BSDA: Assignment 5

Anonymous student

Contents

General Information to include 1

Generalized linear model: Bioassay with Metropolis 1

General Information to include

- **Time used for reading and self-study exercises**: ~ 8 hours.
- **Time used for the assignment**: ~11 hours
- **Good with assignment**: Implement Metropolis algorithm ourselves, it made everything super cleaer on how the basic algorithm works.
- **Things to improve in the assignment**: I felt like there was not enough discussion in class about topics like how to formally pick the starting points, rank normalized split-$\widehat{R}$. I also feel like I wouldn't be able to come up with likelihoods, and priors on my own for my own specific projects.

Generalized linear model: Bioassay with Metropolis

1. Implement the Metropolis algorithm as an R function for the bioassay data. Use the Gaussian prior

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} \sim \mathrm{N}\left(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0\right), \qquad \text{where} \quad \boldsymbol{\mu}_0 = \begin{bmatrix}0\\ 10\end{bmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_0 = \begin{bmatrix}2^2 & 12\\ 12 & 10^2\end{bmatrix}.$$

a) Start by implementing a function called `density_ratio` to compute the density ratio function, r in Eq. (11.1) in BDA3.

```r
library(bsda)
library(tidyverse)
library(gridExtra)

data("bioassay")

density_ratio <- function(alpha_propose, alpha_previous,
                          beta_propose, beta_previous,
                          x, y, n) {

  log_prior_t <- log(dmvnorm(c(alpha_propose, beta_propose), c(0,10),
                       matrix(c(4,12,12,100), nrow = 2,
                              ncol = 2,byrow = TRUE)))

  log_likelihood_t <- bioassaylp(alpha_propose, beta_propose, x, y, n)
  log_posterior_t <- log_likelihood_t + log_prior_t

  log_prior_t_1 <- log(dmvnorm(c(alpha_previous, beta_previous), c(0,10),
                           matrix(c(4,12,12,100), nrow = 2,
                                  ncol = 2,byrow = TRUE)))

  log_likelihood_t_1 <- bioassaylp(alpha_previous, beta_previous, x, y, n)

  log_posterior_t_1 <- log_likelihood_t_1 + log_prior_t_1

  return(exp(log_posterior_t-log_posterior_t_1))
}
```

Acceptance ratio:

$$r = \frac{p(\theta^*|y)}{p(\theta_{t-1}|y)}$$

b) Now implement a function called `Metropolis_bioassay()` which implements the Metropolis algorithm using the `density_ratio()`.

```r
#function to implement Metropolis Algorithm
Metropolis_bioassay <- function(alpha_0, beta_0, sigma_a, sigma_b, S) {
  list_alpha <- list(alpha_0)
  list_beta <- list(beta_0)
  accepted <- 0

  for (i in seq(1, S)){

    proposal_a <- rnorm(1, alpha_0, sigma_a)
    proposal_b <- rnorm(1, beta_0, sigma_b)

    acceptance_ratio <- density_ratio(alpha_propose = proposal_a,
                                      alpha_previous = alpha_0,
                                      beta_propose = proposal_b,
                                      beta_previous = beta_0,
                                      x = bioassay$x,
                                      y = bioassay$y,
                                      n = bioassay$n)

    if (runif(1, 0, 1) < min(acceptance_ratio, 1)) {
      alpha_0 <- proposal_a
      beta_0 <- proposal_b
      accepted <- accepted + 1
    }

    list_alpha[[i+1]] <- alpha_0
    list_beta[[i+1]] <- beta_0
  }
  print(1- accepted/S)
  return(tibble(alpha = as.numeric(list_alpha), beta = as.numeric(list_beta)))
}
```

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).

The Metropolis algorithm is a MCMC method used to generate samples from a desired probability distribution. It combines random walks with an acceptance/rejection rule. Starting from an initial point, it takes a series of random steps in the parameter space. At each step, a new value is proposed, and an acceptance ratio (r) is calculated based on the probability that this value comes from the target distribution. If the acceptance ratio is 1 or higher, the new value is accepted as a sample; otherwise, it is accepted or rejected by generating a uniform random number and comparing it with r. Repeating this process accumulates draws that converge to the desired probability distribution.

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.

