BDA - Assignment 5

Anonymous

12 10 2019

Contents

Used libraries:

```
library(dplyr)
library(ggplot2)
library(doParallel)
library(aaltobda)
library(knitr)
library(rstan)
data("bioassay")
```

Exercise 1

Implement the Metropolis algorithm as an R function for the bioassay data. Use the Gaussian prior as in Assignment 4, that is

$$\begin{pmatrix}\alpha\\ \beta\end{pmatrix} \sim N(\mu_0, \Sigma_0)$$

, where

$$\mu_0 = \begin{pmatrix}0\\ 10\end{pmatrix} \Sigma_0 = \begin{pmatrix}2^2 & 10\\ 10 & 10^2\end{pmatrix}$$

a)

Start by implementing the density ratio function to compute r in Eq. (11.1) in BDA3.

Density ratio with known (to a constant) distribution:

$$r = \frac{p(\theta^*|y)}{p(\theta^{t-1}|y)}$$

I am using as utilities the log-prior -and posterior functions I implemented in one of the previous assignments:

```
p_log_prior <- function(alpha, beta) {

```

```
  s <- matrix(c(4,10,10,100),nrow = 2)
  mu <- c(0,10)

  #Here the two input vectors are bound columnwise, and put in a matrix
  m <- cbind(alpha, beta) %>% as.matrix(.,nrow=length(alpha[,1]))
  # dmvnorm seemed to operate on matrices in a reasonable way
  dmvnorm(m, mu, s, log = T)

}

#these default values for the likelihood are a bit unnecessary in both functions, but here goes
#anyway...
p_log_posterior <- function(alpha, beta, x = bioassay$x, y = bioassay$y, n = bioassay$n) {

  p_log_prior(alpha, beta) + bioassaylp(alpha, beta, x,y,n)

}
```

Now I can just use *p_log_posterior*, and exponentiate in a following manner:

$$\frac{p_*}{p_{t-1}} = \exp(log(p_*) - \log(p_{t-1}))$$

Here is a function to do just that:

```
density_ratio <- function(alpha_propose, alpha_previous, beta_propose, beta_previous,
                          x=bioassay$x, y=bioassay$y, n=bioassay$n) {

  (p_log_posterior(alpha_propose, beta_propose) -
     p_log_posterior(alpha_previous, beta_previous)) %>% exp()

}
```

Let's **test(!)**:

```
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] 1.187524
```

And another **test(!)**:

```
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] 0.8420882
```

Seems to be functioning a ok . . .

b)

Now implement the metropolis algorithm using density_ratio() as a function called metropolis_bioassay(). Be sure to define your starting points and your jumping rule (proposal distribution). Run the simulations long enough for approximate convergence.

I am implementing a function that takes initial values as parameters. Number of initial values defines the number of chains as well as number of threads (I will run the chains in parallel). Also the size of sample,

and the warmup period are given as parameters. In order to make the functions definition to conform to the assignment, I will give all parameters some plausible default values in order to make the function callable also without any arguments.

I am mostly not using the default sample sizes, so I will point them out as they become relevant (when I run the function). The point of convergence was practically found through experimentation, and therefore I didn't hard code any checks for it.

NOTE:I am using rnorm( ) to pull the proposals, so in all cases, the proposal for both alpha, beta is a univariate normal (jumping distance/sigma can be explicitly given as a parameter).

```
metropolis_bioassay <- function(initial_alpha = c(-4,-4,6,6), initial_beta = c(-5,35,-5,35),
                                sample_size = 1500, warmup = 500, step_alpha = 1, step_beta = 5) {

  chains <- length(initial_alpha)

  # There is a more concise and "easy" way of parallelizing this code,
  # but this is kind of a brute force method (just ramming all in)
  cl <- makeCluster(chains)
  clusterExport(cl, varlist=c("density_ratio", "p_log_posterior","p_log_prior",
                              "bioassay"), env=topenv())
  registerDoParallel(cl)

  res <- foreach(i=1:chains,.packages = c('dplyr', 'aaltobda')) %dopar% {  #.combine = 'rbind'

    label <- i
    alphas <- rep(-100, sample_size)
    betas <- rep(-100, sample_size)
    alphas[1] <- initial_alpha[i]
    betas[1] <- initial_beta[i]
    lucky <- runif(sample_size)

    for(j in 2:sample_size) {
      alpha_prop <- rnorm(1,mean=alphas[j-1], step_alpha)
      beta_prop <- rnorm(1, mean=betas[j-1], step_beta)

      r <- density_ratio(alpha_prop, alphas[j-1],
                  beta_prop, betas[j-1])

      if(lucky[j] < r) {
        alphas[j] <- alpha_prop
        betas[j] <- beta_prop
      } else {
        alphas[j] <- alphas[j-1]
        betas[j] <- betas[j-1]
      }

    }

    state <- rep('red', length(alphas))
    state[1:warmup] <- 'blue'
    data.frame(alphas, betas, chain=label, state, iterations=1:sample_size)

  }
```

```
  stopCluster(cl)

  res

}
```

