BDA - Assignment 6

Anonymous

Contents

Exercise 1. **1**
1. 1
2. 8
3. 9
4. 10

Exercise 1.

Generalized linear model: Bioassay with Stan (6 points) Replicate the computations for the bioassay example of section 3.7 (BDA3) using Stan.

1.

Write down the model for the bioassay data in Stan syntax. For instructions in reporting your implementation, you can refer to parts 2 c) - g) in Assignment 5. More information on the bioassay data can be found in Section 3.7 of the course book and in Chapter 3 reading instructions. To get access to data, use the following code:

```
library(aaltobda)
data("bioassay")
```

Use the Gaussian prior as in Assignment 4 and 5, that is

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} \sim N(\mu_0, \Sigma_0), \quad \text{where } \mu_0 = \begin{bmatrix}0\\ 10\end{bmatrix} \quad \textit{and} \quad \Sigma_0 = \begin{bmatrix}2^2 & 12\\ 12 & 10^2\end{bmatrix}.$$

Hint! You will need Stan functions multi_normal and binomial_logit for implementing the prior and observation model, respectively. In Stan code, it is easiest to declare a variable (say theta) which is a two-element vector so that the first value denotes α and latter one β. This is because the multi_normal function that you need for implementing the prior requires a vector as an input.

The bioassay stan model is:

```
"
# Bioassay model
data {
    int<lower=0> N;        // The number of groups
    int<lower=0> n;    // The number of animals used in each group. Should be constant
    real x[N];             // The amount of doses used in each group
    int<lower=0,upper=n> y[N];    // Number of deaths in each group
```

```
}
parameters {
    vector[2] alphaNbeta;  // Vector of 2 values alpha and beta
}
model {
    row_vector[2] mu = [0,10];
    matrix[2,2] sigma =[[4, 12],[12, 100]];
    vector[N] loglikelihood;  // log-likelihood from each group

    // The log prior distribution
    target += multi_normal_lpdf(alphaNbeta | mu, sigma);

    for(i in 1:N) {
        real alpha = alphaNbeta[1];
        real beta = alphaNbeta[2];
        loglikelihood[i] = binomial_logit_lpmf(y[i] | n, alpha + beta * x[i]);
    }
    target += sum(loglikelihood);
}

"
```

```
## [1] "\n# Bioassay model\ndata {\n    int<lower=0> N;       // The number of groups\n    int<lower=0>
```

Now, we load the model

```
set_cmdstan_path("/coursedata/cmdstan/")
```

```
## CmdStan path set to: /coursedata/cmdstan
```

```
file <- file.path("bioassay_model.stan")
model <- cmdstan_model(file)
model$compile(quiet = FALSE)
```

