Assignment 5

anonymous

General information

AI was used in part b to search some related information and code.

Generalized linear model: Bioassay model with Metropolis algorithm

(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 th
is 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
    mu0 <- c(0, 10)
    sigma0 <- matrix(c(2^2, 12, 12, 10^2), ncol=2)

    log_propose <- bioassaylp(alpha_propose, beta_propose, x=x, y=y, n=
n) + dmvnorm(c(alpha_propose, beta_propose), mean=mu0, sigma=sigma0, lo
g=TRUE)
    log_previous <- bioassaylp(alpha_previous, beta_previous, x=x, y=y,
 n=n) + dmvnorm(c(alpha_previous, beta_previous), mean=mu0, sigma=sigma
0, log=TRUE)

    ratio <- exp(log_propose - log_previous)
```

```r
    return(ratio)
}
#test samples
density_ratio(1.89,0.374,24.76,20.04,bioassay$x,bioassay$y,bioassay$n)
```

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

```r
density_ratio(0.374,1.89,20.04,24.76,bioassay$x,bioassay$y,bioassay$n)
```

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

```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_sigm
a, beta_sigma, no_draws, x, y, n){

    burn_in<-2500
    # Total number of iterations considering burn-in period
    total_iter <- no_draws + burn_in

    # Initialize arrays to store our samples
    chain_alpha <- numeric(total_iter)
    chain_beta <- numeric(total_iter)

    chain_alpha[1] <- alpha_initial
    chain_beta[1] <- beta_initial

    for(i in 2:total_iter){
        # Propose new values from normal distribution
        alpha_propose <- rnorm(1, mean=chain_alpha[i-1], sd=alpha_sigma)
        beta_propose <- rnorm(1, mean=chain_beta[i-1], sd=beta_sigma)

        # Calculate the density ratio for the proposed values
        r <- density_ratio(alpha_propose, chain_alpha[i-1], beta_propos
e, chain_beta[i-1], x, y, n)

        # Accept or reject the proposed values
        if(runif(1) < min(1,r)){
            chain_alpha[i] <- alpha_propose
            chain_beta[i] <- beta_propose
        } else {
            chain_alpha[i] <- chain_alpha[i-1]
            chain_beta[i] <- chain_beta[i-1]
        }
    }

    # Discard the burn-in samples
    chain_alpha <- chain_alpha[-(1:burn_in)]
    chain_beta <- chain_beta[-(1:burn_in)]
    # Return the samples in a data frame
    return(data.frame(alpha=chain_alpha, beta=chain_beta))
}