The parallel metropolis returns a list, and for plotting and rstan::Rhat I will need some small utils to transform the data:

```
#this is just an utility function to rowbind the list from metropolis (for plots)
rowbind <- function(lista) {
  res <- foreach(i=1:length(lista), .combine = 'rbind') %do% {
    res[[i]]
  }
  res
}

#This is an utility for column binding for Rhat
colbind <- function(lista, tag) {
  res <- foreach(i=1:length(lista), .combine = 'cbind') %do% {
    tmp <- lista[[i]] %>% select(tag)
  }
  res
}
```

I will do the run in exercise 2 (below). Since I can put everything in as parameter values, I don't have to define and hard code them here. I hope this, and the next exercise covers this part...

Exercise 2.

Include in the report the proposal distribution, number of chains used, the starting points (or the mechanism for generating them), the number of draws generated from each chain, and the warm-up length. Also plot the chains separately for alpha and beta.

Here are the specifications for a single run with parallel metropolis: 6 chains, origin points (α, β) are (-4,-5),(-4,35),(1,-6),(1,40),(6,-5),(6,35) which are purposefully chosen "badly" to demonstrate how the probability mass still pulls the path in the correct direction. Sample size is 2500 (per chain) of which 500 are considered a warmup (500 "blue" and 2000 "red" points).

As **proposal** distributions I will use the given univariate normals, with $\sigma_\alpha = 1$ and $\sigma_\beta = 5$. I did also try out different sigmas (i.e., jump sizes) but these seemed to work just fine.

So, the run:

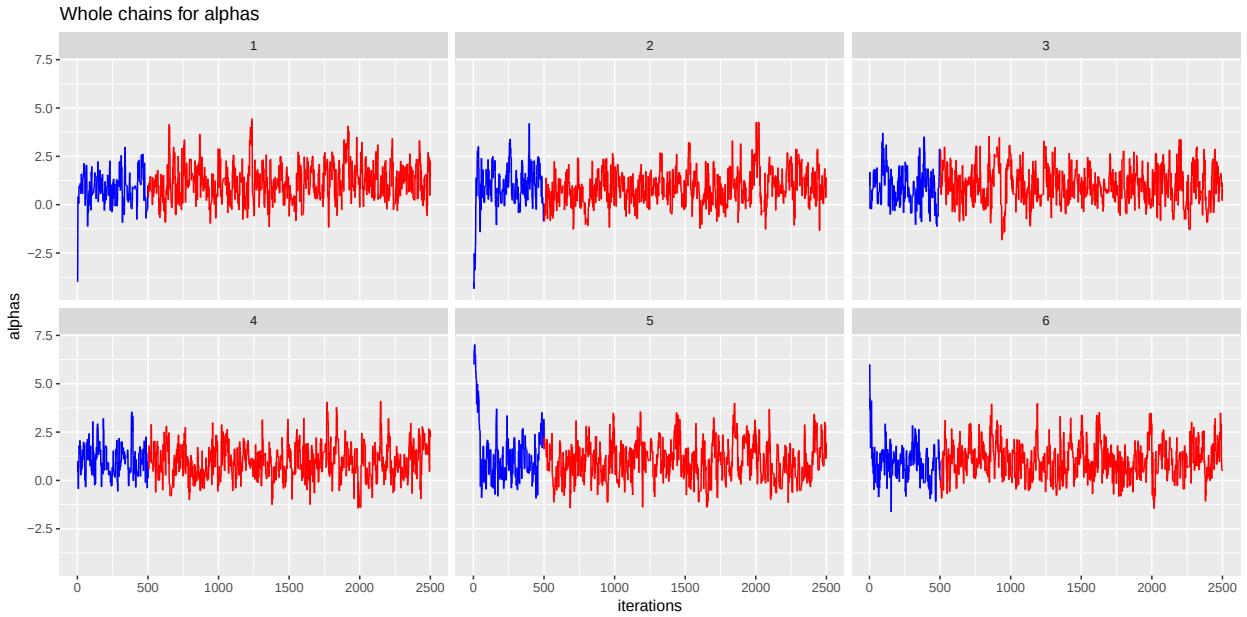
```
res <- metropolis_bioassay(initial_alpha = c(-4,-4,1,1,6,6), initial_beta = c(-5,35,-6,40,-5,35),
                           sample_size = 2500, step_alpha = 1,step_beta = 5)
```

```
## [1] "This is the form of the output. All chains are rowbound into a single data frame"
```

