# Biostat 276 Homework 1

Zian ZHUANG

## Contents



## 1. Sampling from the Banana Distribution

### (a)

Describe how to simulate directly from $\pi(x1; x2)$.

**Answer**: Firstly, we can calculate the conditional probability of x2 given X1 and marginal probability of X1.

$$P(x_2|x_1) \propto \exp\{-\frac{(x_2 - 2(x_1^2 - 5))^2}{2}\}$$

note that P(X_2|X_1) is proportional to the normal kernel. Thus we know,

$$\begin{aligned} P(x_2|x_1) &= c * \exp\{-\frac{(x_2 - 2(x_1^2 - 5))^2}{2}\} \\ &= \frac{1}{\sqrt{2\pi}} \exp\{-\frac{(x_2 - 2(x_1^2 - 5))^2}{2}\} \end{aligned}$$

And we can calculate P(X_1),

$$\begin{aligned} lclP(x_1) \propto & \int \exp\{-\frac{x_1^2}{2}\} \exp\{-\frac{(x_2 - 2(x_1^2 - 5))^2}{2}\} dx_2 \\ \propto & \exp\{-\frac{x_1^2}{2}\} \int \exp\{-\frac{(x_2 - 2(x_1^2 - 5))^2}{2}\} dx_2 \\ \propto & \exp\{-\frac{x_1^2}{2}\} * \sqrt{2\pi} \end{aligned}$$

note that P(X_1) is proportional to the normal kernel. Thus we know,

$$\begin{aligned} lclP(x_1) = & c * \exp\{-\frac{x_1^2}{2}\} * \sqrt{2\pi} \\ = & \frac{1}{\sqrt{2\pi}} \exp\{-\frac{x_1^2}{2}\} \end{aligned}$$

Then we can use inverse transformation to generate X_1 and X_2.

Firstly generate uniformly distributed $u_1$, then apply Box-Muller transform to obtain $x_1 \sim N/(0,1)$.

Then we can generate uniformly distributed $u_2$, then apply Box-Muller transform to obtain $x_2 \sim N/(2(x_1^2 - 5), 1)$.

code source

```
nsim <- 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(1996)
x_q1 <- box_muller(nsim)
x_q2 <- box_muller(nsim)+2*(x_q1^2-5)
```

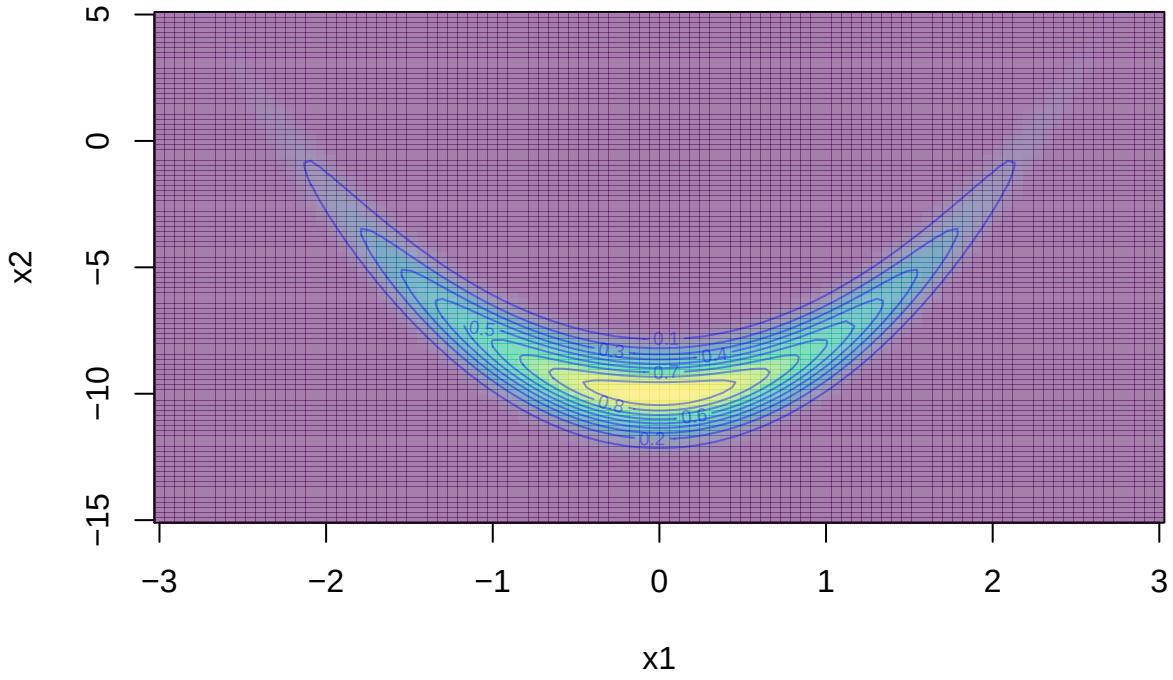
### (b)

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.

Firstly we can visualize the distribution of $x_1$ and $x_2$

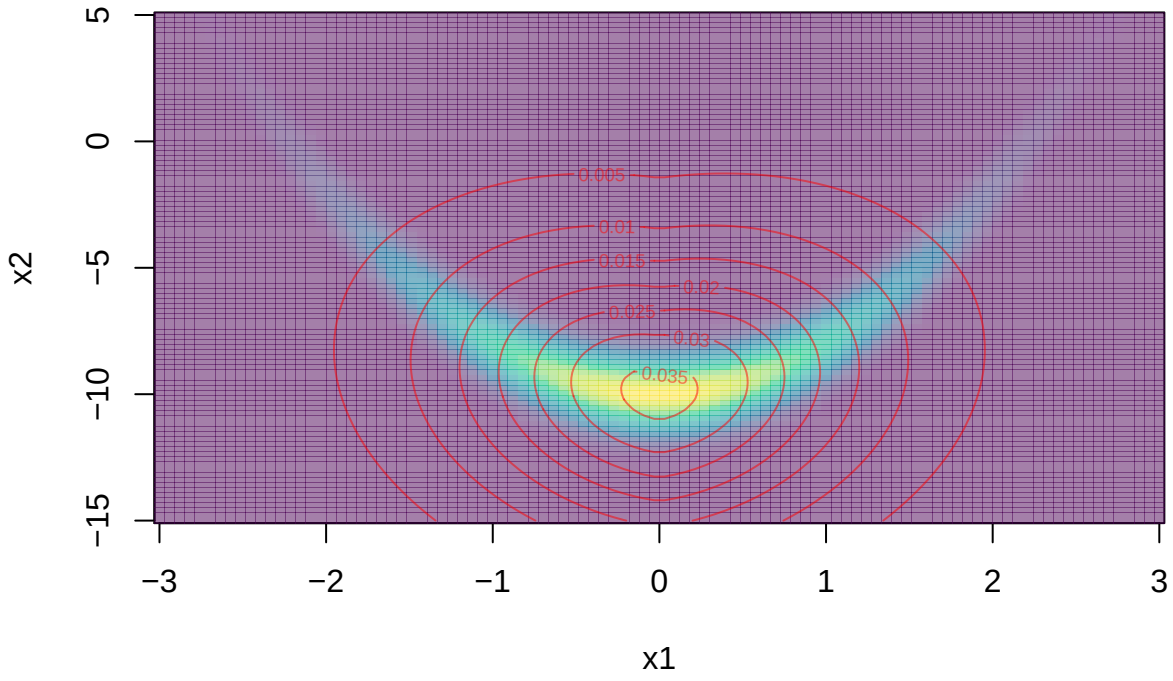
```
nsim <- 100
x1 <- seq(-3,3,length.out=nsim)
x2 <- seq(-15,5,length.out=nsim)
f <- matrix(0, nsim, nsim)

.banana <- function(x1, x2){
  out <- exp(-x1^2/2)*exp(-(x2-2*(x1^2-5))^2/2)
  return(out)
}

for(i in 1:nsim){
  for(j in 1:nsim){
    temp1 <- x1[i]
    temp2 <- x2[j]
    f[i,j] <- .banana(temp1, temp2)
  }
}


image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
```

```
      xlab=expression("x1"),
      ylab=expression("x2"))
contour(x1, x2, f, add=T, col=rgb(0,0,1,alpha=0.4))
```




Since the f is in exp family, it is log-concave and we know that we can always find a majorizing function. According to the previous density plot, we would like to try multivariate t distribution. note that here we use two half of multivariate t distributions that are symmetric with respect to x=0 (y axis), in order to approach the banana distribution more efficiently.

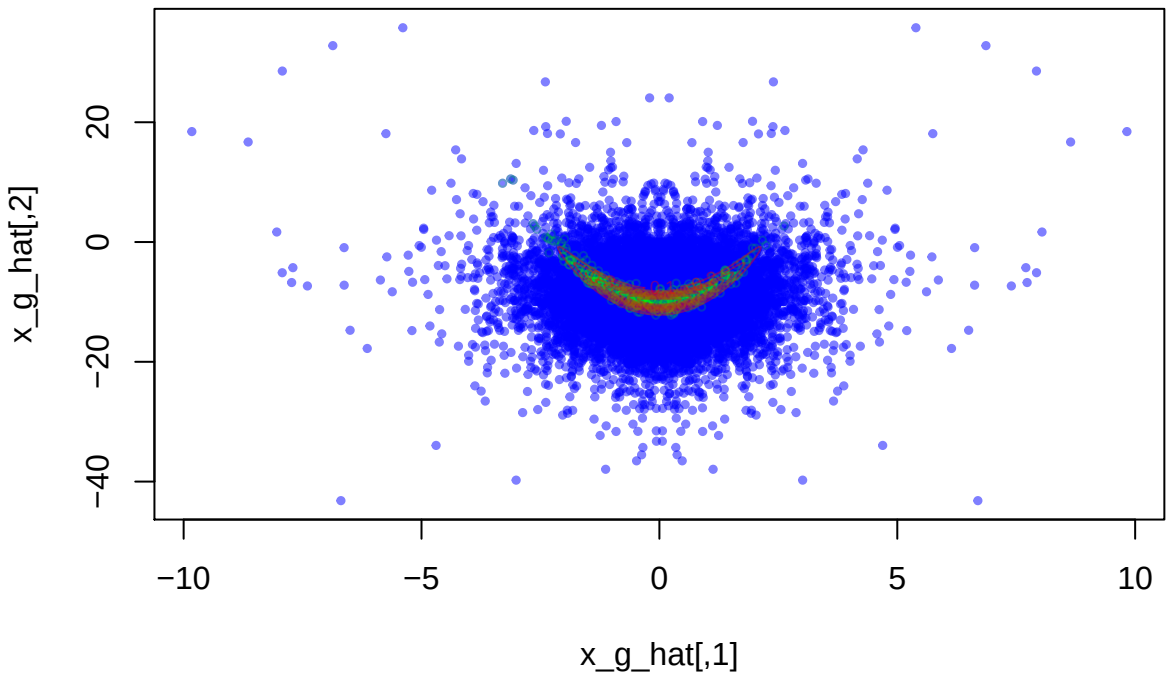
```
library(mvtnorm)
delta <- c(0, -10)
gb <- matrix(0,nsim,nsim)
Sigma <- matrix(c(1,0.9,0.9,20),2,2)
for(i in 1:100){
  for(j in 1:100){
    betai   = c(abs(x1[i]), x2[j])
    gb[i,j] = dmvt(betai, delta = delta, Sigma, df = 5, log=FALSE)
  }
}

image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
      xlab=expression("x1"),
      ylab=expression("x2"))
contour(x1, x2, gb, add=T, col=rgb(1,0,0,alpha=0.5))
```




AR Sampling function

```
set.seed(1996)
nsim   = 15000
Sigma1 = Sigma
nu     = 5
x_g = rmvt(n = nsim, sigma = Sigma1, df = nu, delta = delta)
#plot(x_g, col=rgb(0,0,1, alpha=0.5), pch=19, cex=0.5)
x_g_hat <- x_g[which(x_g[,1]>=0),]
x_g_hat <- rbind(x_g_hat, x_g_hat)
x_g_hat[(dim(x_g_hat)[1]/2+1):dim(x_g_hat)[1],1] <-
  -abs(x_g_hat[(dim(x_g_hat)[1]/2+1):dim(x_g_hat)[1],1])
nsim <- dim(x_g_hat)[1]

plot(x_g_hat, col=rgb(0,0,1, alpha=0.5), pch=19, cex=0.5)
#
# AR Sampling
#
g = dmvt(x_g_hat, sigma=Sigma1, df=nu, delta=delta, log=FALSE)
#
fx <- rep(NA, nsim)
for(i in 1:nsim){
  temp1 <- x_g_hat[i,1]
  temp2 <- x_g_hat[i,2]
  fx[i] <- .banana(temp1, temp2)
}
```

```
f1 = fx
M  = 300
u  = runif(nsim)
#
c1 = f1/g * 1/M
x_AR = x_g_hat[c1 >= u,]
points(x_AR, col=rgb(0,1,0,alpha=0.2), cex=0.5)
contour(x1, x2, f, add=T, col=rgb(1,0,0,alpha=0.4))
```



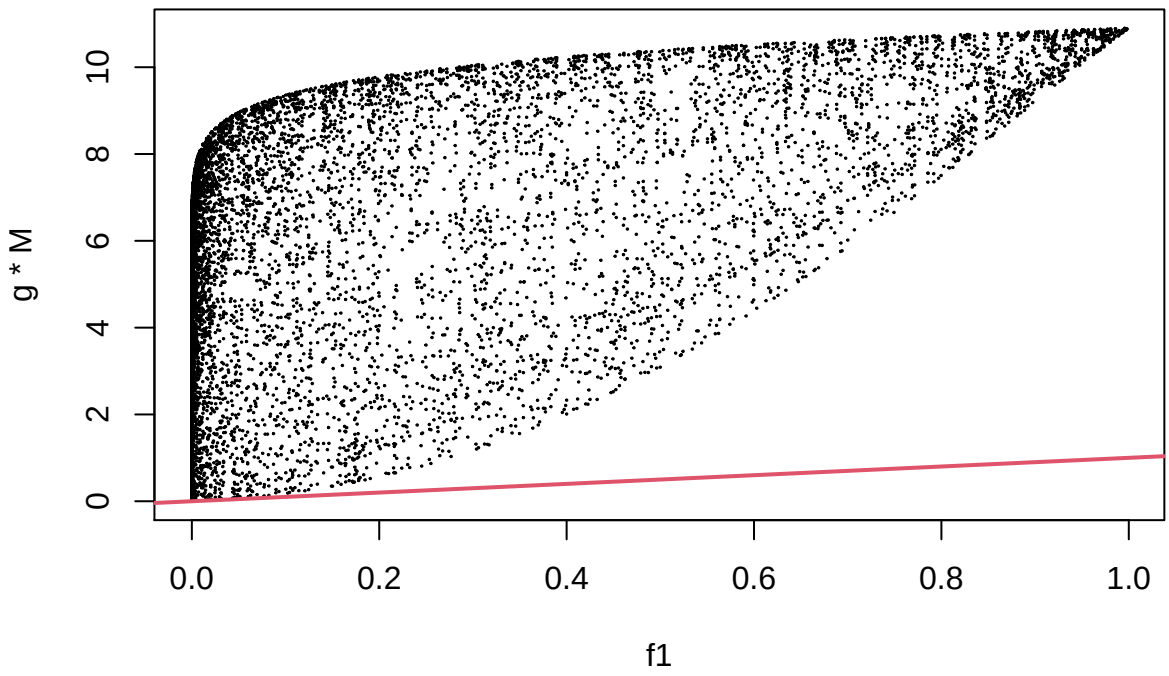

```
#efficiency
dim(x_AR)[1]/nsim
```

```
## [1] 0.02580731
```

```
#model check
which(f1/(g*M)>1) %>% length
```

```
## [1] 0
```

```
plot(f1, g*M, cex=0.1)
abline(a=0, b=1, col=2, lwd=2)
```



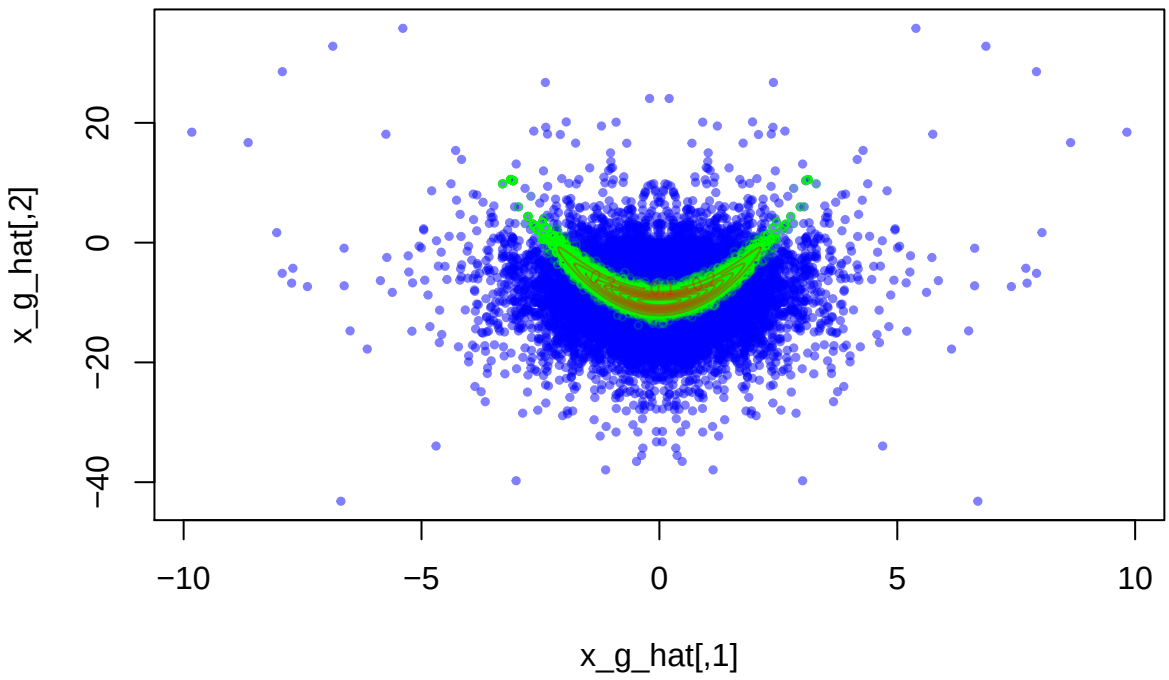

The model has a 2.58% accept rate. And $\frac{fx}{gx*M} \leq 1$ is hold.

**(c)**

Describe how to simulate from $\pi(x_1; x_2)$ using sampling importance resampling. Implement your algorithm and discuss sampling efficiency in relation to the chosen instrument distribution.

- 1) generate a sample
- 2) calculate importance weights
- 3) normalize calculated weights
- 4) re-sample

```
# Using the same simulated values as for the AR algorithm
plot(x_g_hat, col=rgb(0,0,1, alpha=0.5), pch=19, cex=0.5)
# Calculate importance weights
w = f1/g
# normalized weights
w_star = w/sum(w)
samp_ind = seq(1,nsim)
x_ind = sample(samp_ind, prob=w_star, replace=TRUE)
x_IS  = x_g_hat[x_ind,]

points(x_IS,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
contour(x1, x2, f, add=T, col=rgb(1,0,0,alpha=0.4))
```




**(d)**

Construct a Markov Chain to sample from $\pi(x1; x2)$ using the Metropolis Hastings algorithm. Compare convergence and mixing for different proposal jump sizes.

