STA3105 Bayesian Statistics

Jeong Geonwoo

DUE Friday, November 17

We will implement MCMC algorithm for a Bayesian multiple linear regression as

$$Y_i = \beta_0 + \beta_1 X_{1,i} + \beta_2 X_{2,i} + \beta_3 X_{1,i} X_{2,i} + \epsilon_i$$

where $\epsilon_i \sim N\left(0, \sigma^2\right)$ independently for $i = 1, \cdots, 300,000$. We will use independent priors as

$$\beta_j \sim N(0, 10) \text{ for } j = 0, 1, 2, 3$$
$$\sigma^2 \sim \text{IG}(0.01, 0.01)$$

Here, inverse Gamma distribution with shape α, scale β is defined as $p(\sigma) = \frac{\beta^\alpha}{\Gamma(\alpha)}(1/\sigma)^{\alpha+1} \exp(-\beta/\sigma)$ for $\sigma \in (0, \infty)$

1. (10 points) Simulate the dataset as follows.

(a) Let $X_{j,i}$ be the j th predictor for an i th observation. For $i = 1, \cdots, 300,000$ and $j = 1, 2$, simulate $X_{j,i} \sim N(0, 1)$ independently.

(b) For $i = 1, \cdots, 300,000$, simulate Y_i from the above model. Set the true parameter values as $(\beta_0, \beta_1, \beta_2, \beta_3) = (0.5, 1, 2, -1)$ and $\sigma^2 = 1$.

```r
###### Q1 ######
### (a) ###
n = 300000
X <- mvrnorm(n =n, mu = rep(0, 2), Sigma = diag(2))

### (b) ###
beta.true <- c(0.5, 1, 2, -1)
X_design <- cbind( rep(1,n), X, X[,1]*X[,2] )
Y <- X_design %*% beta.true + rnorm(n, 0, 1)
```

First, I generated $n = 300,000$ random variables from $N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$. To create the design matrix, I concated the one vector and $X1X2$ vector to the left and right of X, repectively. Then I generated Y, which follows the given linear regression.

2. (30 points) Implement the MCMC algorithm for the simulated dataset in Problem 1. You may use Gibbs sampler or MH sampler up to your convenience. Here you should write down the algorithm through the R code. Report the trace plots, density plots, 95% HPD intervals, posterior mean, acceptance probability, and effective sample size for all parameters. Check whether your MCMC samples can recover the true $(\beta_0, \beta_1, \beta_2, \beta_3) = (0.5, 1, 2, -1)$ well.

For β, the kernel of posterior for Gibbs sampler is following :

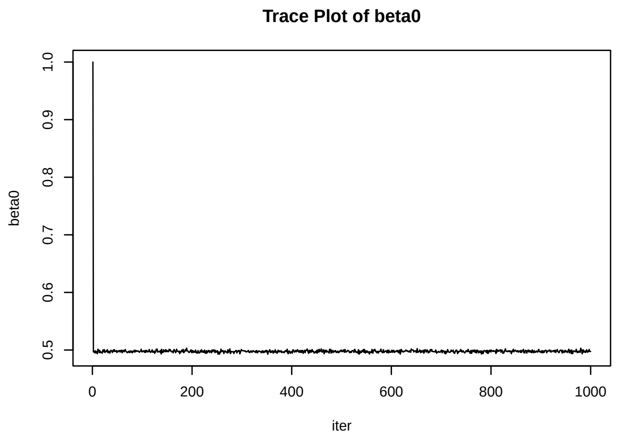
$$
\begin{aligned}
P\left(\beta \mid Y, X, \sigma^2\right) &\propto L\left(Y, X \mid \beta, \sigma^2\right) P(\beta) \\
&\propto \left(\frac{1}{\sqrt{2\pi}}\right)^n \left(\frac{1}{det(\sigma^2 I)}\right)^{\frac{1}{2}} \exp\left(-\frac{(Y-X\beta)^\top (Y-X\beta)}{2\sigma^2}\right) \left(\frac{1}{\sqrt{2\pi}}\right)^p \left(\frac{1}{det(10I)}\right)^{\frac{1}{2}} \exp\left(-\frac{\beta^\top\beta}{20}\right) \\
&\propto \exp\left(-\frac{(Y-X\beta)^\top (Y-X\beta)}{2\sigma^2} - \frac{\beta^\top\beta}{20}\right) \\
&= \exp\left(-\frac{1}{2\sigma^2}Y^\top Y + \frac{1}{2\sigma^2}Y^\top X\beta + \frac{1}{2\sigma^2}\beta^\top XY - \frac{\beta^\top X^\top X\beta}{2\sigma^2} - \frac{\beta^\top\beta}{20}\right) \\
&= \exp\left(-\frac{1}{2}\left(\beta^\top\left(\frac{X^\top X}{\sigma^2} + \frac{I}{10}\right)\beta - \beta^\top\left(\frac{X^\top Y}{\sigma^2}\right) - \left(\frac{Y^\top X}{\sigma^2}\right)\beta + \frac{Y^\top Y}{6^2}\right)\right) \\
&\therefore V_\beta = \left[\frac{X^\top X}{\sigma^2} + \frac{I}{10}\right]^{-1}, \quad m_\beta = V_\beta \frac{X^\top Y}{\sigma^2} \\
&\beta \sim N(m_\beta, V_\beta)
\end{aligned}
$$

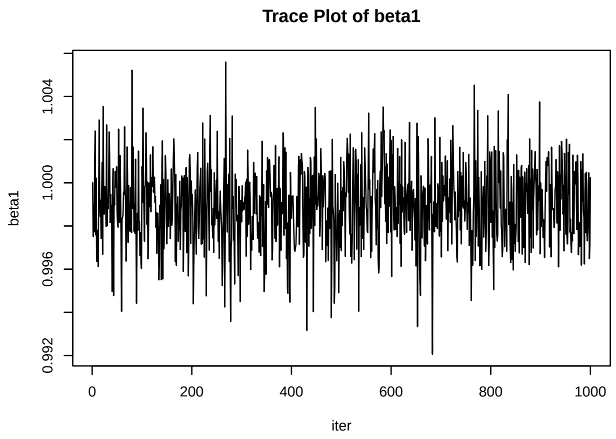
For σ^2, the kernel of posterior for Gibbs sampler is following :

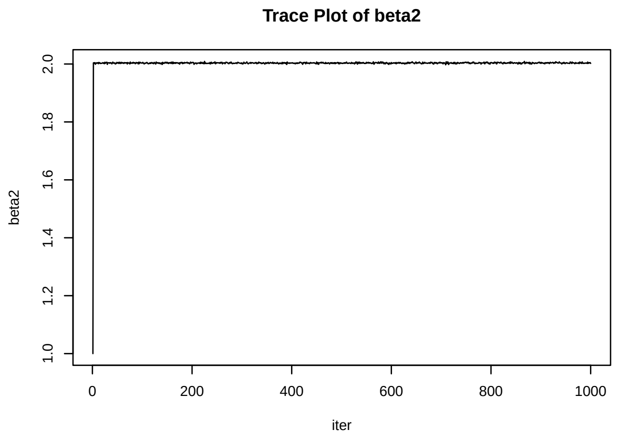
$$
\begin{aligned}
P\left(\sigma^2 \mid, \beta\right) &\propto L\left(Y, X \mid \beta, \sigma^2\right) P(\sigma^2) \\
&\propto \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n \exp\left(-\frac{(Y-X\beta)^\top (Y-X\beta)}{2\sigma^2}\right) \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma^2}\right)^{\alpha+1} \exp(-\frac{\beta}{\sigma^2}) \\
&\propto \left(\frac{1}{\sigma^2}\right)^{\frac{n}{2}+\alpha+1} \exp\left(-\frac{\beta + \frac{1}{2}(Y-X\beta)^\top (Y-X\beta)}{\sigma^2}\right) \\
&\therefore \alpha_{\sigma^2} = \frac{n}{2} + \alpha, \quad \beta_{\sigma^2} = \beta + \frac{1}{2}(Y-X\beta)^\top (Y-X\beta) \\
&\sigma^2 \sim IG(\alpha_{\sigma^2}, \beta_{\sigma^2})
\end{aligned}
$$

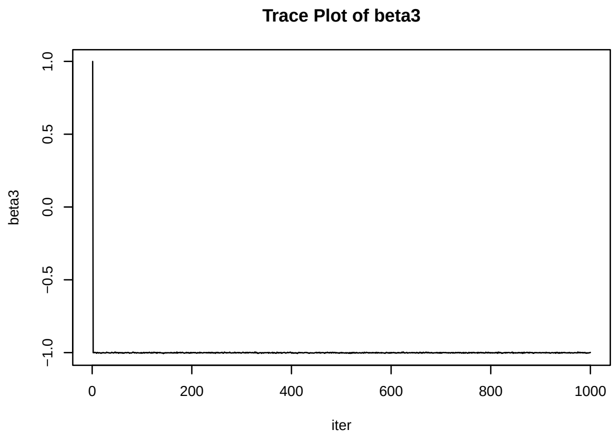
```
##### Q2 #####
p = ncol(X_design)
m.beta <- 0; v.beta <- 10
a.s2 <- 0.01; b.s2 <- 0.01

B = 1000

# sample vector (with initialization)
beta.samps <- matrix(NA, nrow = p, ncol = B)
beta.samps[,1] <- rep(1,p)
s2.samps <- matrix(NA, nrow = 1, ncol = B)
s2.samps[1] <- 1

# Gibbs sampler
ptm <- proc.time()
for(i in 2:B){

  ## beta[i] | s2[i-1]
  V <- solve( t(X_design) %*% X_design / s2.samps[i-1] + diag(p) / v.beta )
  m <- V %*% ( t(X_design) %*% Y / s2.samps[i-1] )
  beta.samps[,i] <- rmvnorm(1, mean = m, sigma = V, method = "svd")

  # s2[i] | beta[i]
  a <- n/2 + a.s2
  b <- b.s2 + t( Y - X_design %*% beta.samps[,i] ) %*% ( Y - X_design %*% beta.samps[,i] ) / 2
  s2.samps[i] <- rinvgamma(1, shape=a, scale=b)
  s2.samps <- t(s2.samps)
}
Rtime <- proc.time() - ptm
```

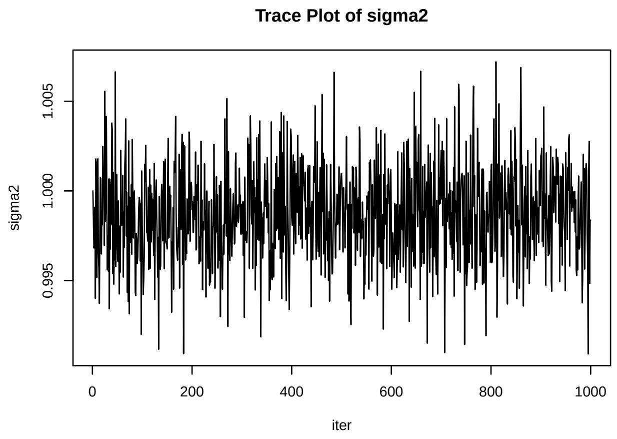
```
# ts plot (For checking the burn-in size I will do)
par(mfrow=c(3,2))
ts.plot(beta.samps[1,], main="Trace Plot of beta0", xlab="iter", ylab="beta0")
ts.plot(beta.samps[2,], main="Trace Plot of beta1", xlab="iter", ylab="beta1")
ts.plot(beta.samps[3,], main="Trace Plot of beta2", xlab="iter", ylab="beta2")
ts.plot(beta.samps[4,], main="Trace Plot of beta3", xlab="iter", ylab="beta3")
ts.plot(s2.samps, main="Trace Plot of sigma2", xlab="iter", ylab="sigma2")
plot.new()
```











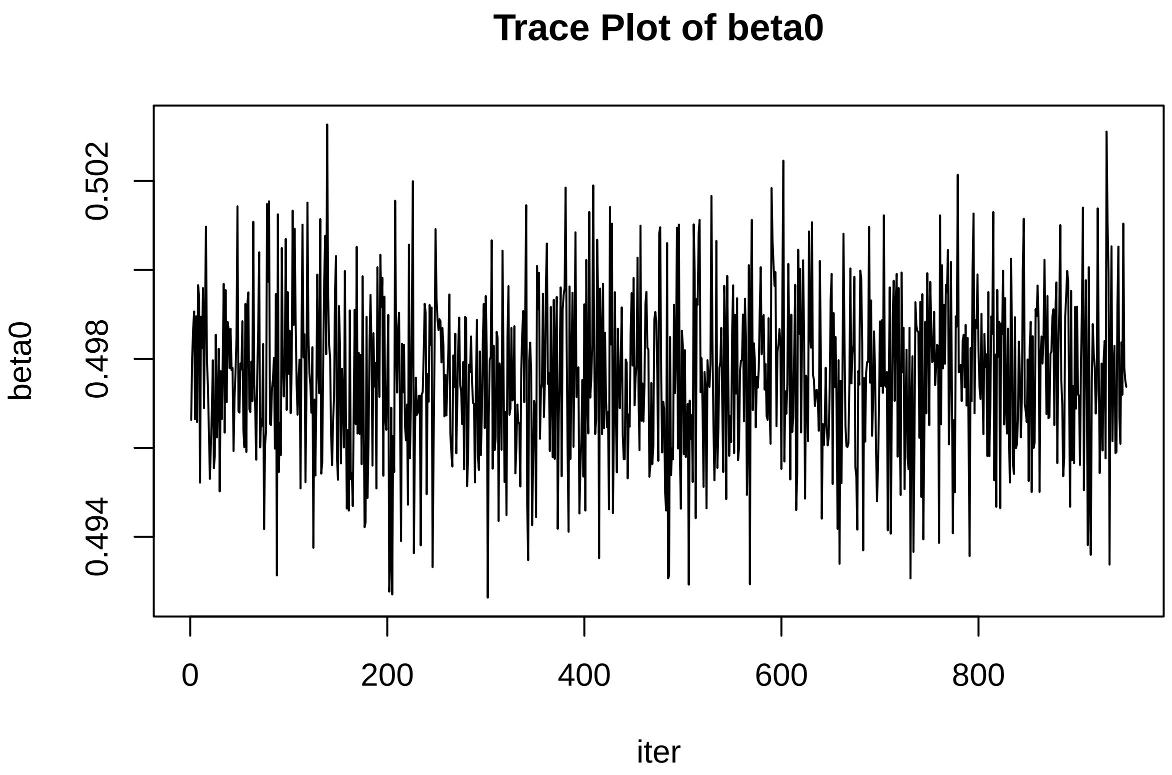
For β_1 and σ^2, (where the initial value was similar to the convergence of the sample obtained through the Gibbs sampler,) an appropriate trace plot was plotted, whereas for the other parameters, burn-in was required. However, since we used Gibbs sampler, I will use a small-size (50-size) burn-in.

```r
# burn-in
beta.samps <- beta.samps[,51:B]
s2.samps <- s2.samps[51:B]
```

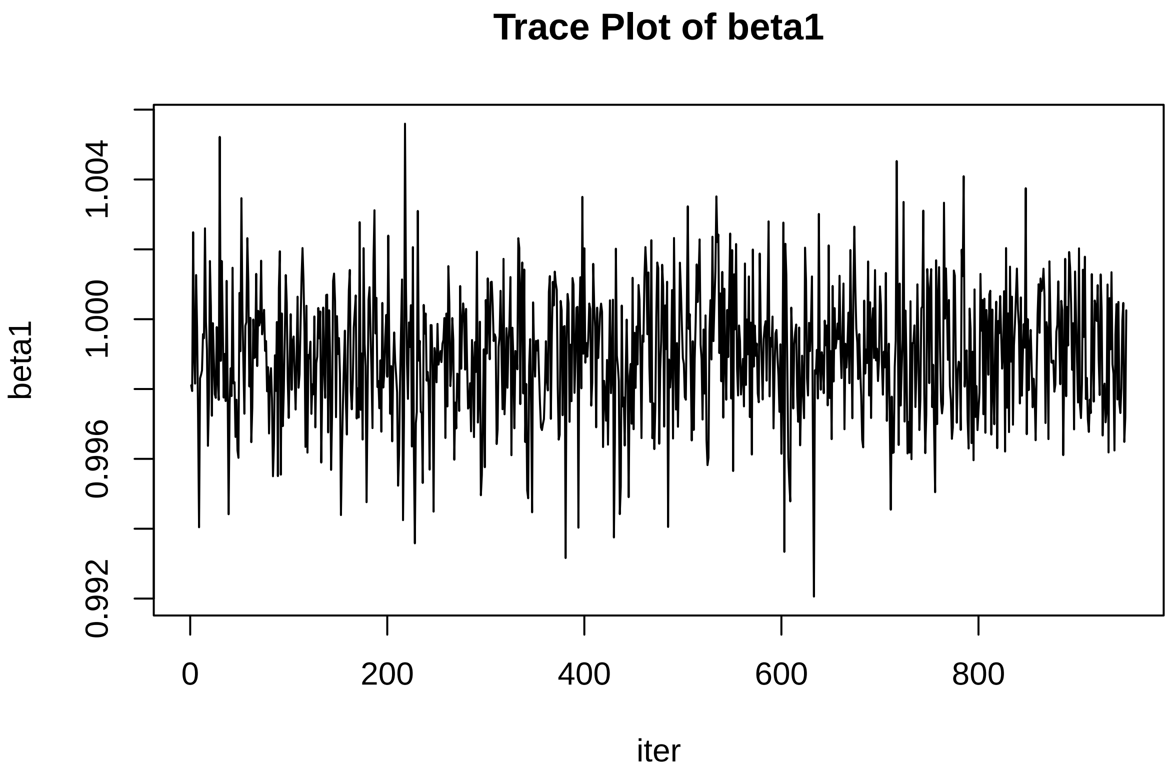
I will check the **trace plot** again.

```r
# ts plot (after burn-in)
par(mfrow=c(1,1))
```

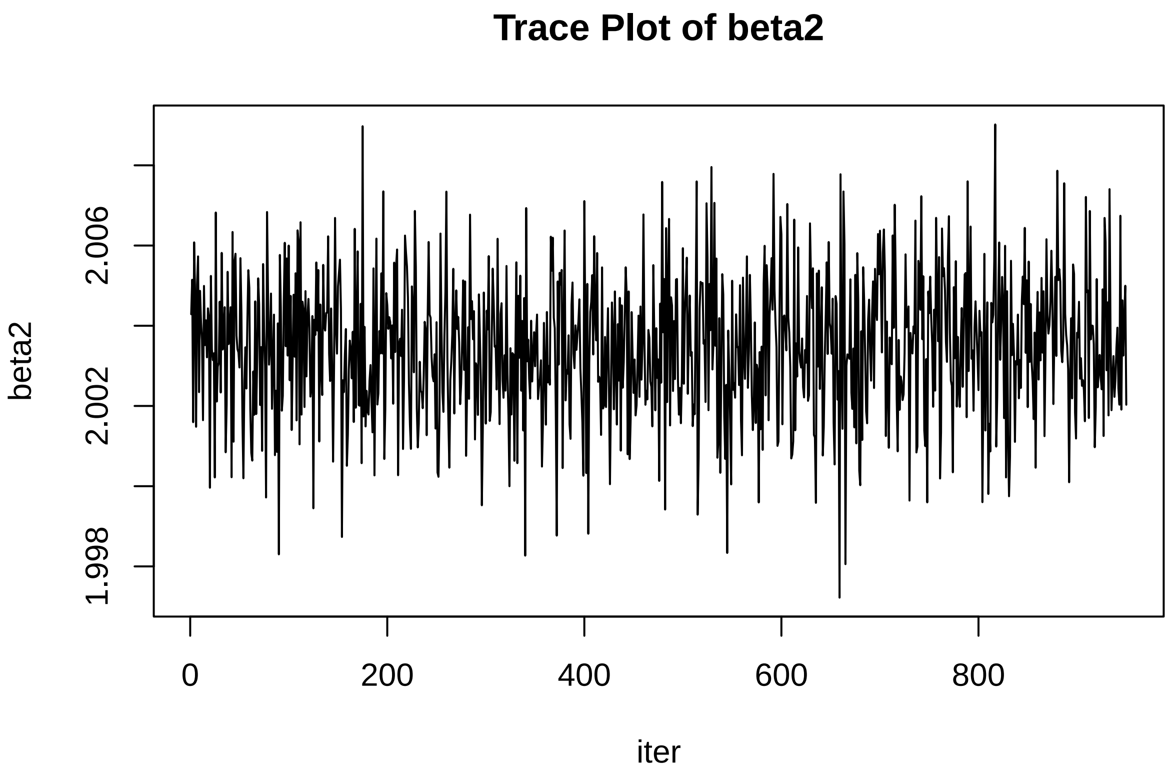
```r
ts.plot(beta.samps[1,], main="Trace Plot of beta0", xlab="iter", ylab="beta0")
```



