stad80A1

Zi Jian Liu

1/25/2021

Contents

1.

(1)

Wrong statements: C. E.
(2)
Wrong statements: A. B. C.
(3)
Wrong statements: C. ## (4)
Wrong statements: A.
(5)
Correct statements: B. D.
(6)
Wrong statements: c. E.

2.

2.a)

$$l(\theta) = \sum_{i=1}^{n}(log(\theta-1)-\theta log(X_i)) = nlog(\theta-1)-\theta\sum_{i=1}^{n} logX_i \Rightarrow \frac{\partial}{\partial\theta} = \frac{n}{\theta-1}-\sum_{i=1}^{n} logX_i = 0n = \sum logX_i(\theta-1) \Rightarrow \hat{\theta}_n = \frac{n}{\sum logX}$$

2.b)

$$I(\theta) = -\int_1^\infty (\frac{\partial^2}{\partial\theta^2} log((\theta-1)x^{-\theta}))(\theta-1)x^{-\theta}dx = -\int_1^\infty (\frac{\partial}{\partial\theta}\frac{1}{(\theta-1)} - logx)(\theta-1)x^{-\theta}dx = \int_1^\infty (\frac{1}{(\theta-1)})x^{-\theta}dx = \int_1^\infty \frac{x^{-\theta}}{\theta-1}dx$$

$$(\hat{\theta_n} - \theta) \xrightarrow{d} N(0, \frac{1}{nI(\theta)}) \Rightarrow \lim_{n\to\infty} \frac{1}{nI(\theta)} = \lim_{n\to\infty} \frac{1}{n}(\theta-1)^2 = 0$$

2.c)

$$C_n = [\hat{\theta_n} - \frac{z_{\alpha/2}}{\sqrt{nI(\hat{\theta_n})}}, \hat{\theta_n} + \frac{z_{\alpha/2}}{\sqrt{nI(\hat{\theta_n})}}] = [\hat{\theta_n} - \frac{z_{0.025}}{\sqrt{n(\frac{1}{(\theta-1)^2})}}, \hat{\theta_n} + \frac{z_{0.025}}{\sqrt{n(\frac{1}{(\theta-1)^2})}}] = [\frac{1}{logX_i} + 1 - \frac{1.96}{n(\frac{1}{(\theta-1)^2})}, \frac{1}{logX_i} + 1 + \frac{1.96}{n(\frac{1}{(\theta-1)^2})}]$$

2.d)

$$F(x) = \int_1^\infty (\theta-1)x^{-\theta}dx = (-1+y)(\frac{x^{-y+1}}{-y+1} - \frac{1}{-y+1}) = (-1+2)(\frac{x^{-2+1}}{-2+1} - \frac{1}{-2+1}) = (\frac{x^{-1}}{-1} + 1) = 1 - \frac{1}{x} \Rightarrow F^{-1}(x) = -\frac{1}{x-1} \quad (1)$$

```
set.seed(1003643587)
N <- 10000
n <- 100
t <- 2
invCDF <- function(u, t) {
  return(-1/(u-1))
}
```

confidence interval

```
lowerBound <- c()
upperBound <- c()
for (i in 1:N) {
  uni <- runif(100, 0, 1)
  samp <- c()
  for (j in 1:100) {
    samp <- append(samp, invCDF(uni[j], t), after = length(samp))
  }
  upperBound <- append(upperBound, 1 + 1/mean(log(samp)) + 1.96/(sqrt(n*mean(log(samp))^2)), after = len
  lowerBound <- append(lowerBound, 1 + 1/mean(log(samp)) - 1.96/(sqrt(n*mean(log(samp))^2)), after = len
}
```

check for theta in confidence intervals

```
lowerBound = replace(lowerBound, lowerBound <= t, 1)
lowerBound = replace(lowerBound, lowerBound > t, 0)
upperBound = replace(upperBound, upperBound < t, 0)
upperBound = replace(upperBound, upperBound >= t, 1)

k <- 0
for (l in 1:N) {
  if(upperBound[l] == 1 & lowerBound[l] == 1) {
    k <- k + 1
  }
}
print(k/N)
```

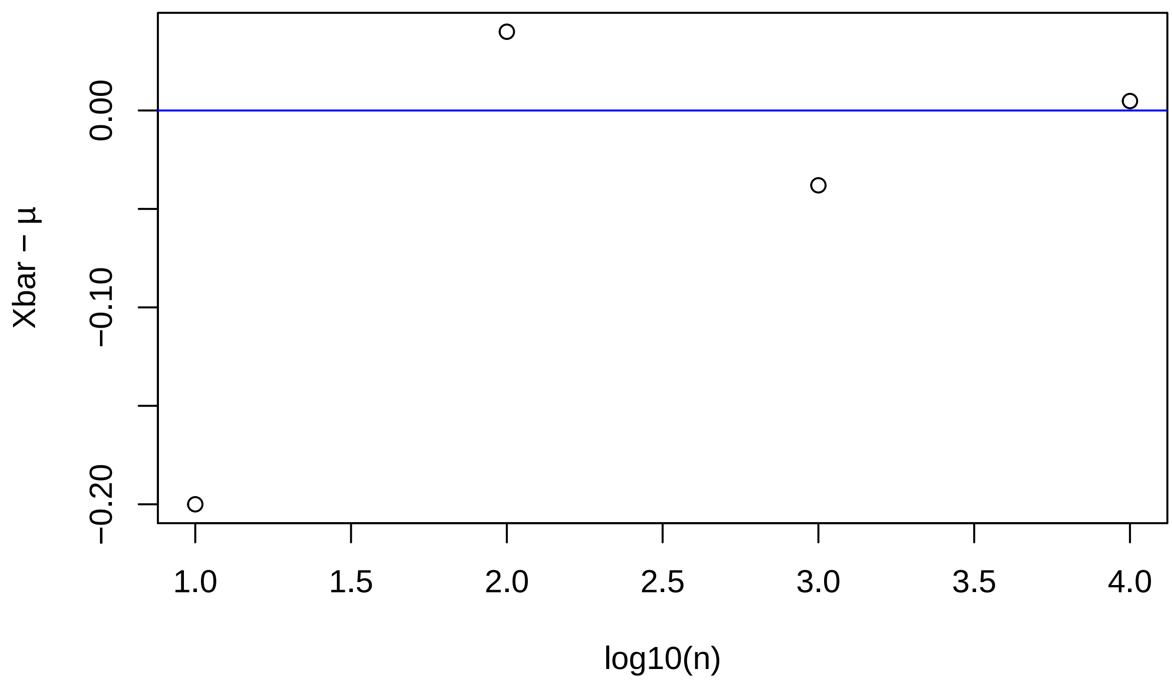
```
## [1] 0.9483
```

As we can see, the frequency is 94.83% which is really close to 95%. Therefore we can claim that the constructed CI is effective

3.