Then, we run the sampling method, with the data from the bioassay experiment

```
# Run MCMC using the 'sample' method
N = length(bioassay$n) # The number of groups
n = (bioassay$n)[1] # The number of animals used in each group. Should be constant
x = bioassay$x  # The amount of doses used in each group
y = bioassay$y # Number of deaths in each group
bioassay_data <- list(N = N, n=n, x=x, y=y)

iter_warmup <- 2000 # Number of warm up iterations
iter_sampling <- 2000 # Number of sampling iterations
chains <- 4 # Number of MCMC

fit_mcmc <- model$sample(
  data = bioassay_data,
  seed = 123,
  iter_warmup = iter_warmup,
  iter_sampling = iter_sampling,
  save_warmup = TRUE, # The warmup iterations is saved for visualization below
  chains = chains,
  parallel_chains = chains
```

```
)
```

```
## Running MCMC with 4 parallel chains...
##
## Chain 1 Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 4000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 4000 [  5%]  (Warmup)
## Chain 1 Iteration:  300 / 4000 [  7%]  (Warmup)
## Chain 1 Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 1 Iteration:  500 / 4000 [ 12%]  (Warmup)
## Chain 1 Iteration:  600 / 4000 [ 15%]  (Warmup)
## Chain 1 Iteration:  700 / 4000 [ 17%]  (Warmup)
## Chain 1 Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 1 Iteration:  900 / 4000 [ 22%]  (Warmup)
## Chain 1 Iteration: 1000 / 4000 [ 25%]  (Warmup)
## Chain 1 Iteration: 1100 / 4000 [ 27%]  (Warmup)
## Chain 1 Iteration: 1200 / 4000 [ 30%]  (Warmup)
## Chain 1 Iteration: 1300 / 4000 [ 32%]  (Warmup)
## Chain 1 Iteration: 1400 / 4000 [ 35%]  (Warmup)
## Chain 1 Iteration: 1500 / 4000 [ 37%]  (Warmup)
## Chain 1 Iteration: 1600 / 4000 [ 40%]  (Warmup)
## Chain 1 Iteration: 1700 / 4000 [ 42%]  (Warmup)
## Chain 1 Iteration: 1800 / 4000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1900 / 4000 [ 47%]  (Warmup)
## Chain 1 Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 1 Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2100 / 4000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 1 Iteration: 2300 / 4000 [ 57%]  (Sampling)
## Chain 1 Iteration: 2400 / 4000 [ 60%]  (Sampling)
## Chain 2 Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 4000 [  2%]  (Warmup)
## Chain 2 Iteration:  200 / 4000 [  5%]  (Warmup)
## Chain 2 Iteration:  300 / 4000 [  7%]  (Warmup)
## Chain 2 Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 2 Iteration:  500 / 4000 [ 12%]  (Warmup)
## Chain 2 Iteration:  600 / 4000 [ 15%]  (Warmup)
## Chain 2 Iteration:  700 / 4000 [ 17%]  (Warmup)
## Chain 2 Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 2 Iteration:  900 / 4000 [ 22%]  (Warmup)
## Chain 2 Iteration: 1000 / 4000 [ 25%]  (Warmup)
## Chain 2 Iteration: 1100 / 4000 [ 27%]  (Warmup)
## Chain 2 Iteration: 1200 / 4000 [ 30%]  (Warmup)
## Chain 2 Iteration: 1300 / 4000 [ 32%]  (Warmup)
## Chain 2 Iteration: 1400 / 4000 [ 35%]  (Warmup)
## Chain 2 Iteration: 1500 / 4000 [ 37%]  (Warmup)
## Chain 2 Iteration: 1600 / 4000 [ 40%]  (Warmup)
## Chain 2 Iteration: 1700 / 4000 [ 42%]  (Warmup)
## Chain 2 Iteration: 1800 / 4000 [ 45%]  (Warmup)
## Chain 2 Iteration: 1900 / 4000 [ 47%]  (Warmup)
## Chain 2 Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 2 Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2100 / 4000 [ 52%]  (Sampling)
## Chain 2 Iteration: 2200 / 4000 [ 55%]  (Sampling)
```

