Introduction to Bayesian Methods

Vanda Inácio

Normal model with unknown mean and known variance

Since the posterior of μ is normal, we can use the functions dnorm, pnorm, and qnorm to summarise the posterior.

```
data_chol <- read.csv2("cholesterol.txt")
data_chol <- as.numeric(data_chol[,1])
y <- log(data_chol)
n <- length(y)

mu0 <- 0
sigma02 <- 100
sigma2 <- var(y)

#posterior mean
meanmu <- ((mu0/sigma02) + n*mean(y)/sigma2)/((1/sigma02) + (n/sigma2))
meanmu
```

```
## [1] 5.730188
```

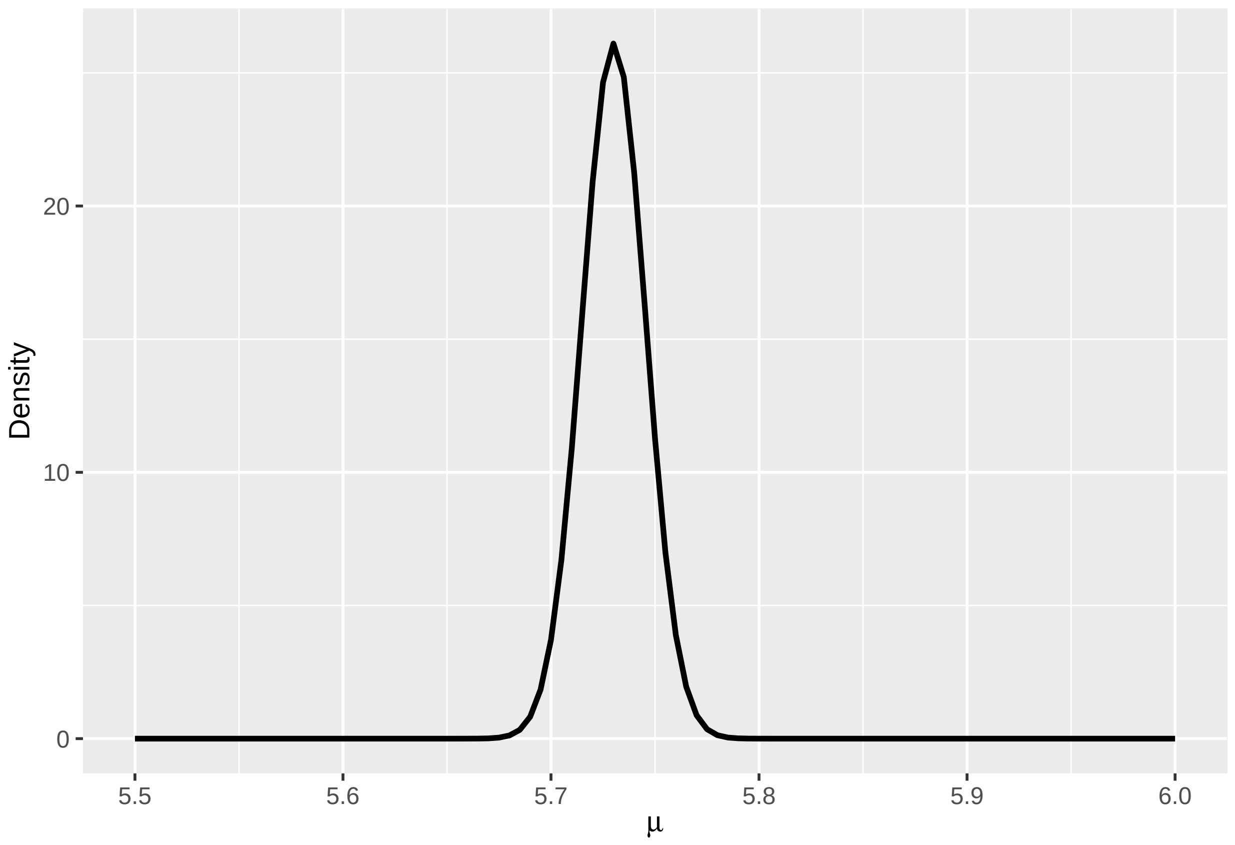
```
#posterior variance
varmu <- 1/((1/sigma02) + (n/sigma2))
varmu
```

```
## [1] 0.000233593
```

```
#equal tailed credible interval
qnorm(c(0.025,0.975), meanmu, sqrt(varmu))
```

```
## [1] 5.700233 5.760144
```

```
#plot posterior density mu
pdf_mu <- dnorm(seq(5.5, 6, len = 500), meanmu, sqrt(varmu))

require(ggplot2)
ggplot(data.frame(x = c(5.5, 6)), aes(x = x)) +
stat_function(fun = dnorm, args = list(mean = meanmu, sd = sqrt(varmu)), linewidth = 1) +
xlab(expression(mu)) +
ylab("Density")
```



Although in this case we can summarise the posterior distribution easily with R built-in functions, in higher dimensional posterior distributions this will not be possible and so we will need to resort to Monte Carlo sampling. For this reason, we also introduce it here.

```
S <- 100000
mu_post <- rnorm(S, meanmu, sqrt(varmu))
mean(mu_post)
```

```
## [1] 5.73013
```

```
quantile(mu_post, c(0.025,0.975))
```

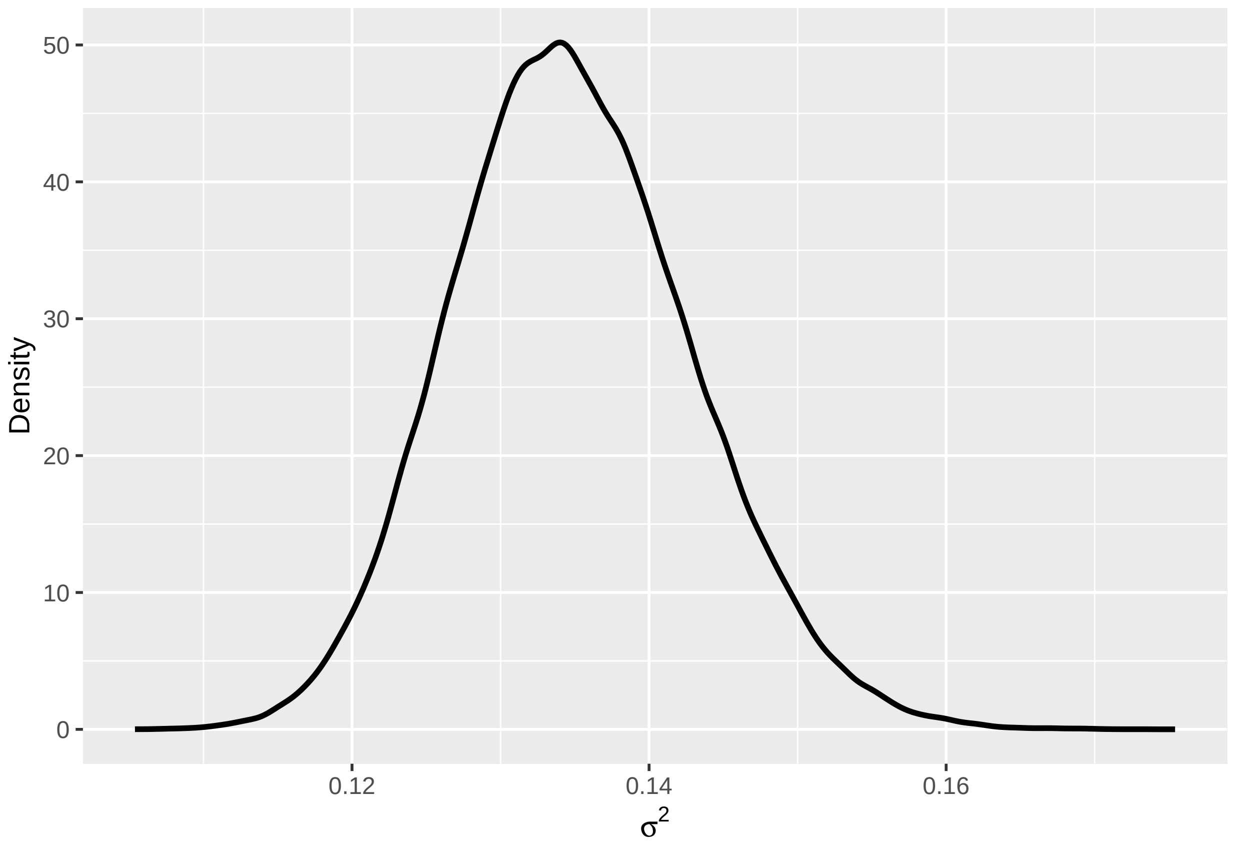
```
##     2.5%    97.5%
## 5.700204 5.760078
```

Normal model with known mean and unknown variance

```
mu <- mean(y)
a <- 1
b <- 1
a_post <- a + (n/2)
b_post <- b + 0.5*sum((y - mu)^2)
S <- 100000
```

```
sigma2 <- 1/rgamma(S, a_post, b_post)

ggplot(data.frame("values" = sigma2), aes(x = values)) +
geom_density(size = 1) +
xlab(expression(sigma^2)) +
ylab("Density")
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



