Assignement 5

Anonymous

Contents

1. Implement the Metropolis algorithm as an R function for the bioassay data **1**

a). 1

b) 2

2. Include in the report the following **4**

a). Metropolis algorithm 4

b). Proposal distribution 4

c). The initial points of your Metropolis chains 4

d). Chain length & simulations 4

e). Warm-up length 5

f). The number of Metropolis chains used 5

g). Plot all chains for alpha in a single line-plot 5

h). Plot all chains for beta in a single line-plot 6

3). Rhat **7**

a). 8

b). 8

4). Scatter plot of alpha and beta **8**

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

a).

Data loading

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

Define parameters for the Gaussian prior distribution

```
Mean = c(0, 10)
Cov = matrix(c(4, 12, 12, 100), nrow=2)
```

How to compute r:

- the density ratio is given by equation (11.1): $\frac{p(\theta^*|y)}{p(\theta^{t-1}|y)}$ that links the proposed and the previous thetas (theta = (alpha, beta)).
- each of these posteriors is proportional to the product: prior posterior. So the fraction can be calculated by dividing the two quantities.
- the log_posterior can be calculated directly using the provided function: bioassaylp, and the prior can be calculated using the provided function: dmvnorm
- the final posterior is the exponential of the sum of the two logs (prior and posterior)

Now we implement the function

```
density_ratio = function(alpha_propose, alpha_previous, beta_propose,
                         beta_previous, x, y, n){

  log_likelihood = bioassaylp(alpha_propose, beta_propose, x, y, n) -
                   bioassaylp(alpha_previous, beta_previous, x, y, n)

  log_prior = log(dmvnorm(c(alpha_propose, beta_propose), Mean, Cov)) -
                   log(dmvnorm(c(alpha_previous, beta_previous), Mean, Cov))

  posterior = exp(log_likelihood + log_prior)

  return(posterior)
}
```

```
density_ratio(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] 1.305179
```

```
density_ratio(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)
```

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

b)

This is the function of the Metropolis algorithm (Explanations are in part 2).

```r
metropolis_bioassay = function(chain_length){

  ######################## Theta 0 choice ########################


  theta_0_condidats = rmvnorm(100, Mean, Cov)

  likelihood_theta_0 = exp(bioassaylp(theta_0_condidats[, 1],
                       theta_0_condidats[, 2], bioassay$x, bioassay$y, bioassay$n))

  prior_theta_0 = dmvnorm(theta_0_condidats, Mean, Cov)

  non_normalized_posterior = likelihood_theta_0 * prior_theta_0

  theta_0 = theta_0_condidats[which.max(non_normalized_posterior), ]



  ######################## Metropolis  ########################

  theta_previous = theta_0
  J_Cov = matrix(c(1, 3, 3, 5^2), nrow=2)

  generated_points = matrix(ncol=2, nrow=chain_length)

  for(t in 1:chain_length){

    J_Mean = theta_previous
    theta_proposed = rmvnorm(1, J_Mean, J_Cov)

    ratio = density_ratio(theta_proposed[, 1], theta_previous[1],
                          theta_proposed[, 2], theta_previous[2],
                          x = bioassay$x, y = bioassay$y, n = bioassay$n)

    prob = runif(1)

    if( prob <= min(ratio, 1) ){

      generated_points[t, ] = theta_proposed
      theta_previous = theta_proposed
    }

    else {

      generated_points[t, ] = theta_previous
    }
  }


  return(generated_points);
}
```

2. Include in the report the following

a). Metropolis algorithm

The metropolis algorithm is used in order to generate samples from a (known but hard to sample from) density. We start with an initial point (that its choice can affect the performance), then we have two choices: either we repeat the previous point (with a defined probability using density ratio), or we move to another point suggested by a proposal density. Repeating this process for enough iterations will start giving samples from the target distribution.

b). Proposal distribution

We choosed a bivariate normal distribution as the joint distribution for $\theta^* = (\alpha^*, \beta^*)$, where: $\alpha^* \sim N(\alpha_{t-1}, \sigma = 1)$, $\beta^* \sim N(\beta_{t-1}, \sigma = 5)$ and we considered the same correlation coefficient of the prior, then: $corr(\alpha^*, \beta^*) = 0.6$

So, $(\alpha^*, \beta^*) \sim N(\theta_{t-1}, \sum^*)$ where:

$$\sum^* = \begin{bmatrix} 1 & 3 \\ 3 & 25 \end{bmatrix}$$

c). The initial points of your Metropolis chains

To generate initial points:

