BDA - Assignment 5

Anynomous

```
# install.packages("remotes")
# remotes::install_github("avehtari/BDA_course_Aalto",
#                         subdir = "rpackage", upgrade="never")
library(aaltobda)
library(posterior)
```

```
This is posterior version 1.3.0


Attaching package: 'posterior'

The following object is masked from 'package:aaltobda':

    mcse_quantile

The following objects are masked from 'package:stats':

    mad, sd, var
```

```
data("bioassay")
```

Exercise 1) - (Code implementations)

a) - (density_ratio)

I have used the *bioassaylp*() function to calculate the log-likelihood for both the propose and previous values. TO calculate the gaussian prior, I have used the *dmvnorm*(). Both functions are provided in the *aaltobda* package.

```
density_ratio <- function(alpha_propose, alpha_previous, beta_propose, beta_previous, x, y

  # Unormalized log posteriror = log-likelihood + log-prior
  covMatrix<- rbind(c(4,12),c(12,100))

  propose <- bioassaylp(alpha_propose, beta_propose, x=x,  y=y, n=n) +
          dmvnorm(x = c(alpha_propose, beta_propose), mean = c(0,10),
                  sigma = covMatrix, log=TRUE)

  previous <- bioassaylp(alpha_previous, beta_previous, x=x,  y=y, n=n) +
          dmvnorm(x = c(alpha_previous, beta_previous), mean = c(0,10),
                  sigma = covMatrix, log=TRUE)

  dens_ratio <- exp(propose - previous) # exp(log(p1) - log(p0))

}
```

b) - (metropolis_biossay and other helping functions)

Below is the code to generate a metropolis algorithm run.

```
metropolis_bioassay <- function(iterations) {
  alpha <- c()
  beta <- c()

  # Generate start value
  alpha[1] <- runif(1, 1, 4) # Random number
  beta[1] <- runif(1, 100, 200)

  for (i in 2:iterations){
    alpha_propose <- rnorm(1, alpha[i-1], 1) # mean = previous_alpha, sd = 1
    beta_propose <- rnorm(1, beta[i-1], 5) # mean = previous_beta, sd = 5

    # Calcualte density ration and take minimum value of 1 or ratio
    r <- density_ratio(alpha_propose, alpha[i-1], beta_propose, beta[i-1],
                           x = bioassay$x, y = bioassay$y, n = bioassay$n)
    r <- min(1,r)

    # Jumping rule
    ## if random number from 0 to 1 is less than r
```

```
    if (runif(1,0,1)< r ) {
      alpha[i] <- alpha_propose
      beta[i] <- beta_propose

    } else { # Keep previous parameter
      alpha[i] <- alpha[i-1]
      beta[i] <- beta[i-1]
    }
  }

  list("alpha" = alpha, "beta" = beta) # Return values of alpha and beta

}
```

Here is the metropolis algorithm that runs the defined amount of chains

```
chained_metropolis <- function(chain_amount, iterations, warmup=1){
  alpha <- c()
  beta <- c()

  for (i in 1:chain_amount){
    result <- metropolis_bioassay(iterations)
    alpha[[i]] <- result$alpha[warmup:length(result$alpha)] #cutoff warm-up
    beta[[i]] <- result$beta[warmup: length(result$beta)] # cutoff warm-up

  }

  list("alpha" = alpha, "beta" = beta) # Return values of alpha and beta
}
```

Exercise 2)

a)

The Metropolis algorithm is a Markov Chain Monte Carlo method for simulating sampling from a distribution. The main idea of the algorithm is to choose an arbitrary value as a starting point and simulate a proposal. With a random probability, the new value is kept, otherwise, the previous value is used.