Random-walk proposal:

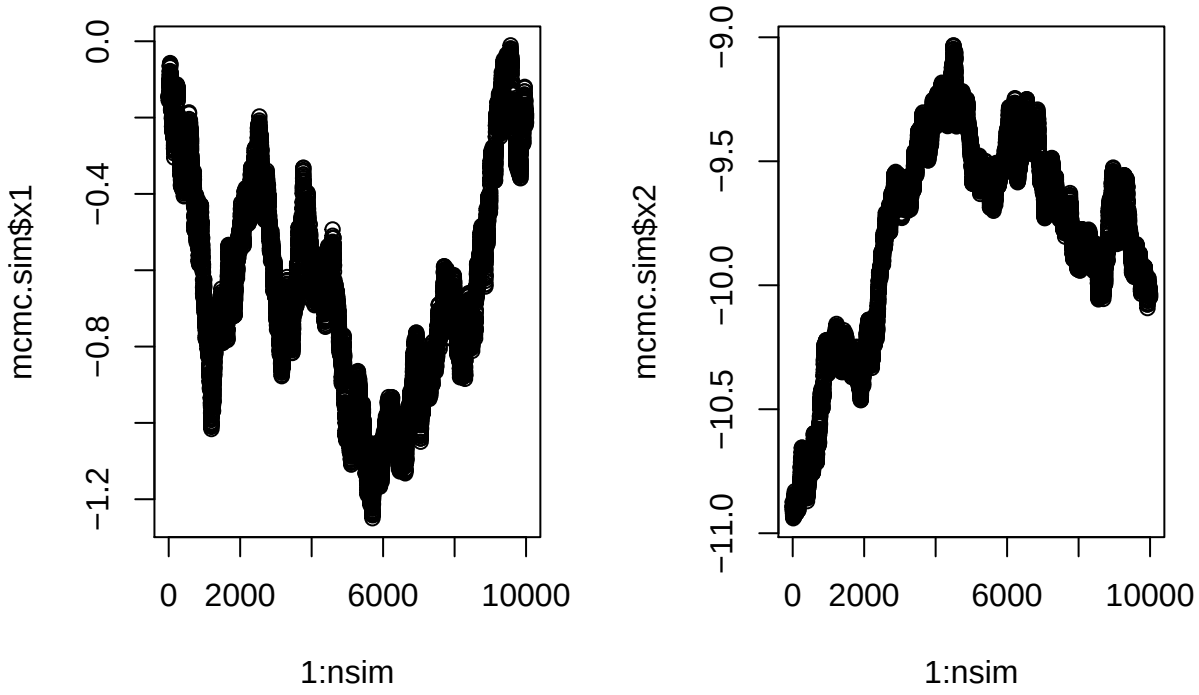
- $x_1^*|x_1 \sim N(x_1, \eta)$
- $x_2^*|x_2 \sim N(x_2, \delta)$

```
mh.sim <- function(nsim=1000, burn=0, eta=3, delta=5, seed=1996){
  set.seed(seed)
  x1 <- 0
  x2 <- -10
  nsim.total <- nsim*(1.0 + burn)
  burn.num <- nsim*burn
  x1.ch <- vector()
  x2.ch <- vector()

  for(i in 1:nsim.total){
    x1_temp <- rnorm(1,x1,eta)
    x2_temp <- rnorm(1,x2,delta)
    P0 <- .banana(x1, x2) %>% log
    P1 <- .banana(x1_temp, x2_temp) %>% log
```

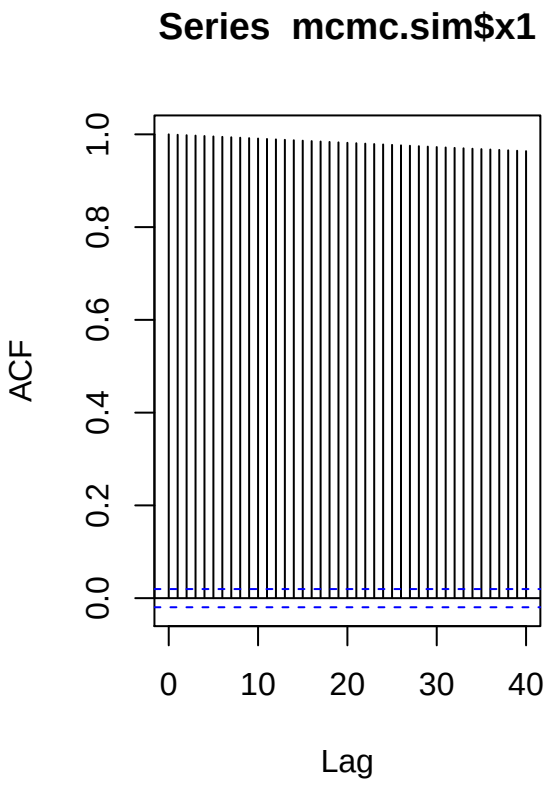
```
    ratio <- P1 - P0
    if(log(runif(1)) < ratio){
      x1 <- x1_temp
      x2 <- x2_temp
    }
    if(i > burn.num){
      i1 <- i - burn.num
      x1.ch[i1] = x1
      x2.ch[i1] = x2
    }
  }
  return(list(x1=x1.ch, x2=x2.ch))
}
```

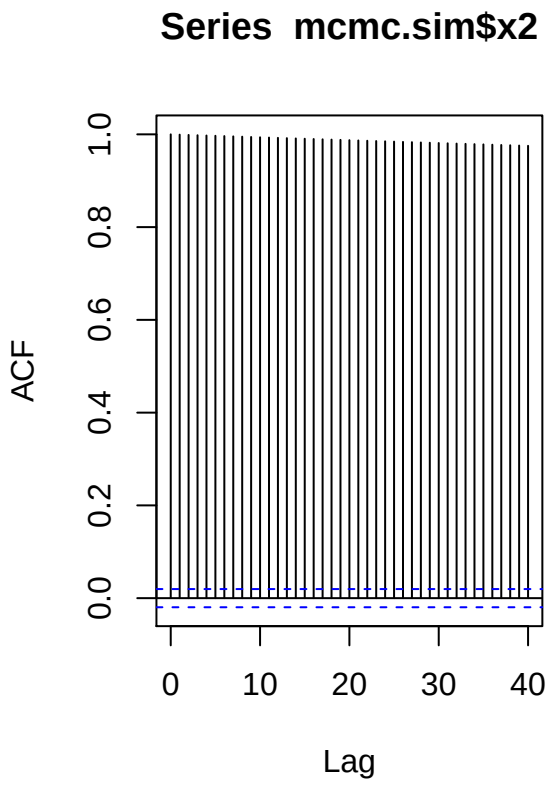
simulation begin

```
burn=0.3
nsim=10000

# scenario 1
eta <- 0.01
delta <- 0.01
mcmc.sim <- mh.sim(nsim=nsim, eta=eta, delta=delta, burn=burn)
# plot
par(mfrow=c(1,2))
plot(1:nsim, mcmc.sim$x1)
plot(1:nsim, mcmc.sim$x2)
```

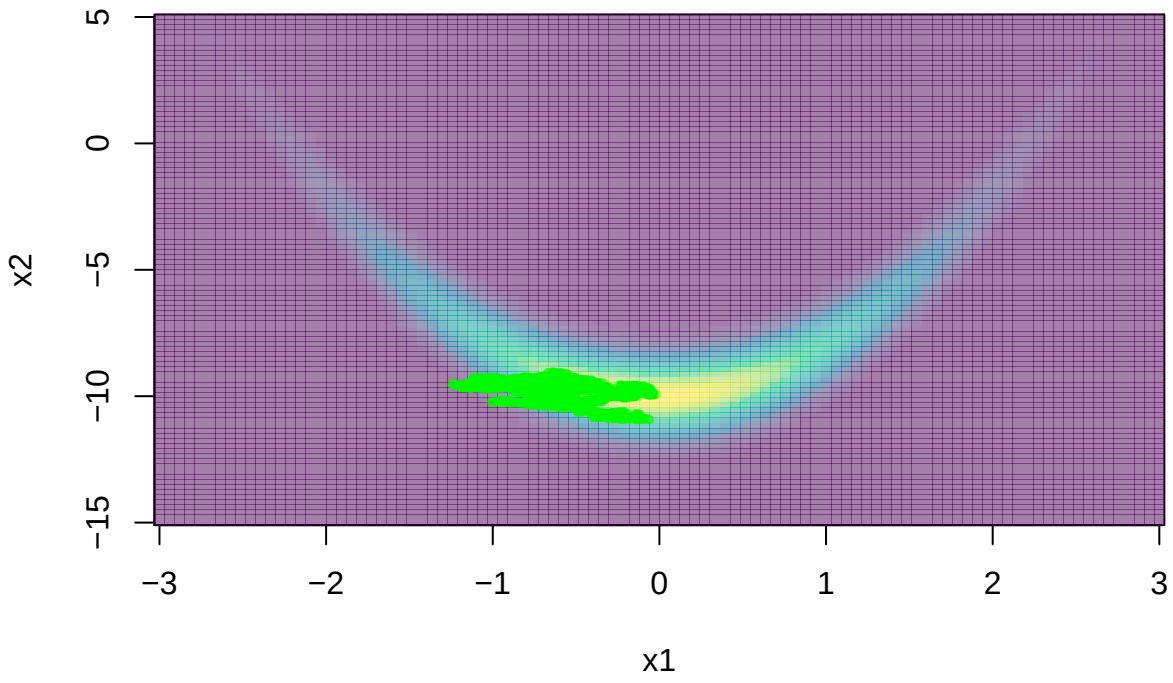



```
acf(mcmc.sim$x1)
acf(mcmc.sim$x2)
```



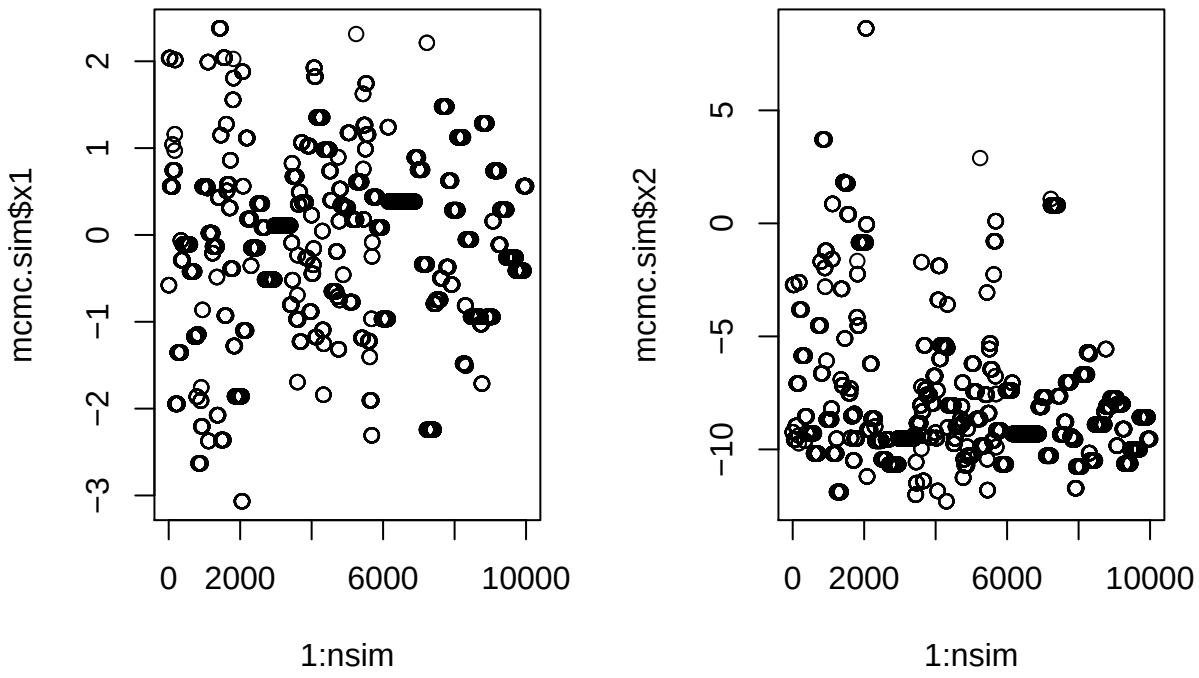




```
par(mfrow=c(1,1))
image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
      xlab=expression("x1"),
      ylab=expression("x2"))
points(mcmc.sim$x1, mcmc.sim$x2,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
```

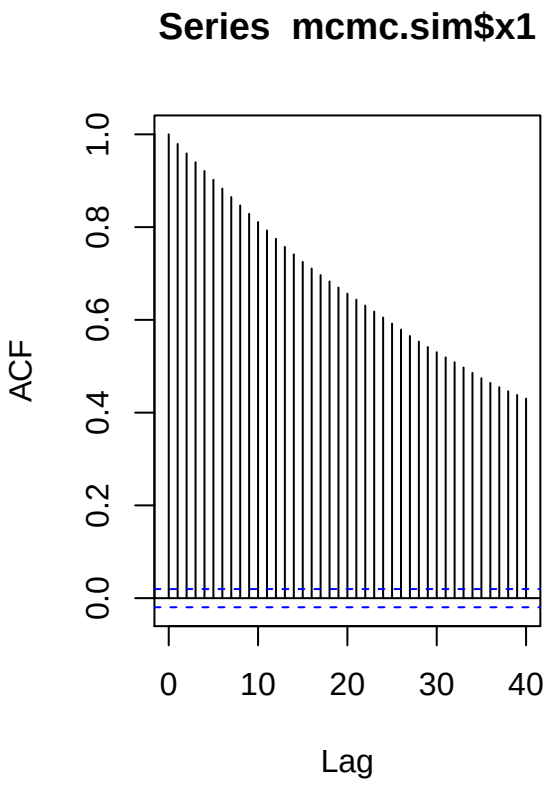


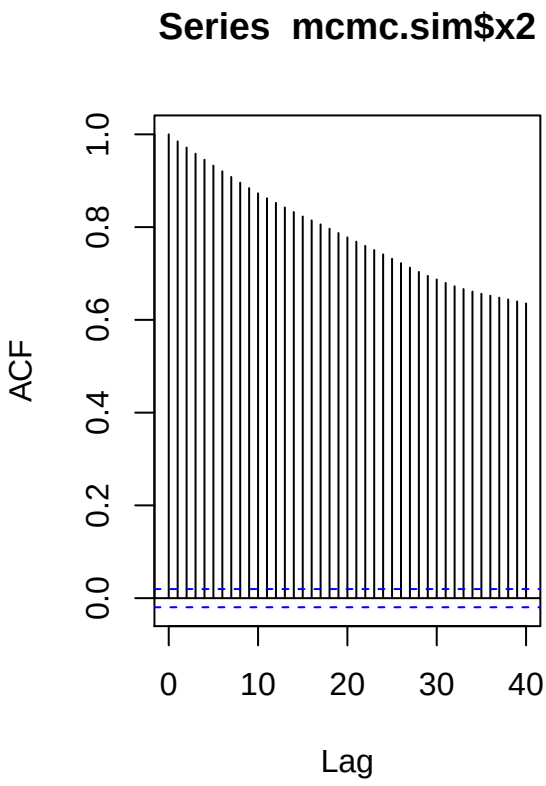

We can tell from the plot that the chain wander around a lot with eta=delta=0.01. Then we tried larger values.

```
# scenario 2
eta <- 10
delta <- 10
mcmc.sim <- mh.sim(nsim=nsim, eta=eta, delta=delta, burn=burn)
# plot
par(mfrow=c(1,2))
plot(1:nsim, mcmc.sim$x1)
plot(1:nsim, mcmc.sim$x2)
```

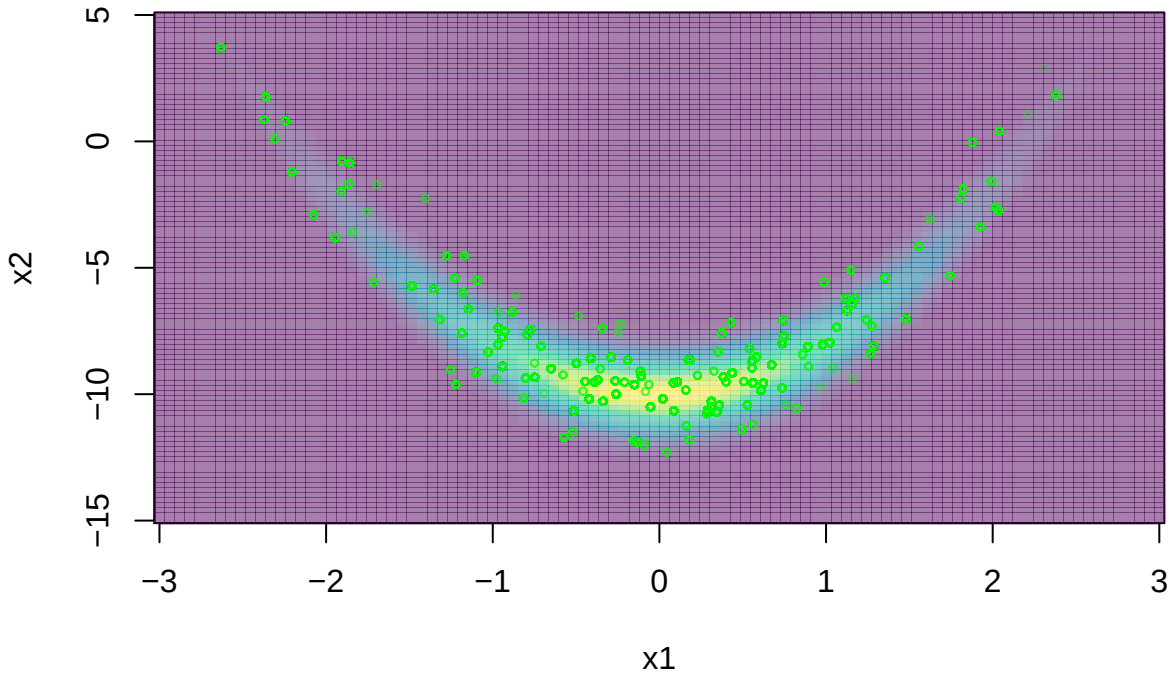



```
acf(mcmc.sim$x1)
acf(mcmc.sim$x2)
```



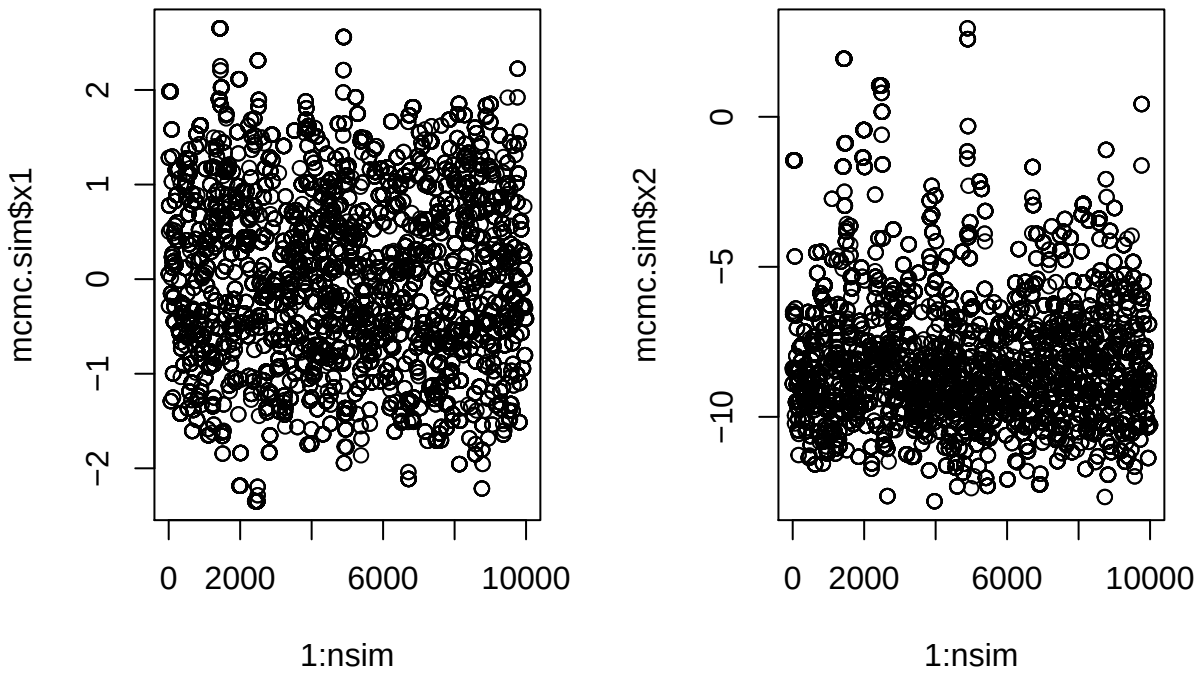




```
par(mfrow=c(1,1))
image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
      xlab=expression("x1"),
      ylab=expression("x2"))
points(mcmc.sim$x1, mcmc.sim$x2,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
```