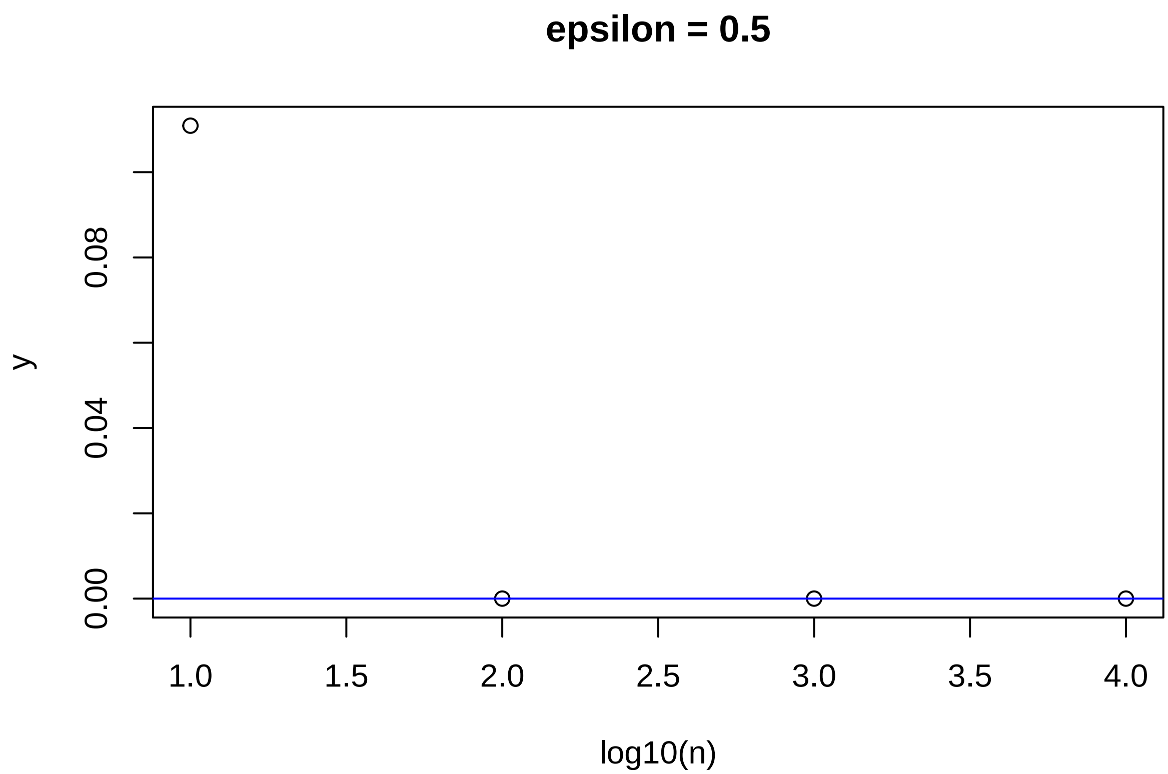
3.a)

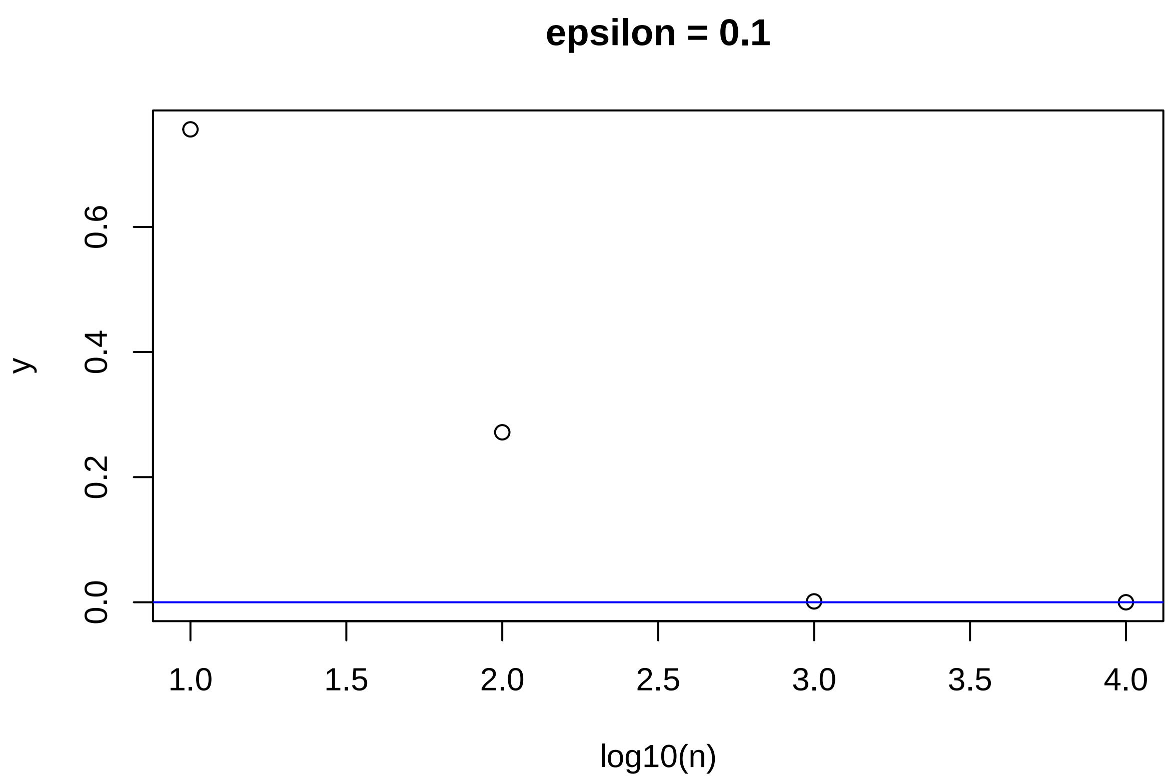
```
set.seed(1003643587)
N <- 10
n <- c(10,100,1000,10000)

# function that samples from uniform
samples <- function(n) {
  samplesReplaced = runif(n, min = 0, max = 1)
  samplesReplaced = replace(samplesReplaced, samplesReplaced<0.5, -1)
  samplesReplaced = replace(samplesReplaced, samplesReplaced>= 0.5, 1)
  return(samplesReplaced)
}
```

