Biostat 276 Homework 1

Zian ZHUANG

Bayesian Probit Regression

In R load the package (survival) and consider the analysis of the data-set (infert). Ignoring dependence due to matching, consider a Bayesian analysis for a logistic regression model relating case status to: age, parity, education, spontaneous and induced. More precisely, assume case status y_i has density $y_i \sim_{ind} Bern(p_i), p_i = \Phi(X_i^{'}\beta)$, where $\Phi(.)$ is the standard Gaussian cdf. Consider a prior $\beta \sim N(0, 10^2(X^{'}X)^{-1})$. We are interested in $p(\beta|Y)$.

```
library(survival)
data("infert")
infert$education <- infert$education %>% as.numeric() - 1
df <- infert %>%
  select(c("case", "education", "age", "parity", "induced", "spontaneous"))
```

(1)

Describe and implement an adaptive Metropolis-Hastings algorithm designed to obtain a MC with stationary distribution $p(\beta|Y)$.

We know that

$$\begin{aligned} P(\beta|Y) &= P(Y|\beta)P(\beta) \\ &= (\prod_{i=1}^{n} \Phi(X_i^{'}\beta)^{y_i}(1-\Phi(X_i^{'}\beta))^{1-y_i})P(\beta) \\ &\propto (\prod_{i=1}^{n} \Phi(X_i^{'}\beta)^{y_i}(1-\Phi(X_i^{'}\beta))^{1-y_i}) * \exp(-\frac{1}{2}\beta^{'}\sum\beta) \end{aligned}$$

Then we design a adaptive proposal:

- $\beta_{t+1}|\beta_t \sim N(\beta_t, c * \sum_t)$

in which $\sum_t = \frac{1}{t}\sum_{j=1}^{t}\beta_j\beta_j^{'}$.

According to (Roberts and Rosenthal, 2009), we also set,

1) $\tilde{\sum}_t = \sum_t + \epsilon I$ to avoid degeneracies
2) $Q(\beta_t, \beta_{t+1}) = \delta N(\beta_t, c * \tilde{\sum}_t) + (1-\delta)N(\beta_t, \sum_0)$ in which $\delta \sim Binomial(p_\delta)$. here we set $p_\delta = 0.7$, and set $\sum_0$ same as the prior $\sum$ of β.

```
bvar_prior <- solve(t(as.matrix(df[,-1])) %*% as.matrix(df[,-1])) * 100
bvar_prior_inv <- t(as.matrix(df[,-1])) %*% as.matrix(df[,-1]) / 100
.cal_ll <- function(beta){
  est <- data.frame(est=pnorm(as.matrix(df[,-1]) %*% t(beta)), y=df[,1])
  out <- apply(est, 1, function(x){ifelse(x[2]==0,1-x[1],x[1])})
  out <- log(out) %>% sum - 1/2 * beta %*% bvar_prior_inv %*% t(beta)
}

mh.sim <- function(nsim=1000, burn=0, delta=0.8, c=3, seed=1996){
  set.seed(seed)

  num_beta <- dim(df)[2]-1
  nsim.total <- nsim*(1.0 + burn)
  burn.num <- nsim*burn
  delta_set <- rbinom(nsim.total, 1, prob = delta)

  beta <- matrix(rep(0,num_beta),1)
  beta.ch <- matrix(NA,nsim,num_beta)
  bvar <- diag(num_beta)*10^(-15)

  for(i in 1:nsim.total){
    if(i <= 1500){ #used adaptive algorithm on the first 1500 simulations
      bvar <- (bvar * (i - 1) + t(beta) %*% beta)/i + diag(num_beta)*10^(-15)
      beta_temp_2 <- rmvnorm(n=1, mean = beta, sigma = bvar_prior)
      delta_temp <- delta_set[i]
    }else{
      delta_temp <- 1
    }
    beta_temp_1 <- rmvnorm(n=1, mean = beta, sigma = c*bvar)
    beta_temp <- beta_temp_1 * delta_temp + beta_temp_2 * (1-delta_temp)
    P0 <- .cal_ll(beta)
    P1 <- .cal_ll(beta_temp)
    ratio <- P1 - P0
    if(log(runif(1)) < ratio){
      beta <- beta_temp
    }
    if(i > burn.num){
      i1 <- i - burn.num
      beta.ch[i1,] = beta
    }
  }
  return(list(beta=beta.ch))
}
```

simulation begin

```
burn=0.3
nsim=5000

# simulation
mh.mcmc.sim <- mh.sim(nsim=nsim, burn=burn, delta=0.7, c=1)

# get results
```

```
results.mh <- apply(mh.mcmc.sim$beta, 2,
                    function(x){
                      quantile(x, c(0.025, 0.5, 0.975))}) %>% round(.,2) %>%
              apply(., 2, function(x){paste0(x[2]," (", x[1],", ",x[3],")")})
```

(2)

Describe and implement a data augmented (DA-MCMC) strategy targeting $p(\beta|Y)$.

