“MCMC_R”

이영노

March 23, 2023

MCMC

- MCMC is a method that “generates” a sequence of “dependent samples” from “target distribution” and computes quantities by using Monte Carlo based on these samples.

  Here, Monte Carlo method is based on Markov Chains.

- MCMC generates sequence of correlated samples X1, X2,…that range through the region of high probability by making a sequence of incremental movements
- Under conditions that follows “Ergodic Theorem”, these approximations are guaranteed to converge to the true value.
- How? With Metropolis-Hastings Algorithm, or Gibbs sampler(special case of MH-Algorithm)

Gibbs Sampling

- Gibbs Sampling is an MCMC algorithm for obtaining approximate draws from a “joint distribution”, based on sampling from “conditional distributions”, “one at a time”. : at each stage, one variable is updated (keeping all the other variables fixed) by drawing from the conditional distribution of that variable given all the other variables.

Gibbs sampling is used at a condition when…

1. each one of full conditional distributions is available to sample from.
2. when we can’t directly draw samples, for example in the case with complicated distributions in high-dimensional spaces.

Gibbs sampling generates sample by…

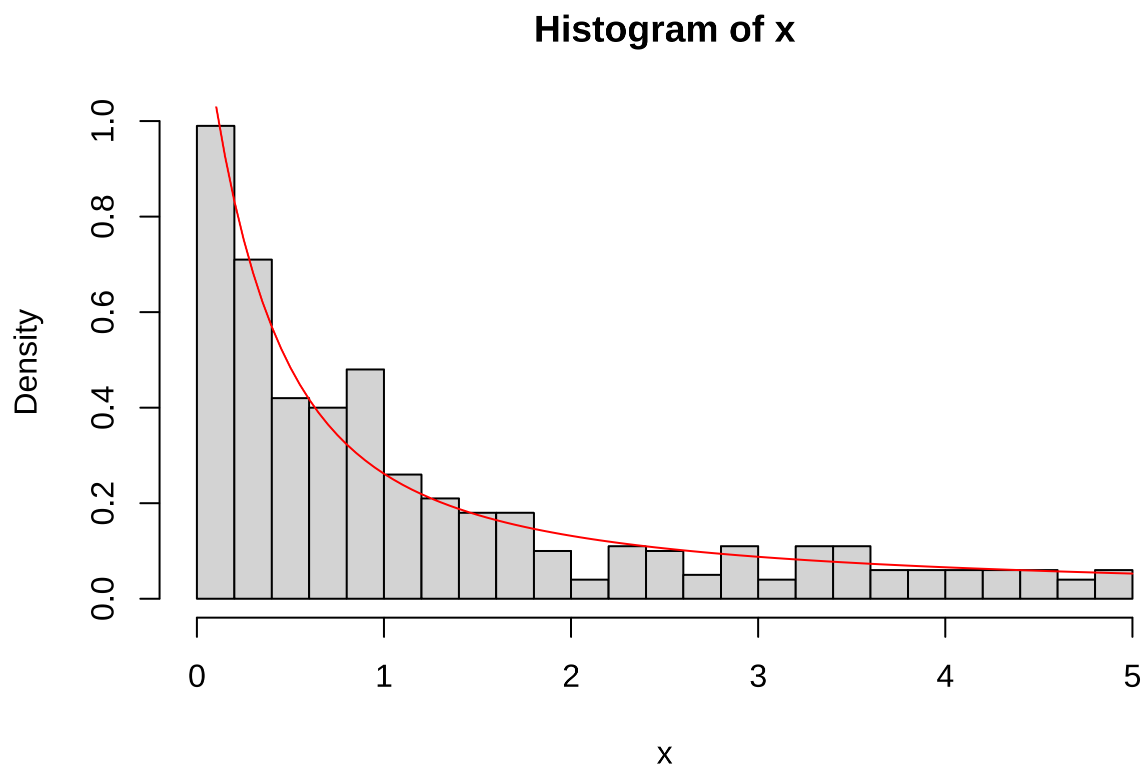
- Decompose joint distribution into conditional distribution in a form that is easy to sample from.
- Update Variables
- Iterate many times, then get **Stationary Distribution of the Markov Chain** which is same as joint distribution (target distribution) that we want to acquire. (we want to specify and find out the distribution)

Example1 : Exponential Distribution (symmetric in respect to x,y)

```
m <- 500; k <- 15; B <- 5

# sample from Exp(X|y=y)*I(X<B) : Likelihood
invcdfx <- function(u,y,B){ log(1-u*(1-exp(-B*y)))/(-y)}

# since joint distribution is symmetric to x,y, we only need to derive above,
# and switch x,y to y,x.
# Conduct gibbs sampling
rng_gibbs <- function(k,B){
  x.tmp = y.tmp = matrix(nrow=k+1) # save empty space for iteration
  y.tmp[1] = runif(1,0,B); u1 = runif(1,0,1) # initial y from unif(0,B)
  x.tmp[1] = invcdfx(u1, y.tmp[1], B)
  # initial x from Exp(x|y)*I(X<B) y is like theta in Exponential dist.
  # iterate many times to get Stationary Distribution of Markov Chain
  for (j in 2:(k+1)){
    u2 <- runif(1,0,1); y.tmp[j] = invcdfx(u2, x.tmp[j-1],B) # P(X|Y)
    u3 <- runif(1,0,1); x.tmp[j] = invcdfx(u3, y.tmp[j],B)
  }
  return (c(x.tmp[k+1],y.tmp[k+1]))
}

xy = replicate(m,rng_gibbs(k,B)); xy <- t(xy) # conduct 500 times. xy is 2*1 matrix
x <- xy[,1]; y <- xy[,2];
hist(x,breaks=30,prob=TRUE) # MCMC Gibbs Sampling 통해 추출된 X의 분포 (위 코드에는 burn-in 없음)
require(pracma)
```

```
## 필요한 패키지를 로딩중입니다: pracma

## Warning: 패키지 'pracma'는 R 버전 4.1.3에서 작성되었습니다
```

```
# compare MCMC histogram with real function mar.x
mar.x = function(x){
  fun = function(x,y){exp(-x*y)};
  const = 1/integral2(fun,0,B,0,B)$Q
  const*(1/x*(1-exp(-B*x)))
}
curve(mar.x(x),0,5,add=T,col="red")
```