```
mean(sigma2)
```

```
## [1] 0.134857
```

```
quantile(sigma2, c(0.025,0.975))
```

```
##      2.5%     97.5%
## 0.1200176 0.1514727
```

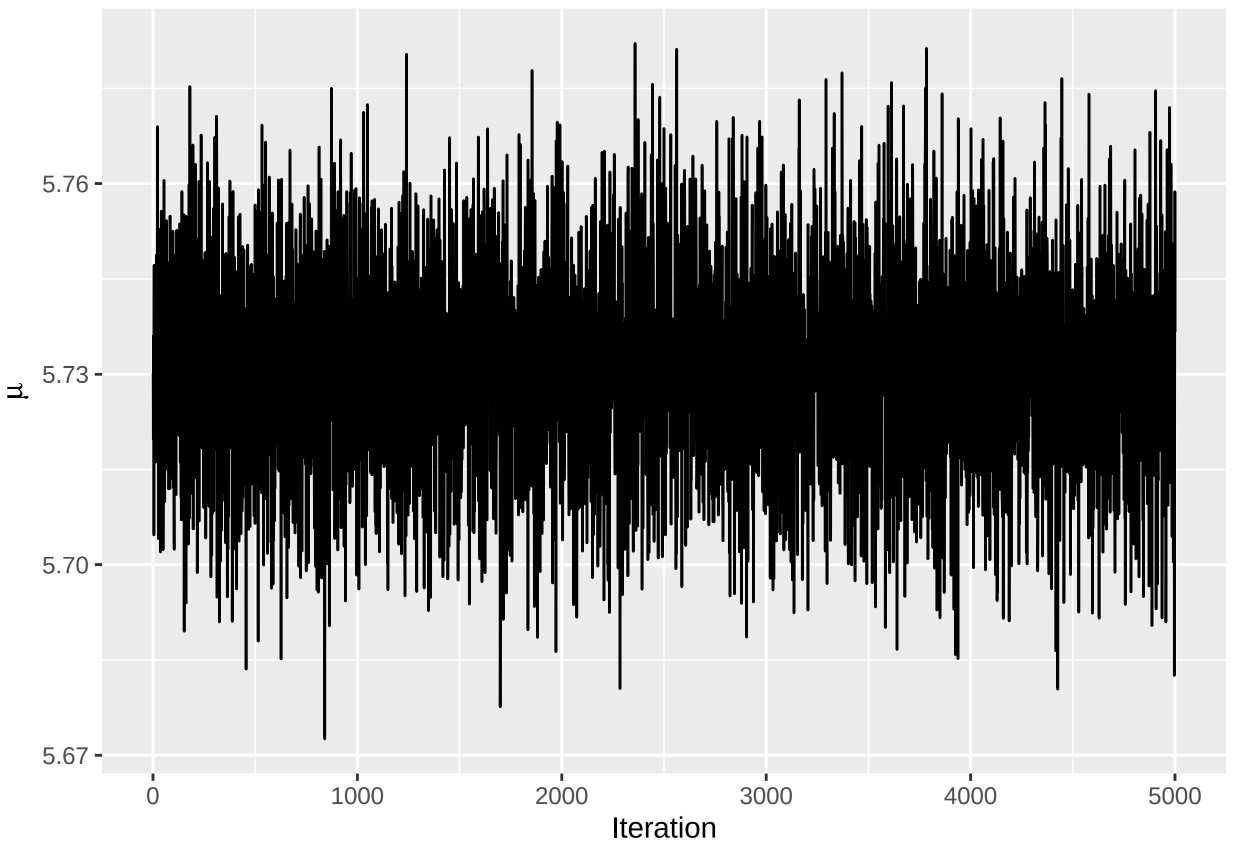
Normal model with unknown mean and unknown variance

Gibbs sampling

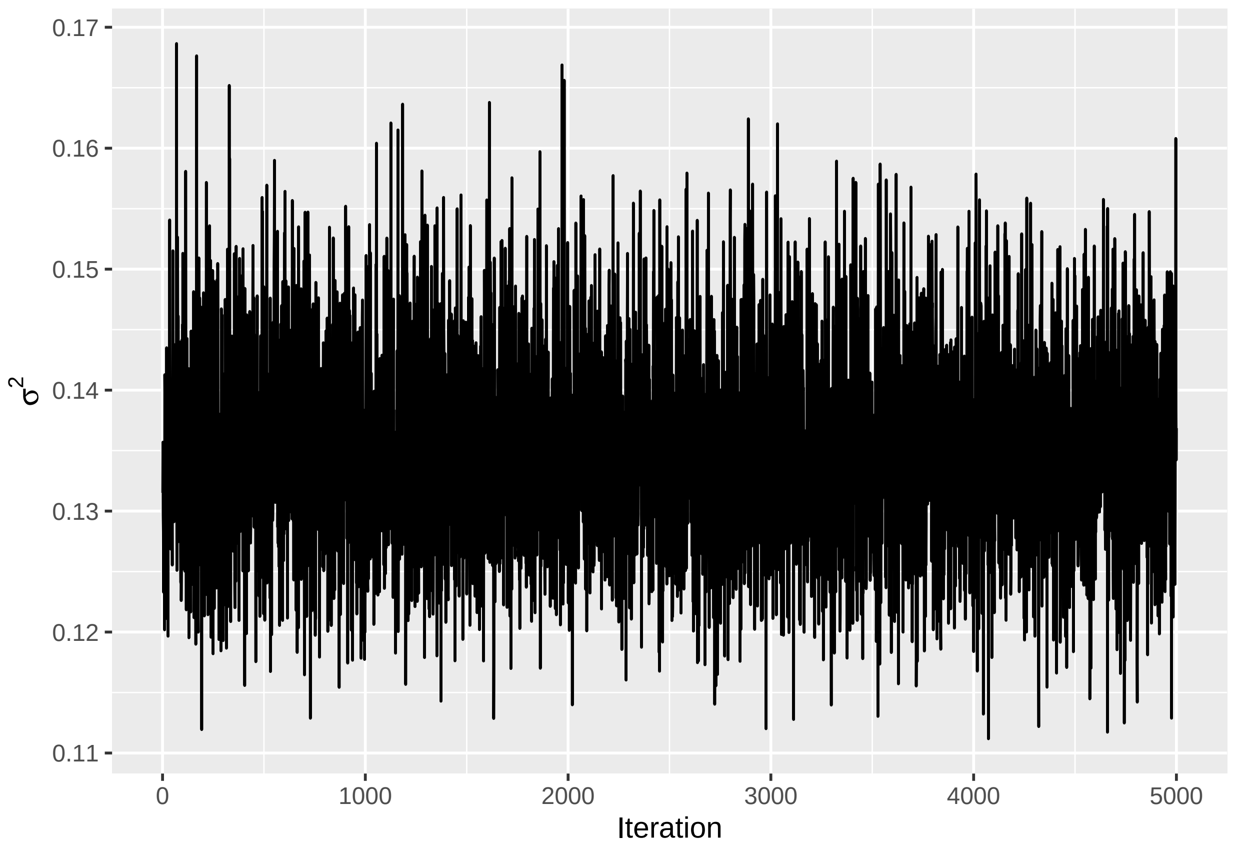
The following code implements the two-step Gibbs sampler to simulate draws from the joint posterior $p(\mu, \sigma^2 \mid \mathbf{y})$.

```
#number of iterations
S <- 5000
#storing the values
mu <- sigma2 <- numeric(S)

#prior information
mu0 <- 0
sigma02 <- 100
a <- b <- 1

#initial values
mu[1] <- mean(y)
sigma2[1] <- var(y)

# alternative initial values (random draw from the prior)
#mu[1] <- rnorm(1, mu0, sqrt(sigma02))
#sigma2[1] <- 1/rgamma(1, a, b)

for(i in 2:S){
  meanmu <- ((mu0/sigma02) + ((n*mean(y))/sigma2[i-1]))/((1/sigma02) + (n/sigma2[i-1]))
  varmu <- 1/((1/sigma02) + (n/sigma2[i-1]))
  mu[i] <- rnorm(1, meanmu, sqrt(varmu))

  a1 <- a + (n/2)
  b1 <- b + 0.5*sum((y - mu[i])^2)
  sigma2[i] <- 1/rgamma(1, a1, b1)
}

qplot(1:S, mu, geom = "line", ylab = expression(mu), xlab = "Iteration")
```

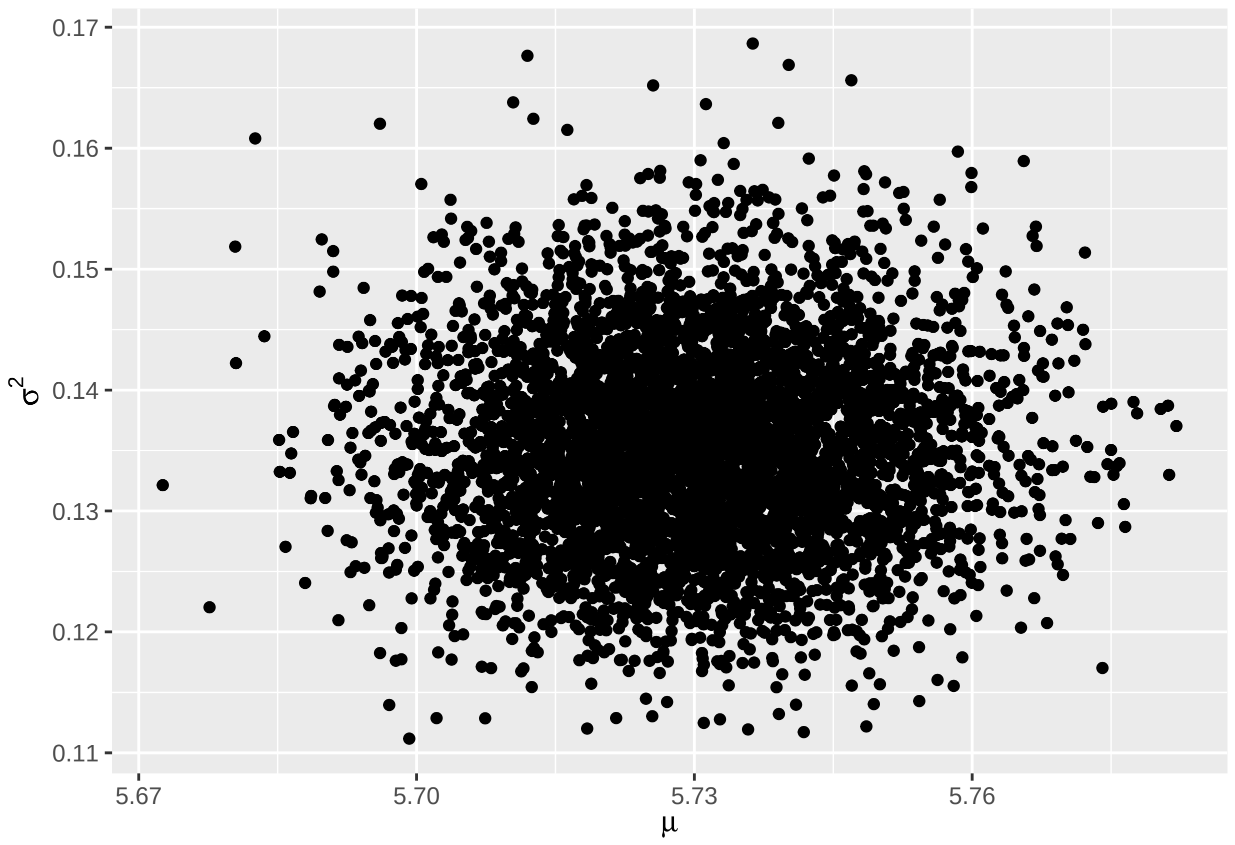
```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



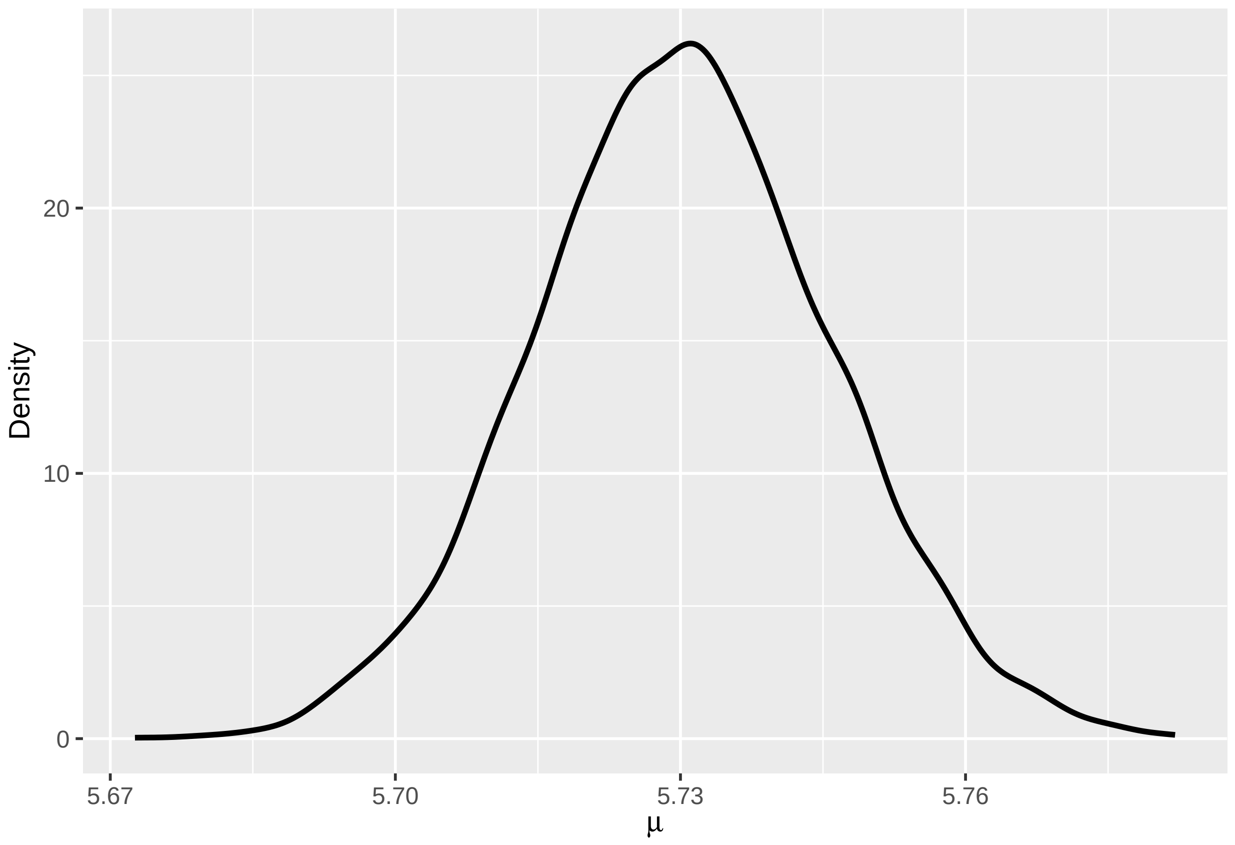
```r
qplot(1:S, sigma2, geom = "line", ylab = expression(sigma^2), xlab = "Iteration")
```