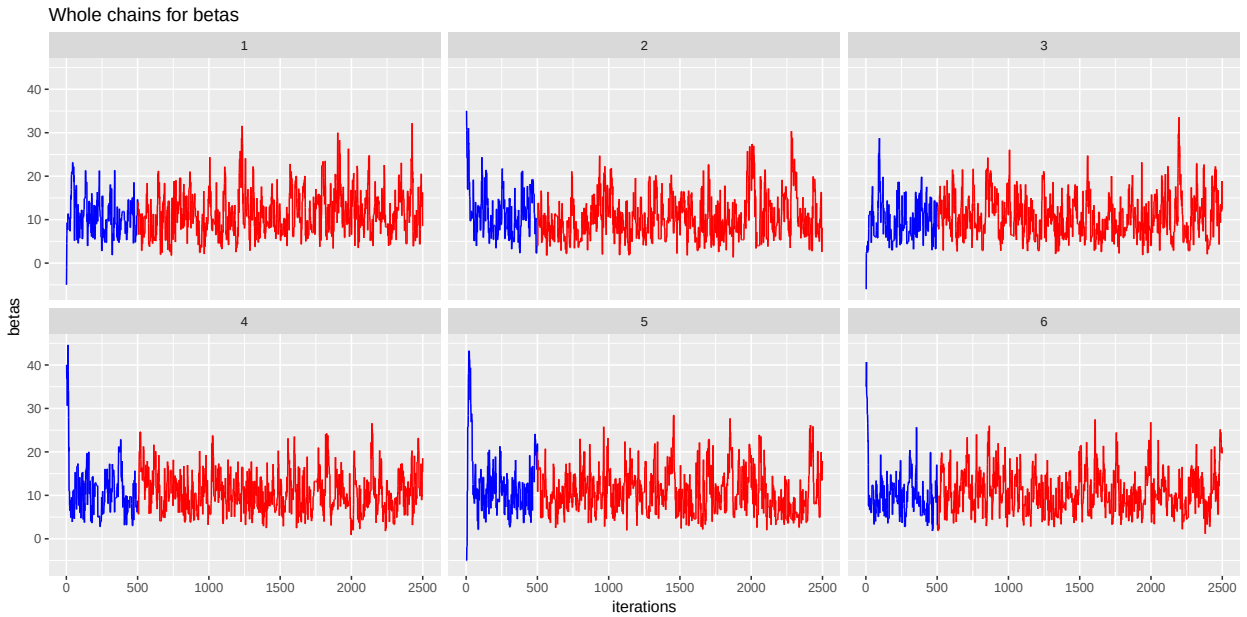
alphas	betas	chain	state	iterations
-4.0000000	-5.0000000	1	blue	1
-3.9168701	-0.9543914	1	blue	2
-1.4920002	4.5245382	1	blue	3
0.2390004	6.6493290	1	blue	4
0.4314110	9.3141294	1	blue	5
0.4314110	9.3141294	1	blue	6

Plots for the chains. Blue for warmup, red for the proper samples (the ones used for Rhat later on):

```
dat <- rowbind(res)
ggplot(data=dat) +
  geom_line(aes(x=iterations, y=alphas), color=dat$state) +
  facet_wrap(~chain) +
  ggtitle('Whole chains for alphas')
```




```
ggplot(data=dat) +
  geom_line(aes(x=iterations, y=betas), color=dat$state) +
  facet_wrap(~chain) +
  ggtitle('Whole chains for betas')
```



Interpretation: Purely based on visual inspection, the chains *seem to converge* after the warmup (first 500 iterations): The deviations are around same values, and of similar magnitude (per parameter of course -

alpha around ~1 and beta around ~10). Based on the graphs both alpha and beta should have low Rhat values and effective sample sizes (which remains to be seen).

Exercise 3.