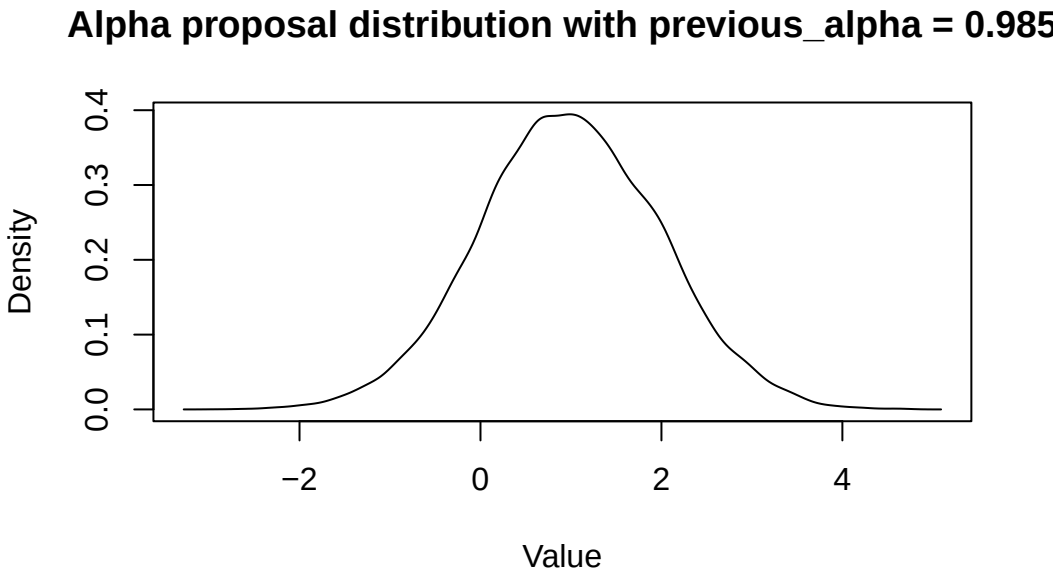
b)

The proposal distributions are normal distributions that use the previous iteration value for the mean and a pre-defined standard deviation value. I have used the standard deviation values provided by the assignment, that is alpha $\sigma = 1$ and beta $\sigma = 5$.

This leaves the final proposal distributions to look like $\alpha^* \sim N(\alpha_{t-1}, \sigma = 1)$ and $\beta^* \sim N(\beta_{t-1}, \sigma = 5)$.

Here is an example of the proposal distribution for alpha.

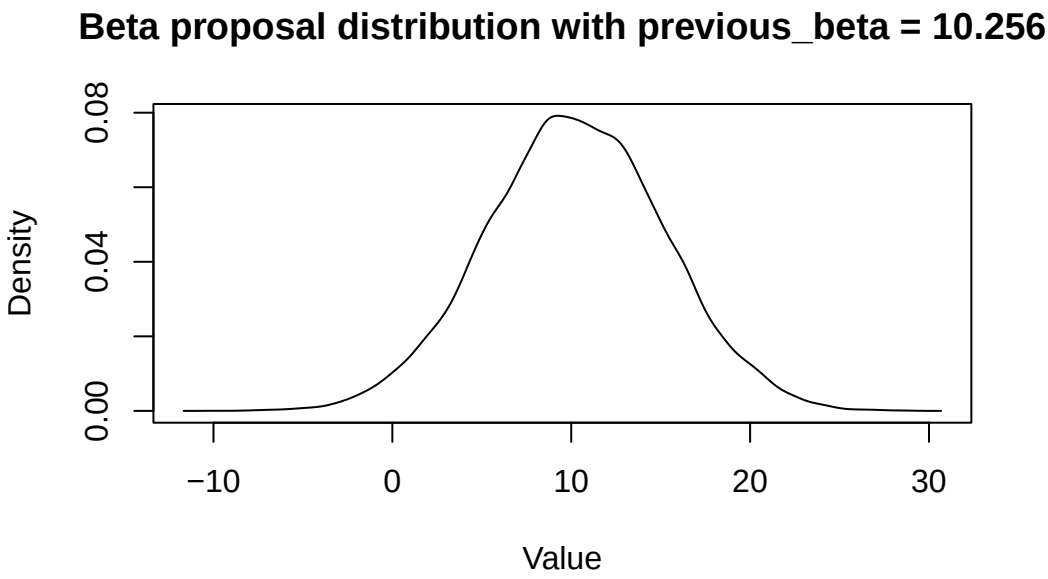
```
aMean <- 0.985
aStd <- 1
dens <- density(rnorm(20000, aMean, aStd ))
plot(dens$x,length(data)*dens$y,type="l",xlab="Value",ylab="Density",
     main="Alpha proposal distribution with previous_alpha = 0.985")
```



Here is an example of the proposal distribution for beta.

```
bMean <- 10.256
bStd <- 5
dens <- density(rnorm(20000, bMean, bStd ))
plot(dens$x,length(data)*dens$y,type="l",xlab="Value",ylab="Density",
```

```
    main="Beta proposal distribution with previous_beta = 10.256")
```



c)