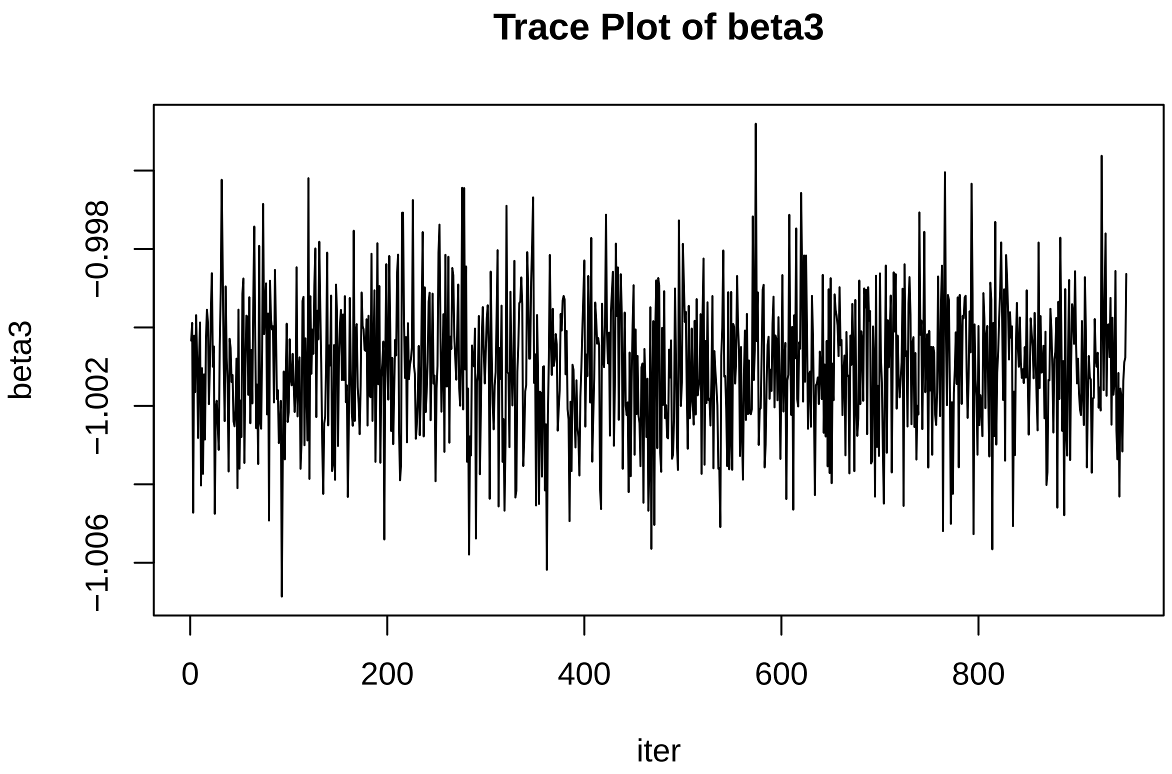
```r
ts.plot(beta.samps[2,], main="Trace Plot of beta1", xlab="iter", ylab="beta1")
```



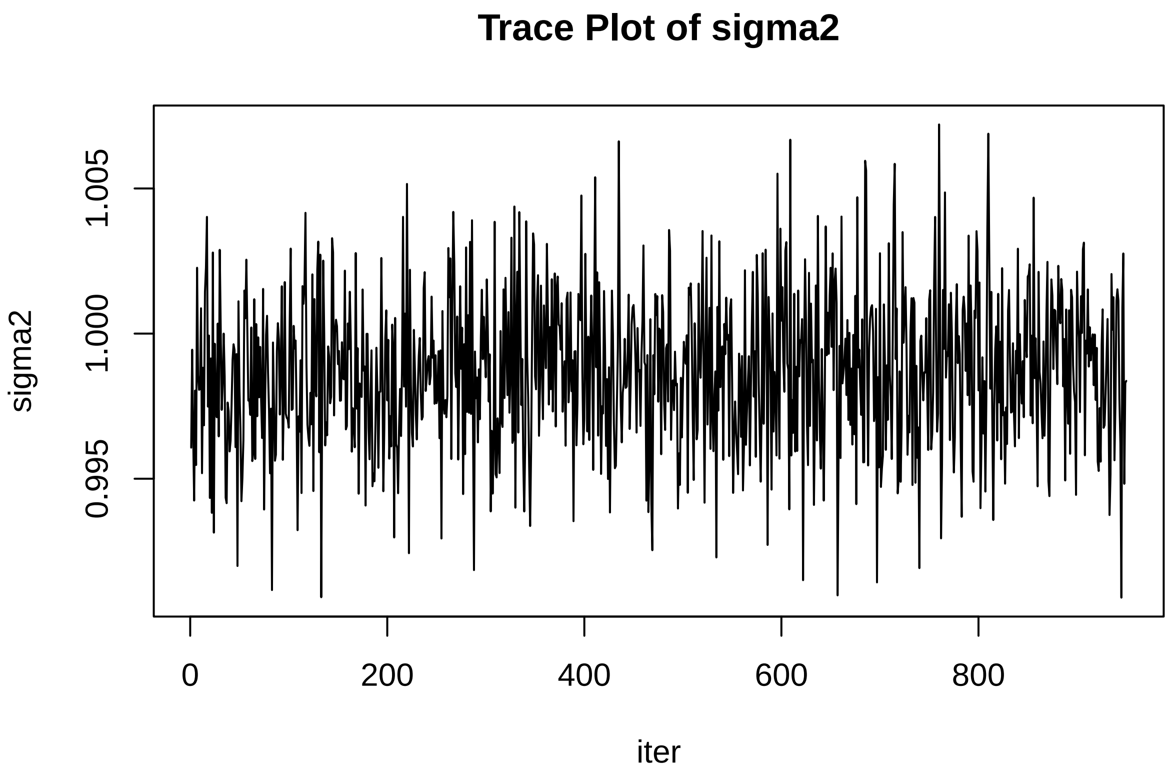
```r
ts.plot(beta.samps[3,], main="Trace Plot of beta2", xlab="iter", ylab="beta2")
```



```r
ts.plot(beta.samps[4,], main="Trace Plot of beta3", xlab="iter", ylab="beta3")
```



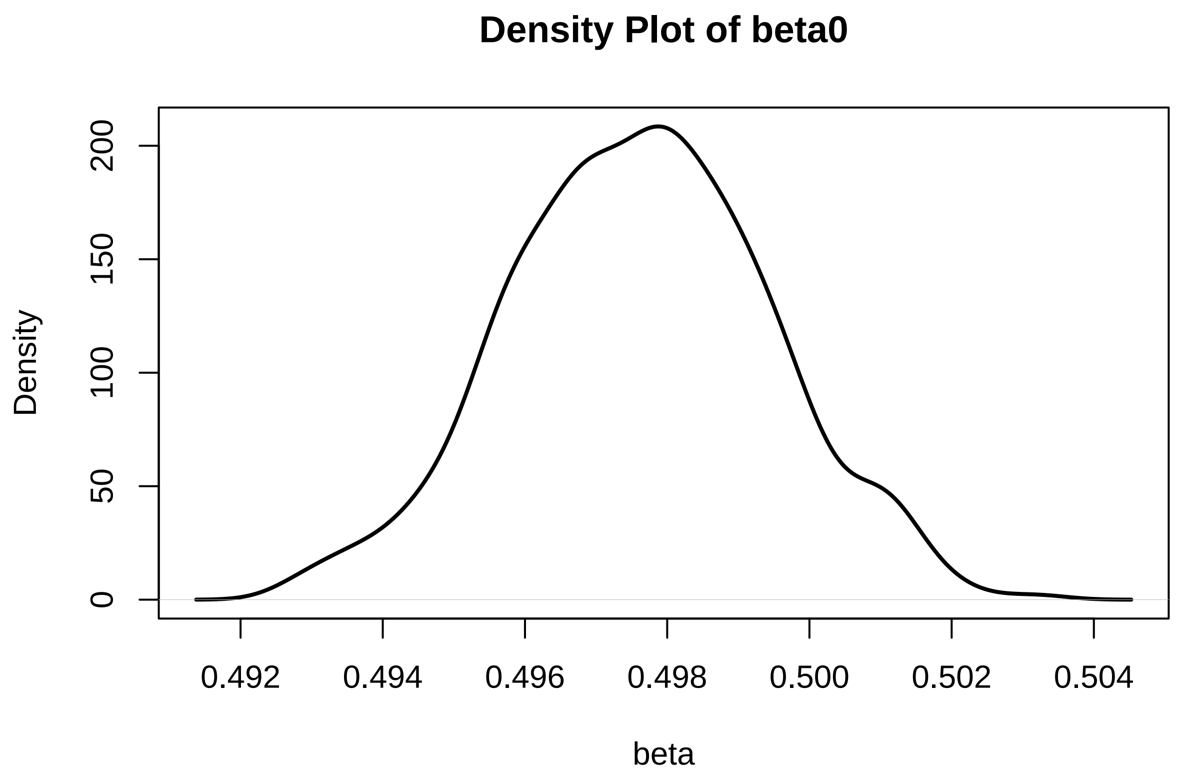
```r
ts.plot(s2.samps, main="Trace Plot of sigma2", xlab="iter", ylab="sigma2")
```



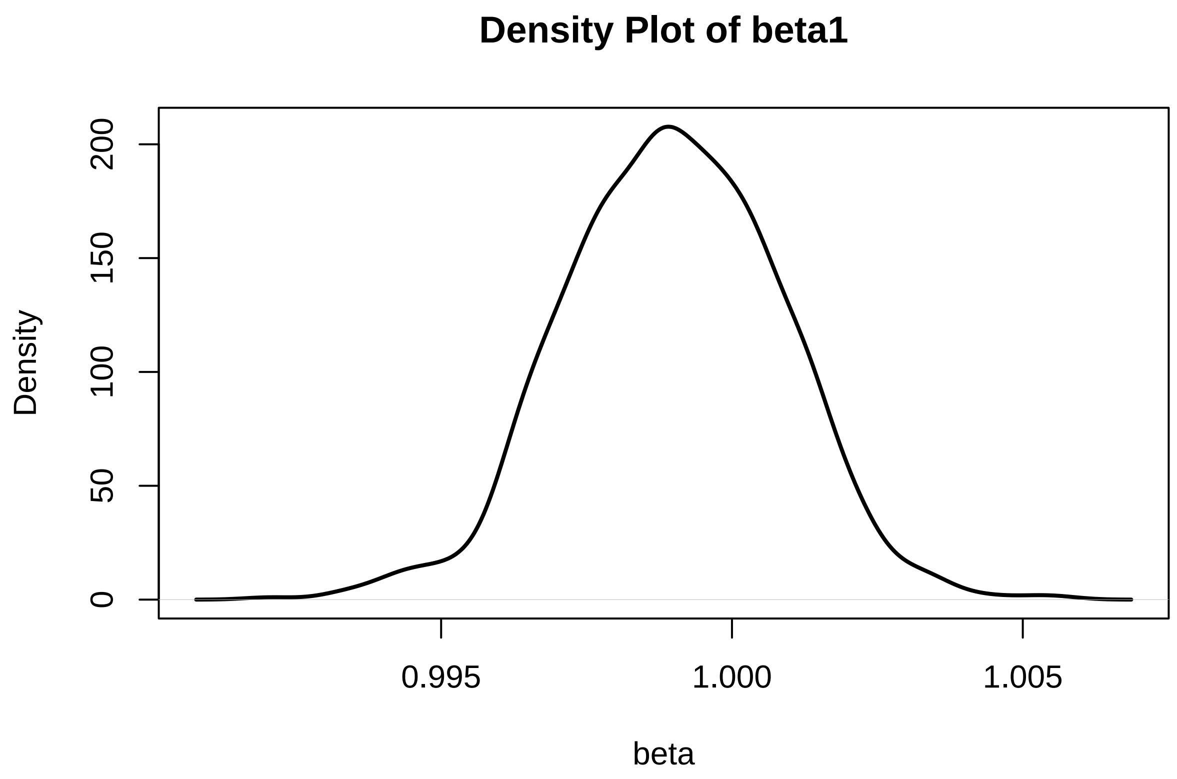
The burn in removed samples that were affected by the initial values, so a proper trace plot was drawn.

Next, I will check the **density plot**

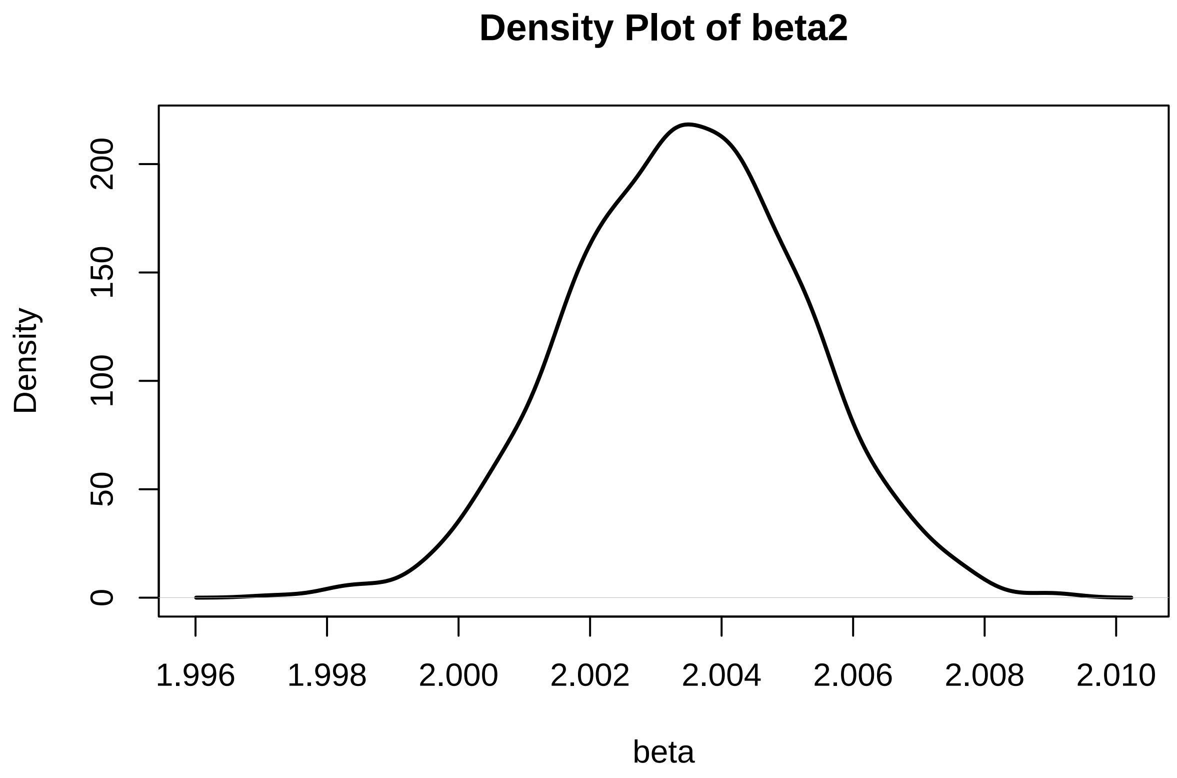
```r
plot(density(beta.samps[1,]), main="Density Plot of beta0", xlab="beta", ylab="Density", lwd=2)
```



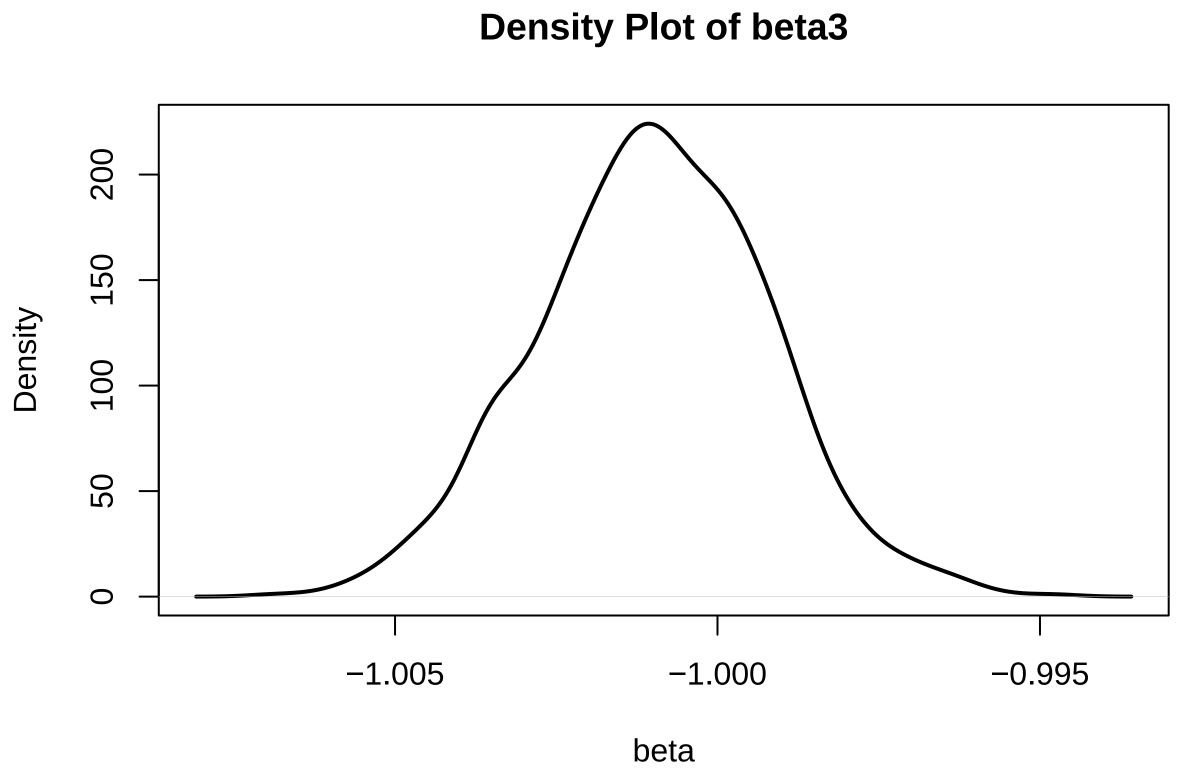
```r
plot(density(beta.samps[2,]), main="Density Plot of beta1", xlab="beta", ylab="Density", lwd=2)
```



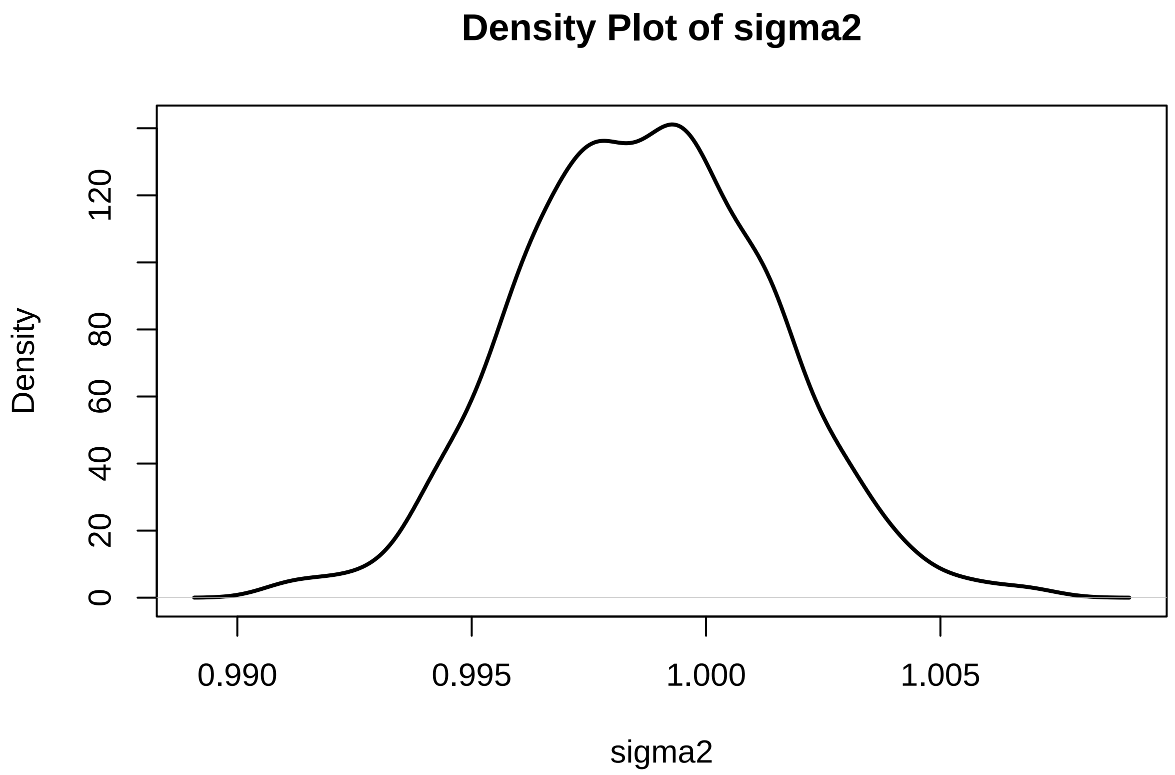
```r
plot(density(beta.samps[3,]), main="Density Plot of beta2", xlab="beta", ylab="Density", lwd=2)
```



```r
plot(density(beta.samps[4,]), main="Density Plot of beta3", xlab="beta", ylab="Density", lwd=2)
```



```r
plot(density(s2.samps), main="Density Plot of sigma2", xlab="sigma2", ylab="Density", lwd=2)
```



