HW3

Haoluan Chen

11/19/2021

Name: Haoluan Chen
Student number: 1003994261
Department: Staitstial Science Program: Master of Statistics
Year: 1
Email: haoluan.chen@mail.utoronto.ca

```
Ydye = t( matrix(
    c(1545, 1440, 1440, 1520, 1580,
         1540, 1555, 1490, 1560, 1495,
         1595, 1550, 1605, 1510, 1560,
         1445, 1440, 1595, 1465, 1545,
         1595, 1630, 1515, 1635, 1625,
         1520, 1455, 1450, 1480, 1445), nrow=5) )
```

a)

```
A = 4
B = 2
C = 6
D = 1

a1 = 0.001/(5+A)
b1 = 0.001/(5+B)
a2 = 0.001/(5+C)
b2 = 0.001/(5+D)
a3 = b3 = 1600

helper1 <- function (theta, mu){
  result <- 0
  for (i in theta){
    iter = (i - mu)^2
    result = result + iter
  }
  result
}




g <- function(V, W, mu, theta){
```

```r
  A = 4
  B = 2
  C = 6
  D = 1

  a1 = 0.001/(5+A)
  b1 = 0.001/(5+B)
  a2 = 0.001/(5+C)
  b2 = 0.001/(5+D)
  a3 = b3 = 1600
  data = t( matrix(
    c(1545, 1440, 1440, 1520, 1580,
         1540, 1555, 1490, 1560, 1495,
         1595, 1550, 1605, 1510, 1560,
         1445, 1440, 1595, 1465, 1545,
         1595, 1630, 1515, 1635, 1625,
         1520, 1455, 1450, 1480, 1445), nrow=5) )

  -b1/V + (-a1-1)*log(V) + (-b2/W) + (-a2-1)*log(W) -
    ((mu-a3)^2/(2*b3)) - (6/2)*log(V) - 0.5*15*log(W) -
    (1/(2*V))*helper1(theta, mu) - sum((data-theta)^2)/(2*W)
}

h <- function(V, W){
  V/W
}

M = 110000  # run length
B = 10000  # amount of burn-in
V <- 5000
W <- 5000
mu <- 200
theta=rnorm(6,mu,sqrt(V))


sigma = 8  # proposal scaling, too low will lead to NAs

Vlist = rep(0,M)  # for keeping track of chain values
Wlist = rep(0,M)
mulist = rep(0,M)
theta1list = rep(0,M)
theta2list = rep(0,M)
theta3list = rep(0,M)
theta4list = rep(0,M)
theta5list = rep(0,M)
theta6list = rep(0,M)

hlist = rep(0,M)  # for keeping track of h function values
numaccept = 0;


for (i in 1:M) {
    V1 = V + sigma * rnorm(1)  # proposal value
```

```r
    W1 = W + sigma * rnorm(1)  # proposal value
    mu1 = mu + sigma * rnorm(1)  # proposal value
    theta1 = theta + sigma * rnorm(6)  # proposal value

    U = log(runif(1))  # for accept/reject
    alpha = g(V1, W1, mu1, theta1) - g(V, W, mu, theta)  # for accept/reject
   if (U < alpha) {
          V = V1  # accept proposal
          W = W1
          mu = mu1
          theta = theta1
        numaccept = numaccept + 1;
    }
    Vlist[i] = V
    Wlist[i] = W
    mulist[i] = mu
    theta1list[i] = theta[1]
    theta2list[i] = theta[2]
    theta3list[i] = theta[3]
    theta4list[i] = theta[4]
    theta5list[i] = theta[5]
    theta6list[i] = theta[6]
    hlist[i] = h(V, W);
}

cat("ran Metropolis algorithm for", M, "iterations, with burn-in", B, "\n");
```

```
## ran Metropolis algorithm for 110000 iterations, with burn-in 10000
```

```r
cat("acceptance rate =", numaccept/M, "\n");
```

```
## acceptance rate = 0.7104545
```

```r
u = mean(hlist[(B+1):M])
cat("mean of h is about", u, "\n")
```

```
## mean of h is about 0.4537426
```

```r
se1 =  sd(hlist[(B+1):M]) / sqrt(M-B)
cat("iid standard error would be about", se1, "\n")
```

```
## iid standard error would be about 0.0004380068
```

```r
varfact <- function(xxx) { 2 * sum(acf(xxx, plot=FALSE, lag.max = 2000)$acf) - 1 }
thevarfact = varfact(hlist[(B+1):M])
se = se1 * sqrt( thevarfact )
cat("varfact = ", thevarfact, "\n")
```

```
## varfact =  3685.873
```