df = metropolis_bioassay(0, 0, 1, 5, 5000, bioassay$x, bioassay$y, bioa
ssay$n)
```

(b)

1. The Metropolis algorithm is a Markov Chain Monte Carlo method used to obtain a series of random samples from probability distributions that are difficult to directly sample. The algorithm starts from an arbitrary initial value. In each step, the algorithm selects candidate values for the next sample value based on the proposed distribution. Then accept the candidate value with a probability related to the ratio of the value of the function on the candidate value to the current sample value. If the candidate value is rejected, the current sample is added to the sequence again. After a sufficiently long step, this process will roughly generate samples from the desired distribution. Start with an arbitrary point x0 as the initial state of the Markov chain,Iteration: For each step t,we need to ropose a candidate state x' using the proposal distribution q(x'|xt),where xt is the current state and then we need compute acceptance probability:$A(x_t, x^{'}) = \min(1, \frac{p(x^{'})q(x_t|x^{'})}{p(x_t)q(x^{'}|x_t)})$. Where p(x) is the target distribution. q(x'|x) is the proposal distribution, which provides the probability of transitioning from state x to x'. And then we need to consider if we Accept the candidate state x' with probability $A(x_t, x^{'})$, if x' is accepted, then set x_{t+1}=x'; otherwise,set x_{t+1}=x_t. Finally we need continue the iteration process for a desired number of steps or until another stopping criterion is met.

2.The proposal distribution: I used a normal distribution as the proposed distribution. Specifically, for α, I use the mean as the current α Normal distribution of values and setting the standard deviation to 1; about β, I use the mean as the current β Normal distribution of values and setting the standard deviation to 5. The reason for choosing this proposed distribution is that the normal distribution has properties that are easy to calculate and understand, and it is symmetric, which is a favorable feature in the Metropolis algorithm. In addition, by adjusting the standard deviation of the normal distribution, the acceptance rate of the algorithm can be easily controlled, thereby adjusting the proposed "jump" size. Before conducting the simulation, I tested several different standard deviations and ultimately chose the value that generated a reasonable acceptance rate.

3.In the Metropolis algorithm, I chose the initial value alpha_Initial=0 and beta_Initial=0 serves as the starting point of the Metropolis chain.

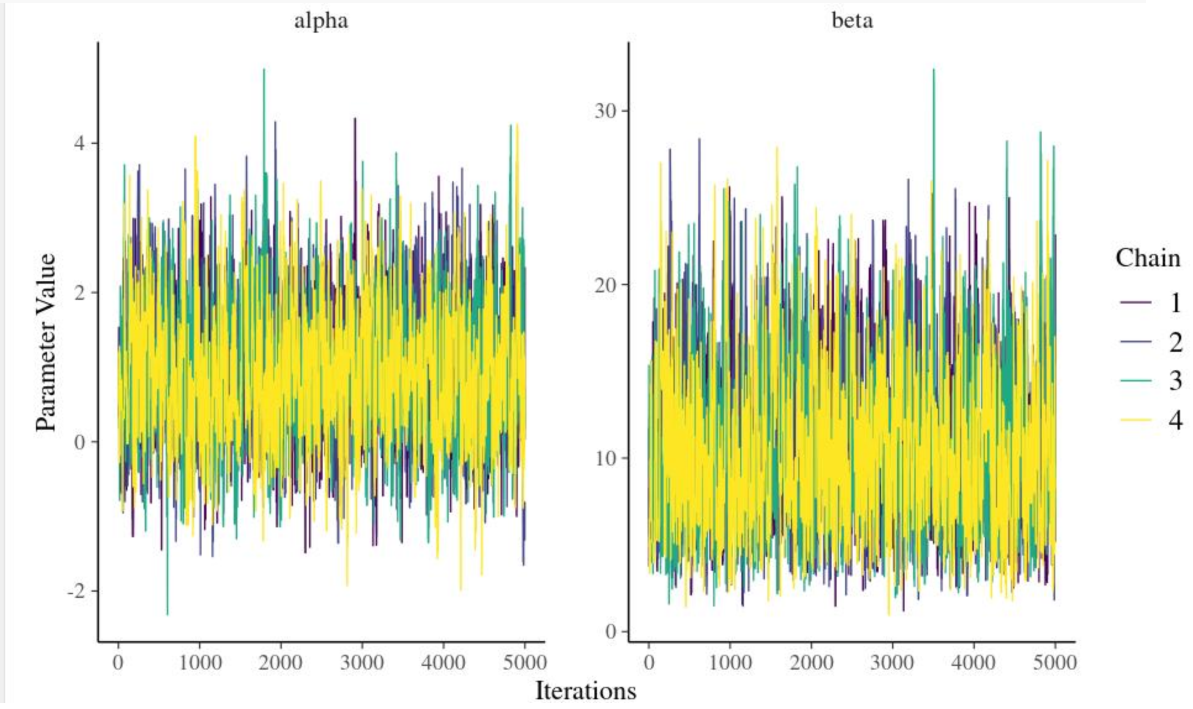
4.To ensure approximate convergence of the algorithm, I choose 5000 as chain length.

5.I choose 2500 as warm-up length.

6.We generated 4 Metropolis chains and simulated 6 parameters. Through the output, we can see that there are four chains named "chain: 1", "chain: 2", "chain: 3", and "chain: 4".

Have a look at bayesplot trace plot examples and tune your plot if wanted/needed. Don't forget to include a title/caption/description.

```r
library(bayesplot)
library(ggplot2)
chains_num <- 4
iterations_per_chain <- 5000

mcmc_results <- array(0, dim = c(iterations_per_chain, chains_num, 2))
dimnames(mcmc_results) <- list(Iteration = NULL, Chain = paste0("chain:
", 1:chains_num), Parameter = c("alpha", "beta"))

