BDA - Assignment 5

Anonymous

Contents

```
# To install aaltobda, see the General information in the assignment.
remotes::install_github("avehtari/BDA_course_Aalto", subdir = "rpackage", upgrade = "never")
```

```
## Skipping install of 'aaltobda' from a github remote, the SHA1 (38f34d35) has not changed since last i
##   Use `force = TRUE` to force installation
```

```
library(aaltobda)
```

Generalized linar model: Bioassay with Metropolis

1

The code below uses the Gaussian prior as described in assignment 4.

```
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)
```

a)

```
data("bioassay")

density_ratio <- function(alpha_propose, alpha_previous,
                          beta_propose, beta_previous,
                          x, y, n) {
  a1 <- alpha_propose
  a0 <- alpha_previous
  b1 <- beta_propose
  b0 <- beta_previous

  # 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)
```

```r
}

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
```

```r
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)

My metropolis algorithm function is described below:

```r
theta0 <- rnorm(8, mean = 72, sd = 72)
theta0 <- matrix(theta0, nrow = 4, ncol = 2)
#theta0 <- rmvnorm(8, mean = mean, sigma = sigma)
maxIter <- 1000
theta_df <- data.frame()
for(c in 1:4) {
  alpha0 <- theta0[c,1]
  beta0  <- theta0[c,2]
  for(i in 1:maxIter) {
    if(i == 1) {
      alpha_previous <- alpha0
      beta_previous  <- beta0
    }

    alpha_propose <- rnorm(1, mean = alpha_previous, sd = 1)
    beta_propose  <- rnorm(1, mean = beta_previous,  sd = 5)

    r <- density_ratio(alpha_propose  = alpha_propose,
                       alpha_previous = alpha_previous,
                       beta_propose   = beta_propose,
                       beta_previous  = beta_previous,
                       x = bioassay$x,
                       y = bioassay$y,
                       n = bioassay$n)
    if(r > 1) {
      alpha_previous <- alpha_propose
      beta_previous  <- beta_propose
    }else{
        u <- runif(1, min = 0, max = 1)
        if(u <= r) {
          alpha_previous <- alpha_propose
          beta_previous  <- beta_propose
        }
      }
    theta_df <- rbind.data.frame(theta_df, c(c, i, alpha_previous, beta_previous, r))
  }
}
names(theta_df) <- c("chain", "t", "alpha", "beta", "r")
chain1 <- theta_df[theta_df$chain == 1,]
chain2 <- theta_df[theta_df$chain == 2,]
chain3 <- theta_df[theta_df$chain == 3,]
chain4 <- theta_df[theta_df$chain == 4,]
```

2)

a)

The idea of the Metropolis algorithm is to compare probabilities of a proposed parameter as it is derived from a previous parameter. The Metropolis algorithm always points (chooses) the direction of relative higher probability as seen in the ratio of density. It selects a new parameter as long as it is higher in probability than the previous one. A centra idea in MCMC algorithms is that the propose parameter depends on the previous parameter. In our case, we are using this jumping function with a normal distribution.

b)

My jumping rule, proposal distributions, are $theta^* \sim N(theta^{t-1}, \Sigma)$ where the diagonals in sigma is the vector $[1, 5]$, while the diagonal off elements are zero (no correlation across the variables). That is, letting aside the starting values, a proposal value jumps from the prior value based on the normal distribution).

c)

While I have seen in practice that people use random numbers for their initial values, such practice caused me problems in my applications of truncated log-likelihood maximization with gradient descent algorithms. For this case, I actually employ the random matrix below:

```
theta0
```

```
##              [,1]      [,2]
## [1,]  50.7113121  40.98990
## [2,]  -0.3568152 163.43435
## [3,] -24.6835833  30.68713
## [4,]  49.9090342 135.09642
```

d)

Since my Metropolis algorithm does not seem to be very laborious, I used four Markov chains with 1000 of length.

e)

My warm-up discards the first halves of each of my four chains.

f)

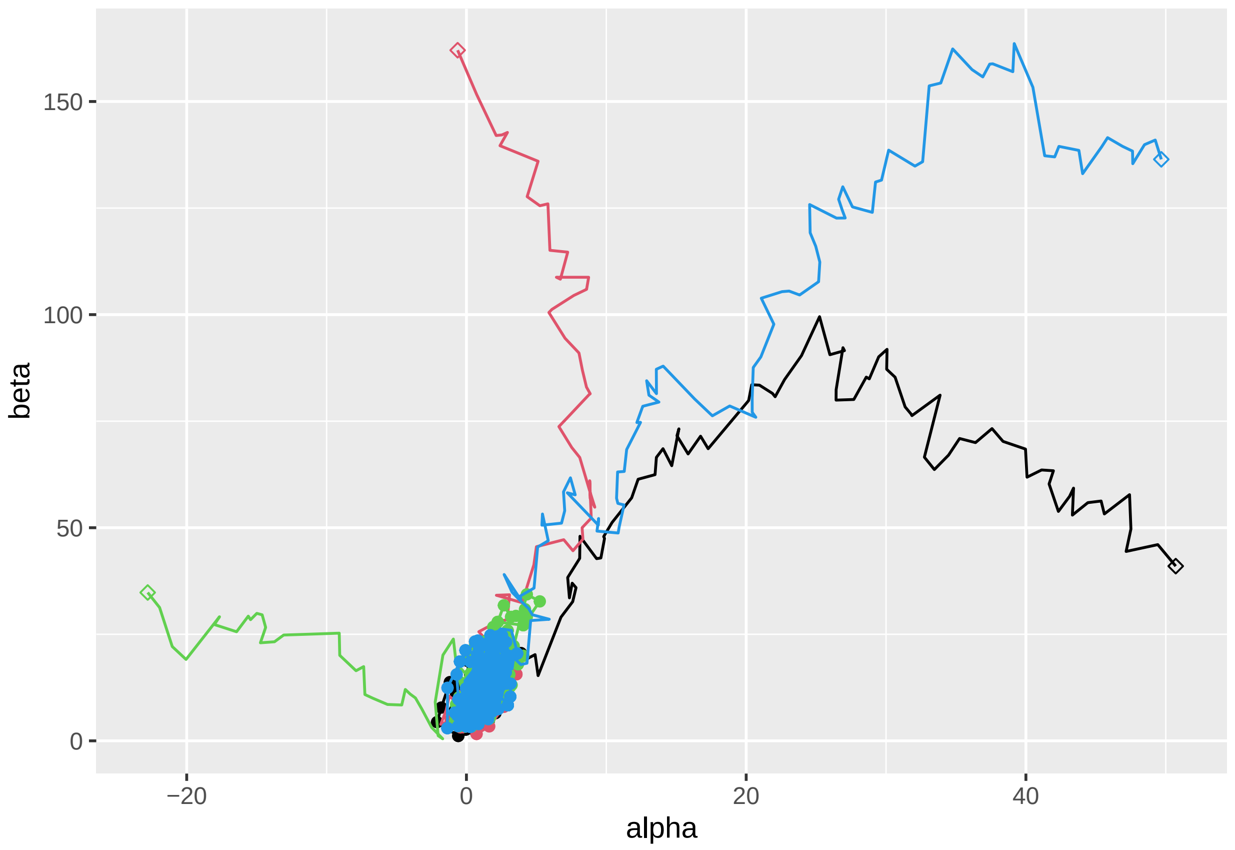
I employed four Markov chains.

g) and h)

Below I assess simultanously the convergence of α and β

```r
warmuprate <- .5
library(ggplot2)

ggplot() +
  geom_path(chain1,
            mapping = aes(x = alpha, y = beta),
            color = '1') +
  geom_point(chain1[chain1$t == 1,],
             mapping = aes(x = alpha, y = beta),
             color = '1', shape = 5) +
  geom_point(chain1[chain1$t > length(chain1$t)*warmuprate,],
             mapping = aes(x = alpha, y = beta),
             color = '1') +
  geom_path(chain2,
            mapping = aes(x = alpha, y = beta),
            color = '2') +
  geom_point(chain2[chain2$t == 1,],
             mapping = aes(x = alpha, y = beta),
             color = '2', shape = 5) +
  geom_point(chain2[chain2$t > length(chain2$t)*warmuprate,],
             mapping = aes(x = alpha, y = beta),
             color = '2') +
  geom_path(chain3,
            mapping = aes(x = alpha, y = beta),
            color = '3') +
  geom_point(chain3[chain3$t == 1,],
             mapping = aes(x = alpha, y = beta),
             color = '3', shape = 5) +
  geom_point(chain3[chain3$t > length(chain3$t)*warmuprate,],
             mapping = aes(x = alpha, y = beta),
             color = '3') +
  geom_path(chain4,
            mapping = aes(x = alpha, y = beta),
            color = '4') +
  geom_point(chain4[chain4$t == 1,],
             mapping = aes(x = alpha, y = beta),
             color = '4', shape = 5) +
  geom_point(chain4[chain4$t > length(chain4$t)*warmuprate,],
             mapping = aes(x = alpha, y = beta),
             color = '4')
```



3)

I am using Eq 11.4, and I show my code below:

```
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## rstan (Version 2.21.3, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
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)
Rhat(alpha_sims)
```

