Biostat276 Project 1

Zhaodong Wu

1/17/2022

Contents


```
rm(list = ls())
library(ggplot2)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v tibble  3.1.0     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

1. Sampling from the Banana Distribution

The two dimensional Banana distribution (Haario, 1999) is defined as:

$$\pi(x_1, x_2) \propto \exp(-\frac{x_1^2}{2}) \exp\left[-\frac{\{x_2 - 2(x_1^2 - 5)\}^2}{2}\right]; \quad x_1, x_2 \in \mathbb{R}.$$

(a) Describe how to simulate directly from $\pi(x_1, x_2)$.

Solution:

From the information of joint probability-density function, we can know that the conditional probability of x_2 given x_1 is proportional to a normal kernel,

$$\pi(x_2|x_1) = \frac{\pi(x_1, x_2)}{\pi(x_1)} \propto \exp\left[-\frac{\{x_2 - 2(x_1^2 - 5)\}^2}{2}\right]$$

Therefore we can have

$$\pi(x_2|x_1) = c * \exp\left[-\frac{\{x_2 - 2(x_1^2 - 5)\}^2}{2}\right], \quad \text{where } c = \frac{1}{\sqrt{2\pi}}$$

Then we can get the marginal distribution of x_1 is also proportional to a normal kernel:

$$\begin{aligned}
\pi(x_1) = \int_{-\infty}^{+\infty} \pi(x_1, x_2) dx_2 \quad &\propto \quad \int_{-\infty}^{+\infty} \exp(-\frac{x_1^2}{2}) \exp\left[-\frac{\{x_2 - 2(x_1^2 - 5)\}^2}{2}\right] dx_2 \\
&= \quad \exp(-\frac{x_1^2}{2}) \int_{-\infty}^{+\infty} \exp\left[-\frac{\{x_2 - 2(x_1^2 - 5)\}^2}{2}\right] dx_2 \\
&= \quad \sqrt{2\pi} \cdot \exp(-\frac{x_1^2}{2}) \\
&\propto \quad \exp(-\frac{x_1^2}{2})
\end{aligned}$$

Therefore we can get x_1 follows a standard normal distribution:

$$\pi(x_1) = \frac{1}{\sqrt{2\pi}} \cdot \exp(-\frac{x_1^2}{2})$$

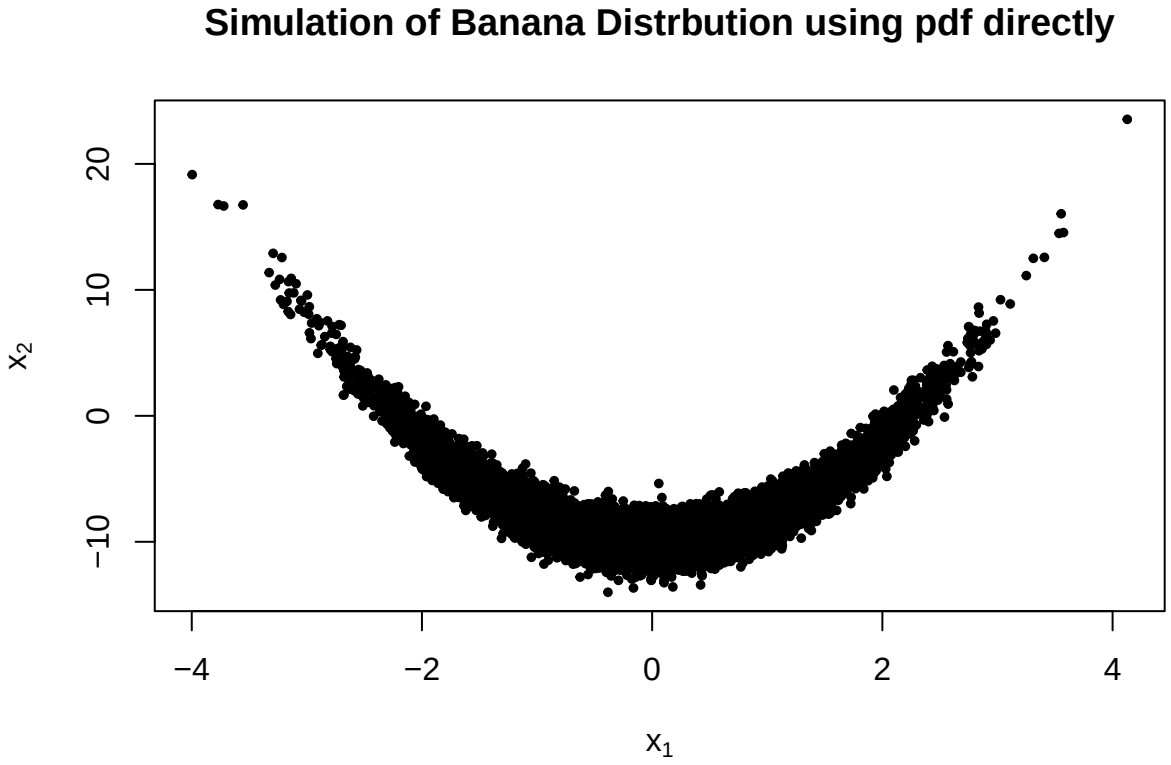
We can use inverse-transformation method to do the simulation:

We generate a R.V. $u \sim Uniform(0, 1)$ and take $x_1 = \Phi^{-1}(u)$;

Then we generate another R.V. $v \sim Uniform(0, 1)$ and take $x_2 = \Phi^{-1}(v) + 2(x_1^2 - 5)$ since $x_2 - 2(x_1^2 - 5)|x_1 \sim N(0, 1)$.

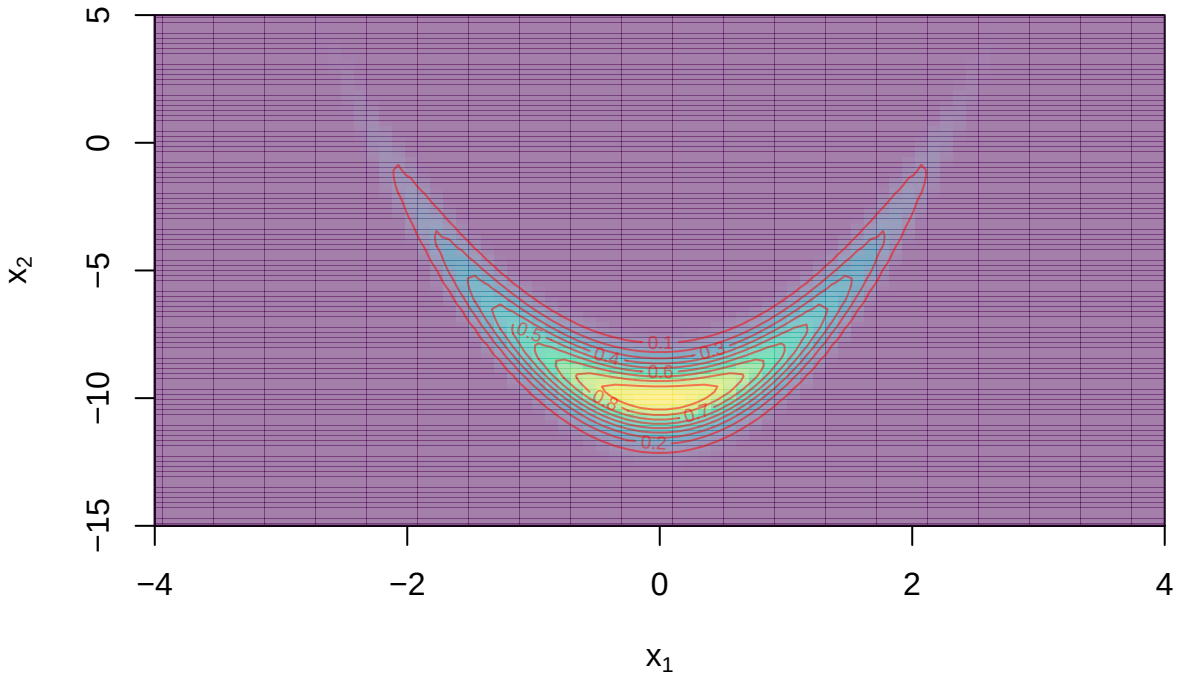
The simulation of standard normal distribution I used is shown below: Resource of standard normal simulation

```
nsim_Q1 <- 15000
#Box-Muller transform
box_muller <- function(n = 1, mean = 0, sd = 1){
  x <- vector("numeric", n)
  i <- 1
  while(i <= n){
    u1 <- runif(1, 0, 1)
    u2 <- runif(1, 0, 1)
    x[i] <- sqrt(-2 * log(u1)) * cos(2 * pi * u2)
    if ((i + 1) <= n){
      x[i + 1] <- sqrt(-2 * log(u1)) * sin(2 * pi * u2)
      i <- i + 1
    }

   i <- i + 1
  }
  x * sd + mean
}
set.seed(1998)
x1_Q1 <- box_muller(nsim_Q1)
x2_Q1 <- box_muller(nsim_Q1) + 2 * (x1_Q1^2 - 5)

plot(x1_Q1, x2_Q1, pch = 20, cex = 0.8,
     xlab = expression(x[1]), ylab = expression(x[2]),
     main = "Simulation of Banana Distrbution using pdf directly")
```



The figure below is just a banana distribution plot of using fixed x_1, x_2, not related to questions.

```r
# generate x1, x2
nsim_Q1 = 100
x1 <- seq(-5, 5, length.out = nsim_Q1)
x2 <- seq(-15, 5, length.out = nsim_Q1)
mat <- matrix(0, nsim_Q1, nsim_Q1)

# define banana distribution
banana_Q1 <- function(x1, x2) {
  fx1x2 <- exp(-(x1^2)/2)*exp(-((x2-2*(x1^2-5))^2)/2)
  return(fx1x2)
}

# plug in values
for (i in 1:nsim_Q1) {
  for (j in 1:nsim_Q1) {
    x = x1[i]
    y = x2[j]
    mat[i,j] = banana_Q1(x,y)
  }
}

# plot contour image
image(x1, x2, mat, col = hcl.colors(100, alpha = 0.5), xlim = c(-4, 4),
     ylim = c(-15, 5),
      xlab = expression(x[1]),
      ylab = expression(x[2]))
contour(x1, x2, mat, add = T, col = rgb(1, 0, 0, alpha = 0.5))
```



(b) Describe how to simulate from $\pi(x_1, x_2)$ using the accept-reject method. Implement your algorithm and discuss sampling efficiency in relation to the chosen instrument distribution.

Solution: For the accept-reject method we need to evaluate the normalizing constant M such that:

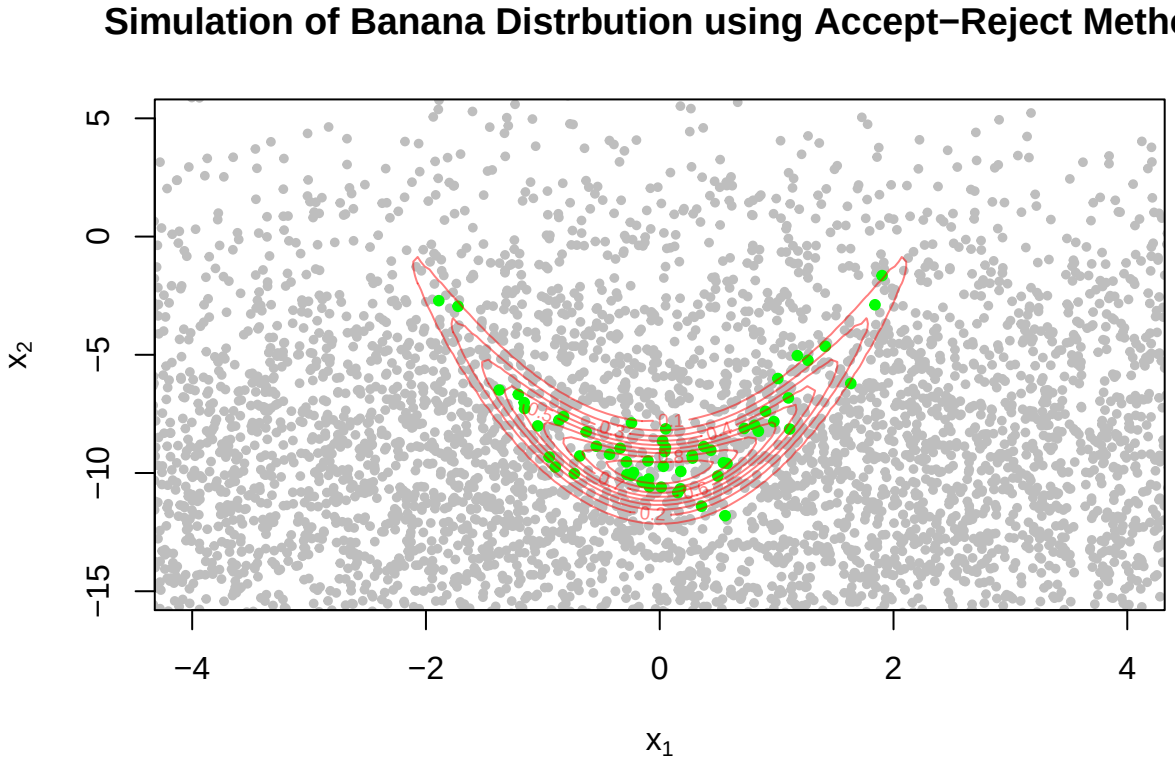
$$\pi(x_1, x_2) \leq M \cdot g(x_1, x_2), \quad \forall x_1, x_2 \in \mathbb{R}$$

```
#define banana distribution again for simplicity of Q2
banana_Q2 <- function(x) {
  x1 = x[1]
  x2 = x[2]
  fx1x2 <- exp(-x1^2/2)*exp(-(x2-2*(x1^2-5))^2/2)
}

library(mvtnorm)
acc_rej <- function(nsim, M, sd = 40, seed_Q2 = 1996) {
  set.seed(seed_Q2)
  # generate g(x)- a multivariate t distribution
  x <- rmvt(n = nsim, sigma = diag(sd, 2), df = 6, delta = c(0, -10),
            type = "shifted")
  # generate u ~ U[0, 1]
  u <- runif(nsim)
  Accept <- vector()
  fx <- vector()
  g <- vector()
  for (i in 1:nsim) {
    gx = dmvt(x = x[i, ], sigma = diag(sd, 2), df = 6, delta = c(0, -10),
              log = FALSE, type = "shifted")
    # check if the condition that f(x) <= M*g(x) is satisfied
    if (banana_Q2(x[i, ]) > M*gx) {
      print(i)
      print(x[i, ])
      print(banana_Q2(x[i, ]))
```

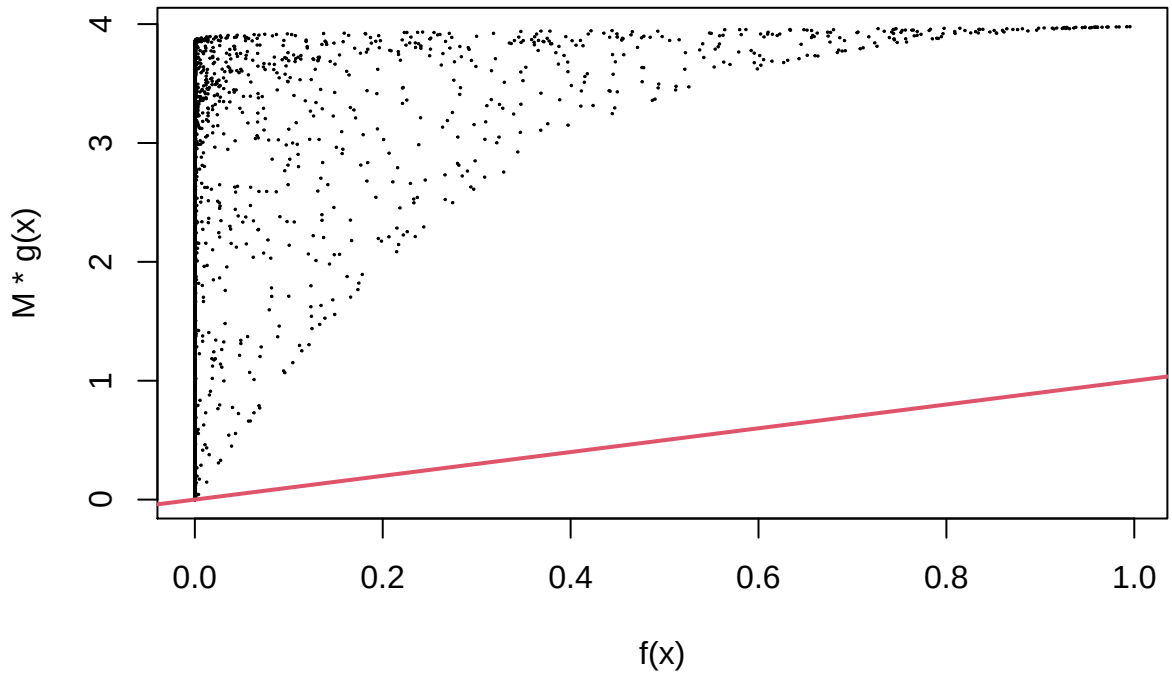
```
      print(M*gx)
      stop("The condition fails")
    }
    # accept y = x if u <= f(x)/(M*g(x))
    if (u[i] <= banana_Q2(x[i, ])/(M*gx)) {
      Accept[i] <- TRUE
      fx[i] <- banana_Q2(x[i, ])
      g[i] <- gx
    } else {
      Accept[i] <- FALSE
      fx[i] <- banana_Q2(x[i, ])
      g[i] <- gx
    }
  }
  return(list(x1 = x[, 1], x2 = x[, 2], Accept = Accept, fx = fx, g = g))
}
Q2 <- acc_rej(10000, 1000)
# accuracy
sum(Q2$Accept)/length(Q2$Accept) # accuracy is low, about 0.005
```

