Accuracy of the Metropolis-Hasting Algorithm

Wenjing Xu

October 13, 2018

This report is about the accuracy of the Metropolis-Hasting algorithm in finding a value of a, for different numbers of iterations.

1st Step: Background Setting

Here we assume a linear relationship between the predictor and the response variable.

Specifically, let's assume $Y = 5X + 0 + \epsilon$, where $\epsilon \sim N(0, 10^2)$

The test data used to fit our model is created as follow.

```
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)
```

2nd Step: Calculate mean and std of a in each resulted value of chain

The code for computing mean and std of a values in chain is as follows.

```
source("C:\\Users\\xwj93\\Documents\\GitHub\\assignment-2-wendy182\\all the functions.R")

compare_outcomes <- function(iterations){
  mean_sd_of_a = array(dim = c(10, 2))
  for (j in 1:10){
    startvalue = c()
    startvalue[1] = runif(1, min=0, max=10)
    startvalue[2] = rnorm(1, sd=5)
    startvalue[3] = runif(1, min=0, max=30)
    chain = array(dim = c(iterations+1, 3))
    chain[1, ] = startvalue
    for (i in 1:iterations){
      proposal = proposalfunction(chain[i, ])
      probab = exp(posterior(proposal) - posterior(chain[i, ]))
      if (runif(1) < probab){
        chain[i+1, ] = proposal
      } else{
        chain[i+1, ] = chain[i, ]
      }
    }
    mean_sd_of_a[j,] = c(mean(chain[, 1]), sd(chain[, 1]))
  }
  return (mean_sd_of_a)
}
```

3rd Step: Compare accuray of estimation of a under different numbers of iterations.

In this step, I directly call the compare_outcomes functions while giving different values of iterations. the reported mean and std of a under each scenario can help to investigate accuracy of this algorithm in estimating a.

```r
compare_outcomes(1000)
```

```
##           [,1]      [,2]
##  [1,] 4.654221 0.8671769
##  [2,] 4.660848 1.0112336
##  [3,] 4.962539 0.2561939
##  [4,] 5.089398 0.2415668
##  [5,] 4.900342 0.2544426
##  [6,] 5.029611 0.2362614
##  [7,] 5.100517 0.3409598
##  [8,] 5.192075 0.4541929
##  [9,] 4.954917 0.2402566
## [10,] 5.171181 0.5931200
```

```r
compare_outcomes(10000)
```

```
##           [,1]      [,2]
##  [1,] 4.998862 0.3523094
##  [2,] 4.964927 0.3526816
##  [3,] 4.983697 0.2633121
##  [4,] 4.994435 0.2341429
##  [5,] 5.002988 0.2117073
##  [6,] 5.013253 0.2237269
##  [7,] 4.991642 0.2416436
##  [8,] 4.968286 0.3618164
##  [9,] 5.017021 0.2428404
## [10,] 5.011793 0.2294754
```

```r
compare_outcomes(100000)
```

```
##           [,1]      [,2]
##  [1,] 5.007691 0.2327633
##  [2,] 5.011391 0.2294345
##  [3,] 5.004318 0.2265049
##  [4,] 5.009911 0.2214607
##  [5,] 5.015706 0.2256078
##  [6,] 5.004574 0.2290443
##  [7,] 5.009030 0.2272541
##  [8,] 5.000613 0.2206178
##  [9,] 5.011928 0.2411105
## [10,] 5.005859 0.2258241
```

The results above shows the mean and std of a in each of 10 loop under different values of iterations (1000,10000,100000). The first column is mean value of a in chain, and the second is the std of a in chain.

Comapring these three matrix, we can see that as iteration times increase, the value of a becomes more stable which is indicated by a slower std, and its value stabilises close to the true value of a (i.e. 5). In other words, when iterate for significantly large times, this algorithm accuracy increases with the increase of iteration times.