ps7

Kehsin Su Esther 3033114294

November 17, 2017

```
knitr::opts_chunk$set(tidy = TRUE, cache = TRUE)
```

1 Q1

We can calculate the standard deviation of the simulated coefficent and compare it to the mean of standard error of coefficent. Or we can compare whether the calculated standard deviation fall into the confident interval which is build by the set of standard deviation

2 Q2

ref: https://math.stackexchange.com/questions/586663/why-does-spectral-norm-equal-the-largest-singular-valuecomment12

3 Q3

3.1 (a)

Use SVD decomposed(Details of calculation as attachment handwritten papers)

3.2 (b)

Details of calculation as attachment handwritten papers

4 Q4

4.1 (a)

Details of calculation as attachment handwritten papers

```
#decompose x
q1=qr(x).q
r1=qr(x).r
#decompose b, whcih equal to inverse of r1 and transpose of a
r2=qr(inv(r1)*t(a)).r
backsolve(r1 ,
  backsolve(t(r1),t(x)y)
  ) +
backsolve(r1,
  backsolve(t(r1),
    t(a)*
    backsolve(r2,
      backsolve(t(r2),
      (b-a*backsolve(r1,t(q1)*y)) )
      )
    )
  )
```

4.2 (b)

As showing in the following, the computing time reduce singifcantly after decomposing.

```
# using QR decomposed
beta_sol <- function(x, y, a, b) {
    ta <- t(a)
    tx <- t(x)
    r1 <- qr.R(qr(x))
    inv_r1 <- solve(r1)
    tr1 <- t(r1)
    q1 <- qr.Q(qr(x))
    tq_y <- t(q1) %*% y
    k <- inv_r1 %*% ta
    r2 <- qr.R(qr(k))
    tr2 <- t(r2)
    c_inv_d <- backsolve(r1, tq_y)
    beta_hat <- backsolve(r1, backsolve(r1, backsolve(r1, tq_y, transpose = T) +
        ta %*% backsolve(r2, backsolve(r2, b - a %*% backsolve(r1, tq_y), transpose = T)),
        transpose = T))

    return(beta_hat)
}

# original method
beta_orig <- function(x, y, a, b) {
    c <- t(x) %*% x
    d <- t(x) %*% y
    beta_hat <- solve(c) %*% d + solve(c) %*% t(a) %*% solve(a %*% solve(c) %*%
        t(a)) %*% (b - a %*% solve(c) %*% d)
    return(beta_hat)
}

# testing data
m <- 500
p <- 700
n <- 10000
x <- matrix(rnorm(n * p, 3, 3), n, p)
y <- c(rnorm(n * 1, 3, 3))
a <- matrix(rnorm(m * p, 3, 3), m, p)
b <- c(rnorm(m * 1, 3, 3))

# measure the performane improved function
start_time1 <- Sys.time()
result1 <- beta_sol(x, y, a, b)
end_time1 <- Sys.time()
end_time1 - start_time1
```

```
## Time difference of 19.87492 secs
```

```
# original function
```

```
start_time2 <- Sys.time()
result2 <- beta_orig(x, y, a, b)
end_time2 <- Sys.time()
end_time2 - start_time2

## Time difference of 5.872808 secs

# identical(result1,result2)
```

5 Q5

5.1 (a)

The main problem for that is the size of matirx will be too large to store if it is not sparse matrix. When trying to decomposed the matrix, we need to do O(n^3), $wherenequalto60million.Forexample,QRdecomposition,thememorycannotha$

5.2 (b)

Calculate beta by the following two rules:
First,calulcate vector first to reduce dimension Second, seperate the multiplation into several pair, and each pair with acceptable dimension(600 or 630) Note: Details of calculation as attachment handwritten papers

```
sls(x,y,z) {
  a=t(x)*z
  b=t(z)*z
  c=t(z)*y
  d=inv(t(z))
  e=a*d
  f=d*c
  result=inv(e*t(a))*(af)
}
```

6 Q6

```
require(Matrix)
require(matrixcalc)
n <- 100
evalu <- matrix(sapply(1:n,function(x) as.numeric(seq(from = 1, to = 1+2^(i-n), length.out = n))) ,n,n)

## Error in seq.default(from = 1, to = 1 + 2^(i - n), length.out = n):   'i'

testpd <- function(evalu){
  z <- matrix(rnorm(n,5,5),1,n)
  a <- crossprod(z,z)
  evect <- eigen(a)$vectors    #eignvector of a
  #dig_m <- Diagonal(x=evalu)
  dig_m  <- matrix(rep(0,n*n),n,n)
  diag(dig_m ) <- evalu
  b <- evect %*% dig_m  %*% t(evect) #simulate a matrix
  #test <- is.positive.definite(b, tol=1e-8)
  #test<- is.symmetric.matrix(b)
  test <- min(eigen(b)$values)>0
```

```r
  #Estimate the reciprocal of the condition number of a matrix.
  cond_num <- rcond(b)

  return( test )
}
result <- vector("list",length = n)
for (i in 1:100){
result[i] <- testpd(evalu[,i])
}
```

```
## Error in testpd(evalu[, i]):   'evalu'
```

```r
mean(result)
```

```
## Warning in mean.default(result):  argument is not numeric or logical:  returning NA
```

```
## [1] NA
```

```r
#The condition number of a regular (square) matrix is the product of the norm of the matrix and the nor
#Note the following codes refer from Cheng-Han Hsieh
library(ggplot2)
n <- 100
Z <- matrix(rnorm(n),1,n)
A <- t(Z) %*% Z
e <- eigen(A)$vectors
myFun <- function(values){
  diagmatrix <- matrix(rep(0,n*n),n,n)
  diag(diagmatrix) <- values
  mymatrix <- e %*% diagmatrix %*% t(e)
  return(mymatrix)
}
test <- vector("list",length = n)
for(i in 1 : n){
  test[[i]] <- as.numeric(seq(from = 1, to = 1+2^(i-n), length.out = n))
}
result <- lapply(test, myFun)
for(j in 1:n){
  if(is.symmetric.matrix(result[[j]]) == FALSE) #should check by whether all eigenvalues are positve, b
    break
}
j
```

```
## [1] 48
```