```
## [1] 0.0059
```

```
# plot
plot(Q2$x1, Q2$x2, xlim = c(-4, 4), pch = 20, cex = 0.8, col = "grey",
     ylim = c(-15, 5), xlab = expression(x[1]), ylab = expression(x[2]),
     main = "Simulation of Banana Distrbution using Accept-Reject Method")
points(Q2$x1[Q2$Accept == "TRUE"], Q2$x2[Q2$Accept == "TRUE"],
       pch = 20, col = "green")
contour(x1, x2, mat, add = T, col = rgb(1, 0, 0, alpha = 0.5))
```



```r
# decision plot
plot(Q2$fx, 1000*Q2$g, cex = 0.1, xlab = "f(x)", ylab = "M * g(x)")
abline(a = 0, b = 1, col = 2, lwd = 2)
```

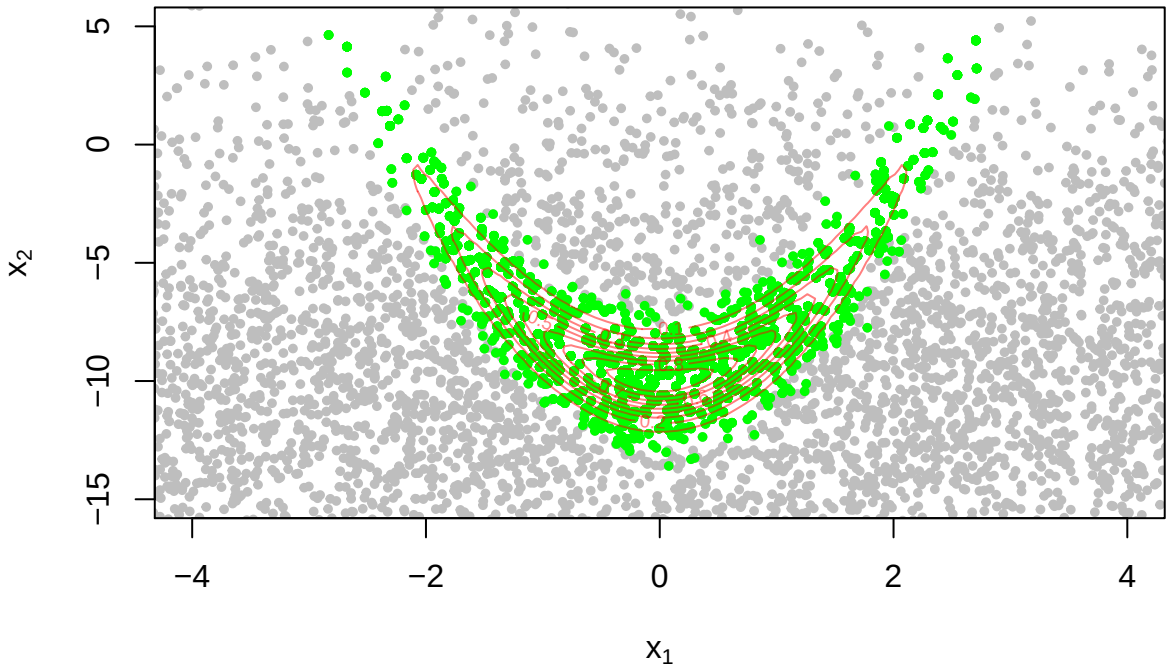


According to the result, the accuracy is 0.005 if we use a multivariate t distribution with a high standard deviation.

(c) Describe how to simulate from $\pi(x_1, x_2)$ using sampling importance re-sampling. Implement your algorithm and discuss sampling efficiency in relation to the chosen importance distribution.

For importance sampling, we can use x-values satisfying a multivariate t distrbution in the previous problem and calculate the weight using f(x) and g(x).

```r
plot(Q2$x1, Q2$x2, pch = 20, cex = 0.8, col = "grey", xlim = c(-4, 4),
     ylim = c(-15, 5),
     xlab = expression(x[1]), ylab = expression(x[2]),
     main = "Simulation of Banana Distrbution using Importance-Sampling Method")
w_Q1 <- Q2$fx/Q2$g
w_Q1star = w_Q1/sum(w_Q1)
samp_ind_Q3 = seq(1, 10000)
x_ind = sample(samp_ind_Q3, prob = w_Q1star, replace = TRUE)
x1_IS <- Q2$x1[x_ind]
x2_IS <- Q2$x2[x_ind]
points(x1_IS, x2_IS, pch = 20, cex = 0.8, col = "green")
contour(x1, x2, mat, add = T, col = rgb(1, 0, 0, alpha = 0.5))
```



We can see the efficiency of importance sampling is nice.

(d) Construct a Markov Chain to sample from $\pi(x_1, x_2)$ using the Metropolis Hastings algorithm. Compare convergence and mixing for different proposal jump sizes.

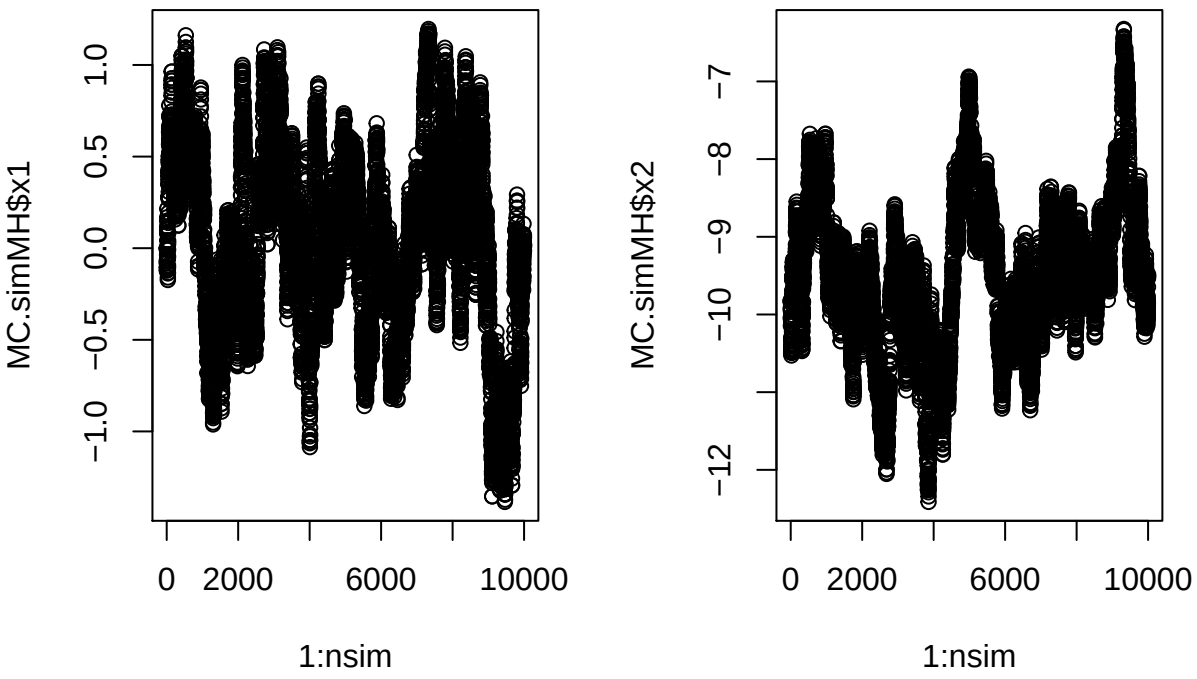
Random walk proposal:

$$x_1^*|x_1 \sim N(x_1, \sigma), \quad x_2^*|x_2 \sim N(x_2, \eta)$$

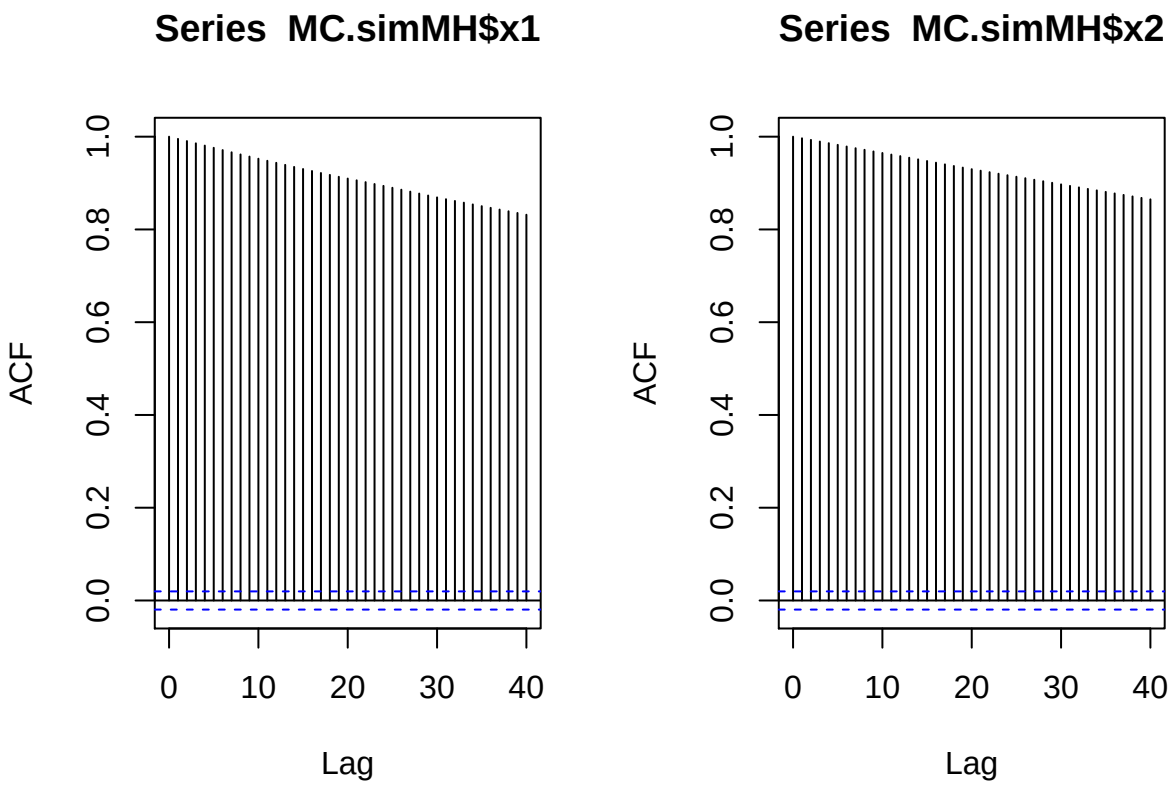
```
mh.Q4 <- function(nsim = 10000, burn = 0.1, # chain parameters
                  sigma = 2, eta = 3, # MH proposal step size
                  seed = 1998) {
  set.seed(seed)
  x1.ch <- vector()
  x2.ch <- vector()
  x1 <- 0
  x2 <- -10
  # run chain
  nsim1 <- nsim*(1 + burn)
  burni <- nsim*burn

  # update x1, x2
  for (isim in 1:nsim1) {
    x1.t <- rnorm(1, x1, sigma)
    x2.t <- rnorm(1, x2, eta)
    P0 <- banana_Q1(x1, x2) %>% log
    P1 <- banana_Q1(x1.t, x2.t) %>%  log
    ratio <- P1 - P0
    if (log(runif(1)) < ratio) {
      x1 <- x1.t
```

```
      x2 <- x2.t
    }
    # Store Chain after burn
    if (isim > burni) {
      i1 <- isim - burni
      x1.ch[i1] <- x1
      x2.ch[i1] <- x2

    }
  }
  return(list(x1 = x1.ch, x2 = x2.ch))
}
```

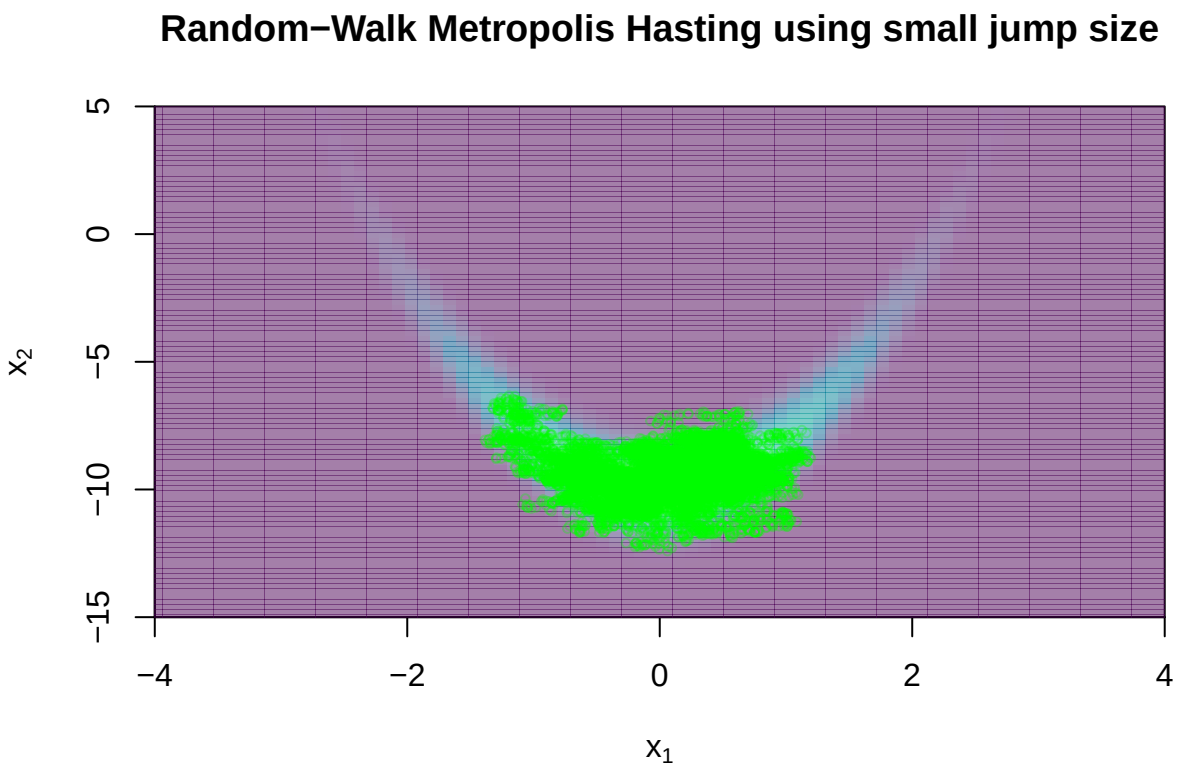
```
# simulation begin