Through trial and error, I experimented with different values for the jump scale (σ) and assessed the rejection rate obtained for each of them. As discussed in class, a general rule for the rejection rate falls between 60% to 90%. I decided to aim for a value that approaches the mean of 75%. This choice strikes a balance between a very low rejection rate, where the chain moves slowly due to small steps, and a rejection rate that is too high, where, again, the chain moves slowly because even though the steps are longer, many are rejected. Therefore, I opted to use proposals $\alpha^* \sim N(\alpha_{t-1}, \sigma = 2)$ and $\beta^* \sim N(\beta_{t-1}, \sigma = 7)$. In my code, I print the rejection rate for each chain, which tends to be close to 75%.

```r
# run the function for 8 different chains
starting_points <- tibble( alpha = list(-2, -1, 1, 3, 4, 6, 2, 0),
                           beta = list(0, 20, 30, 40, 10, 25, 2, -1))

results <- list()
set.seed(1234)
for (chain in 1:nrow(starting_points)){
  chain_results <- Metropolis_bioassay(as.numeric(starting_points$alpha[chain]),
                                       as.numeric(starting_points$beta[chain]),
                                       2, 7, 10000)
  chain_results$chain <- as.character(chain)
  results[[chain]] <- chain_results
}
```

```
## [1] 0.7543
## [1] 0.7517
## [1] 0.749
## [1] 0.7532
## [1] 0.7504
## [1] 0.7473
## [1] 0.747
## [1] 0.7513
```

```r
combined_chains <- do.call(rbind, results)
```

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

I attempted to initiate my chains from different starting points that were adequately overdispersed. This approach simplifies convergence diagnostics. To determine these initial points, I ran 10,000 iterations of the Metropolis algorithm using the proposals $\alpha^* \sim N(\alpha_{t-1}, \sigma = 2)$ and $\beta^* \sim N(\beta_{t-1}, \sigma = 7)$ starting in (0, 0). Subsequently, I plotted the samples on a scatter plot, excluding 5,000 warm-up steps. This visual analysis helped me choose starting points that were sufficiently overdispersed. In the plot the starting points I chose are in red.

Table 1: Intial points for each chain

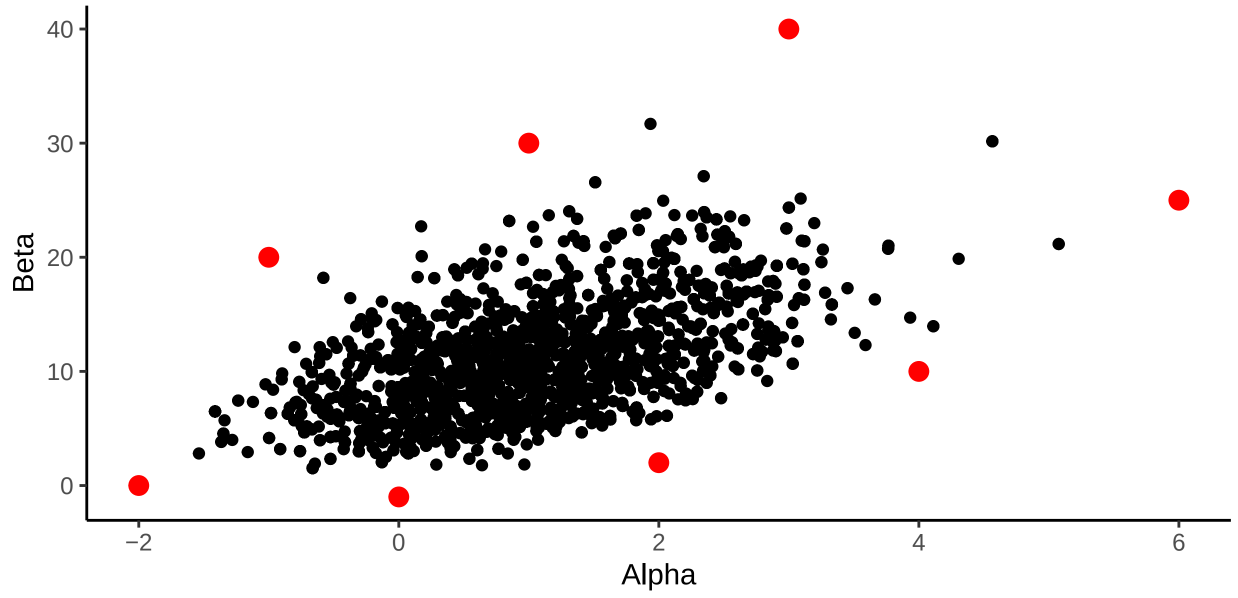
Alpha	Beta
-2	0
-1	20
1	30
3	40
4	10
6	25
2	2
0	-1

```
knitr::kable(starting_points, format = "latex", booktabs = TRUE, align = "c",
             col.names = c("Alpha",
                           "Beta"), caption = "Intial points for each chain")
```

```
first_observation <- combined_chains %>%
  group_by(chain) %>% filter(row_number() == 1)

set.seed(1234)
test <- Metropolis_bioassay(0, 0, 2, 7, 10000)
```

```
## [1] 0.7535
```