Example 2 : Multivariate Normal Distribution (non-symmetric)

① Sample $\theta_1^{(n)}|\theta_2^{(n-1)}, (x_1, x_2)^T \sim N(x_1 + \rho(\theta_2^{(n-1)} - x_2), 1 - \rho^2);$

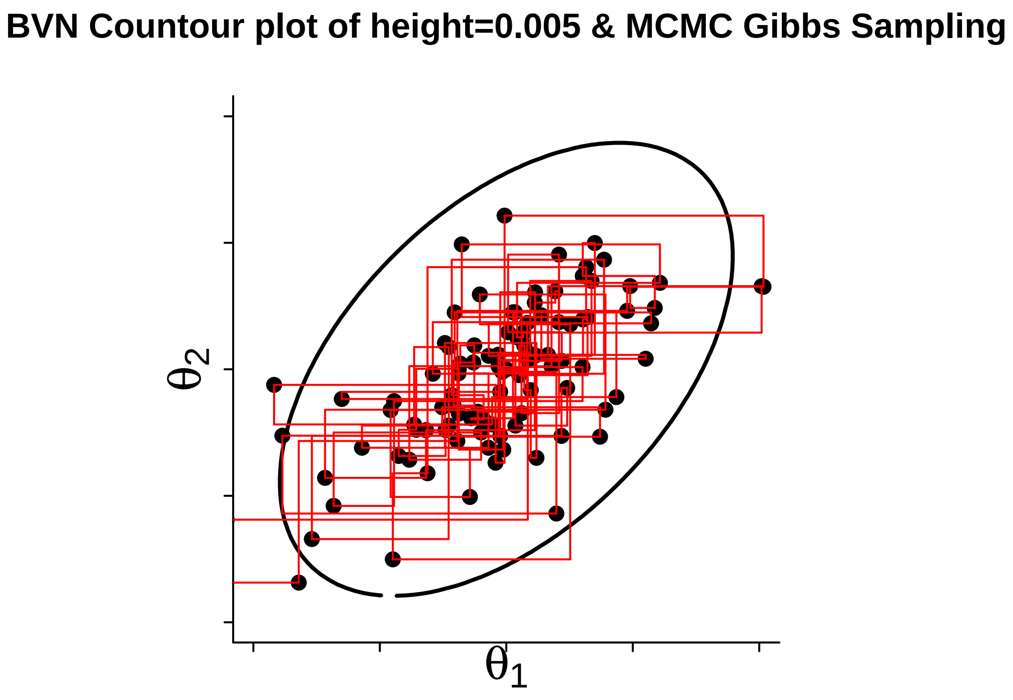
② Sample $\theta_2^{(n)}|\theta_1^{(n)}, (x_1, x_2)^T \sim N(x_2 + \rho(\theta_1^{(n)} - x_1), 1 - \rho^2).$

Figure 1: BVN conditional dist.