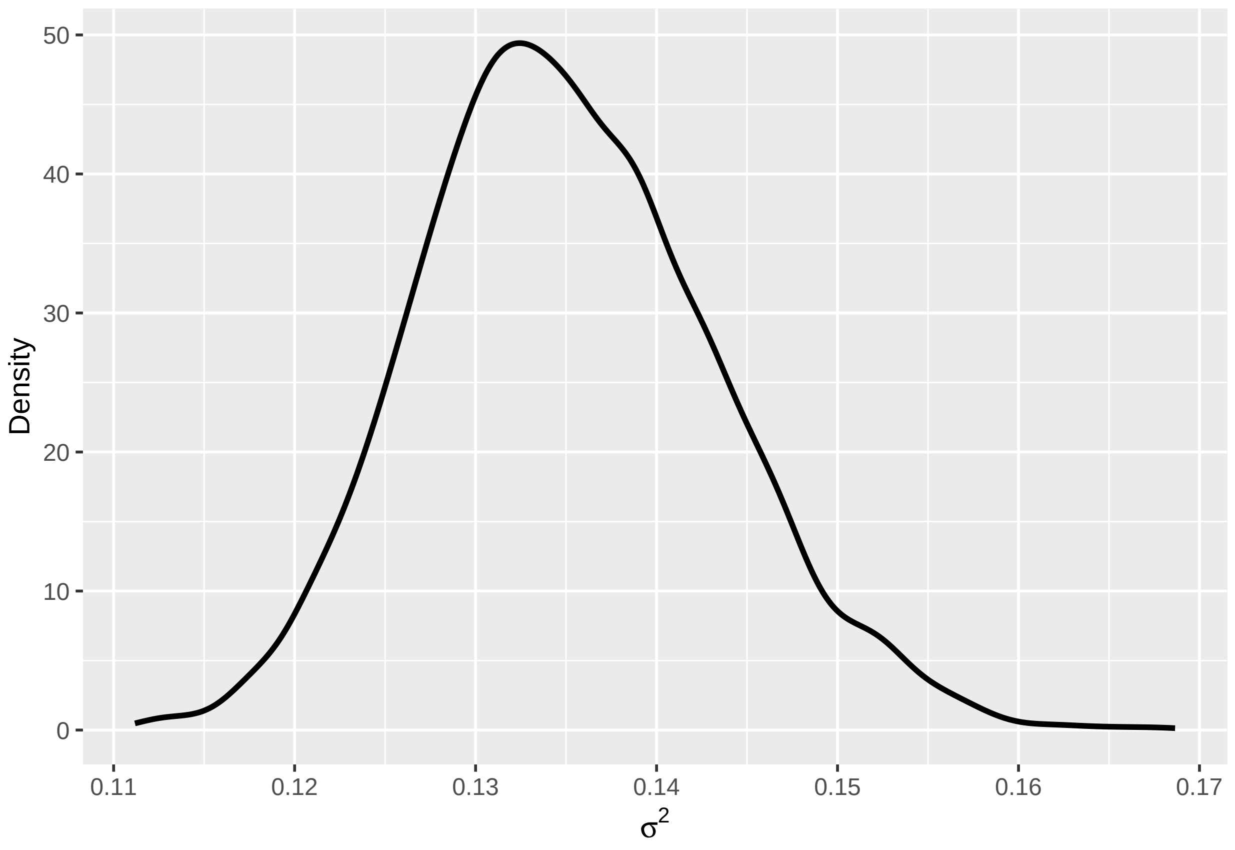
```
qplot(mu, sigma2, geom = "point", xlab = expression(mu), ylab = expression(sigma^2))
```



```
ggplot(data.frame("values" = mu), aes(x = values)) +
geom_density(size = 1) +
xlab(expression(mu)) +
ylab("Density")
```



```
ggplot(data.frame("values" = sigma2), aes(x = values)) +
geom_density(size = 1) +
xlab(expression(sigma^2)) +
ylab("Density")
```



Metropolis–Hastings sampling

```
#number of iterations
S <- 10000

#to use the digamma density function
require(pscl)

#log posterior distribution
log_post <- function(y, mu, sigma2, mu0, sigma02, a, b){
  if(sigma2 <= 0) {return(-Inf)}
  else{
  like <- sum(dnorm(y, mu, sqrt(sigma2), log = TRUE))
  prior_mu <- dnorm(mu, mu0, sqrt(sigma02), log = TRUE)
  prior_sigma2 <- log(densigamma(sigma2, a, b))
  post <- like + prior_mu + prior_sigma2
  return(post)
  }
}

#storing the values
mu <- sigma2 <- numeric(S)

#initial values
mu[1] <- mean(y)
```

```r
sigma2[1] <- var(y)

#std deviation of the proposal distribution
sd_mu_star <- 0.05
sd_sigma2_star <- 0.05

#count the number of candidate values accepted
count_accept_mu <- 0
count_accept_sigma2 <- 0

for(i in 2:S){
  mu_star <- rnorm(1, mu[i-1], sd_mu_star)
  log_r_mu <- log_post(y = y, mu = mu_star, sigma2 = sigma2[i-1],
                       mu0 = mu0, sigma02 = sigma02, a = a, b = b) -
  log_post(y = y, mu = mu[i-1], sigma2 = sigma2[i-1],
           mu0 = mu0, sigma02 = sigma02, a = a, b = b)

  if(log(runif(1)) < log_r_mu){
    mu[i] <- mu_star
    count_accept_mu <- count_accept_mu + 1
  }
  else{
    mu[i] <- mu[i-1]
  }

  sigma2_star <- rnorm(1, sigma2[i-1], sd_sigma2_star)
  log_r_sigma2 <- log_post(y = y, mu = mu[i], sigma2 = sigma2_star,
                           mu0 = mu0, sigma02 = sigma02, a = a, b = b) -
  log_post(y = y, mu = mu[i], sigma2 = sigma2[i-1],
          mu0 = mu0, sigma02 = sigma02, a = a, b = b)

  if(log(runif(1)) < log_r_sigma2){
    sigma2[i] <- sigma2_star
    count_accept_sigma2 <- count_accept_sigma2 + 1
  }
  else{
    sigma2[i] <- sigma2[i-1]
  }
}

(count_accept_mu/S)*100
```

```
## [1] 35.85
```

```r
(count_accept_sigma2/S)*100
```

```
## [1] 19.51
```

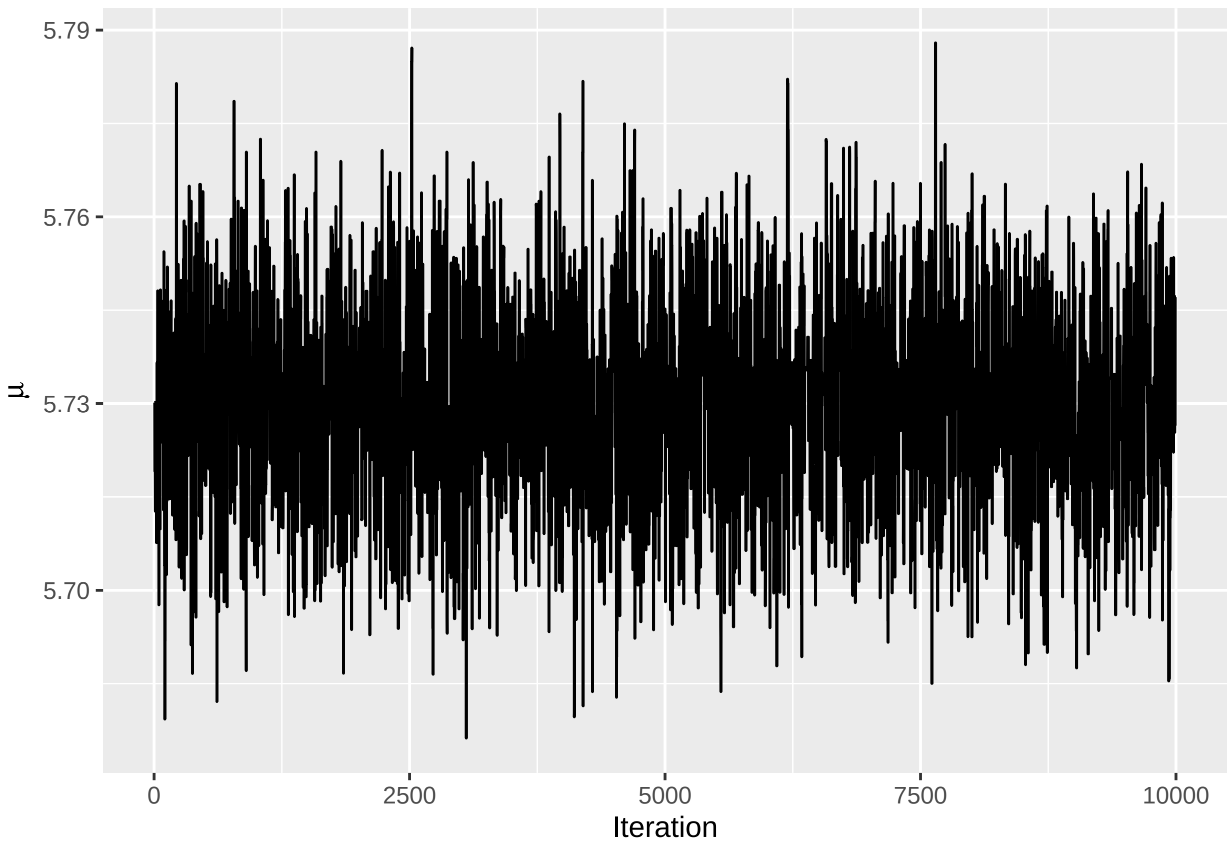
```r
quantile(mu, c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 5.699562 5.760361
```

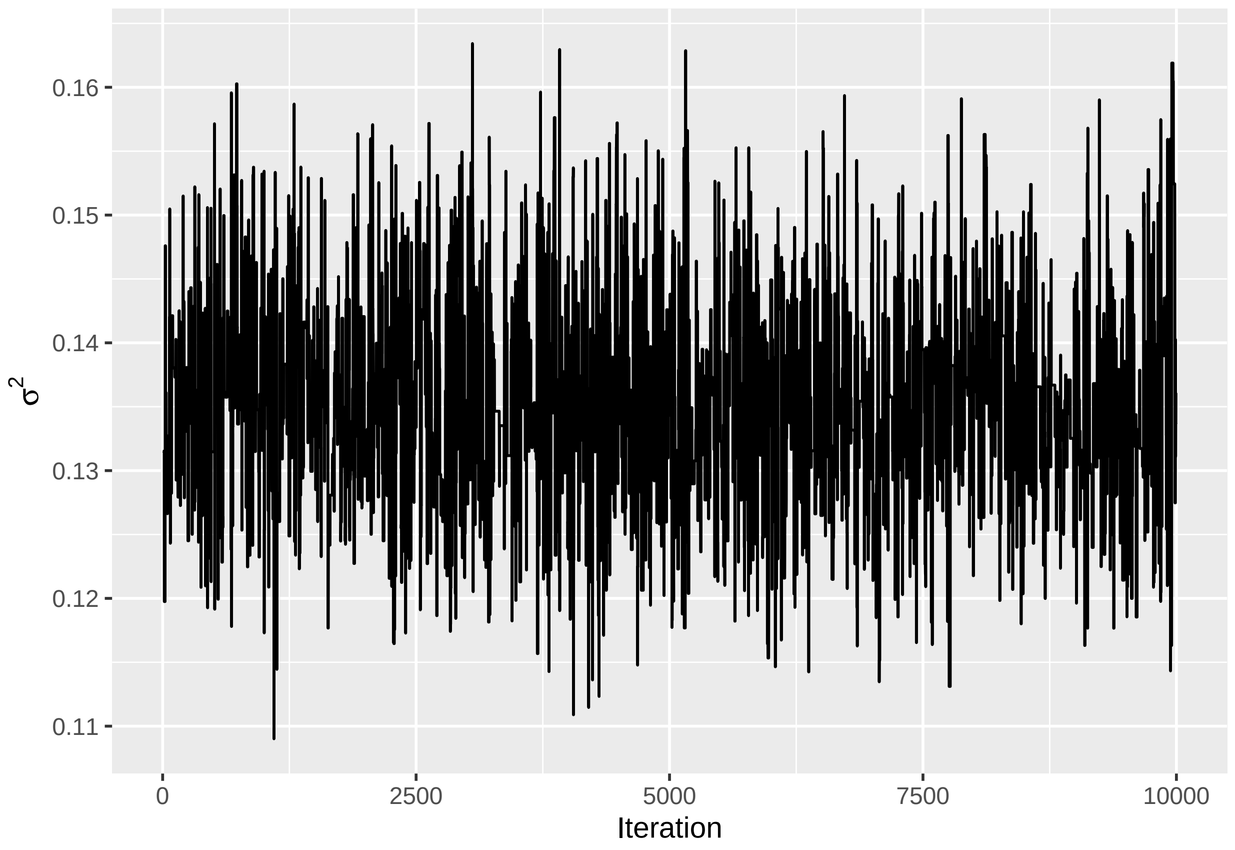
```
quantile(sigma2, c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.1203605 0.1527172
```