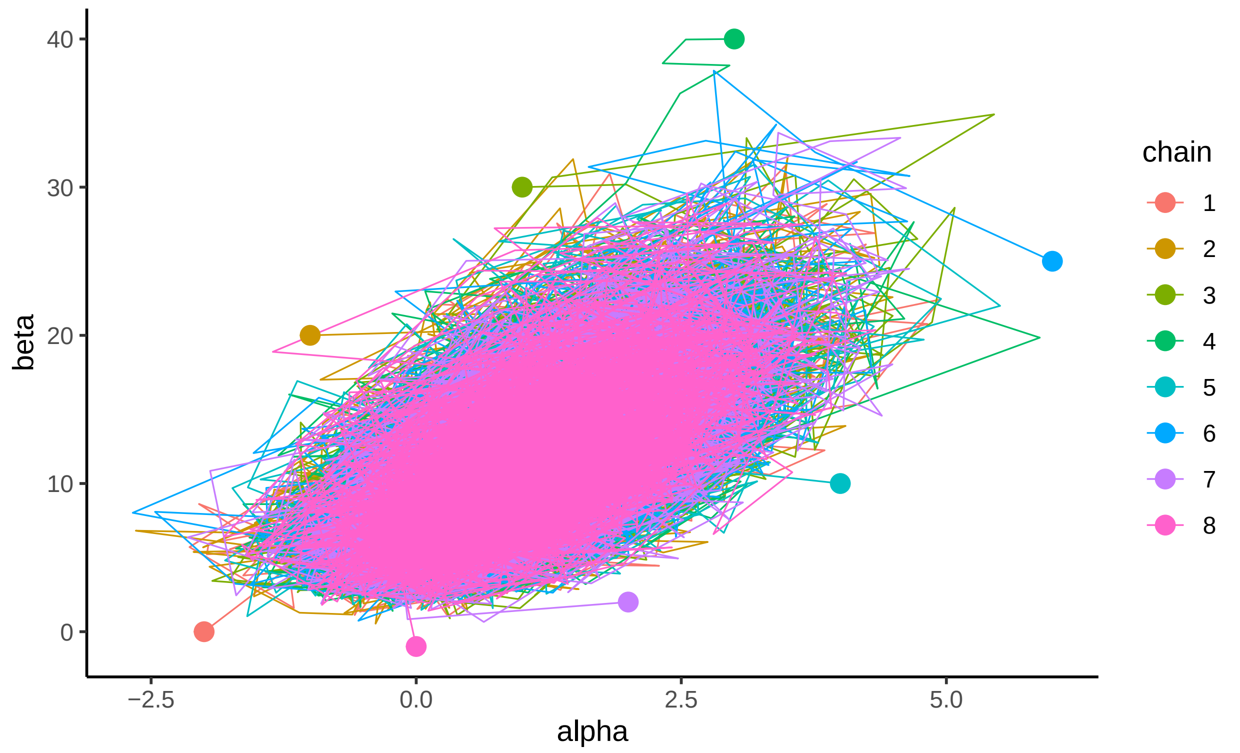
```
test_final <- tail(test, n = 5000)

ggplot(test_final, aes(x = alpha, y = beta)) +
  geom_point() +
  labs(x = "Alpha", y = "Beta") +
  geom_point(data = first_observation, color = 'red', size = 3) +
  theme_classic()
```



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).

I used 10,000 iterations for each chain.

```r
ggplot(combined_chains, aes(x = alpha, y = beta)) +
  geom_path(aes(color = chain), linewidth= 0.3) +
  geom_point(data = first_observation, aes(color = chain), size = 3) +
  theme_classic()
```



e) Report the warm-up length (see BDA Section 11.4).