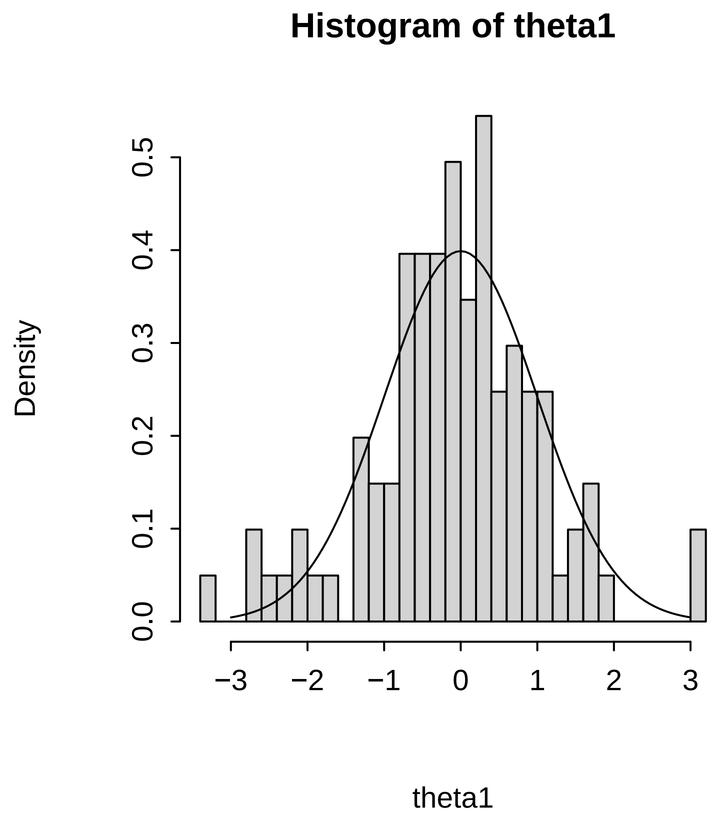
```
library(mnormt)
```

```
## Warning: 패키지 'mnormt'는 R 버전 4.1.3에서 작성되었습니다
```

```
rho=0.5; Sig=matrix(c(1,rho,rho,1),2,2)
theta1=seq(-3,3,0.1); theta2=seq(-3,3,0.1);
n=length(theta1); z=matrix(0,n,n) # empty space for iteration, trace record

# Real Distribution of Bivariate Normal Distribution
for (i in 1:n){z[i,]=dmnorm(cbind(theta1[i],theta2),c(0,0),Sig)}
par(cex=0.9); par(tcl=-0.25); par(mgp=c(4,0.6,0)); par(pty='s');
contour(theta1,theta2,z,levels=seq(0.005,0.005),
labels=" ",lwd=2,axes=F)
axis(1,at=c(-4,-3,-1.5,0,1.5,3,4),labels=rep("",7))
axis(2,at=c(-4,-3,-1.5,0,1.5,3,4),labels=rep("",7))
mtext(expression(theta[2]),side=2,cex=1.5,line=0.5)
mtext(expression(theta[1]),side=1,cex=1.5,line=0.5)
title(main='BVN Countour plot of height=0.005 & MCMC Gibbs Sampling')

# MCMC Gibbs Sampling
iter=101; theta1=numeric(iter); theta2=numeric(iter)
for (i in 2:iter){
theta1[i]=rnorm(1,rho*theta2[i-1],sqrt(1-rho^2))
theta2[i]=rnorm(1,rho*theta1[i],sqrt(1-rho^2))}
points(theta1,theta2,pch=19,lwd=1.5)
for (i in 1:iter){
segments(theta1[i],theta2[i],theta1[i+1],theta2[i],col="red")
segments(theta1[i+1],theta2[i],theta1[i+1],theta2[i+1],col="red")}
```



```
hist(theta1,nclass=30,freq=F); curve(dnorm(x,0,1),-3,3,add=T)
```



We generated Markov Chain (theta1_t, theta2_t) whose successive samples are correlated. If parameters(theta1,theta2) are strongly correlated, then we say that the Markov chain mixes slowly.

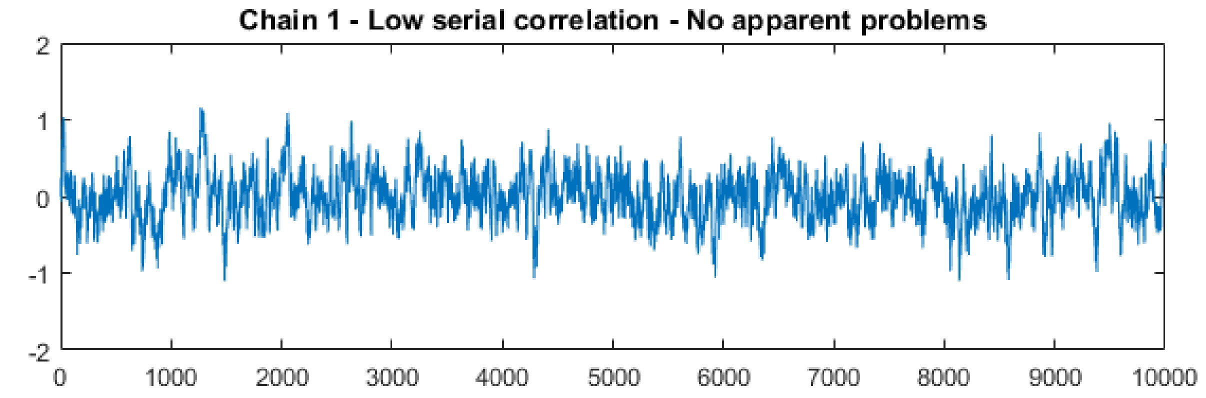
MCMC diagnotics

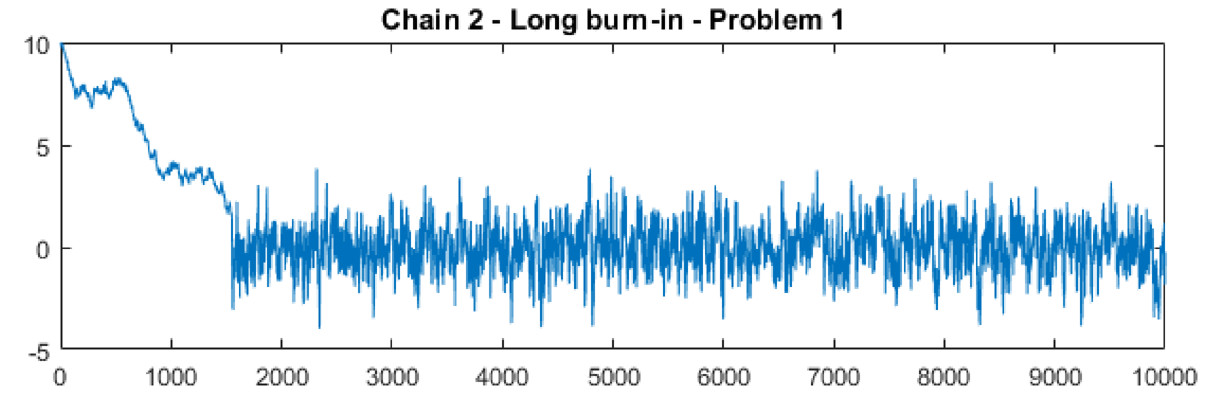
There are ISSUES surrounding MCMC diagnotics

