Chapter 6: Metropolis Hastings Algorithms

Exercise 6.1

a R program for this chain:

```
x=1:10^4
x[1] = rnorm(1)
r=0.9
for (i in 2:10^4) {
    x[i] = r*x[i-1] + rnorm(1) }
hist( x , freq = F , col = "wheat2" , main = "" )
curve( dnorm ( x , sd = 1/sqrt(1-r^2)) , add = T , col = "tomato"
```

Exercise 6.3

when $q(y|x) = g(y)$,

we can get that $\rho(x,y) = \min\left(\frac{f(y)}{f(x)} \frac{q(x|y)}{q(y|x)}, 1\right)$

$= \min\left(\frac{f(y)}{f(x)} \frac{g(x)}{g(y)}, 1\right)$

$\Rightarrow$ since acceptance probability $\frac{f(y)}{f(x)} \frac{g(x)}{g(y)} \geq \frac{f(y)/g(y)}{\max f(x)/g(x)}$

$\Rightarrow \rho(x,y) \geq \min\left(\frac{f(y)/g(y)}{\max f(x)/g(x)}, 1\right)$

it's larger than accept-reject .

Exercise 6.7

a. Generate Metropolis Hastings samples from the density:

```
Nsim = 10^4
a=2.7; b=6.3
X=runif (Nsim)
last=X[1]
for (i in 1: Nsim) {
    cand = rbeta (1,1,1)
    alpha = (dbeta ( cand , a , b ) / dbeta ( last , a , b ))/
  ( dbeta (cand , 1 , 1 ) / dbeta (last , 1 , 1))
    if (runif (1) < alpha)
```

```
        last = cand
      X[i] = last
}
hist(X , pro=TRUE , col="wheat2" , xlab="" , ylab="" , main = "Beta(2.7,3) simulation")
curve (dbeta(x,a,b) ,add=T , lwd=2 , col="sienna2")
```

The acceptance rate is estimated by

```
> length (unique(X)) / 5000
[1] 0.458
```

If instead we use a $\mathscr{B}e(20, 60)$ proposal, the modified lines in the R program are:

```
cand = rbeta(20 , 60 , 1)
alpha = (dbeta (cand , a , b) / dbeta(last , a , b))/
      (dbeta (cand , 20 , 60) / dbeta (last , 20 , 60))
```

and the acceptance rate drops to zero.

b. In the case of a truncated beta, the R program below shows very similar running times but more efficiency for the beta proposal:

```
Nsim = 5000
a=2.7 ; b=6.3 ; c=0.25 ; d=0.75
X=rep (runif(1) , Nsim)
test2 = function(){
      last=X[1]
      for (i in 1 : Nsim){
            cand = rbeta (1,2,6)
            alpha = (dbeta (cand,a,b) / dbeta (last,a,b)) /
         (dbeta (cand,2,6) / dbeta(last,2,6))
            if ((runif(1)<alpha) && (cand<d) && (c<cand))
                  last = cand
            X[i] = last}
      }
test1 = function(){
   last = X[1]
   for (i in 1 : Nsim){
         cand = runif(1,c,d)
         alpha = (dbeta(cand,a,b) / dbeta(last,a,b))
         if ((runif(1)<alpha) && (cand<d) && (c<cand))
            last = cand
         X[i] = last
         }
}
system.time(test1()) ; system.time(test2())
```

Exercise 6.9

a. The Accept Reject algorithm with a Gamma $\mathscr{G}(4,7)$ candidate can be implemented as follows:

```
g47 = rgamma(5000 , 4 , 7)
u = runif (5000 , max = dgamma (g47 , 4 , 7))
x = g47 [u < dgamma(g47 , 4.3 , 6.2)]
par (mfrow = c(1,3) , mar=c(4,4,1,1))
hist (x,freq=FALSE , xlab="" , ylab="" , col="wheat2",
main = "Accept-Reject with Ga(4.7) proposal")
curve (dgamma(x,4.3,6.2) , lwd=2 , col="sienna" , add=T)
```

The efficiency of the simulation is given by:

```
> length(x) / 5000
[1] 0.8374
```

b. The Metropolis-Hastings algorithm with a Gamma $\mathscr{G}(4,7)$ candidate can be implemented as follows

```
X = rep (0 , 5000)
X[1] = rgamma (1 , 4.3 , 6.2)
for (t in 2:5000){
    rho=(dgamma (X[t-1] , 4 , 7) * dgamma (g47[t] , 4.3 , 6.2))/
        (dgamma (g47[t] , 4 , 7) * dgamma (X[t-1] , 4.3 , 6.2))
    X[t] = X[t-1] + (g47[t]-X[t-1]) * (runif(1) < rho)
    }
hist (X , freq = FALSE , xlab="" , ylab="" , col="wheat2",
main = "Metropolis-Hastings with Ga(4,7) proposal")
curve(dgamma(x,4.3,6.2) , lwd=2 , col="sienna" , add=T)
```

Its efficiency is

```
> length(unique(X))/5000
[1] 0.79
```

c. The Metropolis-Hastings algorithm with a Gamma $\mathscr{G}(5,6)$ candidate can be implemented as follows:

```
g56 = rgamma (5000 , 5 , 6)
X[1] = rgamma (1 , 4.3 , 6.2)
for (t in 2:5000){
    rho = (dgamma(X[t-1] , 5 , 6) * dgamma(g56[t] , 4.3 , 6.2))/
        (dgamma(g56[t] , 5 , 6) * dgamma(X[t-1] , 4.3 , 6.2))
    X[t] = X[t-1] + (g56[t]-X[t-1]) * (runif(1) < rho)
    }
hist (X , freq=FALSE , xlab="" , ylab="" , col="wheat2",
main = "Metropolis-Hastings with Ga(5,6) proposal")
```

```
curve (dgamma (x , 4.3 , 6.2) , lwd=2 , col = "sienna" , add = T)
```

Its efficiency is

```
> length (unique(X)) / 5000
[1] 0.7678
```

which is therefore quite similar to the previous proposal.