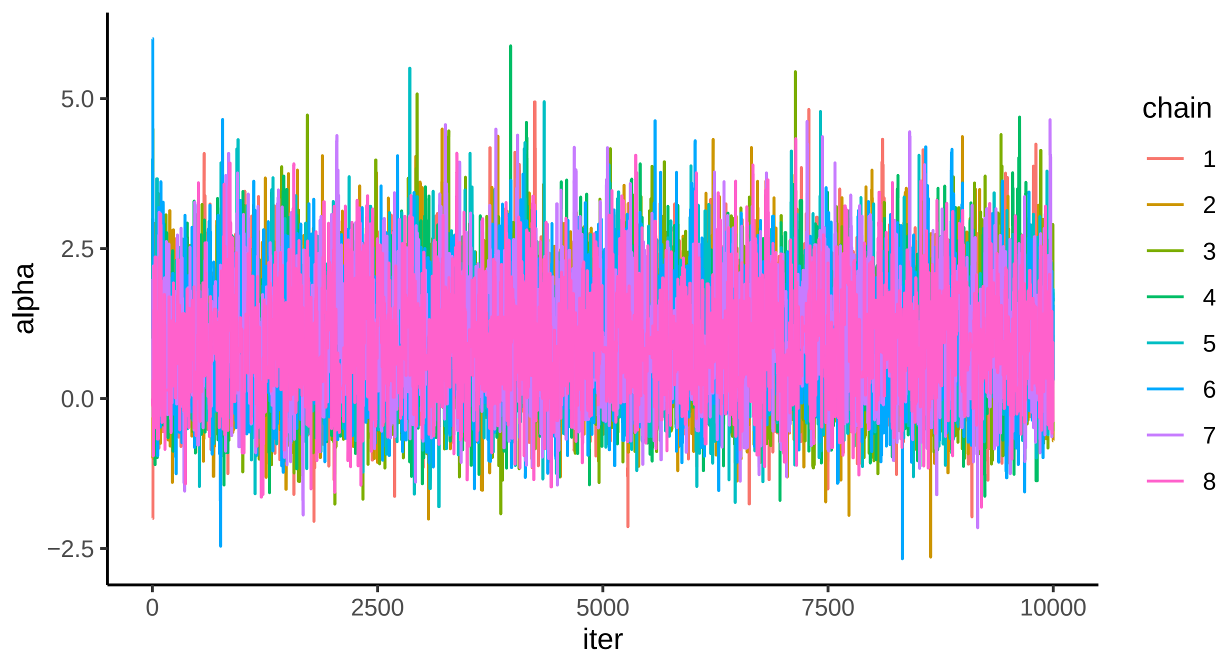
Different warm-up fractions can be appropriate depending on the context, but for this case I decided to use the conservative choice of discarding the first half, proposed in BDA3. Therefore, since I ran 10,000 iterations per chain, the first 5,000 belong to the warm-up period.

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).

I ran 8 chains, with overdispersed starting points.

