243 PS7

Hsieh Cheng Han

November 16, 2017

1.

We have 1000 estimate of the coefficient, then we can calculate its plug-in variance. Comparing this plug-in variance and the mean of 1000 standard error, we can determine whether the standard error properly characterizes the uncertainty of the estimated regression coefficient. (They should be close enough)

2&3

I attached the hand-written paper on the final page.

4.(a)

I attached the hand-written paper on the final page.

(b)

First, we try to use the pseudo-code in (a) to solve beta.

```
# original violence
vbeta <- function(X,Y,A,b){
 C <- crossprod(X)
 d <- t(X) %*% Y
 solve(C) %*% d + solve(C) %*% t(A) %*% solve(A %*% solve(C) %*% t(A)) %*% (-A
}
# pseudo-code
mybeta <- function(X,Y,A,b){
  # QR decomposition of X
  R <- qr.R(qr(X))
```

```
  Q <- qr.Q(qr(X))
  # R' is the QR decomposition of t(A %*% inv(R))
  R1 <- qr.R(qr(t(A %*% solve(R))))
  result <- backsolve(R, backsolve(R, t(X) %*%Y + t(A) %*% backsolve(R1, backs
  return(as.vector(t(result)))
}
```

Also, we use quadratic programming package in R to solve the problem. solve.QP function can solve quadratic programming problems of the form min(-d^Tb + $\frac{1}{2}$ b^TDb) with the constraint A^Tb = b_0.
Here D = X^TX, d = X^TY.

```
library(quadprog)
qd <- function(X,Y,A,b){
  D <- t(X) %*% X
  d <- t(X) %*% Y
  return(solve.QP(D, d, t(A), b, meq = 1, factorized = FALSE)$solution)
}
m <- 1000
n <- 1000
p <- 1000
A <- matrix(rnorm(m*p), nrow = m)
X <- matrix(rnorm(n*p), nrow = n)
beta <- rnorm(p)
Y <- X %*% beta + rnorm(n,sd = 0.01)
b <- A %*% ( beta + rnorm(p,sd = 0.01))
# system time : original violent calculation
system.time(vbeta(X,Y,A,b))

##    user  system elapsed
##  15.961   0.186  16.486

system.time(mybeta(X,Y,A,b))

##    user  system elapsed
##   4.772   0.085   5.077

system.time(qd(X,Y,A,b))
```

```
##    user  system elapsed
##   8.955   0.064   9.185
```

Apparently, the pesudo-code method is much better than violent method.

5.(a)

On calculating $Z(Z^TZ)^{-1}Z^T$, it will be a 60 millon * 60 million matrix. There is no enough memory space to store this giant matrix. Also, the condition that X and Z are both sparse matrices doesn't guarantee $Z(Z^TZ)^{-1}Z^T$ is sparse as well.

(b)

I attached the hand-written paper on the final page.

6.

```
library(matrixcalc)
library(ggplot2)
n <- 100
Z <- matrix(rnorm(n),1,n)
A <- t(Z) %*% Z
e <- eigen(A)$vectors
# calclulate the estimated eigenvalues and also the condition number
myFun <- function(values){
  diagmatrix <- matrix(rep(0,n*n),n,n)
  diag(diagmatrix) <- values
  mymatrix <- e %*% diagmatrix %*% t(e)
  return(mymatrix)
}
# First we check when mymatrix is not numerically p.d
# Create different test sets of eigenvalues
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)
```

The result shows that mymatrix has turned into asymmetric matrix as eigenvalues become more diversified. Let's check when the transformation happens.

```
for(j in 1:n){
  if(is.symmetric.matrix(result[[j]]) == FALSE)
    break
}
j
```

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

That is, when the eigenvalues are from 1 to $1 + 2^{-52}$, the matrix is no more numerically symmetric(of course not pd). Actually, 2^{-52} is very the machine epsilon of R, which makes sense.