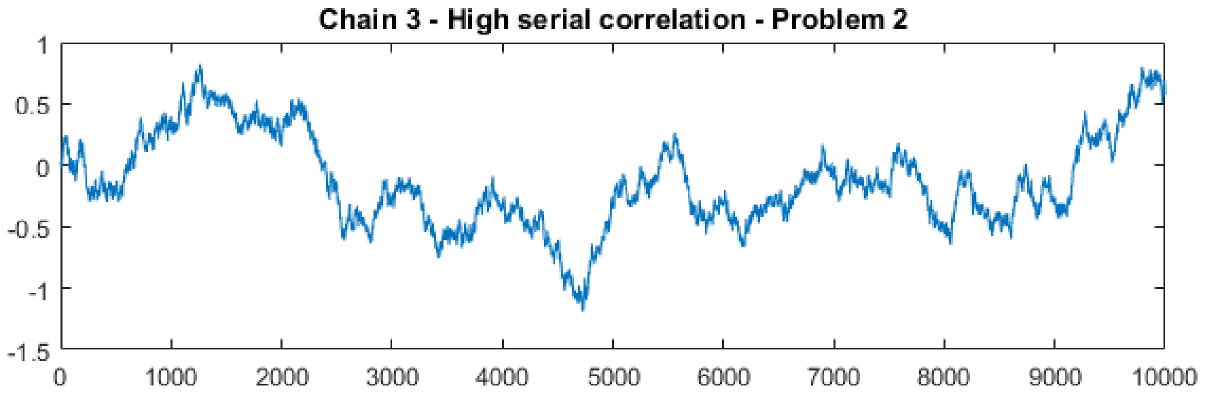
- Selecting the initial values : Do initial values matter?
- Determining convergence : How/when can we tell if the chain is not converging?
- Mixing refers to how quickly the chain moves around the support of the random vector. When mixing is poor, sample averages of Gibbs sampling output do not converge as quickly as they would if the data were independent. How rapidly does it mix? Did it explored sufficient amount of sample spaces?
- Selecting number of samples: How long do we need to run the MCMC to adequately approximate the posterior distribution? How many samples do we need to discard in initial (burn-in) steps? How to create independent parameter draws?

Plus, there are several problems that we will mention behind. That is,

- Problem 1 - A large portion of the sample is drawn from distributions that are significantly different from the target
- Problem 2 - The effective size of the sample is too small







In the first trace plot (Chain 1), there are no apparent anomalies. There seems to be a mild serial correlation between successive draws and the chain seems to explore the sample space many times.

In the second plot (Chain 2), the first part of the sample (until around $t = 1,500$) looks very different from the remaining part. Most likely, the initial distribution and the distributions of the subsequent terms of the chain were very different from the target distribution, but then the chain slowly converged to the target distribution (around $t = 1,500$). We have Problem 1: a large chunk of the sample is drawn from distributions that are significantly different from the target distribution.

In the third plot (Chain 3), there is a lot of serial correlation between successive draws. The chain is very slow in exploring the sample space. The sample space has been explored only few times. In other words, there seems to be few independent observations in our sample. Quite likely, we have Problem 2: the effective size of our sample is too small.

(reference : https://www.statlect.com/fundamentals-of- statistics/Markov-Chain-Monte-Carlo-diagnostics)

MCMC diagnostics : 1. Trace Plots to check mixing

MCMC diagnostics : 2. ACF

Large autocorrelations indicate that chain is not mixing well. i.e., the chaing possibly has not explored the full space of the posterior distribution(has not converged)