Use $\hat{R}$ for convergence analysis.

Rhat is a convergence diagnostic, which compares the between- and within -chain estimates for parameters (here alpha, beta). If the different chains do not converge (i.e., the sampling has not sufficiently approached the "correct"- or steady-state) the value of Rhat is going to be clearly above 1 (typically > 1.05).

I will be using rstan::Rhat, which is defined in: *Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2019). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC. arXiv preprint arXiv:1903.08008.*

In short, the idea is to use rank normalized parameters (here alpha, beta) instead of the raw values, and instead of original number of chains (here 6) each chain is *split* and therefore we use the double the amount of chains.

In the function description, rstan::Rhat specifies a recommended limit value of 1.05 for Rhat, and instead of typical minimum effective sample size of 400, it recommends at least 100 if one is using *bulk effective sample size*. I will calculate for both parameters alpha and beta not just Rhat values, but also the bulk effective sample sizes. This way I should be in the safe zone regarding my inference.

Here are the calculations:

```
R_hat_alpha <- Rhat(colbind(res, 'alphas')[-(1:500),] %>% as.matrix())
R_hat_beta <- Rhat(colbind(res, 'betas')[-(1:500),] %>% as.matrix())

ess_alpha <- ess_bulk(colbind(res, 'alphas')[-(1:500),] %>% as.matrix())

ess_beta <- ess_bulk(colbind(res, 'betas')[-(1:500),] %>% as.matrix())

print(paste("Rhat for alphas (rstan implementation): ", R_hat_alpha %>% round(.,digits = 5)))
```

```
## [1] "Rhat for alphas (rstan implementation):  1.00829"
```

```
print(paste("Rhat for betas (rstan implementation): ", R_hat_beta %>% round(.,digits = 5)))
```

```
## [1] "Rhat for betas (rstan implementation):  1.00642"
```

```
print(paste("effective sample size (bulk) for alphas: ", ess_alpha %>% as.integer()))
```

```
## [1] "effective sample size (bulk) for alphas:  902"
```

```
print(paste("effective sample size (bulk) for betas: ", ess_beta %>% as.integer()))
```

```
## [1] "effective sample size (bulk) for betas:  894"
```

Results: Based on small values of Rhat for both alpha and beta (`1.00829`, `1.00642` respectively) and relatively large bulk effective sample sizes (`902`, `894` respectively), the chains seem to converge.

(**NOTE**: in all the Rhat and S_eff calculations, warmup of size 500 was removed).

Exercise 4.

Plot the draws for α and β (scatter plot).

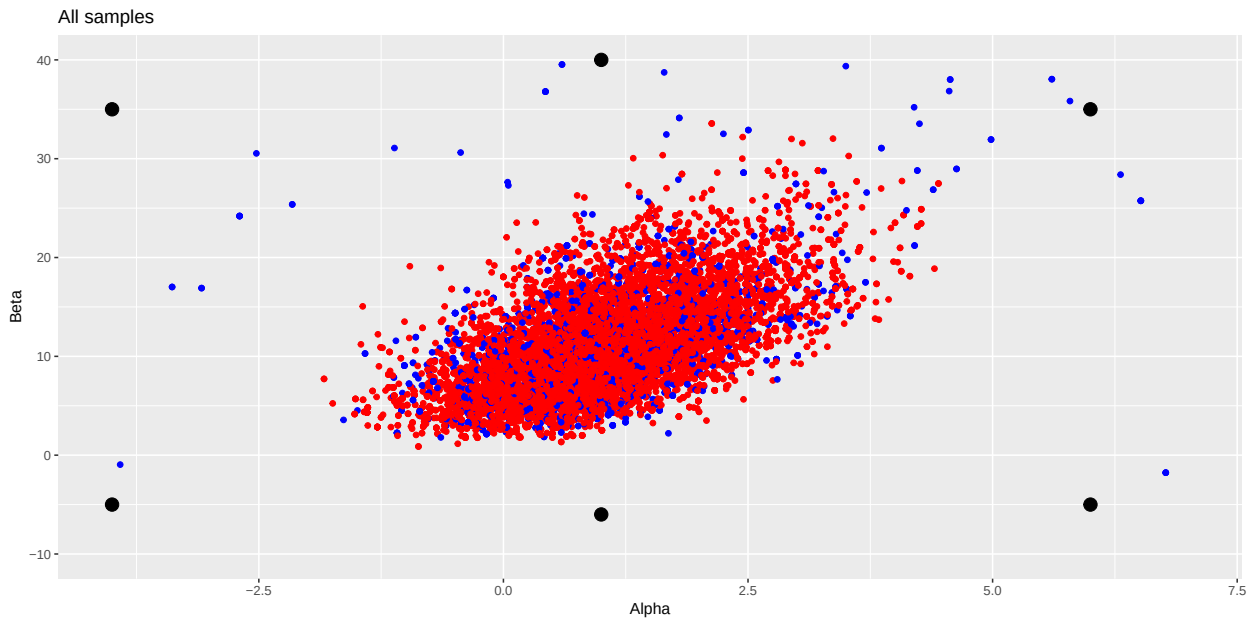
Here is a simple plotting utility:

```
quick_scatter <- function(input, all = T, initial_alpha = c(-4,-4,1,1,6,6), initial_beta = c(-5,35,-6,4
  input <- if(is.list(input)) rowbind(input) else input
  label <- if(all) 'All samples' else 'Warm samples (shows also starting points)'
  input <- if(all) input else input %>% filter(state=='red')

  ggplot() +
    geom_point(aes(x=input$alphas, y=input$betas), color=input$state) +
    ylim(c(-10,40)) +
    xlim(c(-4,7)) +
    xlab('Alpha') +
    ylab('Beta') +
    geom_point(aes(x=initial_alpha, y=initial_beta), color='black', size=4) +
    ggtitle(label)
}
```