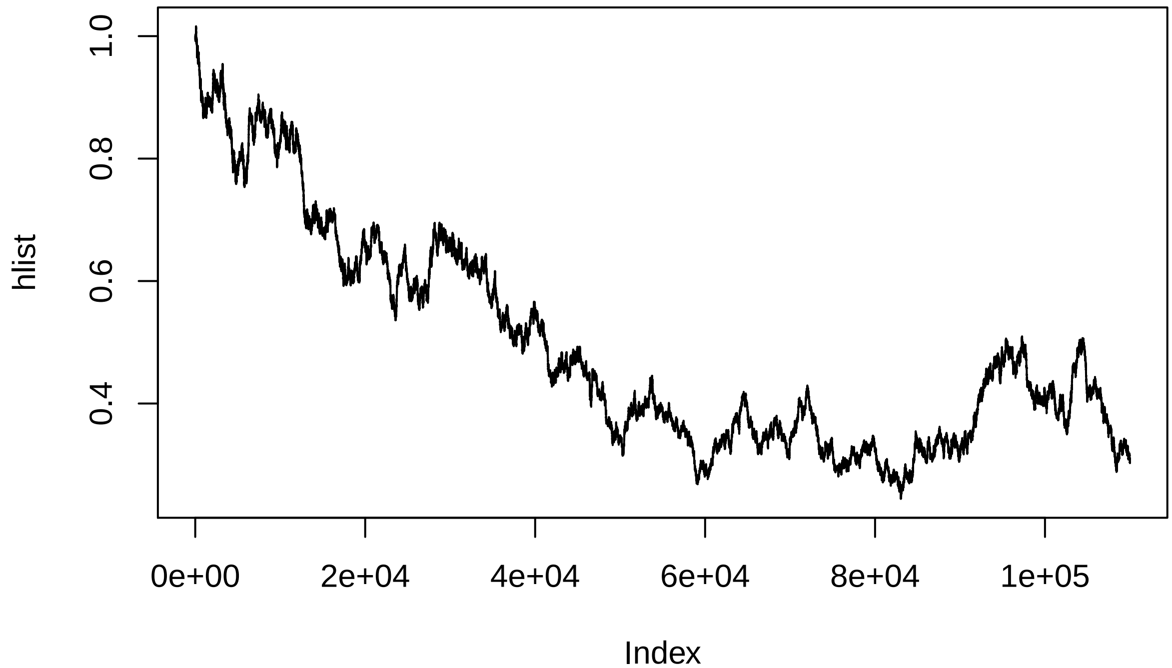
```r
cat("true standard error is about", se, "\n")
```

```
## true standard error is about 0.02659201
```

```r
cat("approximate 95% confidence interval is (", u - 1.96 * se, ",",
                                               u + 1.96 * se, ")\n\n")
```

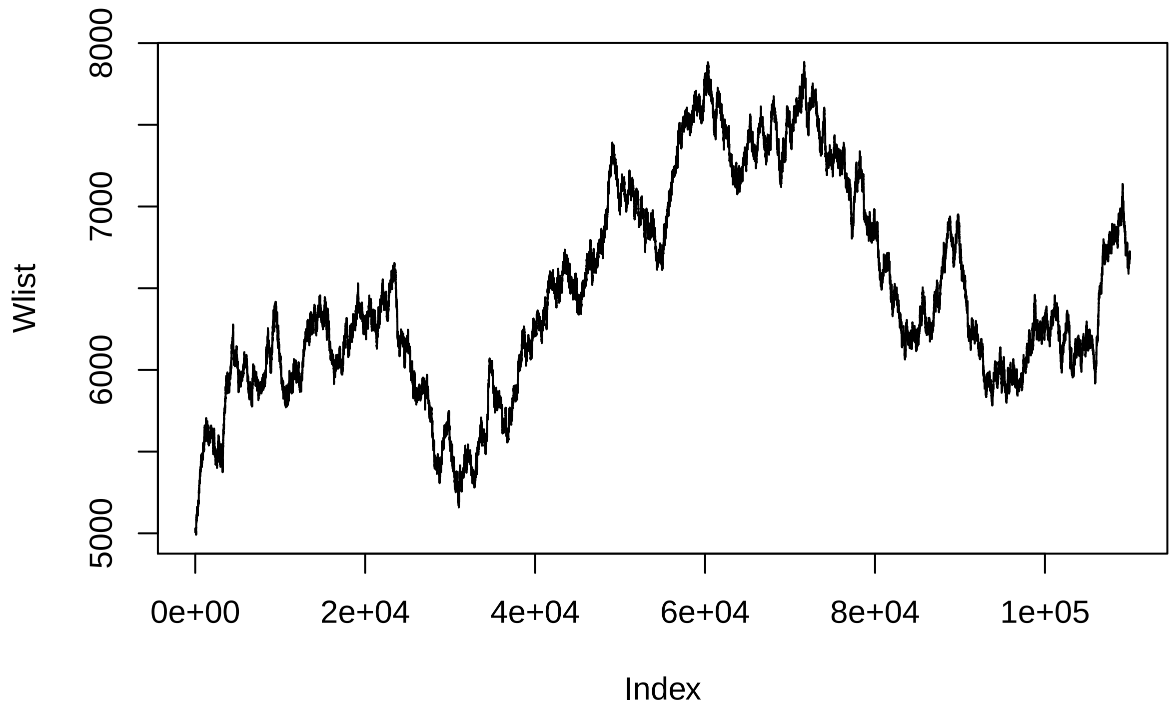
```
## approximate 95% confidence interval is ( 0.4016222 , 0.5058629 )
```

```r
#
# plot(x1list, type='l')
# plot(x2list, type='l')
# plot(x1list, x2list, type='p')
plot(hlist, type = "l")
```




```r
plot(Vlist, type = "l")
```


```
plot(Wlist, type = "l")
```