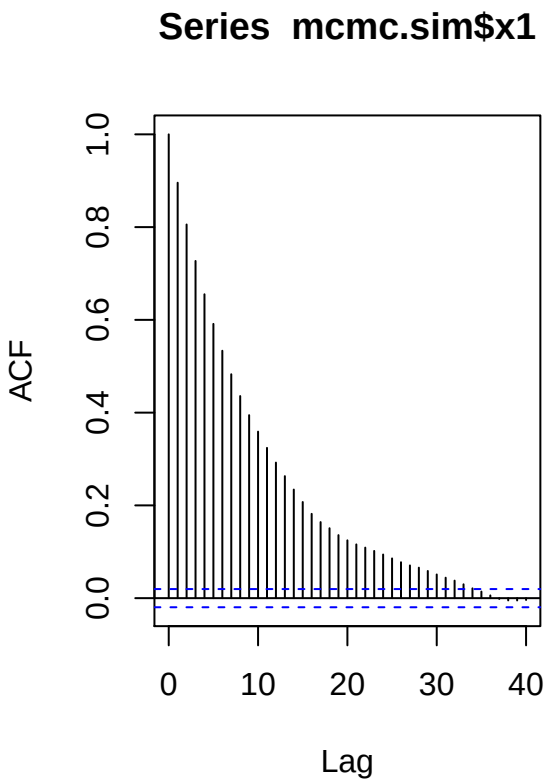


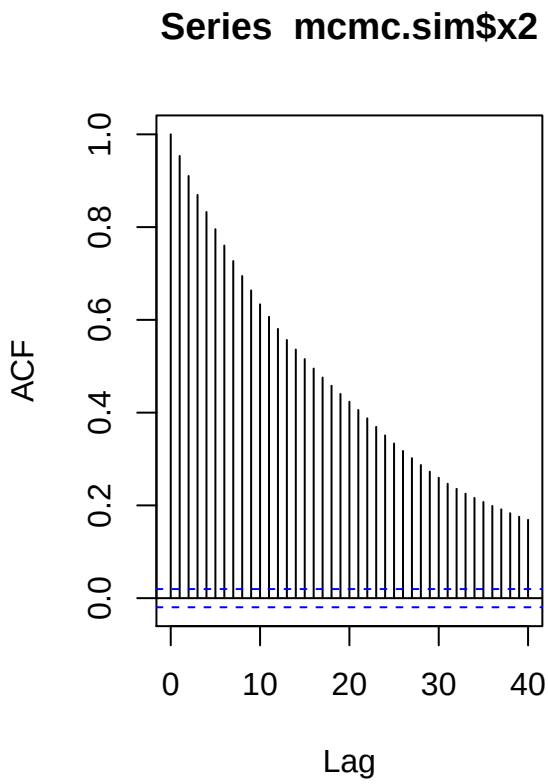

We can tell from the plot that some points of the chain are unchanged during the iteration with eta=delta=10. Then we tried smaller values.

```
# scenario 3
eta <- 3
delta <- 3
mcmc.sim <- mh.sim(nsim=nsim, eta=eta, delta=delta, burn=burn)
# plot
par(mfrow=c(1,2))
plot(1:nsim, mcmc.sim$x1)
plot(1:nsim, mcmc.sim$x2)
```

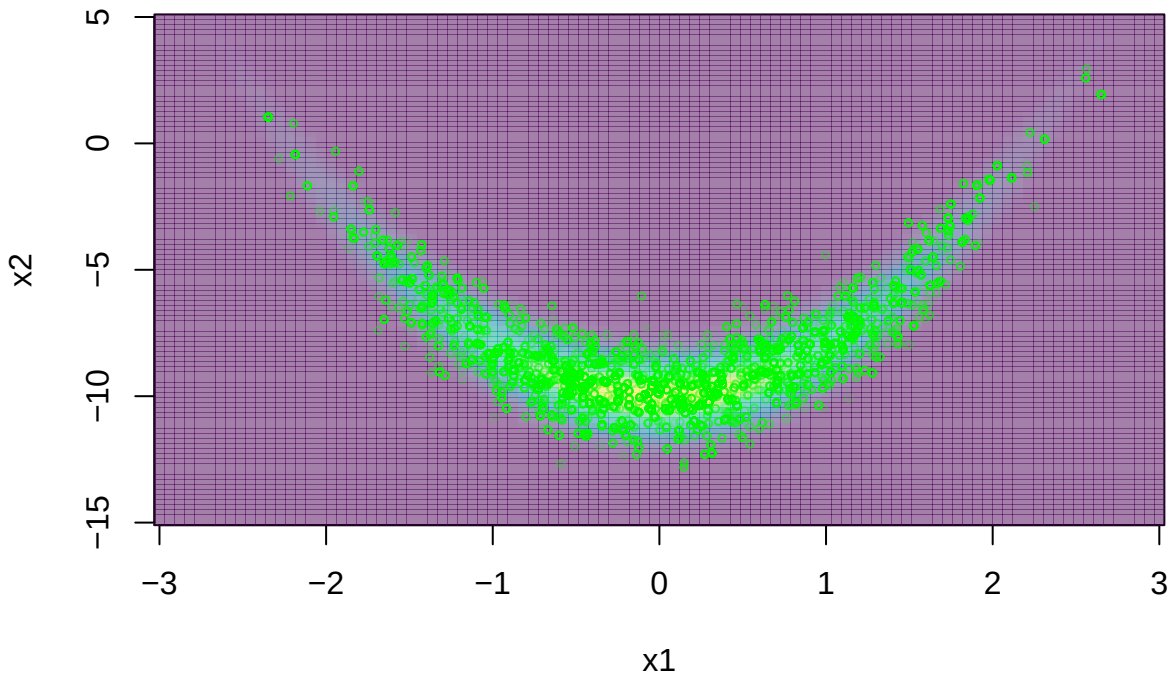



```
acf(mcmc.sim$x1)
acf(mcmc.sim$x2)
```




```
par(mfrow=c(1,1))
image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
      xlab=expression("x1"),
      ylab=expression("x2"))
points(mcmc.sim$x1, mcmc.sim$x2,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
```




```
mcmc.sim.d<-mcmc.sim
```

We can clearly tell from the plot that the chain's convergence are improved with eta=delta=3. ACF plot also shows that autocorrelation approach to 0 faster.

### (e)

Construct a Markov Chain to sample from $\pi(x1; x2)$ updating iteratively $\pi(x1|x2)$ and $\pi(x2|x1)$ in a component-wise fashion, using the Metropolis Hastings algorithm. Compare convergence and mixing for different proposal jump sizes.

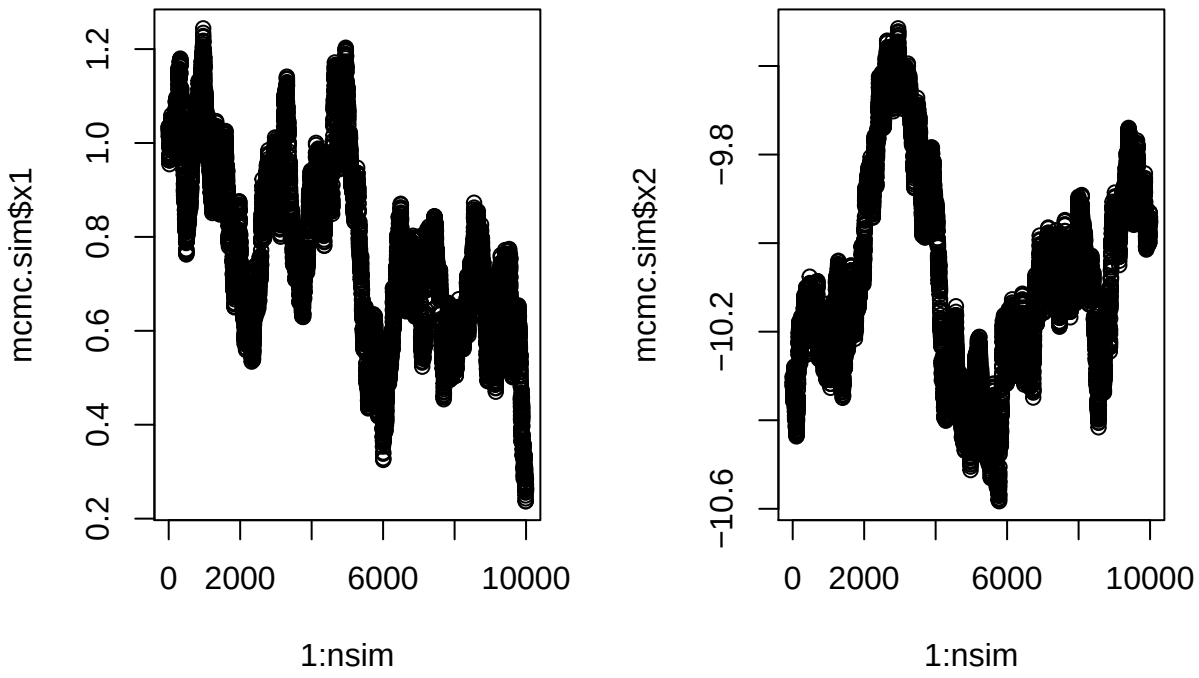
In this method, we tried to sample x1 and x2 iteratively by Metropolis Hastings algorithm.

```
mh.it.sim <- function(nsim=1000, burn=0, eta=3, delta=5, seed=1996){
  set.seed(seed)
  x1 <- 0
  x2 <- -10
  nsim.total <- nsim*(1.0 + burn)
  burn.num <- nsim*burn
  x1.ch <- vector()
  x2.ch <- vector()

  for(i in 1:nsim.total){
    x1_temp <- rnorm(1,x1,eta)
    x2_temp <- rnorm(1,x2,delta)
```

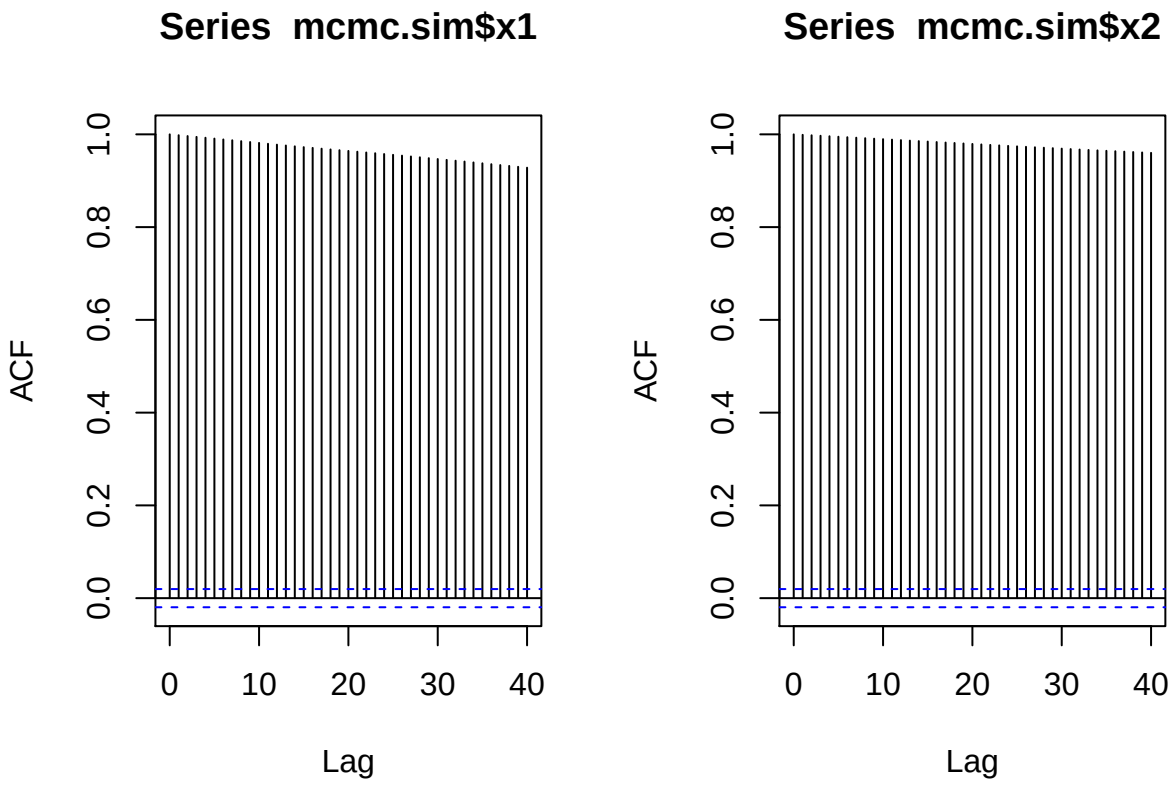
```
    P0 <- .banana(x1, x2) %>% log
    # x1|x2
    P1 <- .banana(x1_temp, x2) %>% log
    ratio <- P1 - P0
    if(log(runif(1)) < ratio){
      x1 <- x1_temp
      P0 <- P1
    }
    # x2|x1
    P1 <- .banana(x1, x2_temp) %>% log
    ratio <- P1 - P0
    if(log(runif(1)) < ratio){
      x2 <- x2_temp
    }
    # burn in
    if(i > burn.num){
      i1 <- i - burn.num
      x1.ch[i1] = x1
      x2.ch[i1] = x2
    }
  }
  return(list(x1=x1.ch, x2=x2.ch))
}
```

simulation begin, using same parameters setting as last question.

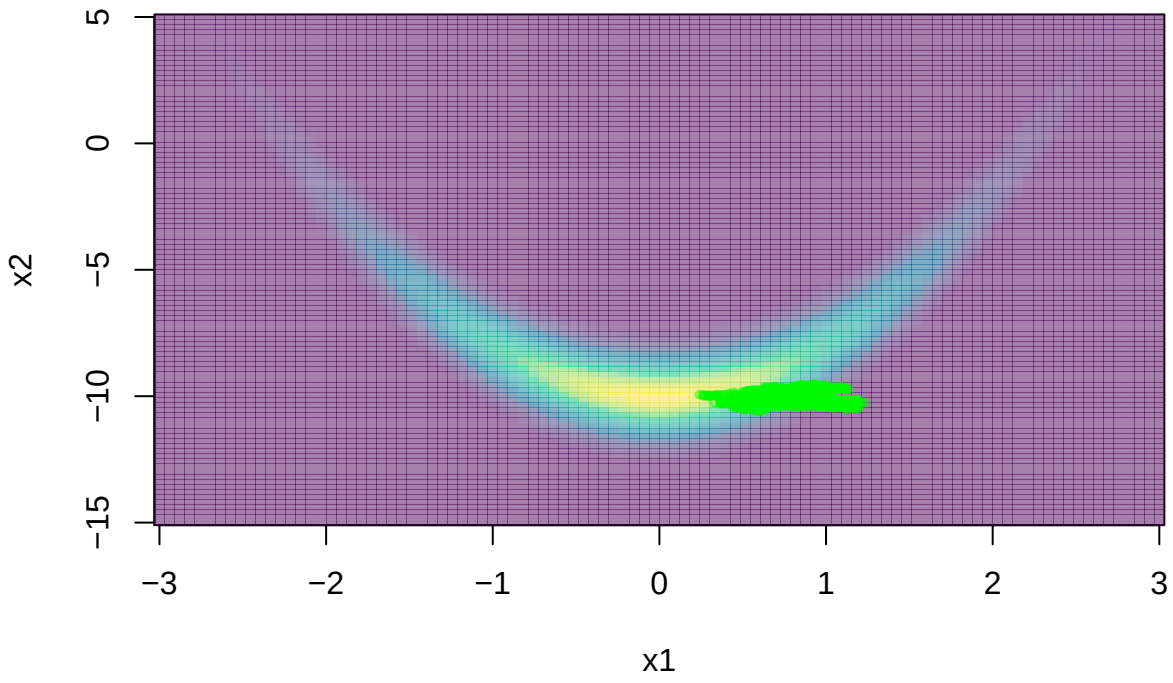
```
burn=0.3
nsim=10000

# scenario 1
eta <- 0.01
delta <- 0.01
mcmc.sim <- mh.it.sim(nsim=nsim, eta=eta, delta=delta, burn=burn)
# plot
par(mfrow=c(1,2))
plot(1:nsim, mcmc.sim$x1)
plot(1:nsim, mcmc.sim$x2)
```



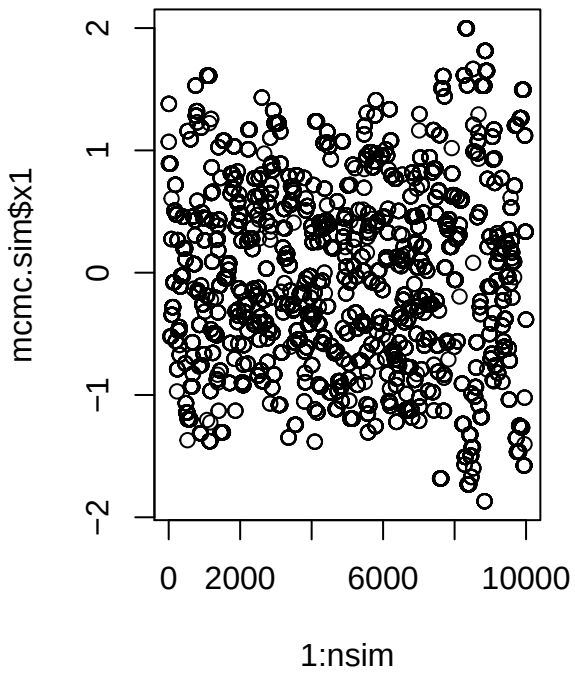

```
acf(mcmc.sim$x1)
acf(mcmc.sim$x2)
```

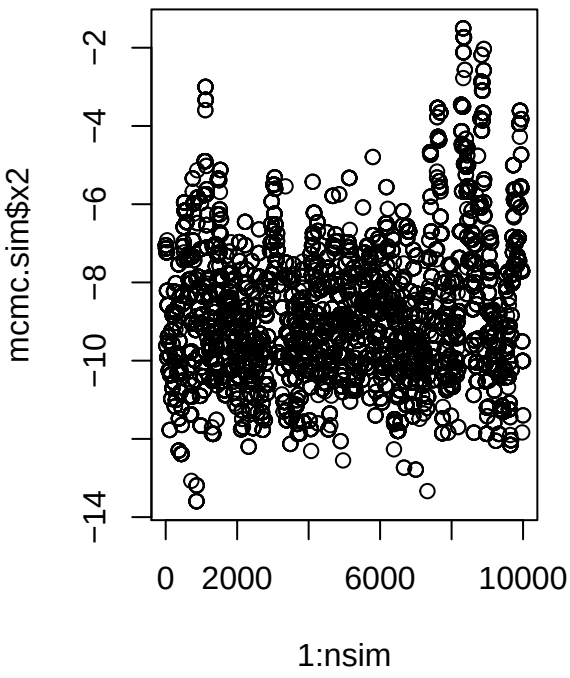



```
par(mfrow=c(1,1))
image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
      xlab=expression("x1"),
      ylab=expression("x2"))
points(mcmc.sim$x1, mcmc.sim$x2,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
```

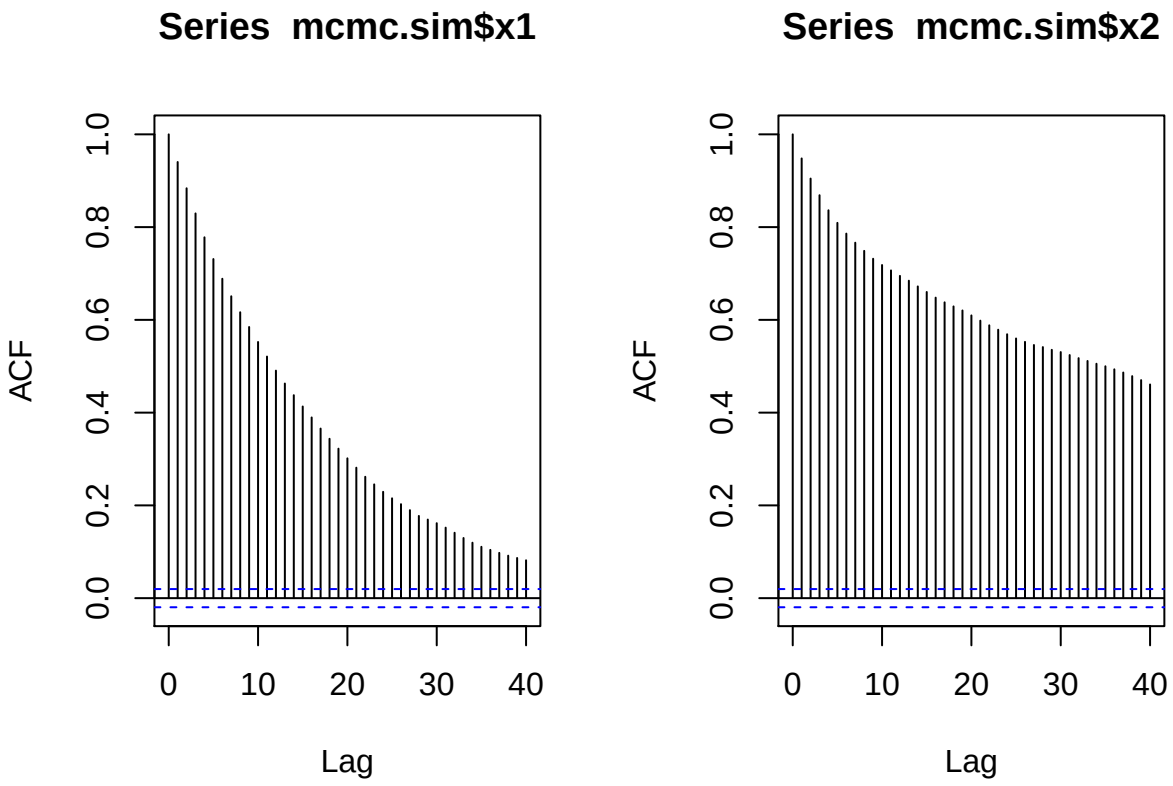



```
# scenario 2
eta <- 10
delta <- 10
mcmc.sim <- mh.it.sim(nsim=nsim, eta=eta, delta=delta, burn=burn)
# plot
par(mfrow=c(1,2))
plot(1:nsim, mcmc.sim$x1)
plot(1:nsim, mcmc.sim$x2)
```