- we draw 100 samples as candidates from the prior distribution of $\theta = (\alpha, \beta)$
- we calculate their priors and likelihoods, and then their non-normalized posteriors
- we chose θ that have the maximum posterior

After these steps, we will get a probable (form posterior density) initial point, that can help to converge quickly.

d). Chain length & simulations

We will choose 5 chains (executions of the algorithm) of 5000 points (iterations) each, which is long enough for approximate convergence.

```
chain_length = 5000
n_chains = 5

chains = array(dim=c(chain_length, 2, n_chains)) # to store the chains

for(m in 1:n_chains){

  chains[ , , m] = metropolis_bioassay(chain_length)
}
```

e). Warm-up length

- As explained in the book (11.4), we will consider the first half of each chain as warm up (2500 points). So, we will use only: thetas from 2501 to 5000.

```
warmed_up_chains = array(dim=c(chain_length/2, 2, n_chains)) # to store values after warm up

warmed_up_chains = chains[2501:5000, , ]
dim(warmed_up_chains)
```

```
## [1] 2500    2    5
```

f). The number of Metropolis chains used

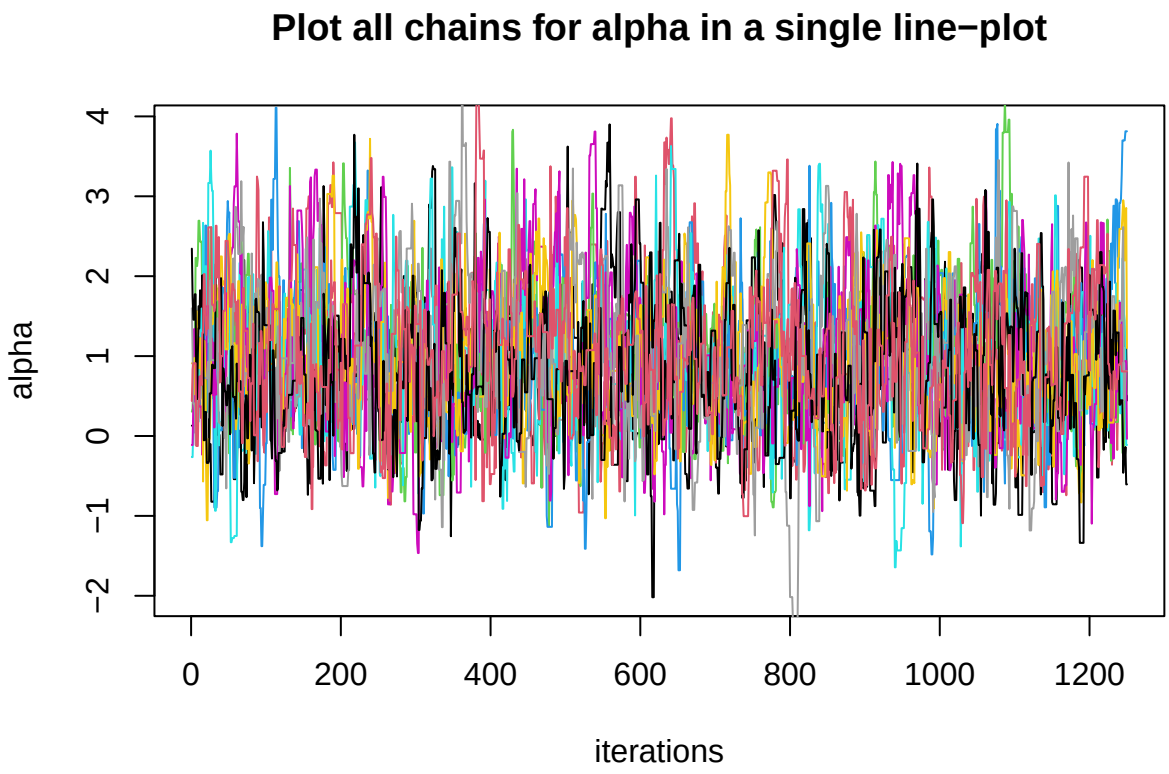
As stated in the book (11.4 page 284) we will split saved sequences (chains) into two parts. As we have 5 chains with 2500 points each after warming up, after the splitting, we will have 10 chains of 1250 points each.

```
splitted_chains = array(dim=c(chain_length/4, 2, 2*n_chains))

for (m in 1:n_chains){

  splitted_chains[, , 2*m-1] = warmed_up_chains[1:1250, , m]
  splitted_chains[, , 2*m] = warmed_up_chains[1251:2500, , m]
}
dim(splitted_chains)
```

```
## [1] 1250    2   10
```

