BDA - Assignment 5

Anonymous

Contents

Metropolis algorithm: Replicate the computations for the bioassay example of section 3.7 in BDA3 using the Metropolis algorithm. The Metropolis algorithm is described in BDA3 Chapter 11.2. More information on the bioassay data can be found in Section 3.7 in BDA3, and in Chapter 3 notes.

Exercise 1

Implement the Metropolis algorithm as an R function for the bioassay data. Use the Gaussian prior as in Assignment 4, that is $\begin{bmatrix}\alpha\\ \beta\end{bmatrix} \sim N(\mu_0, \Sigma_0)$, where $\mu_0 = \begin{bmatrix}0\\ 10\end{bmatrix}$ *and* $\Sigma_0 = \begin{bmatrix}2^2 & 12\\ 12 & 10^2\end{bmatrix}$.

a)

Start by implementing a function called density_ratio to compute the density ratio function, r in Eq. (11.1) in BDA3. Below is an example on how the function should work. You can test the function using markmyassignment.

```
data("bioassay")
bioassay <- bioassay
density_ratio <- function(alpha_propose, alpha_previous, beta_propose,
                          beta_previous, x, y, n) {

  unnorm_log_likelihood_propose = bioassaylp(alpha_propose, beta_propose, x, y, n)
  unnorm_log_likelihood_previous = bioassaylp(alpha_previous, beta_previous, x, y, n)

  prior_mean_vector = c(0, 10)
  prior_cov_matrix = matrix(c(4, 12, 12, 100), nrow=2, ncol=2, byrow = TRUE)

  gaussian_prior_propose <- dmvnorm(c(alpha_propose, beta_propose),
                                    prior_mean_vector, prior_cov_matrix, log=TRUE)
  gaussian_prior_previous <- dmvnorm(c(alpha_previous, beta_previous),
                                     prior_mean_vector, prior_cov_matrix, log=TRUE)


  log_post_propose = unnorm_log_likelihood_propose + gaussian_prior_propose
  log_post_previous = unnorm_log_likelihood_previous + gaussian_prior_previous

  density_ratio_value = exp(log_post_propose - log_post_previous)
  return(density_ratio_value)
}

density_ratio(alpha_propose = 1.89, alpha_previous = 0.374, beta_propose = 24.76,
              beta_previous = 20.04, x = bioassay$x, y = bioassay$y, n = bioassay$n)
```

```
## [1] 1.305179
```

```
## [1] 1.305179

density_ratio(alpha_propose = 0.374, alpha_previous = 1.89, beta_propose = 20.04,
              beta_previous = 24.76, x = bioassay$x, y = bioassay$y, n = bioassay$n)
```

```
## [1] 0.7661784
```

```
## [1] 0.7661784

# mark_my_assignment()
```

Hint! Compute with log-densities. Reasons are explained on page 261 of BDA3 and lecture video 4.1. Remember that $p_1/p_0 = exp(log(p_1) - log(p_0))$. For your convenience we have provided functions that will evaluate the log-likelihood for given α and β (see bioassaylp() in the aaltobda package). Notice that you still need to add the prior yourself and remember the unnormalized log posterior is simply the sum of log-likelihood and log-prior. For evaluating the log of the Gaussian prior you can use the function dmvnorm from package aaltobda.

b)

Now implement a function called metropolis_bioassay() which implements the Metropolis algorithm using the density_ratio(). Hint! Use a simple (normal) proposal distribution. Example proposals are $\alpha^* \sim N(\alpha_{t-1}, \sigma = 1)$ and $\beta^* \sim N(\beta_{t-1}, \sigma = 5)$. There is no need to try to find optimal proposal but test some different values for the jump scale (σ). Remember to report the one you used. Efficient proposals are dicussed in BDA3 p. 295–297 (not part of the course). In real-life a pre-run could be made with an automatic adaptive control to adapt the proposal distribution.

```
metropolis_bioassay <- function(iters, warm_up, alpha_initial, beta_initial,
                                alpha_sd, beta_sd, x, y, n){

  alpha_previous <- alpha_initial
  beta_previous <- beta_initial
  alpha_array <- c(alpha_previous)
  beta_array <- c(beta_previous)

  for (i in 1:iters) {
    alpha_propose <- rnorm(1, mean=alpha_previous, sd=alpha_sd)
    beta_propose <- rnorm(1, mean=beta_previous, sd=beta_sd)
    ratio <- density_ratio(alpha_propose, alpha_previous, beta_propose,
                           beta_previous, x, y, n)
    if (ratio > 1){
      alpha_array <- c(alpha_array, alpha_propose)
      beta_array <- c(beta_array, beta_propose)
      alpha_previous = alpha_propose
      beta_previous = beta_propose
    } else {
      random_prob <- runif(n=1, min = 0, max = 1)[1]
      if (ratio > random_prob){
        alpha_array <- c(alpha_array, alpha_propose)
        beta_array <- c(beta_array, beta_propose)
        alpha_previous = alpha_propose
        beta_previous = beta_propose
      } else {
        alpha_array <- c(alpha_array, alpha_previous)
        beta_array <- c(beta_array, beta_previous)
        alpha_previous = alpha_previous
        beta_previous = beta_previous
      }
    }
  }

  afterWarmUp_alpha <- tail(alpha_array, iters - warm_up)
  afterWarmUp_beta <- tail(beta_array, iters - warm_up)

  return(list(alpha_array, beta_array, afterWarmUp_alpha, afterWarmUp_beta))
}
```

```
# Chain 1
iters <- 10000
warm_up <- 2000
alpha_initial <- 0
beta_initial <- 10
```

```r
alpha_sd <- 1
beta_sd <- 3
x <- bioassay$x
y <- bioassay$y
n <- bioassay$n
chain1 <- metropolis_bioassay(iters, warm_up, alpha_initial, beta_initial,
                                              alpha_sd, beta_sd, x, y , n)

# Chain 2
iters <- 10000
warm_up <- 2000
alpha_initial <- 1
beta_initial <- 11
alpha_sd <- 1
beta_sd <- 3
x <- bioassay$x
y <- bioassay$y
n <- bioassay$n
chain2 <- metropolis_bioassay(iters, warm_up, alpha_initial, beta_initial,
                                              alpha_sd, beta_sd, x, y , n)

# Chain 3
iters <- 10000
warm_up <- 2000
alpha_initial <- -1
beta_initial <- 9
alpha_sd <- 1
beta_sd <- 3
x <- bioassay$x
y <- bioassay$y
n <- bioassay$n
chain3 <- metropolis_bioassay(iters, warm_up, alpha_initial, beta_initial,
                                              alpha_sd, beta_sd, x, y , n)

# Chain 4
iters <- 10000
warm_up <- 2000
alpha_initial <- 2
beta_initial <- 12
alpha_sd <- 1
beta_sd <- 3
x <- bioassay$x
y <- bioassay$y
n <- bioassay$n
chain4 <- metropolis_bioassay(iters, warm_up, alpha_initial, beta_initial,
                                              alpha_sd, beta_sd, x, y , n)

# mark_my_assignment()
```