Next, I will print the 95% HPD intervals, posterior mean, acceptance probability, and effective sample size.

```r
# 95% HPD intervals
cat("95% HPD intervals of beta0:", HPDinterval(as.mcmc(beta.samps[1,]), prob = 0.95), "\n",
    "95% HPD intervals of beta1:", HPDinterval(as.mcmc(beta.samps[2,]), prob = 0.95), "\n",
    "95% HPD intervals of beta2:", HPDinterval(as.mcmc(beta.samps[3,]), prob = 0.95), "\n",
    "95% HPD intervals of beta3:", HPDinterval(as.mcmc(beta.samps[4,]), prob = 0.95), "\n",
    "95% HPD intervals of sigma2:", HPDinterval(as.mcmc(s2.samps), prob = 0.95) )
```

```
## 95% HPD intervals of beta0: 0.4944066 0.5015578
##  95% HPD intervals of beta1: 0.995502 1.002453
##  95% HPD intervals of beta2: 2.000098 2.006862
##  95% HPD intervals of beta3: -1.004724 -0.9978144
##  95% HPD intervals of sigma2: 0.993538 1.003615
```

```r
# posterior mean
cat("posterior mean of beta0:", mean(beta.samps[1,]), "\n",
    "posterior mean of beta1:", mean(beta.samps[2,]), "\n",
    "posterior mean of beta2:", mean(beta.samps[3,]), "\n",
    "posterior mean of beta3:", mean(beta.samps[4,]), "\n",
    "posterior mean of sigma2:", mean(s2.samps) )
```

```
## posterior mean of beta0: 0.4976068
##  posterior mean of beta1: 0.9989775
##  posterior mean of beta2: 2.00346
##  posterior mean of beta3: -1.00101
##  posterior mean of sigma2: 0.9986912
```

```r
# acceptance probability
# (Because I used Gibbs sampler, samples was always accepted !)
cat("Acceptance probability of beta0 :", length(unique(beta.samps[1,]))/B ,"\n",
    "Acceptance probability of beta1 :", length(unique(beta.samps[2,]))/B ,"\n",
    "Acceptance probability of beta2 :", length(unique(beta.samps[3,]))/B ,"\n",
    "Acceptance probability of beta3 :", length(unique(beta.samps[4,]))/B ,"\n",
    "Acceptance probability of sigma2 :", length(unique(s2.samps))/B )
```

```
## Acceptance probability of beta0 : 0.95
##  Acceptance probability of beta1 : 0.95
##  Acceptance probability of beta2 : 0.95
##  Acceptance probability of beta3 : 0.95
##  Acceptance probability of sigma2 : 0.95
```

```r
# effective sample size
cat("Effective Sample size of beta0:", effectiveSize(beta.samps[1,]), "\n",
    "Effective Sample size of beta1:", effectiveSize(beta.samps[2,]), "\n",
    "Effective Sample size of beta2:", effectiveSize(beta.samps[3,]), "\n",
    "Effective Sample size of beta3:", effectiveSize(beta.samps[4,]), "\n",
    "Effective Sample size of sigma2:", effectiveSize(s2.samps) )
```

```
## Effective Sample size of beta0: 950
##  Effective Sample size of beta1: 950
##  Effective Sample size of beta2: 950
##  Effective Sample size of beta3: 950
##  Effective Sample size of sigma2: 950
```

Since I used the Gibbs sampler, all samples are accepted and the acceptance probability for all parameters is 0.95 (0.05 is due to burn-in). For the same reason, the effective sample size is 950.

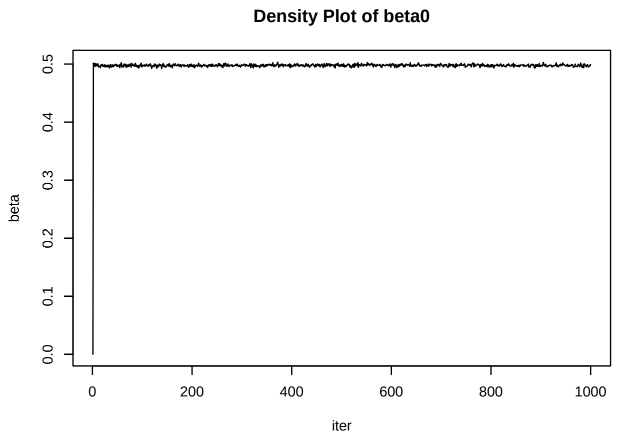
For the β_2, 95% HPD interval do not recover the true $\beta_2 = 2$. However, in that the reason is the interval of Gibbs sampler is very narrow and is very close to true $\beta_2 = 2$, we can not say β_2 is sampled incorrectly.

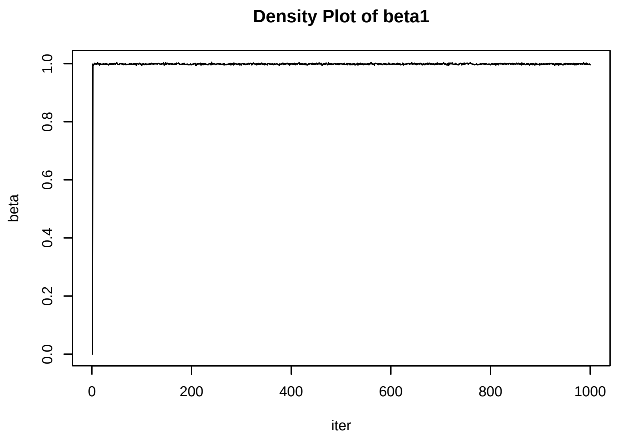
3. (60 points) Implement the same MCMC algorithm from the model specified above. But here, you should write down the C + + code using RcppArmadillo library as follows. Report the trace plots, density plots, 95% HPD intervals, posterior mean, acceptance probability, and effective sample size for all parameters. Check whether your MCMC samples can recover the true $(\beta_0, \beta_1, \beta_2, \beta_3) = (0.5, 1, 2, -1)$ well. Furthermore, compare the posterior densities obtained from Problem 2 and Problem 3. They should provide the same posterior densities.

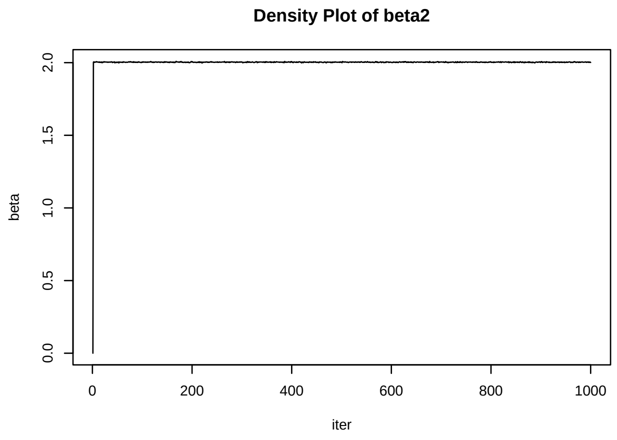
```r
library(Rcpp); library(RcppArmadillo)
setwd("/Users/gunwoojung/Desktop/R_wd/Bayesian_Stat")
sourceCpp("HW5_JeongGeonwoo_2018122062.cpp")
```

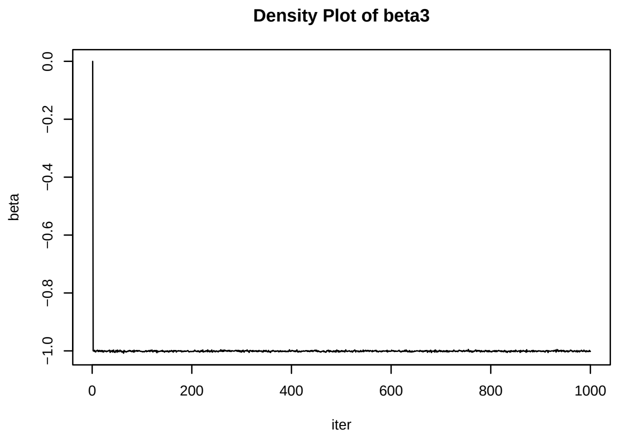
```r
ptm <- proc.time()
MCMC_Rcpp <- RcppGibbs(B, X_design, Y,  m.beta, v.beta, a.s2, b.s2)
Rcpptime <- proc.time() - ptm
beta_samps <- MCMC_Rcpp$beta_samps
s2_samps <- MCMC_Rcpp$s2_samps
```

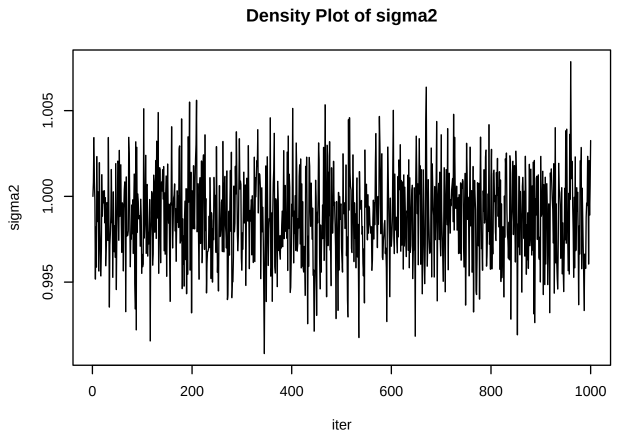
```r
# ts plot (For checking the burn-in size I will do)
par(mfrow=c(3,2))
ts.plot(beta_samps[1,], main="Density Plot of beta0", xlab="iter", ylab="beta")
ts.plot(beta_samps[2,], main="Density Plot of beta1", xlab="iter", ylab="beta")
ts.plot(beta_samps[3,], main="Density Plot of beta2", xlab="iter", ylab="beta")
ts.plot(beta_samps[4,], main="Density Plot of beta3", xlab="iter", ylab="beta")
ts.plot(s2_samps, main="Density Plot of sigma2", xlab="iter", ylab="sigma2")
plot.new()
```











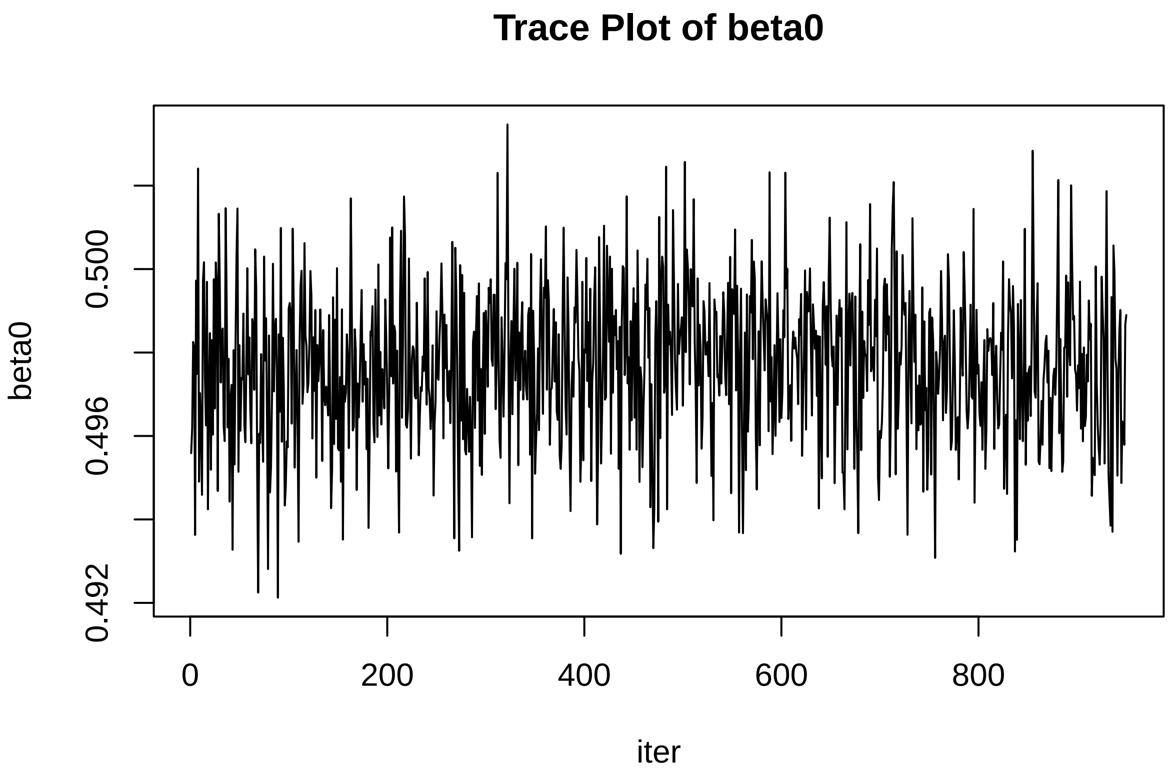
For σ^2, (where the initial value was similar to the convergence of the sample obtained through the Gibbs sampler,) an appropriate trace plot was plotted, whereas for β, burn-in was required. However, since the convergence speed of Gibbs sampler is fast, I will use a small-size (50-size) burn-in.

```
# burn-in
beta_samps <- beta_samps[,51:B]
s2_samps <- s2_samps[51:B]
```

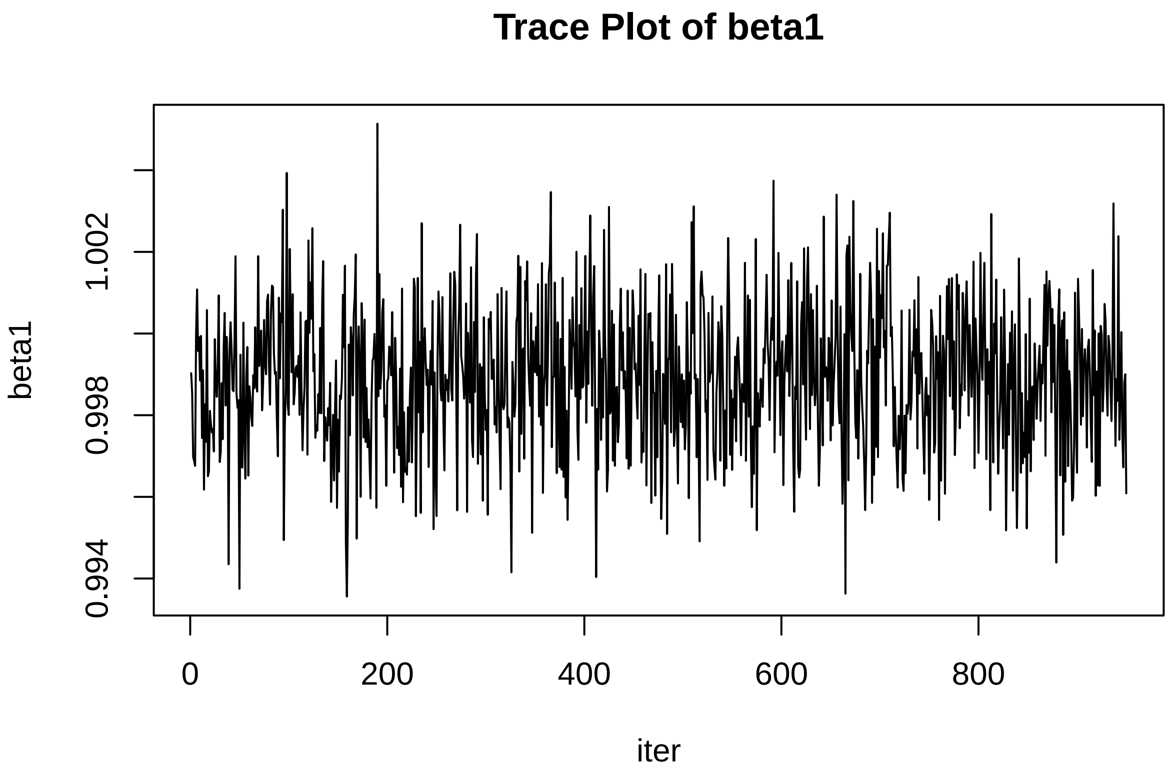
I will check the **trace plot** again.

```
# ts plot (after burn-in)
par(mfrow=c(1,1))
```

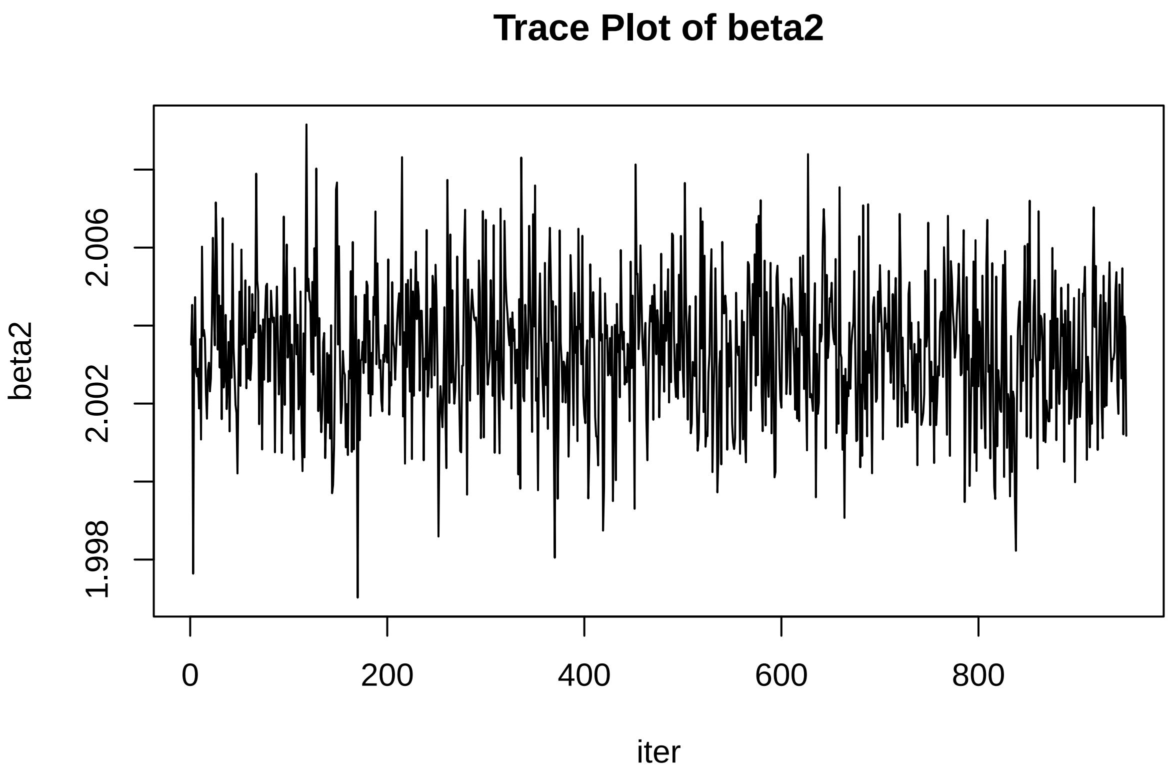
```r
ts.plot(beta_samps[1,], main="Trace Plot of beta0", xlab="iter", ylab="beta0")
```