Using the full-dimensional metropolis algorithm, the result is not good. Based on the plot of the h value, I do not have a good mix. The acceptance rate is too high around 70%, but if I increase my proposal scaling would produce NA value for log(V) and log(W) in the density function. Additionally, my varfact are very large, this means the variance of the estimate are large. Therefore, the full-dimensional metropolis algorithm cannot accurately estimate the mean of V/W

b) Component wise

```
M = 110000  # run length
B = 2000  # amount of burn-in
V <- 5000
W <- 5000
mu <- 200

theta=rnorm(6,mu,sqrt(V))
X <- c(V, W, mu, theta)
sigma = 5  # proposal scaling

Vlist = rep(0,9*M)  # for keeping track of chain values
Wlist = rep(0,9*M)
mulist = rep(0,9*M)
theta1list = rep(0,9*M)
theta2list = rep(0,9*M)
theta3list = rep(0,9*M)
theta4list = rep(0,9*M)
```

```
theta5list = rep(0,9*M)
theta6list = rep(0,9*M)
hlist = rep(0,9*M)  # for keeping track of h function values
numaccept = 0;


for (i in 1:M) {
  for (coord in 1:9){
    Y = X
    Y[coord] = X[coord]+ sigma * rnorm(1)  # proposal value
    U = log(runif(1))  # for accept/reject
    thetaY = c(Y[4],Y[5], Y[6], Y[7], Y[8], Y[9])
    thetaX = c(X[4], X[5], X[6], X[7], X[8], X[9])
    alpha = g(Y[1], Y[2], Y[3], thetaY)- g(X[1],X[2],X[3], thetaX)  # for accept/reject
   if (U < alpha) {
        X = Y  # accept proposal
      numaccept = numaccept + 1;
      }

    Vlist[9*i - 9+coord] = X[1];
    Wlist[9*i - 9+coord] = X[2];
    mulist[9*i - 9+coord] = X[3];
    theta1list[9*i - 9+coord] = X[4];
    theta2list[9*i - 9+coord] = X[5];
    theta3list[9*i - 9+coord] = X[6];
    theta4list[9*i - 9+coord] = X[7];
    theta5list[9*i - 9+coord] = X[8];
    theta6list[9*i - 9+coord] = X[9];
    hlist[9*i - 9+coord] = h(X[1],X[2]);

  }
}

cat("ran Metropolis algorithm for", M, "iterations, with burn-in", B, "\n");
```

```
## ran Metropolis algorithm for 110000 iterations, with burn-in 2000
```

```
cat("acceptance rate =", numaccept/(9*M), "\n");
```

```
## acceptance rate = 0.9561697
```