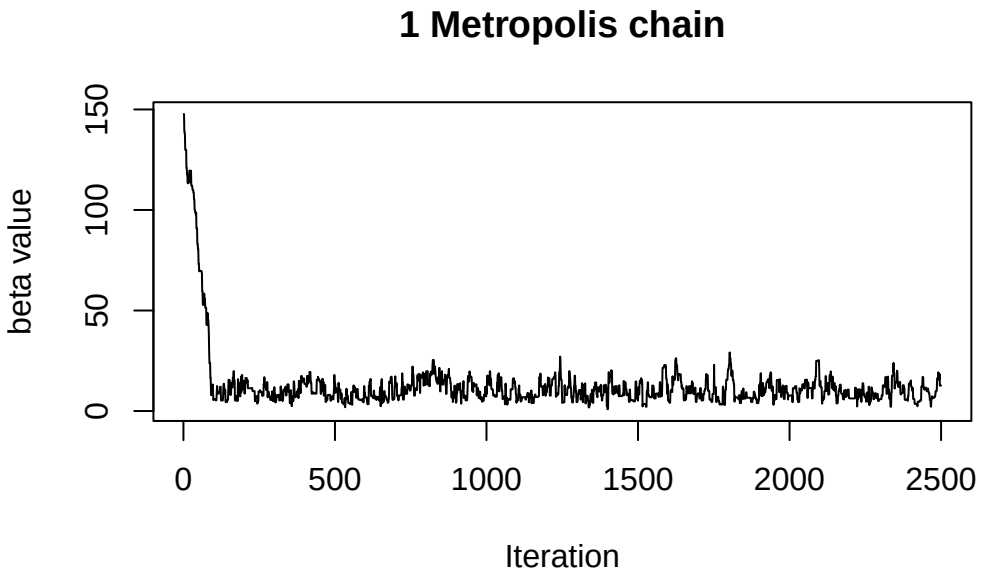
The initial points for my metropolis chains are an arbitrary values. It is chosen randomly from a value range roughly in the domain of the described normal distributions of this exercise. The starting point can be a crude estimate because the values will converge after a warm-up period. The code below is what I have used.

```
# Generate start value
#alpha[1] <- runif(1, 1, 4) # Random number
#beta[1] <- runif(1, 100, 200) # Random number
```

d)

The amount of iterations can be customized. To be sure the algorithm converges, I have decided to use around 2500 iterations. The algorithm already converges before the 250th iteration. We can see this from the plot below.

```
result <- metropolis_bioassay(2500)
plot(result$beta, type="l",xlab="Iteration",ylab="beta value",
     main="1 Metropolis chain" )
```



e)

The warm-up lenght is the period where the probabilities have not yet converged. From the image in section d), we can see that the warm-up length is less than 250. By this point the algorithm has already converged. For safety and to be sure it has converged, I would cut of the first 500, leaving still 2000 samples.

f)

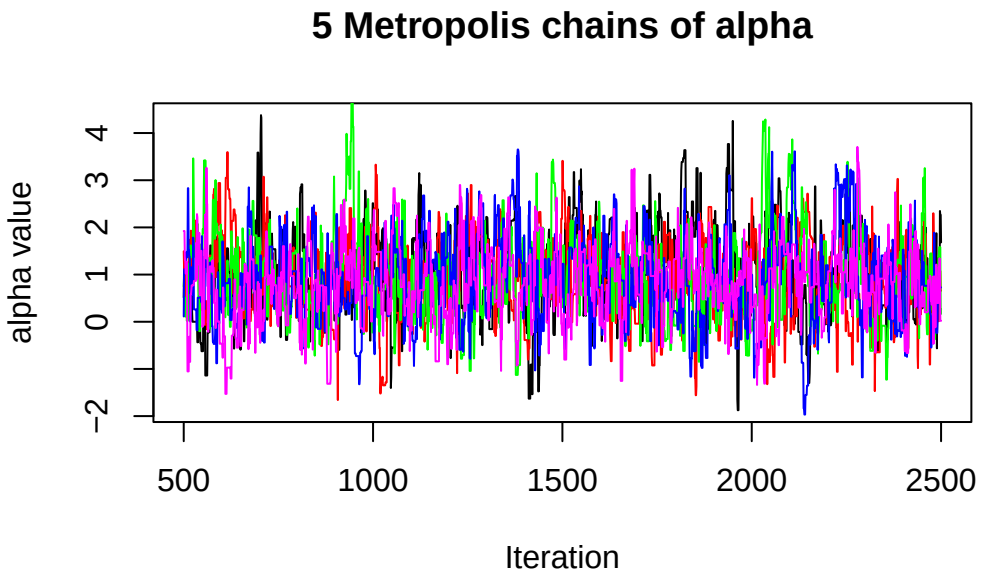
For my analysis, I have used 5 chains. The reason behind this is purely because the course book said that they had used 5 simulations because we need clearly more than one simulation

g)