Exercise 2

Include in the report the following:

a)

Describe in your own words in one paragraph the basic idea of the Metropolis algorithm (see BDA3 Section 11.2, and lecture video 5.1).

The Metropolis algorithm is a Monte Carlo Markov chain method that produces sequential random samples from the PDF of the distribution, whose direct sampling is difficult. In other words, this distribution has the inverse PDF function that is hard to compute or does not exist in analytical form. For the MCMC, we want to ensure the balance property, where the distribution is invariant under transformation:

$\pi(i)p(i,j) = \pi(j)p(j,i)$.

If π satisfies this then π is a stationary distribution of the Markov chain.This intuition of detailed balance is that since the transfer of probability mass at each transition is the same from state i to state j as it is from state j to state i, after each transition of the chain, we remain at the stationary distribution.
The Metropolis algorithm satisfies this condition by proposing an arbitrary jump with probability $q(i,j)$, and then obtain $p(i,j)$ by only accepting the jump with probability $\alpha(i,j)$. When a jump is rejected, the state remains $j = i$.

The formula of α is: $\alpha(i,j) = min(1, \frac{\pi(j)\alpha(j,i)}{\pi(i)\alpha(i,j)})$

This is used in the Metropolis-Hastings algorithm. However, if the jump probability is symmetric then this can be simplified to

$\alpha(i,j) = min(1, \frac{\pi(j)}{\pi(i)})$,

which is used in Metropolis algorithm. If the Markov chain is ergodic (all states are irreducible), then at some point the chain will reach the stationary distribution and we are able to take samples from the target distribution. This is the intuition behind the Metropolis algorithm.

b)

The proposal distribution (related to jumping rule) you used. Describe briefly in words how you chose the final proposal distribution you used for the reported results.

The proposal distribution I used is: $\alpha^* \sim N(\alpha_{t-1}, \sigma = 1)$ and $\beta^* \sim N(\beta_{t-1}, \sigma = 3)$
The reason I choose this proposal distribution is because I want to balance between the exploitation and exploration of MCMC, with the knowledge of the prior Gaussian distribution. I hope that the Markov chain converges and the sampling distribution becomes stationarized. The jump step of 3 for β ensures that far-away initial points can eventually converge instead of always being rejected.

c)

The initial points of your Metropolis chains (or the explicit mechanism for generating them).

In this exercise, I use four different Metropolis chains. The initial points of my Metropolis chains are: $\begin{bmatrix}\alpha_1\\ \beta_1\end{bmatrix} = \begin{bmatrix}0\\ 10\end{bmatrix}$, $\begin{bmatrix}\alpha_2\\ \beta_2\end{bmatrix} = \begin{bmatrix}1\\ 11\end{bmatrix}$, $\begin{bmatrix}\alpha_3\\ \beta_3\end{bmatrix} = \begin{bmatrix}-1\\ 9\end{bmatrix}$, $\begin{bmatrix}\alpha_4\\ \beta_4\end{bmatrix} = \begin{bmatrix}2\\ 12\end{bmatrix}$. I choose initial points of the chains near the Gaussian prior mean because at this moment there is no knowledge about the sample means. These three points are also quite different from each other, so that the Metropolis chain is proven to converge no matter which initial points are chosen.

d)

Report the chain length or the number of iterations for each chain. Run the simulations long enough for approximate convergence (see BDA Section 11.4, and lecture 5.2).

The chain length or number of iterations for each chain is 10000. I believe 10000 iterations is highly likely to guarantee convergence by the algorithm.

e)

Report the warm-up length (see BDA Section 11.4, and lecture 5.2).

The warm-up length is 2000, which is 1/5 of the number of iterations. This is a recommended ratio to eliminate randomness from the beginning.

f)

The number of Metropolis chains used. It is important that multiple Metropolis chains are run for evaluating convergence (see BDA Section 11.4, and lecture 5.2).