```
## Chain 2 Iteration: 2300 / 4000 [ 57%]  (Sampling)
## Chain 2 Iteration: 2400 / 4000 [ 60%]  (Sampling)
## Chain 2 Iteration: 2500 / 4000 [ 62%]  (Sampling)
## Chain 2 Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 2 Iteration: 2700 / 4000 [ 67%]  (Sampling)
## Chain 2 Iteration: 2800 / 4000 [ 70%]  (Sampling)
## Chain 2 Iteration: 2900 / 4000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 2 Iteration: 3100 / 4000 [ 77%]  (Sampling)
## Chain 2 Iteration: 3200 / 4000 [ 80%]  (Sampling)
## Chain 2 Iteration: 3300 / 4000 [ 82%]  (Sampling)
## Chain 2 Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 2 Iteration: 3500 / 4000 [ 87%]  (Sampling)
## Chain 3 Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 4000 [  2%]  (Warmup)
## Chain 3 Iteration:  200 / 4000 [  5%]  (Warmup)
## Chain 3 Iteration:  300 / 4000 [  7%]  (Warmup)
## Chain 3 Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 3 Iteration:  500 / 4000 [ 12%]  (Warmup)
## Chain 3 Iteration:  600 / 4000 [ 15%]  (Warmup)
## Chain 3 Iteration:  700 / 4000 [ 17%]  (Warmup)
## Chain 3 Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 3 Iteration:  900 / 4000 [ 22%]  (Warmup)
## Chain 3 Iteration: 1000 / 4000 [ 25%]  (Warmup)
## Chain 3 Iteration: 1100 / 4000 [ 27%]  (Warmup)
## Chain 3 Iteration: 1200 / 4000 [ 30%]  (Warmup)
## Chain 3 Iteration: 1300 / 4000 [ 32%]  (Warmup)
## Chain 3 Iteration: 1400 / 4000 [ 35%]  (Warmup)
## Chain 3 Iteration: 1500 / 4000 [ 37%]  (Warmup)
## Chain 3 Iteration: 1600 / 4000 [ 40%]  (Warmup)
## Chain 3 Iteration: 1700 / 4000 [ 42%]  (Warmup)
## Chain 3 Iteration: 1800 / 4000 [ 45%]  (Warmup)
## Chain 3 Iteration: 1900 / 4000 [ 47%]  (Warmup)
## Chain 3 Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 3 Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 3 Iteration: 2100 / 4000 [ 52%]  (Sampling)
## Chain 3 Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 3 Iteration: 2300 / 4000 [ 57%]  (Sampling)
## Chain 3 Iteration: 2400 / 4000 [ 60%]  (Sampling)
## Chain 3 Iteration: 2500 / 4000 [ 62%]  (Sampling)
## Chain 3 Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 3 Iteration: 2700 / 4000 [ 67%]  (Sampling)
## Chain 3 Iteration: 2800 / 4000 [ 70%]  (Sampling)
## Chain 3 Iteration: 2900 / 4000 [ 72%]  (Sampling)
## Chain 3 Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 4 Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 4000 [  2%]  (Warmup)
## Chain 4 Iteration:  200 / 4000 [  5%]  (Warmup)
## Chain 4 Iteration:  300 / 4000 [  7%]  (Warmup)
## Chain 4 Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 4 Iteration:  500 / 4000 [ 12%]  (Warmup)
## Chain 4 Iteration:  600 / 4000 [ 15%]  (Warmup)
## Chain 4 Iteration:  700 / 4000 [ 17%]  (Warmup)
## Chain 4 Iteration:  800 / 4000 [ 20%]  (Warmup)
```

```