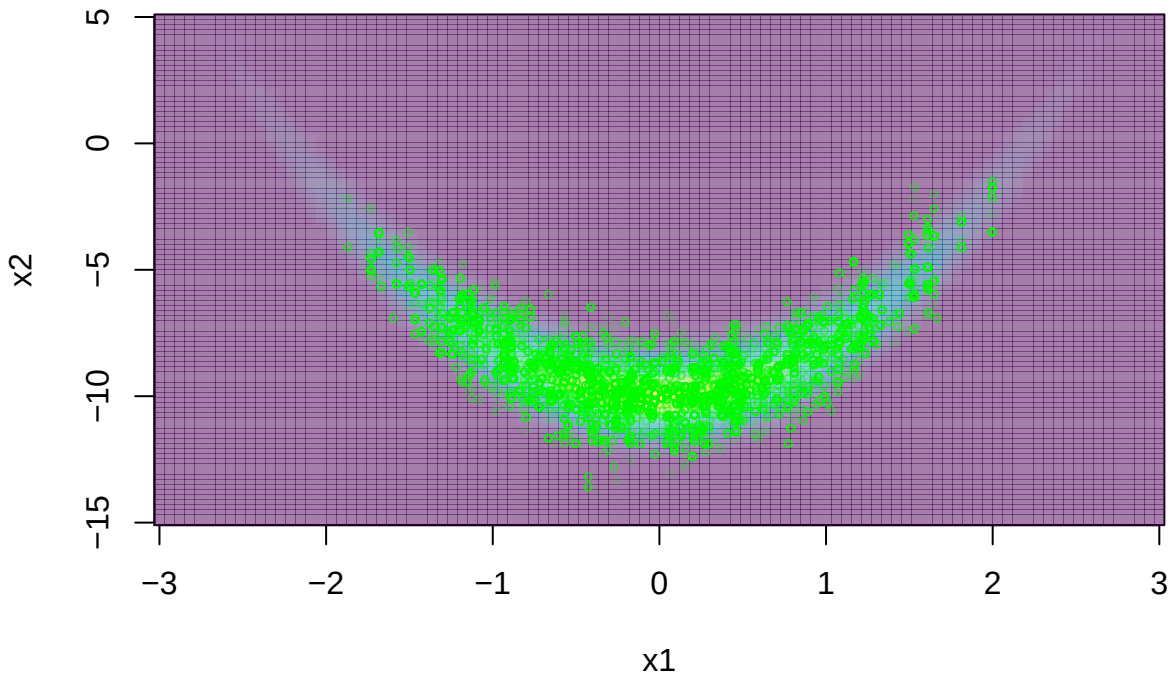


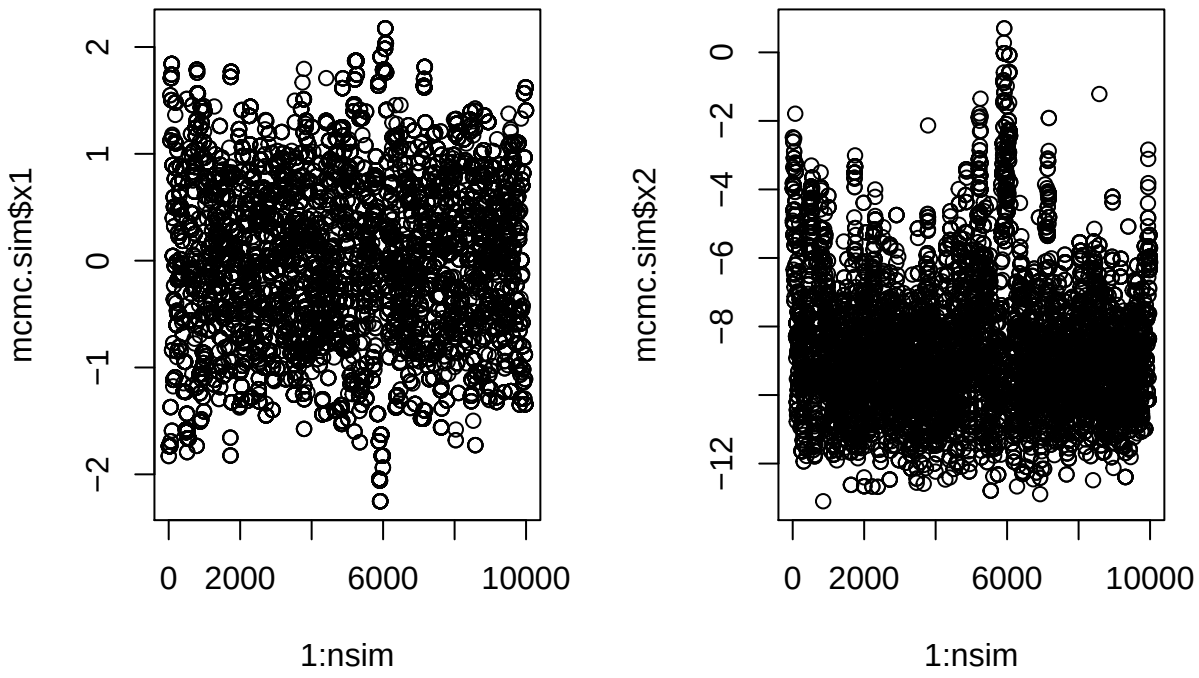

```
acf(mcmc.sim$x1)
acf(mcmc.sim$x2)
```



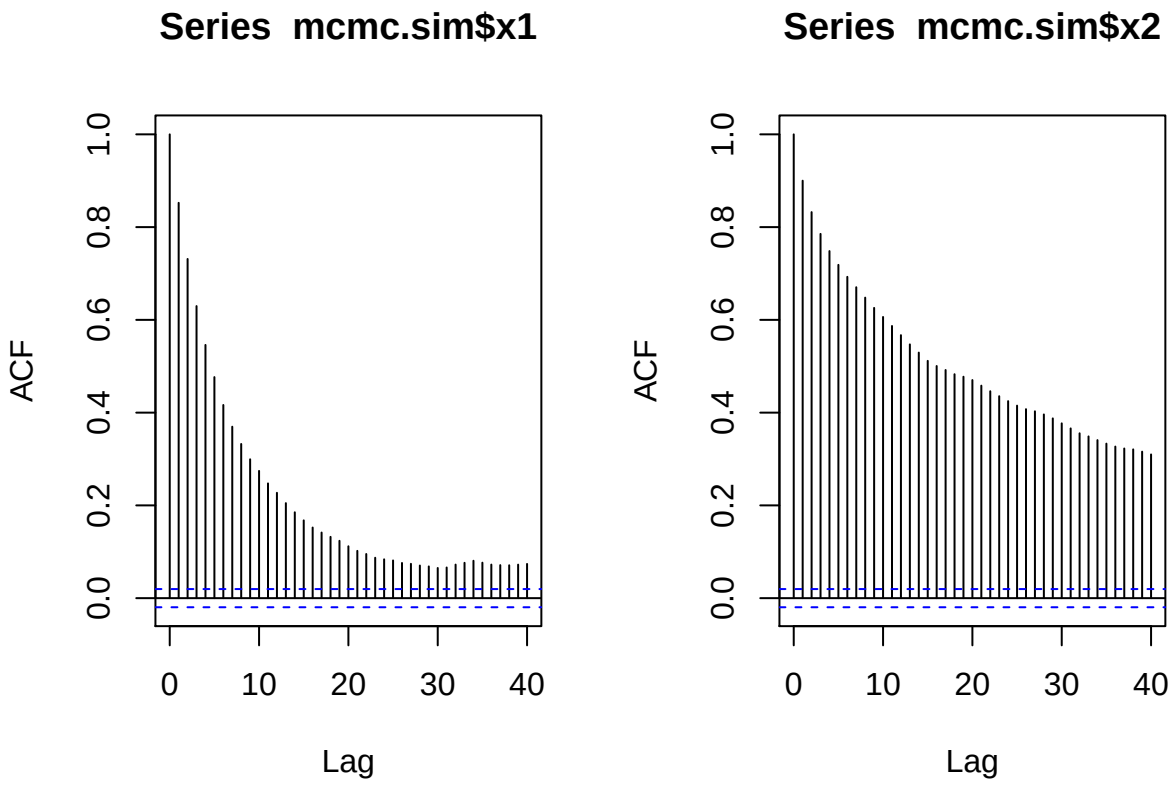

```
par(mfrow=c(1,1))
image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
      xlab=expression("x1"),
      ylab=expression("x2"))
points(mcmc.sim$x1, mcmc.sim$x2,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
```



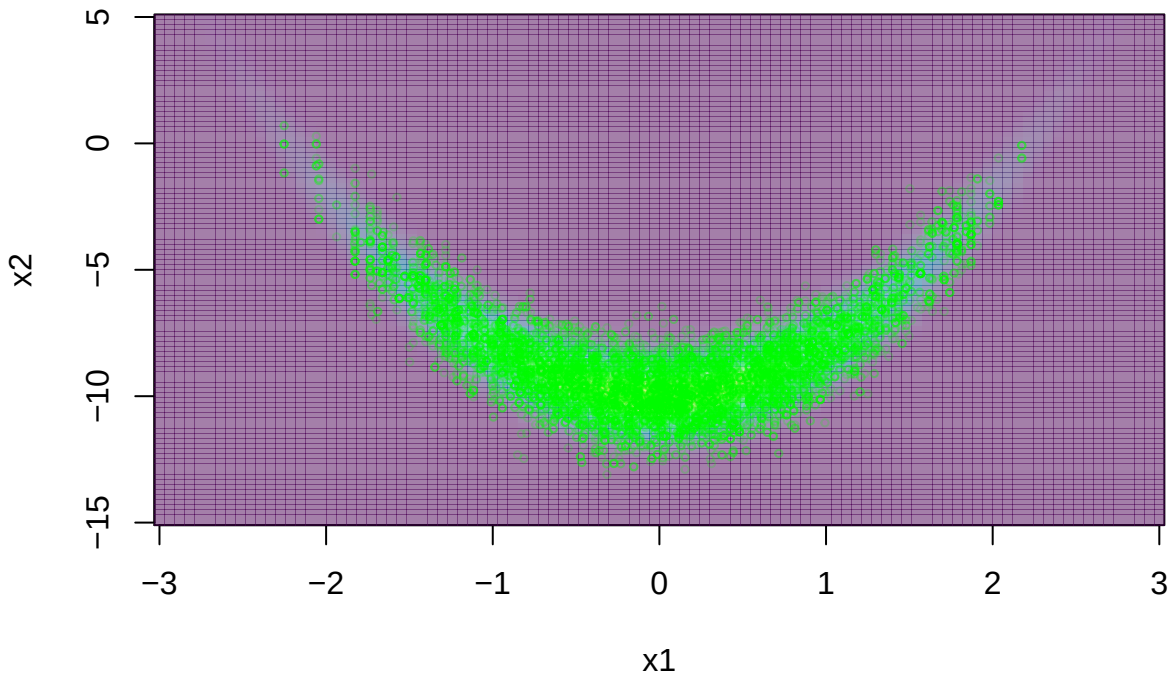

```
burn=0.3
nsim=10000

# scenario 3
eta <- 3
delta <- 3
mcmc.sim <- mh.it.sim(nsim=nsim, eta=eta, delta=delta, burn=burn)
# plot
par(mfrow=c(1,2))
plot(1:nsim, mcmc.sim$x1)
plot(1:nsim, mcmc.sim$x2)
```




```
acf(mcmc.sim$x1)
acf(mcmc.sim$x2)
```




```
par(mfrow=c(1,1))
image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
      xlab=expression("x1"),
      ylab=expression("x2"))
points(mcmc.sim$x1, mcmc.sim$x2,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
```




```
mcmc.sim.e<-mcmc.sim
```

We can clearly tell from the plot that the chain converge well with eta=delta=10 and eta=delta=3 as well. ACF plot also shows that autocorrelation approach to 0 fast. Compared to the method that generates X1 and X2 at the same time, this method provides more flexibility on the eta and delta setting.

### (f)

Repeat (e) using component-wise Gibbs sampling.

From 1 we know that:

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

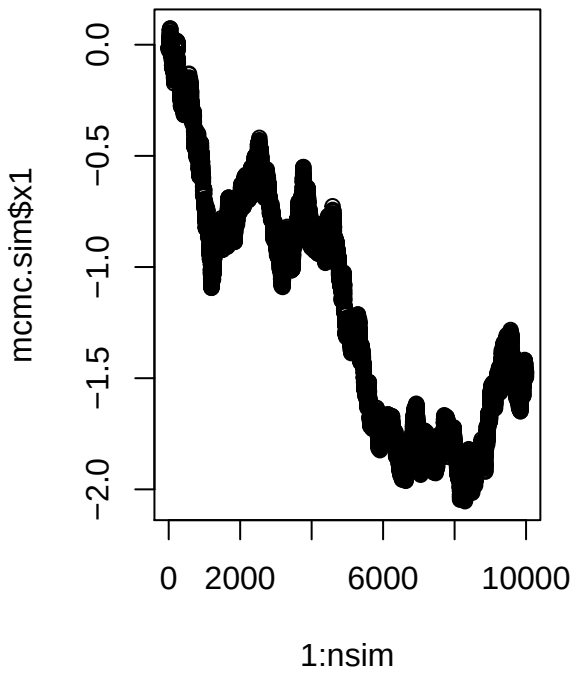
Then we can use component-wise Gibbs sampling:

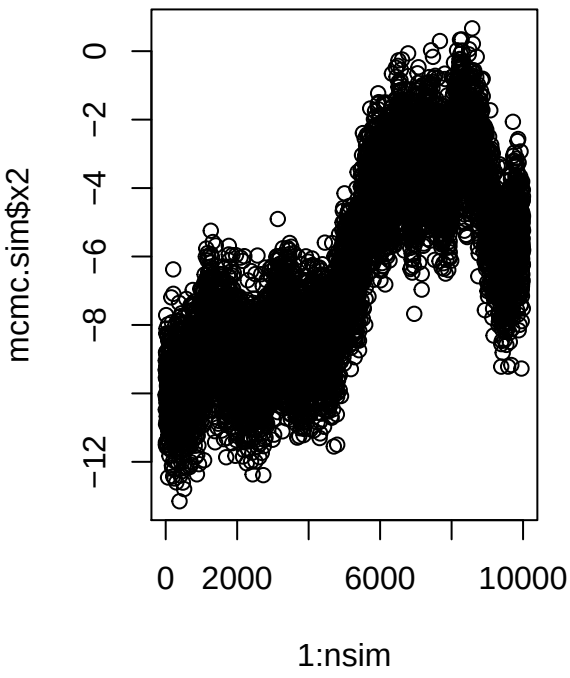
```
mh.gib.sim <- function(nsim=1000, burn=0, eta=3, seed=1996){
  set.seed(seed)
  x1 <- 0
  x2 <- -10
  nsim.total <- nsim*(1.0 + burn)
  burn.num <- nsim*burn
  x1.ch <- vector()
  x2.ch <- vector()
```

```
  for(i in 1:nsim.total){
    x1_temp <- rnorm(1,x1,eta)
    P0 <- .banana(x1, x2) %>% log
    # x1|x2
    P1 <- .banana(x1_temp, x2) %>% log
    ratio <- P1 - P0
    if(log(runif(1)) < ratio){
      x1 <- x1_temp
    }
    # x2|x1
    x2 <- rnorm(1, 2*(x1^2-5), 1)
    # burn in
    if(i > burn.num){
      i1 <- i - burn.num
      x1.ch[i1] = x1
      x2.ch[i1] = x2
    }
  }
  return(list(x1=x1.ch, x2=x2.ch))
}
```

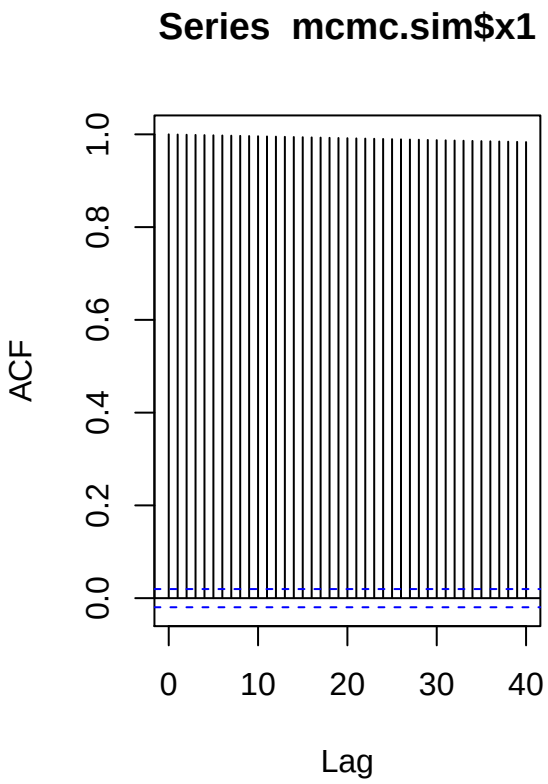
simulation begin, using same parameters setting as last question.