# Condition 1: Small jump sizes
nsim = 10000
burn = 0.1
sigma <- 0.05
eta <- 0.08

MC.simMH <- mh.Q4(nsim = nsim, burn = burn, sigma = sigma, eta = eta)

par(mfrow=c(1,2))
plot(1:nsim, MC.simMH$x1)
plot(1:nsim, MC.simMH$x2)
```




```
# autocorrelation plot
par(mfrow=c(1,2))
acf(MC.simMH$x1)
acf(MC.simMH$x2)
```

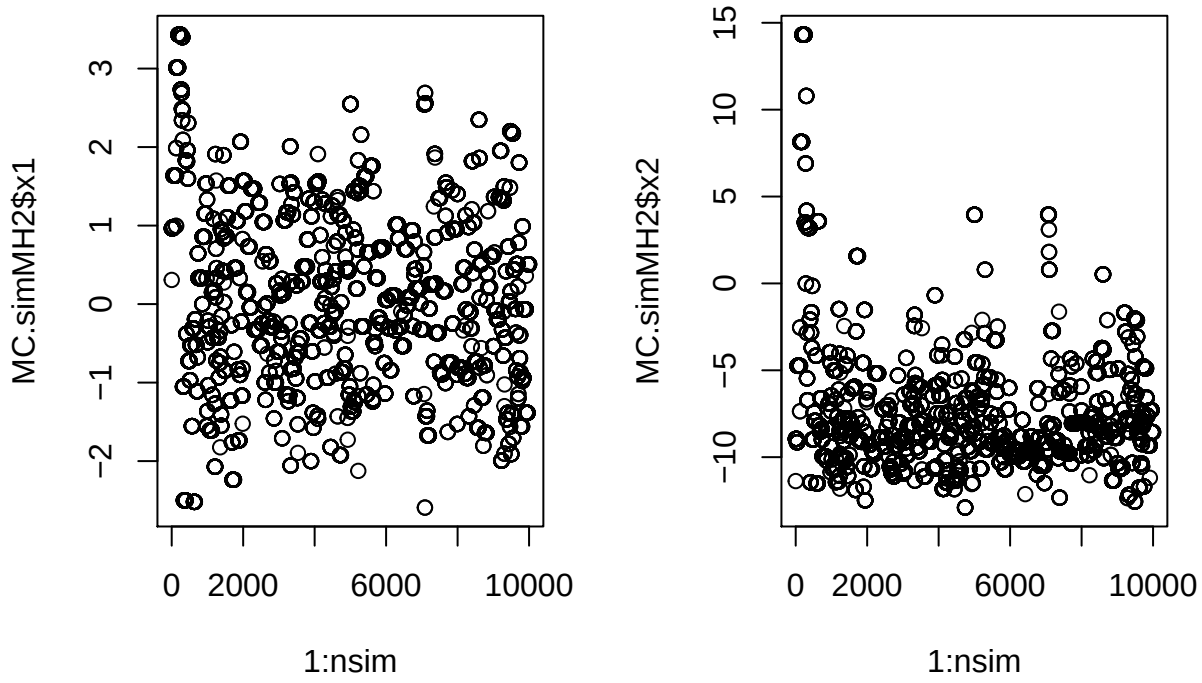



```
image(x1, x2, mat, col = hcl.colors(100, alpha = 0.5), xlim = c(-4, 4),
      ylim = c(-15, 5),
      xlab = expression(x[1]),
      ylab = expression(x[2]),
      main = "Random-Walk Metropolis Hasting using small jump size")
points(MC.simMH$x1, MC.simMH$x2, col = rgb(0,1,0,alpha = 0.3),cex = 0.6)
```

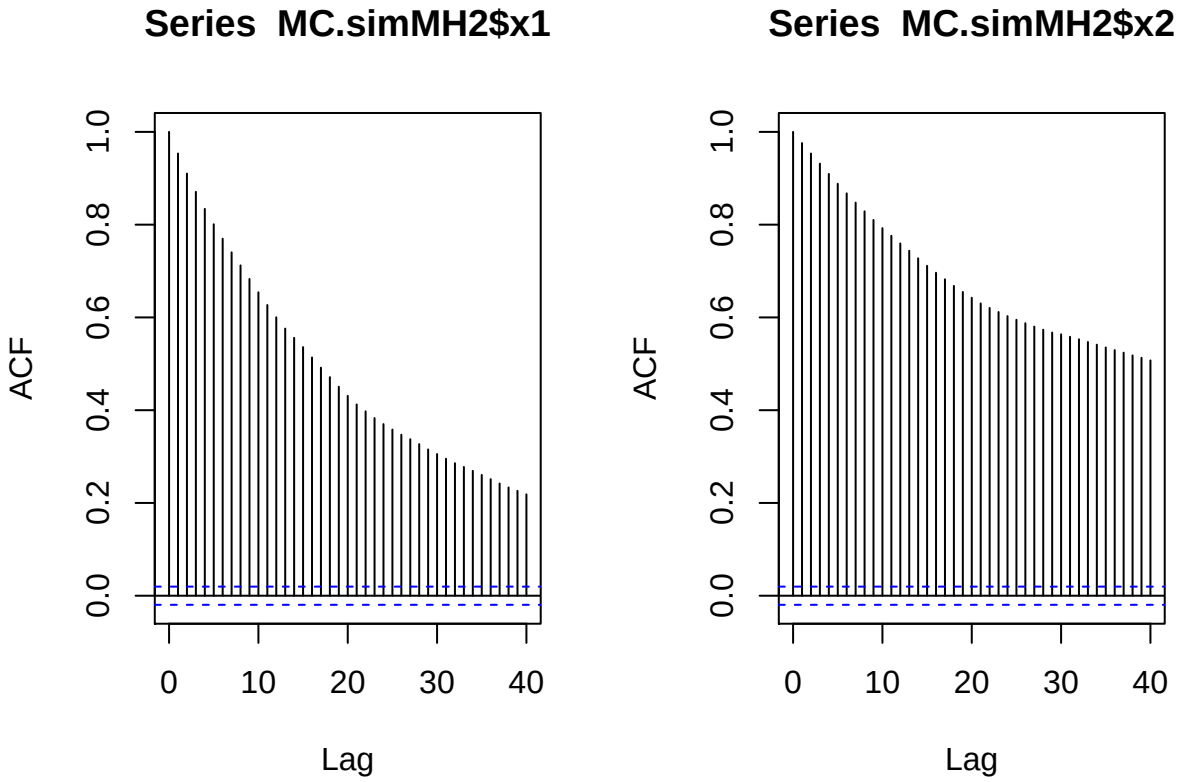


We can see simulation points cant fill the two tails of banana distrbution

```
# Condition 2: large jump sizes
nsim = 10000
burn = 0.1
sigma2 <- 5
eta2 <- 8

MC.simMH2 <- mh.Q4(nsim = nsim, burn = burn, sigma = sigma2, eta = eta2)

par(mfrow=c(1,2))
plot(1:nsim, MC.simMH2$x1)
plot(1:nsim, MC.simMH2$x2)
```

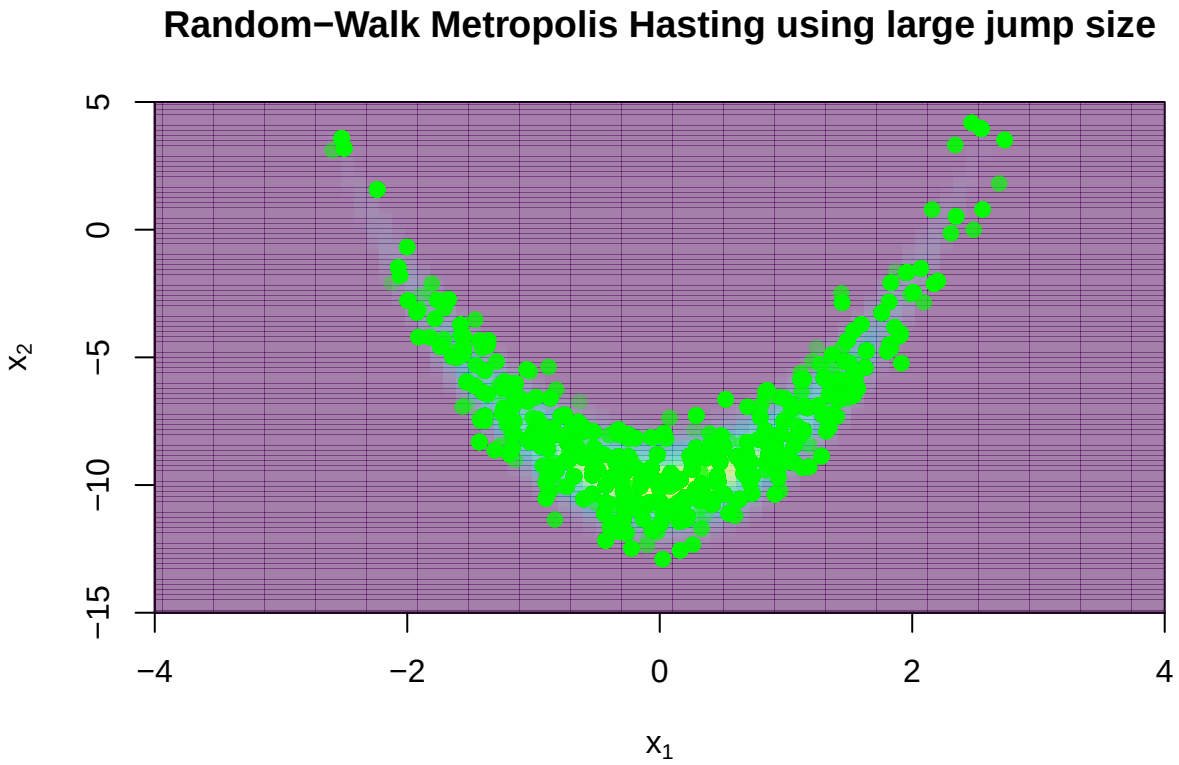



```
par(mfrow=c(1,2))
acf(MC.simMH2$x1)
acf(MC.simMH2$x2)
```




```
image(x1, x2, mat, col = hcl.colors(100, alpha = 0.5), xlim = c(-4, 4),
      ylim = c(-15, 5),
      xlab = expression(x[1]),
      ylab = expression(x[2]),
```

```
     main = "Random-Walk Metropolis Hasting using large jump size")
points(MC.simMH2$x1, MC.simMH2$x2, col = rgb(0,1,0,alpha = 0.3), pch = 19)
```



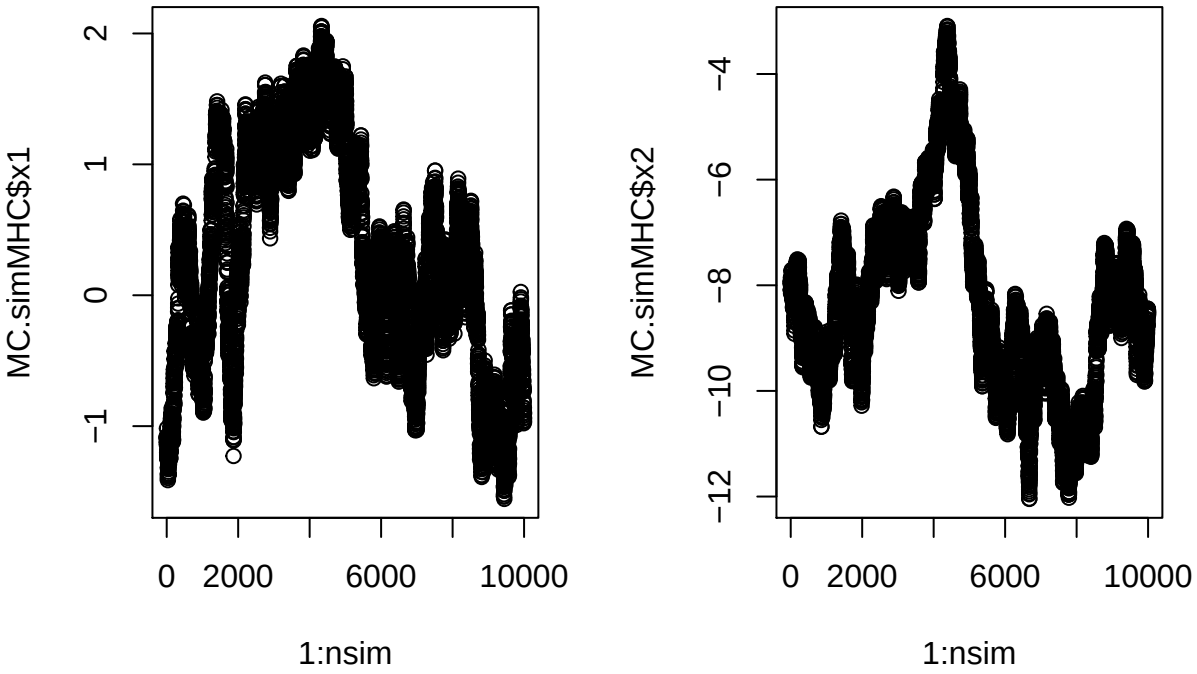
If we set a larger jump size, the convergence of x_1, x_2 becomes better and we can get more accurate values of x_1, x_2 which satisfy a banana distribution. For the mixing time, the auto-correlation becomes much smaller for x values if the jump size is small, and it's the result that we want.

(e) Construct a Markov Chain to sample from $\pi(x_1, x_2)$ updating iteratively $\pi(x_1|x_2)$ and $\pi(x_2|x_1)$ in a component- wise fashion, using the Metropolis Hastings algorithm. Compare convergence and mixing for different proposal jump sizes.

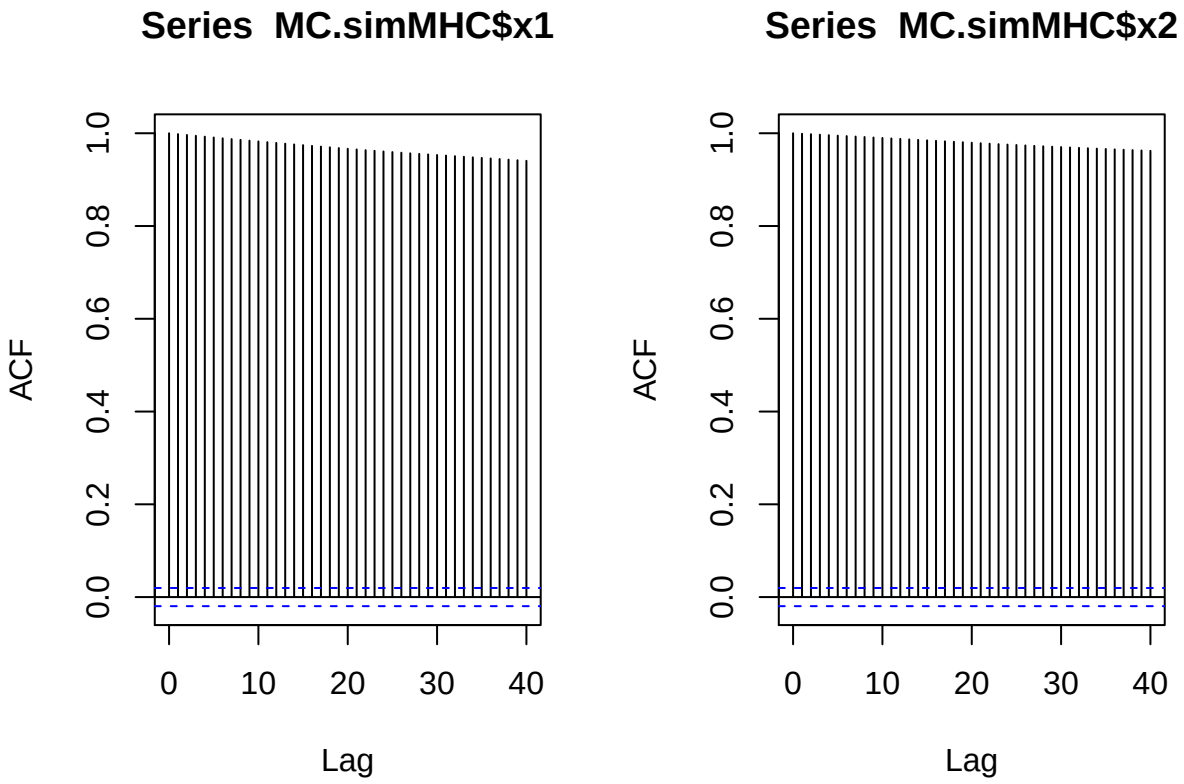
```
mh.Q5 <- function(nsim = 10000, burn = 0.1, # chain parameters
                  sigma = 2, eta = 3, # MH proposal step size
                  seed = 1998) {
  set.seed(seed)
  x1.ch <- vector()
  x2.ch <- vector()
  x1 <- 0
  x2 <- -10
  # run chain
  nsim1 <- nsim*(1 + burn)
  burni <- nsim*burn

  # x2|x1
  for (isim in 1:nsim1) {
    x1.t <- rnorm(1, x1, sigma)
    x2.t <- rnorm(1, x2, eta)
    P0 <- banana_Q1(x1, x2) %>% log
    P1 <- banana_Q1(x1, x2.t) %>%  log
    ratio <- P1 - P0
```

