Lab4

Jun Rao

9/17/2019

MCMC sampling in R

Task 1 (25 points)

In the last lab, I showed that we can create an exact posterior estimate of the mean (θ) of a normally-distributed variate Y, if we have multiple data observations, and if we have a fixed (known) variance. Given that y is a vector of n data observations, then the posterior becomes:

$$\theta|y,\sigma^2 \sim Normal\left(\frac{\frac{\mu_0}{\tau_0^2}+\frac{\overline{y}n}{\sigma^2}}{\frac{1}{\tau_0^2}+\frac{n}{\sigma^2}},\frac{1}{\frac{1}{\tau_0^2}+\frac{n}{\sigma^2}}\right)$$

Here $\overline{y}$ is the average of our data observations, and again, n is the number of data observations.

Before, we calculated our posterior by creating a sequence of possible θ values and manually calculating the posterior of each value. As I discussed in lecture, this becomes exceedingly inefficient if we have many parameters in our model. Therefore, we often use MCMC methods to randomly sample from our unknown posterior distribution.

On the next page, I have provided code for estimating θ from a vector of normally distributed data, with fixed variance, using the MH-MCMC algorithm.

Complete the following:

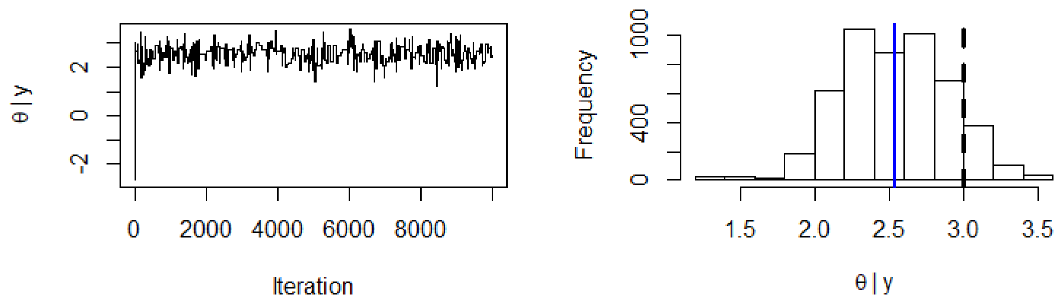
The MH-MCMC sampler for a model with a single parameter, assuming $Y \sim Normal(\theta,\sigma^2)$.

```r
#-----------------------------------
#-----------------------------------
#The mu and sd for the real distribution
mu_true = 3
sd_fixed = 2


#The mu and sd for the prior distribution
mu_prior = 5
sd_prior = 10

#The mu and sd for the proposal distribution
mu_prop = 5
sd_prop = sd_prior * 1.1

#-----------------------------------
#-----------------------------------
#Generate 30 observations from real distributation
n_sample = 30
y_test = rnorm(n_sample, mu_true, sd_fixed)

#-----------------------------------
#-----------------------------------

n_iter = 10000
chosen_theta = NULL

for(i in 1:n_iter){

  # Initializing the old_theta
  if(i == 1){
    old_theta = rnorm(1, mu_prop, sd_prop)
  }

  # for each loop, it will generate a new theta
  new_theta = rnorm(1, mu_prop, sd_prop)

  #calculate the probability based on the old theta and new theta which follo
w proposal distribution
  old_prop_adj = dnorm(old_theta, mu_prop, sd_prop, log = TRUE)
  new_prop_adj = dnorm(new_theta, mu_prop, sd_prop, log = TRUE)

  #calculate the probability based on the old theta and new theta which follo
w prior distribution
  old_prior = dnorm(old_theta, mu_prior, sd_prior, log = TRUE)
  new_prior = dnorm(new_theta, mu_prior, sd_prior, log = TRUE)

  #Calculate the likelihood based on the old theta and new theta
  old_lik = sum(dnorm(y_test, old_theta, sd_fixed, log = TRUE))
  new_lik = sum(dnorm(y_test, new_theta, sd_fixed, log = TRUE))

  # Calculate the posterior
  old_post = old_prior + old_lik
  new_post = new_prior + new_lik

  # calculate log_ratio and ratio
  log_ratio = (new_post - new_prop_adj) - (old_post - old_prop_adj)
  ratio = exp(log_ratio)

  # make a decision
  if(ratio > 1){
    chosen_theta[i] = new_theta
  }else{

    rand = runif(1, min = 0, max = 1)

    if(ratio >= rand){
      chosen_theta[i] = new_theta
    }else{
      chosen_theta[i] = old_theta
    }

  }

  #assign the new value to the old_theta
  old_theta = chosen_theta[i]

}
```