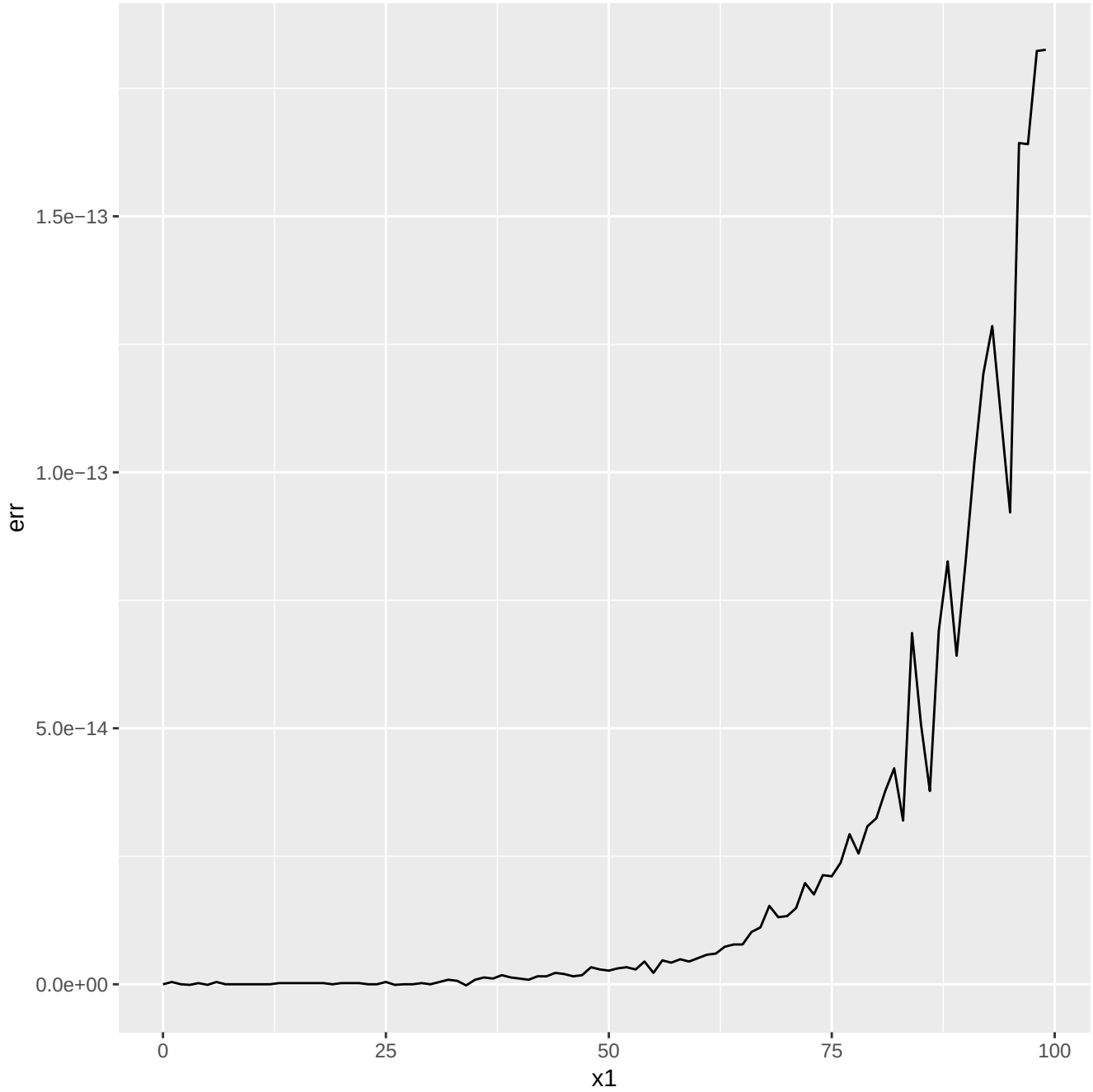
```
# get the condition number of this eigenvalues
log10(kappa(myFun(test[[48]])) - 1)
```

```
## [1] -13.90537
```

As a result, when the condition number increases from 1 to approximately $1 + 10^{-13}$, the matrix becomes numerically asymmetric.
Finally, we check how the error in the estimated eigenvalues relative to the known true values varies with the condition number and the magnitude of the eigenvalues.

