(Y[3,])
Y_bar_4 <- mean(Y[4,])
Y_bar_5 <- mean(Y[5,])
Y_bar_6 <- mean(Y[6,])

n_i <- dim(Y)[2]
k <- dim(Y)[1]
n <- k * n_i

a_i <- 0.0001
b_i <- 0.0001

X[1,] <- c(Y_bar_1,Y_bar_2,Y_bar_3,Y_bar_4,Y_bar_5,Y_bar_6,1,1,1,1)

sum_sq = function(vec, c)
{
  temp <- (vec - c)^2
  sum_sq <- sum(temp)
  return (sum_sq)
}

sum_sq_matrix = function(M,c)
{
  temp <- 0
  for (i in 1:6)
  {
    temp <- temp + sum_sq(M[i,],c[i])
  }

  return (temp)
}

#### generate the chain ####
for (i in 2:N){
  P <- X[i-1, ]
  m <- P[7]*P[8] / (P[8] + n_i*P[9]) + P[9]*n_i*Y_bar_1 / (P[8] + n_i*P[9])
  s <- sqrt(P[8]*P[9] / (P[8] + n_i*P[9]))
  X[i,1] <- rnorm(1, m, s)
```

```
  X[i,2:10] <- X[i-1,2:10]

  P <- X[i, ]
  m <- P[7]*P[8] / (P[8] + n_i*P[9]) + P[9]*n_i*Y_bar_2 / (P[8] + n_i*P[9])
  s <- sqrt(P[8]*P[9] / (P[8] + n_i*P[9]))
  X[i, 2] <- rnorm(1, m, s)
  X[i,3:10] <- X[i-1,3:10]

  P <- X[i, ]
  m <- P[7]*P[8] / (P[8] + n_i*P[9]) + P[9]*n_i*Y_bar_3 / (P[8] + n_i*P[9])
  s <- sqrt(P[8]*P[9] / (P[8] + n_i*P[9]))
  X[i, 3] <- rnorm(1, m, s)
  X[i,4:10] <- X[i-1,4:10]

  P <- X[i, ]
  m <- P[7]*P[8] / (P[8] + n_i*P[9]) + P[9]*n_i*Y_bar_4 / (P[8] + n_i*P[9])
  s <- sqrt(P[8]*P[9] / (P[8] + n_i*P[9]))
  X[i, 4] <- rnorm(1, m, s)
  X[i,5:10] <- X[i-1,5:10]

  P <- X[i, ]
  m <- P[7]*P[8] / (P[8] + n_i*P[9]) + P[9]*n_i*Y_bar_5 / (P[8] + n_i*P[9])
  s <- sqrt(P[8]*P[9] / (P[8] + n_i*P[9]))
  X[i, 5] <- rnorm(1, m, s)
  X[i,6:10] <- X[i-1,6:10]

  P <- X[i, ]
  m <- P[7]*P[8] / (P[8] + n_i*P[9]) + P[9]*n_i*Y_bar_6 / (P[8] + n_i*P[9])
  s <- sqrt(P[8]*P[9] / (P[8] + n_i*P[9]))
  X[i, 6] <- rnorm(1, m, s)
  X[i,7:10] <- X[i-1,7:10]

  P <- X[i, ]
  m <- (mean(P[1:6]))*k*P[10] / (P[9] + k*P[10])
  s <- sqrt(P[9]*P[10] / (P[9] + k*P[10]))
  X[i, 7] <- rnorm(1, m, s)
  X[i,8:10] <- X[i-1,8:10]

  P <- X[i, ]
  m <- 0.5*n + a_i
  s <- (0.5*sum_sq_matrix(Y, P[1:6])) + b_i
  X[i, 8] <- rinvgamma(1, m, s)
  X[i,9:10] <- X[i-1,9:10]

  P <- X[i, ]
  m <- 0.5*k + a_i
  s <- (0.5* n_i *(sum_sq(P[1:6],P[7]))) + b_i
  X[i, 9] <- rinvgamma(1, m, s)
  X[i,10] <- X[i-1,10]

  P <- X[i, ]
  m <- 0.5 + a_i
  s <- 0.5*P[7]^2 + b_i
  X[i, 10] <- rinvgamma(1, m, s)
}

results <- X[1000:10000, ]
theta <- results[,1:6]
par(mfrow= c(3,3))
```

```
for(i in 1:3){
  title <- paste('Theta', i)
  plot(theta[,i], type='l', main = title, ylab='',col=4,las=1)
  hist(theta[,i], main = title, prob = TRUE,las=1)
  lines(density(theta[,i]),col=4,las=1)
  plot(cumsum(theta[,i]) / 1:length(theta[,i]), type = 'l', main = title, ylab =
'',las=1,col=4)

}

titles <- c('Mu','Sigma Square','Tau Square','Sigma_mu Square')
par(mfrow= c(2,3))
for(i in 9:10){
  title <- titles[i-6]
  plot(results[,i], type='l', main = title, ylab='',col=4,las=1)
  hist(results[,i], main = title, prob = TRUE,las=1)
  lines(density(results[,i]),col=4,las=1)
  plot(cumsum(results[,i]) / 1:length(results[,i]), type = 'l', main = title, ylab
= '',las=1,col=4)