Here are some example plots of the output of a single chain:

```r
par(mfrow = c(1, 2))

# Plot the full trace:
plot(chosen_theta ~ c(1:n_iter), type = "l",
     xlab = "Iteration", ylab = expression(theta~"|"~y))

# Discard the burn-in
# Plot the sample of the estimated posterior of \theta
n_burn = n_iter / 2
hist(chosen_theta[n_burn:n_iter], main = "",
     xlab = expression(theta~"|"~y), ylab = "Frequency")

# Add the mean estimate
mu_est = mean(chosen_theta[n_burn:n_iter])
abline(v=mu_est, col = "blue", lwd = 2)
```

```
# Add the true mean
abline(v = mu_true, col = "black", lwd = 3, lty = 2)
```



```
par(mfrow = c(1,1))
```

Run multiple MCMC chains, remove the burn-in (n/2) from each chain, and store the resulting output in an R object (e.g., a data frame). Then, create a function to calculate the Gelman-Ruben statistic, based on the equations in class. Use this function to determine if your chains have converged (15 points).

```
library(matrixStats)
n_iter = 10000

 #-----------------------------------
#-----------------------------------
#Generate 30 observations from real distributation
n_sample = 30
y_test = rnorm(n_sample, mu_true, sd_fixed)

theta <- function(){
    #-----------------------------------
    #-----------------------------------
    #The mu and sd for the real distribution
    mu_true = 3
    sd_fixed = 2

    #The mu and sd for the prior distribution
    mu_prior = 5
    sd_prior = 10

    #The mu and sd for the proposal distribution
    mu_prop = 5
    sd_prop = sd_prior * 1.1

    #-----------------------------------
```

```r
#-----------------------------------


chosen_theta = NULL
result = NULL

for(i in 1:n_iter){

  # Initializing the old_theta
  if(i == 1){
    old_theta = rnorm(1, mu_prop, sd_prop)
  }

  # for each loop, it will generate a new theta
  new_theta = rnorm(1, mu_prop, sd_prop)

  #calculate the probability based on the old theta and new theta which f
ollow proposal distribution
  old_prop_adj = dnorm(old_theta, mu_prop, sd_prop, log = TRUE)
  new_prop_adj = dnorm(new_theta, mu_prop, sd_prop, log = TRUE)

  #calculate the probability based on the old theta and new theta which f
ollow prior distribution
  old_prior = dnorm(old_theta, mu_prior, sd_prior, log = TRUE)
  new_prior = dnorm(new_theta, mu_prior, sd_prior, log = TRUE)

  #Calculate the likelihood based on the old theta and new theta
  old_lik = sum(dnorm(y_test, old_theta, sd_fixed, log = TRUE))
  new_lik = sum(dnorm(y_test, new_theta, sd_fixed, log = TRUE))

  # Calculate the posterior
  old_post = old_prior + old_lik
  new_post = new_prior + new_lik

  # calculate log_ratio and ratio
  log_ratio = (new_post - new_prop_adj) - (old_post - old_prop_adj)
  ratio = exp(log_ratio)

  # make a decision
  if(ratio > 1){
      chosen_theta[i] = new_theta
  }else{

      rand = runif(1, min = 0, max = 1)

      if(ratio >= rand){
          chosen_theta[i] = new_theta
      }else{
          chosen_theta[i] = old_theta
          }

      }

  #assign the new value to the old_theta
  old_theta = chosen_theta[i]
    }

  #remove the burn-in (n/2) from each chain
  result = tail(chosen_theta,length(chosen_theta)/2)

  return(result)
}


#Create an Matrix as a objeCt
object <- matrix(0, 15, ncol = (n_iter/2))
for(row in 1:15){
  result <- theta()
  object[row, ] <- result
}

str(object)

##  num [1:15, 1:5000] 3.54 3.34 3.13 2.32 3.25 ...

#create a function to calculate the Gelman-Ruben statistic
GR <- function(M){

  #mean value for each row in the matrix
  Chain.mean = rowMeans(M)
  #Variance for each row in the matrix
  Chain.Var = rowVars(M)

  #Calculte W in our notes
  W = sum(Chain.Var)/nrow(M)
  #Calculate Beta
  sum = 0
  for (i in 1:nrow(M)) {
    sum = sum +(Chain.mean[i] - mean(M))**2
  }
  Beta = (ncol(M) * sum)/(nrow(M)-1)

  temp = ((ncol(M)-1)*W)/ncol(M) + Beta/ncol(M)

  result = sqrt(temp/W)

  return(result)
}
```

```
GR(object)

## [1] 1.004094
```