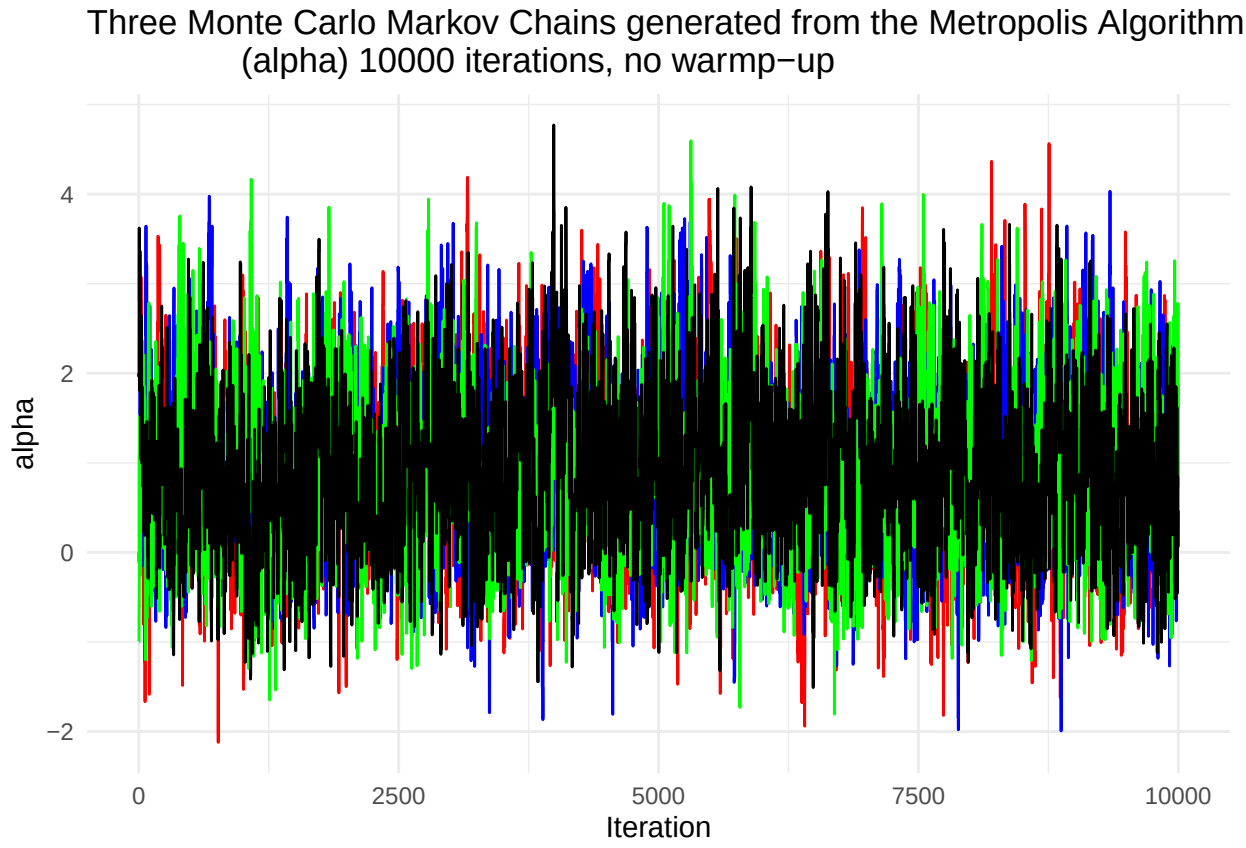
The number of Metropolis chains used in this report are four chains. This is the recommended default number of chains

g)

Plot all chains for α in a single line-plot. Overlapping the chains in this way helps in visually assessing whether chains have converged or not.

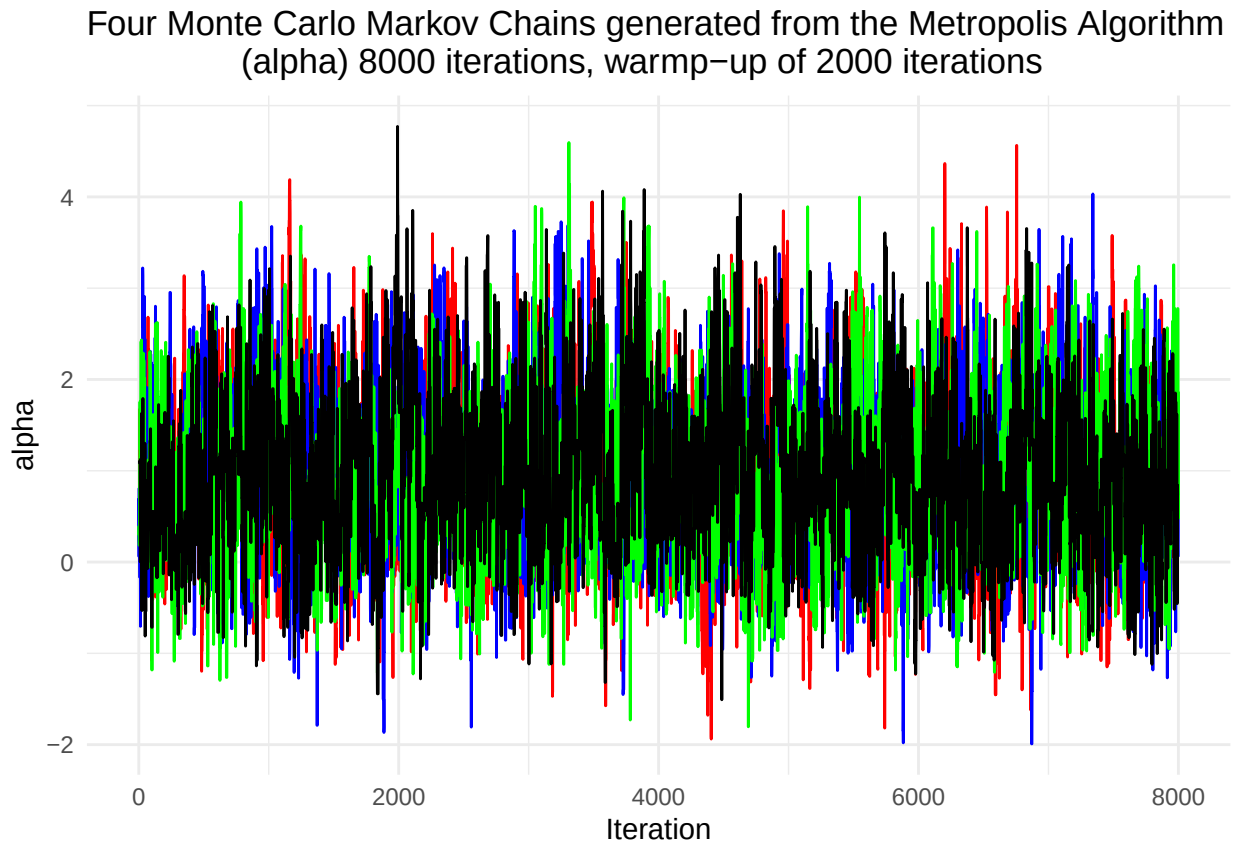
```
library(ggplot2)
alpha_array1 <- unlist(chain1[1])
alpha_array2 <- unlist(chain2[1])
alpha_array3 <- unlist(chain3[1])
alpha_array4 <- unlist(chain4[1])
```

```
indices <- 1:length(alpha_array1)

data <- data.frame(indices, alpha_array1, alpha_array2,
                           alpha_array3, alpha_array4)

ggplot(data, aes(x=indices)) +
  ggtitle(paste("Three Monte Carlo Markov Chains generated from the Metropolis Algorithm
                (alpha)",iters,"iterations, no warmp-up")) +
  xlab("Iteration") +
  ylab("alpha") +
  geom_line(aes(y = alpha_array1), color = "red") +
  geom_line(aes(y = alpha_array2), color = "blue") +
  geom_line(aes(y = alpha_array3), color = "green") +
  geom_line(aes(y = alpha_array4), color = "black")
```




```
afterWarmUp_alpha1 <- unlist(chain1[3])
afterWarmUp_alpha2 <- unlist(chain2[3])
afterWarmUp_alpha3 <- unlist(chain3[3])
afterWarmUp_alpha4 <- unlist(chain4[3])

indices <- 1:length(afterWarmUp_alpha1)

data <- data.frame(indices, afterWarmUp_alpha1, afterWarmUp_alpha2,
                           afterWarmUp_alpha3, afterWarmUp_alpha4)
```

```
ggplot(data, aes(x=indices)) +
  ggtitle(paste("Four Monte Carlo Markov Chains generated from the Metropolis Algorithm
                (alpha)", iters - warm_up,"iterations, warmp-up of",warm_up,"iterations")) +
  xlab("Iteration") +
  ylab("alpha") +
  geom_line(aes(y = afterWarmUp_alpha1), color = "red") +
  geom_line(aes(y = afterWarmUp_alpha2), color = "blue") +
  geom_line(aes(y = afterWarmUp_alpha3), color = "green") +
  geom_line(aes(y = afterWarmUp_alpha4), color = "black")
```