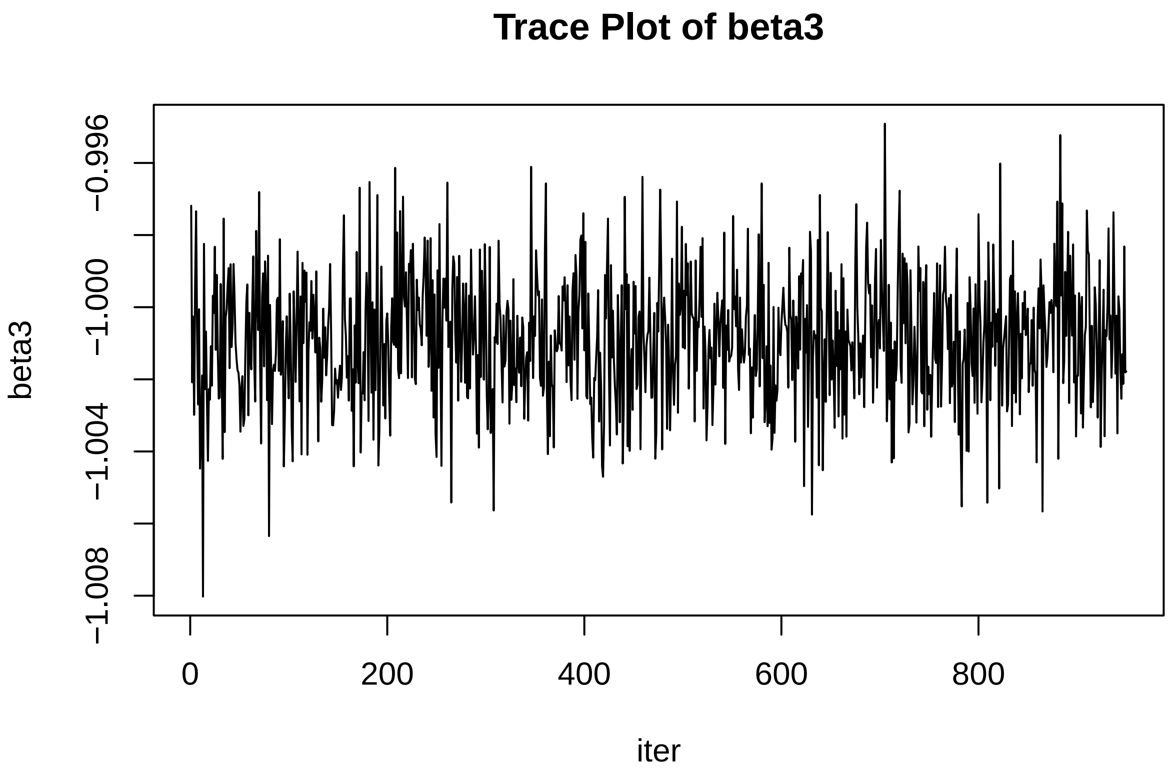
```r
ts.plot(beta_samps[2,], main="Trace Plot of beta1", xlab="iter", ylab="beta1")
```



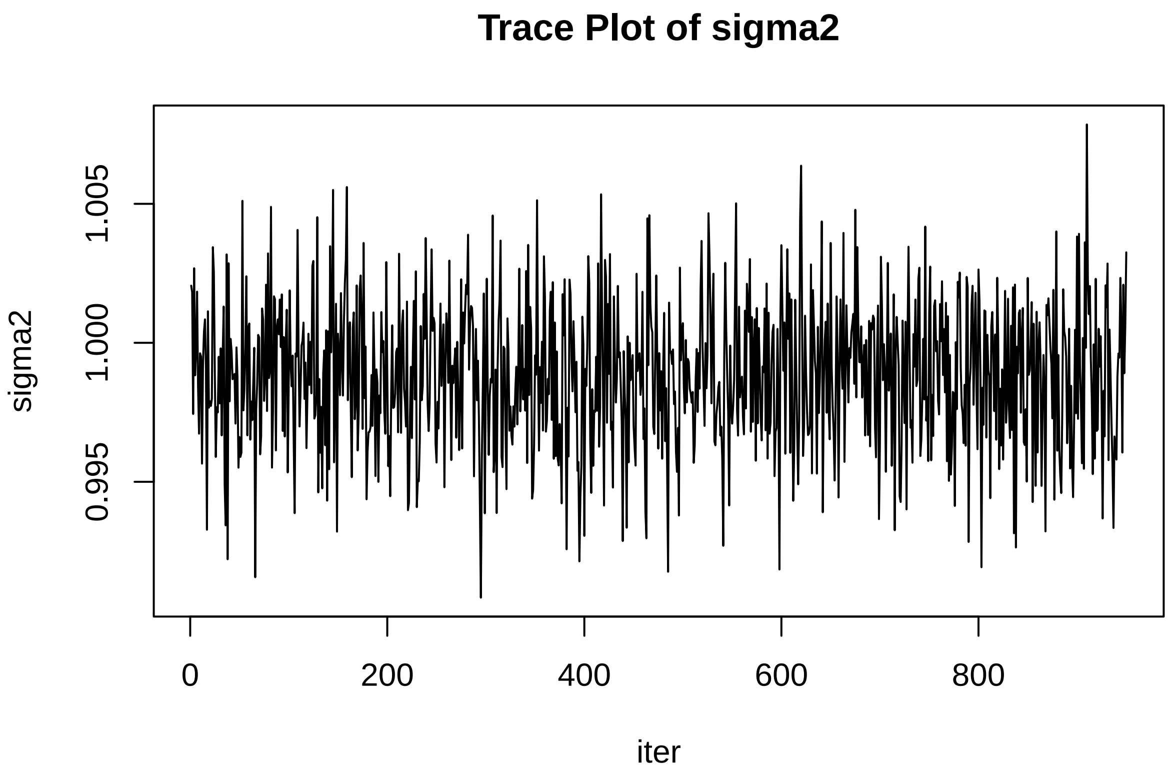
```r
ts.plot(beta_samps[3,], main="Trace Plot of beta2", xlab="iter", ylab="beta2")
```



```r
ts.plot(beta_samps[4,], main="Trace Plot of beta3", xlab="iter", ylab="beta3")
```



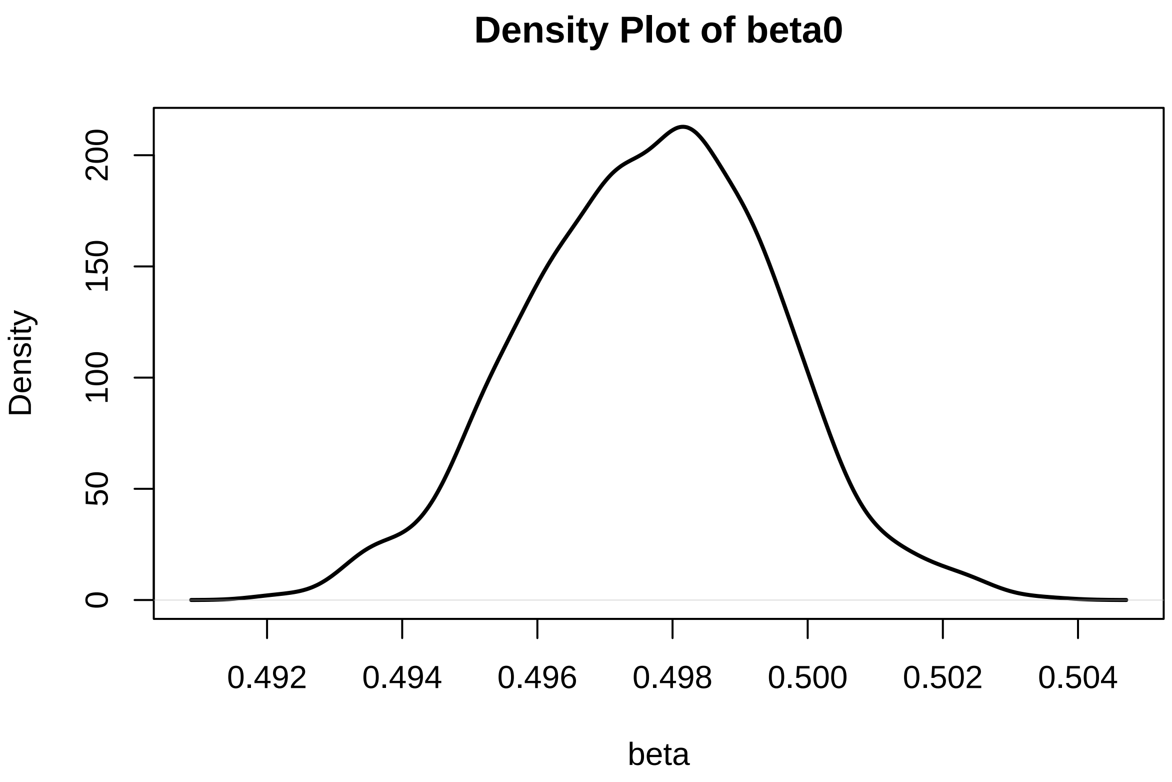
```r
ts.plot(s2_samps, main="Trace Plot of sigma2", xlab="iter", ylab="sigma2")
```



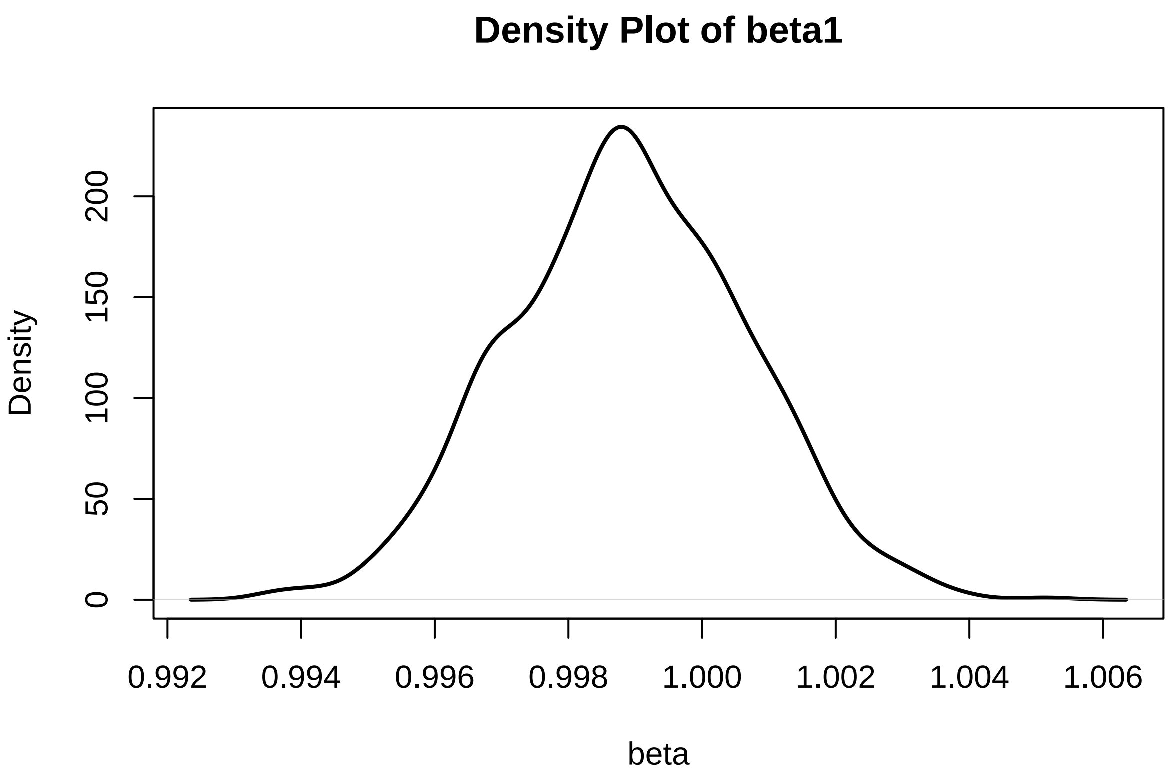
The burn in removed samples that were affected by the initial values, so a proper trace plot was drawn.

Next, I will check the **density plot**

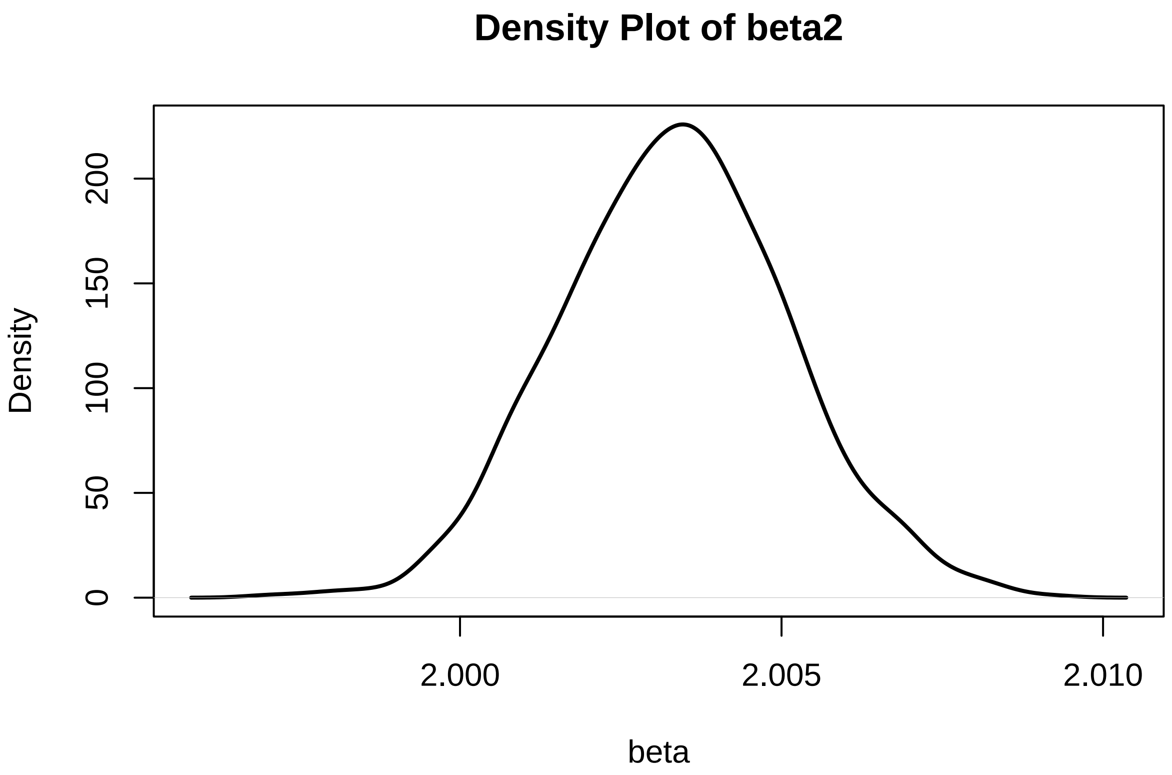
```r
# density plot
plot(density(beta_samps[1,]), main="Density Plot of beta0", xlab="beta", ylab="Density", lwd=2)
```



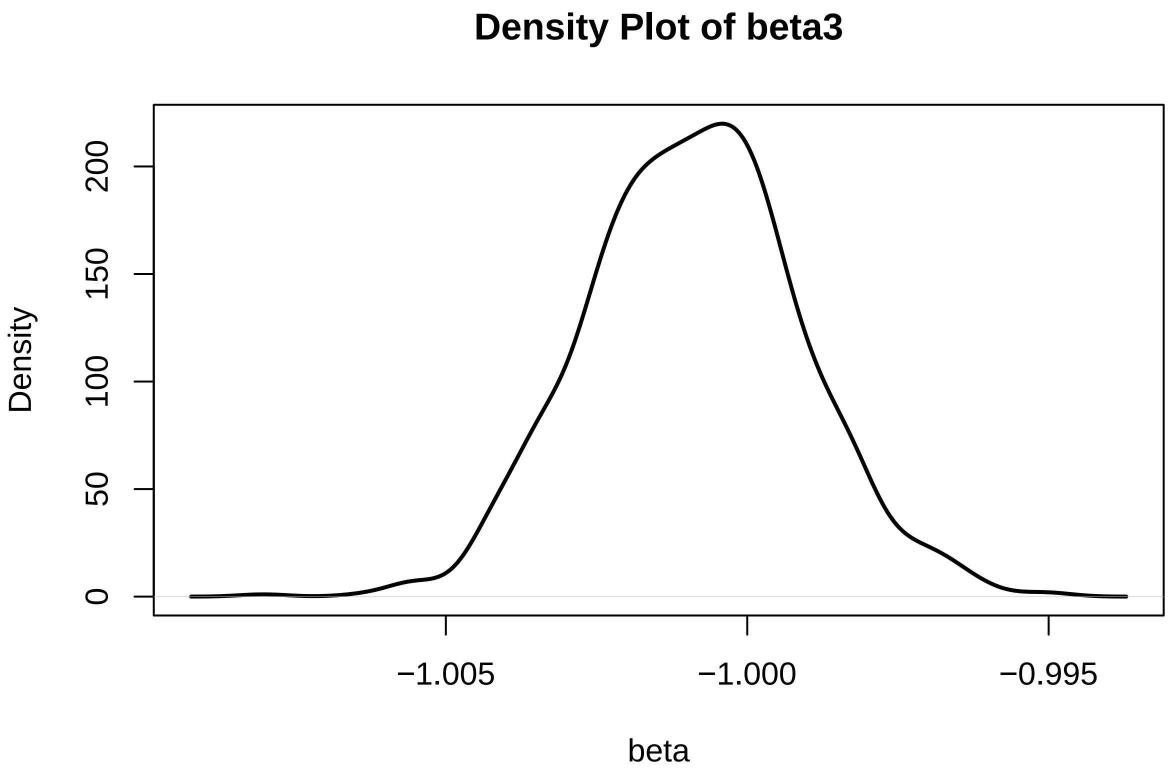
```r
plot(density(beta_samps[2,]), main="Density Plot of beta1", xlab="beta", ylab="Density", lwd=2)
```



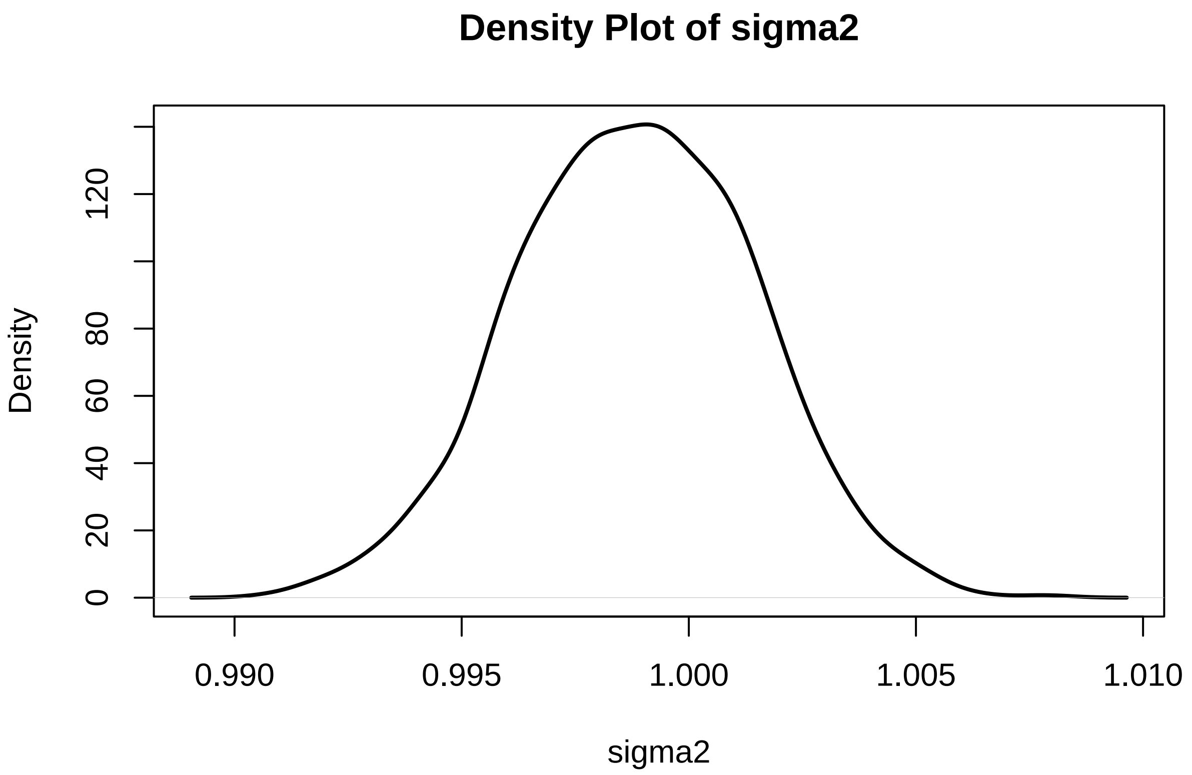
```r
plot(density(beta_samps[3,]), main="Density Plot of beta2", xlab="beta", ylab="Density", lwd=2)
```



```r
plot(density(beta_samps[4,]), main="Density Plot of beta3", xlab="beta", ylab="Density", lwd=2)
```



```r
plot(density(s2_samps), main="Density Plot of sigma2", xlab="sigma2", ylab="Density", lwd=2)
```