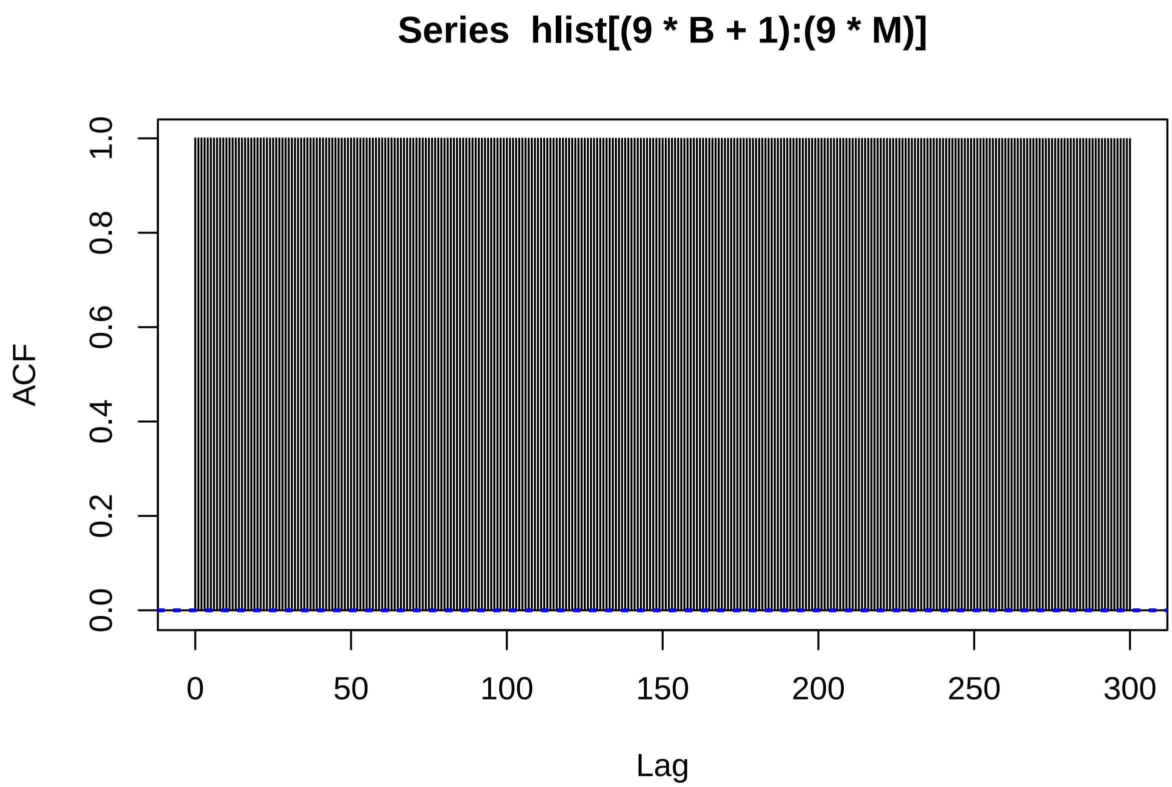
```
u = mean(hlist[(9*B+1):(9*M)])
cat("mean of h is about", u, "\n")
```

```
## mean of h is about 0.641157
```

```
se1 =  sd(hlist[(9*B+1):(9*M)]) / sqrt(9*(M-B))
cat("iid standard error would be about", se1, "\n")
```

```
## iid standard error would be about 0.0001195521
```

```
acf(hlist[(9*B+1):(9*M)], lag.max = 300)
```




```
varfact <- function(xxx) { 2* sum(acf(xxx, plot=FALSE, lag.max = 300)$acf) - 1 }
thevarfact = varfact(hlist[(2*B+1):(9*M)])
se = se1 * sqrt( thevarfact )
cat("varfact = ", thevarfact, "\n")
```

```
## varfact =  600.51
```

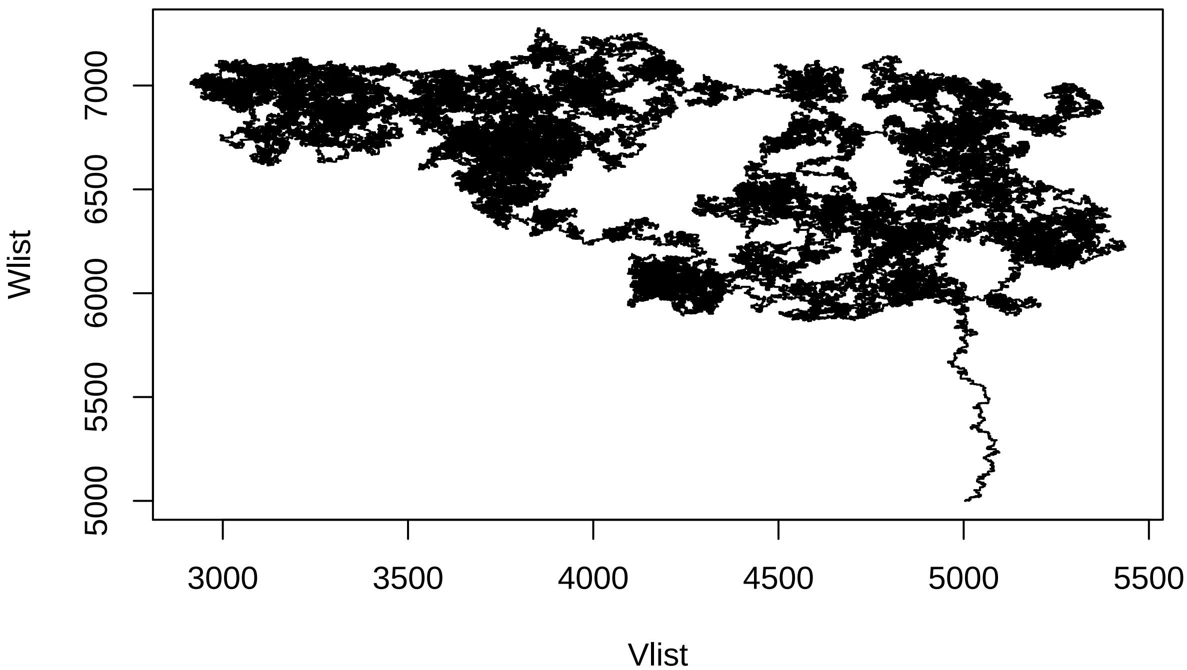
```
cat("true standard error is about", se, "\n")
```

```
## true standard error is about 0.00292966
```