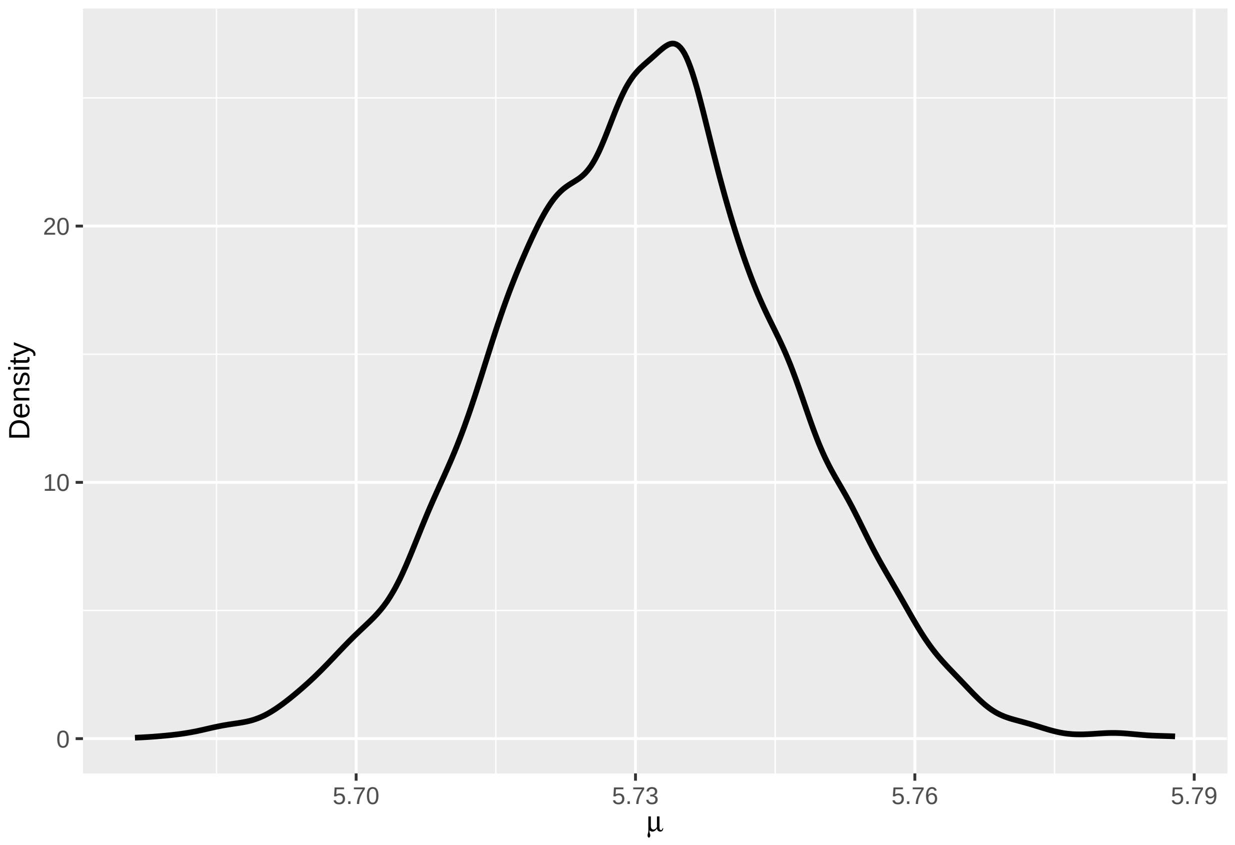
```
qplot(1:S, mu, geom = "line", ylab = expression(mu), xlab = "Iteration")
```



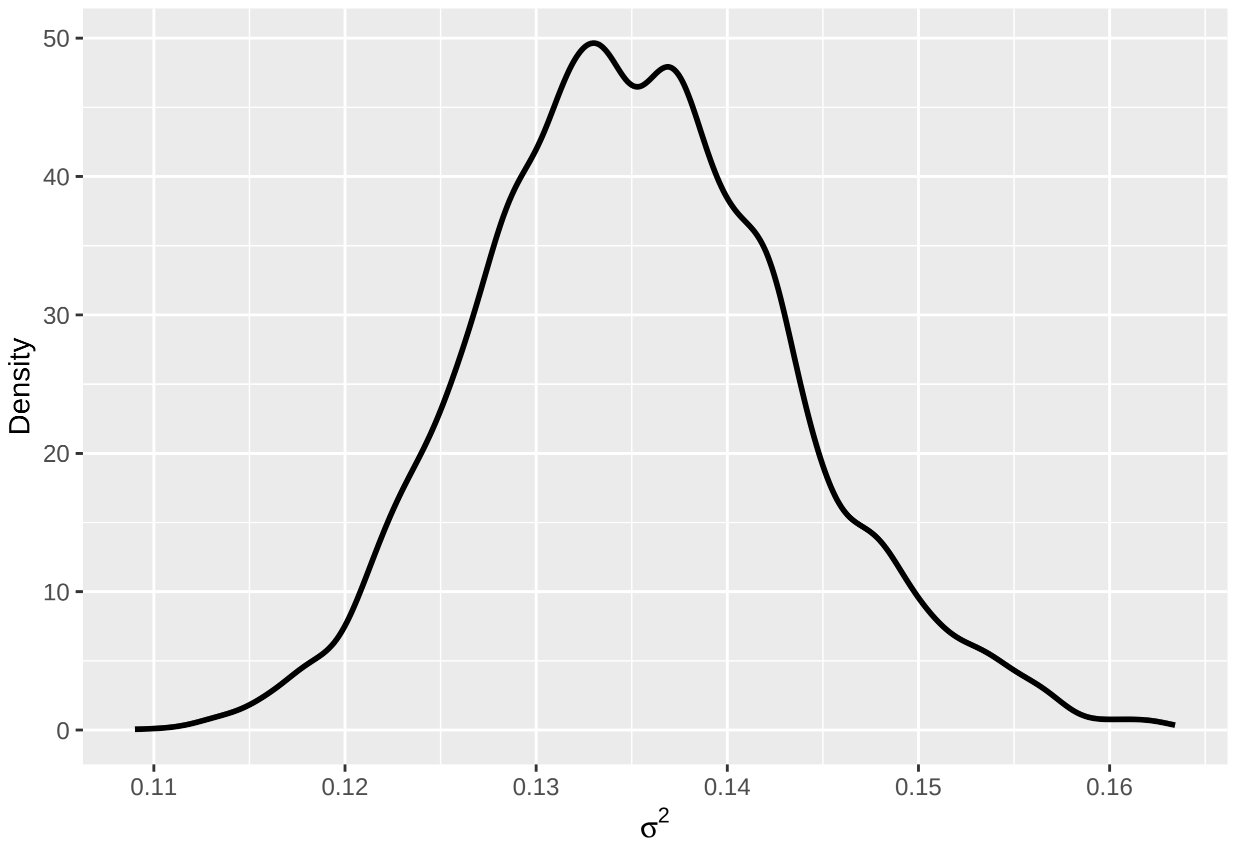
```
qplot(1:S, sigma2, geom = "line", ylab = expression(sigma^2), xlab = "Iteration")
```



```
ggplot(data.frame("values" = mu), aes(x = values)) +
geom_density(size = 1) +
xlab(expression(mu)) +
ylab("Density")
```



```
ggplot(data.frame("values" = sigma2), aes(x = values)) +
geom_density(size = 1) +
xlab(expression(sigma^2)) +
ylab("Density")
```



Metropolis within Gibbs

```
#number of iterations
S <- 10000

#to use the digamma density function
require(pscl)

#log posterior distribution
log_post <- function(y, mu, sigma2, mu0, sigma02, a, b){
  if(sigma2 <= 0) {return(-Inf)}
  else{
  like <- sum(dnorm(y, mu, sqrt(sigma2), log = TRUE))
  prior_mu <- dnorm(mu, mu0, sqrt(sigma02), log = TRUE)
  prior_sigma2 <- log(densigamma(sigma2, a, b))
  post <- like + prior_mu + prior_sigma2
  return(post)
  }
}

mu <- sigma2 <- numeric(S)
mu[1] <- mean(y)
sigma2[1] <- var(y)

sd_sigma2_star <- 0.05
```

```r
count_accept_sigma2 <- 0

for(i in 2:S){
  meanmu <- ((mu0/sigma02) + ((n*mean(y))/sigma2[i-1]))/((1/sigma02) + (n/sigma2[i-1]))
  varmu <- 1/((1/sigma02) + (n/sigma2[i-1]))
  mu[i] <- rnorm(1, meanmu, sqrt(varmu))

  sigma2_star <- rnorm(1, sigma2[i-1], sd_sigma2_star)

  log_r_sigma2 <- log_post(y = y, mu = mu[i], sigma2 = sigma2_star,
                           mu0 = mu0, sigma02 = sigma02, a = a, b = b) -
                  log_post(y = y, mu = mu[i], sigma2 = sigma2[i-1],
                           mu0 = mu0, sigma02 = sigma02, a = a, b = b)

  if(log(runif(1)) < log_r_sigma2){
    sigma2[i] <- sigma2_star
    count_accept_sigma2 <- count_accept_sigma2 + 1
  }
  else{
    sigma2[i] <- sigma2[i-1]
  }
}

(count_accept_sigma2/S)*100
```

```
## [1] 19.78
```

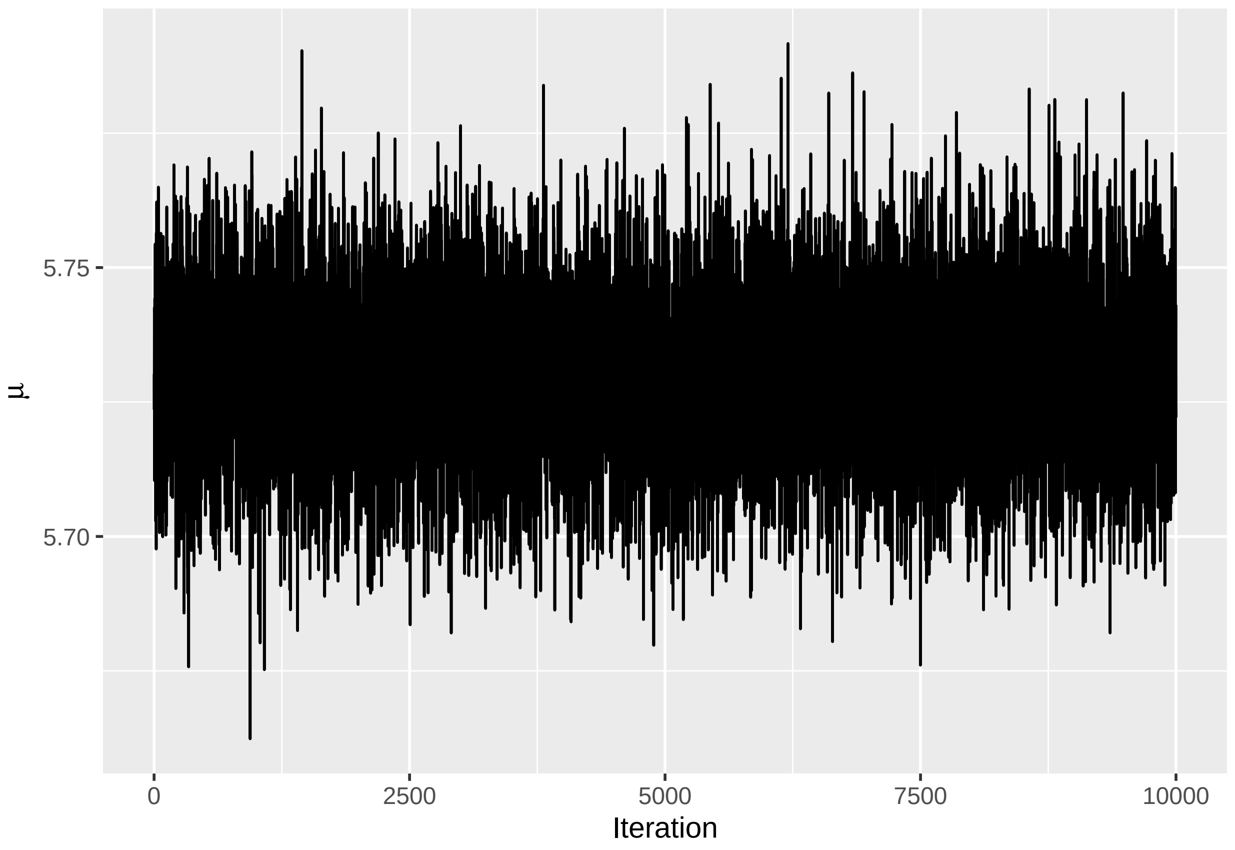
```r
quantile(mu, c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 5.699445 5.760546
```