for (chain in 1:chains_num) {
  df <- metropolis_bioassay(0, 0, 1, 5, iterations_per_chain, bioassay
$x, bioassay$y, bioassay$n)
  mcmc_results[, chain, 1] <- df$alpha
  mcmc_results[, chain, 2] <- df$beta
}
color_scheme_set("viridis")
plot<-mcmc_trace(mcmc_results, pars=c("alpha", "beta"))
plot + labs(x = "Iterations", y = "Parameter Value")
```



(c)

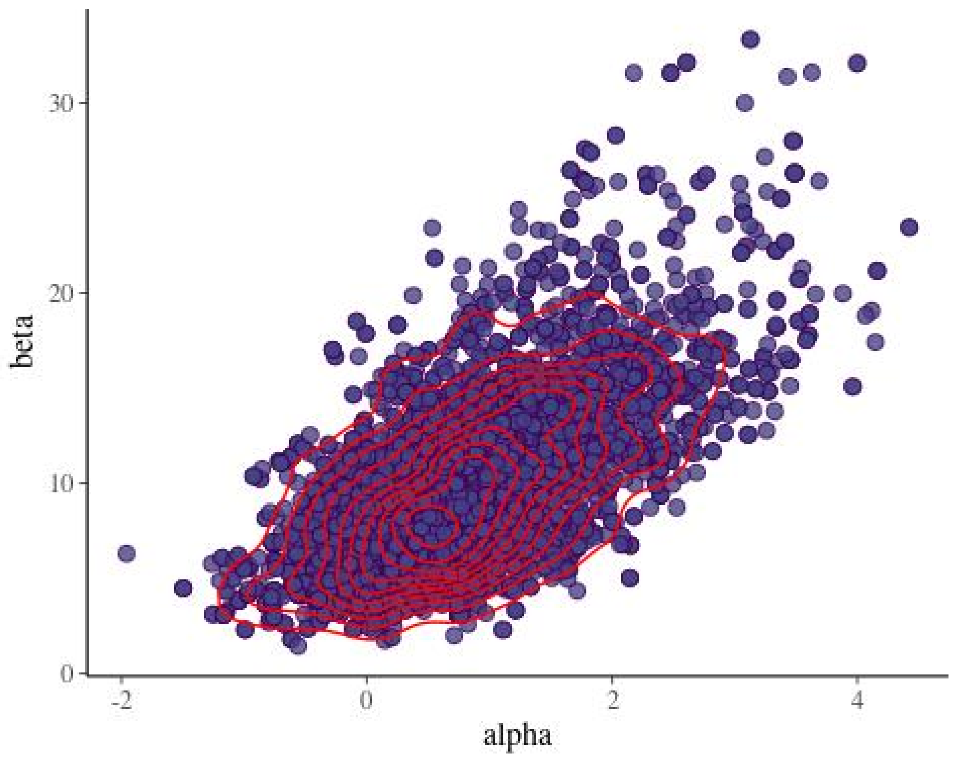
1. basic idea of $\widehat{R}$ and how to to interpret the obtained $\widehat{R}$values: $\widehat{R}$is a statistic used for evaluating whether MCMC chains have converged. It compares the variance both between and within multiple chains to determine if all chains are sampling from the same distribution. Specifically, if each chain is sampling from the target distribution (that is, they have converged), the differences between these chains should merely be random noise. Thus, the essential idea of $\widehat{R}$is to compare the variance between chains with the variance within chains. When the $\widehat{R}$value is close to 1, it suggests that the variance between the chains is similar to the variance within the chains, indicating a sign that the MCMC chains have converged. Typically, if $\widehat{R}$ is less than 1.01, we consider the chains to have converged.

```
rhat_alpha = rhat_basic(mcmc_results[, chain, 1])
rhat_beta = rhat_basic(mcmc_results[, chain, 2])

print(paste("Rhat for alpha: ", rhat_alpha))

## [1] "Rhat for alpha:  1.00966502402461"

print(paste("Rhat for beta: ", rhat_beta))

## [1] "Rhat for beta:  1.00767629913581"

# Useful functions: rhat_basic (from posterior)
```

2.For my runs, the $\widehat{R}$ values for α and β were 1.009 and 1.007 respectively. These values are very close to 1, indicating that I obtained a good $\widehat{R}$ on my first attempt. This means I didn't need to run more iterations or modify the proposal distribution. Such values suggest that my MCMC chains have converged and the chosen proposal distribution was appropriate.

(c)

```
# Useful functions: mcmc_scatter (from bayesplot)
p2<-mcmc_scatter(df, pars=c("alpha", "beta"))
p2 + stat_density_2d(color = "red")
```



beta
30
20
10
0
-2
0
2
4
alpha