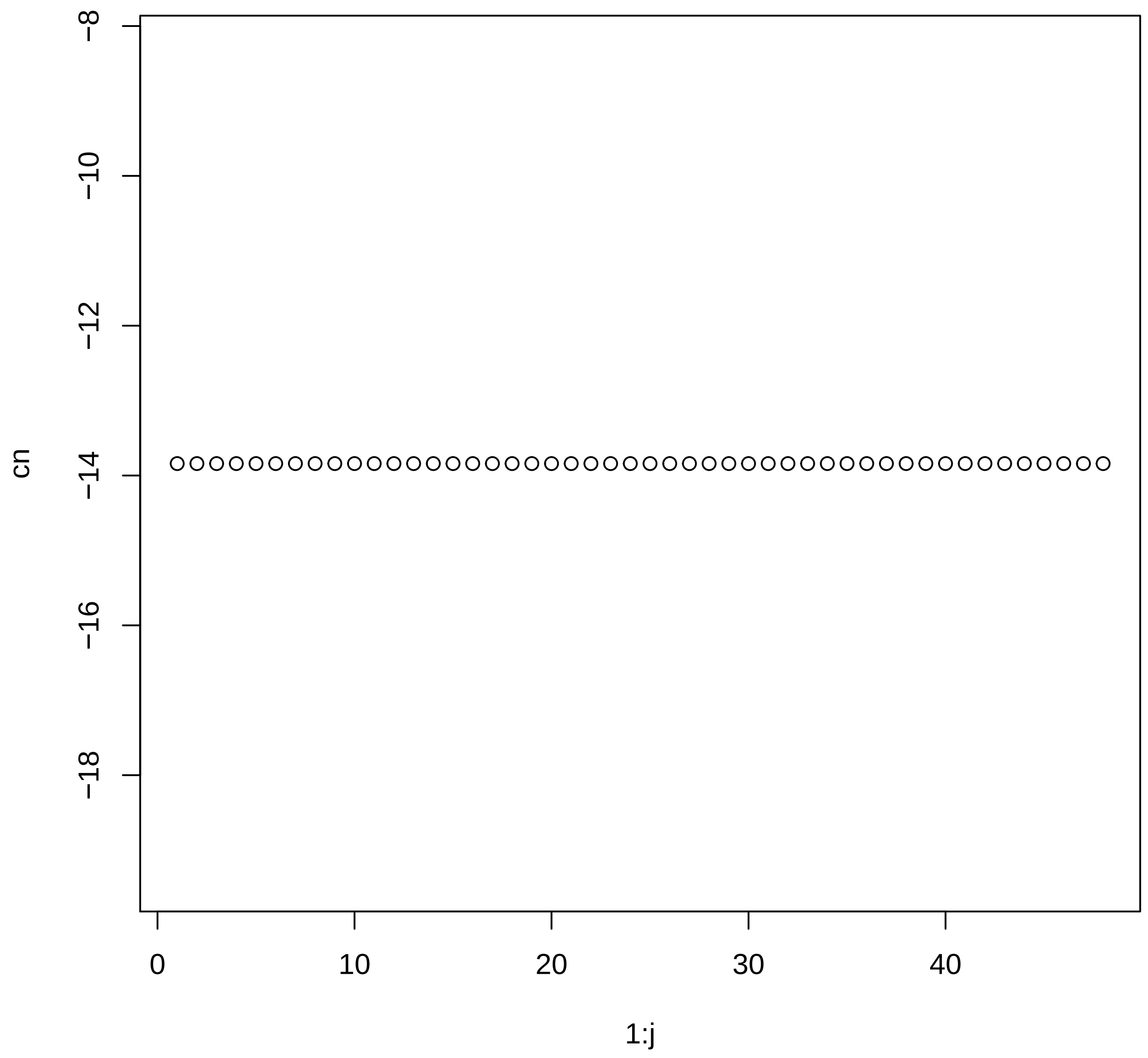
```r
#As 100-j equal to 52, which is machine epson(2^-52)
# condition number of j eigenvalues
log10(kappa(myFun(test[[j]])) - 1)
```

```
## [1] -13.84065
```

```r
rcond(result[[j]])
```

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

```r
cn <- vector("list",j)
for ( i in 1: j ){
  cn[[i]] <- log10(kappa(myFun(test[[i]])) - 1)
  }
plot(1:j,cn)
```



cn
−8
−10
−12
−14
−16
−18
0
10
20
30
40
1:j