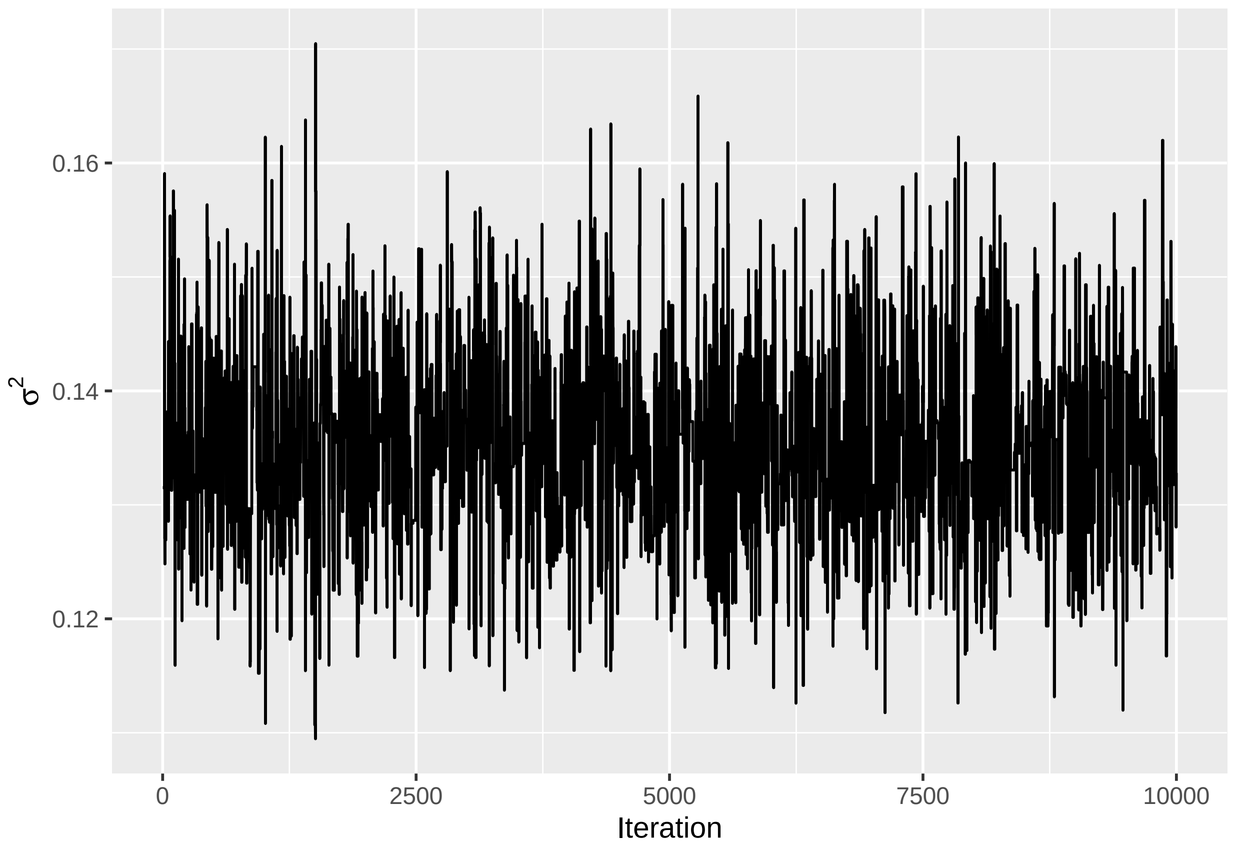
```r
quantile(sigma2, c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.1202075 0.1522488
```

```r
qplot(1:S, mu, geom = "line", ylab = expression(mu), xlab = "Iteration")
```



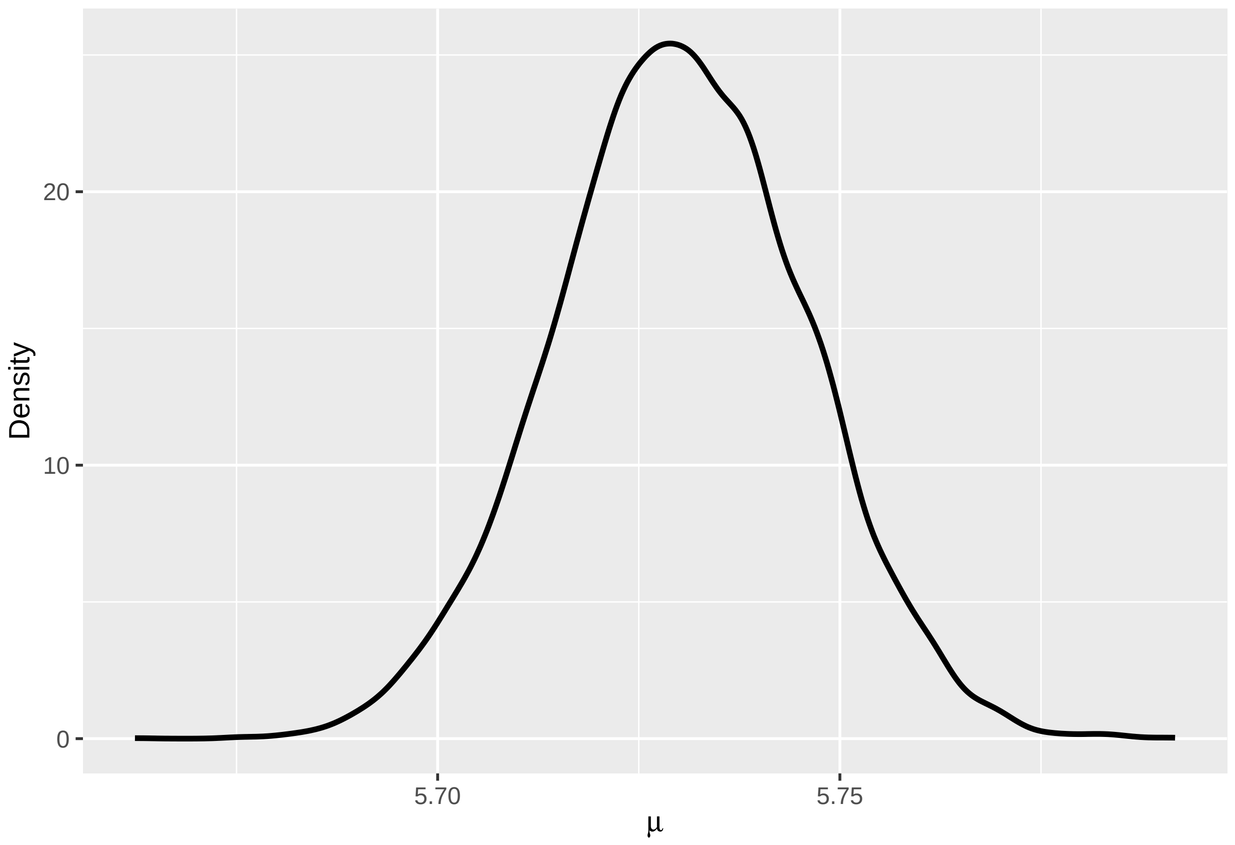
```
qplot(1:S, sigma2, geom = "line", ylab = expression(sigma^2), xlab = "Iteration")
```



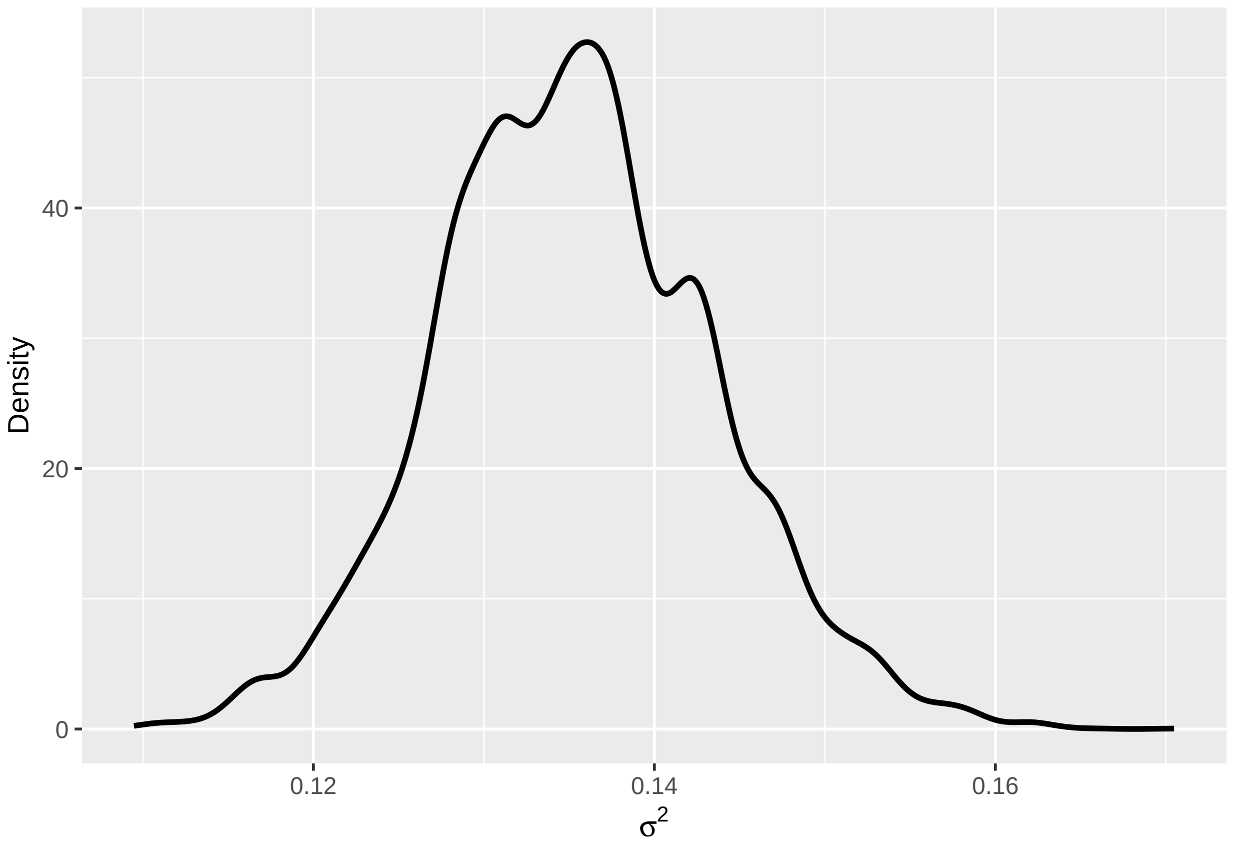
```
require(coda)
effectiveSize(sigma2)
```

```
##     var1
## 1585.306
```

```
ggplot(data.frame("values" = mu), aes(x = values)) +
geom_density(size = 1) +
xlab(expression(mu)) +
ylab("Density")
```



```
ggplot(data.frame("values" = sigma2), aes(x = values)) +
geom_density(size = 1) +
xlab(expression(sigma^2)) +
ylab("Density")
```



