Assignment 2

Po-Hsiang Peng

This file illustrates how to use Metropolis-Hastings MCMC algorithm to draw random samples and estimate parameters. We will use the following model to give a concrete example:

$$y = Ax + b + \epsilon$$

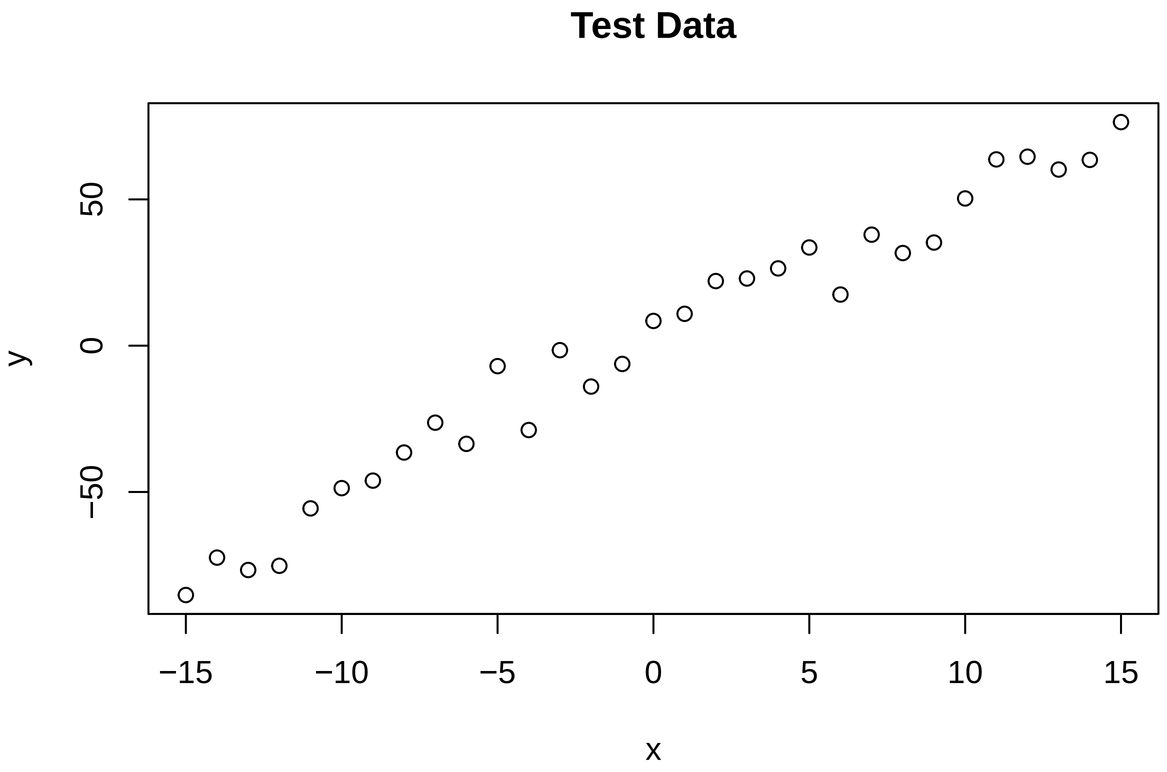
where $\epsilon \sim N(0, \mathrm{SD}^2)$, and the prior distributions of A, B, SD are

(1) $A \sim \mathrm{Unif}(0, 10)$
(2) $B \sim N(0, 25)$
(3) $\mathrm{SD} \sim \mathrm{Unif}(0, 30)$

First of all, we need to generate data. The true values of A, B and SD are setting to be 5, 0 and 10, respectively. Moreover, we generate total 31 samples. We first create independent variables x, and generate dependent data y as mentioned above.

```r
trueA <- 5
trueB <- 0
trueSd <- 10
sampleSize <- 31
x <- (-(sampleSize-1)/2):((sampleSize-1)/2)
y <-  trueA * x + trueB + rnorm(n=sampleSize,mean=0,sd=trueSd)
```