We add a new parameter z here, which makes

$$\begin{aligned} Y_i|z_i &\sim I(z_i > 0) \\ z_i|\beta, \tau_i &\sim N(X_i\beta, 1) \end{aligned}$$

Then we can calculate the full posterior distribution for β,

$$\begin{aligned} P(\beta|z,\tau) &= P(z|\beta) * P(\beta) \\ &\propto \prod \exp(-\frac{(z_i - X_i\beta)^2}{2}) * \exp(-\frac{1}{2}\beta^{'} \sum \beta) \\ &= \exp(-\frac{1}{2}(\beta^{'} X^{'} X\beta - 2 * \beta X^{'} z) - \frac{1}{2} * \beta^{'} \sum \beta) \\ &= \exp(-\frac{1}{2}(\beta^{'}(X^{'}X + \sum)\beta - 2 * \beta X^{'} z)) \end{aligned}$$

This is a normal kernel. Then we found that $\beta|z \sim N(\frac{X^{'}z}{X^{'}X+\sum}, \frac{1}{X^{'}X+\sum})$.

As for z,

$$\begin{aligned} P(z_i|\beta, Y) &= P(Y_i|z_i) * P(z_i|\beta) \\ &= I(z_i > 0) * \exp(-\frac{(z_i - X_i\beta)^2}{2}) \text{ (if Yi=1)} \\ &= I(z_i \leq 0) * \exp(-\frac{(z_i - X_i\beta)^2}{2}) \text{ (if Yi=0)} \end{aligned}$$

Then we found that z_i follows truncated normal distribution with mean $X_i\beta$ and variance 1.

```
da.mh.sim <- function(nsim=1000, burn=0, seed=1996,
                      X=as.matrix(df[,-1]),
                      Y=as.matrix(df[,1])){
  set.seed(seed)

  num_beta <- dim(df)[2]-1
  nsim.total <- nsim*(1.0 + burn)
  burn.num <- nsim*burn

  beta <- matrix(rep(0,num_beta),1)
  beta.ch <- matrix(NA,nsim,num_beta)
  z <- rep(0.5, length(Y))

  range <- cbind(ifelse(Y==0, -Inf, 0),
                 ifelse(Y==0, 0, Inf))
```

```
  for(i in 1:nsim.total){
    #i=1
    #z
    z_info <- as.matrix(cbind(X%*%t(beta), # z_mean
                              1, # z_sd
                              range),) # truncated range
    z <- apply(z_info, 1,
               function(info){rtruncnorm(1, a=info[3], b=info[4],
                                         mean = info[1], sd = info[2])})

    #beta
    bmean <- solve(t(X)%*%X+bvar_prior)%*%t(X)%*%z
    bvar <- solve(t(X)%*%X+bvar_prior)
    beta <- rmvnorm(n=1, mean = bmean, sigma = bvar)

    if(i > burn.num){
      i1 <- i - burn.num
      beta.ch[i1,] = beta
    }
  }
  return(list(beta=beta.ch))
}
```

simulation begin

```
burn=0.3
nsim=5000

# simulation
da.mh.mcmc.sim <- da.mh.sim(nsim=nsim, burn=burn)

# get results
results.da <- apply(da.mh.mcmc.sim$beta, 2,
                    function(x){
                      quantile(x, c(0.025, 0.5, 0.975))}) %>% round(.,2) %>%
              apply(., 2, function(x){paste0(x[2]," (", x[1],", ",x[3],")")})
```

(3)

Describe and implement a parameter expanded - data augmentation (PX-DA MCMC) algorithm targeting $p(\beta|Y)$.

Here we add w and α, which follow

$$\begin{aligned} W_i|\beta,\alpha &\sim N(X_i^{'}\beta\alpha, \alpha^2) \\ Y_i|W_i &= I(W_i > 0) \\ \alpha^2 &\sim IG(a,b) \end{aligned}$$

Then we can calculate the full posterior distribution for β,

$$
\begin{aligned}
P(\beta|w,\alpha) &= P(w|\beta,\alpha) * P(\beta) \\
&\propto \prod \exp(-\frac{(w_i - \alpha X_i\beta)^2}{2\alpha^2}) * \exp(-\frac{1}{2}\beta^{'}\sum\beta) \\
&= \exp(-\frac{1}{2\alpha^2}(\alpha^2\beta^{'} * X^{'}X\beta - 2 * \alpha\beta X^{'}w) - \frac{1}{2} * \beta^{'}\sum\beta) \\
&= \exp(-\frac{1}{2}(\beta^{'}(X^{'}X + \sum)\beta - 2 * \beta X^{'}w/\alpha))
\end{aligned}
$$

This is a normal kernel. Then we found that $\beta|z,\tau \sim N(\frac{X^{'}w}{\alpha X^{'}X+\alpha\sum}, \frac{1}{X^{'}X+\sum})$.

As for w,

$$
\begin{aligned}
P(w_i|\beta,\alpha,Y_i) &= P(Y_i|w_i) * P(w_i|\beta,\alpha) \\
&= I(w_i > 0) * \exp(-\frac{(w_i - \alpha X_i\beta)^2}{2\alpha^2}) \text{ (if Yi=1)} \\
&= I(w_i \leq 0) * \exp(-\frac{(w_i - \alpha X_i\beta)^2}{2\alpha^2}) \text{ (if Yi=0)}
\end{aligned}
$$

Then we found that w_i follows truncated normal distribution with mean $\alpha X_i\beta$ and variance α^2.