Next, I will print the 95% HPD intervals, posterior mean, acceptance probability, and effective sample size.

```r
# 95% HPD intervals
cat("95% HPD intervals of beta0:", HPDinterval(as.mcmc(beta_samps[1,]), prob = 0.95), "\n",
    "95% HPD intervals of beta1:", HPDinterval(as.mcmc(beta_samps[2,]), prob = 0.95), "\n",
    "95% HPD intervals of beta2:", HPDinterval(as.mcmc(beta_samps[3,]), prob = 0.95), "\n",
    "95% HPD intervals of beta3:", HPDinterval(as.mcmc(beta_samps[4,]), prob = 0.95), "\n",
    "95% HPD intervals of sigma2:", HPDinterval(as.mcmc(s2_samps), prob = 0.95) )
```

```
## 95% HPD intervals of beta0: 0.494241 0.5014507
##  95% HPD intervals of beta1: 0.9954335 1.002159
##  95% HPD intervals of beta2: 2.000183 2.007032
##  95% HPD intervals of beta3: -1.004521 -0.9978164
##  95% HPD intervals of sigma2: 0.9936585 1.003676
```

```r
# posterior mean
cat("posterior mean of beta0:", mean(beta_samps[1,]), "\n",
    "posterior mean of beta1:", mean(beta_samps[2,]), "\n",
    "posterior mean of beta2:", mean(beta_samps[3,]), "\n",
    "posterior mean of beta3:", mean(beta_samps[4,]), "\n",
    "posterior mean of sigma2:", mean(s2_samps) )
```

```
## posterior mean of beta0: 0.4976719
##  posterior mean of beta1: 0.9988919
##  posterior mean of beta2: 2.003348
##  posterior mean of beta3: -1.000892
##  posterior mean of sigma2: 0.9988178
```

```r
# acceptance probability
# (Because I used Gibbs sampler, samples was always accepted !)
cat("Acceptance probability of beta0 :", length(unique(beta_samps[1,]))/B ,"\n",
    "Acceptance probability of beta1 :", length(unique(beta_samps[2,]))/B ,"\n",
    "Acceptance probability of beta2 :", length(unique(beta_samps[3,]))/B ,"\n",
    "Acceptance probability of beta3 :", length(unique(beta_samps[4,]))/B ,"\n",
    "Acceptance probability of sigma2 :", length(unique(s2_samps))/B )
```

```
## Acceptance probability of beta0 : 0.95
##  Acceptance probability of beta1 : 0.95
##  Acceptance probability of beta2 : 0.95
##  Acceptance probability of beta3 : 0.95
##  Acceptance probability of sigma2 : 0.95
```

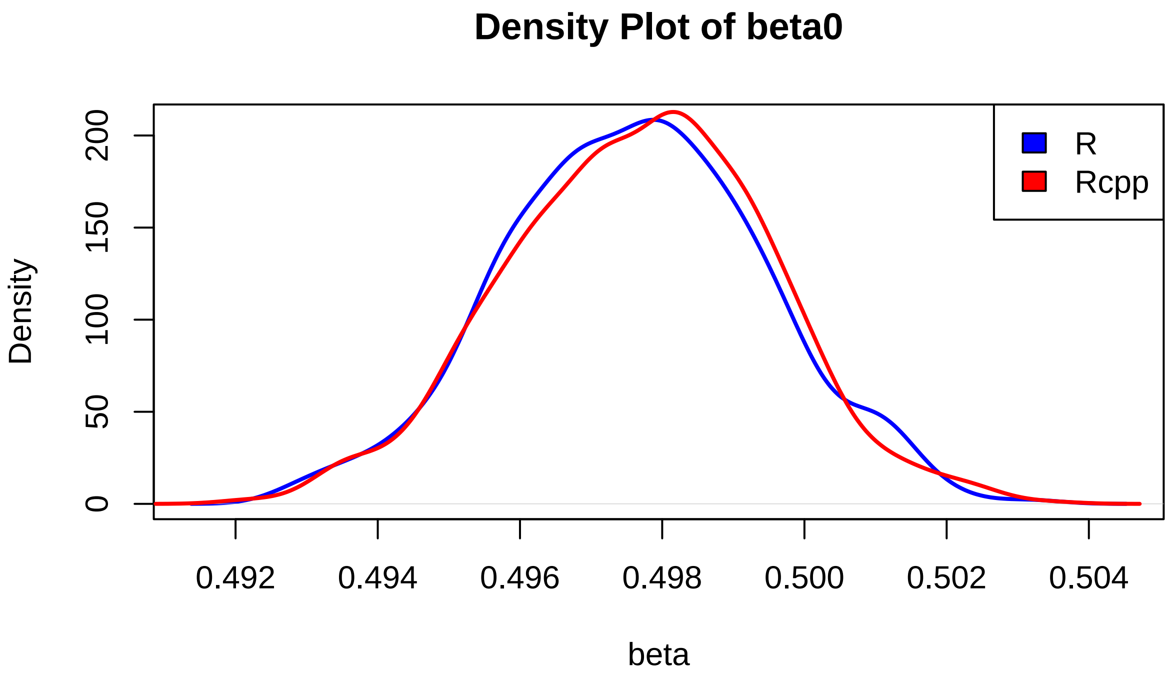
```r
# effective sample size
cat("Effective Sample size of beta0:", effectiveSize(beta_samps[1,]), "\n",
    "Effective Sample size of beta1:", effectiveSize(beta_samps[2,]), "\n",
    "Effective Sample size of beta2:", effectiveSize(beta_samps[3,]), "\n",
    "Effective Sample size of beta3:", effectiveSize(beta_samps[4,]), "\n",
    "Effective Sample size of sigma2:", effectiveSize(s2_samps) )
```

```
## Effective Sample size of beta0: 1205.181
##  Effective Sample size of beta1: 950
##  Effective Sample size of beta2: 950
##  Effective Sample size of beta3: 700.4199
##  Effective Sample size of sigma2: 950
```

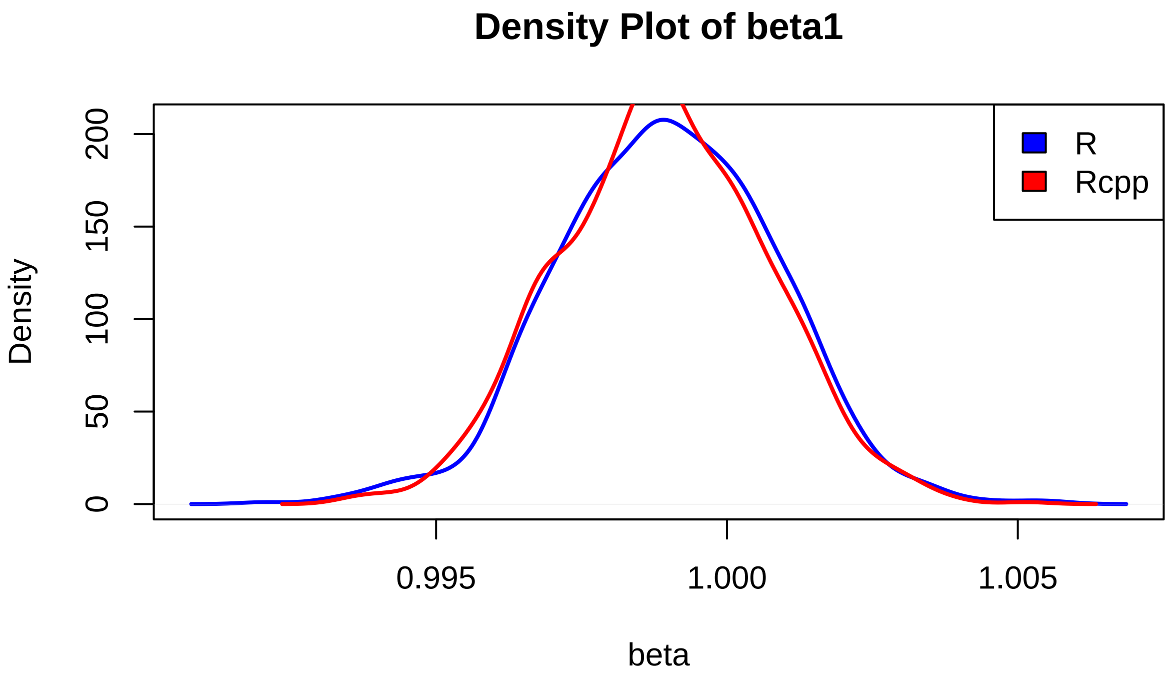
Since I used the Gibbs sampler, all samples are accepted and the acceptance probability for all parameters is 0.95 (0.05 is due to burn-in). The difference between Effective sample size of β_0 1205 and actual sample size of β_0 950 (about 255) is similar to the difference between Effective sample size of β_3 700 and actual sample size of β_3 (about 250). From that we can infer one of the reason could be that β_0 and β_3 is negative correlated.

Then, I will compare the posterior densities of β and σ^2 obtained from Problem 2 and Problem 3.

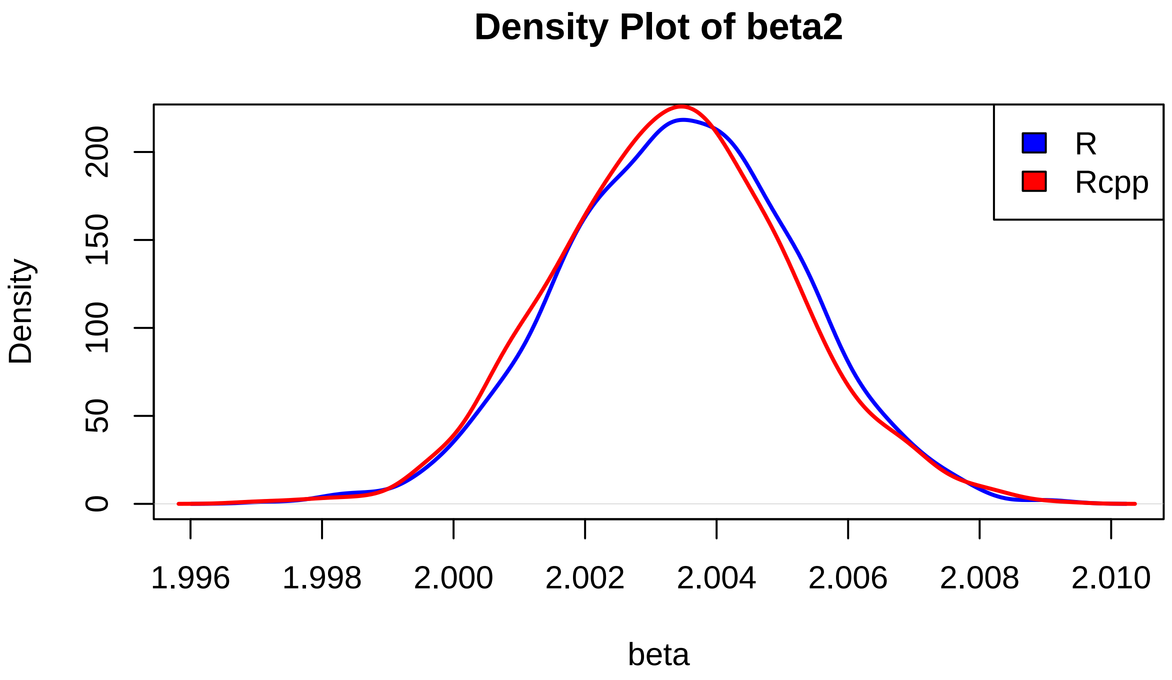
```
plot(density(beta.samps[1,]), main="Density Plot of beta0", xlab="beta", ylab="Density",
     lwd=2, col="blue")
lines(density(beta_samps[1,]), main="Density Plot of beta0", xlab="beta", ylab="Density",
      lwd=2, col="red")
legend("topright", legend=c("R","Rcpp"), fill=c("blue","red"))
```



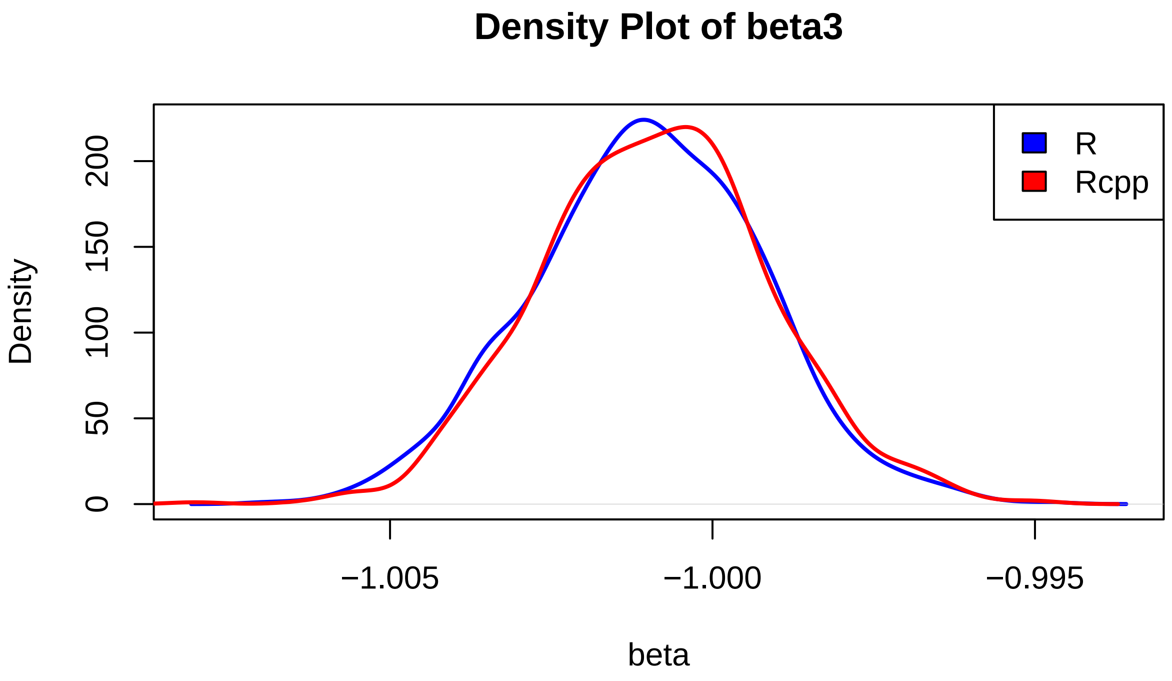
```
plot(density(beta.samps[2,]), main="Density Plot of beta1", xlab="beta", ylab="Density",
     lwd=2, col="blue")
lines(density(beta_samps[2,]), main="Density Plot of beta1", xlab="beta", ylab="Density",
      lwd=2, col="red")
legend("topright", legend=c("R","Rcpp"), fill=c("blue","red"))
```



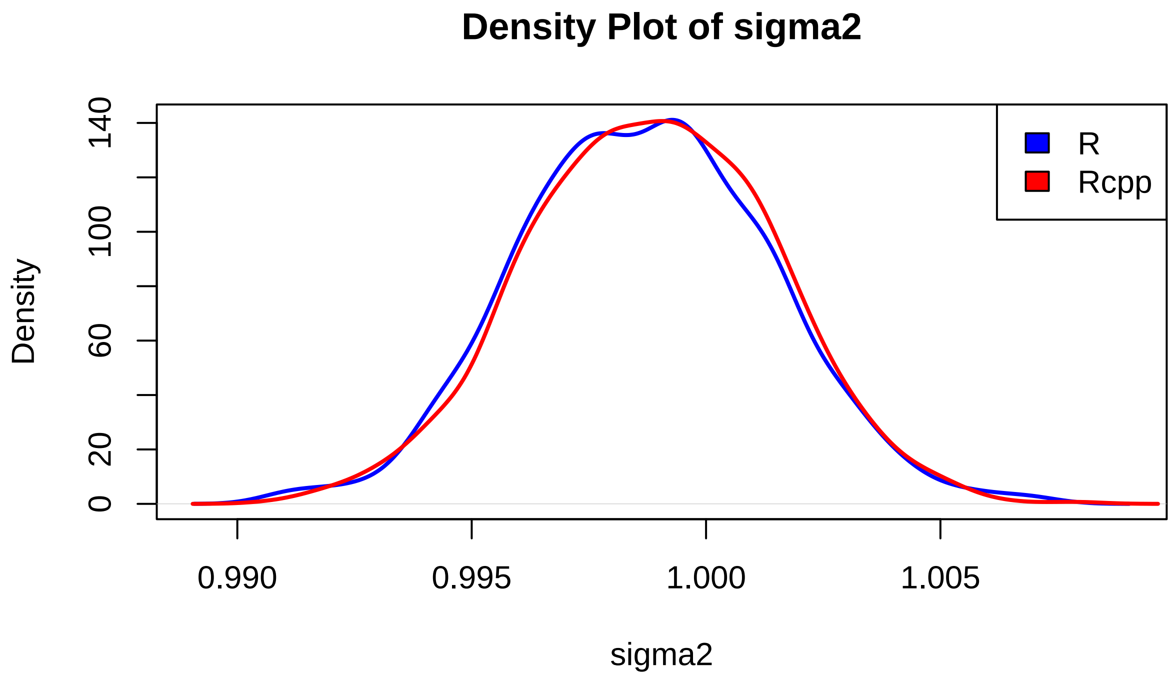
```r
plot(density(beta.samps[3,]), main="Density Plot of beta2", xlab="beta", ylab="Density",
     lwd=2, col="blue")
lines(density(beta_samps[3,]), main="Density Plot of beta2", xlab="beta", ylab="Density",
      lwd=2, col="red")
legend("topright", legend=c("R","Rcpp"), fill=c("blue","red"))
```



```r
plot(density(beta.samps[4,]), main="Density Plot of beta3", xlab="beta", ylab="Density",
     lwd=2, col="blue")
lines(density(beta_samps[4,]), main="Density Plot of beta3", xlab="beta", ylab="Density",
      lwd=2, col="red")
legend("topright", legend=c("R","Rcpp"), fill=c("blue","red"))
```



```r
plot(density(s2.samps), main="Density Plot of sigma2", xlab="sigma2", ylab="Density",
     lwd=2, col="blue")
lines(density(s2_samps), main="Density Plot of sigma2", xlab="sigma2", ylab="Density",
      lwd=2, col="red")
legend("topright", legend=c("R","Rcpp"), fill=c("blue","red"))
```



We can say the densities obtained Problem 2 and the densities obtained Problem 3 are the same posterior densities.

4. (Extra credit: 20 points) Now implement the divide-and-conquer MCMC algorithm to the simulated dataset. Here, you should write down the C++ code using RcppArmadillo and OpenMp libraries as follows.

```
library(parallel) # for checking number of cores (not for implementation)
numcore <- detectCores(); numcore
```

```
## [1] 8
```

```
# this is for openmp
Sys.setenv("PKG_CXXFLAGS"="-fopenmp")
Sys.setenv("PKG_LIBS"="-fopenmp")
```

My personal computer is available 8 cores. That means I divided the entire dataset into 8 shards.

```
ptm <- proc.time()
MCMC_RcppParallel <- RcppGibbsparallel(B, X_design, Y,  m.beta, v.beta, a.s2, b.s2, numcore)
RcppParalleltime <- proc.time() - ptm

beta_samps_ <- MCMC_RcppParallel$beta_samps
s2_samps_ <- matrix(MCMC_RcppParallel$s2_samps, ncol=B, nrow=numcore)
```

(a) The elapsed time of the Gibbs sampler implemented in i) R for problem2, ii) Rcpp for problem3, and iii) Rcpp and OpenMp for problem4 is shown below.

```
Rtime; Rcpptime; RcppParalleltime
```

```
##    user  system elapsed
##  19.687   1.394  21.085

##    user  system elapsed
##   6.645   0.493   7.140

##    user  system elapsed
##  53.012   3.218  56.308
```

The Gibbs sampler implemented in Rcpp on problem3 is much faster than the Gibbs sampler implemented in R on problem2. The Gibbs sampler implemented in Rcpp and OpenMp on problem 4 has a relatively long running time, but can be efficient if the data is very large.