```
## [1] 1.049235
```

```r
beta_sims <- cbind.data.frame(chain1Warmed$beta,
                              chain2Warmed$beta,
                              chain3Warmed$beta,
                              chain4Warmed$beta)
beta_sims <- as.matrix(beta_sims)
Rhat(beta_sims)
```

```
## [1] 1.033329
```

a)

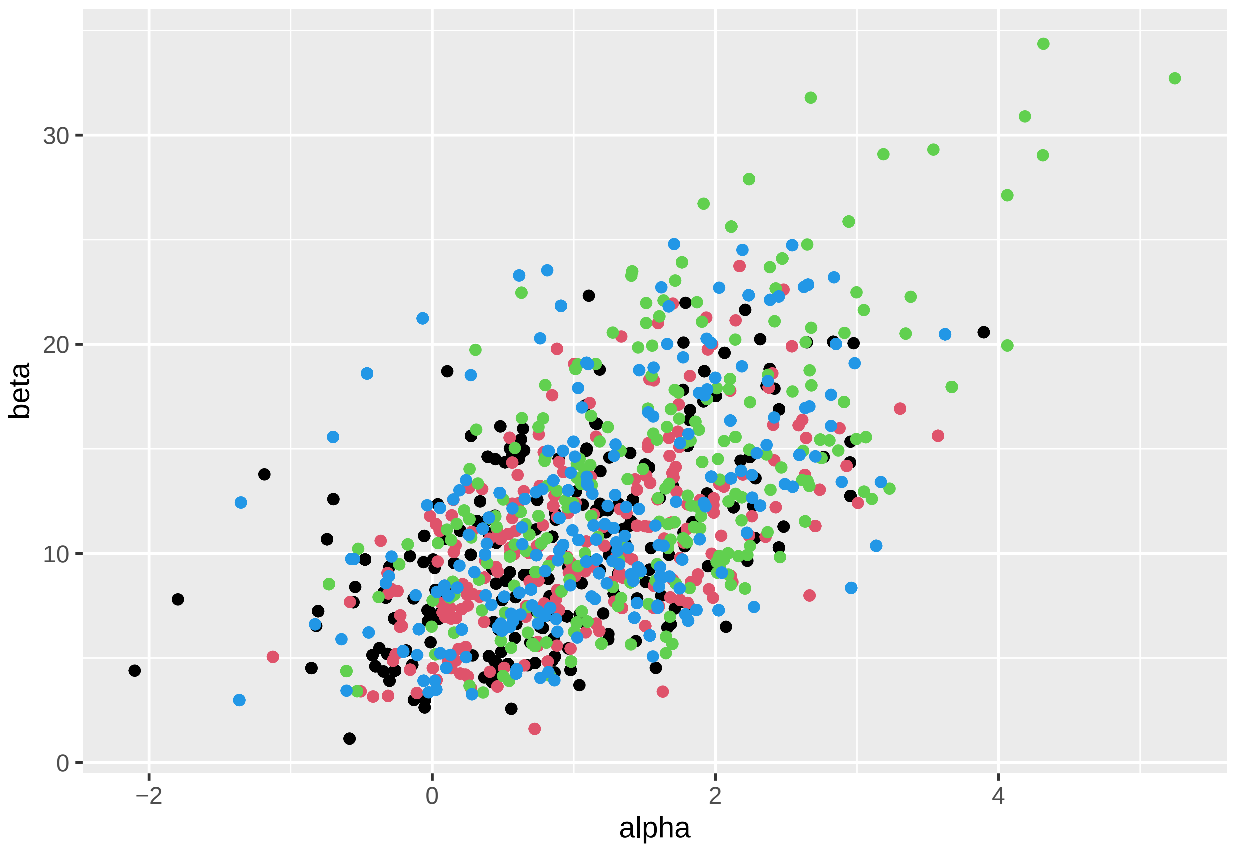
b)

4

Below is my scatter plot of the points that exclude the warm-up.

```r
warmuprate <- .5

ggplot() +
  geom_point(chain1[chain1$t > length(chain1$t)*warmuprate,],
             mapping = aes(x = alpha, y = beta),
             color = '1') +
  geom_point(chain2[chain2$t > length(chain2$t)*warmuprate,],
             mapping = aes(x = alpha, y = beta),
             color = '2') +
  geom_point(chain3[chain3$t > length(chain3$t)*warmuprate,],
             mapping = aes(x = alpha, y = beta),
             color = '3') +
  geom_point(chain4[chain4$t > length(chain4$t)*warmuprate,],
             mapping = aes(x = alpha, y = beta),
             color = '4')
```



beta
30
20
10
0
−2
0
2
4
alpha