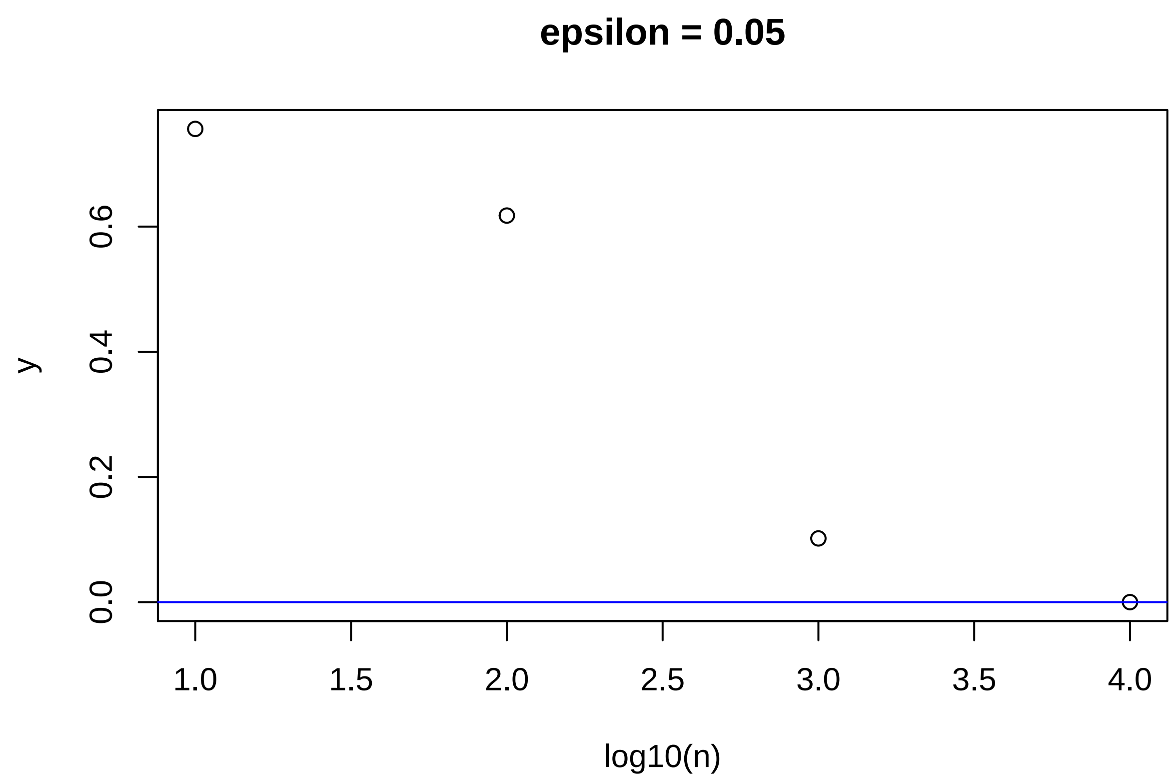


As we can see from the plot above, Xbar-mu is converging to 0 as n goes to infinity. for small n, we see that xbar-mu deviates from 0. However, as n increases, the plot gets increasingly closer to near 0.

3.b)


epsilon = 0.5
y
0.00
0.04
0.08
1.0
1.5
2.0
2.5
3.0
3.5
4.0
log10(n)


epsilon = 0.1
y
0.0
0.2
0.4
0.6
1.0
1.5
2.0
2.5
3.0
3.5
4.0
log10(n)



The plots above illustrates the law of large numbers. As n increases from 10 to 10000, we can see that the data points fall much closer to the mean(mu), 0 for each of the three plots. We can also see that as epsilon becomes smaller, the data points for smaller n's tend to be further away from mu, however they still all will converge to mu with large n.