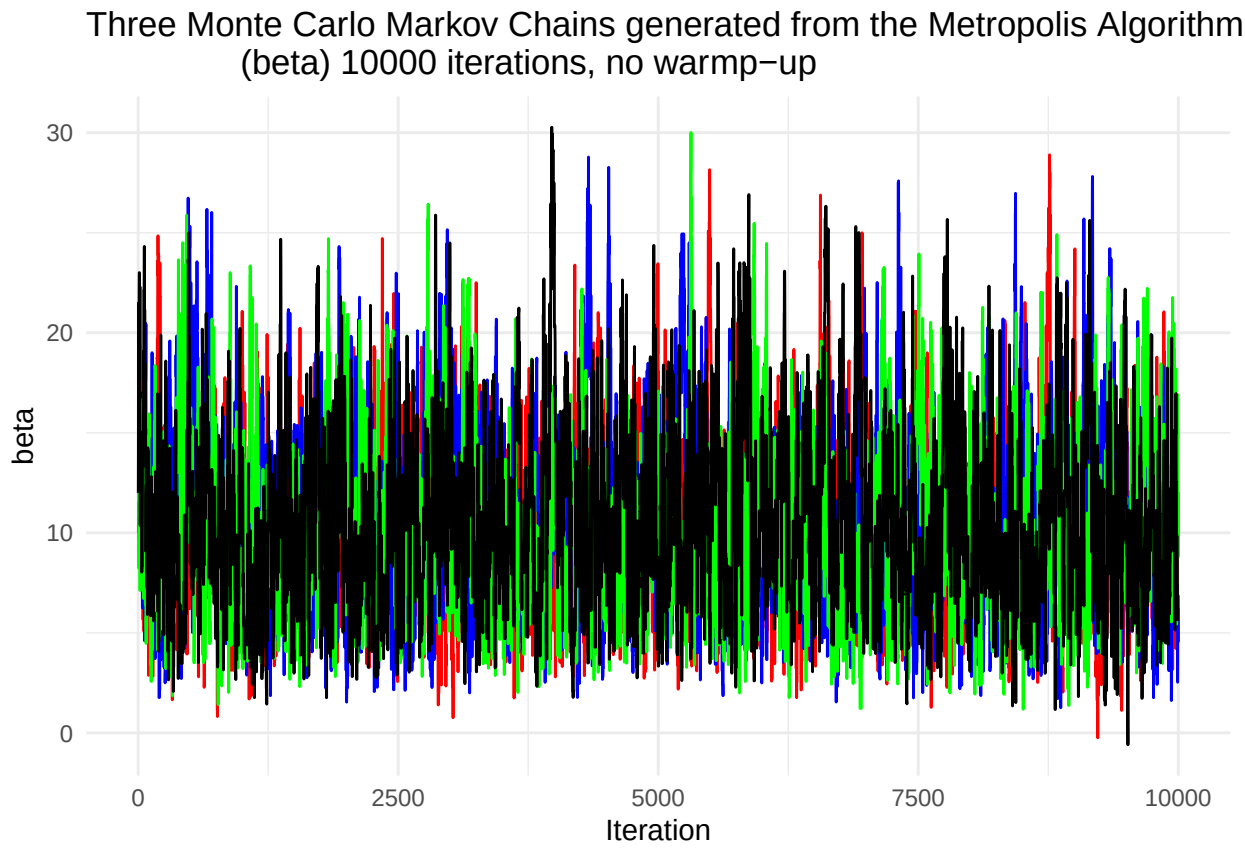


h)

Do the same for β.