As for α^2,

$$
\begin{aligned}
P(\alpha^2|\beta,w,Y) &= P(w|\alpha^2,\beta) * P(\alpha^2) \\
&= (\frac{1}{\sqrt{2\pi\alpha^2}})^n \exp(-\frac{(w - \alpha X\beta)^2}{2\alpha^2}) * (\alpha^2)^{-a-1}\exp(-b/\alpha^2) \\
&\propto (\alpha^2)^{-a-n/2-1}\exp(-(\frac{(w - \alpha X\beta)^2}{2} + b)/\alpha^2)
\end{aligned}
$$

This is difficult to treat it as a specific kernel. Then we consider use MH method to simulate α^2. Here we use truncated normal proposal.

```
.alpha.sq.ll <- function(alpha.sq,beta,w,a,b,X,n){
  out <- (-a-n/2-1)*log(alpha.sq)-
    ((2*b+t(w-sqrt(alpha.sq)*X%*%t(beta))%*%
        (w-sqrt(alpha.sq)*X%*%t(beta)))/2)/alpha.sq
  return(out)
}
px.da.mh.sim <- function(nsim=1000, burn=0, as_sd=1,
                         a=3, b=3, seed=1996,
                         X=as.matrix(df[,-1]),
                         Y=as.matrix(df[,1])){
  set.seed(seed)

  num_beta <- dim(df)[2]-1
  nsim.total <- nsim*(1.0 + burn)
  burn.num <- nsim*burn

  beta <- matrix(rep(0,num_beta),1)
  beta.ch <- matrix(NA,nsim,num_beta)
  alpha.sq <- 1
  n <- length(Y)
```

```
  range <- cbind(ifelse(Y==0, -Inf, 0),
                 ifelse(Y==0, 0, Inf))

  for(i in 1:nsim.total){
    #i=1
    #w
    w_info <- as.matrix(cbind(sqrt(alpha.sq)*X%*%t(beta), # w_mean
                              sqrt(rep(alpha.sq,length(Y))), # w_sd
                              range),) # truncated range
    w <- apply(w_info, 1,
               function(info){rtruncnorm(1, a=info[3], b=info[4],
                                         mean = info[1], sd = info[2])})

    #beta
    bmean <- solve(t(X)%*%X*sqrt(alpha.sq)+
                     bvar_prior*sqrt(alpha.sq))%*%t(X)%*%w
    bvar <- solve(t(X)%*%X+bvar_prior)
    beta <- rmvnorm(n=1, mean = bmean, sigma = bvar)

    #alpha.sq
    as_temp <- rtruncnorm(1, a=0, b=Inf, mean = alpha.sq, sd = as_sd)
    P0 <- .alpha.sq.ll(alpha.sq,beta,w,a,b,X,n)
    P1 <- .alpha.sq.ll(as_temp,beta,w,a,b,X,n)
    ratio <- P1 - P0 - log(dtruncnorm(as_temp,mean = alpha.sq, sd = as_sd)) +
    log(dtruncnorm(alpha.sq,mean = as_temp, sd = as_sd))

    if(log(runif(1)) < ratio){
      alpha.sq <- as_temp
    }

    if(i > burn.num){
      i1 <- i - burn.num
      beta.ch[i1,] <- beta
    }
  }
  return(list(beta=beta.ch))
}
```

simulation begin (set as_sd=1, a=3, b=3)