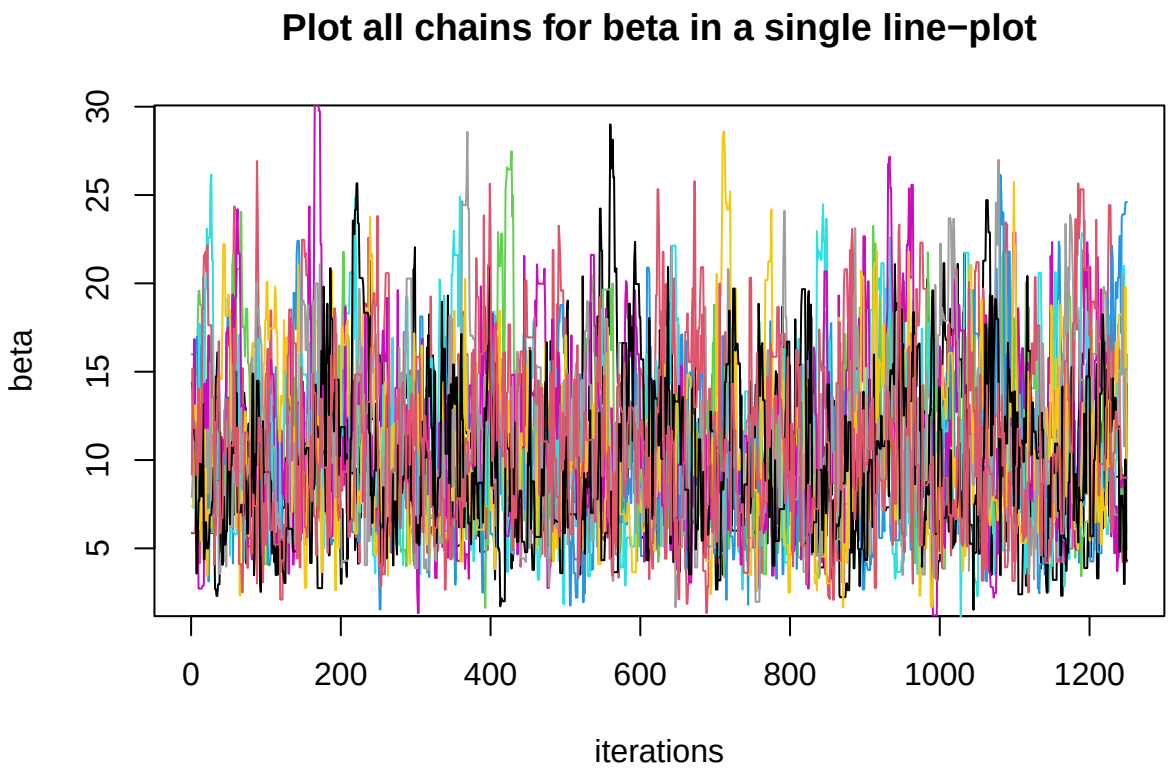
g). Plot all chains for alpha in a single line-plot

```
plot(1:1250, splitted_chains[, 1, 1], type="l", lwd=1, col=1,
    main="Plot all chains for alpha in a single line-plot", xlab="iterations",
    ylab="alpha")


for (m in 2:10)
  {
  lines(1:1250, splitted_chains[, 1, m], type="l", lwd=1, col=m)
  }
```



h). Plot all chains for beta in a single line-plot

```
plot(1:1250, splitted_chains[, 2, 1], type="l", lwd=1, col=1,
     main="Plot all chains for beta in a single line-plot", xlab="iterations"
     , ylab="beta")

for (m in 2:10)
  {
  lines(1:1250, splitted_chains[, 2, m], type="l", lwd=1, col=m)
  }
```



Convergence

As shown in the plots of alpha and beta:

- the sequences are mixed in the sense that they have traced out a common distribution.
- each of the chains are oscillating : fast one and slow one, the slow oscillation is around 1 for alpha and 10 for beta (which are close to the real values according to previous results like assignment 4). So we have stationarity.

Therefore, we can consider that we achieved convergence.

3). Rhat

As recommended, we will use the *rhat_basic()* function from the package *posterior*

```
## Warning: package 'posterior' was built under R version 4.1.3
```

```
## This is posterior version 1.3.1
```

```
##
## Attaching package: 'posterior'
```

```
## The following object is masked from 'package:aaltobda':
##
##     mcse_quantile
```

```
## The following objects are masked from 'package:stats':
##
##     mad, sd, var
```

a).

Rhat or the potential scale reduction measures the ratio of the variance within each sequence and the variance between the sequences. So if the convergence is achieved, Rhat will be closer to 1, otherwise it will be greater that 1.