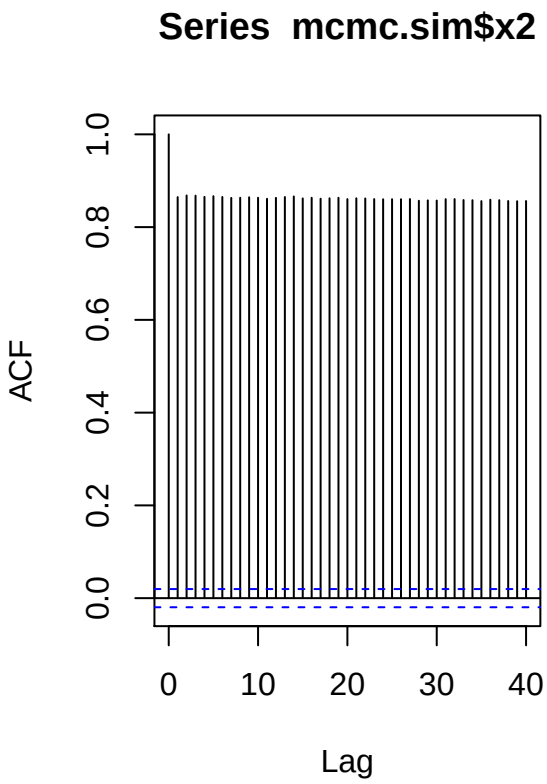
```
burn=0.3
nsim=10000

# scenario 1
eta <- 0.01
mcmc.sim <- mh.gib.sim(nsim=nsim, eta=eta, burn=burn)
# plot
par(mfrow=c(1,2))
plot(1:nsim, mcmc.sim$x1)
plot(1:nsim, mcmc.sim$x2)
```



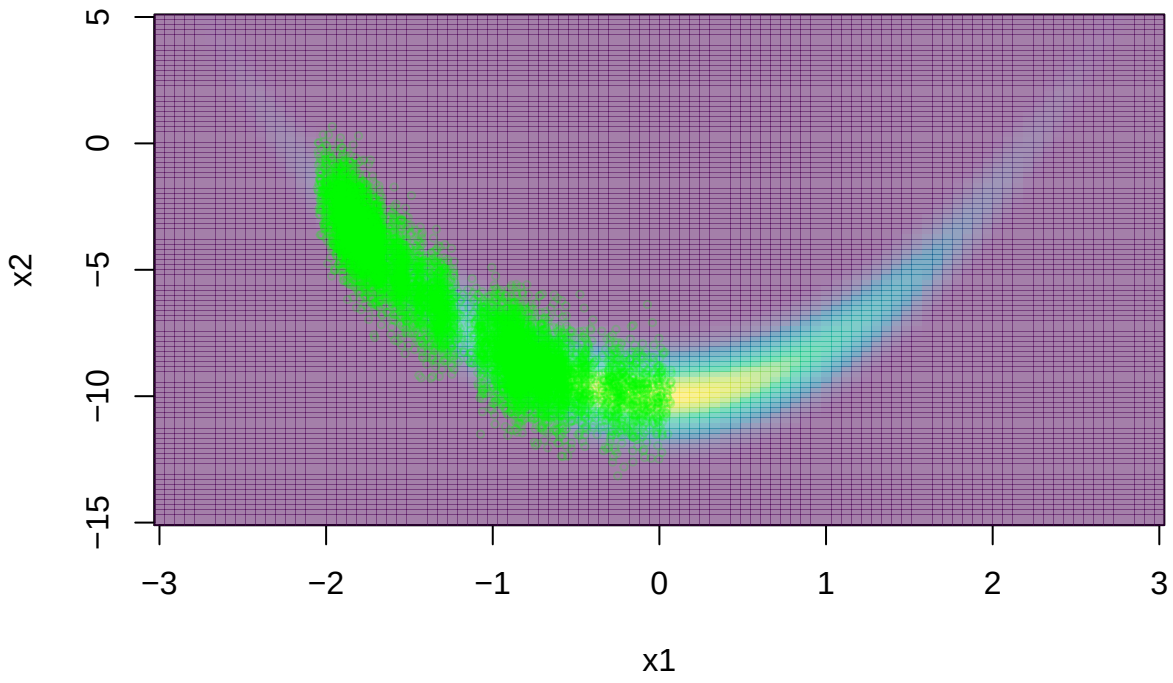




```
acf(mcmc.sim$x1)
acf(mcmc.sim$x2)
```



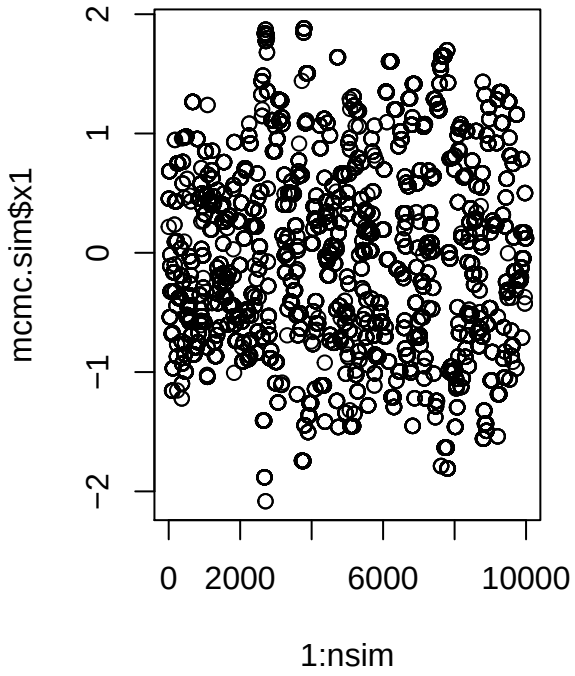



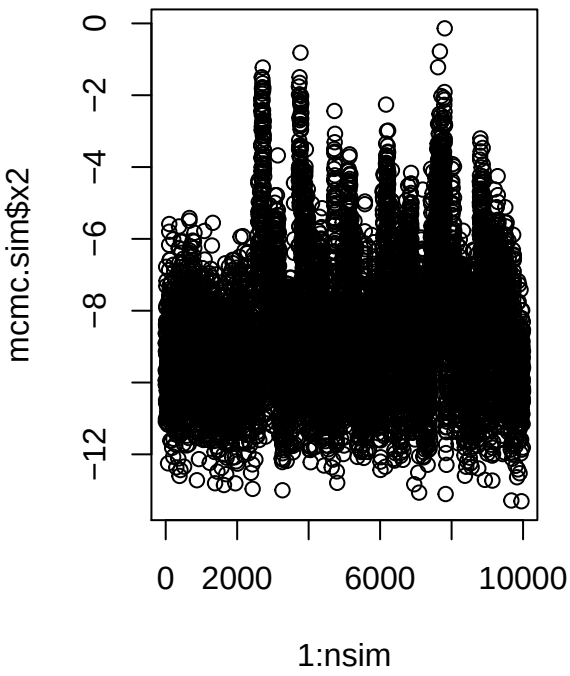

```
par(mfrow=c(1,1))
image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
      xlab=expression("x1"),
      ylab=expression("x2"))
points(mcmc.sim$x1, mcmc.sim$x2,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
```



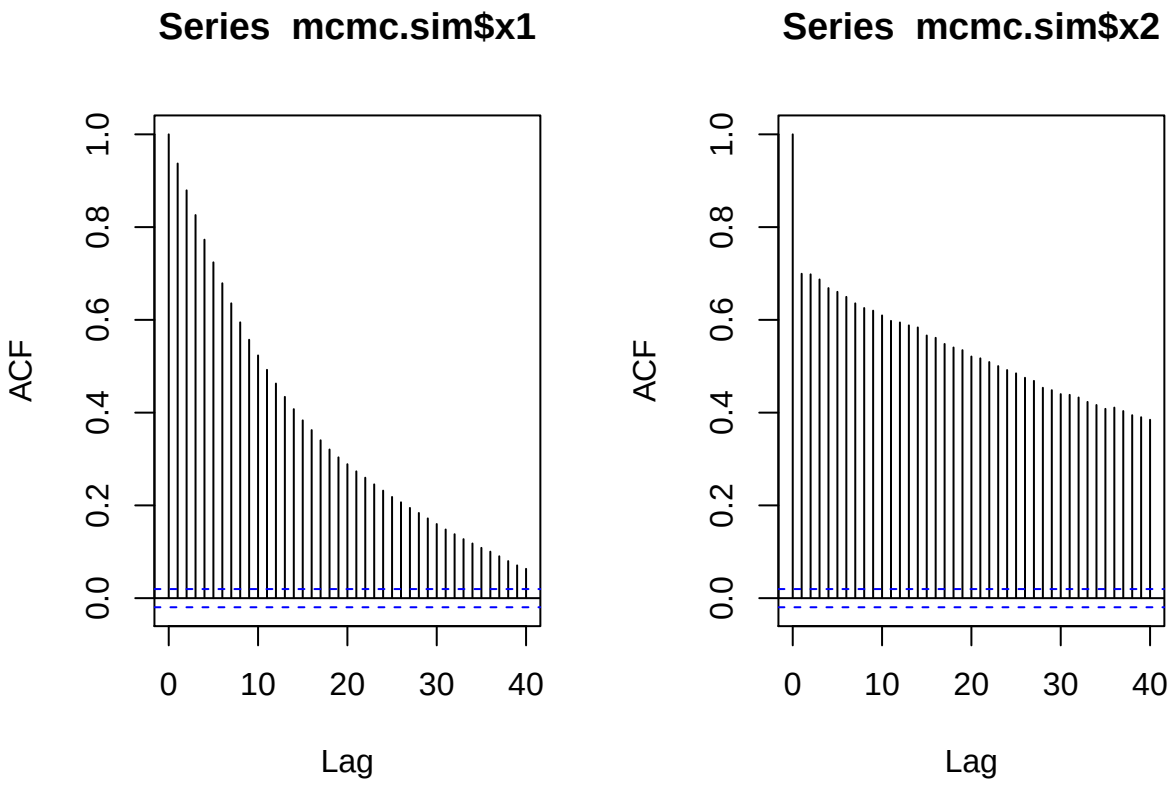

We can tell from the plot that the chain wander around a lot.

```
# scenario 2
eta <- 10
mcmc.sim <- mh.gib.sim(nsim=nsim, eta=eta, burn=burn)
# plot
par(mfrow=c(1,2))
plot(1:nsim, mcmc.sim$x1)
plot(1:nsim, mcmc.sim$x2)
```

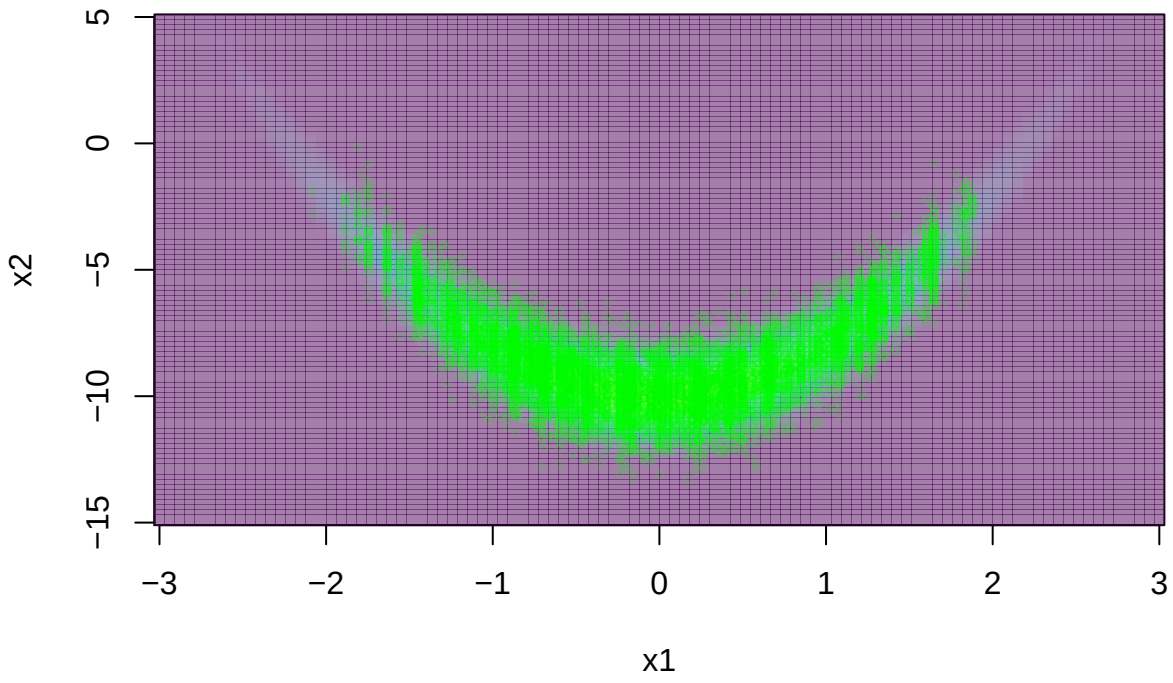



```
acf(mcmc.sim$x1)
acf(mcmc.sim$x2)
```

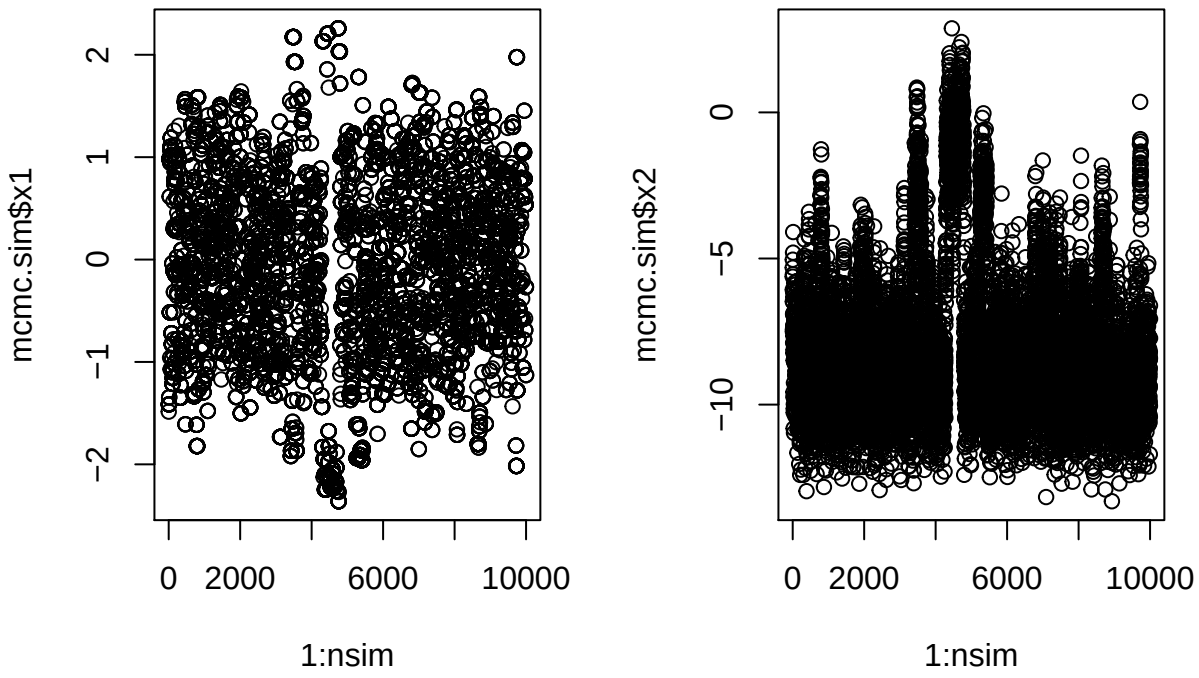



```
par(mfrow=c(1,1))
image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
      xlab=expression("x1"),
      ylab=expression("x2"))
points(mcmc.sim$x1, mcmc.sim$x2,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
```

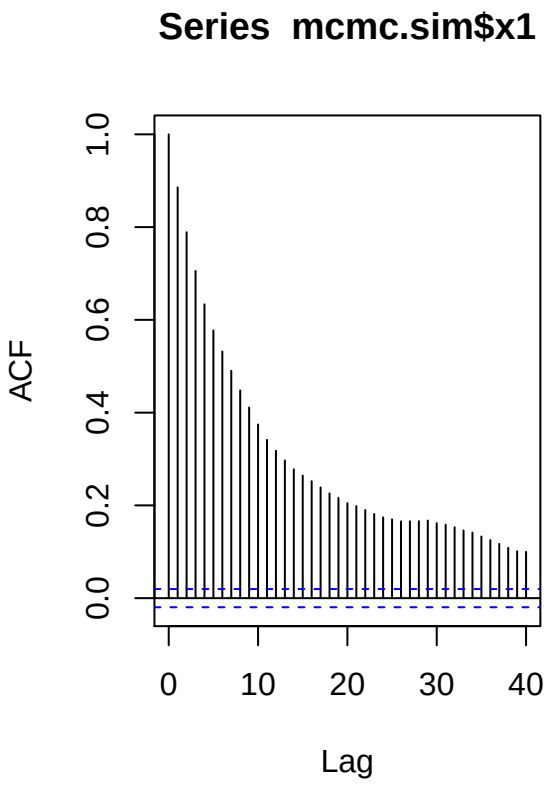


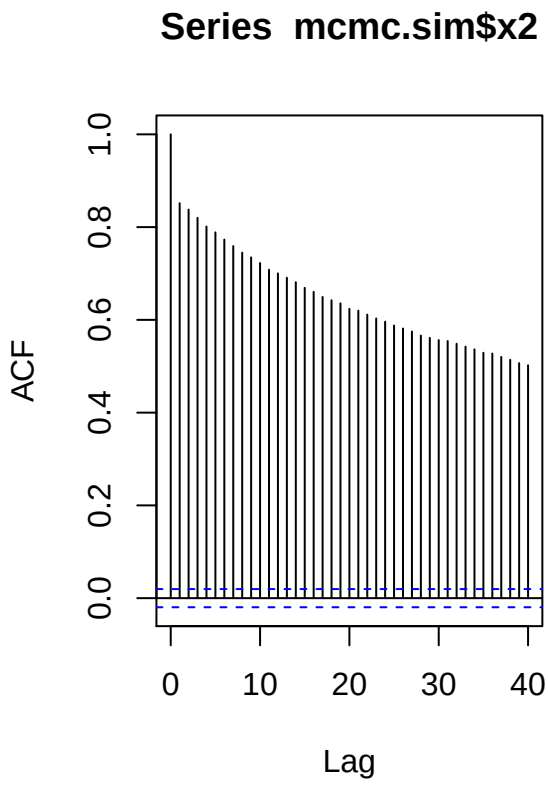

```
mcmc.sim.f<-mcmc.sim
```

```
# scenario 3
eta <- 3
mcmc.sim <- mh.gib.sim(nsim=nsim, eta=eta, burn=burn)
# plot
par(mfrow=c(1,2))
plot(1:nsim, mcmc.sim$x1)
plot(1:nsim, mcmc.sim$x2)
```

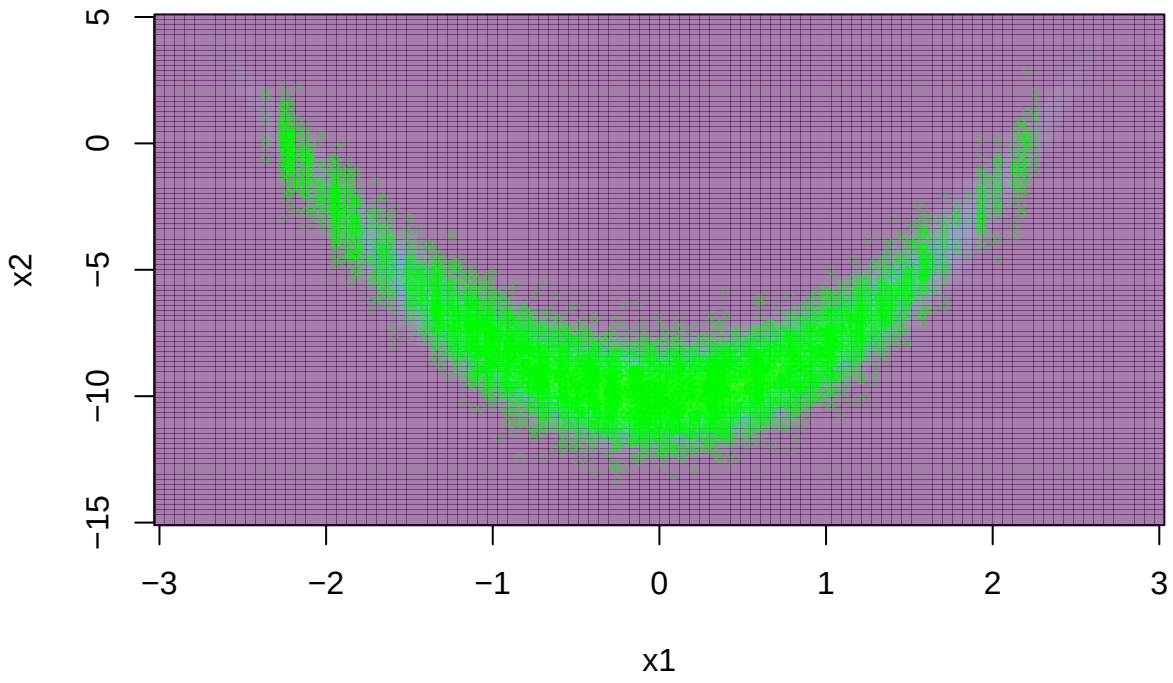



```
acf(mcmc.sim$x1)
acf(mcmc.sim$x2)
```




```
par(mfrow=c(1,1))
image(x1, x2, f, col=hcl.colors(100, alpha=0.5),
      xlab=expression("x1"),
      ylab=expression("x2"))
points(mcmc.sim$x1, mcmc.sim$x2,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
```




We can tell from the plot that the chain converge well with eta=delta=10 and eta=delta=3 as well. ACF plot also shows that autocorrelation approach to 0 fast. Compared to the method that generates X1 and X2 iteratively, this method provides similar flexibility on the eta and delta setting.