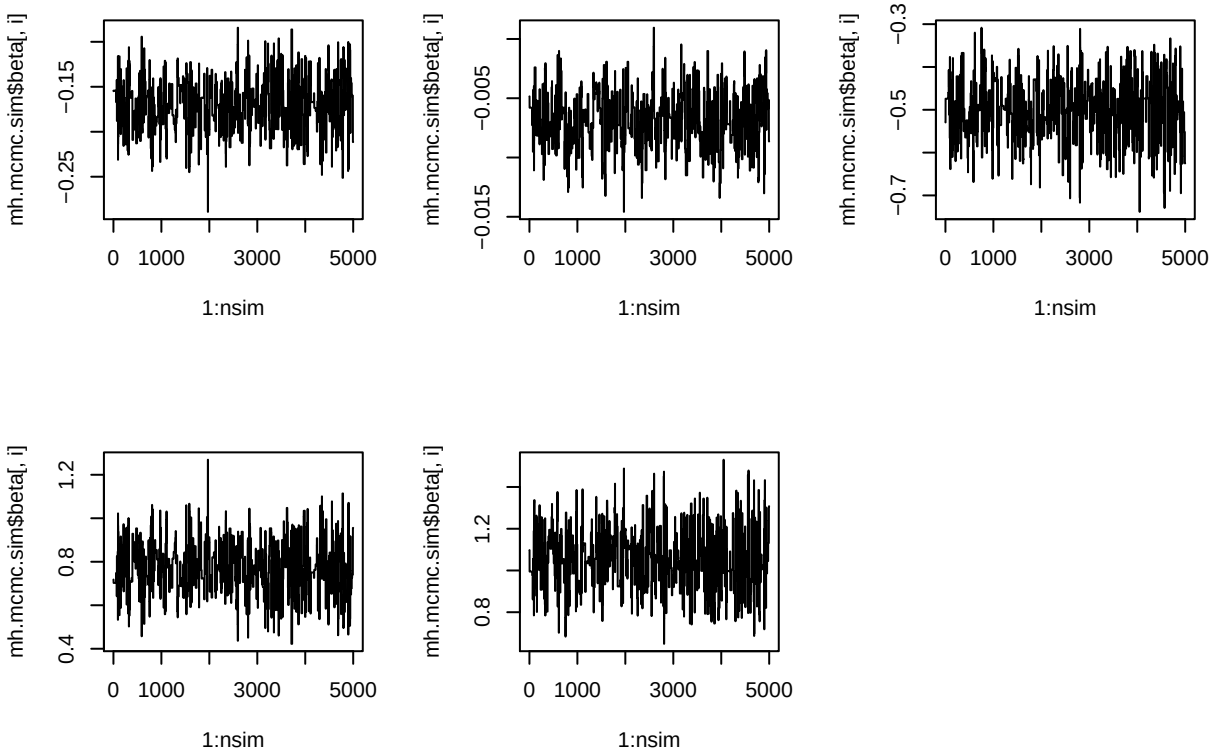
```
burn=0.3
nsim=5000

# simulation
px.da.mh.mcmc.sim <- px.da.mh.sim(nsim=nsim, burn=burn,as_sd=1)

# get results
results.px <- apply(px.da.mh.mcmc.sim$beta, 2,
                    function(x){
                      quantile(x, c(0.025, 0.5, 0.975))}) %>% round(.,2) %>%
              apply(., 2, function(x){paste0(x[2]," (", x[1],", ",x[3],")")})
```

(4)

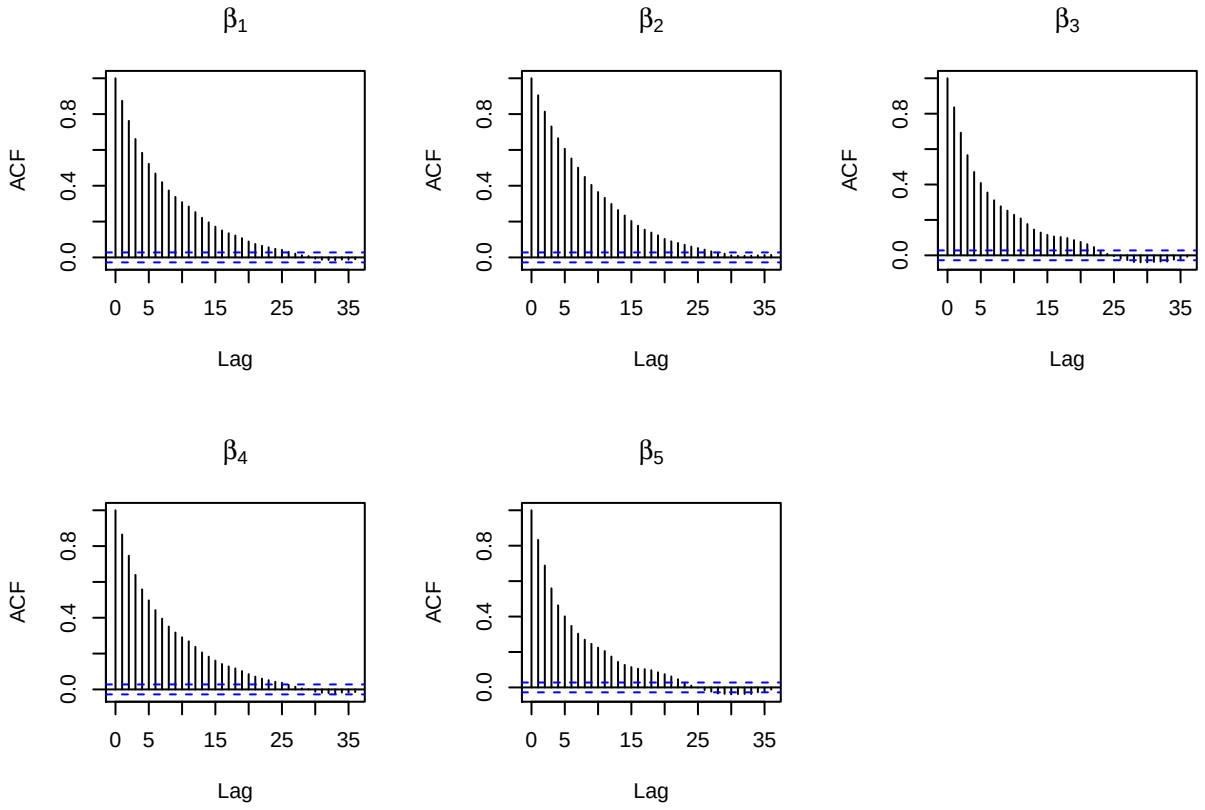
Assess mixing and convergence of the chains induced by the competing transition schemes implemented in 1,2 and 3. Comment on potential trade-offs involving: coding complexity, storage and cpu time.