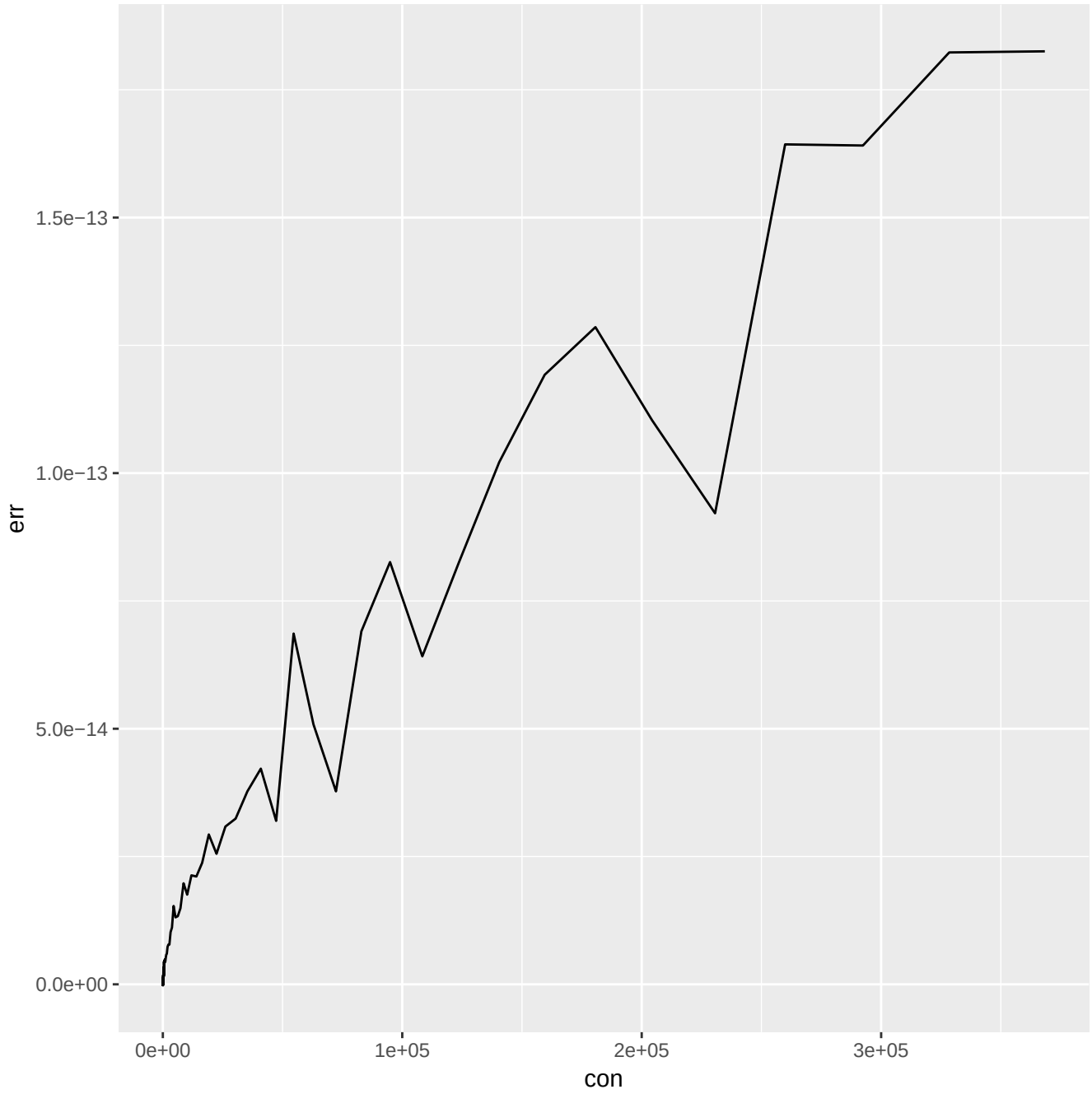
```
# We define a method to define the error in the estimated eigenvalues relati
error <- function(value){
  mymatrix <- myFun(value)
  # estimated eigenvalues in increasing order, which makes it
  # easy to be compared with true values
  myvalue <- sort(eigen(mymatrix)$values, decreasing = FALSE)
  return(log(mean(abs(myvalue / value))))
}
testseq <- vector("list",length = n)
for(i in 1 : n){
  test[[i]] <- as.numeric(seq(from = 1, to = (1.1)^(i-1), length.out = n))
  testseq[[i]] <- i - 1
}
resulterror <- lapply(test, error)
```

```
conditionnumber <- lapply(test, function(x) {return(kappa(myFun(x)))})
# Plot the relationship
data <- data.frame( x1 = unlist(testseq), err = unlist(resulterror), con = unl
ggplot(data, aes(x = x1)) + geom_line(aes(y = err))
```



x1 represents the degree of diversification from eigenvalues(magnitude), and err represents the degree of error between original eigenvalues and estimated eigenvalues, we can see that the error increases with the magnitude.

```
ggplot(data, aes(x = con)) + geom_line(aes(y = err))
```



con are the condition numbers. As a result, we can also see that the error increases with the condition number.