## Chain 4 Iteration:  900 / 4000 [ 22%]  (Warmup)
## Chain 4 Iteration: 1000 / 4000 [ 25%]  (Warmup)
## Chain 4 Iteration: 1100 / 4000 [ 27%]  (Warmup)
## Chain 4 Iteration: 1200 / 4000 [ 30%]  (Warmup)
## Chain 4 Iteration: 1300 / 4000 [ 32%]  (Warmup)
## Chain 4 Iteration: 1400 / 4000 [ 35%]  (Warmup)
## Chain 4 Iteration: 1500 / 4000 [ 37%]  (Warmup)
## Chain 4 Iteration: 1600 / 4000 [ 40%]  (Warmup)
## Chain 4 Iteration: 1700 / 4000 [ 42%]  (Warmup)
## Chain 4 Iteration: 1800 / 4000 [ 45%]  (Warmup)
## Chain 4 Iteration: 1900 / 4000 [ 47%]  (Warmup)
## Chain 4 Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 4 Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 4 Iteration: 2100 / 4000 [ 52%]  (Sampling)
## Chain 4 Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 4 Iteration: 2300 / 4000 [ 57%]  (Sampling)
## Chain 4 Iteration: 2400 / 4000 [ 60%]  (Sampling)
## Chain 4 Iteration: 2500 / 4000 [ 62%]  (Sampling)
## Chain 4 Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 4 Iteration: 2700 / 4000 [ 67%]  (Sampling)
## Chain 4 Iteration: 2800 / 4000 [ 70%]  (Sampling)
## Chain 4 Iteration: 2900 / 4000 [ 72%]  (Sampling)
## Chain 4 Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 4 Iteration: 3100 / 4000 [ 77%]  (Sampling)
## Chain 1 Iteration: 2500 / 4000 [ 62%]  (Sampling)
## Chain 1 Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 1 Iteration: 2700 / 4000 [ 67%]  (Sampling)
## Chain 1 Iteration: 2800 / 4000 [ 70%]  (Sampling)
## Chain 1 Iteration: 2900 / 4000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 1 Iteration: 3100 / 4000 [ 77%]  (Sampling)
## Chain 1 Iteration: 3200 / 4000 [ 80%]  (Sampling)
## Chain 1 Iteration: 3300 / 4000 [ 82%]  (Sampling)
## Chain 1 Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 1 Iteration: 3500 / 4000 [ 87%]  (Sampling)
## Chain 1 Iteration: 3600 / 4000 [ 90%]  (Sampling)
## Chain 1 Iteration: 3700 / 4000 [ 92%]  (Sampling)
## Chain 1 Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 1 Iteration: 3900 / 4000 [ 97%]  (Sampling)
## Chain 1 Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 1 finished in 0.2 seconds.
## Chain 2 Iteration: 3600 / 4000 [ 90%]  (Sampling)
## Chain 2 Iteration: 3700 / 4000 [ 92%]  (Sampling)
## Chain 2 Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 2 Iteration: 3900 / 4000 [ 97%]  (Sampling)
## Chain 2 Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 2 finished in 0.2 seconds.
## Chain 3 Iteration: 3100 / 4000 [ 77%]  (Sampling)
## Chain 3 Iteration: 3200 / 4000 [ 80%]  (Sampling)
## Chain 3 Iteration: 3300 / 4000 [ 82%]  (Sampling)
## Chain 3 Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 3 Iteration: 3500 / 4000 [ 87%]  (Sampling)
## Chain 3 Iteration: 3600 / 4000 [ 90%]  (Sampling)
## Chain 3 Iteration: 3700 / 4000 [ 92%]  (Sampling)
```

```
## Chain 3 Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 3 Iteration: 3900 / 4000 [ 97%]  (Sampling)
## Chain 3 Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 4 Iteration: 3200 / 4000 [ 80%]  (Sampling)
## Chain 4 Iteration: 3300 / 4000 [ 82%]  (Sampling)
## Chain 4 Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 4 Iteration: 3500 / 4000 [ 87%]  (Sampling)
## Chain 4 Iteration: 3600 / 4000 [ 90%]  (Sampling)
## Chain 4 Iteration: 3700 / 4000 [ 92%]  (Sampling)
## Chain 4 Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 4 Iteration: 3900 / 4000 [ 97%]  (Sampling)
## Chain 4 Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 3 finished in 0.2 seconds.
## Chain 4 finished in 0.2 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.2 seconds.
## Total execution time: 0.4 seconds.
```