(g)

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

```
.summary_fun <- function(x1,x2,method){
  df_x1 <- c(mean(x1^2), sd(x1^2), quantile(x1^2, c(0.025, 0.975))) %>% round(2)
  df_x2 <- c(mean(x2), sd(x2), quantile(x2, c(0.025, 0.975))) %>% round(2)
  out <- rbind(df_x1, df_x2)
  out <- cbind(out, method=method)
  colnames(out)[c(1,2)] <- c("E", "sd")
  return(out)
}

df <- rbind(.summary_fun(x_q1,x_q2,method="a"),
            .summary_fun(x_AR[,1],x_AR[,2],method="b"),
            .summary_fun(x_IS[,1],x_IS[,2],method="c"),
            .summary_fun(mcmc.sim.d$x1,mcmc.sim.d$x2,method="d"),
            .summary_fun(mcmc.sim.e$x1,mcmc.sim.e$x2,method="e"),
            .summary_fun(mcmc.sim.f$x1,mcmc.sim.f$x2,method="f"))
df %>%
```

Table 1: summary table

| | E | sd | 2.5% | 97.5% | method |
|---|---|---|---|---|---|
| df_x1 | 0.99 | 1.41 | 0 | 5.03 | a |
| df_x2 | -8.02 | 2.98 | -11.45 | 0.13 | a |
| df_x1 | 1.25 | 1.58 | 0 | 5.4 | b |
| df_x2 | -7.45 | 3.3 | -11.46 | 0.55 | b |
| df_x1 | 1.2 | 1.54 | 0 | 5.45 | c |
| df_x2 | -7.54 | 3.25 | -11.31 | 0.67 | c |
| df_x1 | 0.91 | 1.13 | 0 | 3.94 | d |
| df_x2 | -8.2 | 2.43 | -11.49 | -1.45 | d |
| df_x1 | 0.68 | 0.85 | 0 | 3.19 | e |
| df_x2 | -8.69 | 1.94 | -11.52 | -3.65 | e |
| df_x1 | 0.69 | 0.77 | 0 | 2.72 | f |
| df_x2 | -8.62 | 1.84 | -11.49 | -4.16 | f |

```
  kbl(caption = "summary table") %>%
  kable_classic(full_width = T, html_font = "Cambria")
```

As we can tell from the table, all methods provides consistent estimation of X1 and X2. Generally speaking, MCMC methods provided better (smaller) sample var. Among all methods, Gibbs sampling shows the smallest sample var. Iterative Metropolis Hastings algorithm performed better than the original one. AR method gave the largest sample var.

**(h)**

For the estimation of the quantities in (vii), 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.

```
set.seed(1996)
delta <- c(0, -10)
Sigma <- matrix(c(1,0.9,0.9,20),2,2)
nsim <- 15000
Sigma1 <- Sigma
nu <- 5

x_g1 <- rmvt(n = nsim, sigma = Sigma1, df = nu, delta = delta)
x_g2 <- cbind(-x_g1[,1], x_g1[,2])
x_g <- rbind(x_g1, x_g2)
#plot(x_g, col=rgb(0,0,1, alpha=0.5), pch=19, cex=0.5)
x_g_hat <- x_g[which(x_g[,1]>=0),]
x_g_hat <- rbind(x_g_hat, x_g_hat)
x_g_hat[(dim(x_g_hat)[1]/2+1):dim(x_g_hat)[1],1] <-
  -abs(x_g_hat[(dim(x_g_hat)[1]/2+1):dim(x_g_hat)[1],1])
nsim <- dim(x_g_hat)[1]

#plot(x_g_hat, col=rgb(0,0,1, alpha=0.5), pch=19, cex=0.5)
#
```

Table 2: Antithetic Variates summary table

| | E | sd | 2.5% | 97.5% | method |
|---|---|---|---|---|---|
| df_x1 | 1.25 | 1.58 | 0 | 5.4 | origin |
| df_x2 | -7.45 | 3.3 | -11.46 | 0.55 | origin |
| df_x1 | 0.95 | 1.28 | 0 | 4.16 | Antithetic |
| df_x2 | -8.1 | 2.77 | -11.52 | -1 | Antithetic |

```
# AR Sampling
#
g <- dmvt(x_g_hat, sigma=Sigma1, df=nu, delta=delta, log=FALSE)
#
fx <- rep(NA, nsim)
for(i in 1:nsim){
  temp1 <- x_g_hat[i,1]
  temp2 <- x_g_hat[i,2]
  fx[i] <- .banana(temp1, temp2)
}
f1 <- fx
M  <- 300
u  <- runif(nsim)
#
c1 <- f1/g * 1/M
x_AR_anti <- x_g_hat[c1 >= u,]
#points(x_AR_anti,  col=rgb(0,1,0,alpha=0.2), cex=0.5)
#contour(x1, x2, f, add=T, col=rgb(1,0,0,alpha=0.4))

#compare improvement
df <- rbind(.summary_fun(x_AR[,1],x_AR[,2], method = "origin"),
            .summary_fun(x_AR_anti[,1],x_AR_anti[,2], method = "Antithetic"))
df %>%
  kbl(caption = "Antithetic Variates summary table") %>%
  kable_classic(full_width = T, html_font = "Cambria")
```

**Antithetic Variates (for AR method)** As we can tell from the results, Antithetic Variates improved the estimates (provided a smaller sample var).

**Rao-Blackwell (for Gibbs)** Firstly, from part (a) we know that:

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

Then we can generate a Rao-Blackwellized estimator for X_2. We can get that:

$E(x_2|x_1) = 2(x_1^2 - 5)$

```
mcmc.sim.f$x2.rb <- 2*(mcmc.sim.f$x1^2-5)

#compare improvement
df <- rbind(.summary_fun(mcmc.sim.f$x1,mcmc.sim.f$x2, method = "origin"),
            .summary_fun(mcmc.sim.f$x1,mcmc.sim.f$x2.rb, method = "Rao-Blackwell"))
df %>%
  kbl(caption = "Rao-Blackwell summary table") %>%
  kable_classic(full_width = T, html_font = "Cambria")
```

Table 3: Rao-Blackwell summary table

| | E | sd | 2.5% | 97.5% | method |
|---|---|---|---|---|---|
| df_x1 | 0.69 | 0.77 | 0 | 2.72 | origin |
| df_x2 | -8.62 | 1.84 | -11.49 | -4.16 | origin |
| df_x1 | 0.69 | 0.77 | 0 | 2.72 | Rao-Blackwell |
| df_x2 | -8.62 | 1.53 | -10 | -4.56 | Rao-Blackwell |

As we can tell from the results, Rao-Blackwell method improved the estimates (provided a smaller sample var).

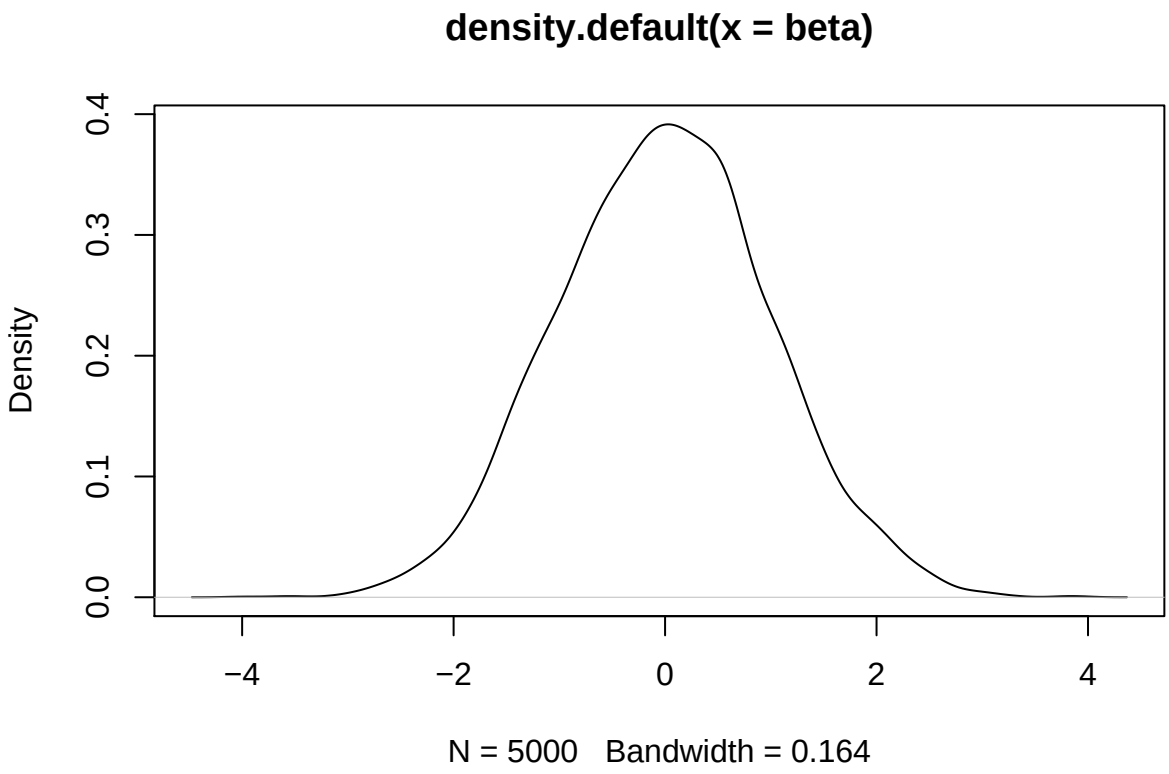
## 2. Bayesian Adaptive Lasso

### (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`.

We know that $\beta|\tau^2 = 1 \sim N_1(0, 1)$. Apply AR method to generate $\beta$.

```
set.seed(1996)
nsim <- 1000000
gx <- 1/100
M <- 100
x <- 100*runif(nsim)-50
fx <- dnorm(x)
u <- runif(nsim)
accept_beta <- x[u <= fx/(M*gx)]
beta <- accept_beta[1:5000]
density(beta) %>% plot()
```

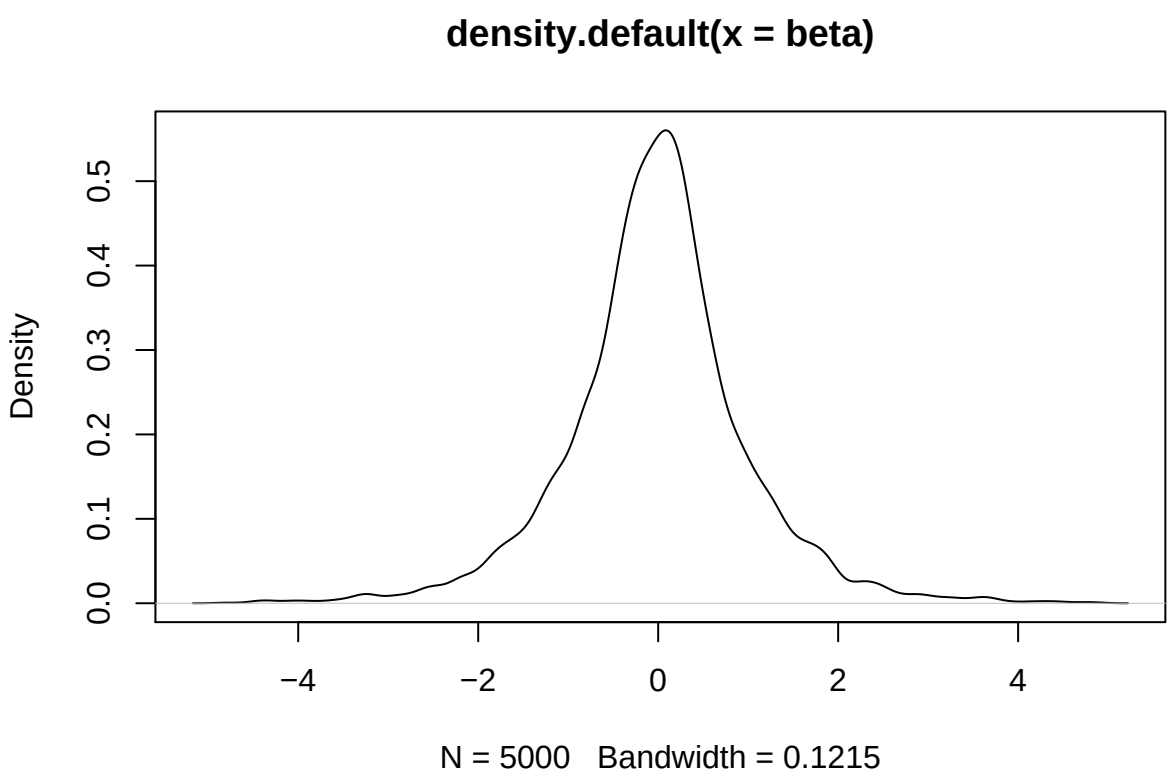



**(b)**

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

Since we know that $E(\tau^2|\lambda) = 1$, we know that $P(\tau^2|\lambda) = exp(-\tau^2)$. Then we use inverse transform method to simulate $\tau^2$ and then AR to simulate $\beta$.

```
set.seed(1996)
nsim <- 1000000
tau <- sqrt(-log(runif(nsim)))
gx <- 1/100
M <- 100
x <- 100*runif(nsim)-50
fx <- dnorm(x,0,tau)
u <- runif(nsim)
accept_beta <- x[u <= fx/(M*gx)]
beta <- accept_beta[1:5000]
density(beta) %>% plot()
```

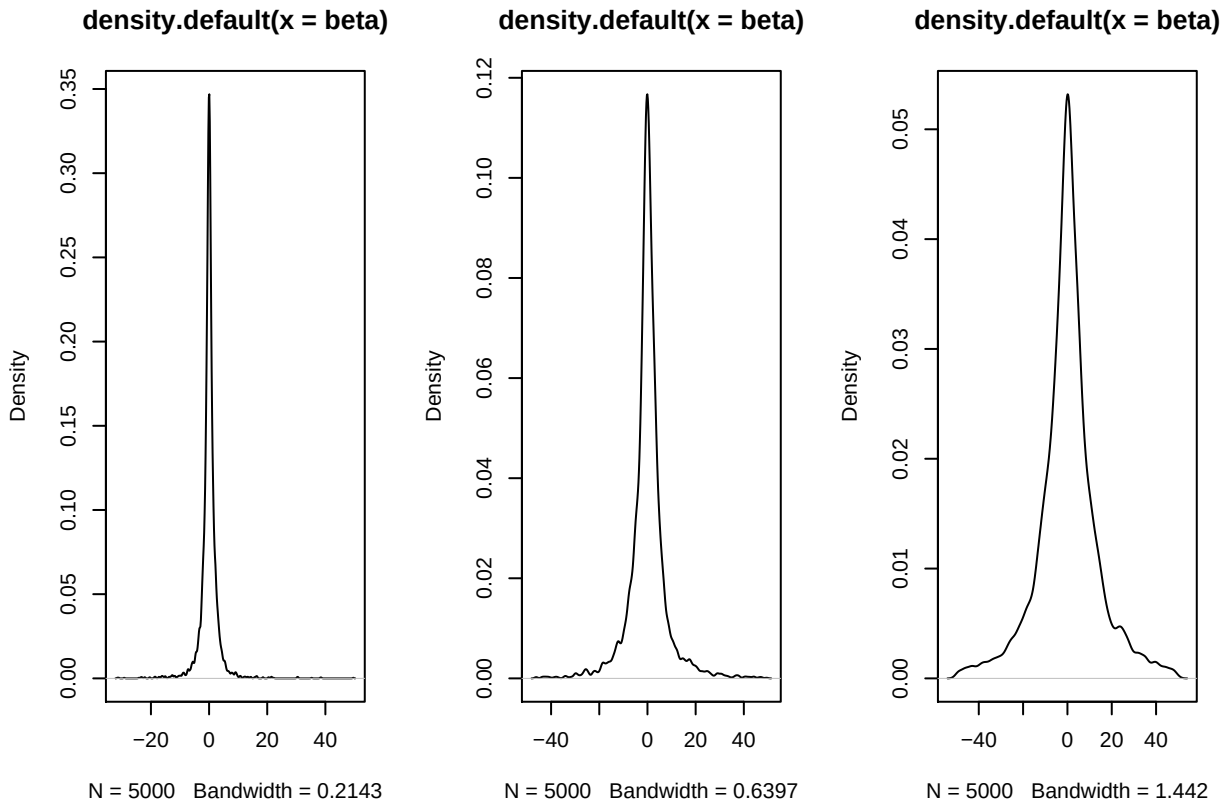



**(c)**

Consider p = 1. Add a hyper prior on $\lambda^2 \sim Gamma(a, rate = b)$. Assess how the marginal prior of $\beta$ changes for $a = 1$ and values of $b \geq 1$. (note that here we use $\lambda^2 \sim Gamma(a, rate = b)$ rather than $1/\lambda^2 \sim Gamma(a, rate = b)$ because of the convenience of calculation in the following part).

We know that Exponential(b) = Gamma(1,b). Then we use inverse transform to generate gamma(1,b).

```
set.seed(1996)
par(mfrow=c(1,3))

for(b in c(1,10,50)){
  lambda_sq <- -1/b*log(runif(nsim))
  tau <- sqrt(-2/lambda_sq*log(runif(nsim)))
  gx <- 1/100
  M <- 100
  x <- 100*runif(nsim)-50
  fx <- dnorm(x,0,tau)
  u <- runif(nsim)
  accept_beta <- x[u <= fx/(M*gx)]
  beta <- accept_beta[1:5000]
  density(beta) %>% plot()
}
```




We can tell from the plot that when $b$ increases, $\beta$ gets more dispersed.

**(d)**

Considering the hyper prior in c., describe a Markov Chain Monte Carlo algorithm to sample from the posterior distribution of $\beta$ and $\sigma^2$.

As for $\beta$, we know that

$$\begin{aligned} lclP(\beta|Y,\sigma^2,\tau^2,\lambda) = & P(\beta|Y,\sigma^2,\tau^2) \\ \propto & P(Y|\beta,\sigma^2) * P(\beta|\tau^2) \\ \propto & \exp(-\frac{1}{2\sigma^2} * (Y - X\beta)^{'}(Y - X\beta)) * \exp(-\frac{1}{2\tau^2} * \beta^{'}\beta) \\ = & \exp(-\frac{1}{2\sigma^2}(\beta^{'} * X^{'}X\beta - 2 * \beta X^{'}Y) - \frac{1}{2\tau^2} * \beta^{'}\beta) \\ = & \exp(-\frac{1}{2}(\beta^{'}(\frac{X^{'}X}{\sigma^2} + \frac{1}{\tau^2}I)\beta - \frac{2}{\sigma^2} * \beta X^{'}Y)) \end{aligned}$$

This is a normal kernel. Then we found that $\beta|Y,\sigma^2,\tau^2,\lambda \sim N(\frac{X^{'}Y}{X^{'}X+\frac{\sigma^2}{\tau^2}}, \frac{1}{\frac{X^{'}X}{\sigma^2}+\frac{1}{\tau^2}})$.

Then, for $\tau^2$, we know that

$$
\begin{aligned}
lclP(\tau_j^2|Y,\sigma^2,\beta_j,\lambda^2) &= P(\tau^2|\lambda^2,\beta_j) \\
&= P(\beta_j|\tau_j^2,\lambda^2)P(\tau_j^2|\lambda^2) \\
&\propto 1/\sqrt{(\tau_j^2)} * \exp(-\frac{(\beta_j)^2}{2*\tau_j^2})\exp(-\frac{\lambda^2}{2}\tau_j^2) \\
&= (\tau_j^2)^{-1/2} * \exp(-\frac{\beta_j^2}{2*\tau_j^2} - \frac{\lambda^2}{2}\tau_j^2)
\end{aligned}
$$

This is inverse gaussian kernel. Then we know that $\tau_j^2|Y,\sigma^2,\beta_j,\lambda^2 \sim InvGaussian(mean = \sqrt{\frac{\beta_j^2}{\tau_j^2}}, shape = \lambda^2)$

Then we can find the density for $\frac{1}{\tau_j^2}$,

$$
\begin{aligned}
lclP(\frac{1}{\tau_j^2}|Y,\sigma^2,\beta_j,\lambda^2) &\propto (\frac{1}{\tau_j^2})^{1/2} * \exp(-\frac{\beta_j^2}{2} * \frac{1}{\tau_j^2} - \frac{\lambda^2}{2\frac{1}{\tau_j^2}}) * |-(\frac{1}{\tau_j^2})^{-2}| \\
&= (\frac{1}{\tau_j^2})^{-3/2}\exp(-\frac{\lambda^2}{2}(\frac{\beta_j^2}{\lambda^2} * \frac{1}{\tau_j^2} + (\frac{1}{\tau_j^2})^{-1}))
\end{aligned}
$$

This is the kernel of inverse gaussian distribution. Then we found that $\frac{1}{\tau_j^2} \sim InvGaussian(mean = \sqrt{\frac{\lambda^2}{\beta_j^2}}, shape = \lambda^2)$

As for $\lambda^2$,

$$
\begin{aligned}
lclP(\lambda^2|\tau^2,Y,\sigma^2,\beta_j) &\propto (\lambda^2)^{a-1}\exp(-b\lambda^2)(\frac{2}{\lambda^2})^p\exp(-\frac{\lambda^2}{2}\sum\tau_j^2) \\
&\propto (\lambda^2)^{a+p-1}\exp(-\lambda^2(b+\frac{1}{2}\sum\tau_j^2))
\end{aligned}
$$

Then we know $\lambda^2|\tau^2,Y,\sigma^2,\beta_j \sim Gamma(a = 1+10, b+\frac{1}{2}\sum\tau_j^2)$.