The image in section d) shows that the data has converged after 250 iterations. This is not shown in the plots below because we have cut off the warm-up period. From the overlapping

chains, we can visually validate that the oscillation in the data is not differing that much. The larger differences in the value are caused by the random walks the algorithm does. This can be fixed by reducing the chance of exploration after a certain amount of iterations.

```
result <- chained_metropolis(5, 2500, 500)
x <- seq(500, 2500)
plot(x, unlist(result$alpha[1]), type="l",xlab="Iteration",ylab="alpha value",
     main="5 Metropolis chains of alpha" )
lines(x, unlist(result$alpha[2]), col="red")
lines(x, unlist(result$alpha[3]), col="green")
lines(x, unlist(result$alpha[4]), col="blue")
lines(x, unlist(result$alpha[5]), col="magenta")
```

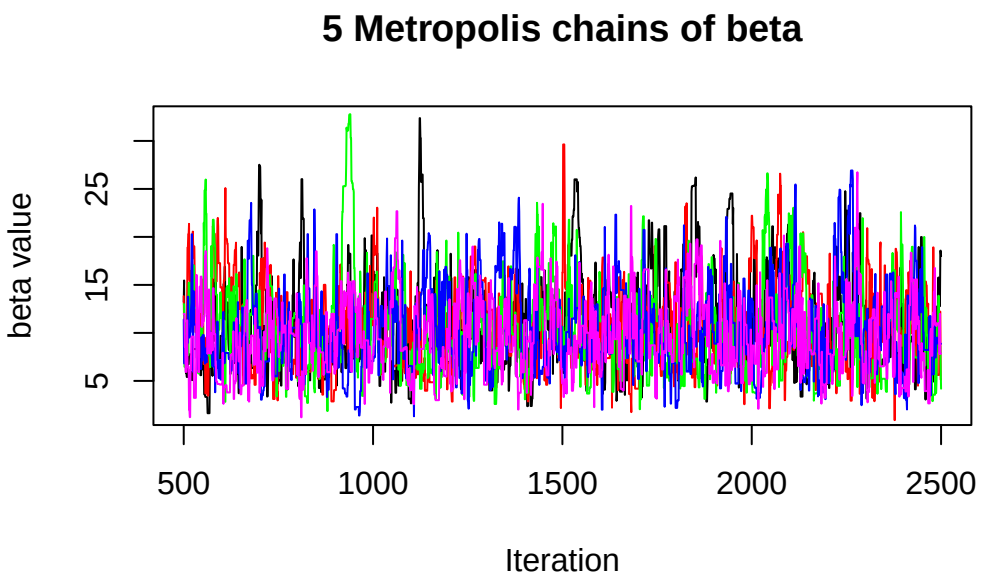


h)

The exact same things mentioned in section g) applies for the beta values.

```
plot(x, unlist(result$beta[1]), type="l",xlab="Iteration",ylab="beta value",
      main="5 Metropolis chains of beta" )
lines(x, unlist(result$beta[2]), col="red")
lines(x, unlist(result$beta[3]), col="green")
```

```
lines(x, unlist(result$beta[4]), col="blue")
lines(x, unlist(result$beta[5]), col="magenta")
```



Exercise 3)

a)

Rhat is a convergence analytic formula that compares chain estimates for alpha and beta in our case. The closer the values are to 1 the better the convergence is. This should mean that all the chains have converged.

b)

I decided to use the recommended function rhat_basic( ). The values I got on the first try are good because I decided to use many iterations. If I would have used less than 250 iterations the Rhat value would be much larger.

```
print(c("Alpha Rhat", rhat_basic(unlist(result$alpha))))
```

```
[1] "Alpha Rhat"       "1.00370152513643"
```

```
print(c("Beta Rhat", rhat_basic(unlist(result$beta))))
```

```
[1] "Beta Rhat"        "1.00750586807755"
```

The alpha Rhat value is 1.0 and beta Rhat value is 1.0

Exercise 4)

The scatterplot looks similar to the one in BDA3 (Figure 3.3b). This would give me some confidence that the algorithm is working as supposed to.

```
plot(unlist(result$alpha[1]), unlist(result$beta[1]), ylim=c(-10,40), xlim=c(-4,10))
points(unlist(result$alpha[2]), unlist(result$beta[2]))
points(unlist(result$alpha[3]), unlist(result$beta[3]))
points(unlist(result$alpha[4]), unlist(result$beta[4]))
points(unlist(result$alpha[5]), unlist(result$beta[5]))
```