3.c)

```
#install.packages("car")
library(car)
```

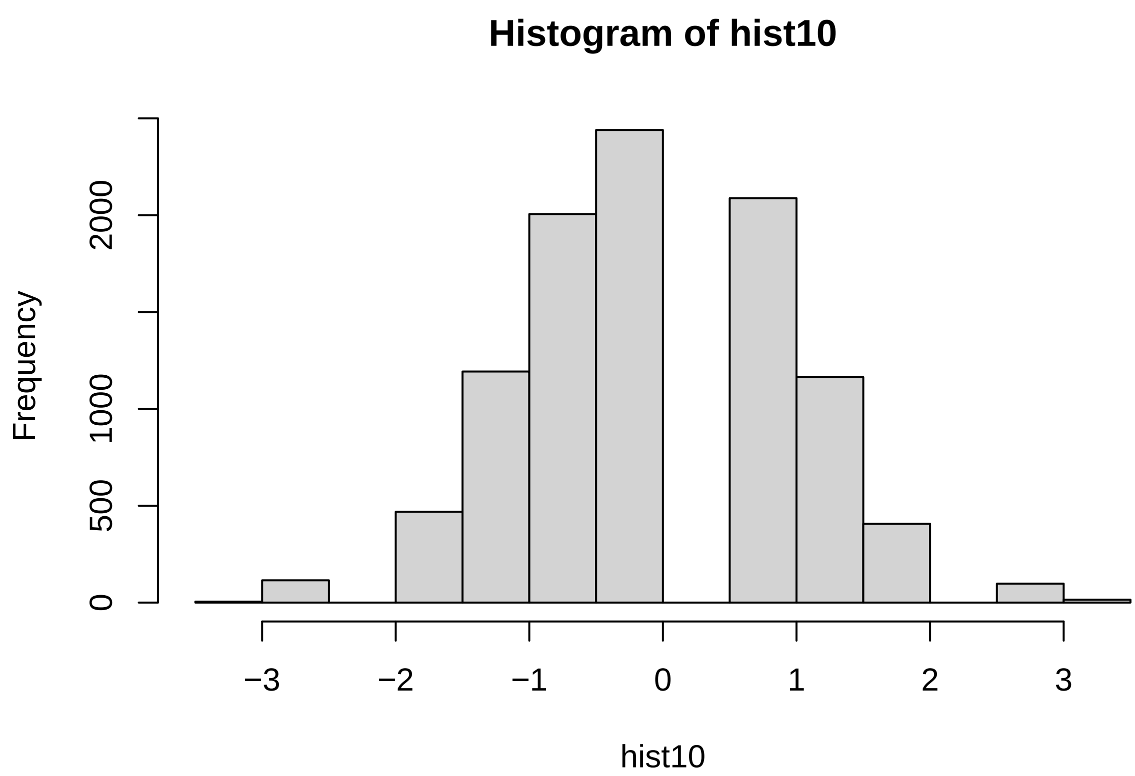
```
## Warning: package 'car' was built under R version 4.0.3
## Loading required package: carData
## Warning: package 'carData' was built under R version 4.0.3
```

```
set.seed(1003643587)
mu <- 0
# var(x) = E(X^2) = 1
# sigma^2 = 1
sigma <- 1
mean10 <- c()
mean100 <- c()
mean1000 <- c()
mean10000 <- c()
for (i in 1:10000) {
  mean10 <- append(mean10, sum(samples(10))/10)
```

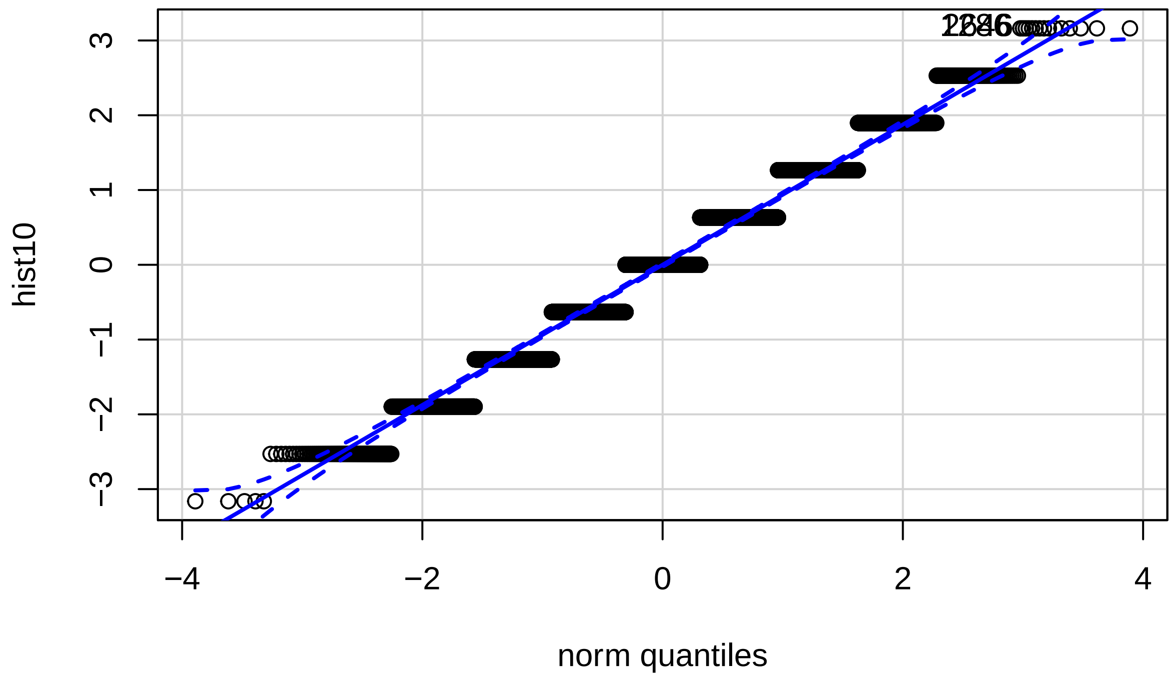
```r
  mean100 <- append(mean100, sum(samples(100))/100)
  mean1000 <- append(mean1000, sum(samples(1000))/1000)
  mean10000 <- append(mean10000, sum(samples(10000))/10000)
}
```