```
    if (log(runif(1)) < ratio) {
      x2 <- x2.t
      P0 <- P1
    }

  # x1|x2
    P1 <- banana_Q1(x1.t, x2) %>% log
    ratio <- P1 - P0

    if (log(runif(1)) < ratio) {
      x1 <- x1.t
    }
    # Store Chain after burn
    if (isim > burni) {
      i1 <- isim - burni
      x1.ch[i1] <- x1
      x2.ch[i1] <- x2

    }
  }
  return(list(x1 = x1.ch, x2 = x2.ch))
}
```

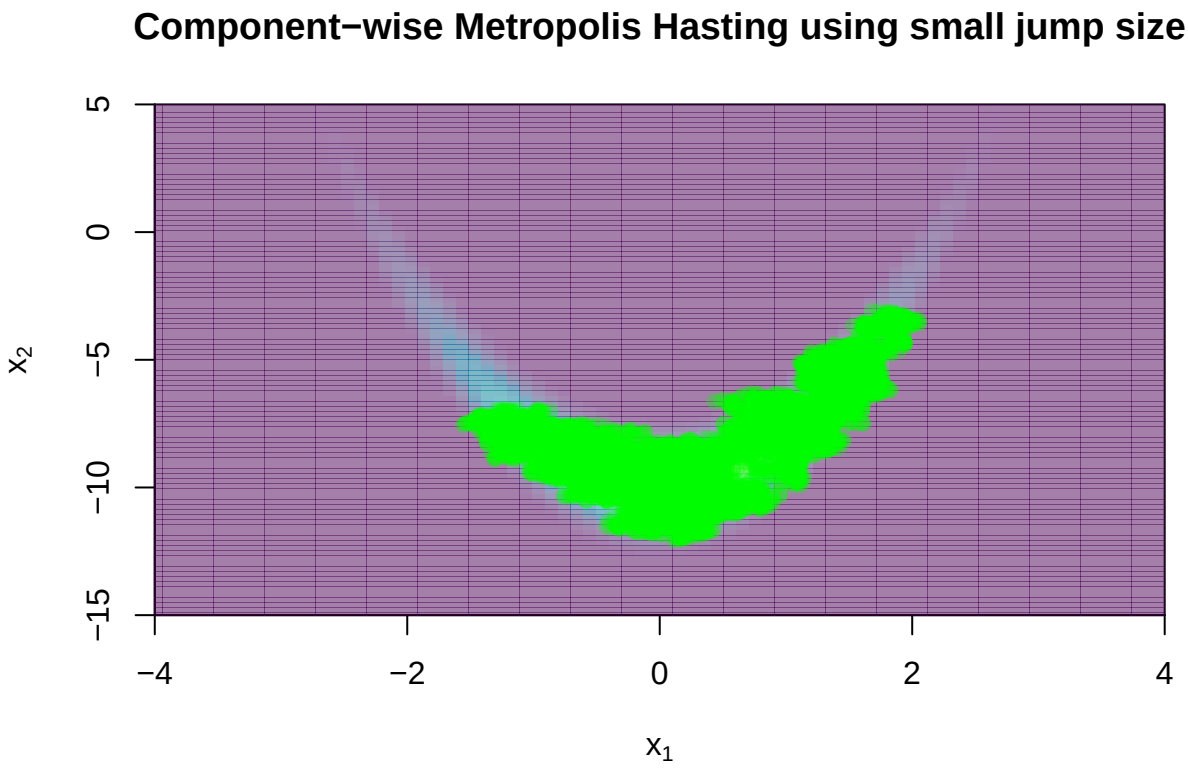
```
# Condition 1: small jump sizes

MC.simMHC <- mh.Q5(nsim = nsim, burn = burn, sigma = sigma, eta = eta)

par(mfrow=c(1,2))
plot(1:nsim, MC.simMHC$x1)
plot(1:nsim, MC.simMHC$x2)
```



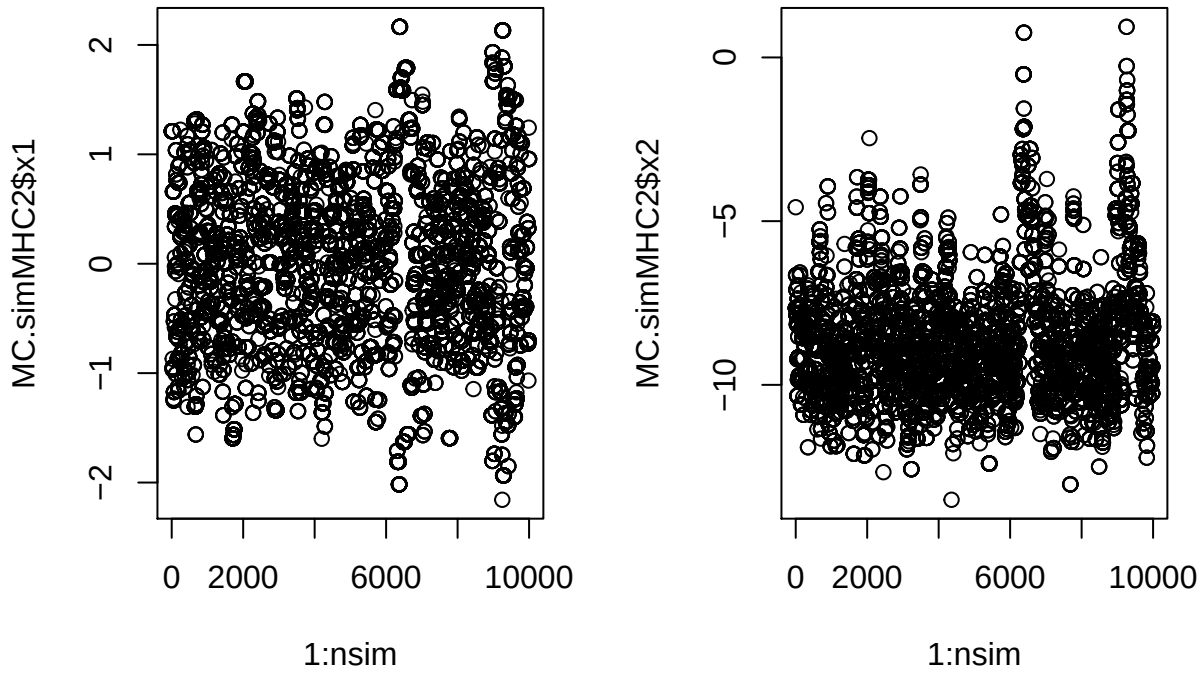
```
par(mfrow=c(1,2))
acf(MC.simMHC$x1)
acf(MC.simMHC$x2)
```



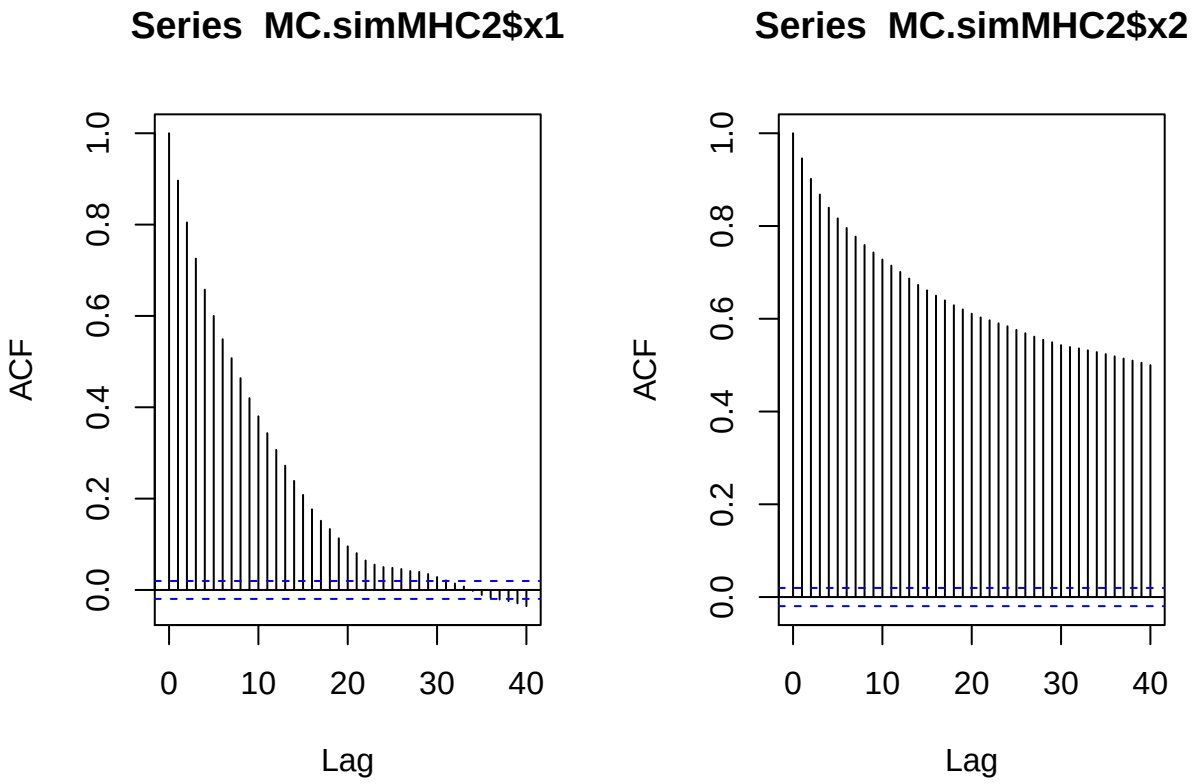

```
image(x1, x2, mat, col = hcl.colors(100, alpha = 0.5), xlim = c(-4, 4),
     ylim = c(-15, 5),
     xlab = expression(x[1]),
     ylab = expression(x[2]),
     main = "Component-wise Metropolis Hasting using small jump size")
points(MC.simMHC$x1, MC.simMHC$x2, col = rgb(0,1,0,alpha = 0.3), pch = 19)
```



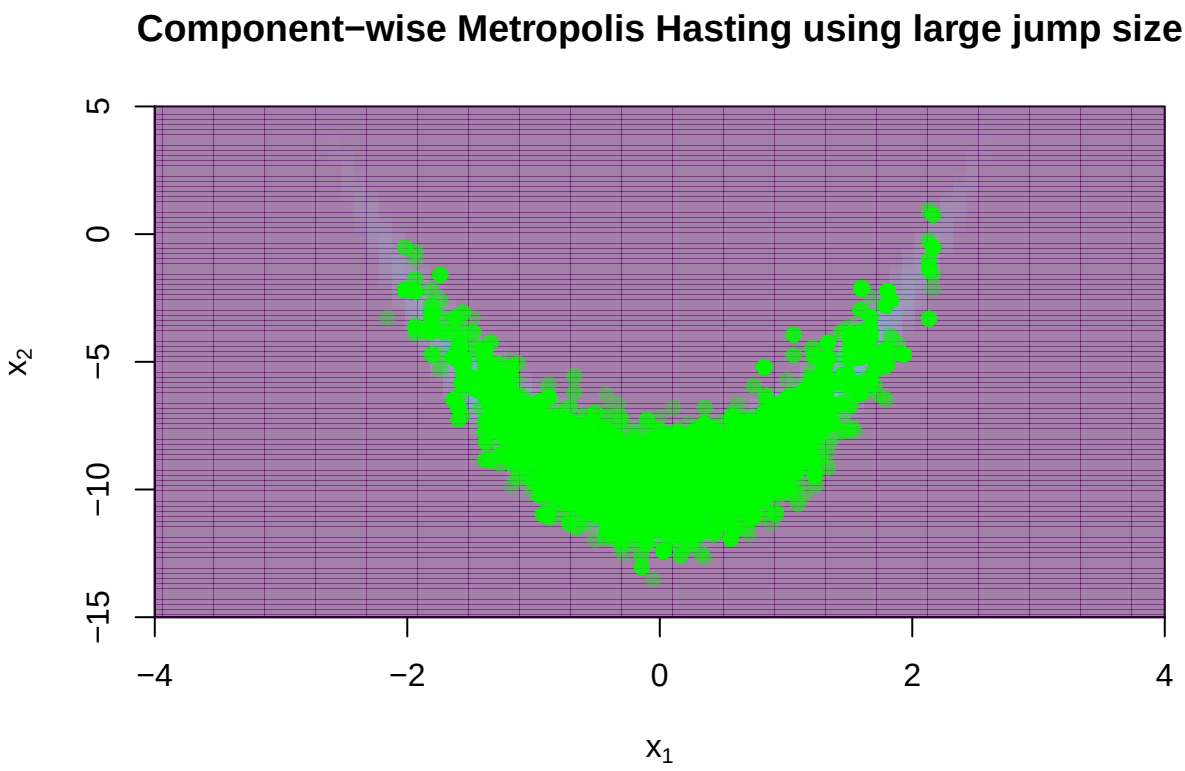

```
# Condition 2: large jump sizes

MC.simMHC2 <- mh.Q5(nsim = nsim, burn = burn, sigma = sigma2, eta = eta2)

par(mfrow=c(1,2))
plot(1:nsim, MC.simMHC2$x1)
plot(1:nsim, MC.simMHC2$x2)
```



```
par(mfrow=c(1,2))
acf(MC.simMHC2$x1)
acf(MC.simMHC2$x2)
```




```
image(x1, x2, mat, col = hcl.colors(100, alpha = 0.5), xlim = c(-4, 4),
      ylim = c(-15, 5),
      xlab = expression(x[1]),
      ylab = expression(x[2]),
      main = "Component-wise Metropolis Hasting using large jump size")
points(MC.simMHC2$x1, MC.simMHC2$x2, col = rgb(0,1,0,alpha = 0.3), pch = 19)
```



(f) Repeat (iv) using component-wise Gibbs sampling.

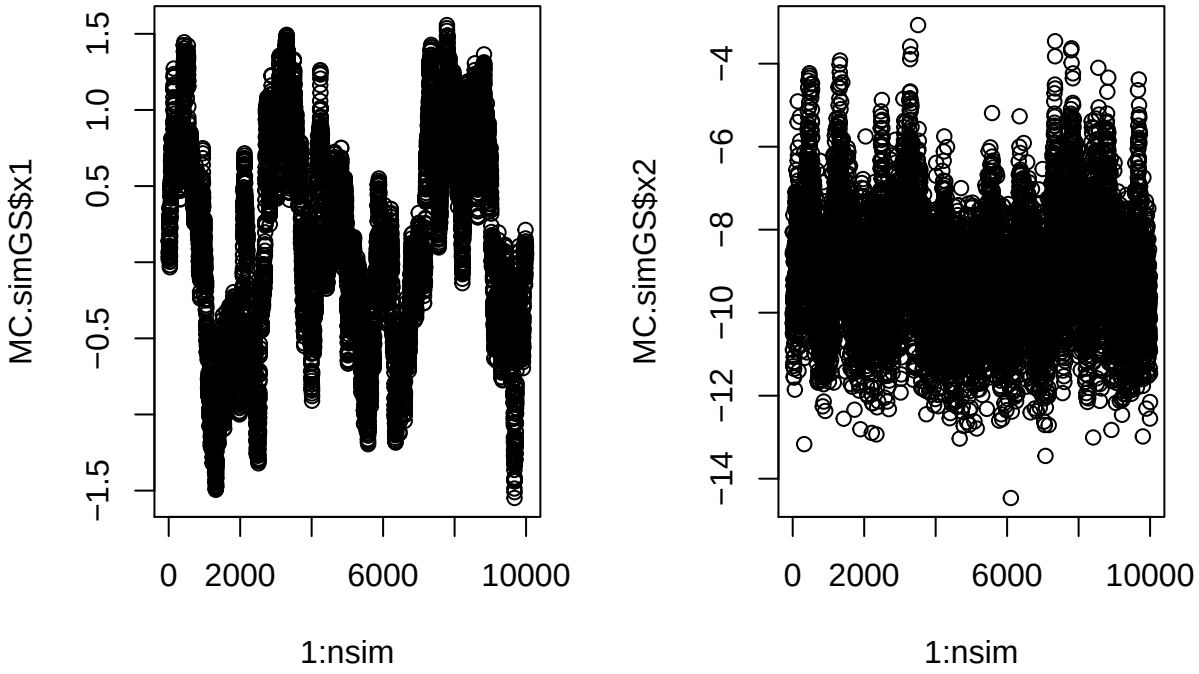
From **(a)** we know that

$$x_2|x_1 \sim N(2(x_1^2 - 5), 1)$$

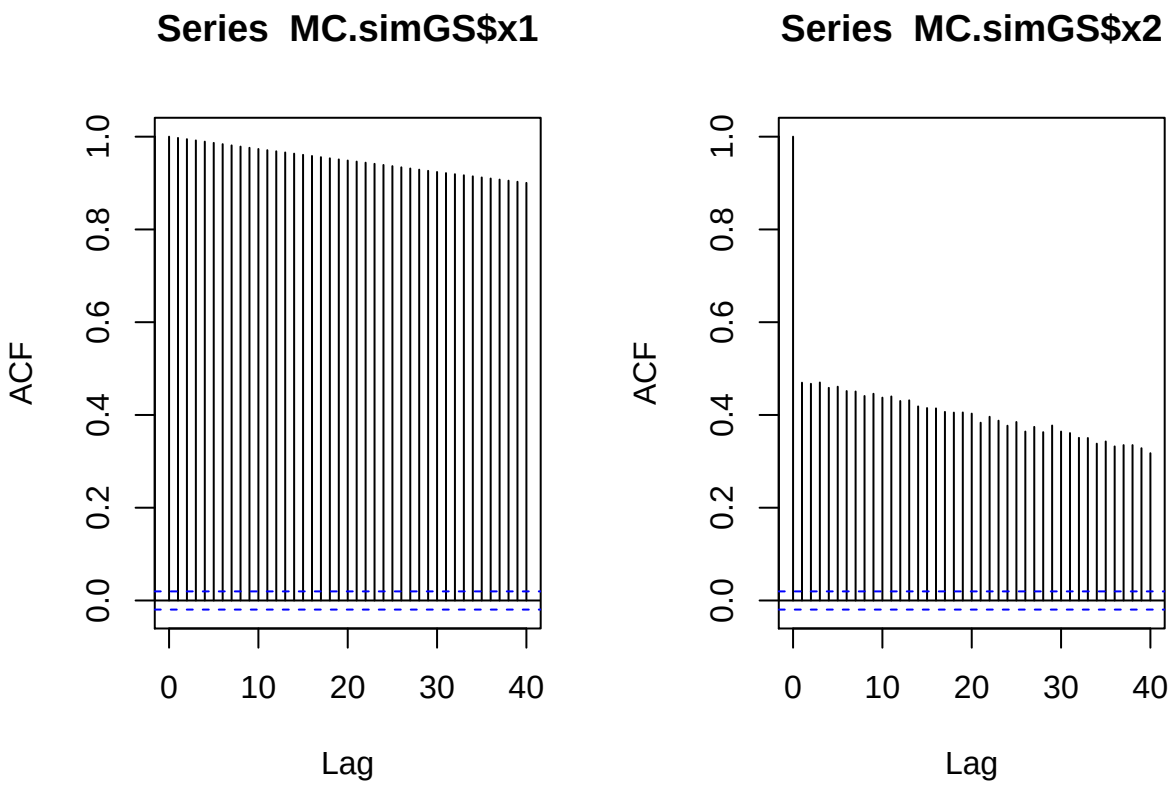
```
gs.Q6 <- function(nsim = 10000, burn = 0.1,
                  sigma = 2, seed = 1998) {
  set.seed(seed)
  x1.ch <- vector()
  x2.ch <- vector()
  x1 <- 0
  x2 <- -10
  # run chain
  nsim1 <- nsim*(1 + burn)
  burni <- nsim*burn