Check mixing and convergence of the chains:

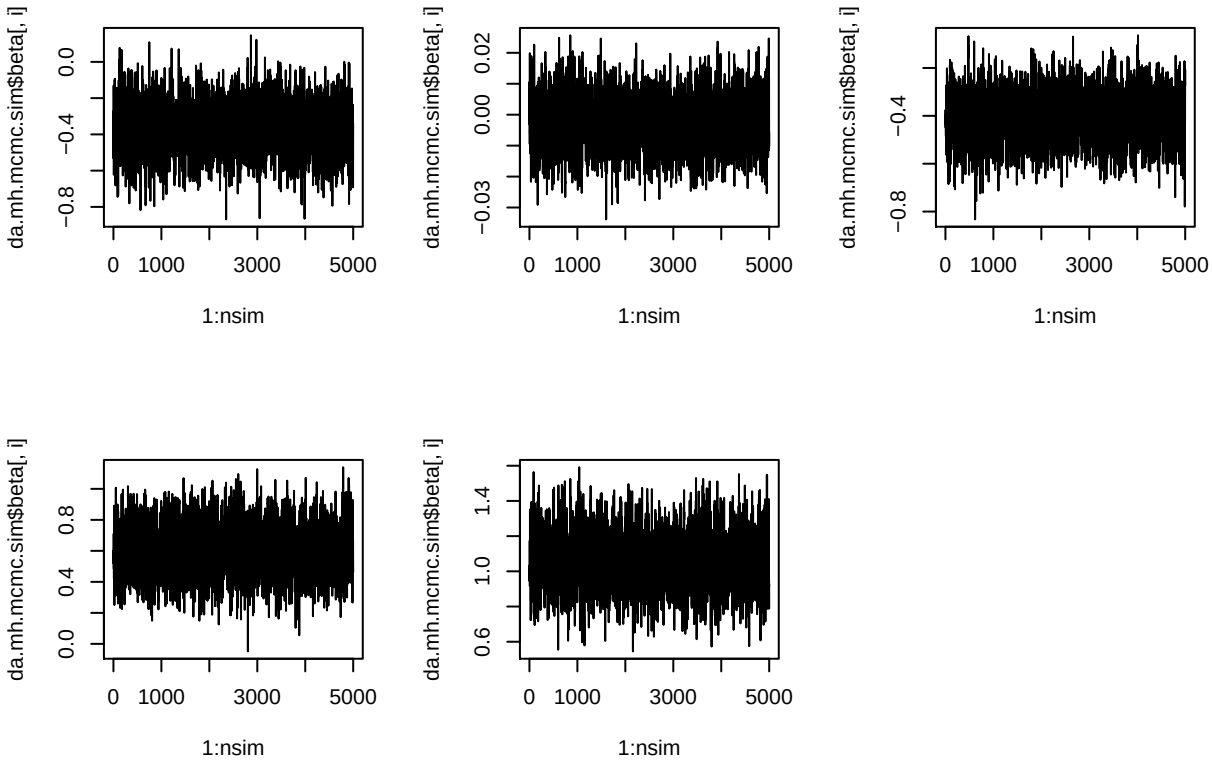
```
# plots for mh method
par(mfrow=c(2,3))
for(i in 1:5){
  plot(1:nsim, mh.mcmc.sim$beta[,i], type="l")
}
par(mfrow=c(2,3))
```



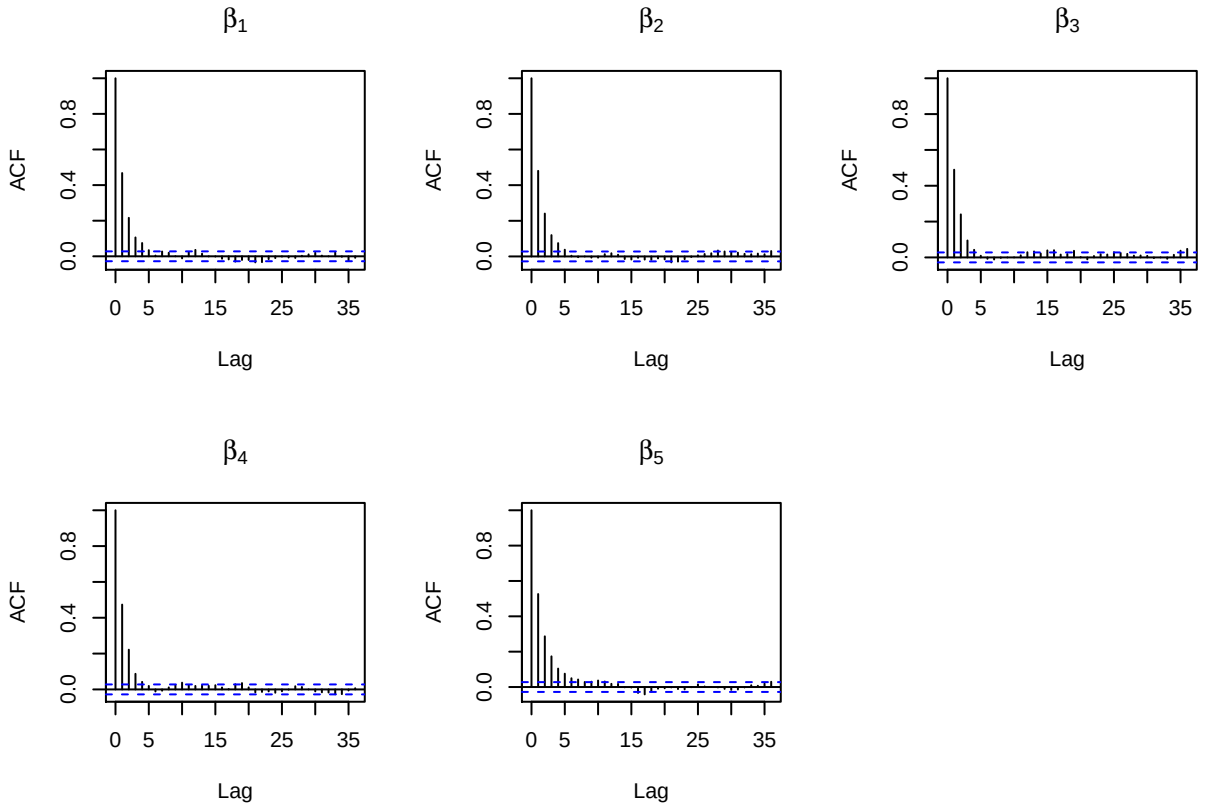

```
for(i in 1:5){
  acf(mh.mcmc.sim$beta[,i],main = substitute(beta[x], list(x = i)))
}

# plots for da-mh method
par(mfrow=c(2,3))
```