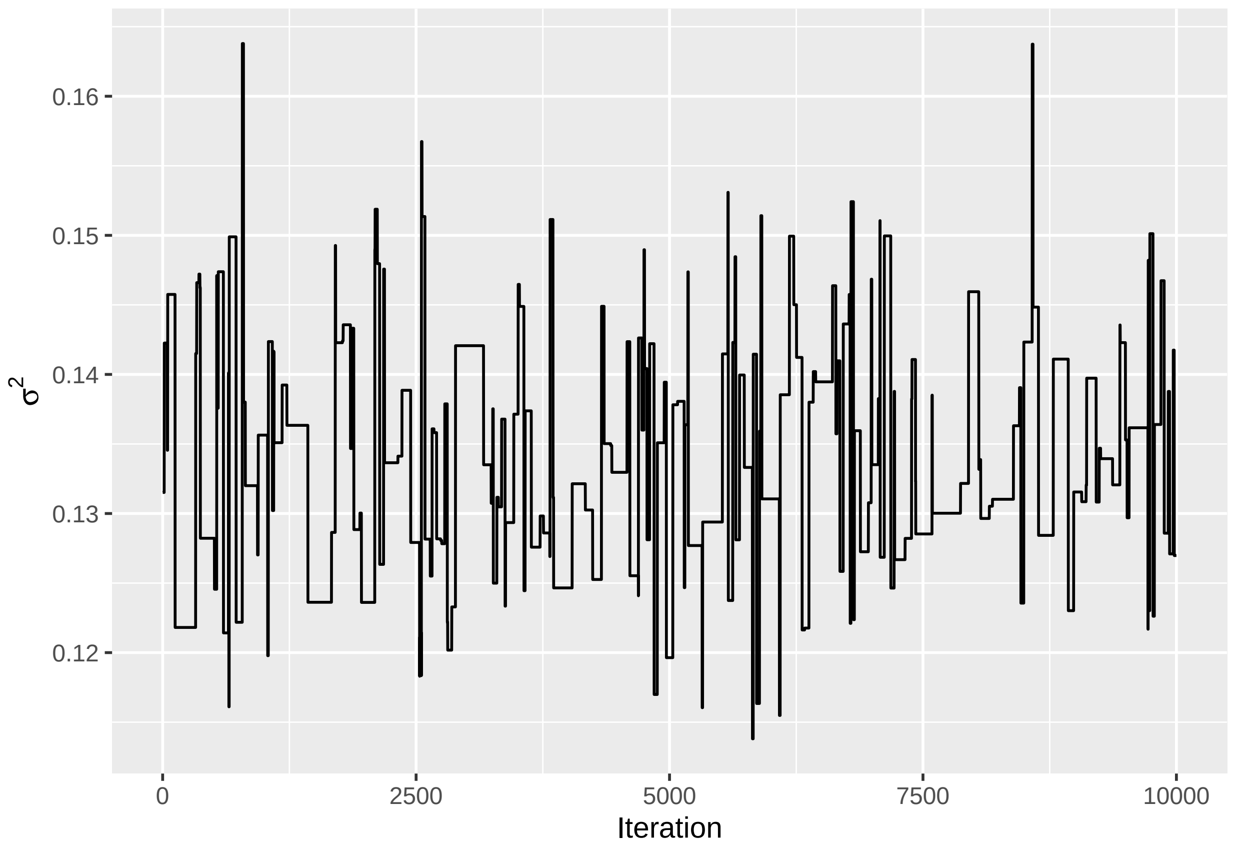
We will now illustrate how the variance of the proposal distribution (in this case for σ^2) can affect how the chain moves and consequently the effective sample size of the collection of draws. Before we have used 0.05 for the standard deviation of the proposal distribution. As an extreme case we will now consider it to be 0.5.

```r
mu0 <- 0
sigma02 <- 100
a <- 1
b <- 1

S <- 10000

log_post <- function(y, mu, sigma2, mu0, sigma02, a, b){
  if(sigma2 <= 0) {return(-Inf)}
  else{
    like <- sum(dnorm(y, mu, sqrt(sigma2), log = TRUE))
    prior_mu <- dnorm(mu, mu0, sqrt(sigma02), log = TRUE)
    prior_sigma2 <- log(densigamma(sigma2, a, b))
    post <- like + prior_mu + prior_sigma2
    return(post)
  }
}

mu <- sigma2 <- numeric(S)
mu[1] <- mean(y)
sigma2[1] <- var(y)
```

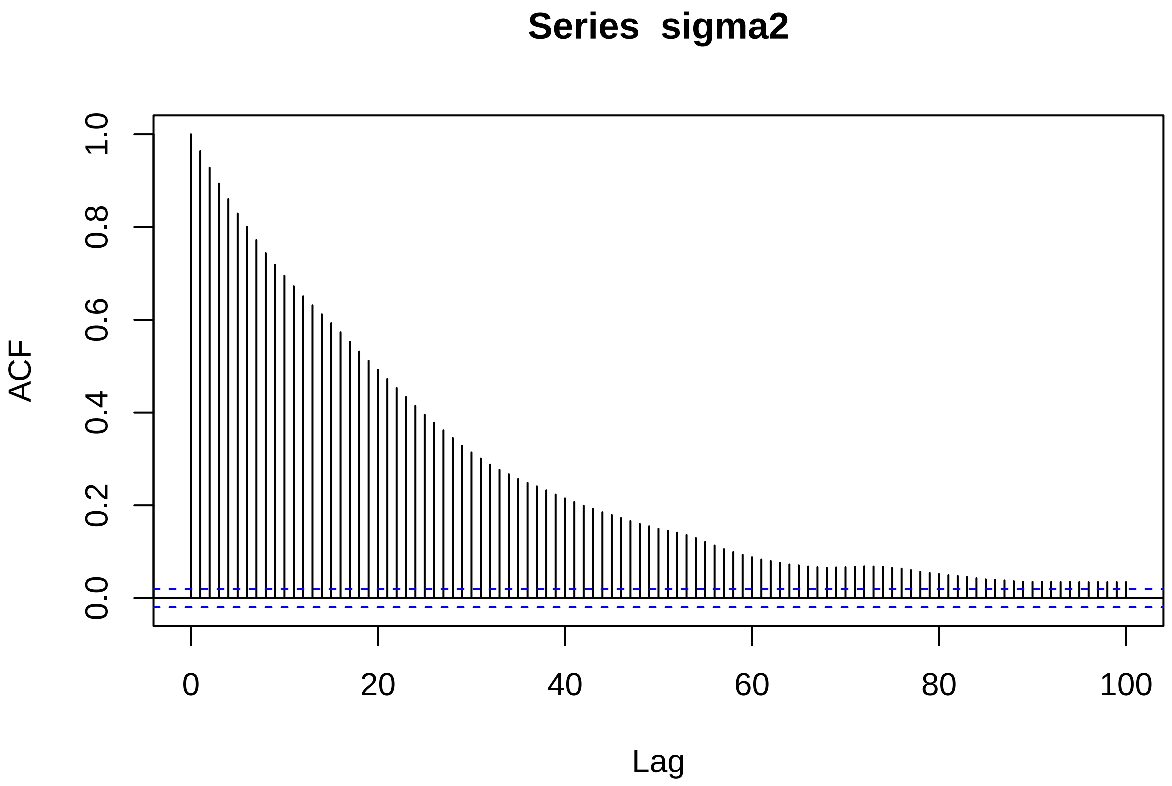
```r
sd_sigma2_star <- 0.5
count_accept_sigma2 <- 0

for(i in 2:S){
  meanmu <- ((mu0/sigma02) + ((n*mean(y))/sigma2[i-1]))/((1/sigma02) + (n/sigma2[i-1]))
  varmu <- 1/((1/sigma02) + (n/sigma2[i-1]))
  mu[i] <- rnorm(1, meanmu, sqrt(varmu))

  sigma2_star <- rnorm(1, sigma2[i-1], sd_sigma2_star)

  log_r_sigma2 <- log_post(y = y, mu = mu[i], sigma2 = sigma2_star,
                           mu0 = mu0, sigma02 = sigma02, a = a, b = b) -
    log_post(y = y, mu = mu[i], sigma2 = sigma2[i-1],
             mu0 = mu0, sigma02 = sigma02, a = a, b = b)

  if(log(runif(1)) < log_r_sigma2){
    sigma2[i] <- sigma2_star
    count_accept_sigma2 <- count_accept_sigma2 + 1
  }
  else{
    sigma2[i] <- sigma2[i-1]
  }
}

(count_accept_sigma2/S)*100
```

```
## [1] 2.11
```

```r
qplot(1:S, sigma2, geom = "line", ylab = expression(sigma^2), xlab = "Iteration")
```



```r
acf(sigma2, lag.max = 100)
```



```
effectiveSize(sigma2)
```

```
##     var1
## 173.9694
```

We can observe that the chain gets stuck at the same values sometimes for a long period (this is due to proposing candidate values that are in a region of low posterior density and, as such, they are frequently rejected, so that the chain stays in the same value). The acceptance probability of the candidate values is very low (only 2%) and the effective sample size is also quite low.\

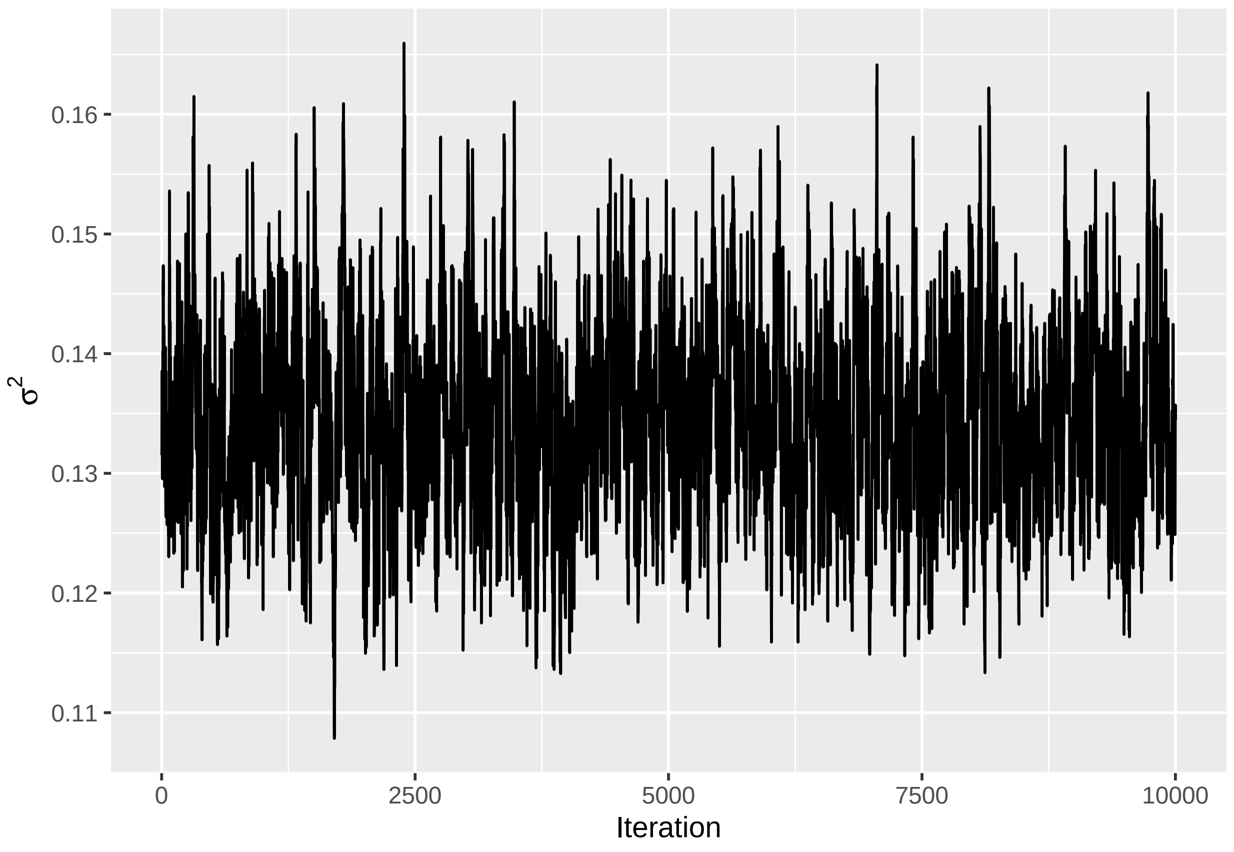
The other extreme case will now be considered. We will consider the case where the variance of the proposal distribution is too low. This results in small moves, and so there is also a high (or higher) autocorrelation between the draws.

```
mu0 <- 0
sigma02 <- 100
a <- 1
b <- 1

S <- 10000

log_post <- function(y, mu, sigma2, mu0, sigma02, a, b){
  if(sigma2 <= 0) {return(-Inf)}
  else{
    like <- sum(dnorm(y, mu, sqrt(sigma2), log = TRUE))
    prior_mu <- dnorm(mu, mu0, sqrt(sigma02), log = TRUE)
    prior_sigma2 <- log(densigamma(sigma2, a, b))
    post <- like + prior_mu + prior_sigma2
    return(post)
  }
}

mu <- sigma2 <- numeric(S)
mu[1] <- mean(y)
sigma2[1] <- var(y)

sd_sigma2_star <- 0.005
count_accept_sigma2 <- 0

for(i in 2:S){
  meanmu <- ((mu0/sigma02) + ((n*mean(y))/sigma2[i-1]))/((1/sigma02) + (n/sigma2[i-1]))
  varmu <- 1/((1/sigma02) + (n/sigma2[i-1]))
  mu[i] <- rnorm(1, meanmu, sqrt(varmu))

  sigma2_star <- rnorm(1, sigma2[i-1], sd_sigma2_star)

  log_r_sigma2 <- log_post(y = y, mu = mu[i], sigma2 = sigma2_star,
                           mu0 = mu0, sigma02 = sigma02, a = a, b = b) -
    log_post(y = y, mu = mu[i], sigma2 = sigma2[i-1],
             mu0 = mu0, sigma02 = sigma02, a = a, b = b)

  if(log(runif(1)) < log_r_sigma2){
    sigma2[i] <- sigma2_star
```