How Rhat can be interpreted?:

- if it is higher than 1, we need to perform more simulations or use more efficient algorithm implementation, because that reduces the ratio of variances and lead to convergence.
- if it is close to 1, we can conclude that each of the sequences of simulated observations is close to the target distribution (convergence).

b).

The proposal distribution used here is still the same as declared in part 2.b

Calculating Rhat for alpha:

```
rhat_basic(splitted_chains[, 1, ]) #spitted_chains is the  matrix of 10 chains that
```

```
## [1] 1.007777
```

```
                                   #we got after warm up and splitting
```

Calculating Rhat for beta:

```
rhat_basic(splitted_chains[, 2, ]) #spitted_chains is the  matrix of 10 chains that
```

```
## [1] 1.005992
```

```
                                   #we got after warm up and splitting
```

Interpretation of Rhat values

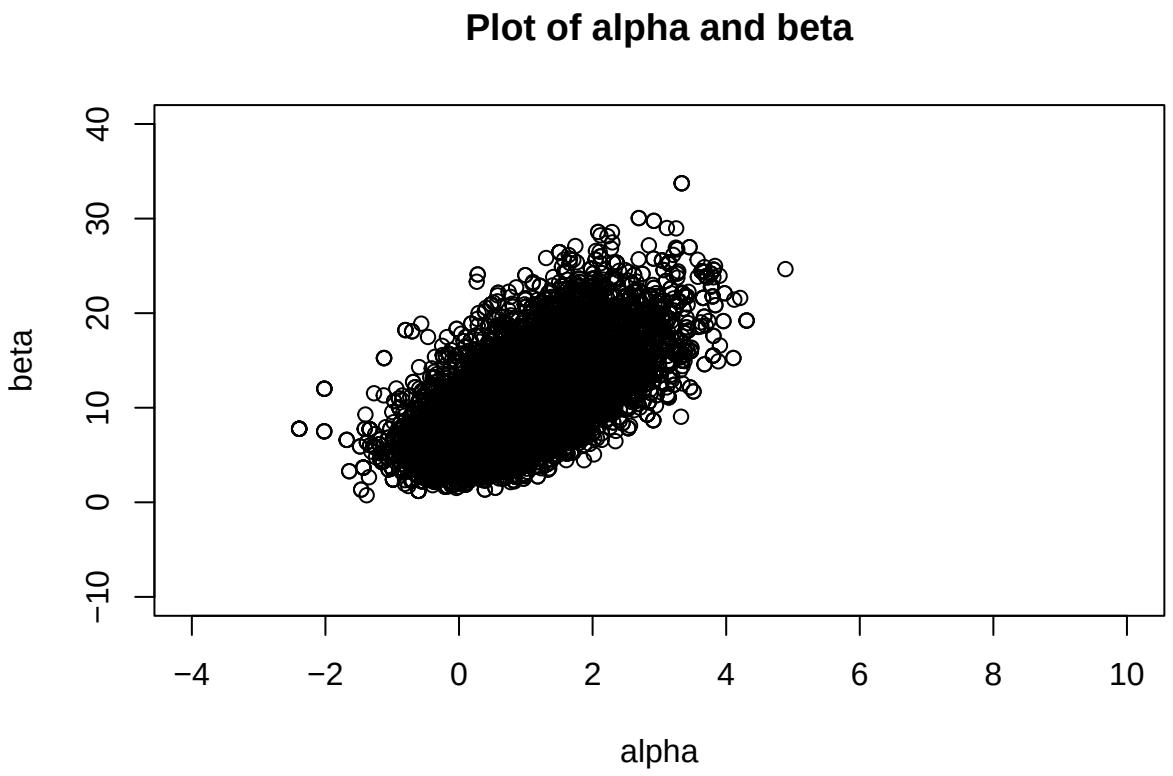
We can see that the values got for Rhat are very close to 1. Therefore, we can consider the convergence achieved, and the generated sequences belong to the target distribution.

So, there is no need to change the proposal distribution or perform more iterations

4). Scatter plot of alpha and beta

For this plot we will use points of alpha and beta from all 10 chains of 1250 points each.

```
plot(splitted_chains[, 1, ], splitted_chains[, 2, ], xlim = c(-4, 10),
     ylim=c(-10, 40), xlab="alpha", ylab="beta", main="Plot of alpha and beta")
```



This looks like the figure 3.3b in the book. But there is a small difference due to the use of different prior, and we used more points (12500), and the fact that our figure is rectangular and the book figure is squared (but I used the same scale).