  # x2|x1
  for (isim in 1:nsim1) {
    x1.t <- rnorm(1, x1, sigma)
    P0 <- banana_Q1(x1, x2) %>% log
    P1 <- banana_Q1(x1.t, x2) %>%  log
    ratio <- P1 - P0

    if (log(runif(1)) < ratio) {
      x1 <- x1.t
    }

    x2 <- rnorm(1, 2*(x1^2-5), 1)
```

```
    # Store Chain after burn
    if (isim > burni) {
      i1 <- isim - burni
      x1.ch[i1] <- x1
      x2.ch[i1] <- x2

    }
  }
  return(list(x1 = x1.ch, x2 = x2.ch))
}
```

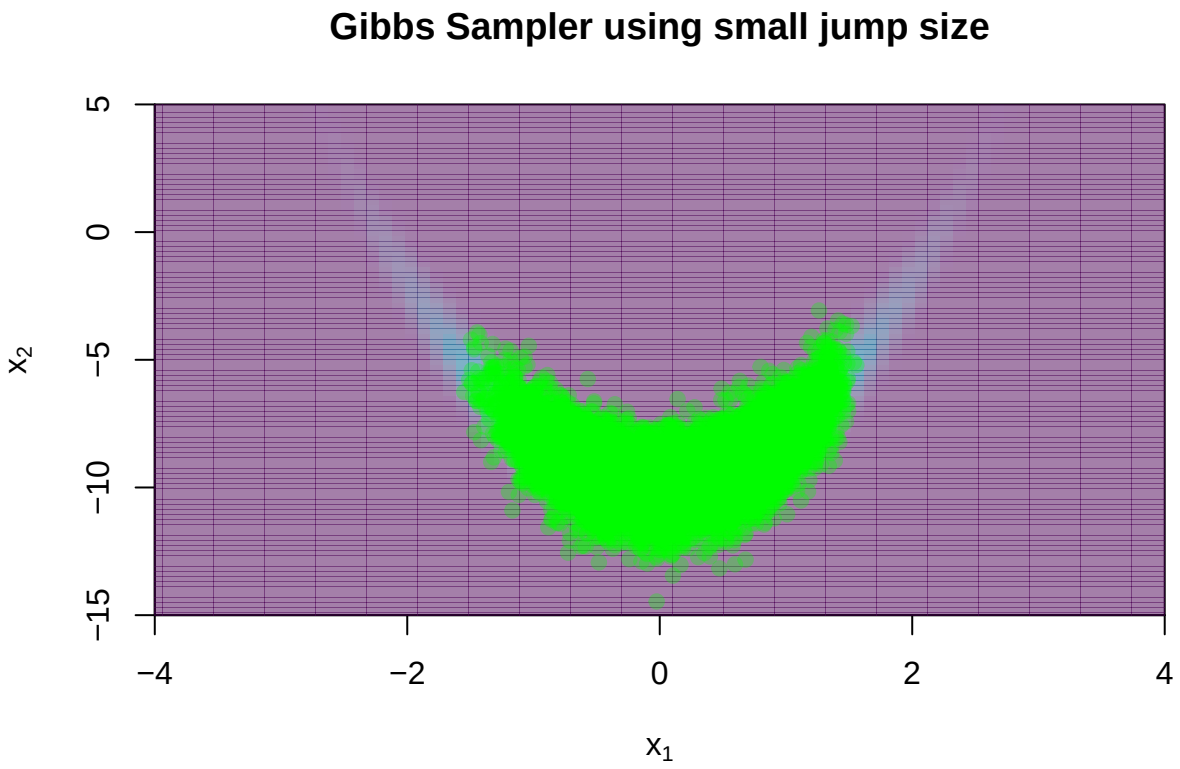
```
# Condition 1: small jump sizes

MC.simGS <- gs.Q6(nsim = nsim, burn = burn, sigma = sigma)

par(mfrow=c(1,2))
plot(1:nsim, MC.simGS$x1)
plot(1:nsim, MC.simGS$x2)
```



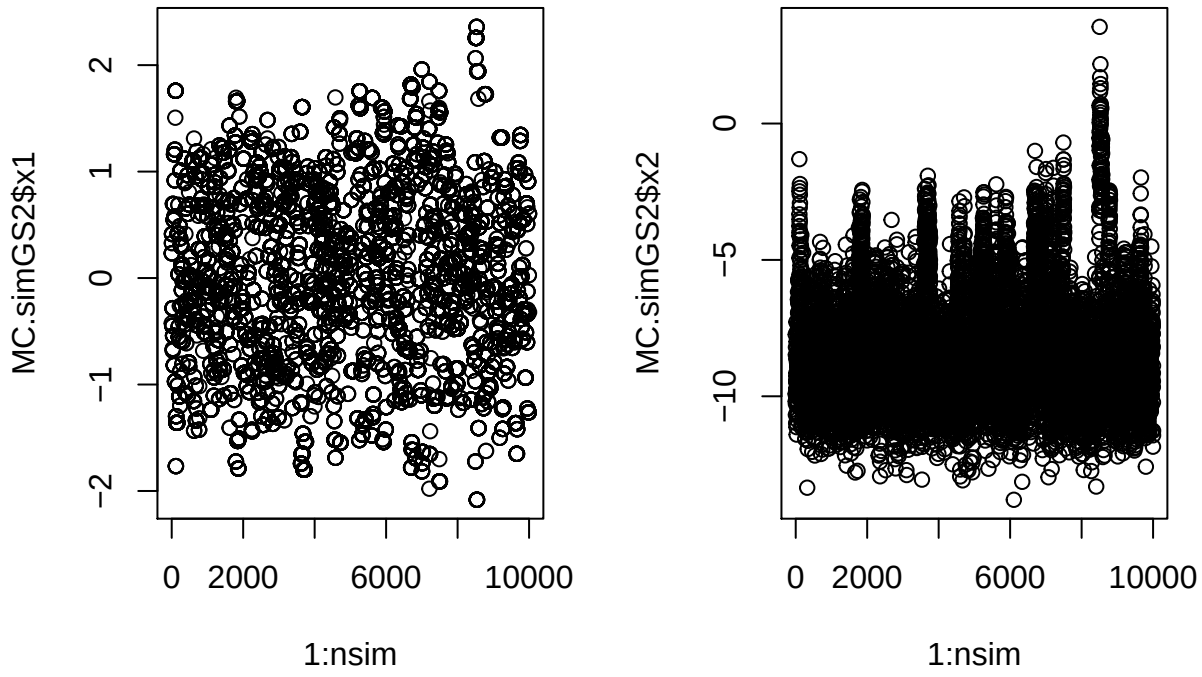

```
par(mfrow=c(1,2))
acf(MC.simGS$x1)
acf(MC.simGS$x2)
```



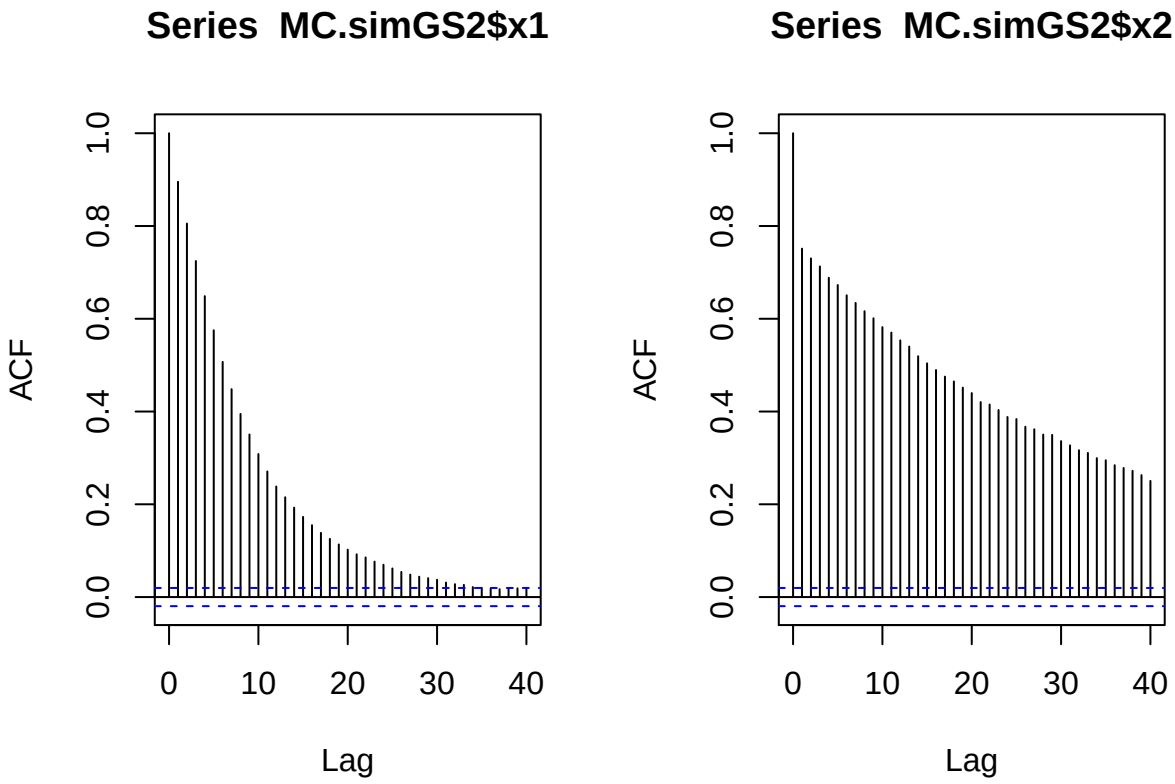

```
image(x1, x2, mat, col = hcl.colors(100, alpha = 0.5), xlim = c(-4, 4),
     ylim = c(-15, 5),
     xlab = expression(x[1]),
     ylab = expression(x[2]),
     main = "Gibbs Sampler using small jump size")
points(MC.simGS$x1, MC.simGS$x2, col = rgb(0,1,0,alpha = 0.3), pch = 19)
```




```
# Condition 2: large jump sizes

MC.simGS2 <- gs.Q6(nsim = nsim, burn = burn, sigma = sigma2)

par(mfrow=c(1,2))
plot(1:nsim, MC.simGS2$x1)
plot(1:nsim, MC.simGS2$x2)
```



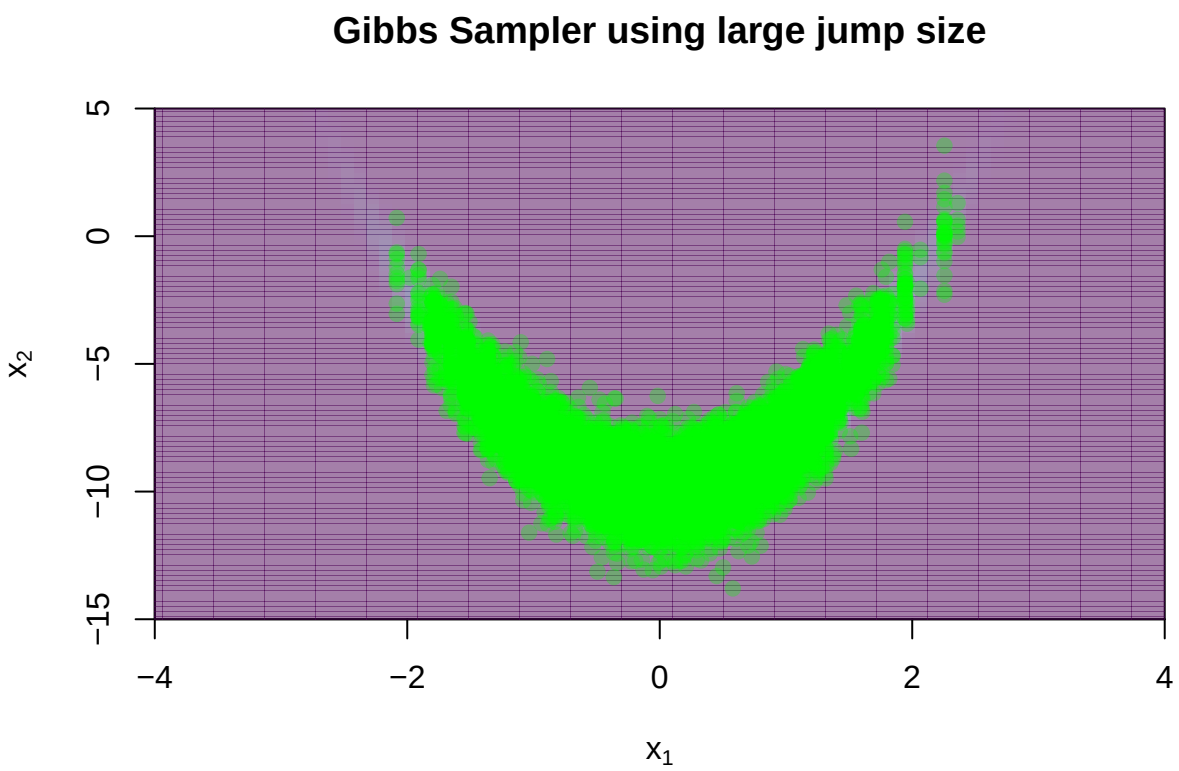
```
par(mfrow=c(1,2))
acf(MC.simGS2$x1)
acf(MC.simGS2$x2)
```




```
image(x1, x2, mat, col = hcl.colors(100, alpha = 0.5), xlim = c(-4, 4),
      ylim = c(-15, 5),
      xlab = expression(x[1]),
      ylab = expression(x[2]),
      main = "Gibbs Sampler using large jump size")
points(MC.simGS2$x1, MC.simGS2$x2, col = rgb(0,1,0,alpha = 0.3), pch = 19)
```

Table 1: Summary of using pdf directly

	Expectation	Standard Deviation	2.5%	97.5%
df_x1	-0.007	0.992	-1.969	1.938
df_x2	0.985	1.387	0.001	4.951
df_x3	-8.032	2.950	-11.440	0.080



(g) Use the algorithms in (a, b, c, d, e, f) to estimate the following quantities: E(x^2_1),E(x_2) and Pr(x_1+x_2>0). Discuss and quantify how different sampling strategies affect Monte Carlo variance.