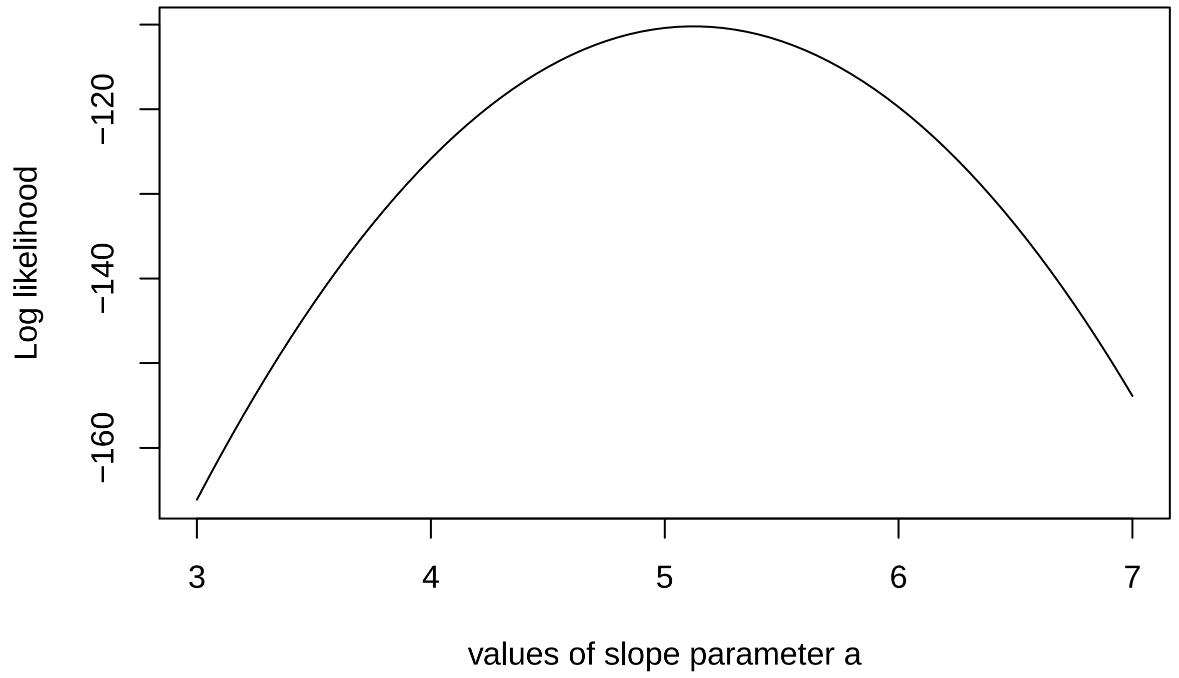
From the following graph, we can see that there is a linear relationship.



Since we would like to draw random samples from posterior distribution, we must specify likelihood function and prior function first.

```r
# Compute its log-likelihood. Since y = Ax+B+N(0,SD^2), we have
# y ~ N(Ax+b,SD^2). Thus, we can use
#   sum( dnorm(y, mean=a*x+b, sd=sd, log=T) )
# to compute its log-likelihood.
likelihood <- function(param){
  a = param[1]
  b = param[2]
  sd = param[3]

  pred = a*x + b
  singlelikelihoods = dnorm(y, mean = pred, sd = sd, log = T)
  sumll = sum(singlelikelihoods)
  return(sumll)
}
# Here we compute the log-pdf of prior distribution. Note that we assume
# A, B and SD are independent so that we can sum up these three log-pdfs.
prior <- function(param){
  a = param[1]
  b = param[2]
  sd = param[3]
  aprior = dunif(a, min=0, max=10, log = T)
  bprior = dnorm(b, sd = 5, log = T)
  sdprior = dunif(sd, min=0, max=30, log = T)
  return(aprior+bprior+sdprior)
}
```

Here we compute the log-likelihood for different a given B and SD.



We can see that the likelihood would be maximized when a is around 5, which is in agreement with our setting.

Then we compute its log-pdf of posterior distribution. Note that

$$f(\theta|x) \propto f(x|\theta)f(\theta)$$

Thus, we can sum the log-likelihood and log-pdf of prior distribution.

```r
posterior <- function(param){
  return (likelihood(param) + prior(param))
}
```

To start MH algorithm, we must decide proposal distribution. Here we choose it as

$$\theta^* \sim N(\theta, \Sigma)$$

where $\Sigma = \text{diag}(0.1, 0.5, 0.3)$

```r
proposalfunction <- function(param){
  return(rnorm(3,mean = param, sd= c(0.1,0.5,0.3)))
}
```

