STAT243-PS7

Jinhui Xu

Nov. 2017

I work with Xin Shi, Xiao Li, Shan Gao, Zicheng Huang, Junyuan Gao, Junyi Tang

1 Question 1

From 1000 estimates of coefficent, we can calculate an standard error $\hat{se}^*$ of these 1000 estimates.
From 1000 standard errors :assume that $se_1 \leq se_2 ... \leq se_{1000}$, we can get a 95% confidence interval $[se_{26}, se_{975}]$
The standard error properly characterizes the uncertainty of the estimated regression coefficient if $\hat{se}^*$ falls into the above 95% confidence interval.

2 Question 2

$$||A||_2 = \sup_{||z||_2=1} \sqrt{(Az)^T(Az)}$$

As A is symmetric, it can be written as $A = U\Sigma U^T$, where U is a basis of R^n, $UU^T = U^TU = I$ and $\Sigma = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$. Then any vector $z \in R^n$ can be written as $U \times z^*$, where $z^* = (z_1, z_2, ..., z_n)$ is the coordinates of z in the basis of U.
In this way, we can compute Az as follows:

$$\begin{aligned} Az &= U\Sigma U^T \times z \\ &= U\Sigma U^T \times U \times z^* \\ &= U\Sigma \times z^* \end{aligned}$$

Then we can compute $\sqrt{(Az)^T(Az)}$:

$$\begin{aligned} \sqrt{(Az)^T(Az)} &= \sqrt{(U\Sigma \times z^*)^T(U\Sigma \times z^*)} \\ &= \sqrt{(z^*)^T\Sigma U^T \times U\Sigma z^*} \\ &= \sqrt{(z^*)^T(\Sigma)^2 z^*} \\ &= \sqrt{\sum_{i=1}^{n} \lambda_i^2 z_i^2} \end{aligned}$$

In this way, when $||z||_2 = 1$

$$\begin{aligned}\sqrt{(Az)^T(Az)} &\leq \sqrt{max(\lambda_i^2)} \\ &= max|\lambda_i|\end{aligned}$$

Assume $|\lambda_j| = max|\lambda_i|, i = 1, ..., n$, then let $z^* = \begin{bmatrix}0 & \cdots 0 & 1 & 0 & \cdots & 0\end{bmatrix}^T \longrightarrow 1$ at jth element, which can satifies the equation:

$$\begin{aligned}\sqrt{(Az)^T(Az)} &= \sqrt{\sum_{i=1}^{n} \lambda_i^2 z_i^2} \\ &= |\lambda_j| \\ &= max|\lambda_i|\end{aligned}$$

which means that $||A||_2 = \sup_{||z||_2=1} \sqrt{(Az)^T(Az)} = max|\lambda_i|$

3 Question 3

3.1 (a)

According to SVD, A always can be decomposed as

$$A = UDV^T \qquad D = \begin{bmatrix}\lambda_1 & & \\ & \ddots & \\ & & \lambda_n\end{bmatrix}$$

where U, V are matrics with althonormal columns which means that $U^TU = I, V^TV = I$, then

$$\begin{aligned}X = UDV^T &\Longrightarrow X^TX = VDU^TUDV^T = VD^2V^T \\ &\Longrightarrow X^TX \times V = VD^2V^TV = VD^2 = \begin{bmatrix}\lambda_1^2 v_1, & \cdots & , \lambda_n^2 v_n\end{bmatrix} \qquad v_i \text{ is column vector of V}\end{aligned}$$

Therefore, V are eigenvectors of X^TX and the corresponding eigenvalues are the diagonal values of D^2, which equals to the squares of diagonal values of D(the squares of the singular values of X).

Proof of semi-positive for X^TX

$$\forall y \in R^n \qquad y^TX^TXY = (Xy)^T(Xy) = ||Xy||^2 \geq 0$$

3.2 (b)

If we have computed the eigendecomposition of Σ, $\Sigma = UDU^T$ where $U^TU = I, D = \begin{bmatrix}\lambda_1 & & \\ & \ddots & \\ & & \lambda_n\end{bmatrix}$

Then we can write cI as $cI = UCU^T$, where $C = \begin{bmatrix}c & & \\ & \ddots & \\ & & c\end{bmatrix}$

In this way,

$$Z = \Sigma + cI = UDU^T + UCU^T = U(C + D)U^T$$

where $C + D = \begin{bmatrix}\lambda_1 + c & & \\ & \ddots & \\ & & \lambda_n + c\end{bmatrix}$, so the eigenvalues of Z are $\lambda_1 + c, ..., \lambda_n + c$, which need n additions.