Since the value at here is less than 1.1, hence we can conclude that our chains are converged.

Validate that the MCMC sampler correctly approximates the theoretical (i.e., analytically-derived) mean and variance of the posterior distribution of the unknown parameter, θ. Look at the equation above to help (3 points).

```
#the mean of MCMC sampler and the mean variance of the MCMC sampler
mean(rowMeans(object))

## [1] 3.246702

mean(rowVars(object))

## [1] 0.1339936

#Now, we will calculate the mean and variance based on the equation
mu = 5
tau = 10
sigma = 2
n=30
y = mean(y_test)
#Based on the equation, I am calculating the mu
mu.equation = ((mu/tau**2) + (y*n/sigma**2))/(1/tau**2 + n/sigma**2)
mu.equation

## [1] 3.247648

#Based on the equation I am calculate the variance
sigma.equation = (1/((1/tau**2) + (n/sigma**2)))**2
sigma.equation

## [1] 0.01773047
```

We can find at here：

the mean in the MCMC sample is 2.684517, and the variance is 0.1357709

the mean from the equation is 2.694757, and the variance is 0.1587276

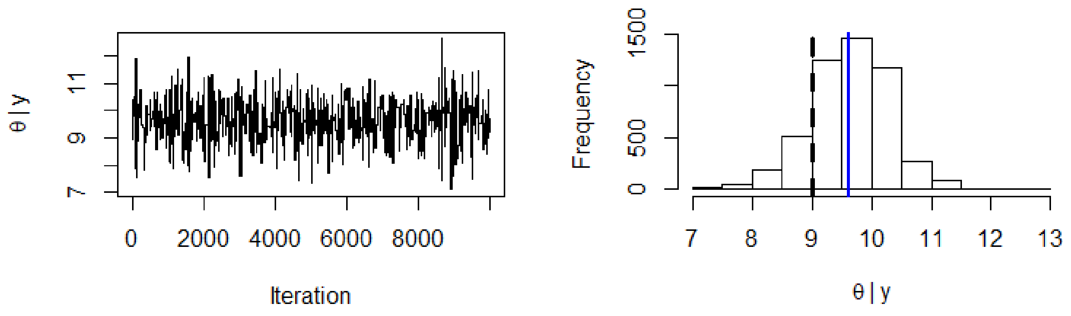
4.Conduct some synthetic experiments with your MCMC sampler. Specifically, use your sampler with different prior and proposal distributions. How does the prior and proposal affect *the MCMC sampler's efficiency and how quickly it reaches a "good" probability space? Do the same with differently sized data sets. I do not want to see the output of these simulations. Just use the MCMC on your own as a "toy" model. Then, using these anecdotal simulations as support, briefly comment on how the priors, proposals, and data set size influence the MCMC sampler (5 points).*