MCMC diagnostics total

```
# We will compare real mvn independent sampling vs.
# MCMC gibbs sampling

x1 <- 1; x2 <- -1; rho <- 0.9; x <- c(x1,x2)
Sigma = matrix(c(1,rho,rho,1),2,2); nDraws <- 10000
library(mnormt);

MC.samples <- rmnorm(nDraws, x, Sigma)
gibbs.samples <- matrix(0,nDraws,2); theta2 <- 0

# gibbs sampling
for (i in 1:nDraws){
theta1 <- rnorm(1, x1 + rho*(theta2-x2), sqrt(1-rho^2))
gibbs.samples[i,1] <- theta1
theta2 <- rnorm(1, x2 + rho*(theta1-x1), sqrt(1-rho^2))
gibbs.samples[i,2] <- theta2
}
print("for the last 1000 samples")
```

```
## [1] "for the last 1000 samples"
```

```
print("x1,x2 mean comparison")
```

```
## [1] "x1,x2 mean comparison"
```

```
apply(MC.samples[9000:10000,],2,mean);
```

```
## [1]  1.0438741 -0.9439465
```

```
apply(gibbs.samples[9000:10000,],2,mean);
```

```
## [1]  1.0941983 -0.9141747
```

```
print("covariance comparison")
```

```
## [1] "covariance comparison"
```

```
cov(MC.samples[9000:10000,]);
```

```
##             [,1]      [,2]
## [1,] 0.9520773 0.8403488
## [2,] 0.8403488 0.9259416
```

```
cov(gibbs.samples[9000:10000,]);
```

```
##           [,1]      [,2]
## [1,] 0.8550517 0.7711643
## [2,] 0.7711643 0.8880858
```

```
print("Real Sigma")
```

```
## [1] "Real Sigma"
```

```
Sigma
```

```
##      [,1] [,2]
## [1,]  1.0  0.9
## [2,]  0.9  1.0
```

```
index <- seq(3000,10000,10) # burn-in leftout indices

print("for the last 7000 samples")
```

```
## [1] "for the last 7000 samples"
```

```
print("x1,x2 mean comparison")
```

```
## [1] "x1,x2 mean comparison"
```

```
apply(gibbs.samples[index,],2,mean);
```

```
## [1]  0.9785067 -1.0251893
```

```
x;
```

```
## [1]  1 -1
```

```
print("covariance of burn-in leftout samples(final)")
```

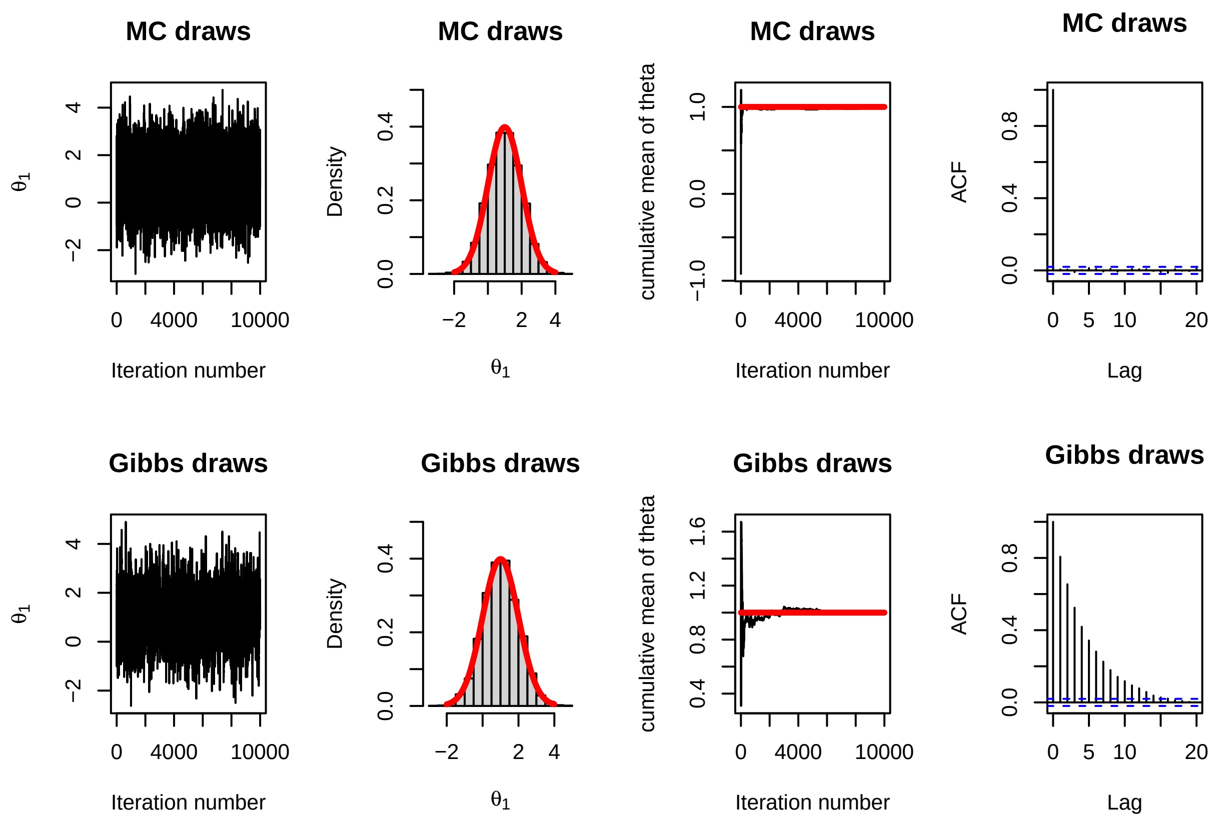
```
## [1] "covariance of burn-in leftout samples(final)"
```

```
cov(gibbs.samples[index,]);
```

```
##           [,1]      [,2]
## [1,] 0.9398437 0.8397009
## [2,] 0.8397009 0.9419464
```