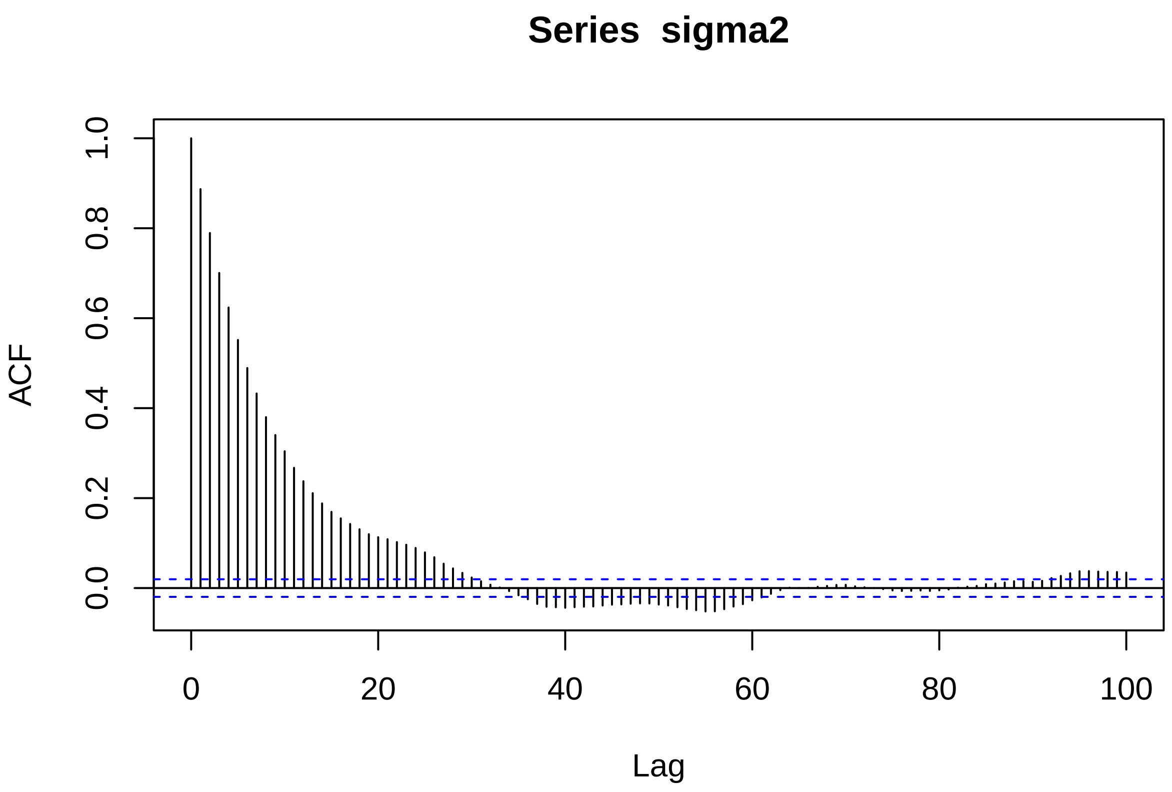
```r
    count_accept_sigma2 <- count_accept_sigma2 + 1
  }
  else{
    sigma2[i] <- sigma2[i-1]
  }
}

(count_accept_sigma2/S)*100
```

```
## [1] 80.37
```

```r
qplot(1:S, sigma2, geom = "line", ylab = expression(sigma^2), xlab = "Iteration")
```



```r
acf(sigma2, lag.max = 100)
```



```
effectiveSize(sigma2)
```

```
##  var1
## 609.3
```

The acceptance probability is now too high (around 80%), but the effective sample size is 651 (compare this against the value of 1261 when we were using 0.05 as the variance of the proposal).

Illustrating the use of JAGS

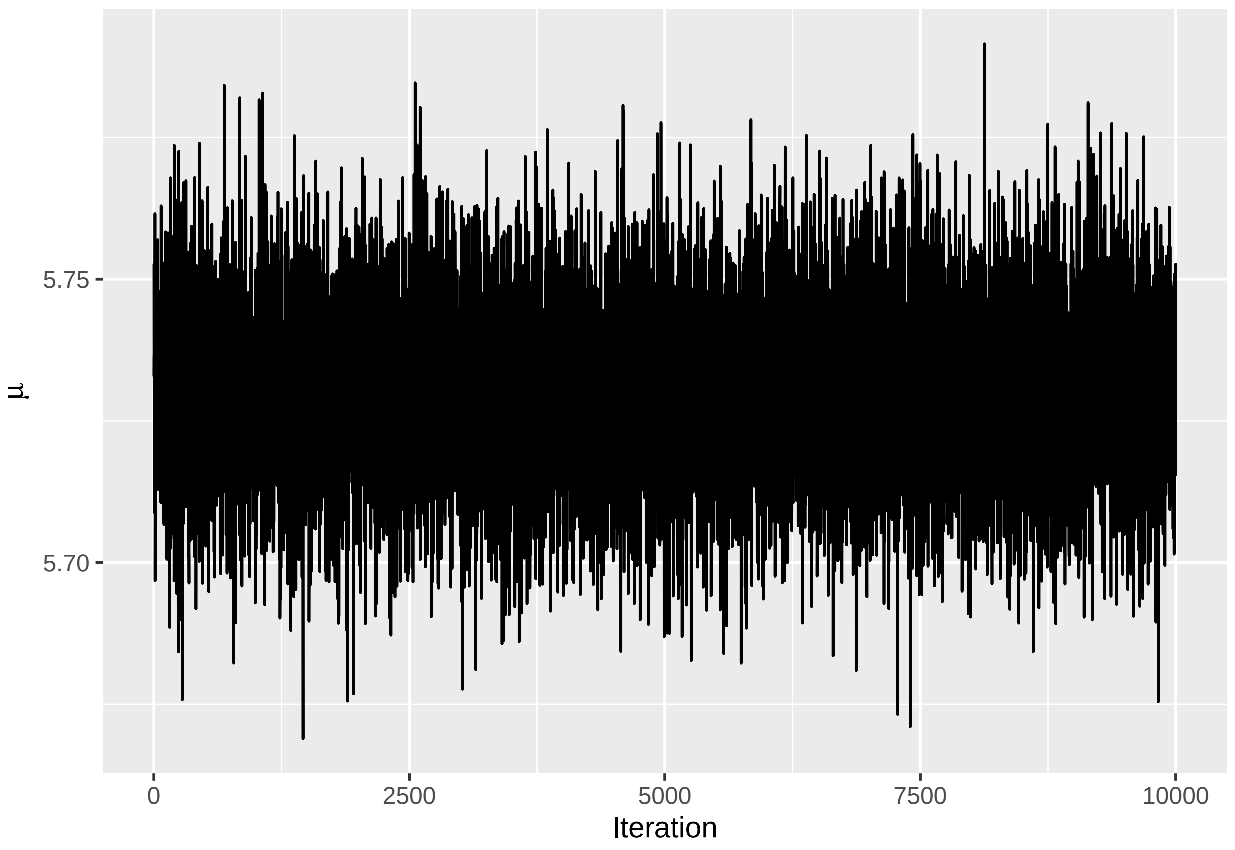
```
require(rjags)
mu0 <- 0
sigma02 <- 100
a <- b <- 1
n <- length(y)
nburn <- 2000
nsim <- 10000

model_string <- "model{
for(i in 1:n){
y[i]~dnorm(mu,tau)
}

mu ~ dnorm(mu0, tau0)
tau0 <- 1/sigma02
tau ~ dgamma(a, b)
sigma2 <- 1/tau
}"

data <- list(n = n, y = y, mu0 = mu0, sigma02 = sigma02, a = a, b = b)

#list of initial values (if not supplied, the function jags.model will generate initial values)
inits <- list(mu = mean(y), tau = 1/var(y))

#passing the model to jags *format*
model <- jags.model(textConnection(model_string), n.chains = 1, data = data, inits = inits)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 563
##    Unobserved stochastic nodes: 2
##    Total graph size: 573
##
## Initializing model
```