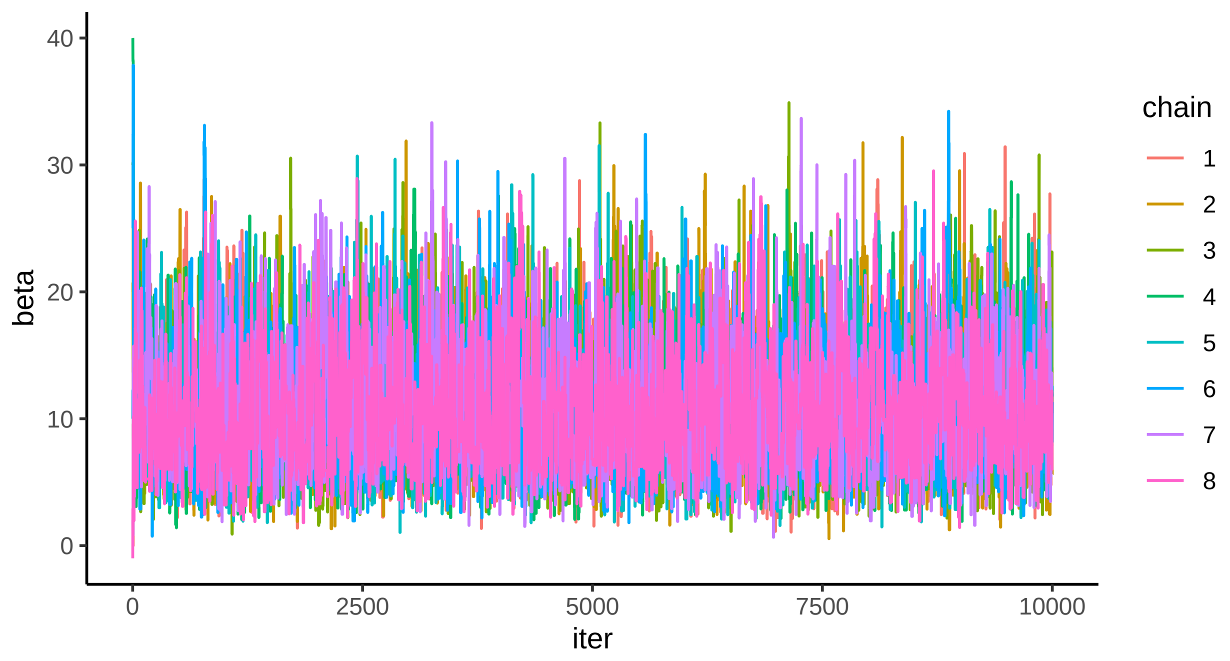
g) Plot all chains for α in a single line-plot. Overlapping the chains in this way helps in visually assessing whether chains have converged or not.

```r
combined_chains <- combined_chains %>% group_by(chain) %>%
  mutate(iter = row_number()) %>% ungroup()

ggplot(combined_chains, aes(x = iter, y = alpha)) +
  geom_line(aes(color = chain))  + theme_classic()
```



h) Do the same for β.

```r
ggplot(combined_chains, aes(x = iter, y = beta)) +
  geom_line(aes(color = chain)) + theme_classic()
```



Figures show the 8 parallel chain simulations for both α and β. It is clear that the sequences have mixed, the variance within each sequence seem pretty close to the variance between sequence in both plots, therefore it seems like both sequences have converged. This is just a visual assessment, we should also check the $\widehat{R}$.

3. In complex scenarios, visual assessment is not sufficient and $\widehat{R}$ is a more robust indicator of convergence of the Markov chains. Use $\widehat{R}$ for convergence analysis. You can either use Eq. (11.4) in BDA3 or the more recent version described here. You should specify which $\widehat{R}$ you used. In R the best choice is to use function `Rhat` from package `rstan`. Remember to remove the warm-up samples before computing $\widehat{R}$. Report the $\widehat{R}$ 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 $\widehat{R}$ and how to to interpret the obtained $\widehat{R}$ values.

The basic idea of $\widehat{R}$ is that for each estimand θ we compute the between and within variance, and we are trying to assess if our chains have converged by checking if the variation between and within in our simulated sequences is roughly equal. If we have $\widehat{R} = 1$ this means that the between and within variance is the same and that our chains are mixed and have converged.

In this case I used the function `Rhat` from `rstan` package which is based in the more recent version[1]. According to the documentation, the function reports $\widehat{R}$ which is the maximum of rank normalized split-$\widehat{R}$ and rank normalized folded-split-$\widehat{R}$. Original $\widehat{R}$ main drawback is that it requires that the target distribution has finite mean and variance, rank normalization fixes this.

The proposal distribution I finally used is $\alpha^* \sim N(\alpha_{t-1}, \sigma = 2)$ and $\beta^* \sim N(\beta_{t-1}, \sigma = 7)$, and I removed 5,000 warm-up steps.

```r
warm_up <- 5000
combined_chains_final <- combined_chains %>%
  filter(iter > warm_up)

seq_alpha <- combined_chains_final %>%
  select(-c(beta)) %>%
  pivot_wider(names_from = chain, values_from = alpha) %>%
  select(-c(iter))

seq_beta <- combined_chains_final %>%
  select(-c(alpha)) %>%
  pivot_wider(names_from = chain, values_from = beta) %>%
  select(-c(iter))

rhat_a <- rstan::Rhat(as.matrix(seq_alpha))
rhat_a
```

```
## [1] 1.001056
```

```r
rhat_b <- rstan::Rhat(as.matrix(seq_beta))
rhat_b
```

```
## [1] 1.002071
```

As mentioned above if we have $\widehat{R} = 1$ this means that the between and within variance is the same and that our chains are mixed and have converged. in this case we obtained values for α and β pretty close to 1 (1.001, and 1.002) meaning that the between and within chain variance are about equal. The usual convention is that if $\widehat{R}$ is big (e.g., $\widehat{R} > 1.01$) we should keep sampling, in this case both values are less than 1.01 and greater than 1, indicating that the chains have converged and mixed.

