Assignment 5

anonymous

1 General information

I did not use AI for this solving exercise.

2 Generalized linear model: Bioassay model with Metropolis algorithm

2.1 (a)

```
# Useful functions: runif, rnorm
# bioassaylp, dmvnorm (from aaltobda)

data("bioassay")
# Start by implementing a function called `density_ratio` to
# compute the density ratio function, $r$ in Eq. (11.1) in BDA3:
density_ratio <- function(alpha_propose, alpha_previous, beta_propose, beta_previous, x, y, n){
    # Do computation here, and return as below.
    # Below are the correct return values for two different calls of this function:

    # 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
    #1.305179

    # 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
    #0.7661784

    sims <- 40000
    mean  <- c(0, 10)
    mean  <- matrix(mean, nrow = 2)
    sigma <- c(2^2, 12, 12, 10^2)
    sigma <- matrix(sigma, nrow = 2, ncol = 2)
    theta <- rmvnorm(sims, mean = mean, sigma = sigma)

    a1 <- alpha_propose
    a0 <- alpha_previous
    b1 <- beta_propose
    b0 <- beta_previous
```

```r
    # The proposal posterior in logs
    p1 <- bioassaylp(a1, b1, x = x, y = y, n = n) + dmvnorm(c(a1, b1), mean, sigma, log = TRUE)
    p0 <- bioassaylp(a0, b0, x = x, y = y, n = n) + dmvnorm(c(a0, b0), mean, sigma, log = TRUE)

    r <- exp(p1 - p0)

    return(r)
}
# Then implement a function called `metropolis_bioassay()` which
# implements the Metropolis algorithm using the `density_ratio()`:
metropolis_bioassay <- function(alpha_initial, beta_initial, alpha_sigma, beta_sigma, no_draws, x, y
    # Do computation here, and return as below.
    # Below are "wrong" values (unlikely to actually occur)
    # in the "correct" format (such that they work with the plotting functions further down).
    alpha_current <- alpha_initial
    beta_current <- beta_initial
    accepted_samples <- data.frame(alpha = numeric(0), beta = numeric(0))

    for (i in 1:no_draws) {
      alpha_proposed <- rnorm(1, mean = alpha_current, sd = alpha_sigma)
      beta_proposed <- rnorm(1, mean = beta_current, sd = beta_sigma)

      dens_ratio <- density_ratio(alpha_propose  = alpha_proposed,
                                  alpha_previous = alpha_current,
                                  beta_propose   = beta_proposed,
                                  beta_previous  = beta_current,
                                  x = x,
                                  y = y,
                                  n = n)

      # Accept or reject the proposed parameters
      if (1 < dens_ratio) {
        alpha_current <- alpha_proposed
        beta_current <- beta_proposed
      } else {
        u <- runif(1, min = 0, max = 1)
        if(u <= dens_ratio) {
          alpha_current <- alpha_proposed
          beta_current  <- beta_proposed
        }
      }

      # Store accepted samples
      accepted_samples <- rbind(accepted_samples, data.frame( t = i,
                                                              alpha = alpha_current,
                                                              beta = beta_current,
                                                              ratio = dens_ratio ))
    }

    return(accepted_samples)
}
theta0 <- rnorm(8, mean = 1, sd = 1)
theta0 <- matrix(theta0, nrow = 4, ncol = 2)
```

```
N = 1000
chain1 = metropolis_bioassay(theta0[1,1], theta0[1,2], 1, 5, N, bioassay$x, bioassay$y, bioassay$n)
chain2 = metropolis_bioassay(theta0[2,1], theta0[2,2], 1, 5, N, bioassay$x, bioassay$y, bioassay$n)
chain3 = metropolis_bioassay(theta0[3,1], theta0[3,2], 1, 5, N, bioassay$x, bioassay$y, bioassay$n)
chain4 = metropolis_bioassay(theta0[4,1], theta0[4,2], 1, 5, N, bioassay$x, bioassay$y, bioassay$n)
```

2.2 (b)

1. Metropolis-Hastings is a Monte Carlo method used for drawing samples from a "target distribution" when the parameters of that distribution are unknown or difficult to sample directly. Instead, it approximates the target distribution by iteratively modifying other distributions, gradually bringing them closer to the desired target. The goal is to make these approximating distributions converge to the target distribution, and this process is implemented as a Markov Chain simulation.