```
summary_Q7 <- function(x1,x2){
  df_x1 <- c(mean(x1), sd(x1), quantile(x1, c(0.025, 0.975))) %>% round(3)
  df_x2 <- c(mean(x1^2), sd(x1^2), quantile(x1^2, c(0.025, 0.975))) %>% round(3)
  df_x3 <- c(mean(x2), sd(x2), quantile(x2, c(0.025, 0.975))) %>% round(3)
  df_x4 <- sum(x1 + x2 > 0)/length(x1)
  out <- rbind(df_x1, df_x2, df_x3)
  out2 <- df_x4  %>% round(3)
  colnames(out)[c(1,2)] <- c("Expectation", "Standard Deviation")
  return(list(out = out, out2 = out2))

}
```

```
# Question a (use pdf directly)
Q7_a <- summary_Q7(x1_Q1, x2_Q1)
Q7_a$out %>%
  kbl(caption = "Summary of using pdf directly") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

Table 2: Summary of using accept-reject method

	Expectation	Standard Deviation	2.5%	97.5%
df_x1	-0.002	0.963	-1.816	1.748
df_x2	0.912	1.625	0.001	3.598
df_x3	-7.932	3.452	-11.147	-2.128

Table 3: Summary of using importance sampling

	Expectation	Standard Deviation	2.5%	97.5%
df_x1	0.067	1.000	-1.882	2.004
df_x2	1.004	1.398	0.001	5.087
df_x3	-8.007	2.964	-11.338	-0.330

```
Q7_a$out2
```

```
## [1] 0.029
```

```
# Question b (AR)
Q7_b <- summary_Q7(Q2$x1[Q2$Accept == "TRUE"], Q2$x2[Q2$Accept == "TRUE"])
Q7_b$out %>%
  kbl(caption = "Summary of using accept-reject method") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

```
Q7_b$out2
```

```
## [1] 0.034
```

```
# Question c (Importance Sampling)
Q7_c <- summary_Q7(x1_IS, x2_IS)
Q7_c$out %>%
  kbl(caption = "Summary of using importance sampling") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

```
Q7_c$out2
```

```
## [1] 0.033
```

```
# Question d (Random-walk Metropolis Hasting)
Q7_d <- summary_Q7(MC.simMH2$x1, MC.simMH2$x2)
Q7_d$out %>%
  kbl(caption = "Summary of using Random-walk Metropolis Hasting") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

```
Q7_d$out2
```

```
## [1] 0.045
```

Table 4: Summary of using Random-walk Metropolis Hasting

	Expectation	Standard Deviation	2.5%	97.5%
df_x1	0.081	1.063	-1.781	2.341
df_x2	1.136	1.710	0.003	6.347
df_x3	-7.727	3.602	-11.493	3.317

Table 5: Summary of using Component-wise Metropolis Hasting

	Expectation	Standard Deviation	2.5%	97.5%
df_x1	0.049	0.819	-1.560	1.666
df_x2	0.673	0.832	0.000	3.024
df_x3	-8.660	1.893	-11.432	-3.880

Table 6: Summary of using Gibbs Sampling

	Expectation	Standard Deviation	2.5%	97.5%
df_x1	0.077	0.869	-1.547	1.659
df_x2	0.761	0.872	0.001	3.081
df_x3	-8.459	2.016	-11.385	-3.465

```
# Question e (Component-wise Metropolis Hasting)
Q7_e <- summary_Q7(MC.simMHC2$x1, MC.simMHC2$x2)
Q7_e$out %>%
  kbl(caption = "Summary of using Component-wise Metropolis Hasting") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

```
Q7_e$out2
```

```
## [1] 0.003
```

```
# Question f (Gibbs Sampling)
Q7_f <- summary_Q7(MC.simGS2$x1, MC.simGS2$x2)
Q7_f$out %>%
  kbl(caption = "Summary of using Gibbs Sampling") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

```
Q7_f$out2
```

```
## [1] 0.005
```

According to the result, Importance Sampling can help us reduce the Monte Carlo variances comparing to Accept-Reject method. In addition, component-wise Metropolis Hasting also reduces the variances compared to the Random-Walk one. Gibbs sampling is better than the Random-Walk Metropolis Hasting, but worse than component-wise Metropolis Hasting. In my case, component-wise Metropolis Hasting helps reduce reduce the Monte Carlo variances most.

(h) For the estimation of the quantities in **(g)**, discuss at least two strategies that could be implemented for some of the algorithms to reduce Monte Carlo variance. Implement these strategies and quantify your improved estimators in a Monte Carlo study.

In class we discussed about methods of reducing Monte Carlo variances including Antithetic sampling, Control variates and Rao-Blackwellization except for Importance Sampling.

(1) Rao-Blackwell

```
# Generate x_2 using Rao-Blackwell (Component-wise Metropolis Hasting)
MC.simMHC2.rb <- 2*(MC.simMHC2$x1^2-5)
summary_Q7(MC.simMHC2$x1, MC.simMHC2.rb)$out %>%
  kbl(caption = "Summary of using Rao-Blackwell") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

Table 7: Summary of using Rao-Blackwell

	Expectation	Standard Deviation	2.5%	97.5%
df_x1	0.049	0.819	-1.560	1.666
df_x2	0.673	0.832	0.000	3.024
df_x3	-8.654	1.663	-9.999	-3.953

```
summary_Q7(MC.simMHC2$x1, MC.simMHC2.rb)$out2
```

```
## [1] 0.004
```

The standard deviation of x_2 reduces to 1.663 compared to 1.893 getting from Component-wise Metropolis Hasting.

(2) Antithetic Variate

```
# compared to AR method

library(mvtnorm)
acc_rej_anti <- function(nsim, M, sd = 40, seed_Q2 = 1996) {
  set.seed(seed_Q2)
  # generate g(x)- a multivariate t distribution
  x <- rmvt(n = nsim/2, sigma = diag(sd, 2), df = 6, delta = c(0, -10),
            type = "shifted")
  # generate antithetic variates
  x_anti <- rbind(x, cbind(-x[, 1], x[, 2]))
  # generate u ~ U[0, 1]
  u <- runif(nsim)
  Accept <- vector()
  fx <- vector()
  g <- vector()
  for (i in 1:nsim) {
    gx = dmvt(x = x_anti[i, ], sigma = diag(sd, 2), df = 6, delta = c(0, -10),
              log = FALSE, type = "shifted")
    # check if the condition that f(x) <= M*g(x) is satisfied
    if (banana_Q2(x_anti[i, ]) > M*gx) {
      print(i)
      print(x_anti[i, ])
      print(banana_Q2(x_anti[i, ]))
      print(M*gx)
      stop("The condition fails")
    }
    # accept y = x if u <= f(x)/(M*g(x))
    if (u[i] <= banana_Q2(x_anti[i, ])/(M*gx)) {
      Accept[i] <- TRUE
      fx[i] <- banana_Q2(x_anti[i, ])
      g[i] <- gx
    } else {
      Accept[i] <- FALSE
      fx[i] <- banana_Q2(x_anti[i, ])
      g[i] <- gx
    }
  }
  return(list(x1 = x_anti[, 1], x2 = x_anti[, 2],
```

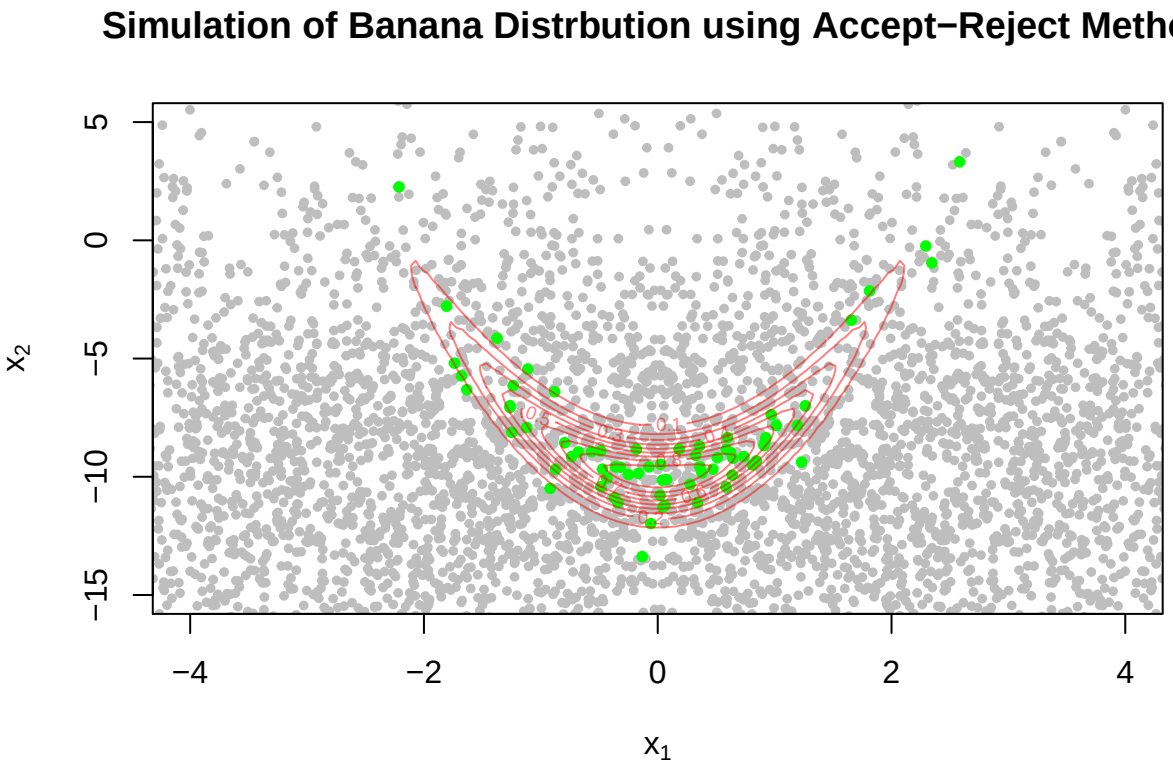
Table 8: Summary of using antithetic variate

	Expectation	Standard Deviation	2.5%	97.5%
df_x1	0.020	1.015	-1.758	2.308
df_x2	1.015	1.426	0.001	5.326
df_x3	-8.157	3.102	-11.471	0.447

```
              Accept = Accept, fx = fx, g = g))
}
Q8 <- acc_rej_anti(10000, 1000)
# accuracy
sum(Q8$Accept)/length(Q8$Accept) # accuracy is low, about 0.005
```

```
## [1] 0.007
```

```
# plot
plot(Q8$x1, Q8$x2, xlim = c(-4, 4), pch = 20, cex = 0.8, col = "grey",
     ylim = c(-15, 5), xlab = expression(x[1]), ylab = expression(x[2]),
     main = "Simulation of Banana Distrbution using Accept-Reject Method")
points(Q8$x1[Q8$Accept == "TRUE"], Q8$x2[Q8$Accept == "TRUE"],
       pch = 20, col = "green")
contour(x1, x2, mat, add = T, col = rgb(1, 0, 0, alpha = 0.5))
```




```
summary_Q7(Q8$x1[Q8$Accept == "TRUE"], Q8$x2[Q8$Accept == "TRUE"])$out %>%
  kbl(caption = "Summary of using antithetic variate") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

```
summary_Q7(Q8$x1[Q8$Accept == "TRUE"], Q8$x2[Q8$Accept == "TRUE"])$out2
```

```
## [1] 0.057
```

Compared to the standard deviation of estimators we got from AR method, the standard deviations we got from Antithetic Variate is kind of smaller.

2. Bayesian Adaptive Lasso

Please describe all your work in clear terms, before implementing R code. Each question should include a description of your approach with clear indication of where I can find the associated source code. Your code should be attached to your assignment or uploaded online to a repository I can freely access.

Consider the implementation of a posterior simulation algorithm for Bayesian adaptive LASSO. More precisely consider the following model:

$$Y|\beta,\sigma^2 \sim N(X\beta,\sigma^2 I_n),$$

with $Y \in \mathbb{R}^n, X \in \mathbb{R}^{n\times p}$, and $\beta \in \mathbb{R}^p$. The model is completed with priors:

$$\beta|\tau_1^2,\cdots,\tau_p^2 \sim N_p(\ \{0,\text{diag}(\tau_1^2,\cdots,\tau_p^2)\},$$

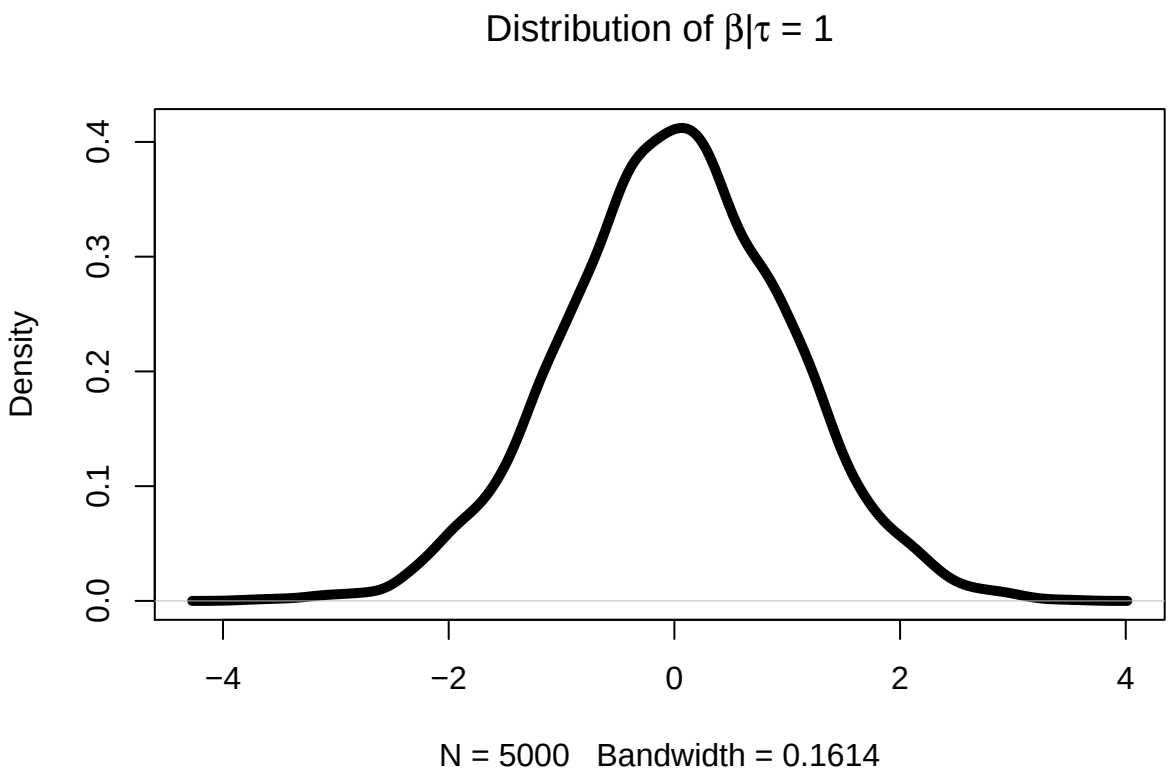
$$p(\tau_j^2|\lambda^2) \propto \exp -\frac{\lambda^2}{2}\tau_j^2; \quad j=1,\cdots,p$$

$$h := 1/\sigma^2 \sim Gamma(0.1, rate = 10)$$

(a). Consider $p = 1$. Simulate $5,000$ Monte Carlo samples from the conditional prior $\beta|\tau^2 = 1$ and obtain a plot of the density using the R function density.

If $p = 1$, then $\beta|\tau = 1 \sim N(0,1)$, we can use Accept-Reject method to generate β:

```
set.seed(1998)
nsim_2a <- 5000000
u_2a <- runif(nsim_2a)
gx_2a <- 1/50
M_2a <- 50
x_2a <- 50*runif(nsim_2a) - 25
fx_2a <- dnorm(x_2a)
beta_2 <- x_2a[u_2a <= fx_2a/(M_2a*gx_2a)]
beta_2a <- beta_2[1:5000]
density(beta_2a) %>% plot(., main = expression(paste("Distribution of ",
                                                     beta,"|", tau, " = 1")),
                          lwd = 5)
```

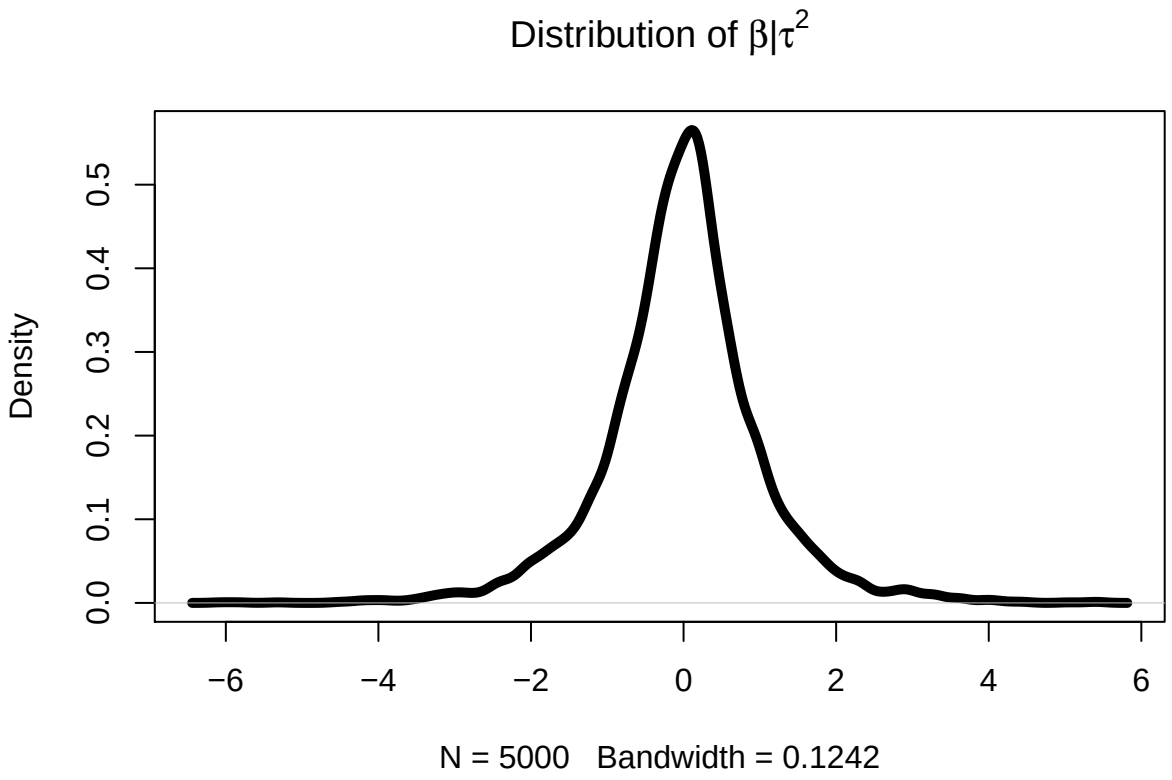


(b). Consider $p = 1$. Simulate $5,000$ Monte Carlo samples from the marginal prior β, considering $\lambda^2 = 2$, so that $E(\tau^2|\lambda^2) = 1$. Obtain a plot of the density as in **(a)**.

We know $p = 1$ and $\lambda^2 = 2$ so that $P(\tau^2|\lambda^2) = \exp(-\tau^2), E(\tau^2|\lambda^2) = 1$. We can use inverse transformation method to generate τ and then use Accept-Reject method like **(a)** to generate β:

$$0 \leq \exp(-\tau^2) \leq 1 \Rightarrow \text{generate } \tau = \sqrt{-\log U}, \text{ where } U \sim Uniform[0,1]$$

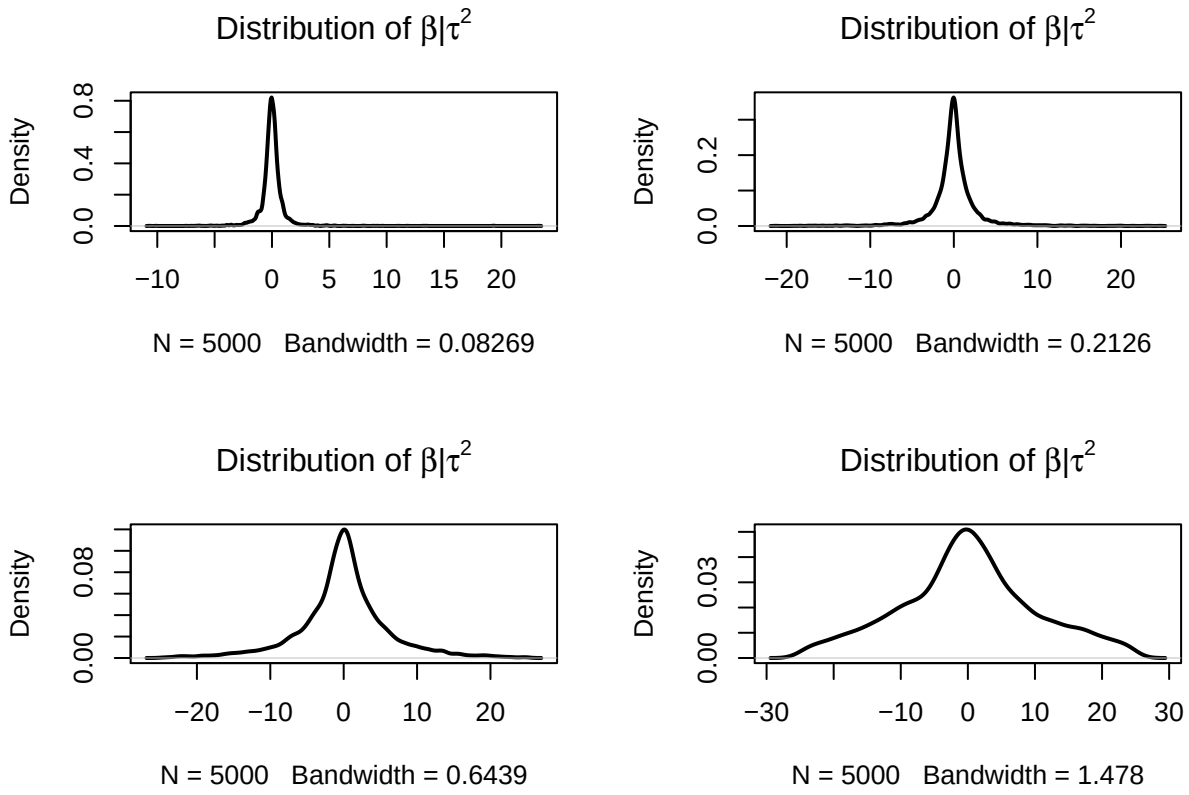
```
set.seed(1998)
nsim_2b <- 5000000
tau_2b <- sqrt(-log(runif(nsim_2b)))
u_2b <- runif(nsim_2b)
gx_2b <- 1/50
M_2b <- 50
x_2b <- 50*runif(nsim_2b) - 25
fx_2b <- dnorm(x_2b, mean = 0, sd = tau_2b)
beta_22 <- x_2b[u_2b <= fx_2b/(M_2b*gx_2b)]
beta_2b <- beta_22[1:5000]
density(beta_2b) %>% plot(., main = expression(paste("Distribution of ",
                                                     beta, "|", tau^2,)),
                          lwd = 5)
```



(c). Consider $p = 1$. Add a hyper prior on $\gamma = \lambda^2 \sim Gamma(a, rate = b)$. Assess how the marginal prior of β changes for $a = 1$ and values of $b \geq 1$.

We know that $p = 1$ and $a = 1$, $\gamma \sim Exponential(b)$

```
par(mfrow=c(2,2))
set.seed(1998)
for (b in c(0.1, 1, 10, 100)) {
  nsim_2c <- 5000000
  gamma <- -1/b*log(runif(nsim_2c))
  lambda_sq <- gamma
  tau_2c <- sqrt(-2*log(runif(nsim_2c))/lambda_sq)
  u_2c <- runif(nsim_2c)
  gx_2c <- 1/50
  M_2c <- 50
  x_2c <- 50*runif(nsim_2c) - 25
  fx_2c <- dnorm(x_2c, mean = 0, sd = tau_2c)
  beta_23 <- x_2c[u_2c <= fx_2c/(M_2c*gx_2c)]
  beta_2c <- beta_23[1:5000]
  density(beta_2c) %>% plot(., main = expression(paste("Distribution of ",
                                                       beta, "|", tau^2)),
                            lwd = 2)
}
```



(d). Considering the hyper prior in **(c)**, describe a Markov Chain Monte Carlo algorithm to sample from the posterior distribution of β and σ^2.

From **(c)** we know that $\lambda^2 \sim InverseGamma(a, b)$, $\sigma^2 \sim InverseGamma(0.1, 10)$ and $p(\tau_j^2|\lambda^2) \propto \exp -\frac{\lambda^2}{2}\tau_j^2$; $\quad j = 1, \cdots, p$, we can get the joint distribution of λ^2, τ^2:

$$p(\lambda^2, \{\tau_1^2, \cdots, \tau_p^2\}) = [\prod_{i=1}^{n} p(\tau_j^2|\lambda^2)]p(\lambda^2)$$

Then we can get the posterior distrbution of σ^2:

$$\begin{aligned} p(\sigma^2|Y, \beta, \{\tau_1^2, \cdots, \tau_p^2\}, \lambda^2) &= p(\sigma^2|Y, \beta) \\ &\propto p(Y|\sigma^2, \beta)p(\sigma^2) \\ &= \exp[-\frac{(Y - X\beta)^T(Y - X\beta)}{2\sigma^2}] \cdot (\frac{1}{\sigma^2})^{\frac{n}{2}} \cdot \exp[-\frac{10}{\sigma^2}] \cdot (\frac{1}{\sigma^2})^{a+1} \\ &\sim Inversegamma(0.1 + \frac{n}{2}, \frac{(Y - X\beta)^T(Y - X\beta)}{2} + 10) \end{aligned}$$

and also the posterior distribution of β:

$$
\begin{aligned}
p(\beta|Y,\sigma^2,\{\tau_1^2,\cdots,\tau_p^2\},\lambda^2) &= p(\beta|Y,\sigma^2,\{\tau_1^2,\cdots,\tau_p^2\}) \\
&\propto p(Y|\sigma^2,\beta)p(\beta|\tau_1^2,\cdots,\tau_p^2) \\
&\propto \exp\{-\frac{1}{2}[\beta^T(\frac{X^TX}{\sigma^2}+\text{diag}(1/\tau_1^2,\cdots,1/\tau_p^2))\beta-\frac{2\beta^TX^TY}{\sigma^2}+\frac{Y^TY}{\sigma^2}]\} \\
&\sim N([\frac{X^TX}{\sigma^2}+\text{diag}(1/\tau_1^2,\cdots,1/\tau_p^2)]^{-1}\ \frac{X^TY}{\sigma^2},[\frac{X^TX}{\sigma^2}+\text{diag}(1/\tau_1^2,\cdots,1/\tau_p^2)]^{-1})
\end{aligned}
$$

For the full conditional distribution of τ_i^2:

$$
\begin{aligned}
p(\tau_i^2|Y,\beta,\sigma^2,\lambda^2) &\propto (\tau_i^2)^{-\frac{1}{2}}\exp(-\frac{1}{2}\beta^TD_\tau^{-1}\beta)\exp(-\frac{\lambda^2\tau_i^2}{2}) \\
&\propto (\tau_i^2)^{-\frac{1}{2}}\exp(-\frac{\beta_i^2}{2\tau_i^2})\exp(-\frac{\lambda^2\tau_i^2}{2})
\end{aligned}
$$

Thus $p(\tau_i^2|Y,\beta,\sigma^2,\lambda^2)$ is independent of τ_j^2, i.e. τ_i^2 and τ_j^2 are conditionally independent. Then if we explore the distribution of $1/\tau_i^2$:

$$
\begin{aligned}
p(x|Y,\beta,\sigma^2,\lambda^2) &\propto x^{\frac{1}{2}}\exp[-\frac{\lambda^2}{2}(\frac{\beta_i^2}{\lambda^2}x+\frac{1}{x})]\frac{1}{x^2} \\
&\propto x^{-3/2}\exp[-\frac{\lambda^2}{2}(\frac{\beta_i^2}{\lambda^2}x+\frac{1}{x})] \\
&\sim InverseGaussian(\frac{\lambda}{\beta_i},\lambda^2)
\end{aligned}
$$

which denotes a inverse gaussian distribution with mean $\frac{\lambda}{\beta_i}$ and shape parameter λ^2.

For the full conditional distribution of λ^2:

$$
\begin{aligned}
p(\lambda^2|Y,\beta,\sigma^2,\{\tau_1^2,\cdots,\tau_p^2\}) &\propto (\lambda^2)^{a+p-1}\exp(-b\lambda^2-\frac{1}{2}\sum_{j=1}^p\tau_j^2) \\
&\sim Gamma(a+p,b+\frac{1}{2}\sum_{j=1}^p\tau_j^2)
\end{aligned}
$$

Now, we have full conditional distribution for all the parameters and hence can run Metropolis Hasting with Gibbs sampler using the above full conditionals and thus generate samples from the joint posterior.

Steps:

(1) Sample λ^2 from the posterior distribution $p(\lambda^2|Y,\sigma^2,\beta,\{\tau_1^2,\cdots,\tau_p^2\}) \propto Gamma(a+p,b+\frac{1}{2}\sum_{j=1}^p\tau_j^2)$ given values of τ^2.

(2) Sample τ^2 from the posterior distribution $p(1/\tau_j^2|Y,\beta_j,\sigma^2,\lambda^2) \propto InverseGaussain(\lambda/\beta_i,\lambda^2)$ given values of λ and β_i.

(3) Sample σ^2 from the posterior distribution $p(\sigma^2|Y,\beta,\{\tau_1^2,\cdots,\tau_p^2\},\lambda^2) \propto Inversegamma(0.1+\frac{n}{2},\frac{(Y-X\beta)^T(Y-X\beta)}{2}+10)$ given values of β.

(4) Sample β from the posterior distribution $p(\beta|Y, \sigma^2, \{\tau_1^2, \cdots, \tau_p^2\}, \lambda^2) \propto N([\frac{X^TX}{\sigma^2} + \text{diag}(1/\tau_1^2, \cdots, 1/\tau_p^2)]^{-1} \frac{X^TY}{\sigma^2}, [\frac{X^TX}{\sigma^2} + \text{diag}(1/\tau_1^2, \cdots, 1/\tau_p^2)]^{-1})$ given values of τ^2 and σ^2.

(e). Implement such algorithm in `R` and compare your results with estimates obtained using `glmnet()`. In particular, you should test your results on the `diabetes` data available from `lars`, (use the matrix of predictors x).