n = 10

```r
hist10 <- sqrt(10)*mean10/sigma
hist(hist10)
```



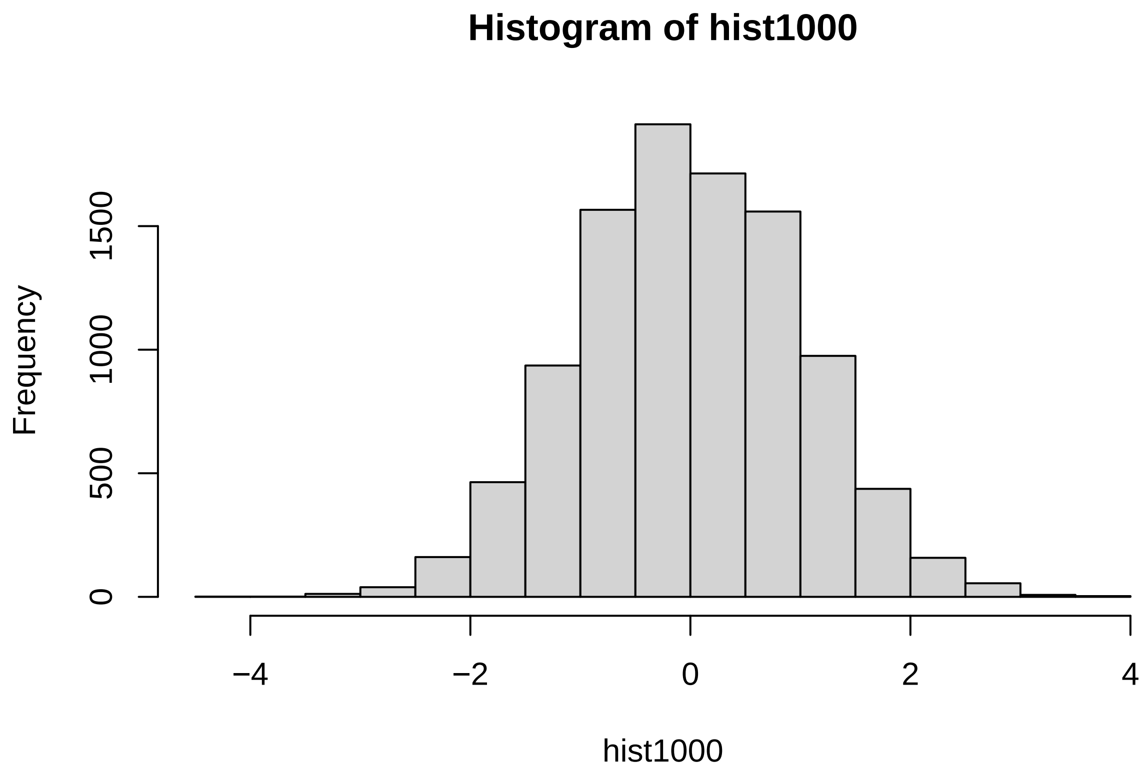

```r
qqPlot(hist10)
```



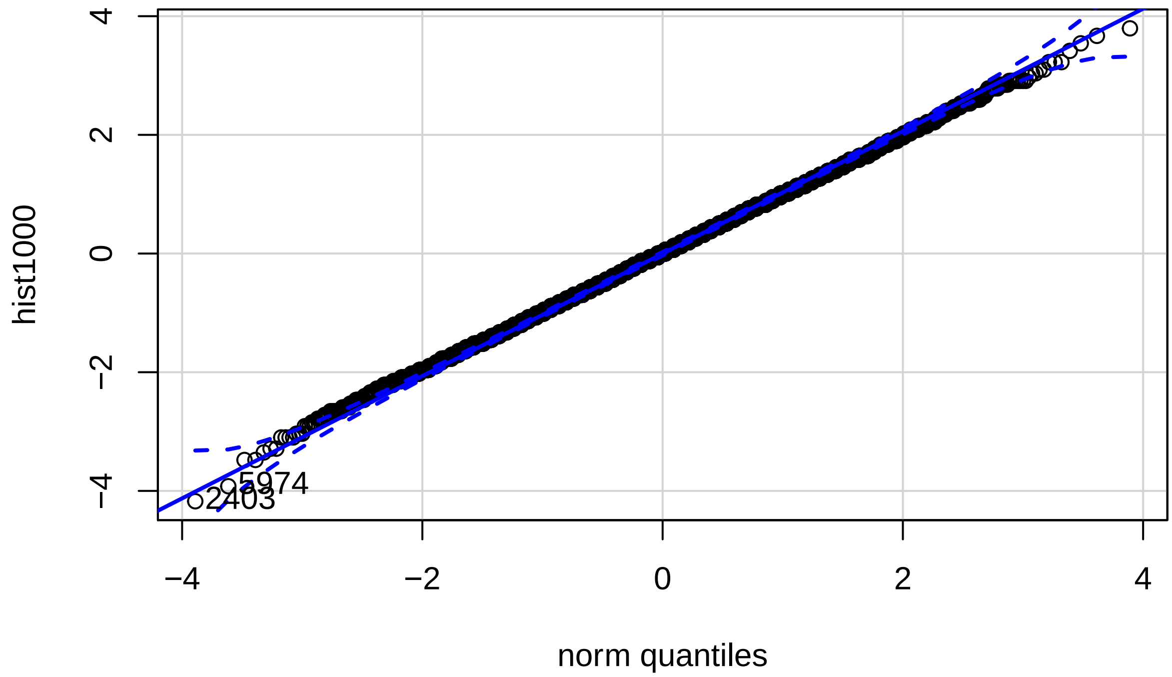

```
## [1] 1286 2646
```

n = 1000

```
hist1000 <- sqrt(1000)*mean1000/sigma