As for $\sigma^2$, we know that

$$
\begin{aligned}
lclP(\sigma^2|Y,\tau^2,\beta,\lambda^2) &= P(\sigma^2|Y,\beta) \\
&\propto P(Y|\beta,\sigma^2) * P(\sigma^2) \\
&\propto (2\pi*\sigma^2)^{-\frac{n}{2}}\exp(-\frac{1}{2\sigma^2} * (Y-X\beta)'(Y-X\beta)) * (\sigma^2)^{-a-1}\exp(-\frac{b}{\sigma^2}) \\
&= (\sigma^2)^{-a-\frac{n}{2}-1} * \exp(-\frac{1/2*(Y-X\beta)'(Y-X\beta)+b}{\sigma^2})
\end{aligned}
$$

This is an Inverse Gamma kernel. Then we found that $\sigma^2|Y,\beta \sim InvGamma(0.1+\frac{n}{2}, 1/2*(Y-X\beta)'(Y-X\beta)+10)$

After obtaining conditional posterior distributions for all parameters, we can apply Markov Chain Monte Carlo (Gibbs) algorithm to sample $\beta$ and $\sigma^2$.

**(e)**

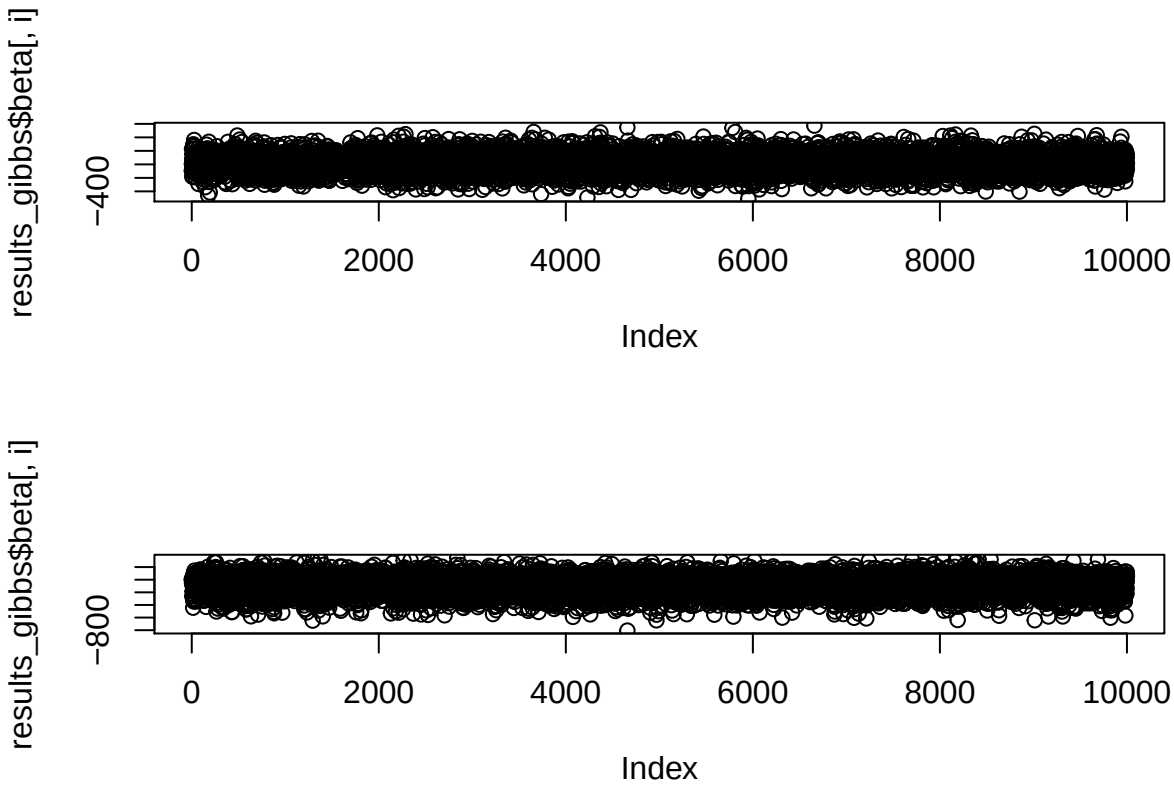
Implement such algorithm in R and compare your results with estimates obtained using `glmnet()`. In particular, you should test your results on the diabetes data available from `lars`, (use the matrix of predictors x).

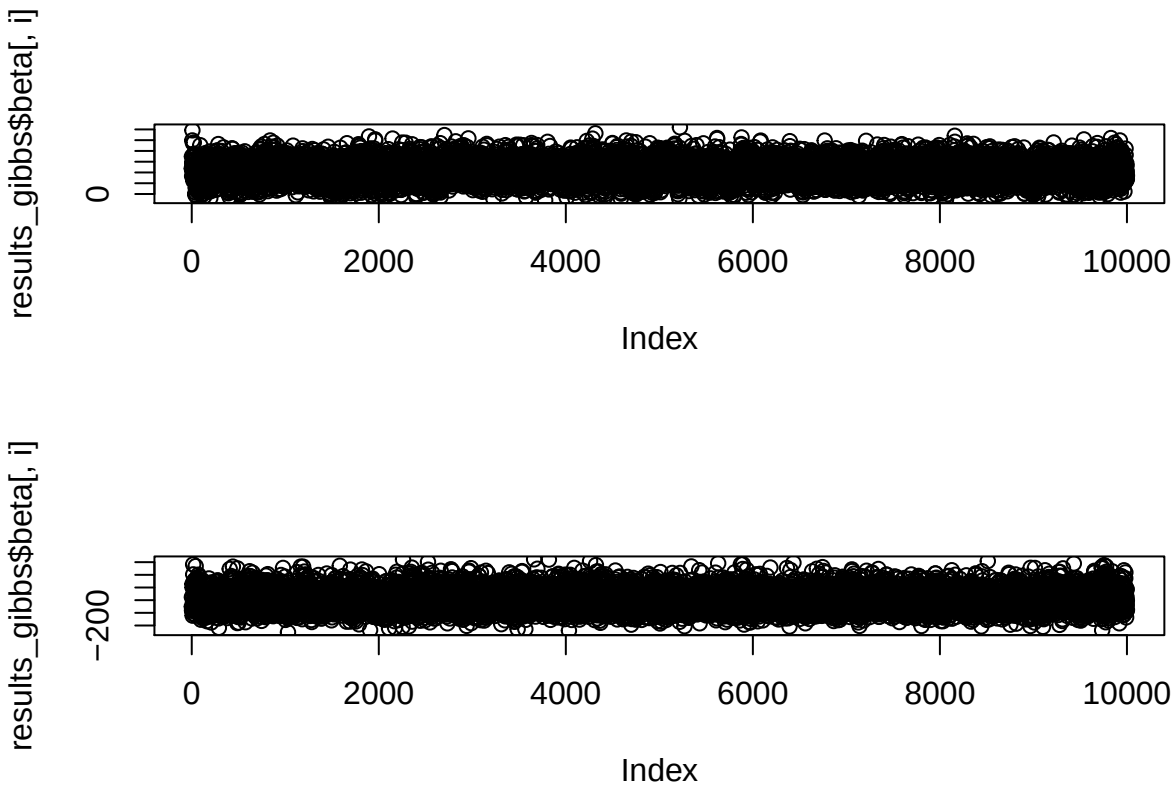
```
data("diabetes")
dia_x <- diabetes$x
dia_y <- diabetes$y
initial_value <- list(tau_sq=rep(10,10),
                      beta=rep(0,10),
                      sigma_sq=0.01,
                      lambda_sq=1)
mcmc.sim.gibbs <- function(initial_value, reg_path=F,
                           nsim=1000, burn=0, a=1, b=2,
                           X=dia_x, Y=dia_y, seed=199609){
  set.seed(seed)
  nsim.total <- nsim*(1.0 + burn)
  burn.num <- nsim*burn

  tau_sq <- initial_value$tau_sq
  beta <- initial_value$beta
  sigma_sq <- initial_value$sigma_sq
  lambda_sq <- initial_value$lambda_sq

  beta.ch <- matrix(NA,nsim,10)
  bmean.ch <- matrix(NA,nsim,10)
  tau_sq.ch <- matrix(NA,nsim,10)
  sigma_sq.ch <- vector()
  lambda_sq.ch <- vector()

  for(i in 1:nsim.total){
    #i=2
    #beta
    bmean <- solve(t(X)%*%X+diag(sigma_sq/tau_sq))%*%t(X)%*%Y
    bvar <- solve(t(X)%*%X/sigma_sq+diag(1/tau_sq))
    beta <- rmvnorm(n=1, mean = bmean, sigma = bvar)

    #lambda_sq
    if(reg_path==T){
      lambda_sq <- lambda_sq
    }else{
      lambda_sq <- rgamma(1, a+10, rate = b+sum(tau_sq)/2)
    }

    #sigma_sq
    sigma_sq <- 1/rgamma(1,shape = 0.1+221,
                         rate = 1/2*t(Y-X%*%t(beta))%*%(Y-X%*%t(beta))+10)

    #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(i > burn.num){
      i1 <- i - burn.num
      bmean.ch[i1,] <- bmean
      beta.ch[i1,] <- beta
```

```
      sigma_sq.ch[i1] <- sigma_sq
      tau_sq.ch[i1,] <- tau_sq
      lambda_sq.ch[i1] <- lambda_sq
    }
  }
  return(list(beta=beta.ch, sigma_sq=sigma_sq.ch, tau_sq=tau_sq.ch))
}

results_gibbs <- mcmc.sim.gibbs(initial_value, nsim=10000, burn=0.5, b=2)
```

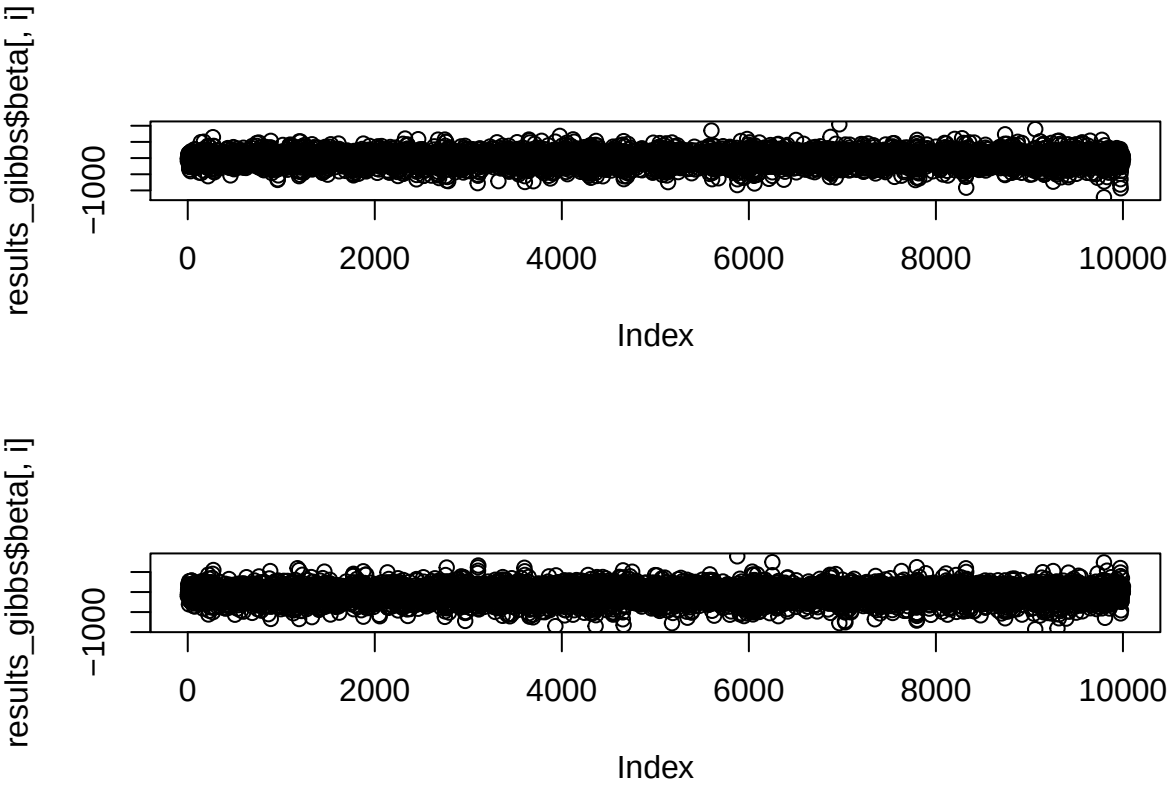
check results

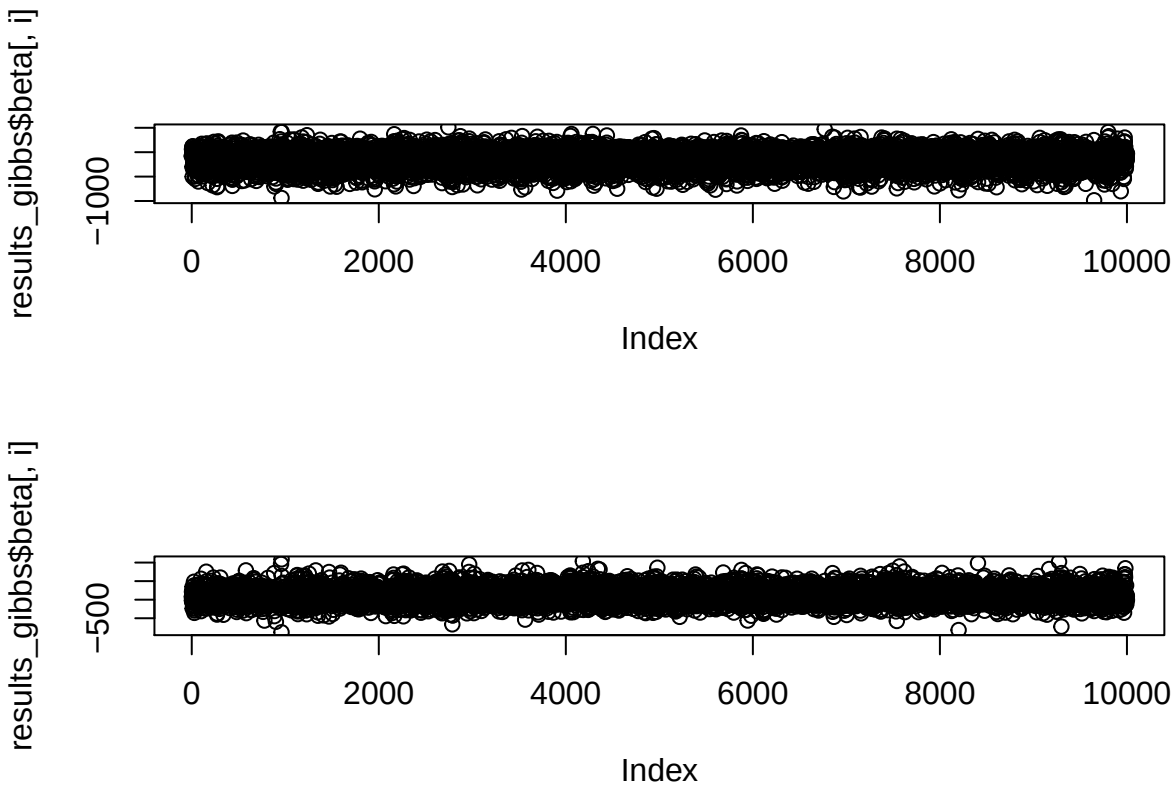
```
par(mfrow=c(2,1))
for(i in 1:10){
  plot(results_gibbs$beta[,i])
}
```



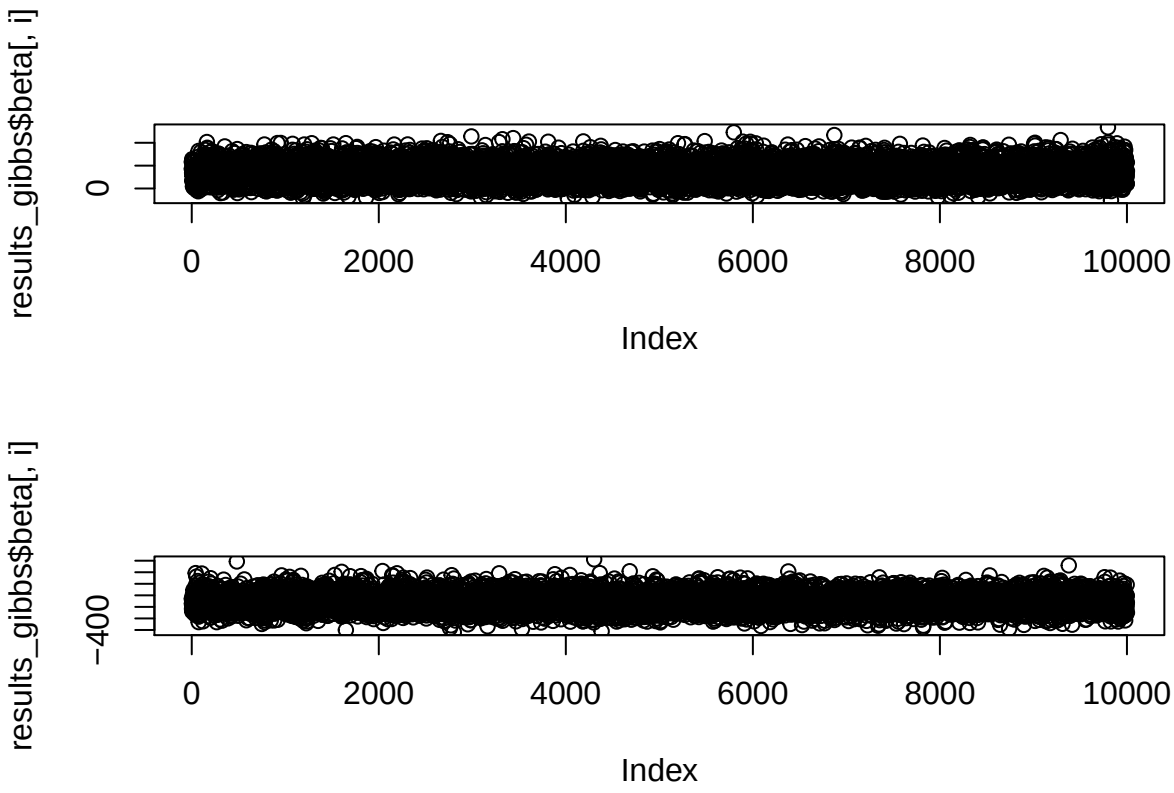

results_gibbs$beta[, i]
0
0
2000
4000
6000
8000
10000
Index


results_gibbs$beta[, i]
−200
0
2000
4000
6000
8000
10000
Index

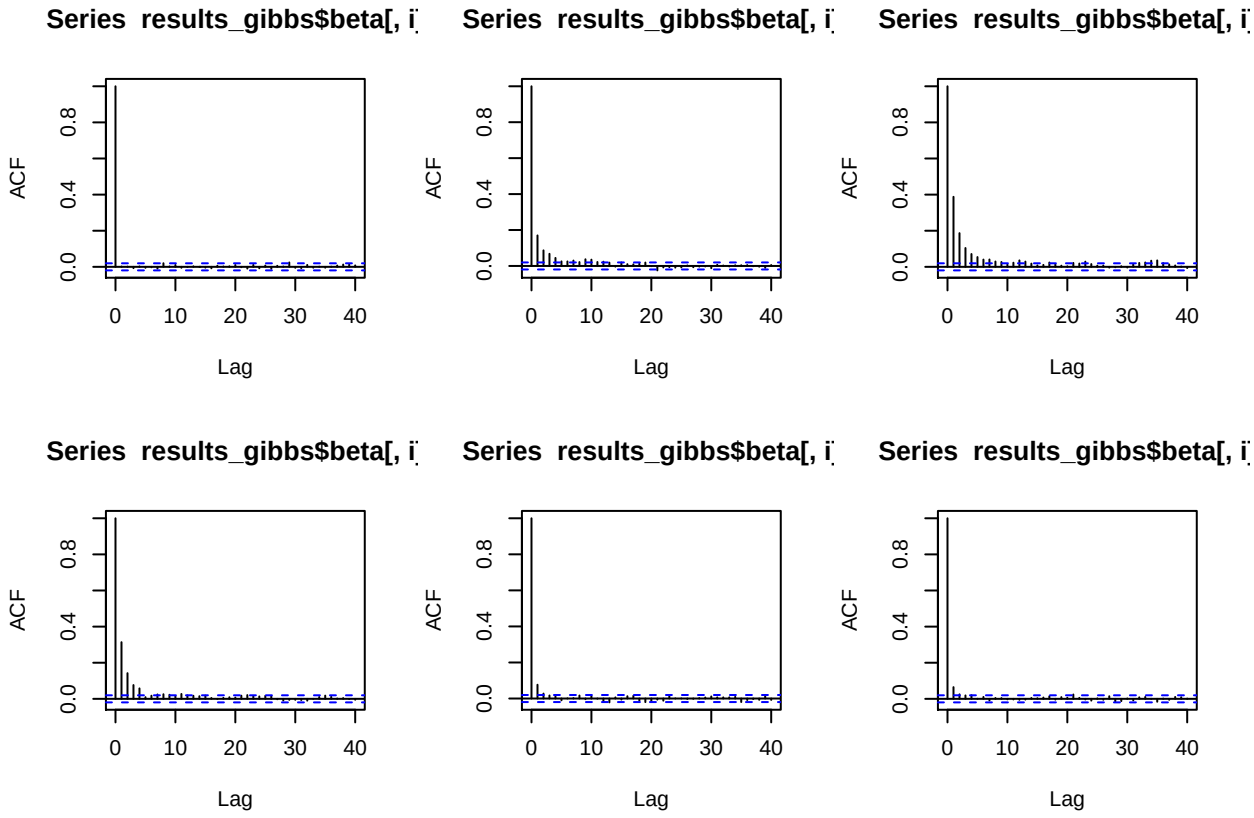

results_gibbs$beta[, i]
−1000
0
2000
4000
6000
8000
10000
Index
results_gibbs$beta[, i]
−1000
0
2000
4000
6000
8000
10000
Index