}
```

	Mean	Sd	lower	upper
θ_1	1555.37	28.48	1499.30	1611.80
θ_2	1503.50	28.79	1447.39	1561.48
θ_3	1522.08	28.31	1465.80	1577.17
θ_4	1509.73	28.85	1453.16	1566.20
θ_5	1522.91	28.40	1469.31	1580.90
θ_6	1550.55	28.33	1496.18	1607.05
μ	1467.87	331.79	941.01	1909.73
σ^2	4478.23	1397.89	2218.08	7178.89
τ_2	840686.15	5275551.11	218.41	1466706.88
σ_μ^2	10466580702.17	435632416730.93	0.00	614466261.07

Table 1 mean, standard deviations and 95% credible intervals for the mean

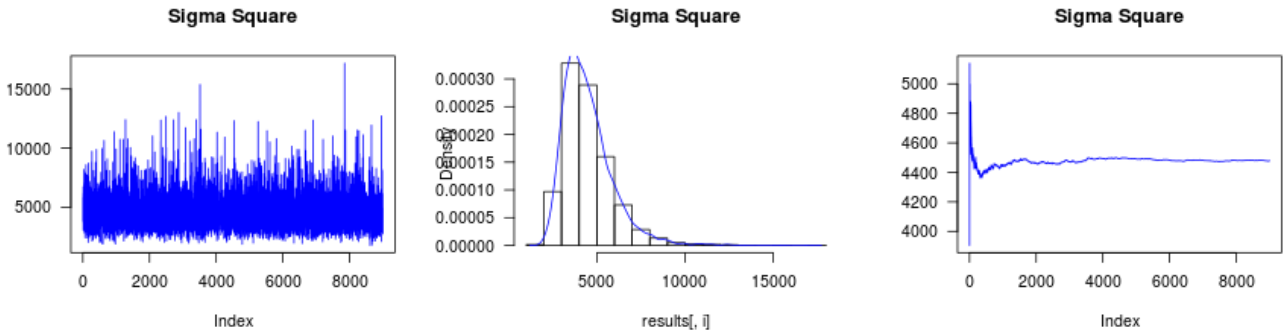


Figure 2 Sigma Square

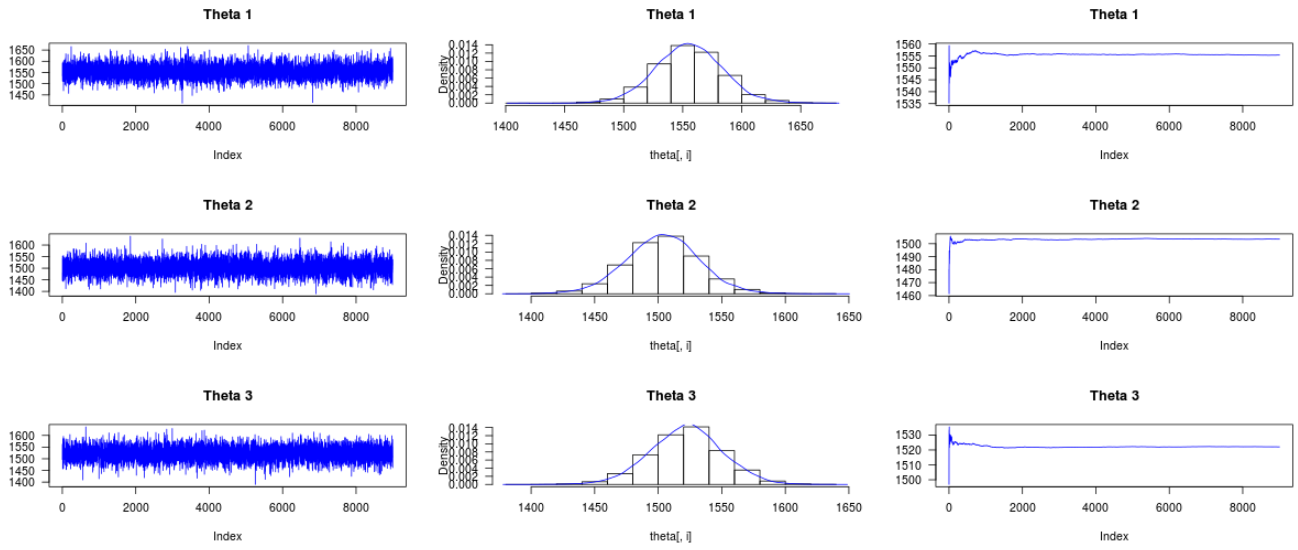


Figure 2 Theta 1 – 3

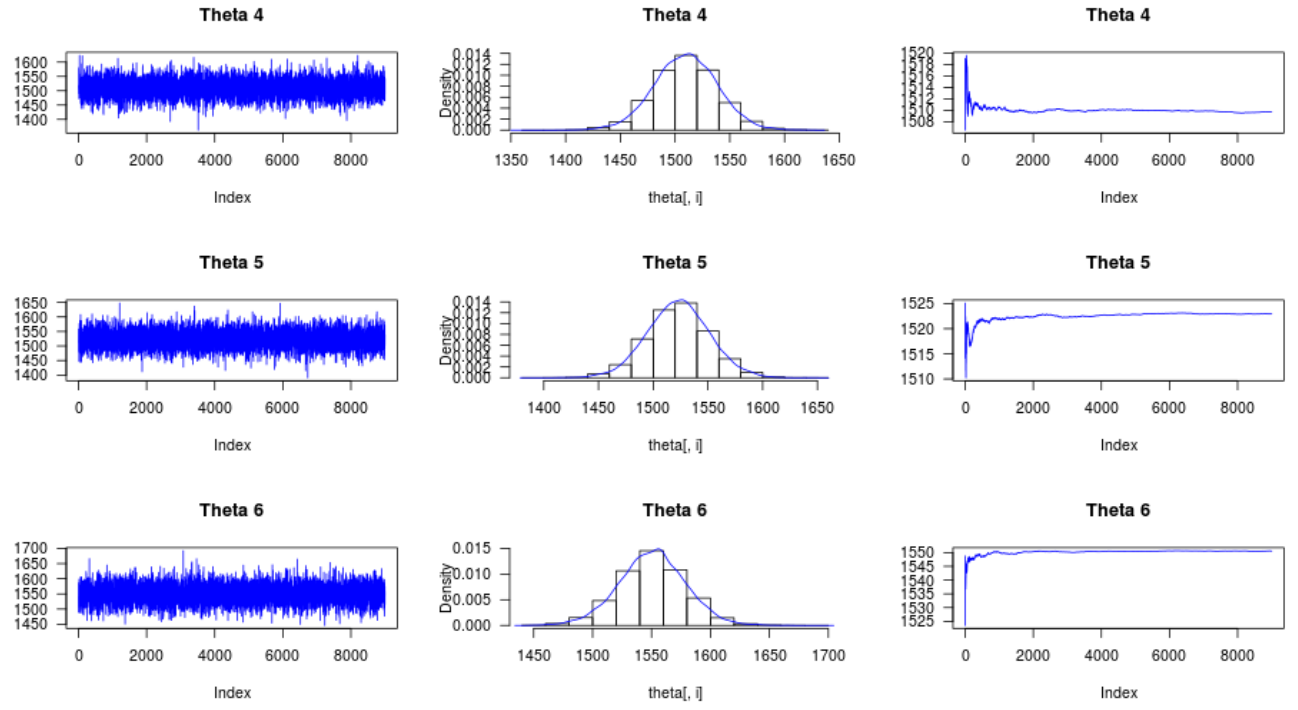


Figure 3 Theta 4 – 6

All the table and Figures above show that our method works very well.

MCMCregress Analysis

Since we can see in the following table the estimates of Theta's are quit close to our own codes, but the Sigma square is quit different.

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

	Mean	SD	Naive SE	Time-series SE
beta.(Intercept)	0.007846	1.005	0.01005	0.01005
b.(Intercept).1	1504.737280	23.164	0.23164	0.22812
b.(Intercept).2	1527.600689	23.048	0.23048	0.22637