The target distribution, denoted as $p(\theta)$, is proportional to a known function $g(\theta)$. The Metropolis-Hastings algorithm follows these steps:

- **Initialization**: Start with an initial value for θ, denoted as θ_0.
- **Main Loop** (for t in iterations):
  - **Candidate Sample**: Generate a candidate sample θ^* from a proposal distribution $q(\theta^*|\theta_{t-1})$.
  - **Density Ratio Computation**: Calculate the ratio of densities, denoted as r.
  - **Acceptance/Rejection Step**: Depending on the value of this density ratio, either accept θ^* as the new value θ_t, or reject it and keep θ_t unchanged. This step is governed by the following rule:

$$\theta_t = \begin{cases} \theta^* \text{ with probability min(r, 1)} \\ \theta_{t-1} \text{ otherwise} \end{cases}$$

In essence, the Metropolis-Hastings algorithm is a powerful technique for exploring and sampling from complex probability distributions by iteratively adjusting and accepting/rejecting candidate samples to ultimately approximate the desired target distribution.

2. Simple normal proposal distribution was employed for both α and β. To propose a new value for α, we sampled from a normal distribution, denoted as $\alpha^* \sim N(\alpha_{t-1}, \sigma)$, where the mean is set to the previously accepted α value. The choice of the normal distribution aligns with the use of a bivariate Gaussian prior for both α and β during the posterior calculation.

Similarly, for proposing a new β value, we used $\beta^* \sim N(\beta_{t-1}, \sigma)$, where the mean is based on the last accepted β. In this context, σ is set to 1 for the distribution related to α and 5 for the distribution related to β.

For higher values of σ, the acceptance rate of the Metropolis algorithm is lower, meaning that there are more rejected candidate values, so the convergence is slower and the performance, in general, is worse. This is the reason why the previous distributions were selected. To summarize, the chosen distributions meet the following requirements, stated in the book BDA3:

- Sampling from the proposal distribution is straightforward.
- Computing the ratio is straightforward.
- Each step in the parameter space covers a reasonable distance.
- The jumps are not rejected too frequently.

3. The preferred approach involves selecting overdispersed initial values for each parameter. In the Metropolis-Hasting method, the first sample is a random value generated during the initialization step. In this case, I have opted to derive the initial values from a Normal distribution..

For the given case, I used following generated matrix:

```
print(theta0)
```

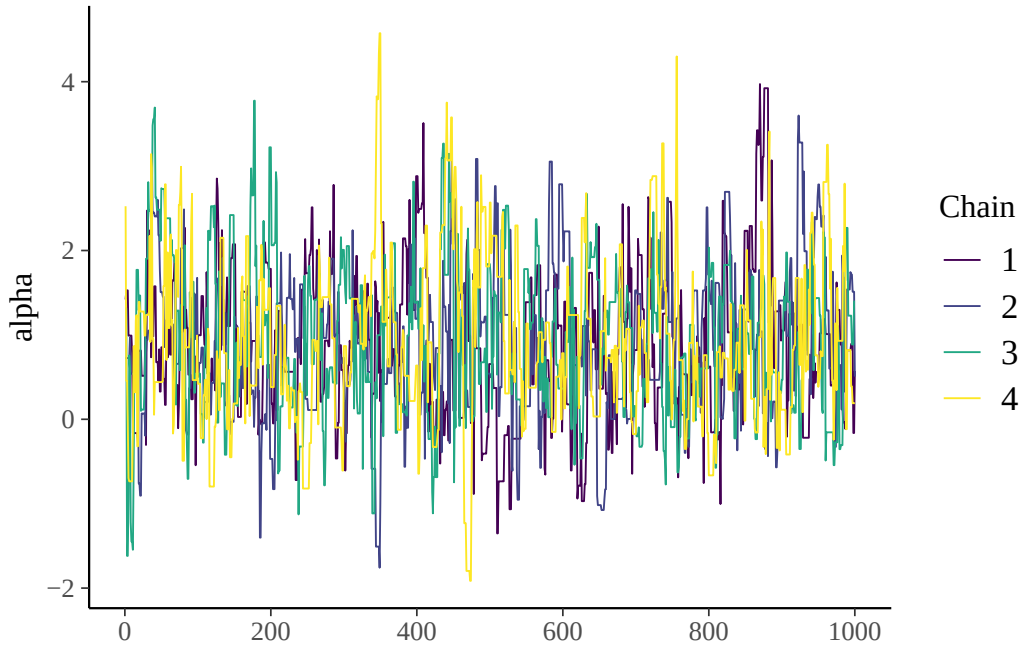
```
         [,1]       [,2]
[1,] 1.419735 -0.4540684
[2,] 1.011133  2.0163612
[3,] 1.315790  1.7943332
[4,] 2.524510  1.1361324
```

4. The chains were built by iterating the Metropolis algorithm a total number of 1000 times, meaning that this is the length of each of the chains. The reason why this number was chosen is based on graphical evidence by carrying out a trial error process using scatter plots above this report.
5. The warm-up length is used to "*diminish the influence of the starting values*" (BDA3). As proposed in the book, half of each sequence is discarded, meaning that the final number of points is 500.
6. In the given case, 4 Metropolis chains were computed.