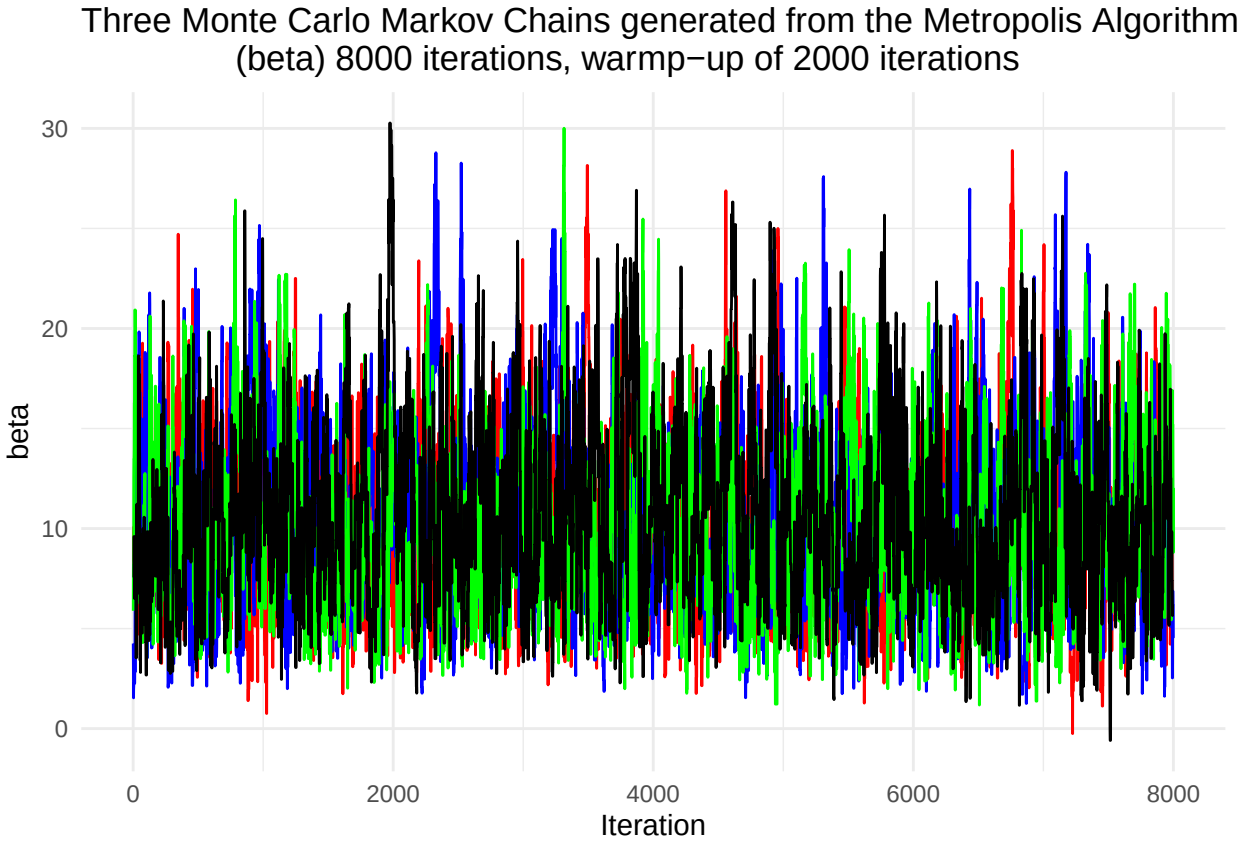
```
library(ggplot2)
beta_array1 <- unlist(chain1[2])
beta_array2 <- unlist(chain2[2])
beta_array3 <- unlist(chain3[2])
beta_array4 <- unlist(chain4[2])

indices <- 1:length(beta_array1)

data <- data.frame(indices, beta_array1, beta_array2, beta_array3, beta_array4)
```

```
ggplot(data, aes(x=indices)) +
  ggtitle(paste("Three Monte Carlo Markov Chains generated from the Metropolis Algorithm
                (beta)",iters,"iterations, no warmp-up")) +
  xlab("Iteration") +
  ylab("beta") +
  geom_line(aes(y = beta_array1), color = "red") +
  geom_line(aes(y = beta_array2), color = "blue") +
  geom_line(aes(y = beta_array3), color = "green") +
  geom_line(aes(y = beta_array4), color = "black")
```




```
afterWarmUp_beta1 <- unlist(chain1[4])
afterWarmUp_beta2 <- unlist(chain2[4])
afterWarmUp_beta3 <- unlist(chain3[4])
afterWarmUp_beta4 <- unlist(chain4[4])

indices <- 1:length(afterWarmUp_beta1)

data <- data.frame(indices, afterWarmUp_beta1, afterWarmUp_beta2,
                           afterWarmUp_beta3, afterWarmUp_beta4)

ggplot(data, aes(x=indices)) +
  ggtitle(paste("Three Monte Carlo Markov Chains generated from the Metropolis Algorithm
                (beta)", iters - warm_up,"iterations, warmp-up of",warm_up,"iterations")) +
  xlab("Iteration") +
  ylab("beta") +
```

```
  geom_line(aes(y = afterWarmUp_beta1), color = "red") +
  geom_line(aes(y = afterWarmUp_beta2), color = "blue") +
  geom_line(aes(y = afterWarmUp_beta3), color = "green") +
  geom_line(aes(y = afterWarmUp_beta4), color = "black")
```



Exercise 3

In complex scenarios, visual assessment is not sufficient and $\hat{R}$ is a more robust indicator of convergence of the Markov chains. Use $\hat{R}$ for convergence analysis. You can either use Eq. (11.4) in BDA3 or the more recent version described in a recent article. You should specify which $\hat{R}$ you used. In R the best choice is to use function rhat_basic() from the package posterior. Remember to remove the warm-up samples before computing $\hat{R}$. Report the $\hat{R}$ values for α and β separately. Report the values for the proposal distribution you finally used.

a)

Describe briefly in your own words the basic idea of $\hat{R}$ and how to to interpret the obtained $\hat{R}$ values.

The identity $\hat{R}$ is convergence diagnostic that compares the between- and within-chain estimates for model parameters and other univariate quantities of interest. If the chains have not mixed well (in other words, the between- and within-chain estimates do not agree with each other), $\hat{R}$ is larger than 1.