```
# load packages
library(mvtnorm)
library(lars)
```

```
## Loaded lars 1.2
```

```
library(MCMCpack)
```

```
## Loading required package: coda
```

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
## ##
## ## Markov Chain Monte Carlo Package (MCMCpack)
```

```
## ## Copyright (C) 2003-2022 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park
```

```
## ##
## ## Support provided by the U.S. National Science Foundation
```

```
## ## (Grants SES-0350646 and SES-0350613)
## ##
```

```
library(statmod)

# load data
data("diabetes")
dia2e_x <- diabetes$x # matrix with 10 columns
dia2e_y <- diabetes$y # a numerical vector
beta_hat <- solve(t(dia2e_x) %*% dia2e_x) %*% t(dia2e_x) %*% dia2e_y

init_2e<- list(tau_sq = rep(5, 10),
               beta = beta_hat,
               sigma_sq = 0.02,
               lambda_sq = 1)

gibbs_2e <- function(init_2e,                      # initial values
                     nsim = 1500, burn = 0.1,      # chain parameters
                     a = 1, b = 2,                 # prior
                     X = dia2e_x, Y = dia2e_y,     # dataset
                     seed = 1998,
                     reg_path = FALSE,             # 2f
                     sen_analysis = FALSE){        # 2g
  set.seed(seed)
  # initialization-------------------------------------------
  tau_sq <- init_2e$tau_sq
```

```
  beta <- init_2e$beta
  sigma_sq <- init_2e$sigma_sq
  lambda_sq <- init_2e$lambda_sq

  tau_sq.ch <- matrix(NA, nrow = nsim, ncol = 10)
  beta.ch <- matrix(NA, nrow = nsim, ncol = 10)
  betamean.ch <- matrix(NA, nrow = nsim, ncol = 10)
  sigma_sq.ch <- vector()

  # Run chain-----------------------------------------------------
  nsim1 <- nsim * (1.0 + burn)
  burni <- nsim * burn

  for (isim in 1:nsim1) {

    # beta ~ multivariate normal
    betamean <- solve(t(X) %*% X + diag(sigma_sq/tau_sq)) %*% t(X) %*% Y
    betavar <- solve((t(X) %*% X)/sigma_sq + diag(1/tau_sq))
    beta <- rmvnorm(n = 1, mean = betamean, sigma = betavar)

    # sigma_sq ~ inverse gamma
    sigma_sq <- 1/(rgamma(1, shape = 0.1 + 442/2,
                          rate = 0.5 * t(Y - X %*% t(beta)) %*%
                            (Y - X %*% t(beta)) + 10))

    #lambda_sq ~ gamma
    if(reg_path == TRUE){               # for 2f (fix lambda)
      lambda_sq <- lambda_sq
    }else if(sen_analysis == TRUE){     # for 2g (free lambda but a, b change)
      lambda_sq <- rgamma(1, shape = a + 10, rate = b + sum(tau_sq)/2)
    }else{
      lambda_sq <- rgamma(1, shape = a + 10, rate = b + sum(tau_sq)/2)
    }

    # tau_sq
    tau_sq <- apply(matrix(beta), 1, function(x) {
      out <- rinvgauss(1, mean = sqrt(lambda_sq/x^2),
                       shape = lambda_sq)
      return(1/out)
    }
    )


     if(isim > burni){
       i1 <- isim - burni
       betamean.ch[i1,] <- betamean
       beta.ch[i1,] <- beta
       sigma_sq.ch[i1] <- sigma_sq
       tau_sq.ch[i1,] <- tau_sq
    }
  }
  return(list(beta = beta.ch, sigma_sq = sigma_sq.ch, tau_sq = tau_sq.ch))
}
```

MCMC-Gibbs(25th Percentile)	MCMC-Gibbs(Median)	MCMC-Gibbs(75th Percentile)	Glmnet
-47.957911	12.52458	78.822826	-9.057332
-162.131806	-66.09061	3.351504	-238.879818
298.835176	428.61473	557.929435	520.885704
78.676523	185.12080	305.639532	323.449275
-98.495951	-11.02966	59.758079	-680.104304
-109.600135	-22.27693	46.965577	390.311899
-226.945631	-109.75138	-19.824873	48.691747
-10.264366	71.91060	191.951227	159.307982
202.056575	337.69835	484.012000	710.278264
-9.188768	61.20476	155.206515	67.555464

```
results_gibbs_2e <- gibbs_2e(init_2e, nsim = 10000, burn = 0.1, b = 2)
```

```
# use glmnet to get estimators
library(MASS)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-1
```

```
lasso_2e <- glmnet(dia2e_x, dia2e_y, intercept = FALSE)
coeff_2e <- coef(lasso_2e, s = min(lasso_2e$lambda))

# comparison
coeff_gibbs <- apply(results_gibbs_2e$beta, 2,
                     function(x){quantile(x,c(0.25, 0.5, 0.75))})

coeff_25th <- coeff_gibbs[1, ]
coeff_med <- coeff_gibbs[2, ]
coeff_75th <- coeff_gibbs[3, ]

comparison <- tibble("MCMC-Gibbs(25th Percentile)" = coeff_25th,
                     "MCMC-Gibbs(Median)" = coeff_med,
                     "MCMC-Gibbs(75th Percentile)" = coeff_75th,
                     "Glmnet" = coeff_2e[-1])

comparison %>%
  kbl(caption = NULL) %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

According to the result of the table, we can see some of the estimators are close but others are quite different, even not on the same magnitude. I think Bayesian Lasso is more general than the classic Lasso and it can give us more reasonable result of estimators. The classic one also has the problem of shrinkage of coefficient estimates towards zero.

(f). For the diabetes data, fix λ and produce a regularization path for adaptive Bayesian Lasso obtained on a grid of values for the tuning parameter λ. Describe your approach and compare your result with the path obtained using `glmnet()`.

For this question we can add an option in our function of **(e)**, we generate some λ and set them as initial values in Gibbs sampler, without changing in the whole process of each turn of simulation.

```
# fix values of lambda
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(plotmo)
```

```
## Loading required package: Formula
```

```
## Loading required package: plotrix
```

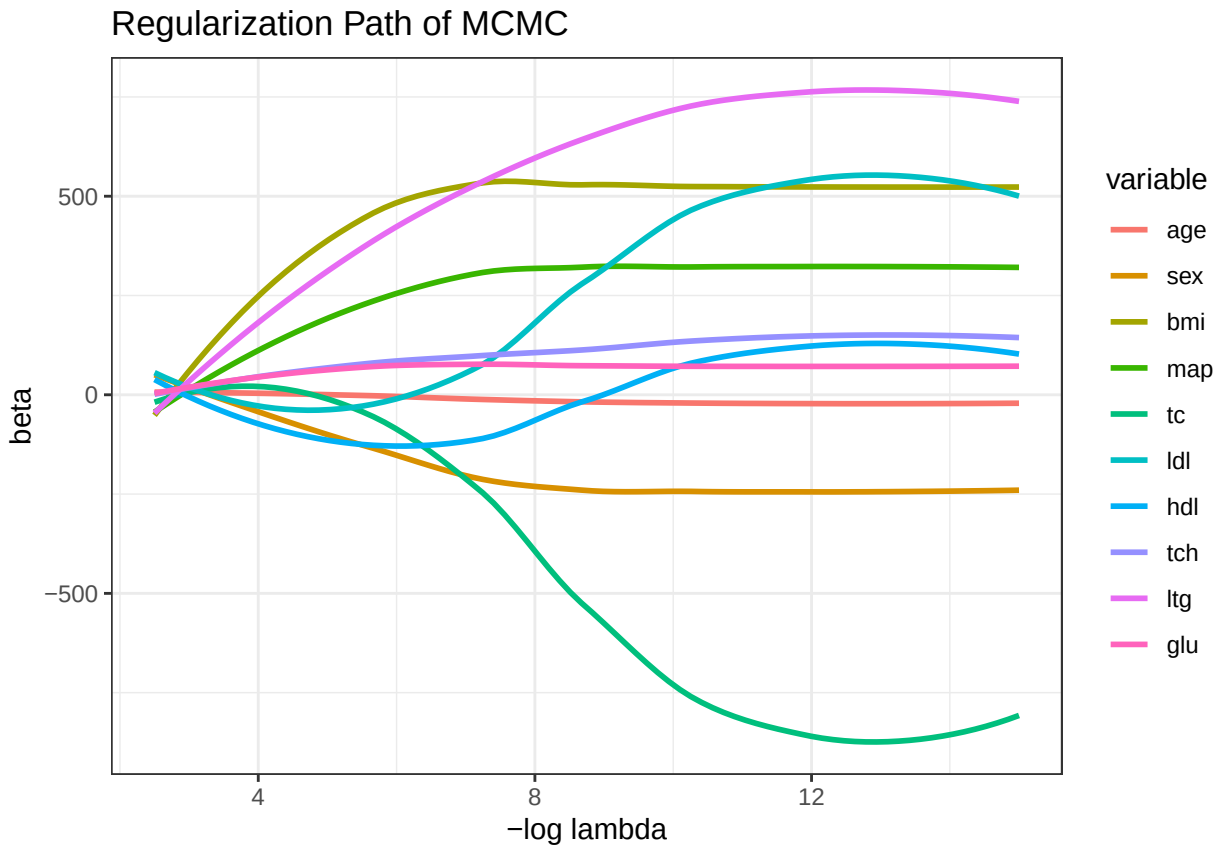
```
## Loading required package: TeachingDemos
```

```
lambda_2f <- data.frame(lambda_sq = exp(seq(-15, 5, 0.1))^2) # lambda_sq

# regression_path
test_2f <- apply(lambda_2f, 1,
                 function(x){
                   init_2e$lambda_sq <- x
                   return(gibbs_2e(init_2e, nsim = 1000,
                                   burn = 0.5, b = 2, reg_path = TRUE))})

results_2f <- lapply(test_2f, function(x){apply(x$beta, 2, median)})
results_plot_2f <- as.data.frame(Reduce("rbind", results_2f))
colnames(results_plot_2f) <- colnames(dia2e_x)
results_plot_2f$lambda <- log(sqrt(lambda_2f$lambda_sq)) # return to log lambda
results_plot_2f2 <- melt(results_plot_2f, id.vars = names(results_plot_2f)[11])
```

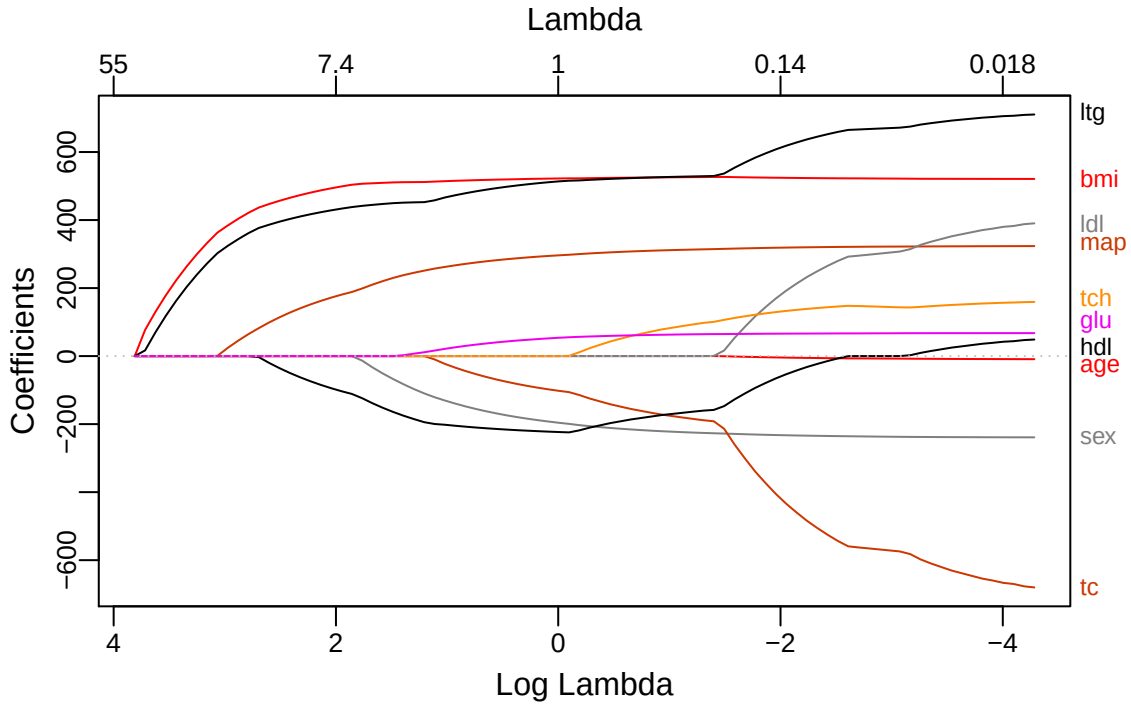
```
ggplot(results_plot_2f2, aes(-lambda, value, color = variable)) +
  geom_smooth(method = loess, se = FALSE) +
  theme_bw() +
  xlim(2.5, 15) +
  ggtitle("Regularization Path of MCMC") + xlab("-log lambda") + ylab("beta")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: Removed 750 rows containing non-finite values (stat_smooth).
```




```
# lasso using glmnet
plot_glmnet(lasso_2e)
```



As we can see from the regularization path, the betas estimated by Bayesian Adaptive Lasso and `glmnet` method are mostly consistent.

(g). Free λ and carry out a sensitivity analysis assessing the behavior of the posterior distribution of β and σ^2, as hyper parameters a and b are changed. Explain clearly the rationale you use to assess sensitivity and provide recommendations for the analysis of the `diabetes` data.

For this question we set different values of a and b and add another option in our function of **(e)**, generate λ with different a and b and get β.

```
# set different pairs of a and b
ab_2g <- data.frame(a = c(1, 1, 1, 5, 10),
                    b = c(1, 5, 10, 10, 10))

test_2g <- apply(ab_2g, 1,
              function(ab){
                return(gibbs_2e(init_2e, nsim = 10000, burn = 0.1,
                                a = ab[1], b = ab[2], sen_analysis = TRUE))})
# assess the behavior of beta
results_2g_beta <- lapply(test_2g,
                     function(x){apply(x$beta, 2,
                                       function(beta){
                                         quantile(beta, c(0.25, 0.5, 0.75))}
                                       )})
# assess the behavior of sigma^2
results_2g_sigmasq <- lapply(test_2g,
                     function(x){apply(x$sigma_sq %>% as.matrix, 2,
                                       function(sigma_sq){
                                         quantile(sigma_sq, c(0.25, 0.5, 0.75))}
                                       )})


# merge the result of beta and sigma^2
results_ci_2g_beta <- data.frame(Reduce("rbind", results_2g_beta))
results_ci_2g_sigma_sq <- data.frame(Reduce("rbind", results_2g_sigmasq))
ci_2g <- cbind(results_ci_2g_beta, results_ci_2g_sigma_sq)


colnames(ci_2g) <- c(colnames(dia2e_x), "sigma^2")

compare_2g <- matrix(NA, 11, 5)
for(i in 0:4){
  for(j in 1:11){
    compare_2g[j, i + 1] <- paste0(round(ci_2g[3 * i + 2, j], 2),"(",
                                   round(ci_2g[3 * i + 1, j], 2), ", ",
                                   round(ci_2g[3 * i + 3, j], 2), ")")
  }
}

rownames(compare_2g) <- c(colnames(dia2e_x), "sigma_sq")
colnames(compare_2g) <- paste0("a = ", ab_2g$a, ", b = ", ab_2g$b)

compare_2g %>%
  kbl(caption = NULL) %>%
  kable_classic(full_width = T, html_font = "Cambria")
```

	a = 1, b = 1	a = 1, b = 5	a = 1, b = 10	a = 5, b = 10	a = 10, b = 10
age	12.52(-47.96, 78.82)	12.52(-47.96, 78.82)	12.57(-47.96, 78.82)	0.04(-0.91, 1)	0.01(-0.65, 0.71)
sex	-66.09(-162.13, 3.35)	-66.1(-162.14, 3.35)	-66.23(-162.18, 3.35)	0.01(-1, 0.98)	0.01(-0.68, 0.71)
bmi	428.61(298.83, 557.93)	428.62(298.84, 557.94)	428.62(299.12, 558)	0.07(-0.91, 1.16)	0.04(-0.67, 0.8)
map	185.12(78.68, 305.64)	185.13(78.68, 305.65)	185.14(78.68, 305.81)	0.05(-0.89, 1.09)	0.04(-0.63, 0.77)
tc	-11.03(-98.49, 59.76)	-11.03(-98.5, 59.76)	-11.03(-98.57, 59.76)	0.04(-0.94, 1.06)	0.02(-0.71, 0.73)
ldl	-22.28(-109.6, 46.97)	-22.28(-109.6, 46.97)	-22.29(-109.61, 46.95)	0.02(-0.96, 1.03)	0.02(-0.68, 0.72)
hdl	-109.75(-226.94, -19.82)	-109.76(-226.95, -19.83)	-109.78(-227.08, -19.79)	-0.04(-1.07, 0.94)	-0.03(-0.77, 0.67)
tch	71.91(-10.26, 191.95)	71.91(-10.27, 191.95)	71.98(-10.27, 191.96)	0.05(-0.89, 1.1)	0.02(-0.66, 0.75)
ltg	337.7(202.05, 484.01)	337.7(202.07, 484.01)	337.69(202.09, 484.05)	0.06(-0.89, 1.1)	0.02(-0.64, 0.73)
glu	61.2(-9.19, 155.2)	61.21(-9.19, 155.21)	61.19(-9.15, 155.23)	0.04(-0.94, 1.05)	0.03(-0.68, 0.76)
sigma_sq	26568.45(25362.01, 27835.97)	26568.41(25362, 27835.98)	26568.37(25362, 27835.99)	29075.05(27780.29, 30473.79)	29072.55(27777.43, 30469.81)

As we can see, if we fix $a = 1$, the coefficient estimates look reasonable but a can heavily influence our estimation, therefore we need to set a to be 1. For b, since τ^2 is really large so the value of b can't influence the result that much, it still makes sense if we set b to be larger.