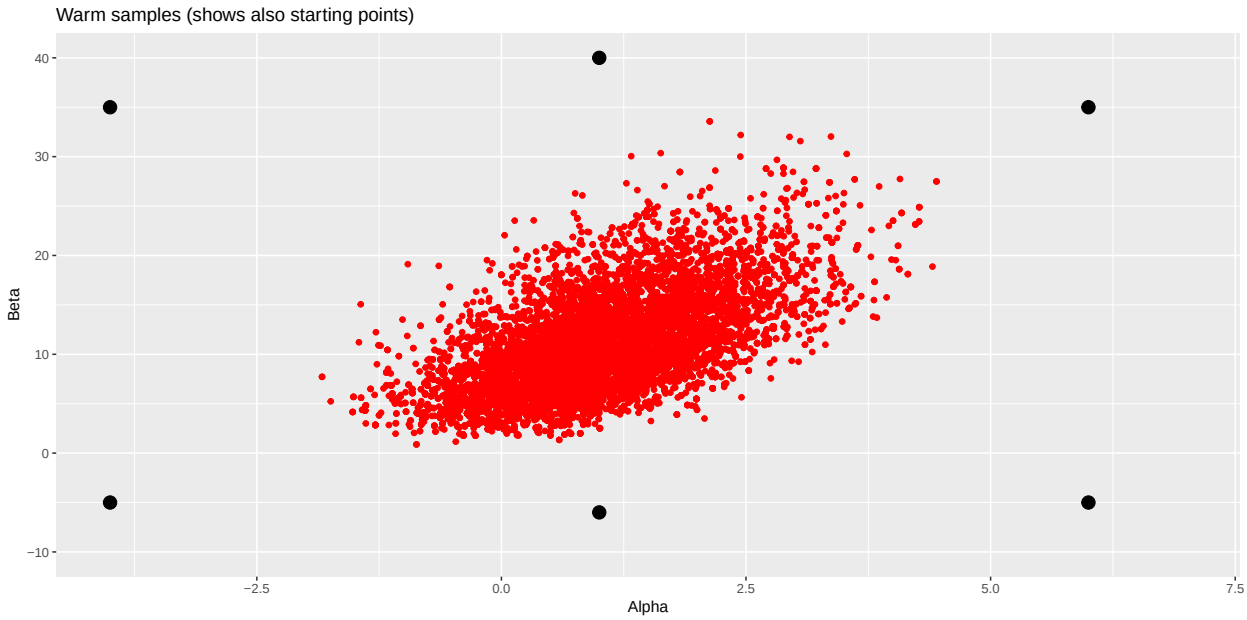
And here is the plot for the main run (500 + 2000 samples, 6 chains). Blues are warmup, reds are the rest, blacks are the starting points:

```
quick_scatter(res, T)
```



Here is the same plot without the warmup (still shows the starting points though):

```
quick_scatter(res,F)
```



The **result** seems to be a reasonable sample, i.e., no reason to suspect the sampling (which was concluded with the Rhat already).