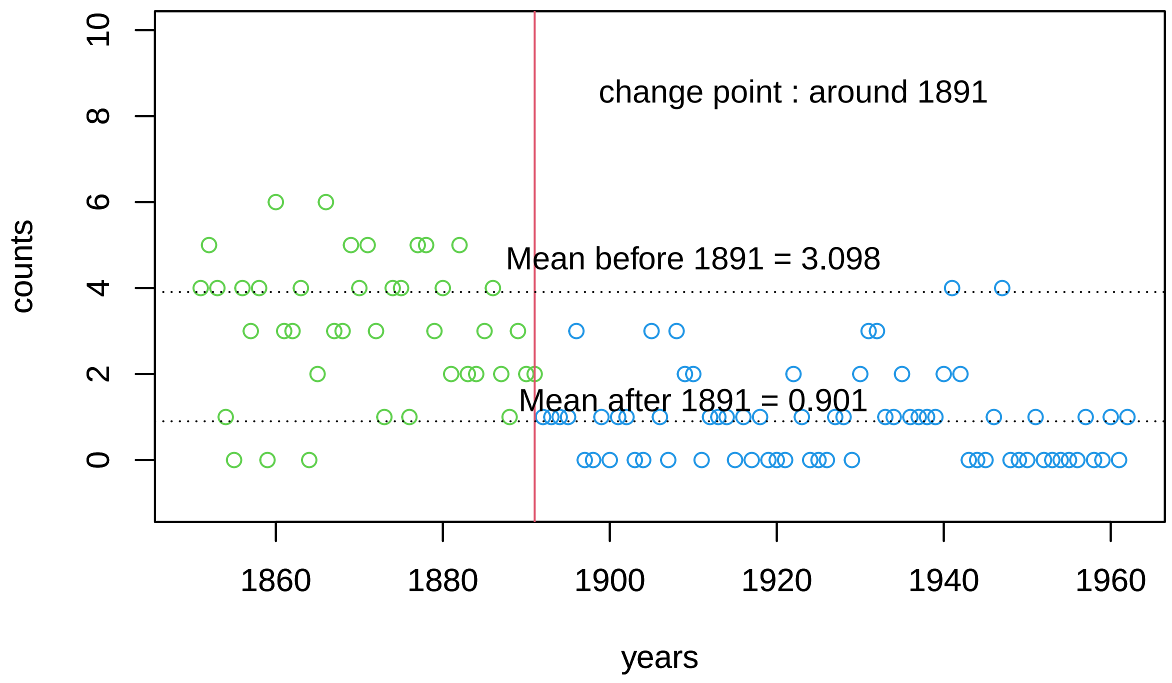
```
par(mfrow=c(2,4))
plot(MC.samples[,1], type="l", main='MC draws',
xlab='Iteration number', ylab=expression(theta[1]))
hist(MC.samples[,1], freq = FALSE, main='MC draws',
ylim = c(0,0.5), xlab=expression(theta[1]))
curve(dnorm(x,x1,1),-2,4,add=T,col="red",lwd=3)
plot(cumsum(MC.samples[,1])/seq(1,nDraws),type="l",main="MC draws",
xlab='Iteration number', ylab="cumulative mean of theta")
```

```
lines(seq(1,nDraws),1*matrix(1,1,nDraws),col="red",lwd=3)
acf(MC.samples[,1], main='MC draws', lag.max = 20)
plot(gibbs.samples[,1], type="l", main='Gibbs draws',
xlab="Iteration number", ylab=expression(theta[1]))
hist(gibbs.samples[,1], freq = FALSE, main="Gibbs draws",
ylim = c(0,0.5), xlab=expression(theta[1]))
curve(dnorm(x,x1,1),-2,4,add=T,col="red",lwd=3)
plot(cumsum(gibbs.samples[,1])/seq(1,nDraws),type="l",
main="Gibbs draws",xlab="Iteration number",
ylab="cumulative mean of theta")
lines(seq(1,nDraws),1*matrix(1,1,nDraws),col="red",lwd=3)
acf(gibbs.samples[,1], main='Gibbs draws', lag.max = 20)
```



# Change Point Analysis

```
x = c(4,5,4,1,0,4,3,4,0,6,3,3,4,0,2,6,3,3,5,4,5,3,1,4,4,1,5,5,
3,4,2,5,2,2,3,4,2,1,3,2,2,1,1,1,1,3,0,0,1,0,1,1,0,0,3,1,0,3,2,
2,0,1,1,1,0,1,0,1,0,0,0,2,1,0,0,0,1,1,0,2,3,3,1,1,2,1,1,1,1,2,
4,2,0,0,0,1,4,0,0,0,1,0,0,0,0,0,1,0,0,1,0,1)
year=1851:1962; n = length(x)
plot(year[1:41],x[1:41],col=3,pch=1,xlim=c(1850,1962),
ylim=c(-1,10),xlab="years",ylab="counts")
par(new=T)
plot(year[42:112],x[42:112],col=4,pch=1,xlim=c(1850,1962),
ylim=c(-1,10),xlab="years",ylab="counts"); abline(v=1891,col=2)
text(1922,8.5, "change point : around 1891")
text(1910,4.7,paste("Mean before 1891 = ",
round(mean(x[1:41]),3),sep="")); abline(h=3.908,lty=3)
text(1910,1.4,paste("Mean after 1891 = ",
round(mean(x[42:n]),3),sep="")); abline(h=0.901,lty=3)
```