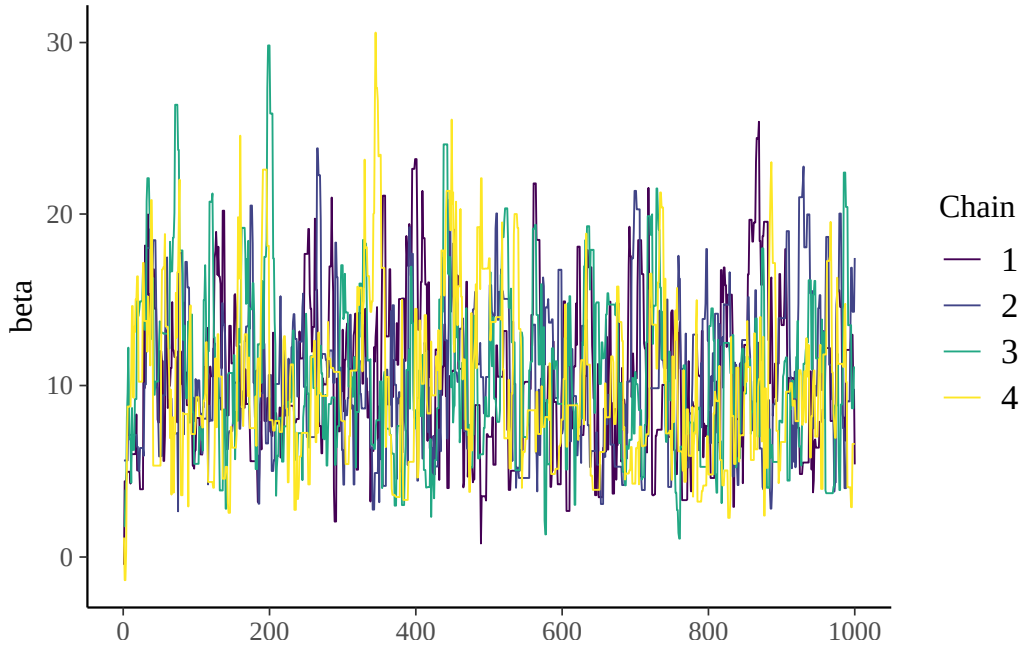
7 and 8.

```
df <- rbind(
  data.frame(chain = 1, chain1),
  data.frame(chain = 2, chain2),
  data.frame(chain = 3, chain3),
  data.frame(chain = 4, chain4)
)
# Set the color scheme (optional)
color_scheme_set("viridis")

# Create an MCMC trace plot for the parameter "alpha" and "beta"
mcmc_trace(df, regex_pars = "alpha")
```




```
mcmc_trace(df, regex_pars = "beta")
```



In the analysis, four separate Markov chains are employed to estimate the values of parameters α and β using chosen proposal distributions. The observed behavior of these chains reveals a notable convergence trend, with sampled parameter values progressively concentrating around a single point. This convergence signifies that the Markov chains are effectively exploring the parameter space and settling towards the most likely values for α and β given the observed data.

2.3 (c)

```
# Useful functions: rhat_basic (from posterior)
warmuprate <- .5
chain1Warmed <- chain1[chain1$t > length(chain1$t)*warmuprate,]
chain2Warmed <- chain2[chain2$t > length(chain2$t)*warmuprate,]
chain3Warmed <- chain3[chain3$t > length(chain3$t)*warmuprate,]
chain4Warmed <- chain4[chain4$t > length(chain4$t)*warmuprate,]

alpha_sims <- cbind.data.frame(chain1Warmed$alpha,
                               chain2Warmed$alpha,
                               chain3Warmed$alpha,
                               chain4Warmed$alpha)
alpha_sims <- as.matrix(alpha_sims)
r_alpha <- rhat_basic(alpha_sims)
cat(paste("R-hat for", ' alpha ', ":", r_alpha, "\n"))
```

```
R-hat for  alpha  : 1.01501997106251
```

```
beta_sims <- cbind.data.frame(chain1Warmed$beta,
                              chain2Warmed$beta,
                              chain3Warmed$beta,
                              chain4Warmed$beta)
beta_sims <- as.matrix(beta_sims)
r_beta <-rhat_basic(beta_sims)
cat(paste("R-hat for", ' beta ', ":", r_beta, "\n"))
```

```
R-hat for  beta  : 1.02899163853839
```

The basic idea of $\hat{R}$:

$\hat{R}$ is a metric that quantifies the variance between two parameters which

- First one measures the between sequence variance, meaning how different is each sequence from the others, and
- Second one measures the within sequence variance, meaning how different parameters within the same sequence are.