It is recommended that at least four chains are run by default and only using the sample if $\hat{R}$ is less than 1.05.

b)

Tell whether you obtained good $\hat{R}$ with first try, or whether you needed to run more iterations or how did you modify the proposal distribution.

```
sims <- matrix(c(afterWarmUp_alpha1, afterWarmUp_alpha2, afterWarmUp_alpha3, afterWarmUp_alpha4), ncol =
print("The Rhat value for alpha after warm-up is: ")
```

```
## [1] "The Rhat value for alpha after warm-up is: "
```

```
rhat_basic(x=sims, split=TRUE)
```

```
## [1] 1.001883
```

```
sims <- matrix(c(afterWarmUp_beta1, afterWarmUp_beta2, afterWarmUp_beta3, afterWarmUp_beta4), ncol = 4)
print("The Rhat value for beta after warm-up is: ")
```

```
## [1] "The Rhat value for beta after warm-up is: "
```

```
rhat_basic(x=sims, split=TRUE)
```

```
## [1] 1.003581
```

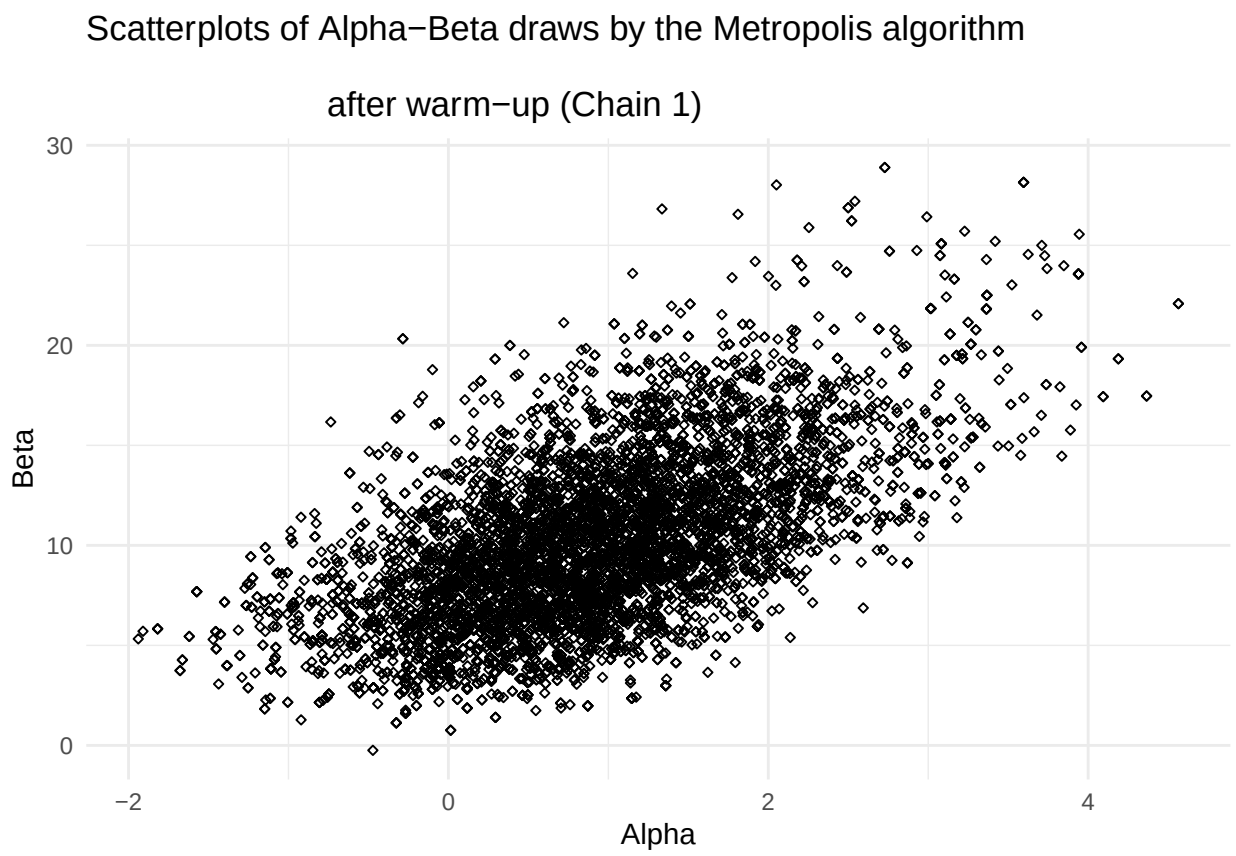
Since $\hat{R}_\alpha < 1.05$ and $\hat{R}_\beta < 1.05$, it indicates that all the chains have successfully converged for both α and β. I have obtained a good $\hat{R}$ in the first try and I do not need to run more iterations. The final proposal distribution I used is:
$\alpha^* \sim N(\alpha_{t-1}, \sigma = 1)$ and $\beta^* \sim N(\beta_{t-1}, \sigma = 3)$