[1] Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2019). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC. arXiv preprint arXiv:1903.08008.

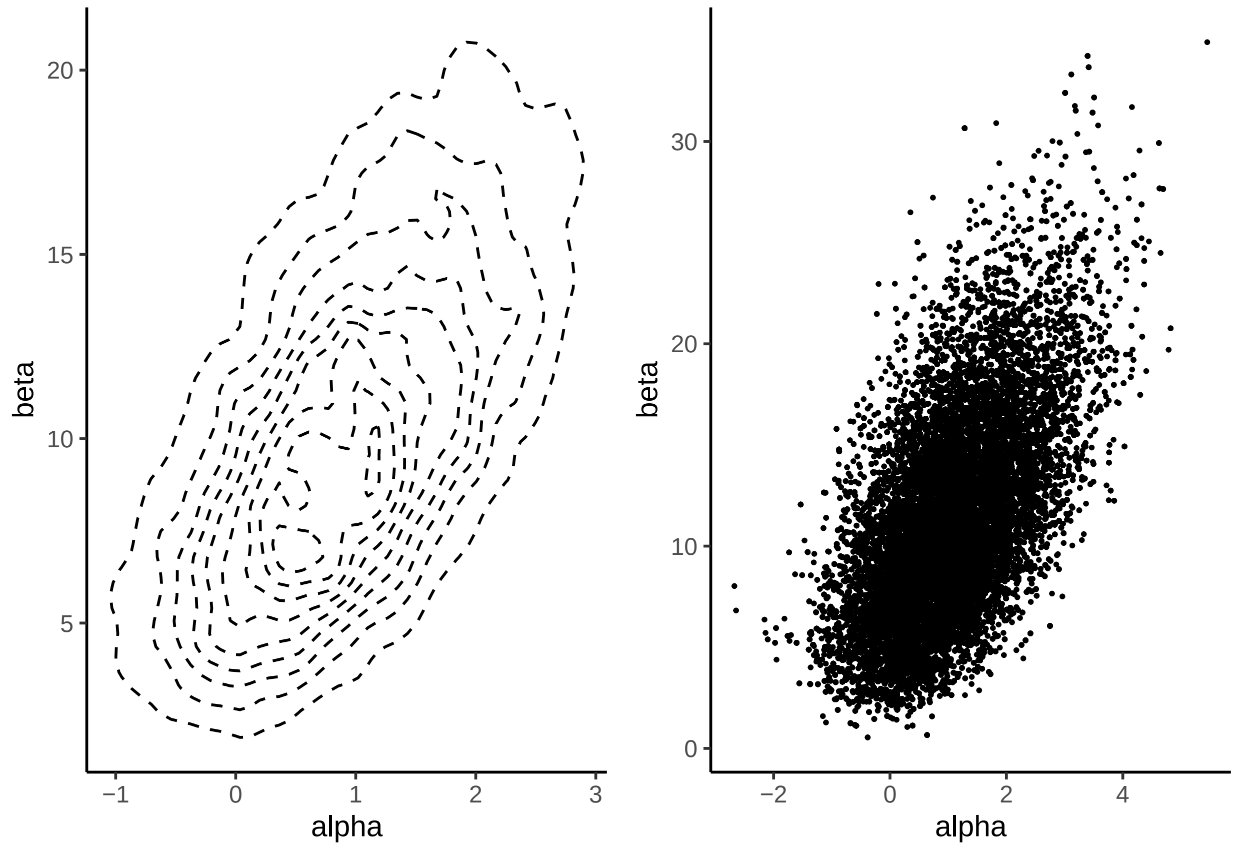
b) Tell whether you obtained good $\widehat{R}$ with first try, or whether you needed to run more iterations or how did you modify the proposal distribution.