hist(hist1000)
```



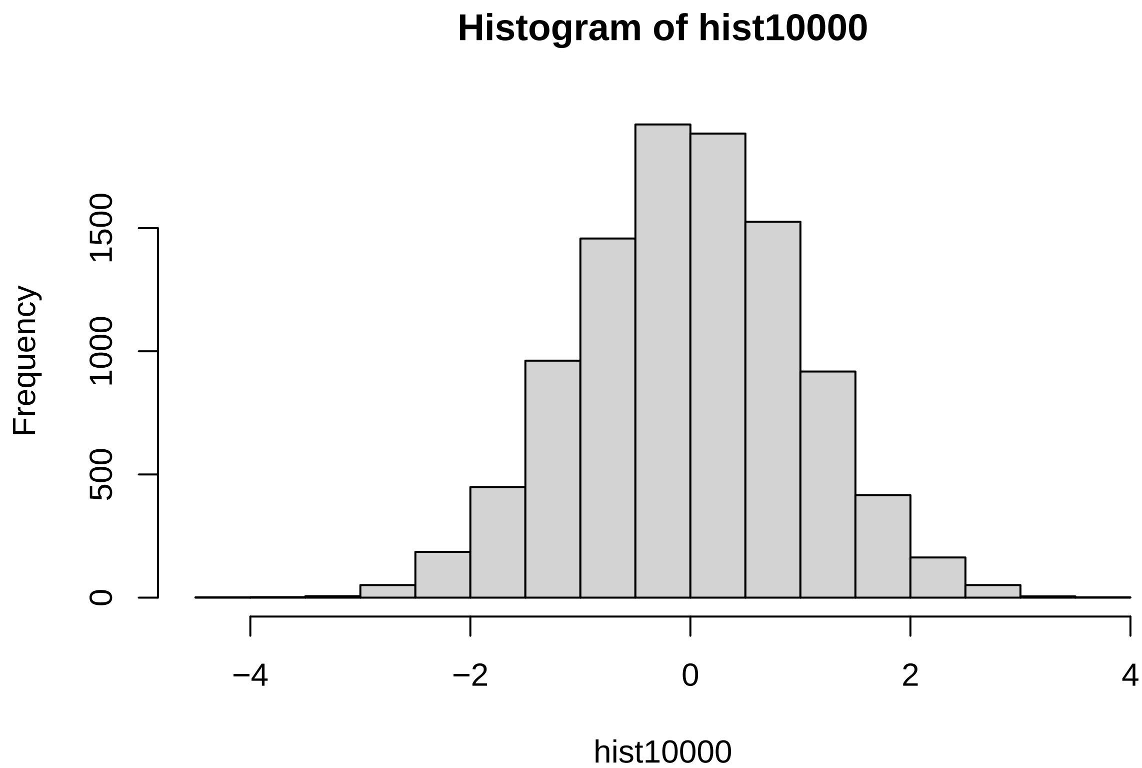

```
qqPlot(hist1000)
```



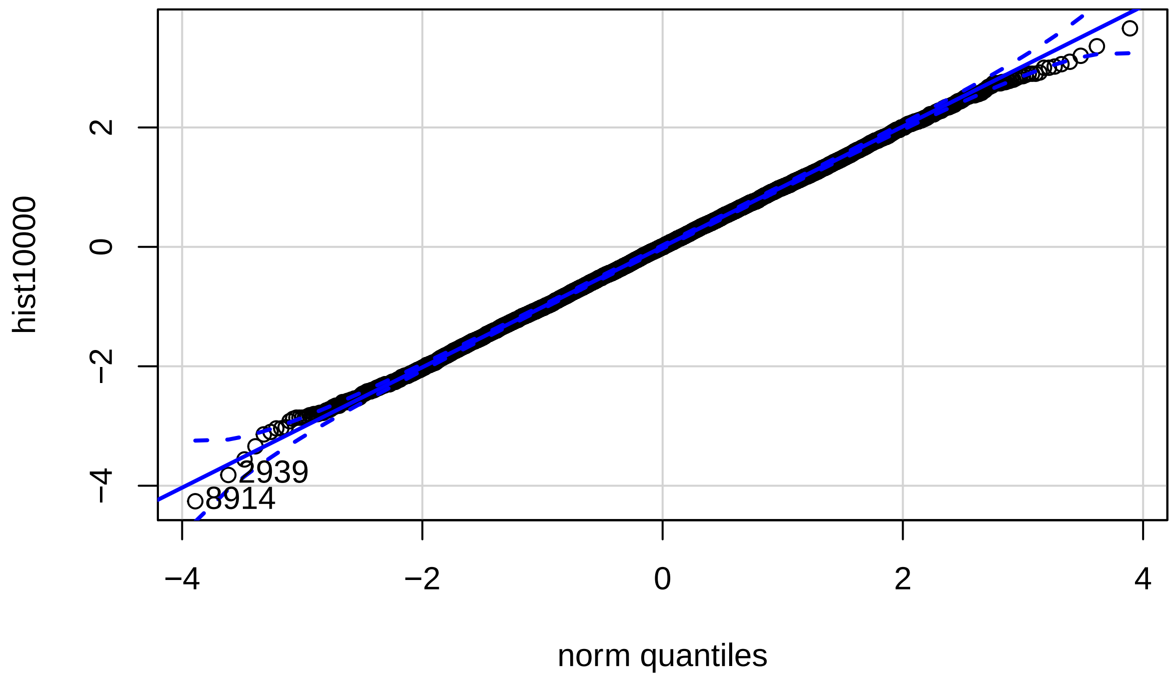

```
## [1] 2403 5974
```

n = 10000

```
hist10000 <- sqrt(10000)*mean10000/sigma
hist(hist10000)
```




```
qqPlot(hist10000)
```