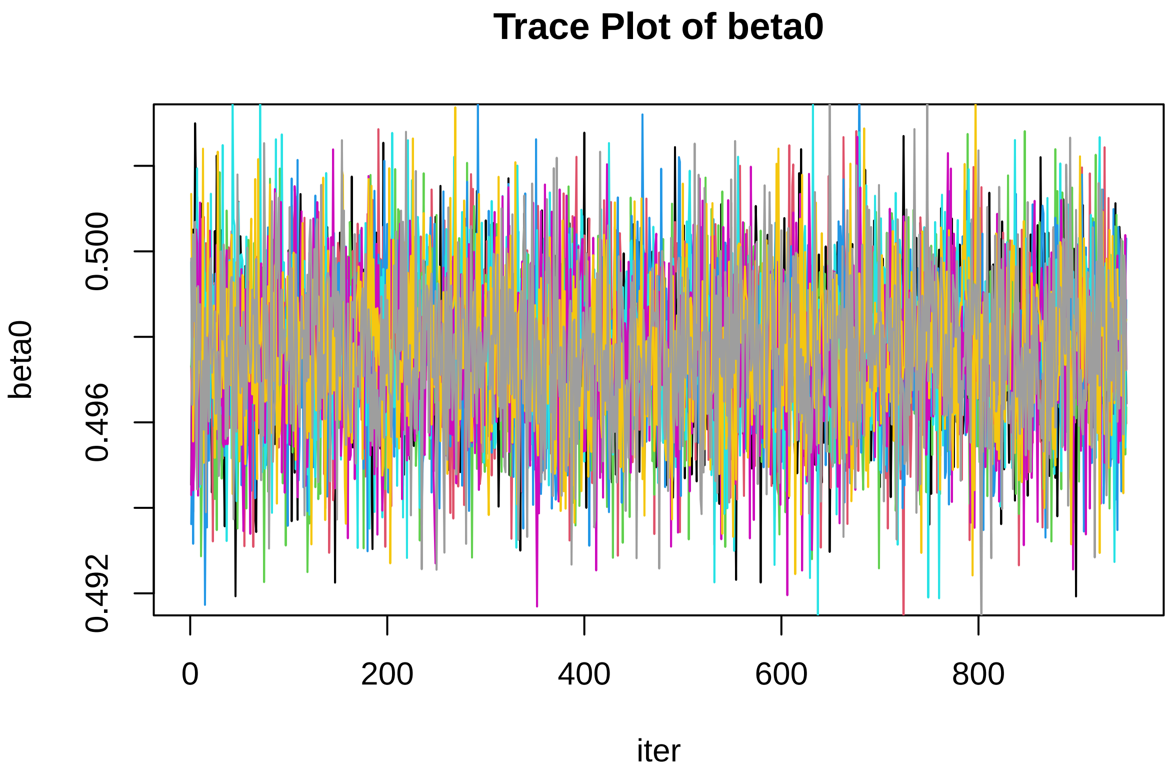
I will discard 50 samples for each parameters, as same as Problem 3.

```
# burn-in
burn_in <- 50
beta_samps_ <- beta_samps_[,(burn_in+1):B,]
s2_samps_ <- s2_samps_[,(burn_in+1):B]
```

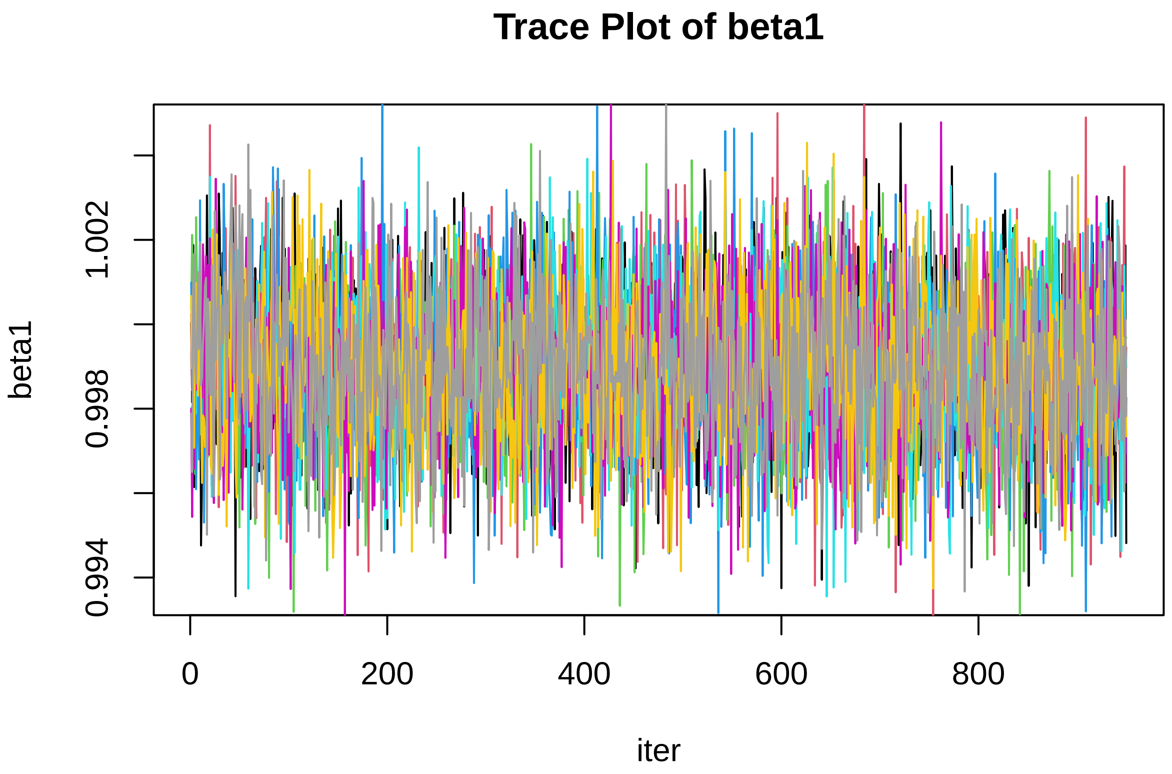
(b) I will check the **trace plot**.

```
# ts plot (after burn-in)
par(mfrow=c(1,1))
```

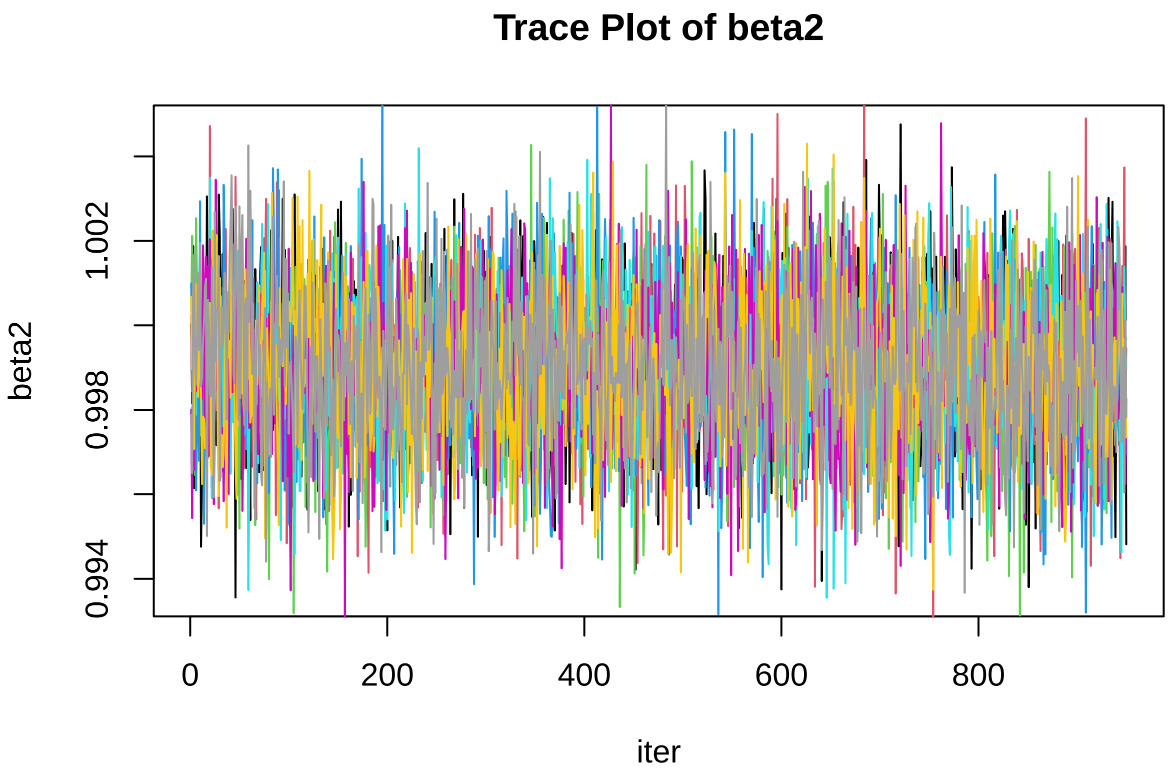
```
ts.plot(beta_samps_[,,1][1,],main="Trace Plot of beta0", xlab="iter", ylab="beta0")
for(i in 2:numcore){ lines(beta_samps_[,,i][1,],col=i) }
```



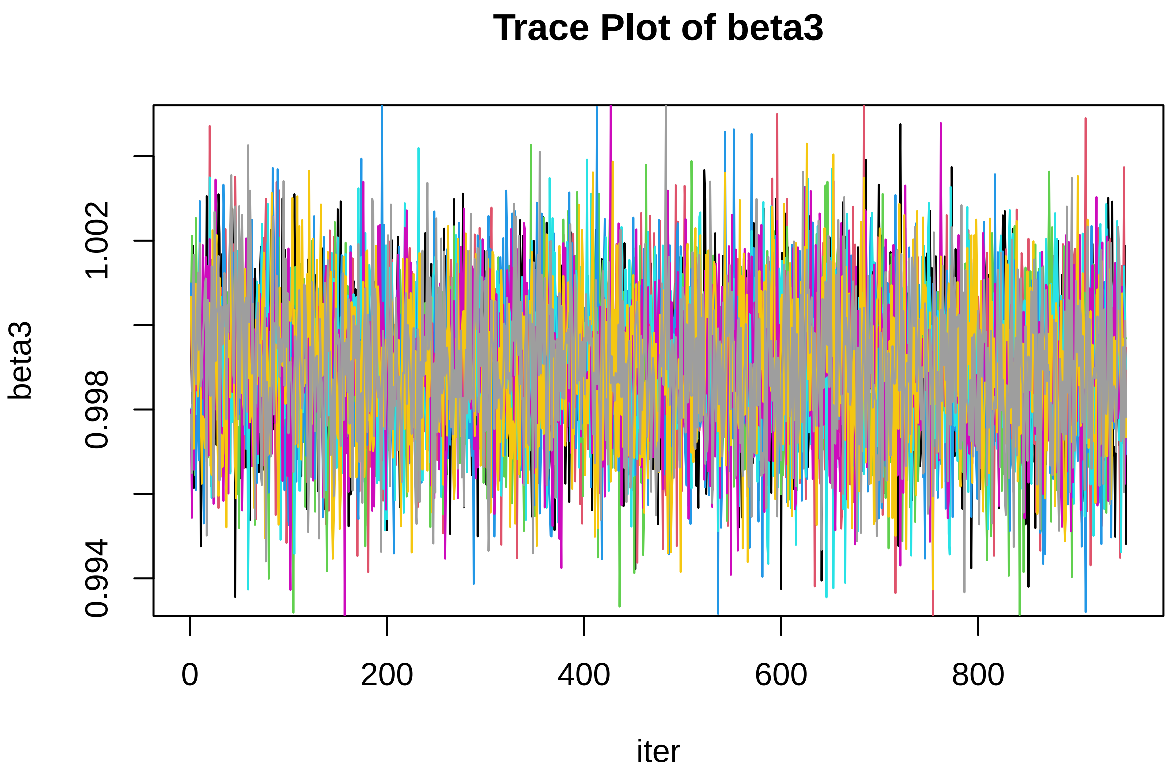
```
ts.plot(beta_samps_[,,1][2,],main="Trace Plot of beta1", xlab="iter", ylab="beta1")
for(i in 2:numcore){ lines(beta_samps_[,,i][2,],col=i) }
```



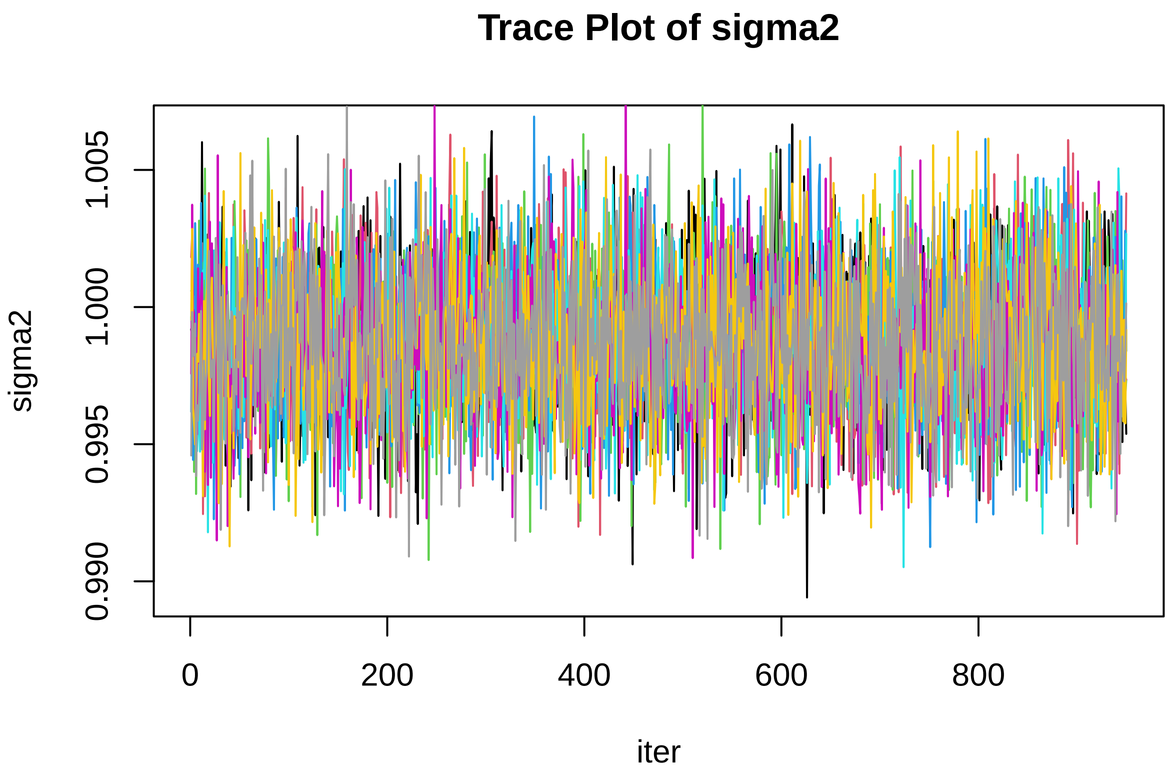
```r
ts.plot(beta_samps_[,,1][2,],main="Trace Plot of beta2", xlab="iter", ylab="beta2")
for(i in 2:numcore){ lines(beta_samps_[,,i][2,],col=i) }
```



```r
ts.plot(beta_samps_[,,1][2,],main="Trace Plot of beta3", xlab="iter", ylab="beta3")
for(i in 2:numcore){ lines(beta_samps_[,,i][2,],col=i) }
```



```
ts.plot(s2_samps_[1,],main="Trace Plot of sigma2", xlab="iter", ylab="sigma2")
for(i in 2:numcore){ lines(s2_samps_[i,],col=i) }
```

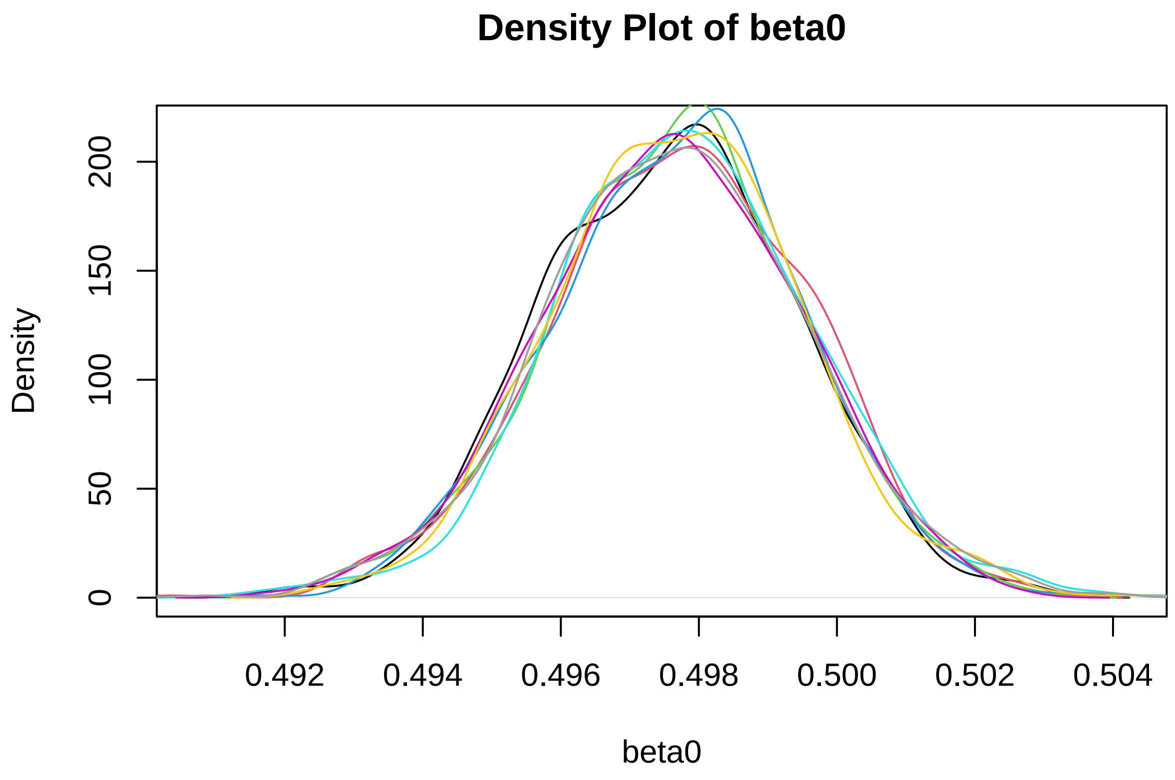


The burn in removed samples that were affected by the initial values, so a proper trace plot was drawn. I divided the data into 8 shards, and the samples generated in parallel on the each shards has a similar distribution.