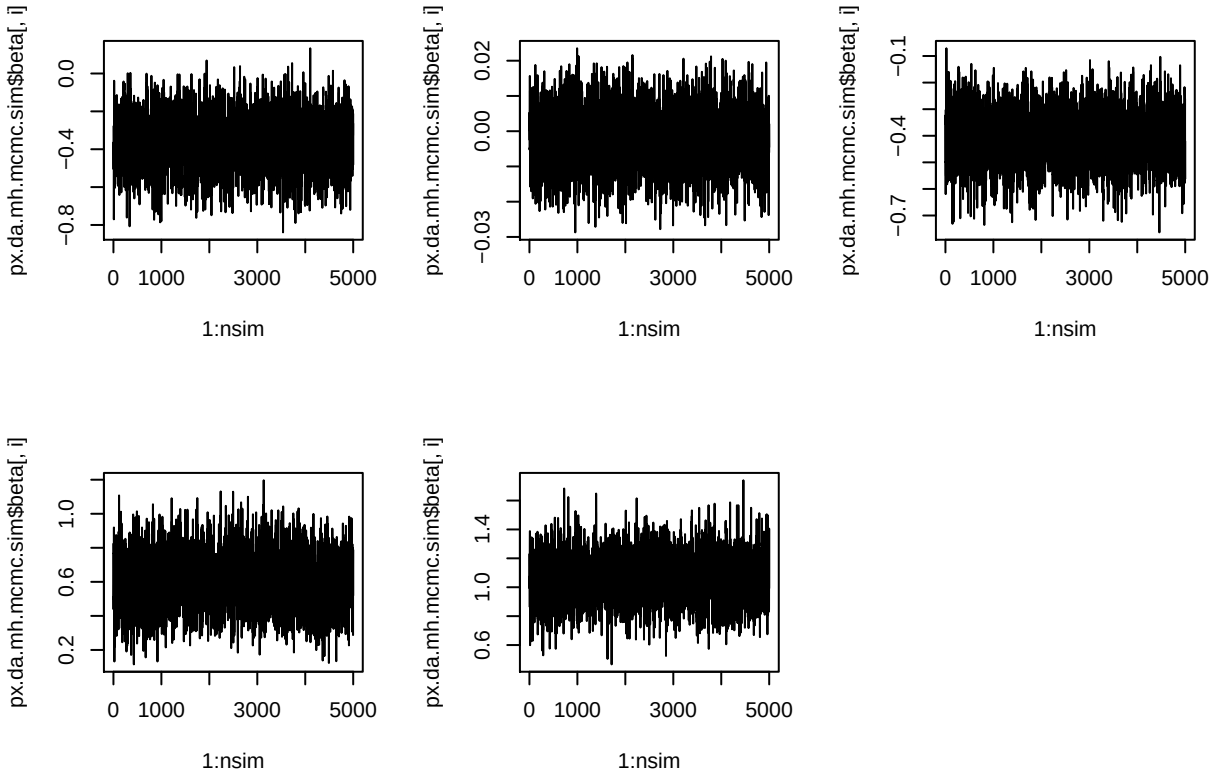

```
for(i in 1:5){
  plot(1:nsim, da.mh.mcmc.sim$beta[,i], type="l")
}
par(mfrow=c(2,3))
```




```
for(i in 1:5){
  acf(da.mh.mcmc.sim$beta[,i], main = substitute(beta[x], list(x = i)))
}

# plots for px-da-mh method
par(mfrow=c(2,3))
```




```
for(i in 1:5){
  plot(1:nsim, px.da.mh.mcmc.sim$beta[,i], type = "l")
}
par(mfrow=c(2,3))
```