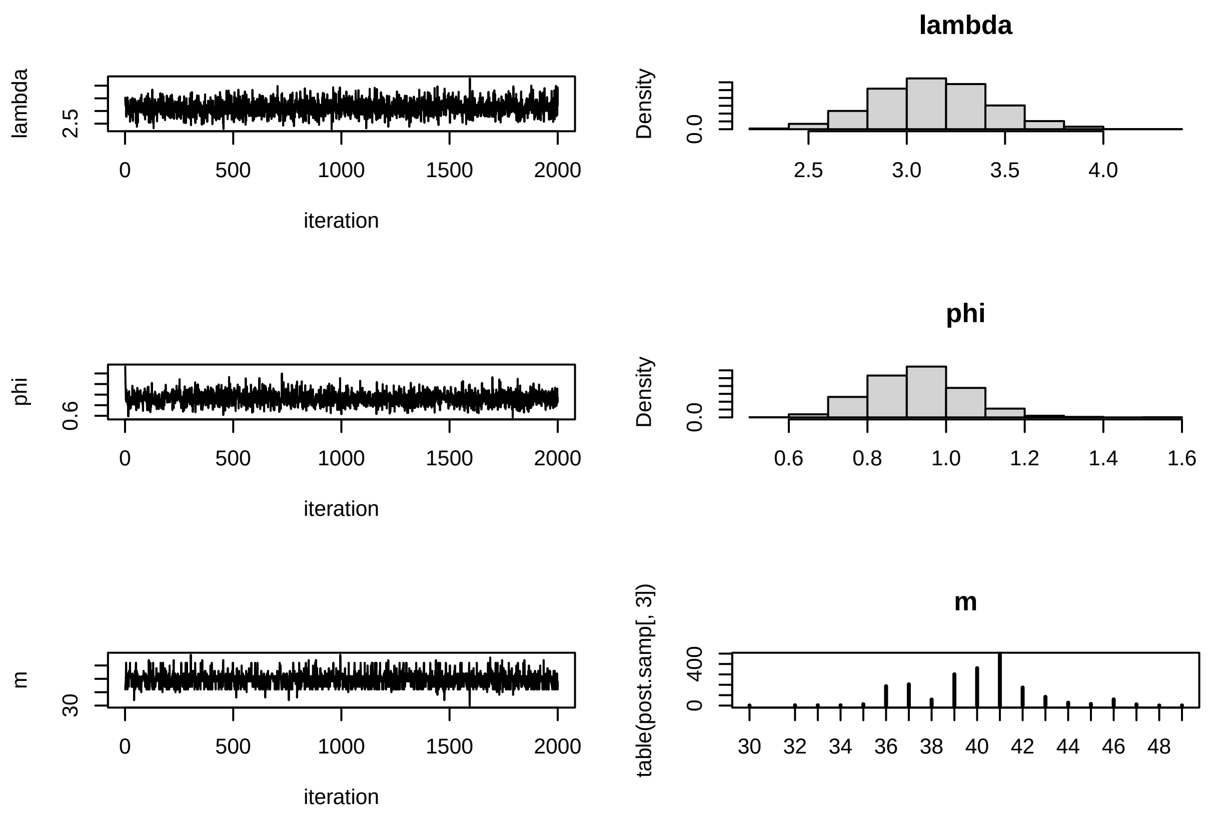


```
alpha = 0.001; beta = 0.001; delta = 0.001; gamma = 0.001; m = 10
set.seed(123456); M=2000; postsamp = NULL; par(mfrow=c(3,2))
fullcm1 = function(m,lambda,phi,x,n,alpha,beta,gamma,delta){
lambda^(alpha-1+sum(x[1:m]))*exp(-(beta+m)*lambda)*
phi^(gamma-1+sum(x)-sum(x[1:m]))*exp(-(delta+n-m)*phi)}
for (i in 1:M){
lambda = rgamma(1,sum(x[1:m])+alpha,m+beta)
phi = rgamma(1,sum(x)-sum(x[1:m])+gamma,n-m+delta)
fullcm = NULL
for (j in 1:n) {
fullcm=c(fullcm,fullcm1(j,lambda,phi,x,n,alpha,beta,gamma,delta))
}
fullcm=fullcm/sum(fullcm); m = sample(1:n,size=1,prob=fullcm)
postsamp = rbind(postsamp,c(lambda,phi,m))
}
post.samp <- data.frame(postsamp);names <- c("lambda","phi","m")
```

```
summary(post.samp)
```

```
##        X1               X2                X3
##  Min.   :2.280   Min.   :0.5699   Min.   :30.00
##  1st Qu.:2.932   1st Qu.:0.8451   1st Qu.:39.00
##  Median :3.132   Median :0.9255   Median :40.00
##  Mean   :3.134   Mean   :0.9268   Mean   :39.92
##  3rd Qu.:3.326   3rd Qu.:1.0026   3rd Qu.:41.00
##  Max.   :4.287   Max.   :1.5297   Max.   :49.00
```

```
for (i in 1:2){
ts.plot(post.samp[,i],xlab="iteration",ylab=names[i])
hist(post.samp[,i],xlab="",main=names[i],prob=T)}
ts.plot(post.samp[,3],xlab="iteration",ylab=names[3])
plot(table(post.samp[,3]),type="h",xlab="",main=names[3])
```



```
library(rjags)
```