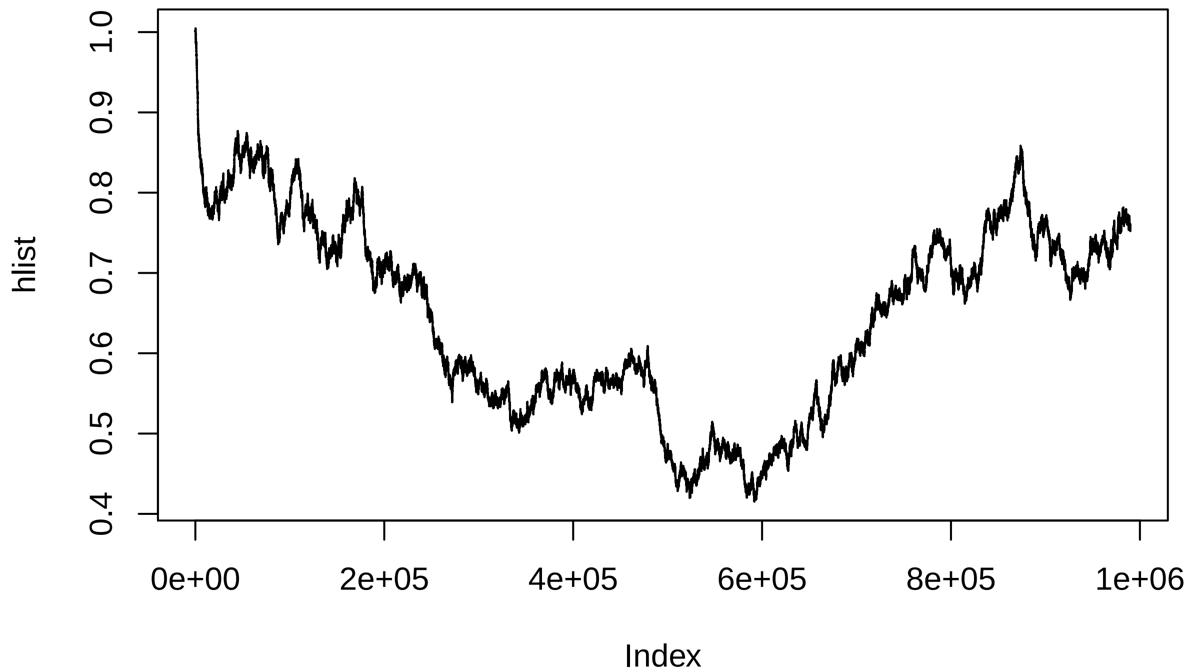
```
cat("approximate 95% confidence interval is (", u - 1.96 * se, ",",
                                              u + 1.96 * se, ")\n\n")
```

```
## approximate 95% confidence interval is ( 0.6354149 , 0.6468992 )
```

```
plot(Vlist, Wlist, type='l')
```




```
plot(hlist, type = "l")
```



Using the componentwise metropolis dos not produce good result. This have same problem as using full-dimensional Metropolis that if I increase my proposal scaling would produce NA value for log(V) and log(W) in the density function. (I don't know if I did something wrong in my code). Also the plot for h(x) does not mix well and it has high varfact value. Therefore, the componentwise metropolis algorithm cannot accurately estimate the mean of V/W.