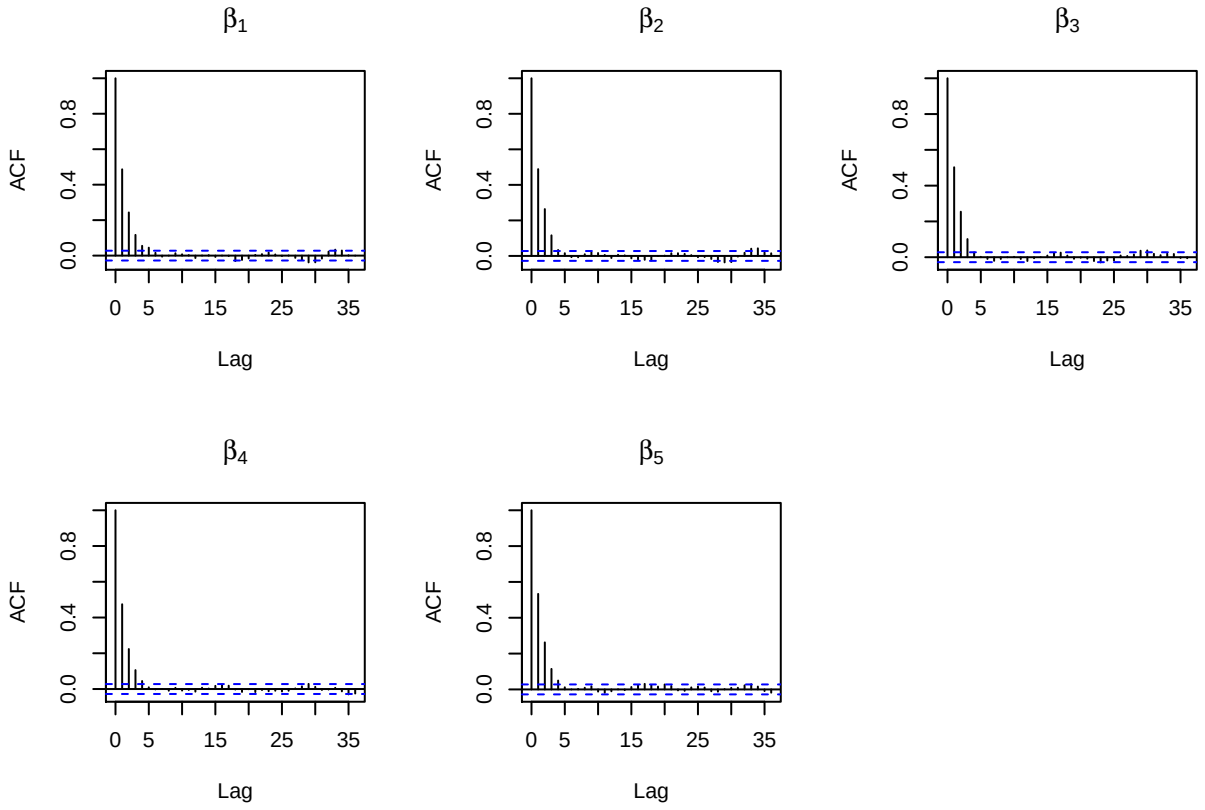

```
for(i in 1:5){
  acf(px.da.mh.mcmc.sim$beta[,i], main = substitute(beta[x], list(x = i)))
}

results <- rbind(results.mh, results.da, results.px) %>% as.data.frame
colnames(results) <- colnames(df)[-1]
results %>%
  kbl(caption = "Beta Summary Table (95% CI)") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

\begin{table}

\caption{Beta Summary Table (95% CI)}

	education	age	parity	induced	spontaneous
results.mh	-0.17 (-0.23, -0.12)	-0.01 (-0.01, 0)	-0.5 (-0.64, -0.37)	0.76 (0.55, 1.02)	1.05 (0.79, 1.32)
results.da	-0.36 (-0.63, -0.1)	0 (-0.02, 0.01)	-0.41 (-0.6, -0.22)	0.61 (0.31, 0.92)	1.06 (0.77, 1.37)
results.px	-0.37 (-0.64, -0.11)	0 (-0.02, 0.01)	-0.4 (-0.6, -0.22)	0.61 (0.31, 0.92)	1.06 (0.75, 1.38)

\end{table}



As we can tell from the plots, auto correlation of chain generated by `adaptive-mh` method decrease much slower than that of `px-da-mh` and `da-mh` method. In addition, auto correlation for some parameters' chains (e.g. β_5) of `px-da-mh` decrease slightly faster than that of `da-mh` method. As for values of βs, three methods provided largely consistent estimates.

Calculate memory usage and running time of the chain:

```
# set same situation
burn=0.3
nsim=500

# calculate memory usage
mem.mh <- profmem(mh.sim(nsim=nsim, burn=burn))
mem.da.mh <- profmem(da.mh.sim(nsim=nsim, burn=burn))
mem.px.da.mh <- profmem(px.da.mh.sim(nsim=nsim, burn=burn))

mem_use <- data.frame(method=c("adaptive-mh","da.mh","px.da.mh"),
                      memory_usage=c(mem.mh$bytes %>% sum(na.rm = T),
                                     mem.da.mh$bytes %>% sum(na.rm = T),
                                     mem.px.da.mh$bytes %>% sum(na.rm = T)))
mem_use %>%
  kbl(caption = "Memory Usage Summary Table") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

Table 1: Memory Usage Summary Table

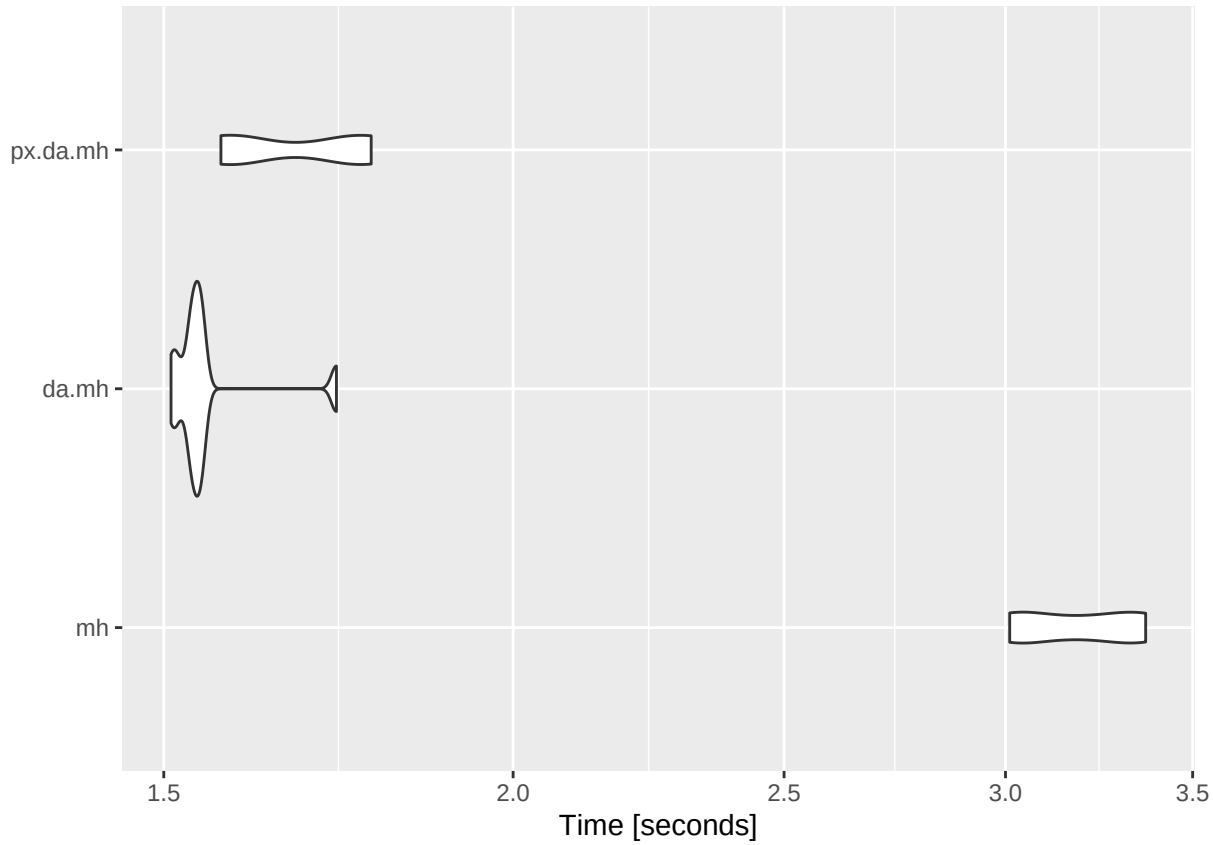
method	memory_usage
adaptive-mh	58159376
da.mh	460175672
px.da.mh	472736840

```
# calculate running time
stats <- microbenchmark(mh = {mh.sim(nsim=nsim, burn=burn)},
                        da.mh = {da.mh.sim(nsim=nsim, burn=burn)},
                        px.da.mh = {px.da.mh.sim(nsim=nsim, burn=burn)},
                        times = 10, unit = "ms")
stats
```

```
## Unit: milliseconds
##      expr      min       lq     mean   median       uq      max neval cld
##        mh 3010.472 3028.619 3185.825 3184.099 3341.571 3367.114    10   b
##     da.mh 1508.983 1530.647 1554.086 1538.988 1548.044 1729.210    10  a
##  px.da.mh 1572.284 1582.896 1674.338 1671.075 1763.771 1779.397    10  a
```

```
autoplot(stats)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



As we can tell from the memory usage summary table, `adaptive-mh` method used the smallest memory. `da.mh` and `px.da.mh` methods used very similar memory. As for the efficiency, `da.mh` and `px.da.mh` have similar running time, which is nearly half of `adaptive-mh` method's.

Conclusion: `px.da.mh` method had the best convergence of the chains. `da.mh` method operated most efficient and `adaptive-mh` method occupied the smallest memory.