Now we extract the MCMC data:/

```
# Extracting the data
stanfit <- rstan::read_stan_csv(fit_mcmc$output_files())
samples <- stanfit@sim$samples

chain1 <- samples[1][[1]]
alpha1 <- chain1$alphaNbeta.1
beta1 <- chain1$alphaNbeta.2

chain2 <- samples[2][[1]]
alpha2 <- chain2$alphaNbeta.1
beta2 <- chain2$alphaNbeta.2

chain3 <- samples[3][[1]]
alpha3 <- chain3$alphaNbeta.1
beta3 <- chain3$alphaNbeta.2

chain4 <- samples[4][[1]]
alpha4 <- chain4$alphaNbeta.1
beta4 <- chain4$alphaNbeta.2
```

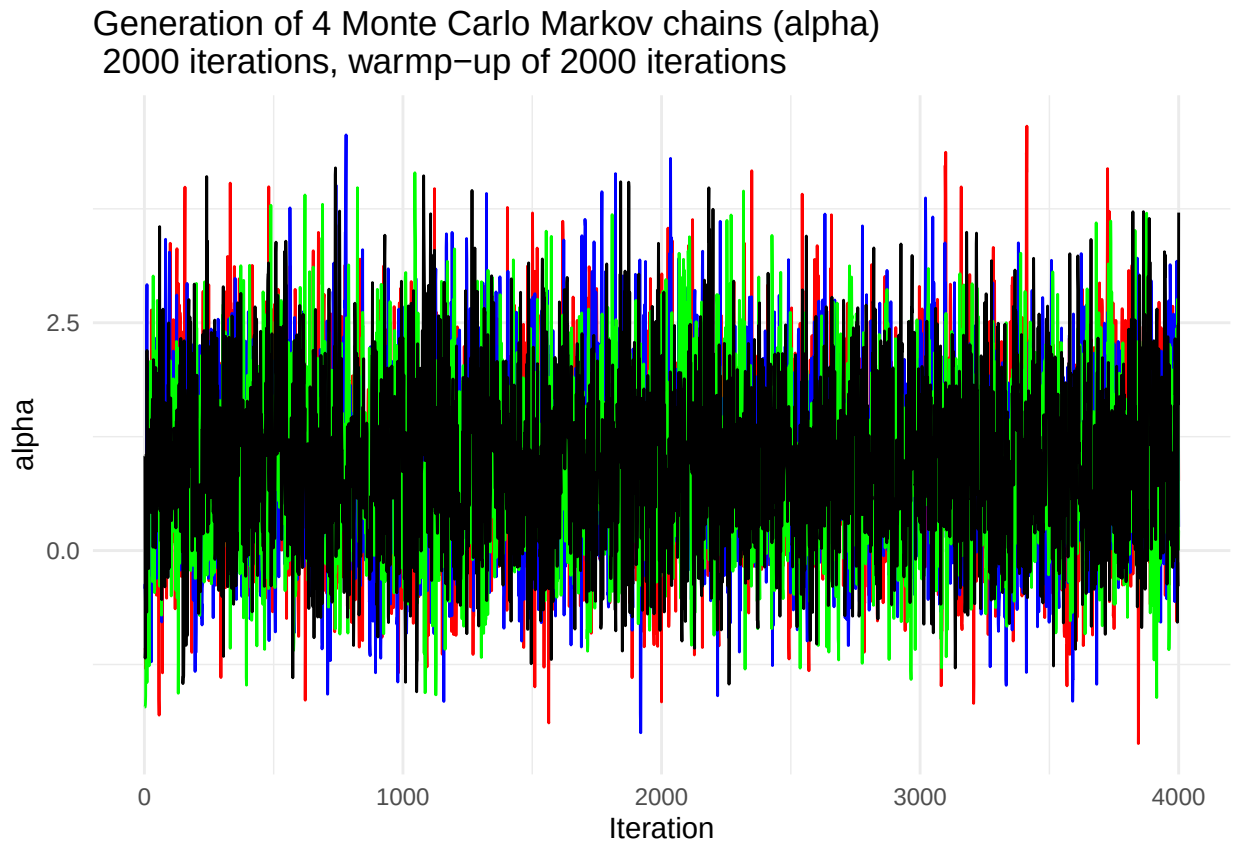
Plotting the MCMC for alpha

```
indices <- 1:length(alpha1)

data <- data.frame(indices, alpha1, alpha2, alpha3, alpha4)

ggplot(data, aes(x=indices)) +
  ggtitle(paste("Generation of", chains, "Monte Carlo Markov chains (alpha) \n",
                iter_sampling, "iterations, warmp-up of", iter_warmup,"iterations")) +
  xlab("Iteration") +
  ylab("alpha") +
  geom_line(aes(y = alpha1), color = "red") +
  geom_line(aes(y = alpha2), color = "blue") +
  geom_line(aes(y = alpha3), color = "green") +
```

```
  geom_line(aes(y = alpha4), color = "black")
```