c) Gibbs sampler

```
M = 110000  # run length
B = 2000  # amount of burn-in
V <- 3000
W <- 2000
mu <- 1000
J=rep(1:5)

theta <- rnorm(6, mu,sqrt(V))

Vlist = rep(0,M)  # for keeping track of chain values
Wlist = rep(0,M)
mulist = rep(0,M)
theta1list = rep(0,M)
theta2list = rep(0,M)
theta3list = rep(0,M)
theta4list = rep(0,M)
theta5list = rep(0,M)
```

```r
theta6list = rep(0,M)
hlist = rep(0,M)  # for keeping track of h function values
numaccept = 0

for (i in 1:M) {
    V1 = 1/rgamma(1,a1+3, b1+1/2*sum((theta-mu)^2)) # proposal value
    W1 = 1/rgamma(1,a2+1/2*15,(b2+sum((Ydye-theta)^2/2)))
    mu1 = rnorm(1,(a3*V1+b3*sum(theta))/(V1+6*b3),sqrt((b3*V1)/(V1+6*b3)))
    for (j in 1:6){
      Y_j=Ydye[j,]
      theta1[j]=rnorm(1,(mu*W+V*sum(Y_j))/(W+V*j),sqrt(V*W/(W+V*j)))
    }

    V = V1  # accept proposal
      W = W1
      mu = mu1
      theta = theta1
    numaccept = numaccept + 1;

    Vlist[i] = V
    Wlist[i] = W
    mulist[i] = mu
    theta1list[i] = theta[1]
    theta2list[i] = theta[2]
    theta3list[i] = theta[3]
    theta4list[i] = theta[4]
    theta5list[i] = theta[5]
    theta6list[i] = theta[6]
    hlist[i] = h(V, W);
}

cat("ran Metropolis algorithm for", M, "iterations, with burn-in", B, "\n");
```

```
## ran Metropolis algorithm for 110000 iterations, with burn-in 2000
```

```r
cat("acceptance rate =", numaccept/M, "\n");
```

```
## acceptance rate = 1
```

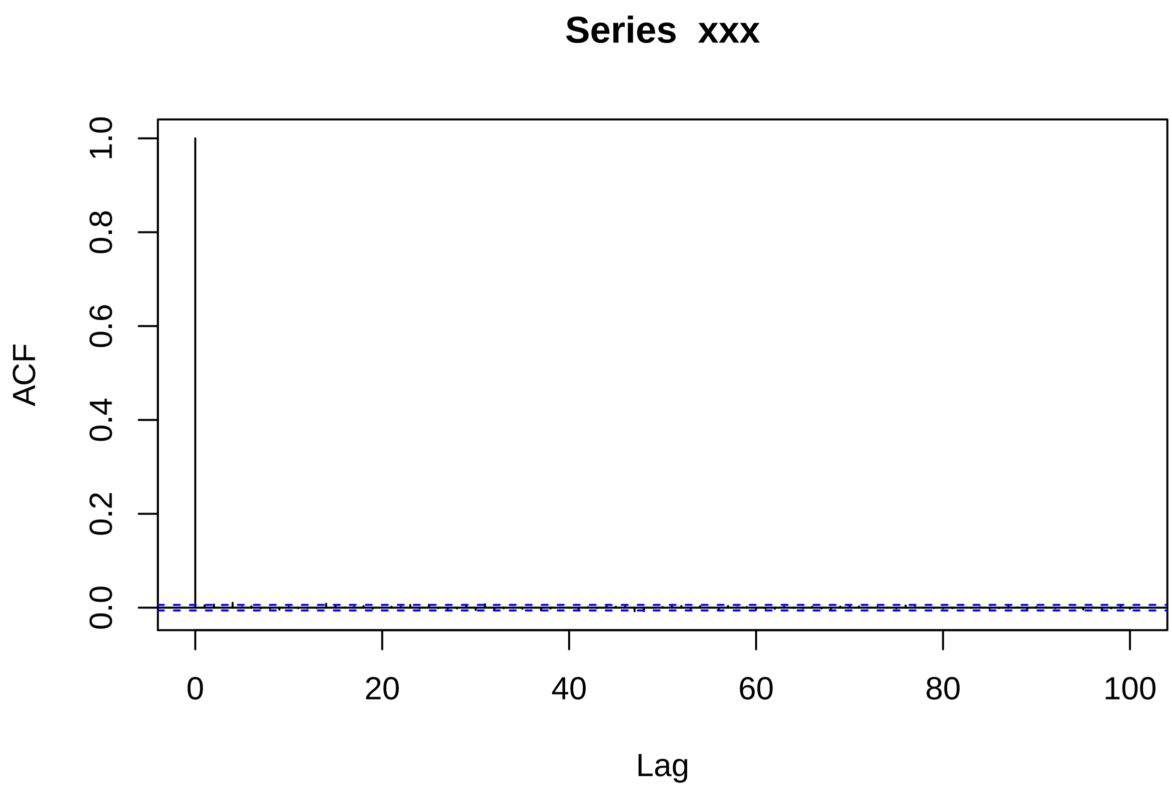
```r
u = mean(hlist[(B+1):M])
cat("mean of h is about", u, "\n")
```

```
## mean of h is about 0.7020742
```

```r
se1 =  sd(hlist[(B+1):M]) / sqrt(M-B)
cat("iid standard error would be about", se1, "\n")
```

```
## iid standard error would be about 0.002319867
```

```r
varfact <- function(xxx) { 2 * sum(acf(xxx, lag.max = 100)$acf) - 1 }
thevarfact = varfact(hlist[(B+1):M])
```




```
se = se1 * sqrt( thevarfact )
cat("varfact = ", thevarfact, "\n")
```

```
## varfact =  1.059422
```