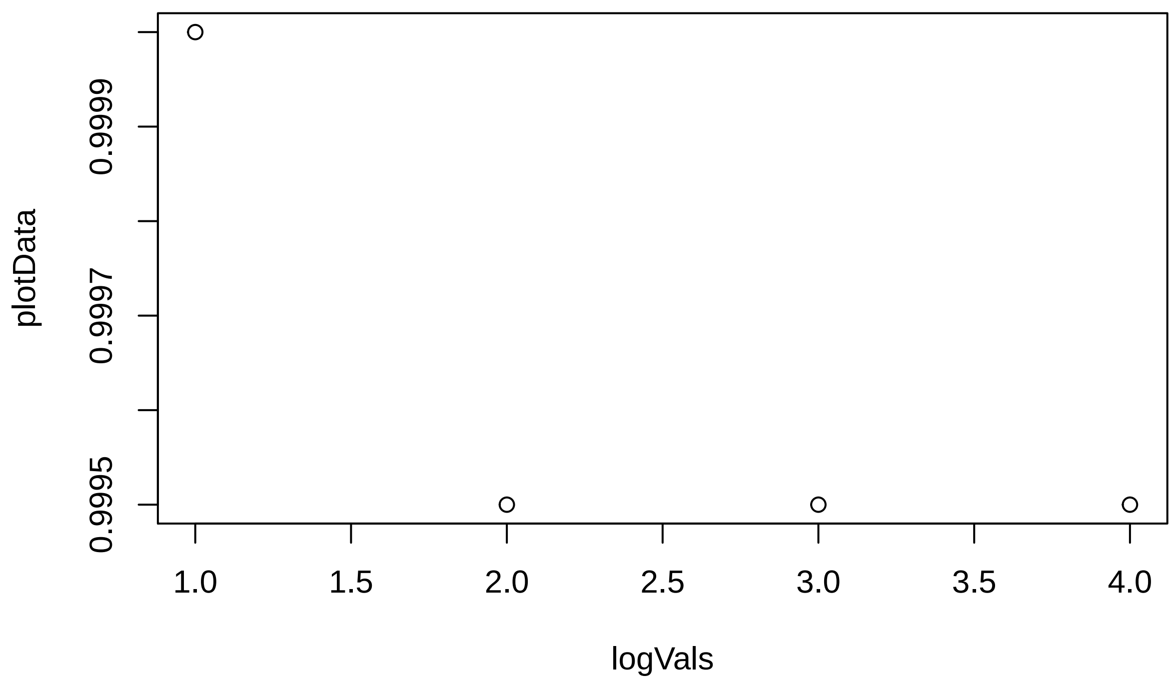

```
## [1] 8914 2939
```

As we can see from the QQ plots and histograms, as n increases we can see that the data is more like standard normal. The histograms look more like normal as n increases and the qq plots are more linear with less outliers. These plots also show the Central Limit Theorem as we have mu = E(x) and simga^2 = var(x). As n approaches infinity, sqrt(n)(xbar - mu)/sigma approaches the standard normal in distribution.
3.d)

```
averag <- function(data, e) {
  data <- abs(data)
  data <- replace(data, data > e, 1)
  data <- replace(data, data <= e, 0)
  return(data)
}
```

```
logVals = c(log(10, base = 10), log(100, base = 10), log(1000, base = 10), log(10000, base = 10))
Y10 <- rnorm(10,0,1)
Y100 <- rnorm(100,0,1)
Y1000 <- rnorm(1000,0,1)
Y10000 <- rnorm(10000,0,1)
data10 <- sqrt(10)*mean10 - Y10
data10 <- averag(data10, 0.001)
data100 <- sqrt(100)*mean100 - Y100
data100 <- averag(data100, 0.001)
data1000 <- sqrt(1000)*mean1000 - Y1000
data1000 <- averag(data1000, 0.001)
data10000 <- sqrt(10000)*mean10000 - Y10000
```

```r
data10000 <- averag(data10000, 0.001)
plotData <- c(mean(data10), mean(data100), mean(data1000), mean(data10000))
plot(logVals, plotData)
```



From the plot above, we see that it does not converge to Y in probability because even as n increases, the values do not necessarily approach 0. They still converge in distribution but this does not mean that all the values will be converging. As the standardized values of x approaches y in distribution, but not in probability.

4.

4.a)

```r
#install.packages("bigmemory")
library(bigmemory)
```

```
## Warning: package 'bigmemory' was built under R version 4.0.3
```

```r
X <- read.big.matrix("ratings.dat", type = "integer", col.names = c("UserID", "ProfileID", "Rating"))
head(X)
```

```
##      UserID ProfileID Rating
## [1,]  56669     39491      6
## [2,]  56919      8035     10
## [3,] 108853    102321     10
## [4,] 116784     52568      2
## [5,] 132748    220878     10
```

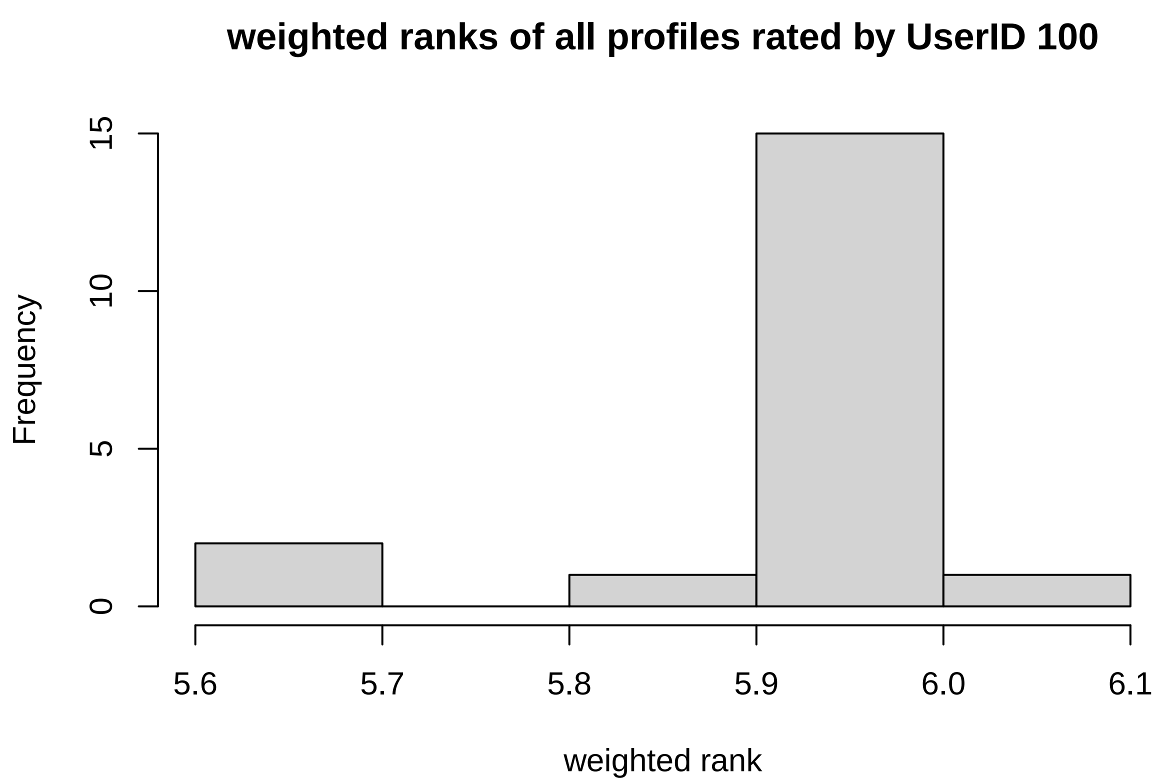
```
## [6,] 120139     29077      9
```

```r
  RatingAll <- X[, 3]
  C <- sum(RatingAll)/3000000
```

```r
weighted.rank <- function(ProfileID) {
  byProfile <- mwhich(X, "ProfileID", ProfileID, "eq")
  newX <- X[byProfile, ]
  byRating <- newX[, 3]
  v <- length(byRating)
  R <- sum(byRating)/v
  m <- 4182
  weightedRank <- ((v/(v+m))*R)+((m/(v+m))*C)
  return(weightedRank)
}
weighted.rank(39491)
```

```
## [1] 5.977027
```

```r
byUser <- mwhich(X, "UserID", 100, "eq")
byUser <- X[byUser, ]
byUser <- byUser[, 2]
eachProfile <- lapply(byUser, weighted.rank)
plotData <- unlist(eachProfile)
hist(plotData, main = "weighted ranks of all profiles rated by UserID 100", xlab = "weighted rank")
```