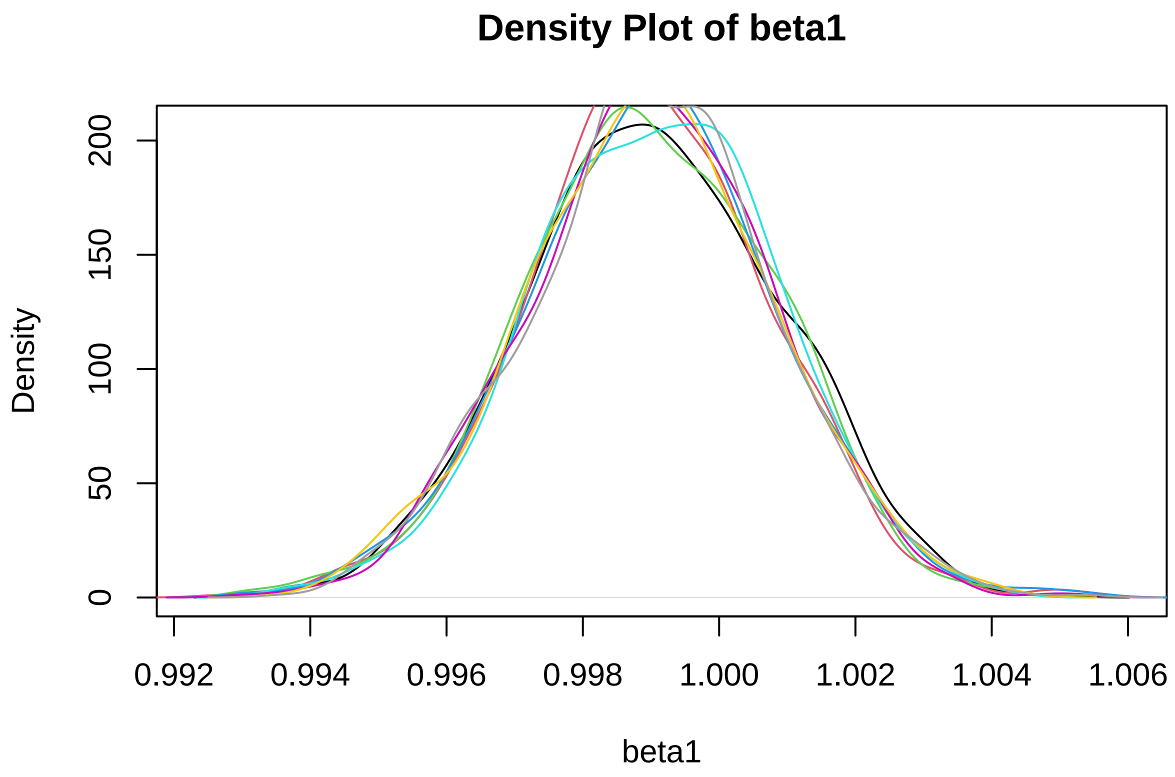
Next, I will check the **density plot**

```
# density plot
plot(density(beta_samps_[,,1][1,]), main="Density Plot of beta0", xlab="beta0", ylab="Density")
```

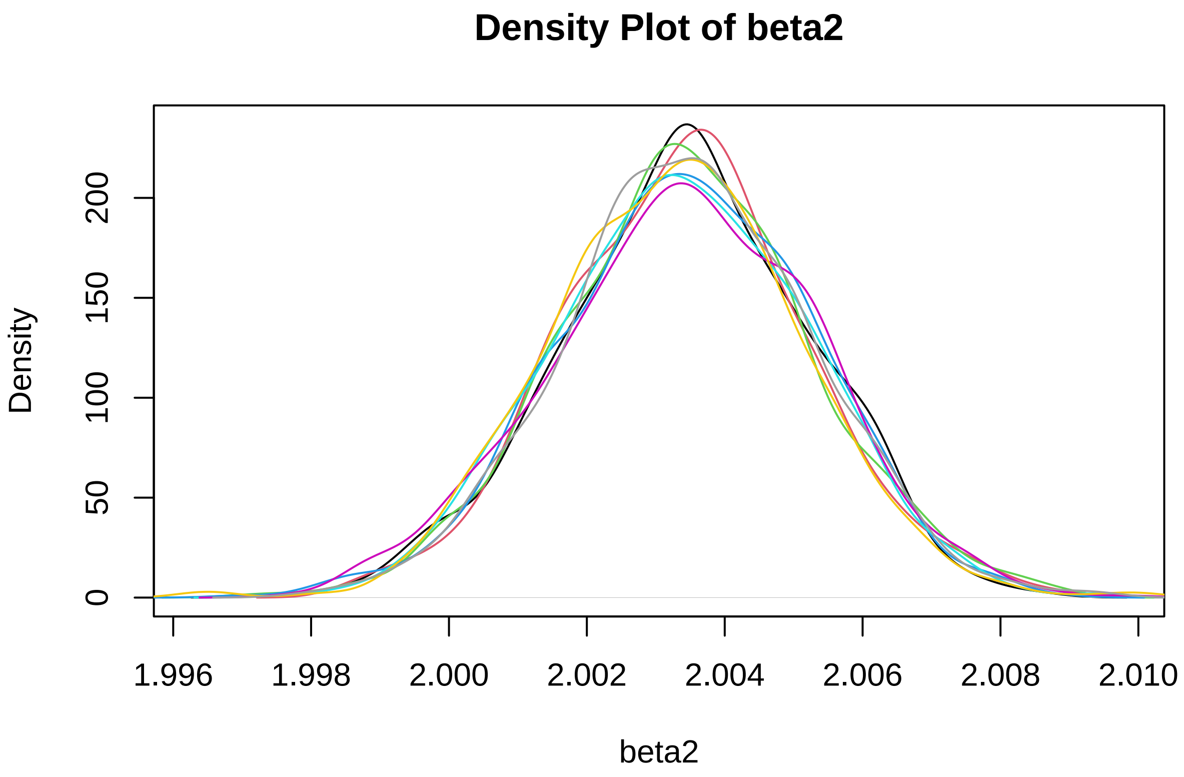
```r
for(i in 2:numcore){ lines(density(beta_samps_[,,i][1,]), col=i) }
```



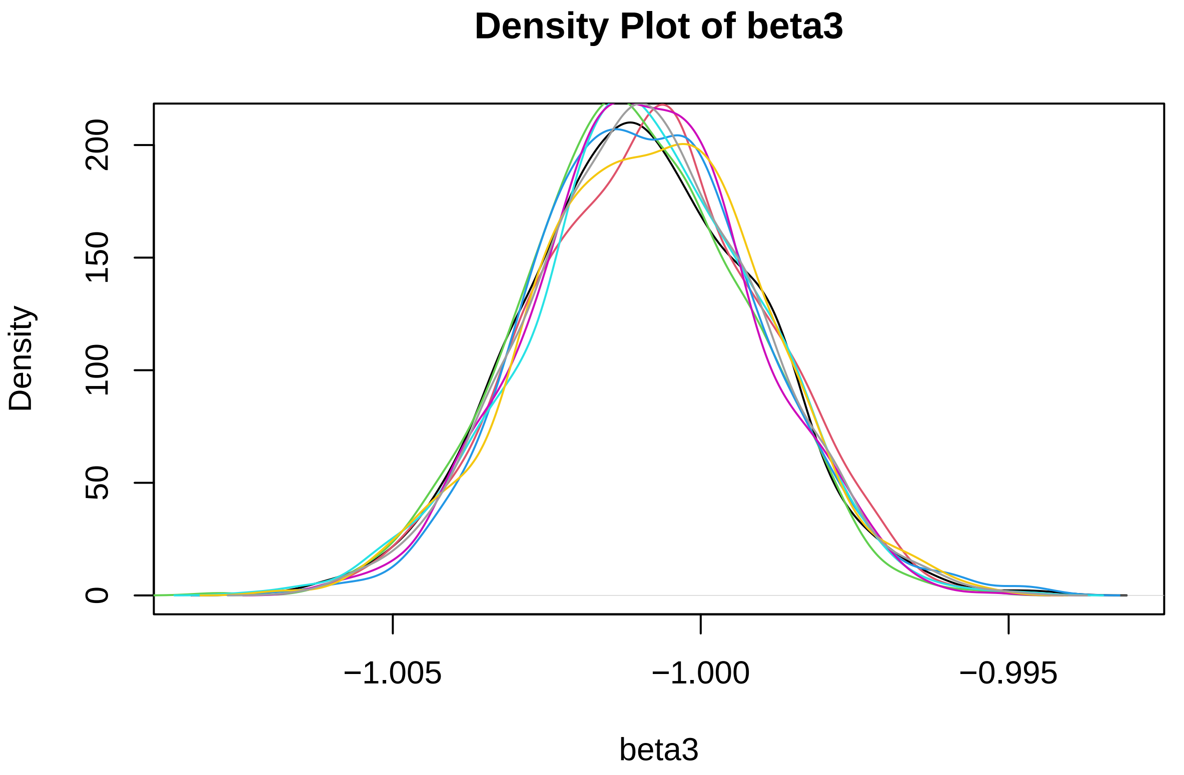
```r
plot(density(beta_samps_[,,1][2,]), main="Density Plot of beta1", xlab="beta1", ylab="Density")
for(i in 2:numcore){ lines(density(beta_samps_[,,i][2,]), col=i) }
```



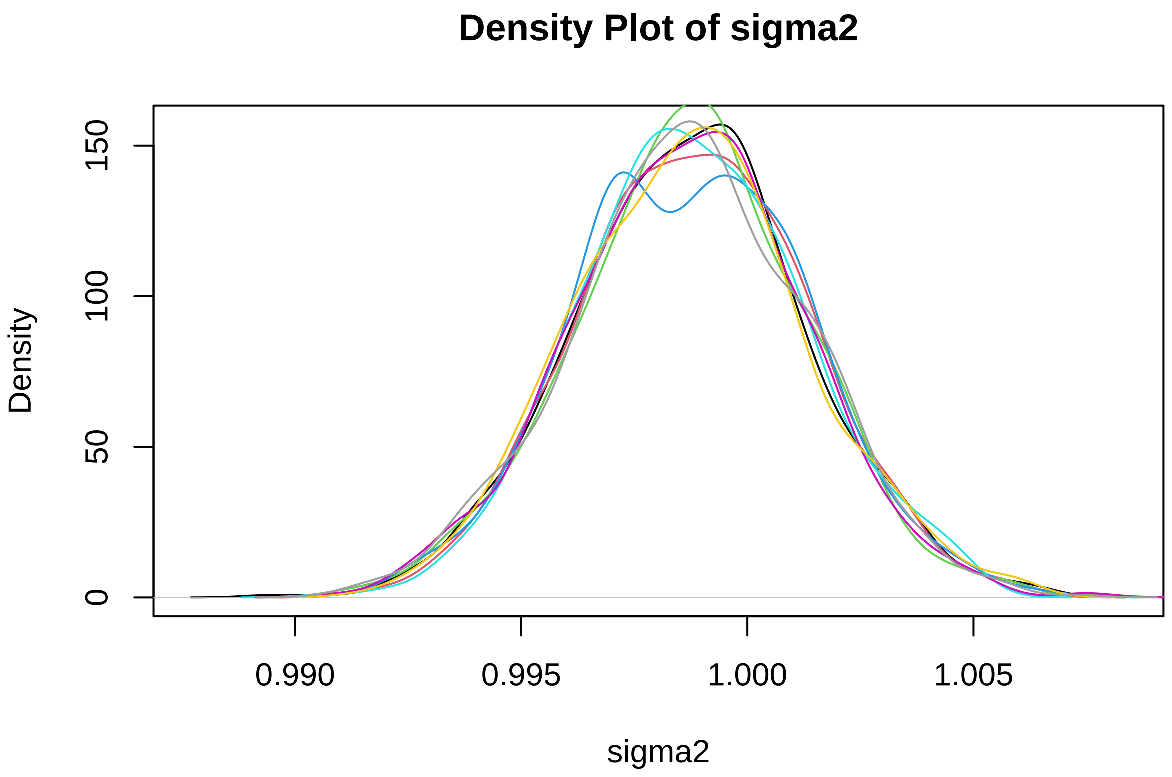
```r
plot(density(beta_samps_[,,1][3,]), main="Density Plot of beta2", xlab="beta2", ylab="Density")
for(i in 2:numcore){ lines(density(beta_samps_[,,i][3,]), col=i) }
```



```r
plot(density(beta_samps_[,,1][4,]), main="Density Plot of beta3", xlab="beta3", ylab="Density")
for(i in 2:numcore){ lines(density(beta_samps_[,,i][4,]), col=i) }
```



```r
plot(density(s2_samps_[1,]), main="Density Plot of sigma2", xlab="sigma2", ylab="Density")
for(i in 2:numcore){ lines(density(s2_samps_[i,]), col=i) }
```



Next, I will print the 95% HPD intervals, posterior mean, acceptance probability, and effective sample size.

```r
# 95% HPD intervals
print_interval <- function(core){
  for (j in 1:p){
    for (i in 1:core){
      cat("95% HPD intervals of beta",j-1,"/shard", i,":",
          HPDinterval(as.mcmc(beta_samps_[,,i][j,]), prob = 0.95), "\n")}
    cat("\n")}
  for (i in 1: core){
    cat("95% HPD intervals of sigma2 /shard", i,":",
        HPDinterval(as.mcmc(s2_samps_[i,]), prob = 0.95), "\n")}}

print_interval(numcore)
```

```
## 95% HPD intervals of beta 0 /shard 1 : 0.4940723 0.5008636
## 95% HPD intervals of beta 0 /shard 2 : 0.4943476 0.5015217
## 95% HPD intervals of beta 0 /shard 3 : 0.4939847 0.5010592
## 95% HPD intervals of beta 0 /shard 4 : 0.494069 0.5008848
## 95% HPD intervals of beta 0 /shard 5 : 0.4941562 0.5013782
## 95% HPD intervals of beta 0 /shard 6 : 0.4940714 0.5012032
## 95% HPD intervals of beta 0 /shard 7 : 0.494324 0.5013443
## 95% HPD intervals of beta 0 /shard 8 : 0.4938241 0.5014464
##
## 95% HPD intervals of beta 1 /shard 1 : 0.9952328 1.002286
## 95% HPD intervals of beta 1 /shard 2 : 0.9955009 1.002303
## 95% HPD intervals of beta 1 /shard 3 : 0.995677 1.00244
## 95% HPD intervals of beta 1 /shard 4 : 0.9955373 1.002698
## 95% HPD intervals of beta 1 /shard 5 : 0.9956413 1.002466
## 95% HPD intervals of beta 1 /shard 6 : 0.9956006 1.002267
## 95% HPD intervals of beta 1 /shard 7 : 0.9951698 1.002257
## 95% HPD intervals of beta 1 /shard 8 : 0.9956625 1.002685
##
## 95% HPD intervals of beta 2 /shard 1 : 1.999652 2.006528
## 95% HPD intervals of beta 2 /shard 2 : 1.999785 2.00693
## 95% HPD intervals of beta 2 /shard 3 : 1.999901 2.007046
## 95% HPD intervals of beta 2 /shard 4 : 1.999889 2.00689
## 95% HPD intervals of beta 2 /shard 5 : 1.999609 2.006614
## 95% HPD intervals of beta 2 /shard 6 : 1.999597 2.007137
## 95% HPD intervals of beta 2 /shard 7 : 1.999944 2.006817
## 95% HPD intervals of beta 2 /shard 8 : 2.000151 2.007021
##
## 95% HPD intervals of beta 3 /shard 1 : -1.004349 -0.9971836
## 95% HPD intervals of beta 3 /shard 2 : -1.004275 -0.997083
## 95% HPD intervals of beta 3 /shard 3 : -1.004494 -0.9977056
## 95% HPD intervals of beta 3 /shard 4 : -1.004286 -0.9972883
## 95% HPD intervals of beta 3 /shard 5 : -1.004708 -0.9974638
## 95% HPD intervals of beta 3 /shard 6 : -1.004395 -0.9976791
## 95% HPD intervals of beta 3 /shard 7 : -1.004764 -0.9975033
## 95% HPD intervals of beta 3 /shard 8 : -1.004755 -0.9977082
##
## 95% HPD intervals of sigma2 /shard 1 : 0.9938275 1.003751
## 95% HPD intervals of sigma2 /shard 2 : 0.9939229 1.003588
## 95% HPD intervals of sigma2 /shard 3 : 0.9933296 1.003276
## 95% HPD intervals of sigma2 /shard 4 : 0.9938909 1.003827
## 95% HPD intervals of sigma2 /shard 5 : 0.9940441 1.003735
## 95% HPD intervals of sigma2 /shard 6 : 0.9932815 1.00322
## 95% HPD intervals of sigma2 /shard 7 : 0.9935013 1.00357
## 95% HPD intervals of sigma2 /shard 8 : 0.9937832 1.003857
```

```r
# posterior mean
print_posterior_mean <- function(core){
  for (j in 1:p){
    for (i in 1:core){
      cat("posterior mean of beta",j-1, "/shard:", i,":",
          mean(beta_samps_[,,i][j,]), "\n")}
    cat("\n")}
  for (i in 1: core){
    cat("posterior mean of sigma2 /shard", i,":",
        mean(s2_samps_[i,], prob = 0.95), "\n")}}

print_posterior_mean(numcore)
```

```
## posterior mean of beta 0 /shard: 1 : 0.4975827
## posterior mean of beta 0 /shard: 2 : 0.4977206
## posterior mean of beta 0 /shard: 3 : 0.4976408
## posterior mean of beta 0 /shard: 4 : 0.497695
## posterior mean of beta 0 /shard: 5 : 0.4978001
## posterior mean of beta 0 /shard: 6 : 0.4975925
## posterior mean of beta 0 /shard: 7 : 0.497688
## posterior mean of beta 0 /shard: 8 : 0.497668
##
```

```
## posterior mean of beta 1 /shard: 1 : 0.9990393
## posterior mean of beta 1 /shard: 2 : 0.9989513
## posterior mean of beta 1 /shard: 3 : 0.9989611
## posterior mean of beta 1 /shard: 4 : 0.9990062
## posterior mean of beta 1 /shard: 5 : 0.9990697
## posterior mean of beta 1 /shard: 6 : 0.9989934
## posterior mean of beta 1 /shard: 7 : 0.9989741
## posterior mean of beta 1 /shard: 8 : 0.9990078
##
## posterior mean of beta 2 /shard: 1 : 2.003433
## posterior mean of beta 2 /shard: 2 : 2.003433
## posterior mean of beta 2 /shard: 3 : 2.003436
## posterior mean of beta 2 /shard: 4 : 2.003436
## posterior mean of beta 2 /shard: 5 : 2.003384
## posterior mean of beta 2 /shard: 6 : 2.003416
## posterior mean of beta 2 /shard: 7 : 2.003284
## posterior mean of beta 2 /shard: 8 : 2.003469
##
## posterior mean of beta 3 /shard: 1 : -1.000979
## posterior mean of beta 3 /shard: 2 : -1.00086
## posterior mean of beta 3 /shard: 3 : -1.001104
## posterior mean of beta 3 /shard: 4 : -1.000874
## posterior mean of beta 3 /shard: 5 : -1.000944
## posterior mean of beta 3 /shard: 6 : -1.000942
## posterior mean of beta 3 /shard: 7 : -1.000858
## posterior mean of beta 3 /shard: 8 : -1.000923
##
## posterior mean of sigma2 /shard 1 : 0.9987546
## posterior mean of sigma2 /shard 2 : 0.9988304
## posterior mean of sigma2 /shard 3 : 0.9987511
## posterior mean of sigma2 /shard 4 : 0.9987659
## posterior mean of sigma2 /shard 5 : 0.9987908
## posterior mean of sigma2 /shard 6 : 0.9986846
## posterior mean of sigma2 /shard 7 : 0.998734
## posterior mean of sigma2 /shard 8 : 0.9987091
```

```
# acceptance probability
# (Because I used Gibbs sampler, samples was always accepted !)
print_acc_prob <- function(core){
  for (j in 1:p){
    for (i in 1:core){
      cat("Acceptance probability of beta", j-1, "/shard", i,":",
          length(unique(beta_samps_[,,i][j,]))/B ,"\n")}
    cat("\n")}
  for (i in 1: core){
    cat("Acceptance probability of sigma2 /shard", i,":",
        length(s2_samps_[i,])/B, "\n")}}

print_acc_prob(numcore)
```

```
## Acceptance probability of beta 0 /shard 1 : 0.95
## Acceptance probability of beta 0 /shard 2 : 0.95
## Acceptance probability of beta 0 /shard 3 : 0.95
## Acceptance probability of beta 0 /shard 4 : 0.95
## Acceptance probability of beta 0 /shard 5 : 0.95
## Acceptance probability of beta 0 /shard 6 : 0.95
## Acceptance probability of beta 0 /shard 7 : 0.95
## Acceptance probability of beta 0 /shard 8 : 0.95
##
## Acceptance probability of beta 1 /shard 1 : 0.95
## Acceptance probability of beta 1 /shard 2 : 0.95
## Acceptance probability of beta 1 /shard 3 : 0.95
## Acceptance probability of beta 1 /shard 4 : 0.95
## Acceptance probability of beta 1 /shard 5 : 0.95
## Acceptance probability of beta 1 /shard 6 : 0.95
## Acceptance probability of beta 1 /shard 7 : 0.95
## Acceptance probability of beta 1 /shard 8 : 0.95
##
## Acceptance probability of beta 2 /shard 1 : 0.95
## Acceptance probability of beta 2 /shard 2 : 0.95
## Acceptance probability of beta 2 /shard 3 : 0.95
## Acceptance probability of beta 2 /shard 4 : 0.95
## Acceptance probability of beta 2 /shard 5 : 0.95
## Acceptance probability of beta 2 /shard 6 : 0.95
## Acceptance probability of beta 2 /shard 7 : 0.95
## Acceptance probability of beta 2 /shard 8 : 0.95
##
## Acceptance probability of beta 3 /shard 1 : 0.95
## Acceptance probability of beta 3 /shard 2 : 0.95
## Acceptance probability of beta 3 /shard 3 : 0.95
## Acceptance probability of beta 3 /shard 4 : 0.95
## Acceptance probability of beta 3 /shard 5 : 0.95
## Acceptance probability of beta 3 /shard 6 : 0.95
## Acceptance probability of beta 3 /shard 7 : 0.95
## Acceptance probability of beta 3 /shard 8 : 0.95
##
## Acceptance probability of sigma2 /shard 1 : 0.95
## Acceptance probability of sigma2 /shard 2 : 0.95
## Acceptance probability of sigma2 /shard 3 : 0.95
## Acceptance probability of sigma2 /shard 4 : 0.95
## Acceptance probability of sigma2 /shard 5 : 0.95
## Acceptance probability of sigma2 /shard 6 : 0.95
## Acceptance probability of sigma2 /shard 7 : 0.95
## Acceptance probability of sigma2 /shard 8 : 0.95
```