b.(Intercept).3	1563.436784	23.193	0.23193	0.23193
b.(Intercept).4	1497.542896	23.155	0.23155	0.23155
b.(Intercept).5	1599.416036	23.071	0.23071	0.22578
b.(Intercept).6	1469.639794	23.272	0.23272	0.23272
VCV.(Intercept).(Intercept)	2768403.634708	2181862.591	21818.62591	21818.62591
sigma2	2681.544750	850.396	8.50396	10.49865
Deviance	320.546141	4.301	0.04301	0.05413

2. Quantiles for each variable:

	2.5%	25%	50%	75%	97.5%
beta.(Intercept)	-1.95	-0.6656	0.008974	0.6796	2.011
b.(Intercept).1	1458.80	1489.3055	1504.789726	1520.0128	1549.648
b.(Intercept).2	1481.68	1512.7453	1527.789610	1542.6936	1573.812
b.(Intercept).3	1517.57	1547.9565	1563.545310	1578.6787	1609.432
b.(Intercept).4	1451.24	1482.2187	1497.456942	1512.4841	1543.284
b.(Intercept).5	1554.21	1584.2422	1599.307265	1614.7616	1644.876
b.(Intercept).6	1422.77	1454.2027	1469.993666	1485.0459	1515.979
VCV.(Intercept).(Intercept)	872619.22	1553860.7587	2192525.972641	3273046.1511	7960320.959
sigma2	1498.57	2087.9652	2526.796184	3110.9213	4696.756
Deviance	314.54	317.3766	319.773775	322.9391	330.918

Table 2 MCMCRegress Results