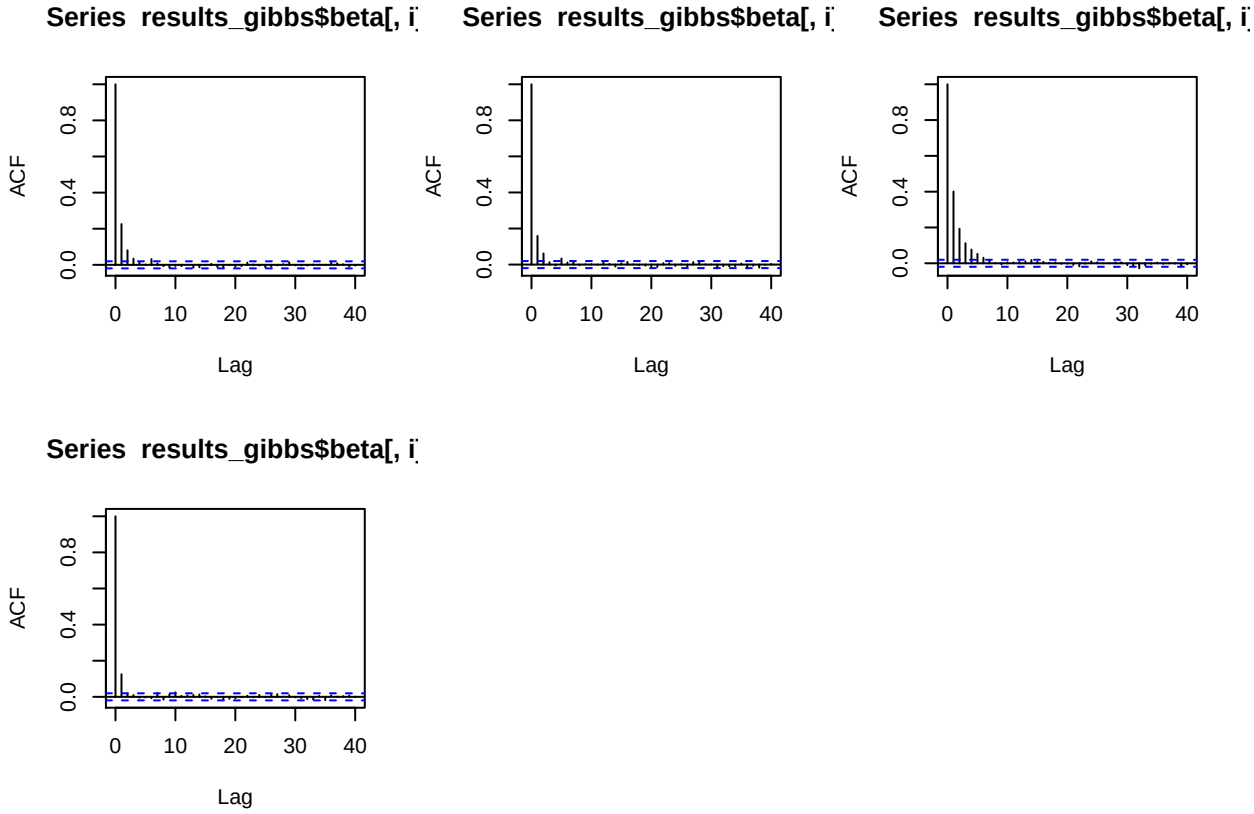

results_gibbs$beta[, i]
−1000
0
2000
4000
6000
8000
10000
Index


results_gibbs$beta[, i]
−500
0
2000
4000
6000
8000
10000
Index




```
par(mfrow=c(2,3))
for(i in 1:10){
  acf(results_gibbs$beta[,i])
}
```


Series results_gibbs$beta[, i
Series results_gibbs$beta[, i
Series results_gibbs$beta[, i
Series results_gibbs$beta[, i
Series results_gibbs$beta[, i
Series results_gibbs$beta[, i
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag




We can tell from the plots that the algorithm converged well.

Apply `glmnet` function to obtain estimators (using elastic net).

```
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
```

```
fit.lasso <- glmnet(dia_x, dia_y)
cvfit <- cv.glmnet(dia_x, dia_y)
coef(cvfit, s = "lambda.min")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  152.13348
```

Table 4: summary table

| | age | sex | bmi | map | tc | ldl | hdl | tch | ltg | glu |
|---|---|---|---|---|---|---|---|---|---|---|
| 2.5% | -223.71 | -374.58 | 62.50 | -59.16 | -370.72 | -388.68 | -506.02 | -215.13 | -5.60 | -161.59 |
| 50% | 13.71 | -66.46 | 422.49 | 191.12 | -8.48 | -21.48 | -114.23 | 72.18 | 341.91 | 64.39 |
| 97.5% | 260.02 | 143.61 | 803.92 | 548.23 | 298.52 | 247.64 | 143.03 | 491.14 | 770.57 | 370.16 |

```
## age            .
## sex         -220.78374
## bmi          525.78823
## map          310.97637
## tc          -175.07730
## ldl            .
## hdl         -169.42518
## tch           82.55659
## ltg          526.57119
## glu           62.66858
```

```
df <- apply(results_gibbs$beta, 2,
            function(x){round(quantile(x,c(0.025,0.5,0.975)),2)})
colnames(df) <- colnames(dia_x)
df %>%
  kbl(caption = "summary table") %>%
  kable_classic(full_width = T, html_font = "Cambria")
```

```
results_gibbs$sigma_sq %>% quantile(.,c(0.025,0.5,0.975))
```

```
##     2.5%      50%    97.5%
## 23377.36 26612.42 30354.64
```

As we can tell from the results, beta estimated by Bayesian Adaptive Lasso method and glmnet method are largely consistent.

### (f)

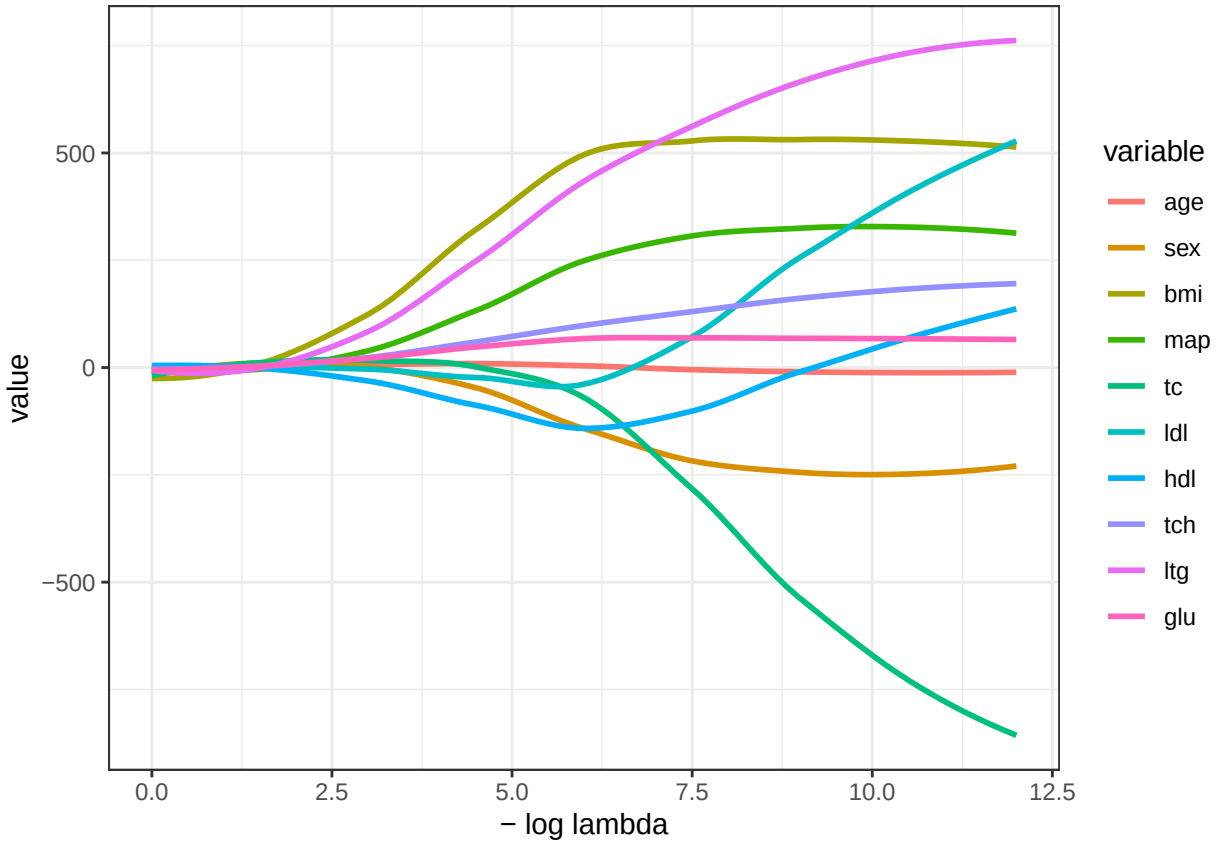
For the diabetes data, fix $\lambda$ and produce a regularization path for adaptive Bayesian Lasso obtained on a grid of values for the tuning parameter $\lambda$ . Describe your approach and compare your result with the path obtained using `glmnet`().

```
lambda_sq_grid <- data.frame(lambda_sq=exp(seq(-12,0,0.3))^2)

'start_time <- Sys.time()
test <- apply(lambda_sq_grid, 1,
              function(lambda_sq){
                initial_value$lambda_sq <- lambda_sq
                return(mcmc.sim.gibbs(initial_value,nsim=5000,
                                      burn=0.5, b=2, reg_path=T))})
saveRDS(test, "regularization_path.rds")
end_time <- Sys.time()
end_time - start_time'
```

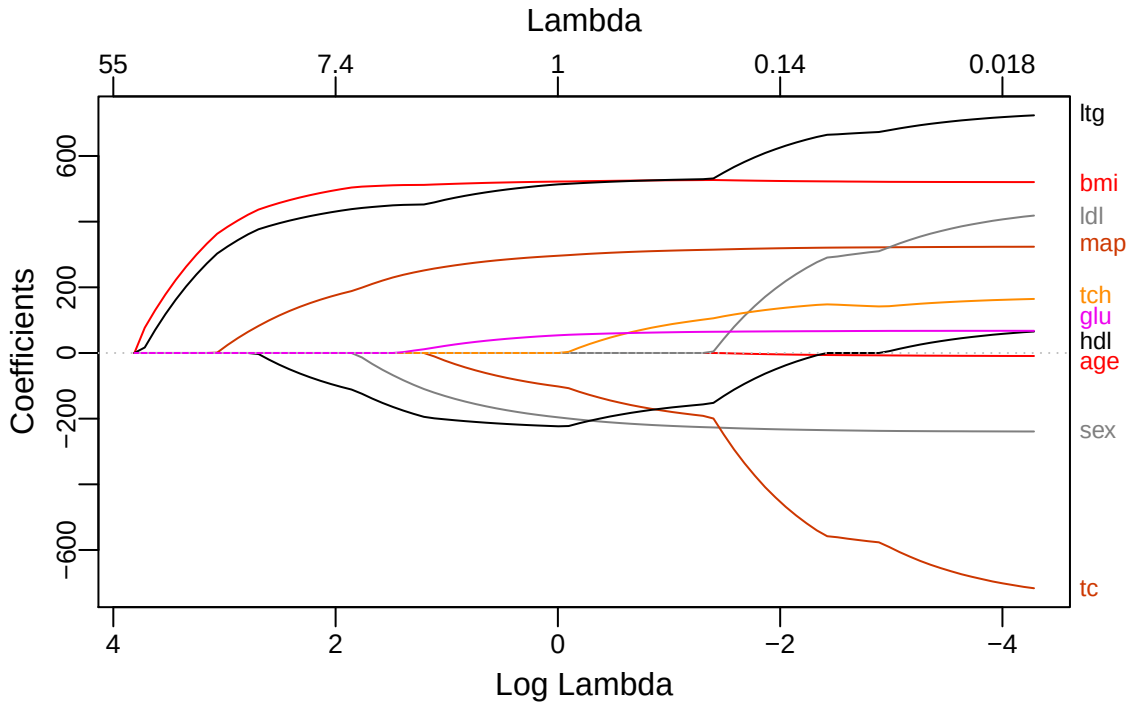
```
## [1] "start_time <- Sys.time()\ntest <- apply(lambda_sq_grid, 1, \n                function(lambda_sq){"
```

```
test <- readRDS("regularization_path.rds")
results <- lapply(test, function(x){apply(x$beta,2,median)})
results_plot <- data.frame(Reduce("rbind", results))
colnames(results_plot) <- colnames(dia_x)
results_plot$lambda <- log(sqrt(lambda_sq_grid$lambda_sq))
results_plot <- melt(results_plot, id.vars = names(results_plot)[11])
ggplot(results_plot, aes(-lambda, value, color=variable)) +
  #geom_point()+
  geom_smooth(method = loess, se = FALSE) +
  labs(x = "- log lambda")+
  theme_bw()
```

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




```
plot_glmnet(fit.lasso)
```




As we can tell from the regularization path, beta estimated by Bayesian Adaptive Lasso method and `glmnet` method are largely consistent. The corresponding $\lambda$ for regularization path is different with that from `glmnet` method. The possible reason is that we use a slightly different way to parameterize the prior of $\beta$ ($\beta|\tau \sim N(0, D_\tau)$ rather than $\beta|\tau \sim N(0, \sigma^2 D_\tau)$).

### (g)

Free $\lambda$ and carry out a sensitivity analysis assessing the behavior of the posterior distribution of $\beta$ and $\sigma^2$, 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.

To conduct sensitivity analysis, we prepare several value pairs of a and b.

```
(df_sen <- data.frame(a=c(1,1,1,2,5,2),
                      b=c(1,5,10,1,1,2)))
```

```
##   a  b
## 1 1  1
## 2 1  5
## 3 1 10
## 4 2  1
## 5 5  1
## 6 2  2
```

```
'start_time <- Sys.time()
test <- apply(df_sen, 1,
              function(sen){
                return(mcmc.sim.gibbs(initial_value, nsim=5000, burn=0.3,
                                      a=sen[1], b=sen[2]))})
saveRDS(test, "sensitivity_analysis.rds")
end_time <- Sys.time()'
```

```
## [1] "start_time <- Sys.time()\ntest <- apply(df_sen, 1, \n              function(sen){\n
```

```
test <- readRDS("sensitivity_analysis.rds")
results_beta <- lapply(test,
                       function(x){apply(x$beta,2,
                                         function(beta){
                                           quantile(beta,c(0.025, 0.5, 0.975))}
                                         )})
results_sigma_sq <- lapply(test, function(x){apply(x$sigma_sq %>% as.matrix,2,
                                          function(sigma_sq){
                                            quantile(sigma_sq,c(0.025,0.5,0.975))}
                                          )})
results_ci_beta <- data.frame(Reduce("rbind", results_beta))
results_ci_sigma_sq <- data.frame(Reduce("rbind", results_sigma_sq))
results_ci <- cbind(results_ci_beta, results_ci_sigma_sq)
colnames(results_ci) <- c(colnames(dia_x), "sigma_sq")

df_compare <- matrix(NA, 11, 6)
for(i in 0:5){
  for(j in 1:11){
    df_compare[j,i+1] <- paste0(round(results_ci[3*i+2,j],2),"(",
                                round(results_ci[3*i+1,j],2), ", ",
                                round(results_ci[3*i+3,j],2), ")")
  }
}
rownames(df_compare) <- c(colnames(dia_x), "sigma_sq")
colnames(df_compare) <- paste0("a = ", df_sen$a, ", b = ", df_sen$b)
df_compare %>%
  kbl(caption = "Sensitivity Analysis") %>%
  kable_classic(full_width = T, html_font = "Cambria")
```

As we can tell from the table, changing of parameter $a$ will heavily affect the estimate of $\beta$, while the value of $b$ have little effects. Thus we need to fix $a = 1$. As for $b$, I will recommend to use $b = 10$ because it gives a more vague (dispersed) prior of $\lambda$ and allow observations to provide more information.

Table 5: Sensitivity Analysis

| | a = 1, b = 1 | a = 1, b = 5 | a = 1, b = 10 | a = 2, b = 1 | a = 5, b = 1 | a = 2, b = 2 |
|---|---|---|---|---|---|---|
| age | 10.8(-220.1, 261.71) | 10.8(-220.1, 261.71) | 10.79(-220.1, 261.72) | 0(-2.6, 2.8) | 0(-1.42, 1.53) | 0(-3.66, 3.98) |
| sex | -65.82(-393.25, 148.3) | -65.83(-393.25, 148.3) | -65.83(-393.26, 148.31) | 0.01(-2.88, 2.82) | 0.01(-1.49, 1.54) | 0.01(-4.15, 4.01) |
| bmi | 433.2(61.17, 801.18) | 433.2(61.17, 801.19) | 433.21(61.17, 801.21) | 0.04(-2.61, 2.83) | 0.01(-1.49, 1.51) | 0.07(-3.64, 4.11) |
| map | 194.04(-57.28, 552.7) | 194.04(-57.28, 552.71) | 194.04(-57.28, 552.72) | 0.02(-2.62, 2.85) | 0.02(-1.52, 1.57) | 0.04(-3.66, 4.12) |
| tc | -13.15(-368.98, 301.52) | -13.16(-369, 301.53) | -13.16(-369.01, 301.53) | -0.01(-2.7, 2.78) | -0.01(-1.39, 1.45) | 0(-3.77, 3.96) |
| ldl | -22.35(-374.23, 258.55) | -22.36(-374.23, 258.56) | -22.36(-374.24, 258.57) | 0.01(-2.61, 2.92) | 0(-1.44, 1.51) | 0.02(-3.66, 4.11) |
| hdl | -112.38(-519.57, 137.93) | -112.39(-519.6, 137.94) | -112.39(-519.63, 137.94) | 0(-2.82, 2.53) | -0.01(-1.51, 1.45) | -0.01(-4.1, 3.51) |
| tch | 68.6(-222.88, 479.84) | 68.6(-222.89, 479.85) | 68.6(-222.9, 479.85) | 0(-2.67, 2.83) | 0(-1.55, 1.51) | 0(-3.78, 4.09) |
| ltg | 344.54(-2.97, 783.51) | 344.56(-2.97, 783.52) | 344.57(-2.96, 783.53) | 0(-2.67, 2.84) | 0.01(-1.52, 1.62) | 0.01(-3.65, 4.17) |
| glu | 67.52(-168.86, 373.71) | 67.52(-168.86, 373.71) | 67.52(-168.87, 373.71) | 0.03(-2.7, 3.01) | 0.01(-1.47, 1.46) | 0.05(-3.76, 4.32) |
| sigma_sq | 26550.58(23277.55, 30448.05) | 26550.55(23277.55, 30448.06) | 26550.5(23277.55, 30448.08) | 29103.98(25576.88, 33314.72) | 29087.21(25569.29, 33224.14) | 29104.9(25577.55, 33319.42) |