The expression for $\hat{R}$, according to BDA3 is:

$$\hat{R_{metric}} = \sqrt[2]{(\frac{\hat{var}^{+}(\phi|y)}{W})}$$

For a larger number of samples, this metric tends to approach 1, indicating that the variability between the two aforementioned parameters is minimal. This concept is analogous to clustering metrics, where inter-cluster and intra-cluster similarities are compared to assess the success of clustering. It is recommended running at least four chains by default and in general only fully trust the sample if R-hat is less than 1.01. In early workflow, $\hat{R}$ below 1.1 is often sufficient, which is our case here.

Obtained $\hat{R}$ is good:

Calculating the $\hat{R}$ values for the chains becomes straightforward after removing the warm-up phase of the chain. In the given scenario with 1000 iterations (excluding the initial 500 warm-up iterations from the calculation), the metric for α is

$$R_{\text{metric, alpha}} = 1.02 < 1.1$$

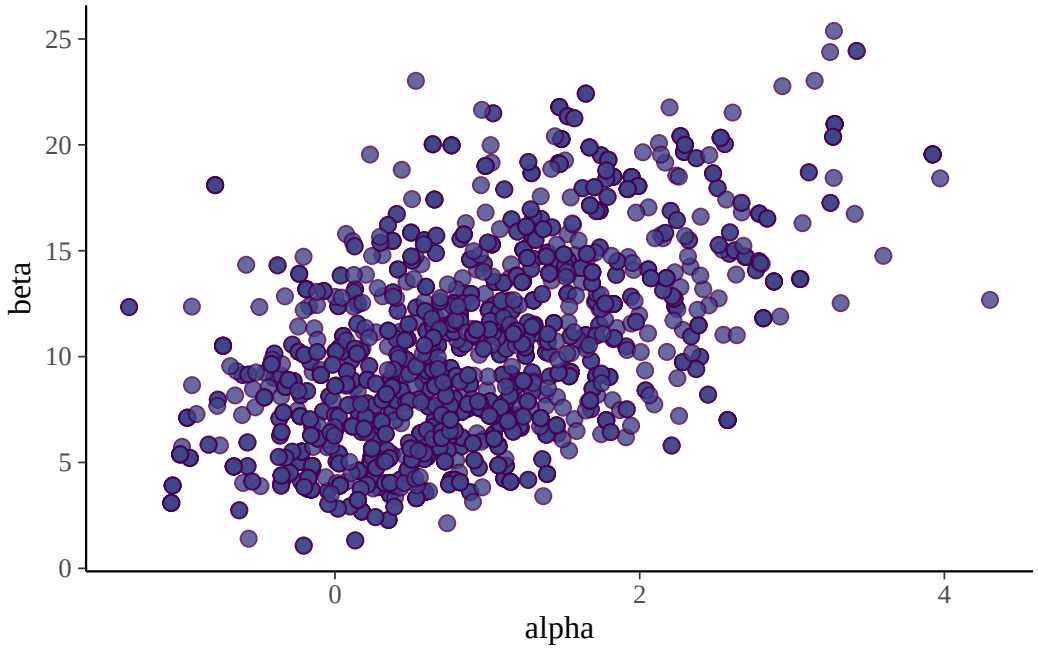
, and for β, it is

$$R_{\text{metric, beta}} = 1.03 < 1.1$$

. In both cases, the values are close to 1, and according to the Stan documentation, a metric lower than 1.1 indicates that the sample is acceptable, which is the current situation.

2.4 (d)

```r
# Useful functions: mcmc_scatter (from bayesplot)
df <- rbind(
  data.frame(chain = 1, chain1[chain1$t > length(chain1$t)*warmuprate,]),
  data.frame(chain = 2, chain2[chain2$t > length(chain2$t)*warmuprate,]),
  data.frame(chain = 3, chain3[chain3$t > length(chain3$t)*warmuprate,]),
  data.frame(chain = 4, chain4[chain4$t > length(chain4$t)*warmuprate,])
)
# Set the color scheme (optional)
color_scheme_set("viridis")
mcmc_scatter(df, pars=c("alpha", "beta"))
```


beta
25
20
15
10
5
0
0
2
4
alpha