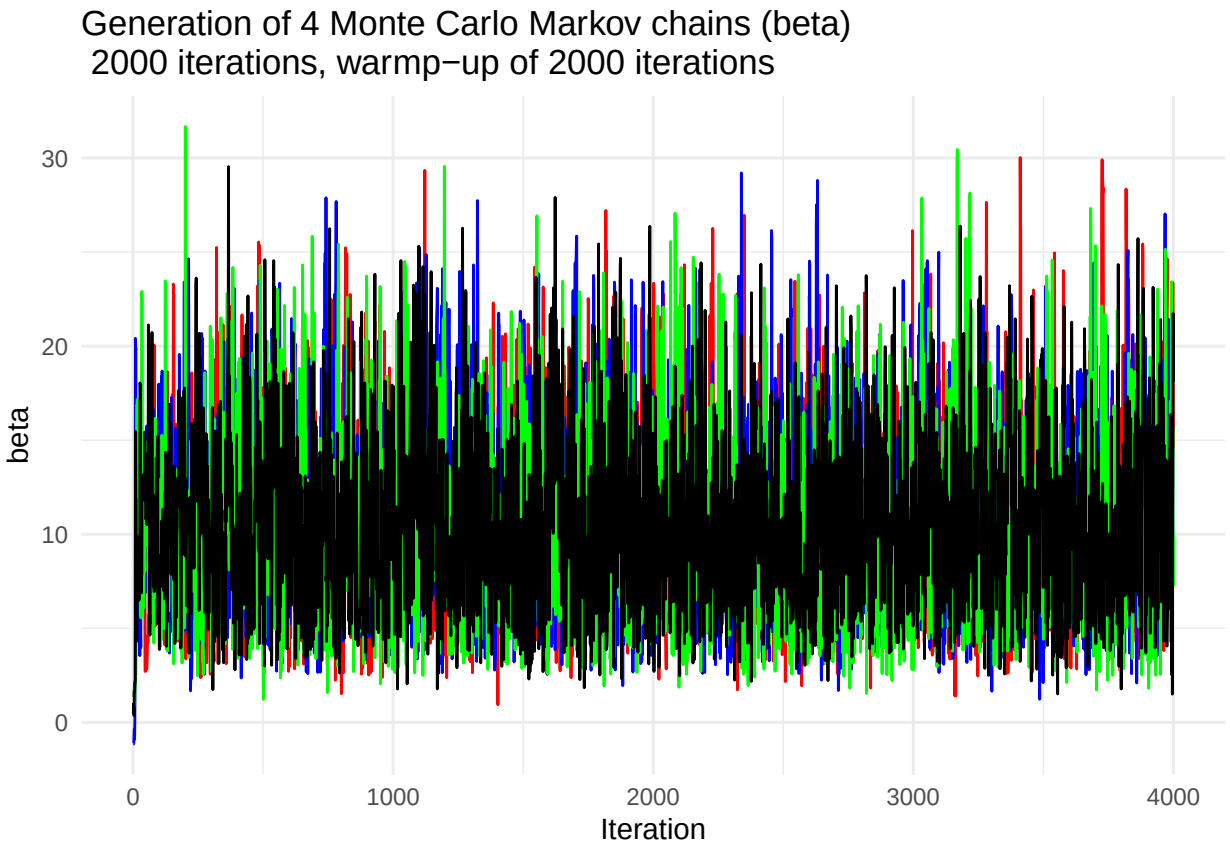


Plotting the MCMC for beta

```
library(ggplot2)
indices <- 1:length(beta1)

data <- data.frame(indices, beta1, beta2, beta3, beta4)

ggplot(data, aes(x=indices)) +
  ggtitle(paste("Generation of", chains, "Monte Carlo Markov chains (beta) \n",
                iter_sampling, "iterations, warmp-up of", iter_warmup,"iterations")) +
  xlab("Iteration") +
  ylab("beta") +
  geom_line(aes(y = beta1), color = "red") +
  geom_line(aes(y = beta2), color = "blue") +
  geom_line(aes(y = beta3), color = "green") +
  geom_line(aes(y = beta4), color = "black")
```



From the time series plot, we can see that both alpha and beta chains seem to have successfully converged.

2.