Now, we are ready to implement MH algorithm to draw random sample from posterior distribution.

```r
run_metropolis_MCMC <- function(startvalue, iterations){

  # We first construct an matrix to store the following chains.
  # The size of this matrix is (N+1)*3, where N is the number of
  # iterations and 3 is the parameters we interested. Note that the
  # first row of this matrix is the initial value.
  chain = array(dim = c(iterations+1,3))
  chain[1,] = startvalue

  for (i in 1:iterations){

    # Then, we will draw a sample. We will accept this sample if U < A
    # where U ~ Unif(0,1) and A = f(theta_new|x)/f(theta_current|x)
    # That is, if the new sample is more likely than the current one, then
    # we will accept is. Otherwise, we will accept it at appropriate chance.
    proposal = proposalfunction(chain[i,])

    if(prior(proposal) == -Inf) {
      chain[i+1,] = chain[i,]
      next
    }
    probab = exp(posterior(proposal) - posterior(chain[i,]))
    if (runif(1) < probab){
      chain[i+1,] = proposal
    }else{
      chain[i+1,] = chain[i,]
    }
  }
  return(chain)
}
```

We will use 4, 0 and 10 as initial value. Since we choose it at random, we must specify a *burn in* number. That is, we discard first few samples. Here, we generate 10,000 samples and discard first 5,000 of them.

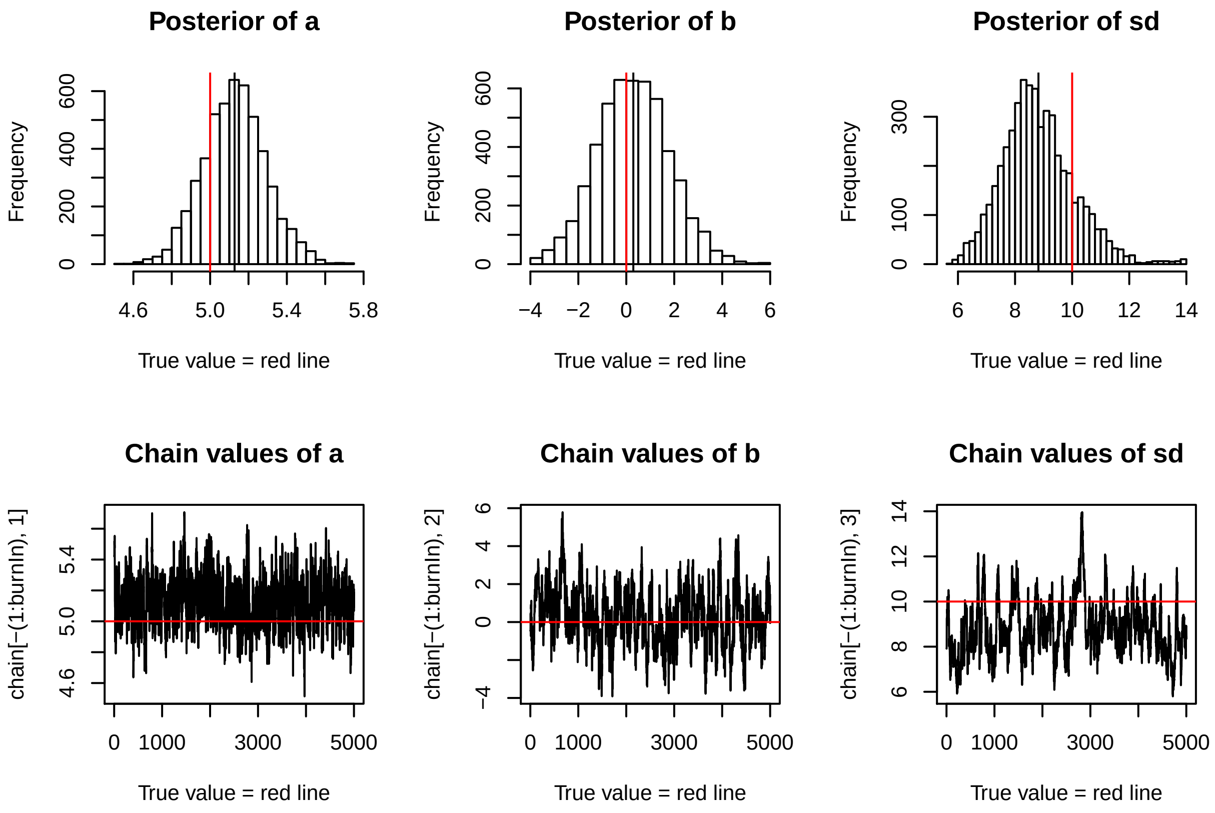
```r
startvalue = c(4,0,10)
burnIn = 5000
chain = run_metropolis_MCMC(startvalue, 10000)
```

It is important to see the acceptance rate, which is affected by the standard deviation of proposal distribution. If acceptance is too low, then we should choose a smaller standard deviation.

```
( acceptance = 1-mean(duplicated(chain[-(1:burnIn),])) )
```

```
## [1] 0.7488502
```

The best way to examine the result is plotting the result.