The MH-MCMC sampler for a model with a single parameter, assuming $Y \sim Normal(\theta, \sigma^2)$.

```r
#-----------------------------------
#-----------------------------------
#The mu and sd for the real distribution
mu_true = 9
sd_fixed = 4


#The mu and sd for the prior distribution
mu_prior = 5
sd_prior = 10

#The mu and sd for the proposal distribution
mu_prop = 5
sd_prop = sd_prior * 1.1

#-----------------------------------
#-----------------------------------
#Generate 30 observations from real distributation
n_sample = 30
y_test = rnorm(n_sample, mu_true, sd_fixed)

#-----------------------------------
#-----------------------------------

n_iter = 10000
chosen_theta = NULL

for(i in 1:n_iter){

  # Initializing the old_theta
  if(i == 1){
    old_theta = rnorm(1, mu_prop, sd_prop)
  }

  # for each loop, it will generate a new theta
  new_theta = rnorm(1, mu_prop, sd_prop)
  #calculate the probability based on the old theta and new theta which follo
w proposal distribution
  old_prop_adj = dnorm(old_theta, mu_prop, sd_prop, log = TRUE)
  new_prop_adj = dnorm(new_theta, mu_prop, sd_prop, log = TRUE)

  #calculate the probability based on the old theta and new theta which follo
w prior distribution
  old_prior = dnorm(old_theta, mu_prior, sd_prior, log = TRUE)
  new_prior = dnorm(new_theta, mu_prior, sd_prior, log = TRUE)

  #Calculate the likelihood based on the old theta and new theta
  old_lik = sum(dnorm(y_test, old_theta, sd_fixed, log = TRUE))
  new_lik = sum(dnorm(y_test, new_theta, sd_fixed, log = TRUE))

  # Calculate the posterior
  old_post = old_prior + old_lik
  new_post = new_prior + new_lik

  # calculate log_ratio and ratio
  log_ratio = (new_post - new_prop_adj) - (old_post - old_prop_adj)
  ratio = exp(log_ratio)

  # make a decision
  if(ratio > 1){
    chosen_theta[i] = new_theta
  }else{

    rand = runif(1, min = 0, max = 1)

    if(ratio >= rand){
      chosen_theta[i] = new_theta
    }else{
      chosen_theta[i] = old_theta
    }

  }

  #assign the new value to the old_theta
  old_theta = chosen_theta[i]

}
```

Here are some example plots of the output of a single chain:

```r
par(mfrow = c(1, 2))

# Plot the full trace:
plot(chosen_theta ~ c(1:n_iter), type = "l",
     xlab = "Iteration", ylab = expression(theta~"|"~y))

# Discard the burn-in
# Plot the sample of the estimated posterior of \theta
n_burn = n_iter / 2
hist(chosen_theta[n_burn:n_iter], main = "",
     xlab = expression(theta~"|"~y), ylab = "Frequency")

# Add the mean estimate
mu_est = mean(chosen_theta[n_burn:n_iter])
abline(v=mu_est, col = "blue", lwd = 2)

# Add the true mean
abline(v = mu_true, col = "black", lwd = 3, lty = 2)
```



```
par(mfrow = c(1,1))
```

Both prior and proposal can affect the MCMC sample's efficiency. Little change of the mean will not change the sample distribution too much. However, the low standard deviation of prior can make the MCMC sampler trace plot varies frequently. It means the estimate mean will be far away the true mean.

Task 2 (25 points)

Complete the following:

1. Extend your MCMC sampler to accommodate 2 parameters. Here we will estimate both the mean and the standard deviation of the data (15 points). A hint for the proposal distributions: you can sum the proposal densities of the two parameters together on the log scale to get the total proposal adjustment. Also, be sure to store your output correctly. Your code should store the marginal posteriors of both parameters of interest.

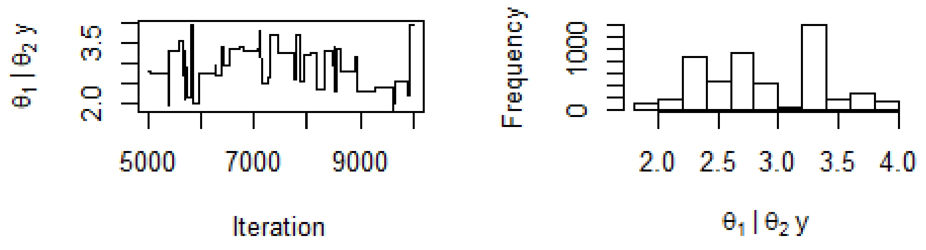
```
#-----------------------------------
#-----------------------------------

mu_true = 3 # Parameter 1 (unknown)
sd_true = 2 # Parameter 2 (unknown)

##### FOR PARAMETER 1
# Prior for Parameter 1
# Normal Distribution:
mu_prior = 5
sd_prior = 10

# Proposal for Parameter 1
# Normal Distribution
mu_prop = 5
sd_prop = sd_prior * 1.1

##### FOR PARAMETER 2
# Prior for Parameter 2
# Gamma distribution:
shape_prior = 1
rate_prior = 0.5

# Proposal for Parameter 2
# Gamma distribution
shape_prop = 1
rate_prop = rate_prior*0.5

#-----------------------------------
#-----------------------------------
n_sample = 30
y_test = rnorm(n_sample, mu_true, sd_true)
#-----------------------------------
#-----------------------------------



n_iter = 10000
```