Exercise 4

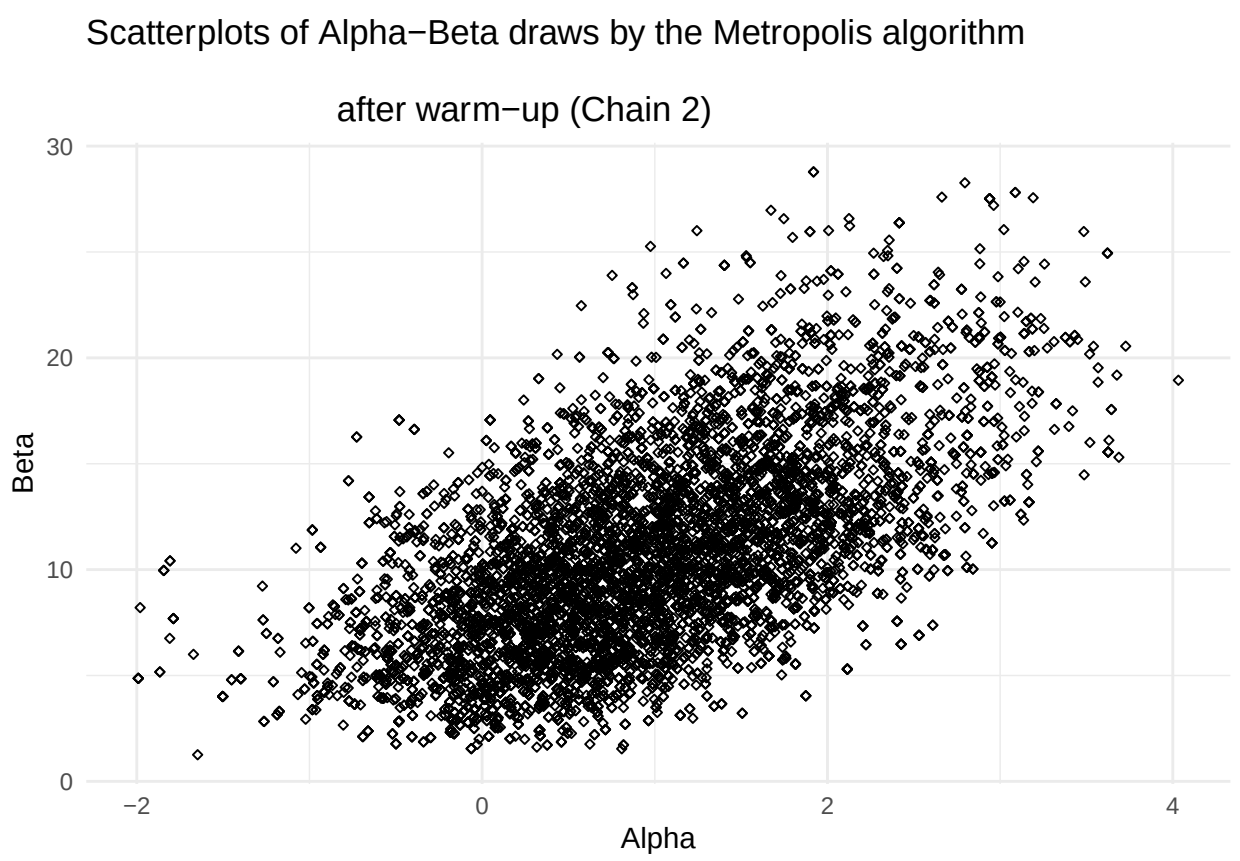
Plot the draws for α and β (scatter plot) and include this plot in your report. You can compare the results to Figure 3.3b in BDA3 to verify that your code gives sensible results. Notice though that the results in Figure 3.3b are generated from posterior with a uniform prior, so even when if your algorithm works perfectly, the results will look slightly different (although fairly similar).

From the figures plotted below, we can observe that they closely resemble the scatter plot in Figure 3.3b in BDA3, suggesting that the Metropolis algorithm successfully converges but with non-uniform prior as compared to the uniform prior case in BDA3.

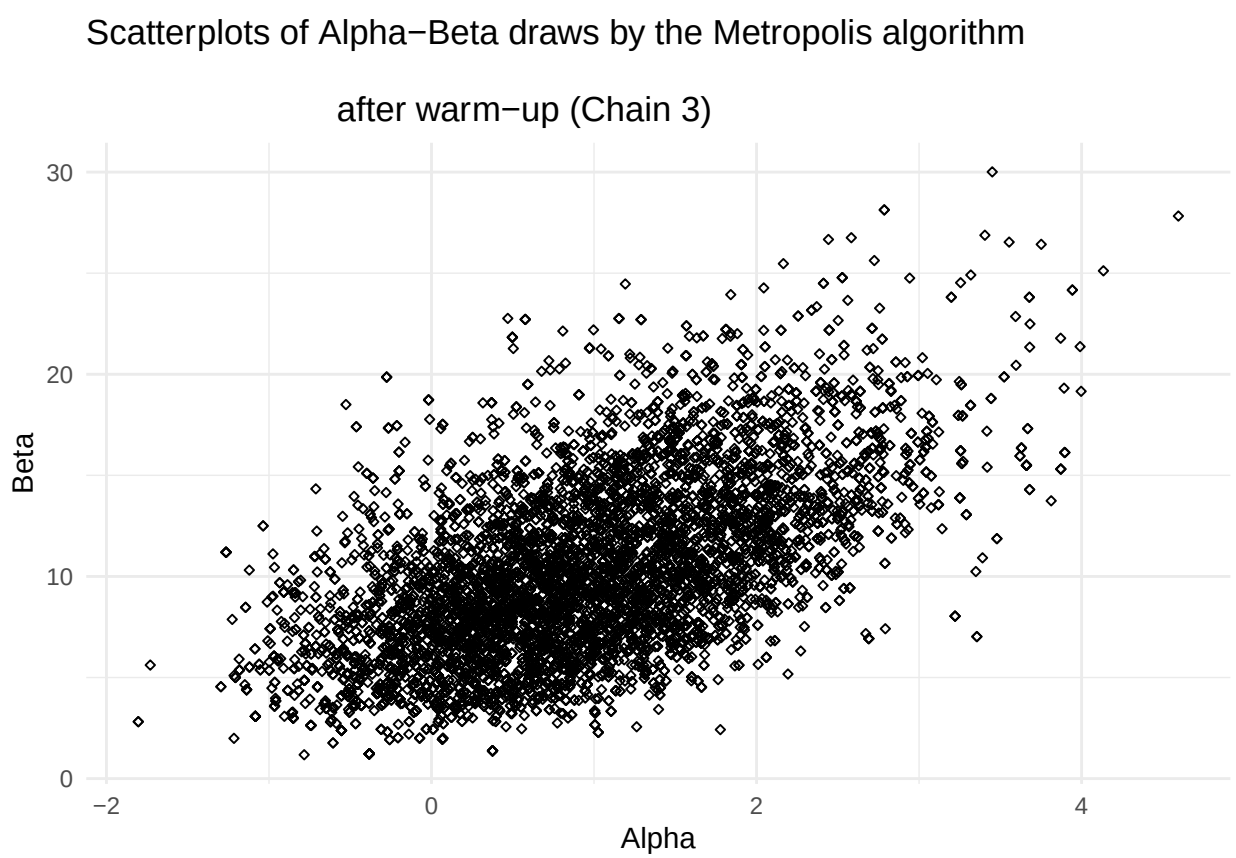
```
indices <- 1:length(afterWarmUp_alpha1)
scatterplots <- data.frame(indices, afterWarmUp_alpha1, afterWarmUp_alpha2,
                           afterWarmUp_alpha3, afterWarmUp_alpha4, afterWarmUp_beta1,
                           afterWarmUp_beta2, afterWarmUp_beta3, afterWarmUp_beta4)

ggplot(scatterplots, aes(x = afterWarmUp_alpha1, y=afterWarmUp_beta1)) +
  geom_point(size=1, shape=23) +
  ggtitle("Scatterplots of Alpha-Beta draws by the Metropolis algorithm
          \n                            after warm-up (Chain 1)") +
  xlab("Alpha") +
  ylab("Beta")
```