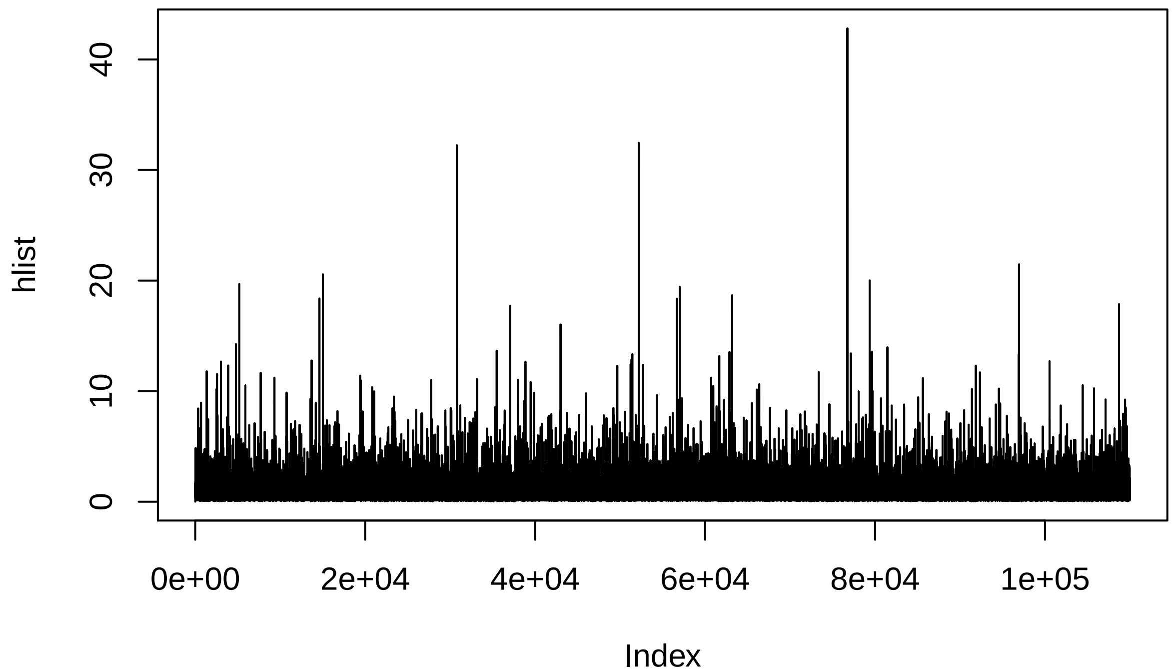
```
cat("true standard error is about", se, "\n")
```

```
## true standard error is about 0.002387797
```