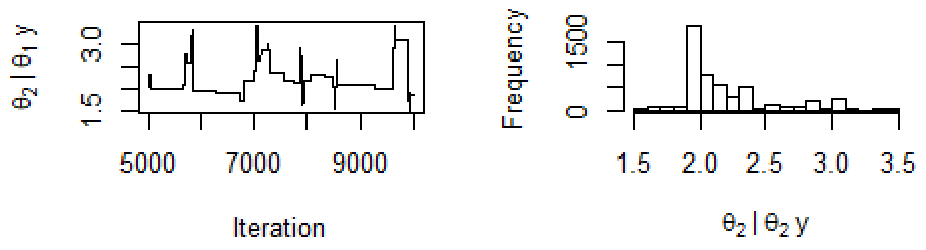
```
H <- function(n_iter=n_iter){

    chosen_mu = NULL
    chosen_sd = NULL

    for(i in 1:n_iter){

        if(i==1){
          old_mu = rnorm(1, mu_prop, sd_prop)
          old_sd = rgamma(1, shape_prop, rate_prop)
        }

        new_mu = rnorm(1, mu_prop, sd_prop)
        new_sd = rgamma(1, shape_prop, rate_prop)

        old_mu_prop_adj = dnorm(old_mu, mu_prop, sd_prop, log = TRUE)
        new_mu_prop_adj = dnorm(new_mu, mu_prop, sd_prop, log = TRUE)

        old_sd_prop_adj = dgamma(old_sd, shape_prop, rate_prop, log = TRUE)
        new_sd_prop_adj = dgamma(new_sd, shape_prop, rate_prop, log = TRUE)

        old_total_prop_adj = old_mu_prop_adj + old_sd_prop_adj
        new_total_prop_adj = new_mu_prop_adj + new_sd_prop_adj
        ##################################################################
        old_mu_prior = dnorm(old_mu, mu_prior, sd_prior, log = TRUE)
        new_mu_prior = dnorm(new_mu, mu_prior, sd_prior, log = TRUE)
        old_sd_prior = dgamma(old_sd, shape_prior, rate_prior, log = TRUE)
        new_sd_prior = dgamma(new_sd, shape_prior, rate_prior, log = TRUE)


        old_lik = sum(dnorm(y_test, old_mu, old_sd, log = TRUE))
        new_lik = sum(dnorm(y_test, new_mu, new_sd, log = TRUE))


        old_mu_post = old_mu_prior + old_sd_prior + old_lik
        new_mu_post = new_mu_prior + old_sd_prior + new_lik

        old_sd_post = old_mu_prior + old_sd_prior + old_lik
        new_sd_post = old_mu_prior + new_sd_prior + new_lik

        mu_log_ratio = (new_mu_post - new_total_prop_adj) - (old_mu_post - ol
d_total_prop_adj)
        sd_log_ratio = (new_sd_post - new_total_prop_adj) - (old_sd_post - ol
d_total_prop_adj)

        mu_ratio = exp(mu_log_ratio)
        sd_ratio = exp(sd_log_ratio)
        if(mu_ratio > 1){
            chosen_mu [i] = new_mu
        }
        else{
            rand = runif(1,min=0,max=1)
          if(mu_ratio >= rand){
              chosen_mu[i] = new_mu
          }else{
              chosen_mu[i] = old_mu
          }
        }

        if(sd_ratio > 1){
            chosen_sd [i] = new_sd
        }
        else{
            rand = runif(1,min=0,max=1)
          if(sd_ratio >= rand){
              chosen_sd[i] = new_sd
          }else{
              chosen_sd[i] = old_sd
          }
        }


        old_mu =  chosen_mu[i]

        old_sd =  chosen_sd[i]
    }
    # Discard your burn-in
    mu_theta = tail(chosen_mu,length(chosen_mu)/2)
    sd_theta = tail(chosen_sd,length(chosen_sd)/2)

    temp <- matrix(0, nrow=2, ncol = length(mu_theta))
    temp[1,] <- mu_theta
    temp[2,] <- sd_theta

    return(temp)
}
```

2.Run multiple MCMC chains and store the output appropriately in an object. This can be done in a data frame object if you are careful, but ’s is also helpful. You would have p × n × m dimensions in the array, where m is the number of chains, n is the number of iterations in each chain, and p is the number of parameters.