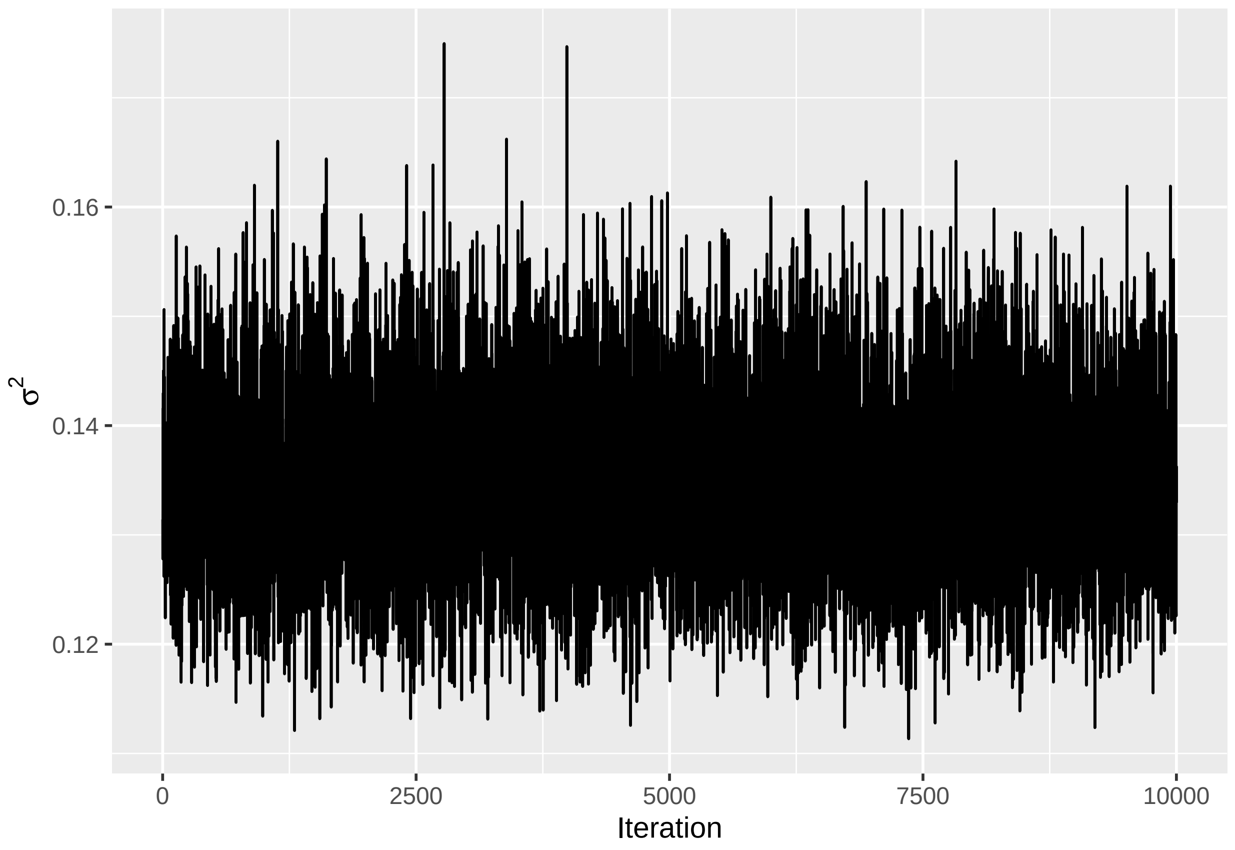
```
#burnin of 1000 samples
update(model, nburn)

#running the model
res <- jags.samples(model,
                    variable.names=c("mu", "sigma2"),
                    n.iter = nsim)
str(res)
```

```
## List of 2
##  $ mu    : 'mcarray' num [1, 1:10000, 1] 5.75 5.73 5.74 5.72 5.73 ...
##   ..- attr(*, "varname")= chr "mu"
##   ..- attr(*, "type")= chr "trace"
##   ..- attr(*, "iterations")= Named num [1:3] 2001 12000 1
##   .. ..- attr(*, "names")= chr [1:3] "start" "end" "thin"
##  $ sigma2: 'mcarray' num [1, 1:10000, 1] 0.131 0.128 0.137 0.142 0.137 ...
##   ..- attr(*, "varname")= chr "sigma2"
##   ..- attr(*, "type")= chr "trace"
##   ..- attr(*, "iterations")= Named num [1:3] 2001 12000 1
##   .. ..- attr(*, "names")= chr [1:3] "start" "end" "thin"
```

```
mu <- res$mu[1, ,1]
sigma2 <- res$sigma2[1, ,1]

qplot(1:nsim, mu, geom = "line", ylab = expression(mu), xlab = "Iteration")
```

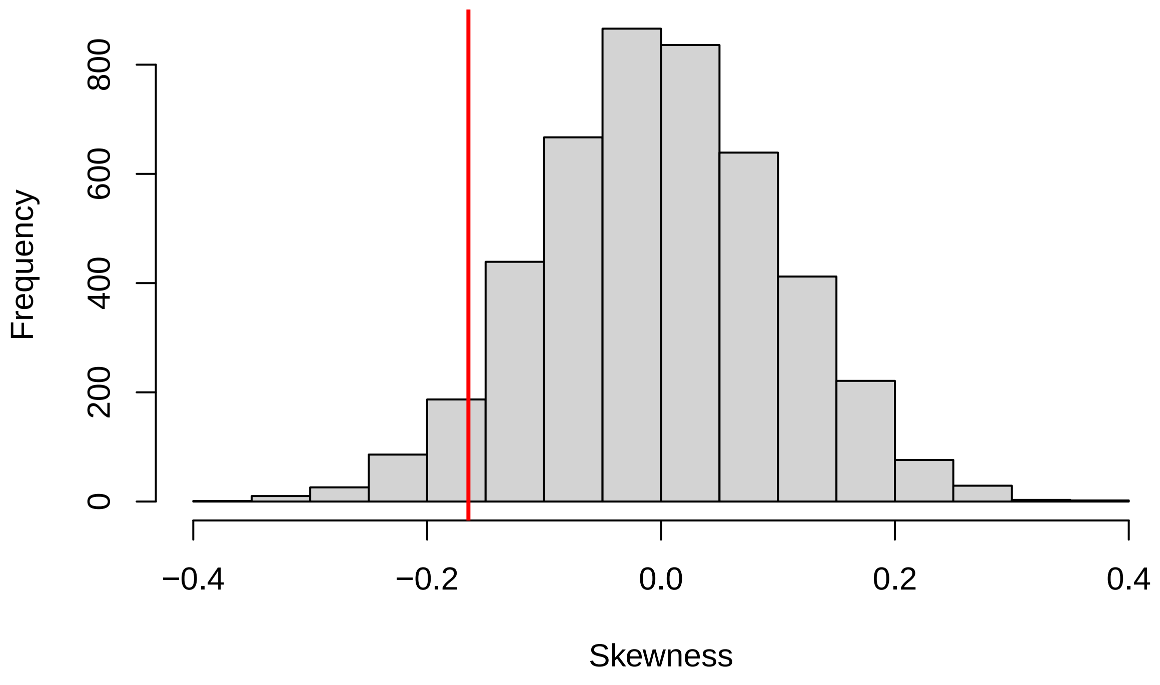


```r
qplot(1:nsim, sigma2, geom = "line", ylab = expression(sigma^2), xlab = "Iteration")
```

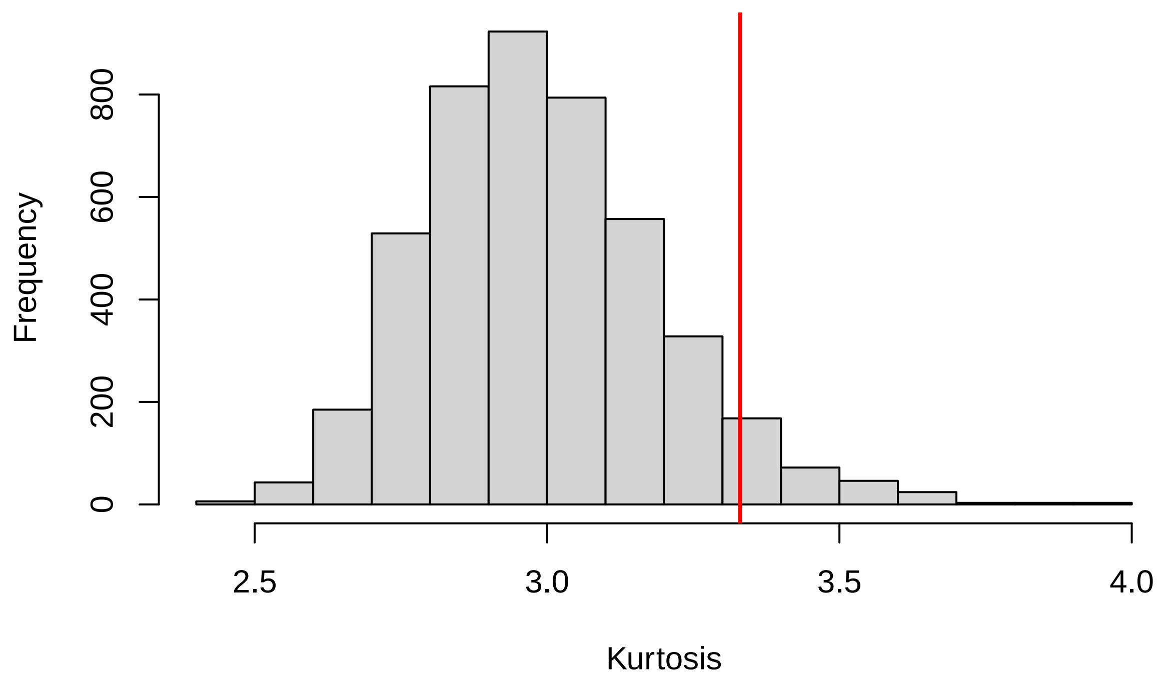


Posterior predicitive checks

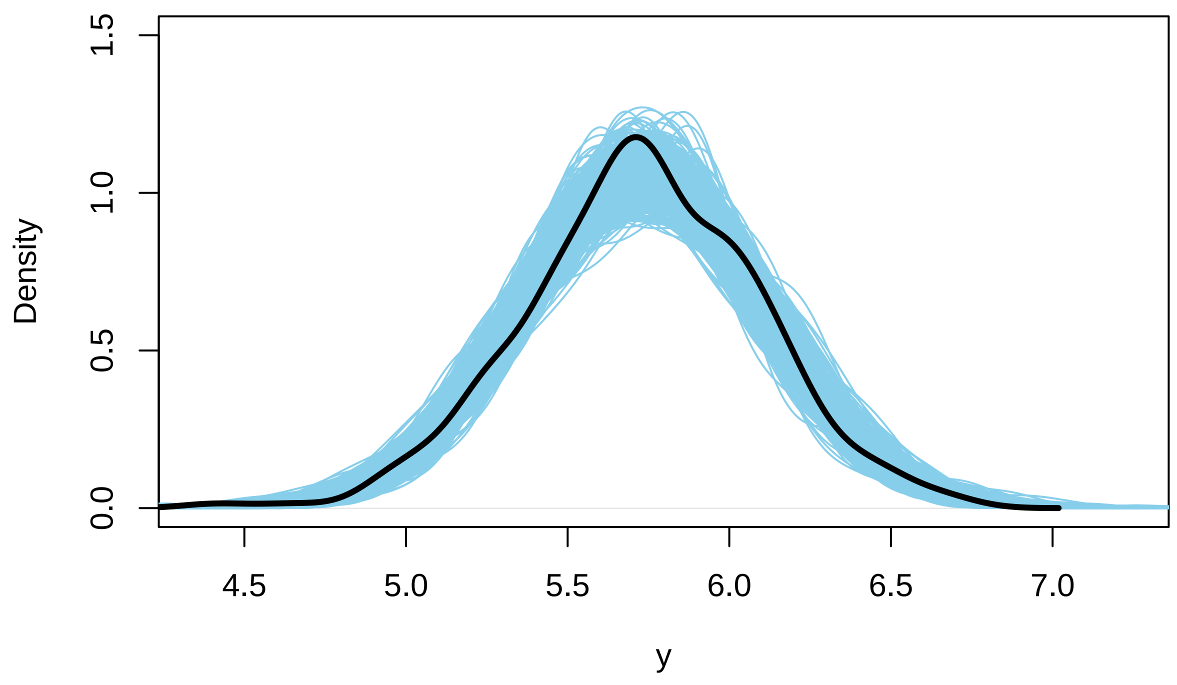
```
data_chol <- read.csv2("cholesterol.txt")
data_chol <- as.numeric(data_chol[,1])
y <- log(data_chol)
n <- length(y)

S <- 5000
#storing the values
mu <- sigma2 <- numeric(S)

#prior information
mu0 <- 0
sigma02 <- 100
a <- b <- 1

#initial values
mu[1] <- mean(y)
sigma2[1] <- var(y)

set.seed(123)
for(i in 2:S){
  meanmu <- ((mu0/sigma02) + ((n*mean(y))/sigma2[i-1]))/((1/sigma02) + (n/sigma2[i-1]))
  varmu <- 1/((1/sigma02) + (n/sigma2[i-1]))
  mu[i] <- rnorm(1, meanmu, sqrt(varmu))

  a1 <- a + (n/2)
  b1 <- b + 0.5*sum((y - mu[i])^2)
  sigma2[i] <- 1/rgamma(1, a1, b1)
}

nburn <- 500
ypred <- matrix(0, nrow = n, ncol = (S-nburn))
for(i in (nburn + 1): S){
ypred[, i-nburn] <- rnorm(n, mu[i], sqrt(sigma2[i]))
}

require(moments)
skew_ypred <- apply(ypred, 2, skewness)
kurtosis_ypred <- apply(ypred, 2, kurtosis)

hist(skew_ypred, main = "", xlab = "Skewness")
abline(v = skewness(y), col = "red", lwd =2)
```



```
hist(kurtosis_ypred, main = "", xlab = "Kurtosis")
abline(v = kurtosis(y), col = "red", lwd =2)
```



```
set.seed(123)
ind <- sample(x = (S-nburn), size = 500, replace = FALSE)

plot(density(ypred[,ind[1]]), col = "skyblue", ylim = c(0, 1.5),
     xlab = expression(y), main = "")
for(i in 2:length(ind)){
  lines(density(ypred[, ind[i]]), col = "skyblue")
}
lines(density(y), col = "black", lwd = 3)
```


Density
y