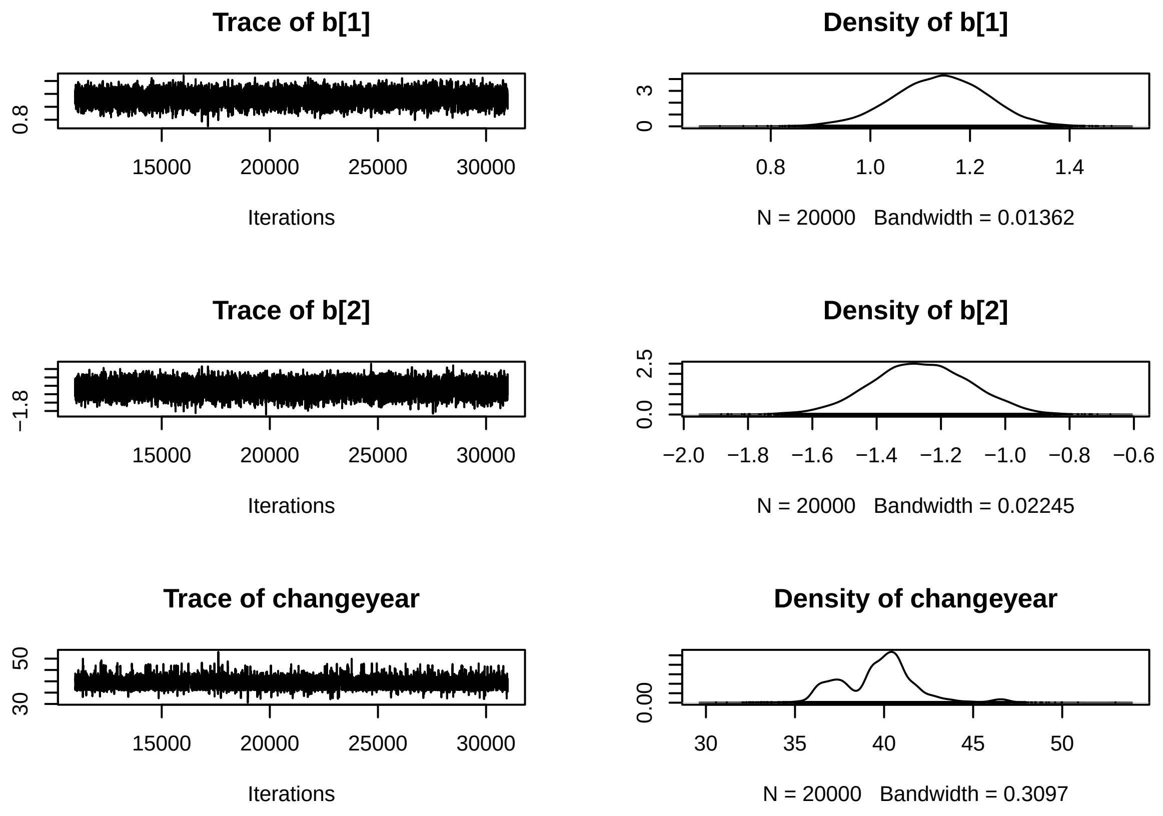
```
## Warning: 패키지 'rjags'는 R 버전 4.1.3에서 작성되었습니다

## 필요한 패키지를 로딩중입니다: coda

## Warning: 패키지 'coda'는 R 버전 4.1.3에서 작성되었습니다

## Linked to JAGS 4.3.0
## Loaded modules: basemod,bugs
```

```
jags_code = "model{
for(year in 1 : n){ D[year] ~ dpois(lambda[year])
log(lambda[year]) <- b[1]+step(year - changeyear)*b[2]}
for (j in 1:2) {b[j] ~ dnorm(0.0,1.0E-6)}
changeyear ~ dunif(1,n)
lambda1 <- exp(b[1])
phi1 <- exp(b[1]+b[2])}"
xdata=list(n=112, D=c(4,5,4,1,0,4,3,4,0,6,3,3,4,0,2,6,3,3,5,
4,5,3,1,4,4,1,5,5,3,4,2,5,2,2,3,4,2,1,3,2,1,1,1,1,1,3,0,0,1,0,1,1,
0,0,3,1,0,3,2,2,0,1,1,1,0,1,0,1,0,0,0,2,1,0,0,0,1,1,0,2,2,3,1,1,2,
1,1,1,1,2,4,2,0,0,0,1,4,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0))
init_value = function() {list(b = c(0, 0) , changeyear = 50)}
model<-jags.model(textConnection(jags_code),n.chains=1,data=xdata)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 112
##    Unobserved stochastic nodes: 3
##    Total graph size: 793
##
## Initializing model
```

```
update(model, 10000, progress.bar="none")
samp <- coda.samples(model, variable.names=c("b","changeyear"),
n.iter=20000, progress.bar="none")
summary(samp); plot(samp)
```

```
##
## Iterations = 11001:31000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean      SD  Naive SE Time-series SE
## b[1]          1.140 0.09309 0.0006583       0.001282
## b[2]         -1.262 0.15353 0.0010856       0.001976
## changeyear 39.717 2.19151 0.0154963       0.039720
##
## 2. Quantiles for each variable:
##
##                 2.5%    25%    50%    75%   97.5%
## b[1]          0.9503  1.078  1.142  1.204  1.3191
## b[2]       -1.5641 -1.366 -1.263 -1.157 -0.9673
## changeyear 36.0893 37.995 39.922 40.833 45.3282
```


Trace of b[1]
Iterations
Density of b[1]
N = 20000 Bandwidth = 0.01362
Trace of b[2]
Iterations
Density of b[2]
N = 20000 Bandwidth = 0.02245
Trace of changeyear
Iterations
Density of changeyear
N = 20000 Bandwidth = 0.3097