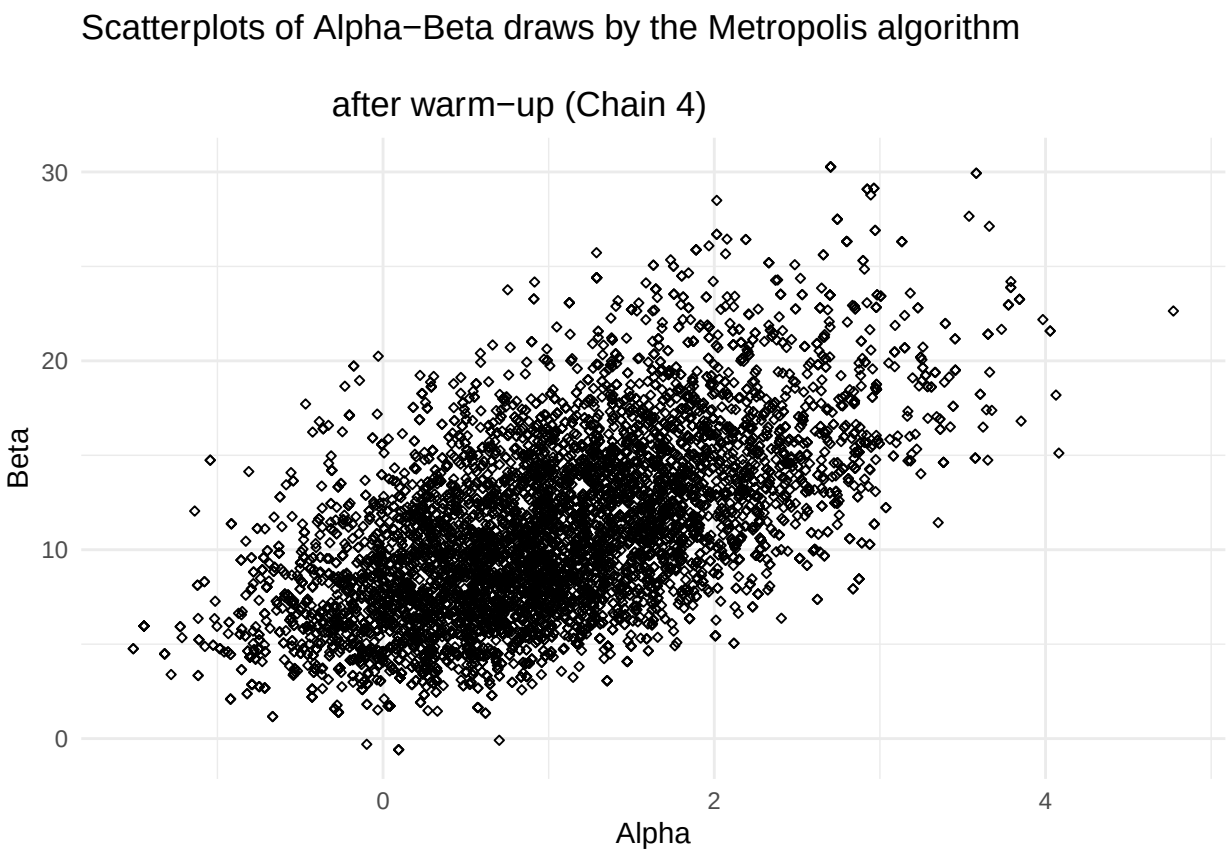

```
ggplot(scatterplots, aes(x = afterWarmUp_alpha2, y=afterWarmUp_beta2)) +
  geom_point(size=1, shape=23) +
  ggtitle("Scatterplots of Alpha-Beta draws by the Metropolis algorithm
          \n                            after warm-up (Chain 2)") +
  xlab("Alpha") +
  ylab("Beta")
```




```
ggplot(scatterplots, aes(x = afterWarmUp_alpha3, y=afterWarmUp_beta3)) +
  geom_point(size=1, shape=23) +
  ggtitle("Scatterplots of Alpha-Beta draws by the Metropolis algorithm
          \n                              after warm-up (Chain 3)") +
  xlab("Alpha") +
  ylab("Beta")
```




```
ggplot(scatterplots, aes(x = afterWarmUp_alpha4, y=afterWarmUp_beta4)) +
  geom_point(size=1, shape=23) +
  ggtitle("Scatterplots of Alpha-Beta draws by the Metropolis algorithm
          \n                              after warm-up (Chain 4)") +
  xlab("Alpha") +
  ylab("Beta")
```


Scatterplots of Alpha−Beta draws by the Metropolis algorithm
after warm−up (Chain 4)
Beta
Alpha
30
20
10
0
0
2
4