Use $\hat{R}$ for convergence analysis. You can either use Eq. (11.4) in BDA3 or the later version that can be found in a recent article. You should specify which Rb you used. In R the best choice is to use function rhat_basic() or rhat() from the posterior package (see ?posterior::rhat_basic). To check $\hat{R}$ and other diagnostics, you can also call fit$summary(), where fit is the fit object returned by Stan's sampling function. Report the $\hat{R}$ values both for α and β and discuss the convergence of the chains. Briefly explain in your own words how to interpret the obtained $\hat{R}$ values.

Calling the statistical summary from the fitting model

```
fit_mcmc$summary()
```

```
## # A tibble: 3 x 10
##   variable        mean median    sd   mad     q5   q95  rhat ess_bulk ess_tail
##   <chr>          <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>    <dbl>    <dbl>
## 1 lp__          -7.82  -7.50  1.00  0.700 -9.87  -6.87  1.00    2803.    3868.
## 2 alphaNbeta[1]  0.961  0.927 0.890 0.903 -0.413  2.49  1.00    2296.    3004.
## 3 alphaNbeta[2] 10.6    9.99  4.59  4.60   4.18  19.0   1.00    2394.    2650.
```

From the summary table, we can see that $\hat{R}$ of alpha is 1.001038 and $\hat{R}$ of beta is 1.001000486. Conventionally, $\hat{R}$ values smaller than 1.05 indicate the convergence of the chains. Since $\hat{R}$ of both alpha and beta are smaller than 1.05, it means that all MCMC of alpha and beta have successfully converged. The posterior draws thus can be obtained from the sampling iterations of the MCMC.

3.

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 your algorithm works perfectly, the results will look slightly different (although fairly similar).

Obtaining the draws for α and β

```
draws <- fit_mcmc$draws()
as_draws_df(draws)
```

```
## # A draws_df: 2000 iterations, 4 chains, and 3 variables
##     lp__ alphaNbeta[1] alphaNbeta[2]
## 1   -9.8          3.04            20
## 2  -10.1          2.05            23
## 3   -9.9          2.15            23
## 4   -9.3          2.78            19
## 5   -8.9          0.79            17
## 6   -8.1          1.48            17
## 7   -7.5          1.76            13
## 8   -7.9          2.13            13
## 9   -7.4          1.76            11
## 10  -7.3          1.58            13
## # ... with 7990 more draws
## # ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

Finally, the scatterplot of alpha-beta is:

```
color_scheme_set("teal")
plot <- mcmc_scatter(draws, pars = c("alphaNbeta[1]", "alphaNbeta[2]"))
plot +
  labs(
    title = "                                                    Alpha-Beta scatterplot",
    x = "alpha",
    y = "beta"
   )
```

Since this figure closely resembles the figure 3.3b, it means the code implementation is working properly.

4.

To develop the course and provide feedback to Stan developers, we collect information on
which Stan setup you used and whether you had any problems in setting it up or using it. Please report,

- Operating system (Linux, Mac, Windows) or jupyter.cs.aalto.fi?
  This assignment is done on Rstudio of jupyter.cs.aalto.fi

- Programming environment used: R or Python?
  The language I used is R

- Interface used: RStan, CmdStanR, PyStan, or CmdStanPy?
  The interface I use is CmdStanR

- Did you have installation or compilation problems?
  Yes. On my local laptop, I run CmdStanR and it reports the error
  *** Error in set_cmdstan_path(PATH_TO_CMDSTAN) :
  *** object 'PATH_TO_CMDSTAN' not found
  I cannot seem to use cmdstanr in my Rstudio because I cannot configure the path to the installation of CmdStanR. Then I manually search for the path and set the path with set_cmdstan_path(). When the path is correctly configured, as I load the model, another error about C++ dependencies occur. At this point I decided to move to jupiter.cs.aalto.fi to do my assignment

- Did you try first installing locally, but switched to jupyter.cs.aalto.fi?
  Yes, I did try to install first locally. But the installation and getting it to work is so confusing so I decided to switch to jupyter.cs.aalto.fi

- In addition of these you can write what other things you found out difficult (or even frustrating) when making this assignment with Stan.
  Yes. There are lots of errors with stan model loading, especially data type error about Array data type. I finally just ignored the error by setting model$compile(quiet = FALSE).