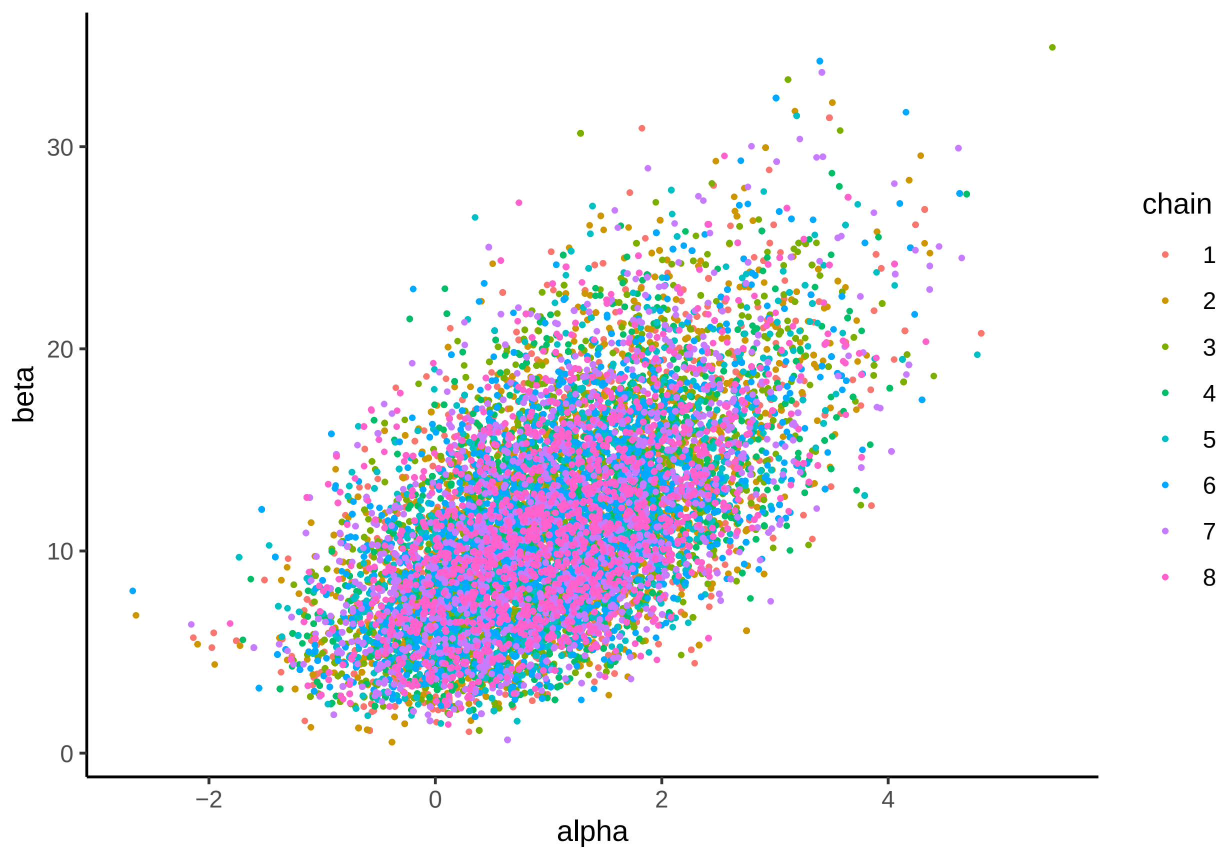
At the beginning I was just running 1,000 iterations per chain and the $\widehat{R}$ values I was getting for α and β were slightly larger than 1.01, then after increasing my iterations to 10,000 my $\widehat{R}$ shrinked.

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).

The first figure tries to replicate Figure~3.3 in BDA3 and the second figure only shows the replication of Figure~3.3b using my draws for α and β.

```
p_1 <- ggplot(combined_chains_final, aes(x = alpha, y = beta)) +
  geom_point(size= 0.35) +
  theme_classic()

p_2 <- ggplot(combined_chains_final, aes(x = alpha, y = beta)) +
  geom_density_2d(bins = 10, linetype = 2, color = 'black', linewidth= 0.5) +
  theme_classic()

grid.arrange(p_2, p_1, ncol = 2)
```




beta
30
20
10
0
−2
0
2
4
alpha
chain
1
2
3
4
5
6
7
8