3.Discard your burn-in, plot the traces of each parameter, and use your function to calculate the $\hat{R}$ for each parameter (5 points).

```
num.chains=15

Multiple_MCM <- function(){
    object <- array(0,dim = c(2,15,5000))
    for(i in 1:15){
        temp <- H(10000)

        object[,i,] <- temp
    }
  return(object)
}


obj <- Multiple_MCM()

par(mfrow=c(2,2))


plot(obj[1,1,] ~ c(5001:10000),
     type="l",
     xlab = "Iteration",
     ylab =expression(theta[1]~"|"~theta[2]~y)
     )

hist(obj[1,1,], main = "",
     xlab =expression(theta[1]~"|"~theta[2]~y),
     ylab = "Frequency")


plot(obj[2,1,] ~ c(5001:10000),
     type="l",
     xlab = "Iteration",
     ylab =expression(theta[2]~"|"~theta[1]~y)
     )

hist(obj[2,1,], main = "",
     xlab =expression(theta[2]~"|"~theta[2]~y),
     ylab = "Frequency")
```





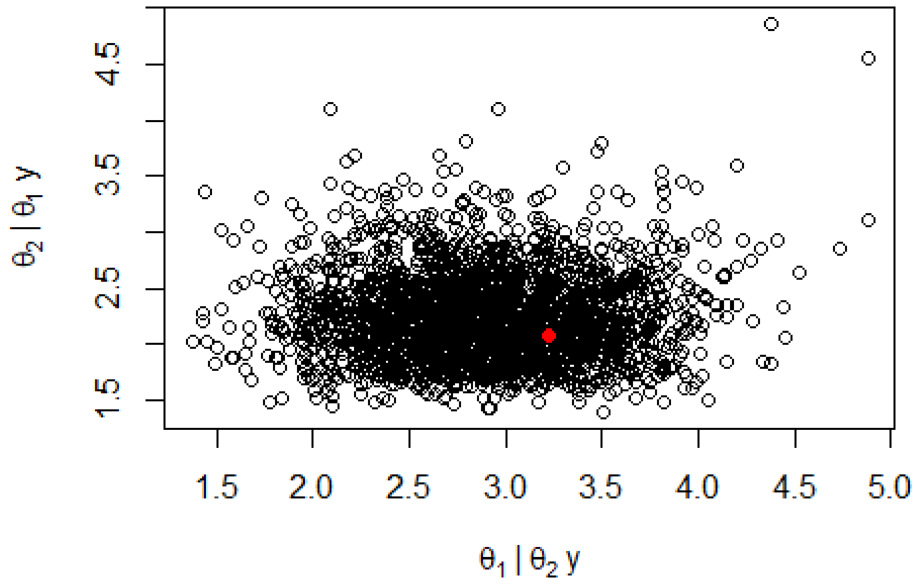
Now, we will compute $\hat{R}$

```
R.hat.mu <- GR(obj[1,,])
R.hat.sd <- GR(obj[2,,])
R.hat.mu

## [1] 1.024216

R.hat.sd

## [1] 1.041624
```

4.Calculate the mean, marginal posterior estimates for each parameter. Plot the joint posterior samples of parameter 1 and parameter 2 and place a large "point" at the location of these mean estimates. Hint: You can use the function. (5 points)

```
result <- H(500000)
mu_estimate <- mean(result[1])
sd_estimate <- mean(result[2])
mu_estimate

## [1] 3.225676

sd_estimate

## [1] 2.068743
```

```r
plot(x=result[1,],
     y = result[2,],
     xlab = expression(theta[1]~"|"~theta[2]~y),
     ylab = expression(theta[2]~"|"~theta[1]~y),
     )

points(x=mu_estimate, y=sd_estimate, col='red', pch=19)
```



1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
$\theta_1 | \theta_2$ y
$\theta_2 | \theta_1$ y