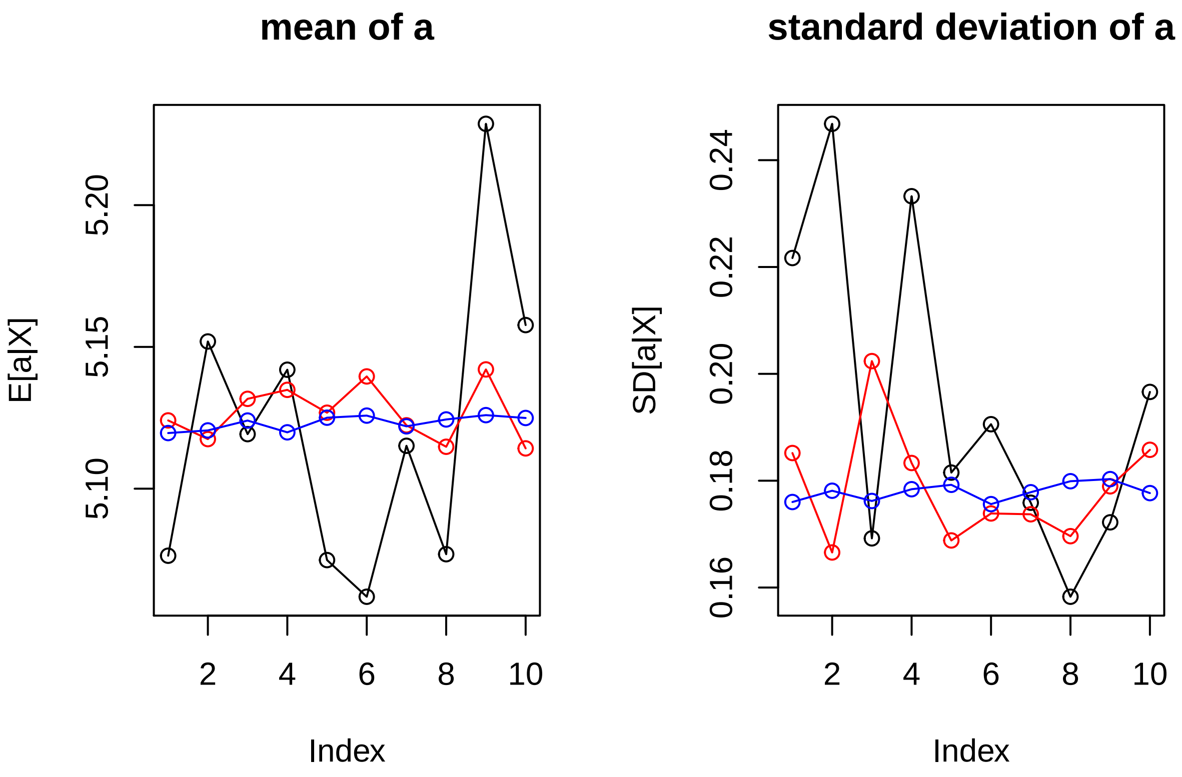
Now, we examine the effect of different numbers of iteration. We will try different iteration number (1000, 10000 and 100000) 10 times and compare their result. The initial value will be drawn from prior distribution.

```
compare_outcomes = function(iteration, burnIn,ntimes=10) {

  a = array(dim=c(ntimes,2))

  for(i in 1:ntimes) {

    # random initial value generated from its prior distribution
    initial = c(runif(1,0,10), rnorm(1,0,5), runif(1,0,30))
    chain = run_metropolis_MCMC(initial, iteration)

    m = apply(chain[-(1:burnIn),],2,mean)
    s = apply(chain[-(1:burnIn),],2,sd)

    a[i,1] = m[1]
    a[i,2] = s[1]
  }
  return(a)
}
```

For iteration number 1000, burin in number is set to be 500. For the other two cases, burn in number equal to 5000.

```
out1 = compare_outcomes(1000,500)
out2 = compare_outcomes(10000,5000)
out3 = compare_outcomes(100000,5000)
```

Here, we focus on the slope parameter, a. From the following graph, the black/red/blue line is the result for 1000/10000/100000 iteration number. We can see that the result becomes stable when iteration number increases.


mean of a
E[a|X]
5.10
5.15
5.20
2
4
6
8
10
Index
standard deviation of a
SD[a|X]
0.16
0.18
0.20
0.22
0.24
2
4
6
8
10
Index