4.b)

```r
load("users.Rdata")
```

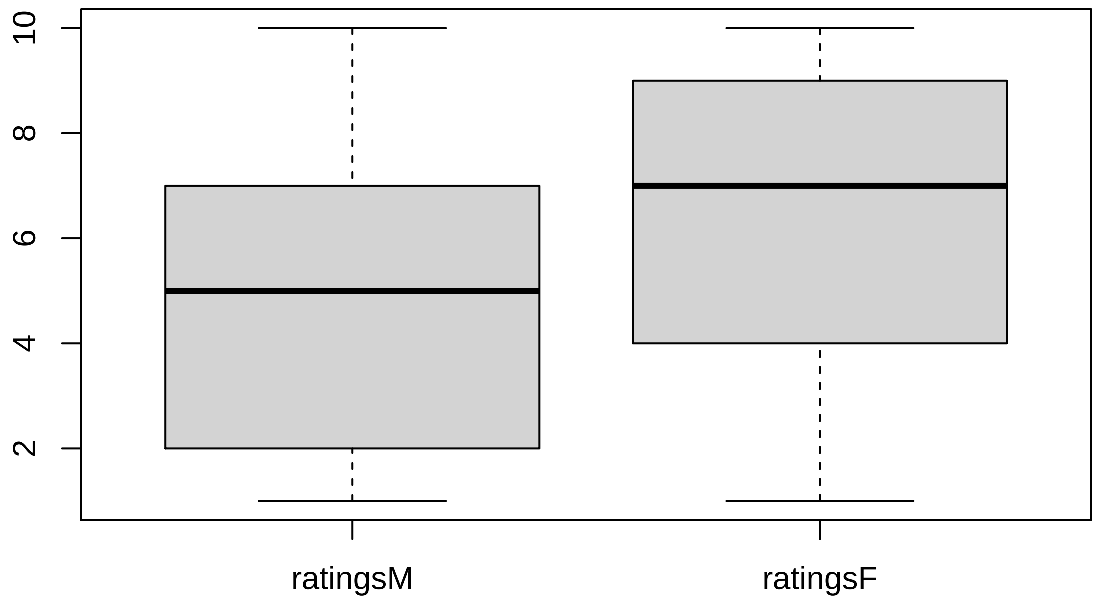
1) male users coming from New York State, State==New York

```r
ratingsM <- c()
newYorkM <- User[which(User$Gender == 'M' & User$State == 'New York'), ]
newYorkM <- newYorkM[ , 1]
for(i in newYorkM) {
  byUser <- mwhich(X, "UserID", i, "eq")
  byUser <- X[byUser, ]
  if(identical(nrow(byUser), NULL)){
    byUser <- byUser[3]
    ratingsM <- append(ratingsM, byUser)
  } else {
    byUser <- byUser[ , 3]
    ratingsM <- append(ratingsM, byUser)
  }
}
```

2) female users coming from California, State==CA

```r
ratingsF <- c()
CAF <- User[which(User$Gender == 'F' & User$State == 'CA'), ]
CAF <- CAF[ , 1]
for(i in CAF) {
  byUser <- mwhich(X, "UserID", i, "eq")
  byUser <- X[byUser, ]
  if(identical(nrow(byUser), NULL)){
    byUser <- byUser[3]
    ratingsF <- append(ratingsF, byUser)
  } else {
    byUser <- byUser[ , 3]
    ratingsF <- append(ratingsF, byUser)
  }
}
```

```r
z <- c("ratingsM", "ratingsF")
histData <- lapply(z, get, envir=environment())
names(histData) <- z
boxplot(histData)
```



Above is a boxplot of ratings of Males from New York, against ratings of Females from CA

4.c)

```r
#install.packages("biganalytics")
library(biganalytics)
```

```
## Warning: package 'biganalytics' was built under R version 4.0.3
## Loading required package: foreach
## Warning: package 'foreach' was built under R version 4.0.3
## Loading required package: biglm
## Warning: package 'biglm' was built under R version 4.0.3
## Loading required package: DBI
## Warning: package 'DBI' was built under R version 4.0.3
```

```r
head(X)
```

```
##      UserID ProfileID Rating
## [1,]  56669     39491      6
## [2,]  56919      8035     10
## [3,] 108853    102321     10
## [4,] 116784     52568      2
## [5,] 132748    220878     10
## [6,] 120139     29077      9
```

N=3000000 # number of rating records Nu=135359 # maximum of UserID Np=220970 # maximum of ProfileID user.rat=rep(0,Nu) # user.rat[i] denotes the sum of ratings given by user i user.num=rep(0,Nu) # user.num[i] denotes the number of ratings given by user i profile.rat=rep(0,Np) # profile.rat[i] denotes the sum of ratings given to profile i profile.num=rep(0,Np) # user.rat[i] denotes the number of ratings given to profile i for (i in 1:N){ # In each iteration, we update the four arrays, i.e. user.rat, user.num, profile.rat, profile.num, using one rating record. user.rat[X[i,‘UserID’]]=user.rat[X[i,‘UserID’]]+X[i,‘Rating’] # The matrix X here comes from the file ‘ratings.dat’ user.num[X[i,‘UserID’]]=user.num[X[i,‘UserID’]]+1 profile.rat[X[i,‘ProfileID’]]=profile.rat[X[i,‘ProfileID’]]+X[i,‘Rating’] profile.num[X[i,‘ProfileID’]]=profile.num[X[i,‘ProfileID’]]+1 if (i %% 10000==0) print(i/10000) } user.ave=user.rat/user.num profile.ave=profile.rat/profile.num X1=big.matrix(nrow=nrow(X), ncol=ncol(X), type= “double”, dimnames=list(NULL, c(‘UsrAveRat’,‘PrfAveRat’,‘Rat’))) X1[,‘Rat’]=X[,‘Rating’] X1[,‘UsrAveRat’]=user.ave[X[,’UserID’]] X1[,‘PrfAveRat’]=profile.ave[X[,’ProfileID’]] # X1 is the new data matrix we will work with in regression. user.ave profile.ave