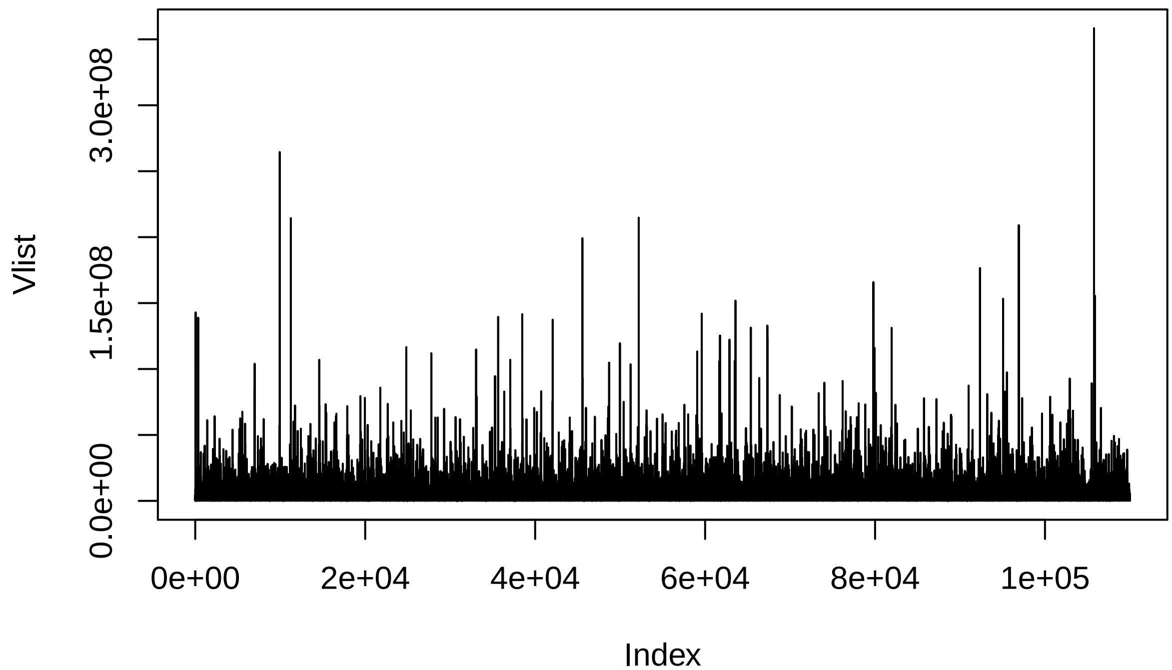
```
cat("approximate 95% confidence interval is (", u - 1.96 * se, ",",
                                                u + 1.96 * se, ")\n\n")
```

```
## approximate 95% confidence interval is ( 0.6973942 , 0.7067543 )
```

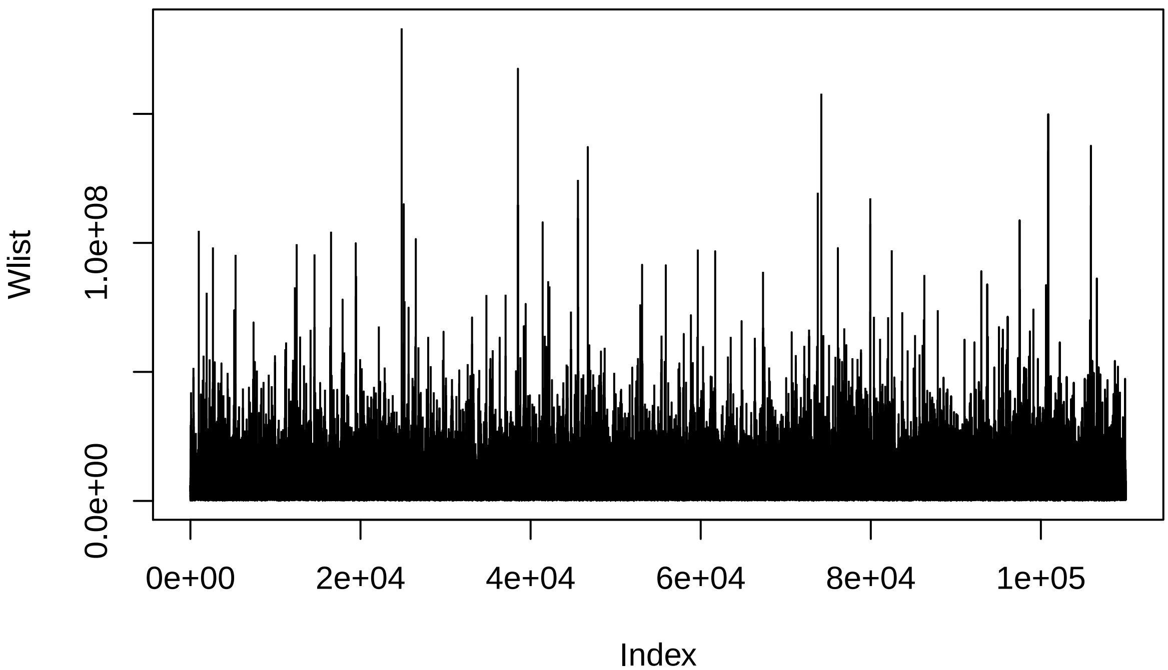
```
#
# plot(x1list, type='l')
# plot(x2list, type='l')
# plot(x1list, x2list, type='p')
plot(hlist, type = "l")
```




```
plot(Vlist, type = "l")
```




```
plot(Wlist, type = "l")
```



The Gibbs sampler is best at estimating the mean of V/W compared to full-dimensional metropolis and component-wise metropolis. The Gibbs sampler estimated the mean of V/W is around 0.7. The plot of the h(x), V and W are mixed very well. Additionally, Gibbs sampler is very much stable compared to full-dimensional metropolis and component-wise metropolis.