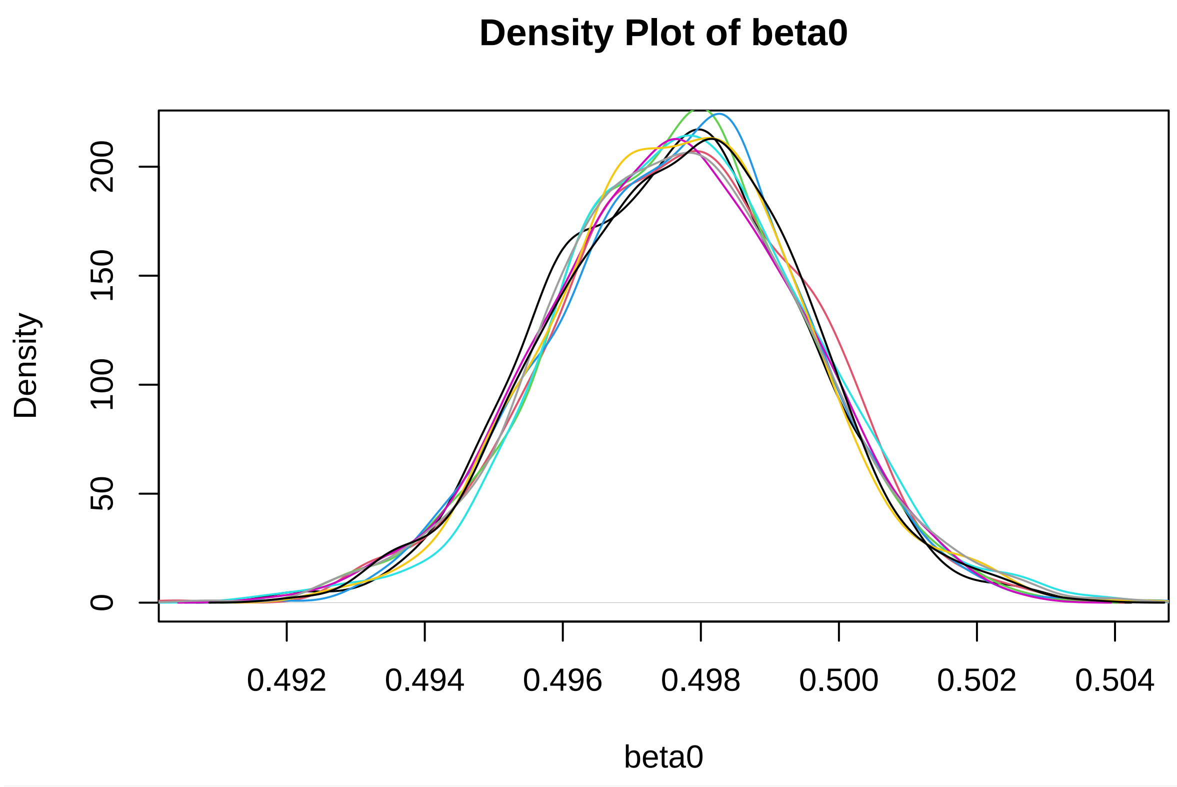
```r
# effective sample size
print_ess <- function(core){
  for (j in 1:p){
    for (i in 1:core){
      cat("Effective Sample size of beta", j-1, "/shard", i,":",
          effectiveSize(beta_samps_[,,i][p,]), "\n")}
    cat("\n")}
  for (i in 1: core){
    cat("Acceptance probability of sigma2 /shard", i,":",
        effectiveSize(s2_samps_[i,]), "\n")}}
print_ess(numcore)
```

```
## Effective Sample size of beta 0 /shard 1 : 950
## Effective Sample size of beta 0 /shard 2 : 1366.338
## Effective Sample size of beta 0 /shard 3 : 950
## Effective Sample size of beta 0 /shard 4 : 950
## Effective Sample size of beta 0 /shard 5 : 1302.724
## Effective Sample size of beta 0 /shard 6 : 950
## Effective Sample size of beta 0 /shard 7 : 950
## Effective Sample size of beta 0 /shard 8 : 1200.155
##
## Effective Sample size of beta 1 /shard 1 : 950
## Effective Sample size of beta 1 /shard 2 : 1366.338
## Effective Sample size of beta 1 /shard 3 : 950
## Effective Sample size of beta 1 /shard 4 : 950
## Effective Sample size of beta 1 /shard 5 : 1302.724
## Effective Sample size of beta 1 /shard 6 : 950
## Effective Sample size of beta 1 /shard 7 : 950
## Effective Sample size of beta 1 /shard 8 : 1200.155
##
## Effective Sample size of beta 2 /shard 1 : 950
## Effective Sample size of beta 2 /shard 2 : 1366.338
## Effective Sample size of beta 2 /shard 3 : 950
## Effective Sample size of beta 2 /shard 4 : 950
## Effective Sample size of beta 2 /shard 5 : 1302.724
## Effective Sample size of beta 2 /shard 6 : 950
## Effective Sample size of beta 2 /shard 7 : 950
## Effective Sample size of beta 2 /shard 8 : 1200.155
##
## Effective Sample size of beta 3 /shard 1 : 950
## Effective Sample size of beta 3 /shard 2 : 1366.338
## Effective Sample size of beta 3 /shard 3 : 950
## Effective Sample size of beta 3 /shard 4 : 950
## Effective Sample size of beta 3 /shard 5 : 1302.724
## Effective Sample size of beta 3 /shard 6 : 950
## Effective Sample size of beta 3 /shard 7 : 950
## Effective Sample size of beta 3 /shard 8 : 1200.155
##
## Acceptance probability of sigma2 /shard 1 : 950
## Acceptance probability of sigma2 /shard 2 : 950
## Acceptance probability of sigma2 /shard 3 : 1052.369
## Acceptance probability of sigma2 /shard 4 : 950
## Acceptance probability of sigma2 /shard 5 : 950
## Acceptance probability of sigma2 /shard 6 : 916.4552
## Acceptance probability of sigma2 /shard 7 : 956.4089
## Acceptance probability of sigma2 /shard 8 : 950
```

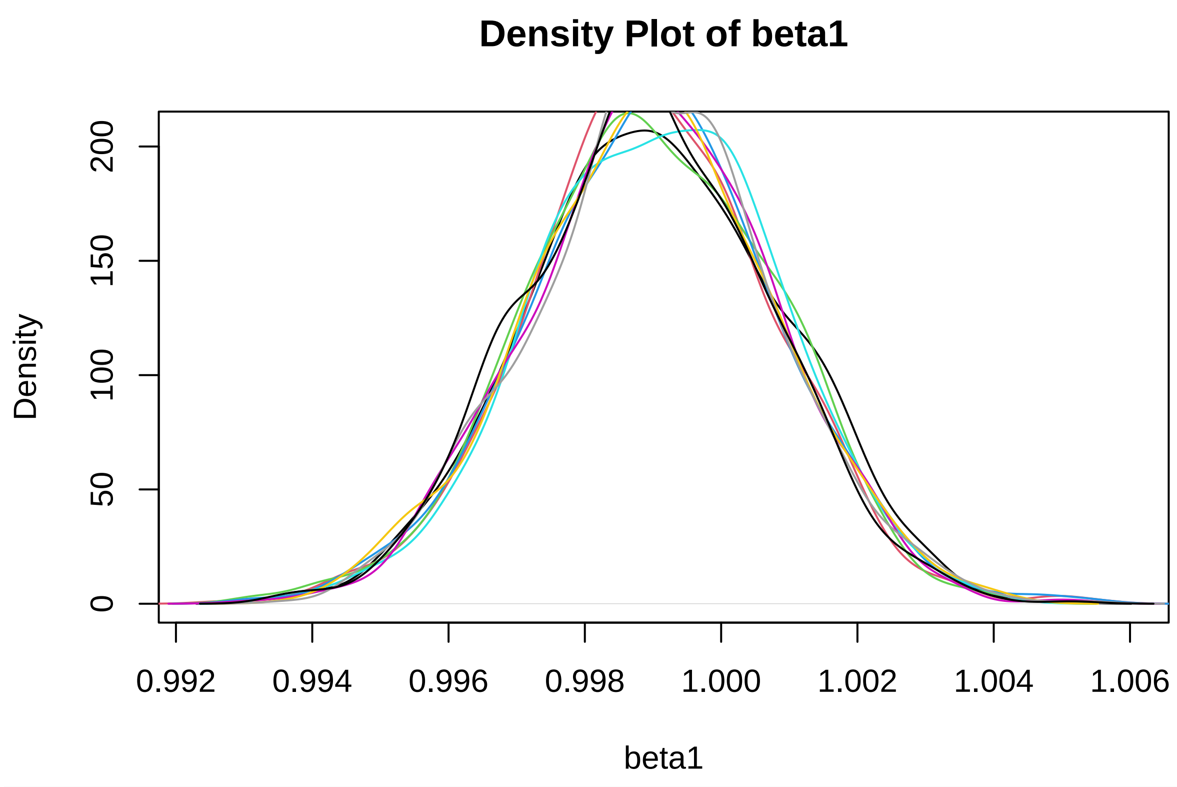
Since I used the Gibbs sampler, all samples are accepted and the acceptance probability for all parameters is 0.95 (0.05 is due to burn-in).

(c) I will check the density plot of 8 subset posteriors is similar to the density of combined posterior

```r
# density plot
plot(density(beta_samps_[,,1][1,]), main="Density Plot of beta0", xlab="beta0", ylab="Density")
for(i in 2:numcore){ lines(density(beta_samps_[,,i][1,]), col=i) }
lines(density(beta_samps[1,]))
```

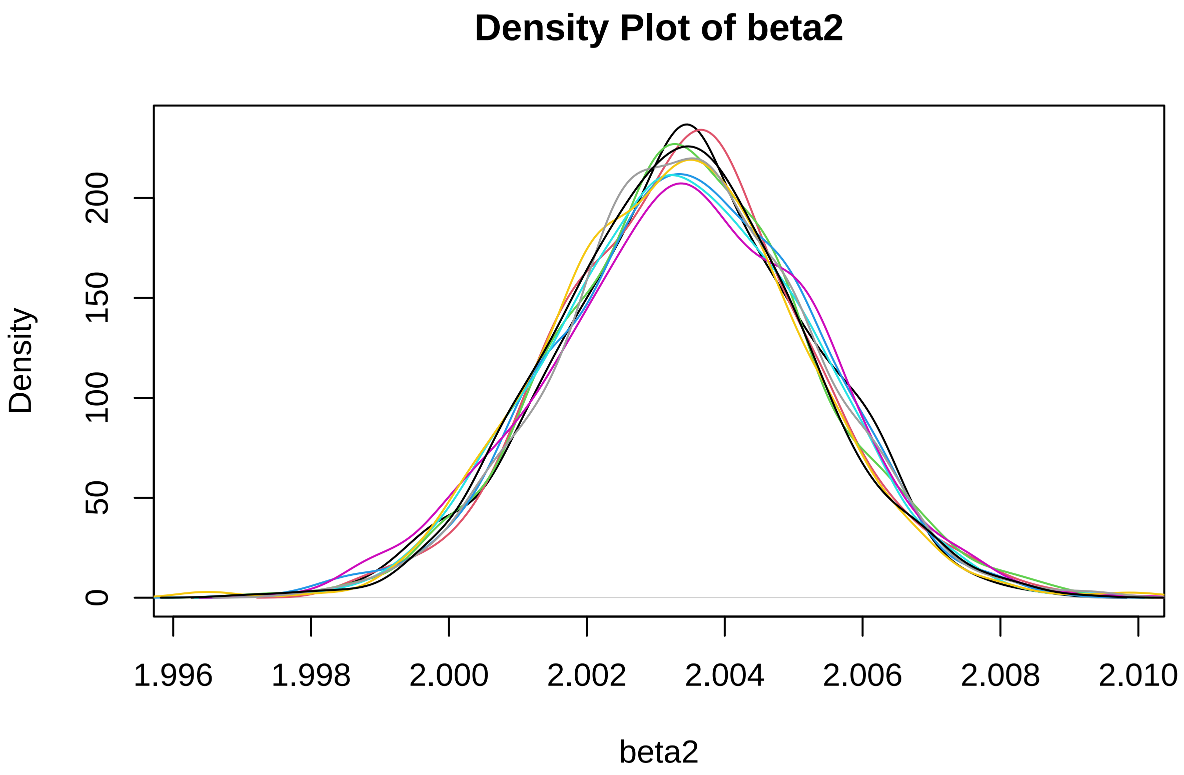


```r
plot(density(beta_samps_[,,1][2,]), main="Density Plot of beta1", xlab="beta1", ylab="Density")
for(i in 2:numcore){ lines(density(beta_samps_[,,i][2,]), col=i) }
lines(density(beta_samps[2,]))
```

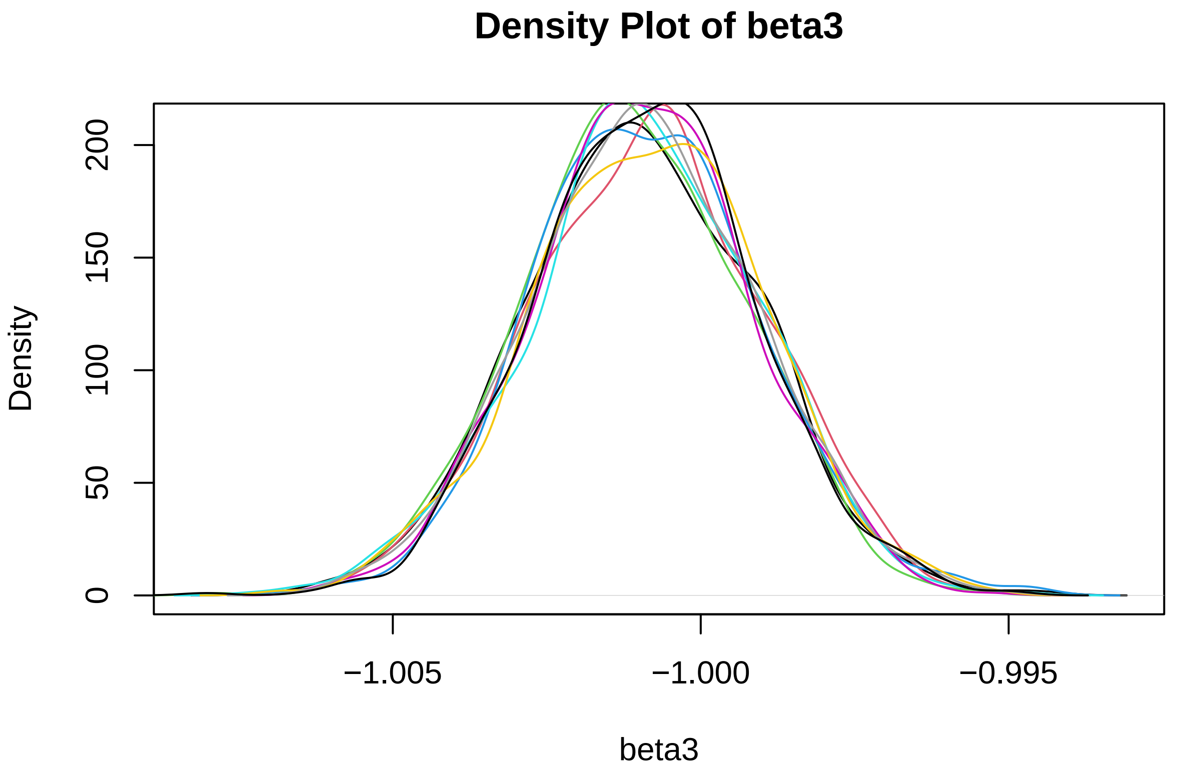


```r
plot(density(beta_samps_[,,1][3,]), main="Density Plot of beta2", xlab="beta2", ylab="Density")
for(i in 2:numcore){ lines(density(beta_samps_[,,i][3,]), col=i) }
```

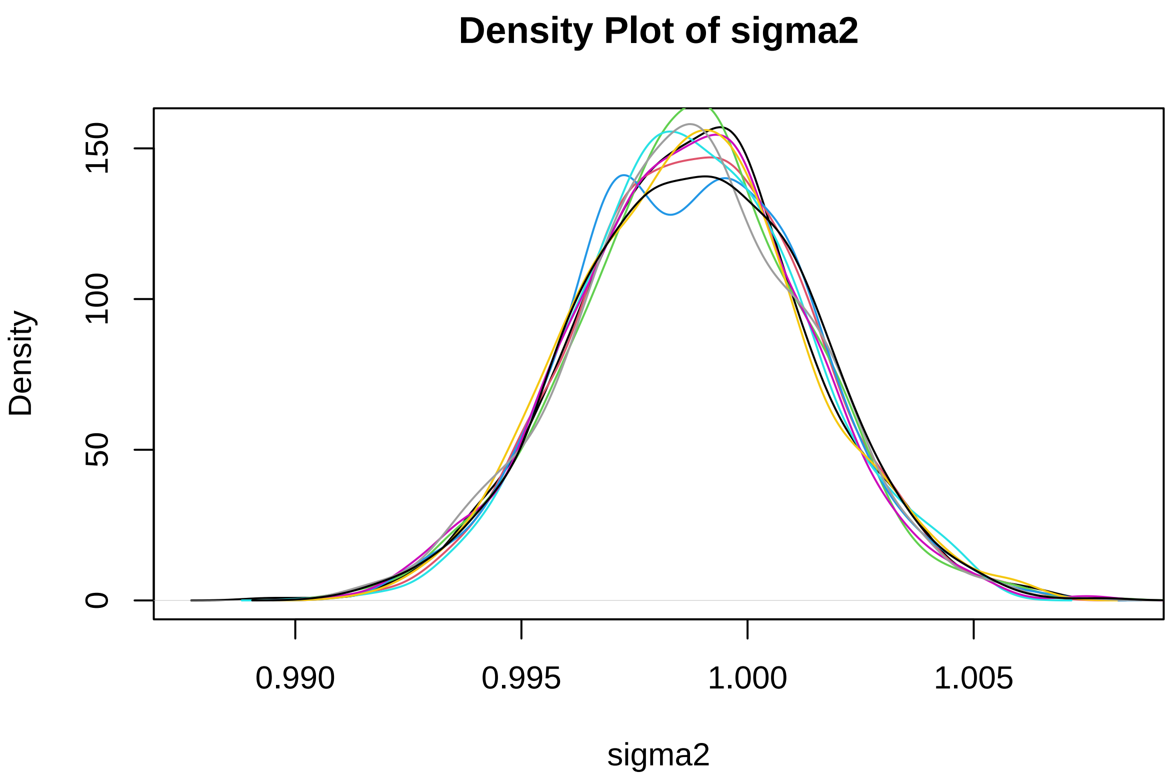
```
lines(density(beta_samps[3,]))
```



```
plot(density(beta_samps_[,,1][4,]), main="Density Plot of beta3", xlab="beta3", ylab="Density")
for(i in 2:numcore){ lines(density(beta_samps_[,,i][4,]), col=i) }
lines(density(beta_samps[4,]))
```



```r
plot(density(s2_samps_[1,]), main="Density Plot of sigma2", xlab="sigma2", ylab="Density")
for(i in 2:numcore){ lines(density(s2_samps_[i,]), col=i) }
lines(density(s2_samps))
```



I overlaid the densities of the samples obtained from problem4 with Rcpp / OpenMp with samples obtained from problem3 with Rcpp. All posteriors have similar densities.