4 Question 4

4.1 (a)

As QR decomposition has less computation cost and be more stable than Cholesky decomposition, I first do QR decomposition for X.Then :

$$C = X^T X = (QR)^T (QR) = R^T Q^T QR = R^T R$$

Calculate $\hat{\beta}$

$$\begin{aligned}\hat{\beta} &= (R^T R)^{-1} d + (R^T R)^{-1} A^T (A(R^T R)^{-1} A^T)^{-1} (-A(R^T R)^{-1} d + b) \\ &= (R^T R)^{-1} d + (R^T R)^{-1} A^T ((AR^{-1})(AR^{-1})^T)^{-1} (-A(R^T R)^{-1} d + b)\end{aligned}$$

As $R^T R$ and R are both upper triangular matrics, we can use backsolve to do above computation. R code just like the following:

```
QR_beta<-backsolve(R, backsolve(R,
              t(X)%*% Y+ t(A) %*% backsolve(AR,backsolve(AR,-A%*%backsolve(R, t(Q)%*%Y)
                  +b,transpose=T)),transpose=T))
```

4.2 (b)

First define two function to calculate $\hat{\beta}$, one uses QR decomposition, the other one uses solve to inverse matrics.

```
#define the function which uses QR decomposition
QR_beta<-function(A,X,Y,b){

  #get Q, R of in QR decompodition of X
  X_R<-qr.R(qr(X))
  X_Q<-qr.Q(qr(X))

  #get R in QR decomposition of AR^{-1}
  AR_R<-qr.R(qr(t(A%*%solve(X_R))))

  #calculate beta hat in the same way in question (a)
  beta<-backsolve(X_R, backsolve(X_R, t(X)%*%Y+t(A)%*%
                                   backsolve(AR_R,backsolve(AR_R,
                                             -A%*%backsolve(X_R, t(X_Q)%*%Y)+b,transpose=T)),
                                 transpose=T))
 return(beta)
}

#define the function of methods which uses solve to calculate the inverse of matrics
solve_beta<-function(A,X,Y,b){
    d<-t(X)%*%Y
    solve(crossprod(X))%*%d+
             solve(crossprod(X))%*%t(A)%*%
                  solve(A%*%solve(crossprod(X))%*%t(A))%*%(-A%*%(solve(crossprod(X))%*%d)+b)
}
```

Then I give an example to examine the efficency of QR decomposition

```
#set basic parameters
m<-100
n<-1000
p<-1000
A<-matrix(rnorm(m*p),m)
X<-matrix(rnorm(n*p),n)
Y<-rnorm(n)
b<-rnorm(m)
d<-t(X)%*%Y

#compare the result of two methods
solve_beta(A,X,Y,b)[1:5]

## [1] -0.276921371  0.005403068 -0.067521364  0.035844192 -0.158073372

QR_beta(A,X,Y,b)[1:5]

## [1] -0.276921371  0.005403068 -0.067521364  0.035844192 -0.158073372

#compare the time used to run the two functions
system.time(solve_beta(A,X,Y,b))

##    user  system elapsed
##   6.322   0.083   6.597

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

##    user  system elapsed
##   2.426   0.054   2.590
```

From the result, we can see QR decomposition runs faster and gets the same result with the oridinary method. In addition, as we know, the advantage in speed can be much more obvious when p is larger. Therefore, using QR decomposition should be a better choice when we face large dimension matrics multiplication.

5 Question 5

5.1 (a)

Although X, Z are sparse, $\hat{X}$ can be dense matrics. So it may take large memory use. And in this way, calculating $\hat{\beta}$ also has large computation cost and takes large memory use. So we can not do calculation in that way.

5.2 (b)

As X, Z are both sparse, we can combine two equations and only use X and Z to calculate the value of $\hat{\beta}$

$$\hat{X} = Z(Z^TZ)^{-1}Z^TX \Longrightarrow \hat{X}^T\hat{X} = (Z(Z^TZ)^{-1}Z^TX)^TZ(Z^TZ)^{-1}Z^TX = X^TZ(Z^TZ)^{-1}Z^TX$$

Then

$$\begin{aligned}\hat{\beta} &= (\hat{X}^T\hat{X})^{-1}\hat{X}^Ty \\ &= (X^TZ(Z^TZ)^{-1}Z^TX)^{-1}(Z(Z^TZ)^{-1}Z^TX)^Ty \\ &= (X^TZ(Z^TZ)^{-1}Z^TX)^{-1}(X^TZ)(Z^TZ)^{-1}(Z^Ty)\end{aligned}$$

Because we can use spam package in R to deal with sparse matrics multiplications, we only care about several dense matrics. we can find the dimensions of $X^TZ(Z^TZ)^{-1}Z^TX$, X^TZ, Z^TZ,Z^Ty are 600×600, 600×630, 630×630, 630×1, which are relatively small. So we can do these computations in R more efficiently.

6 Question 6

```
#create eigenvectors
set.seed(1)
Z<-matrix(rnorm(10000),100)
true_eigenvec<-eigen(crossprod(Z))$vectors

#the vector used to record whether matrix is not positive definite
non_positive<-c(rep(0,100))

#the vectors used to record condition number and error
con_num<-c()
error<-c()

for(i in 0:99){
  #set condition number range from 1 to about 1e25
  Min<-10^(-i/8)
  Max<-10^(i/8)
  Sigma<-diag(c(Min,rep(1,98),Max))
  true_eigenval<-diag(Sigma)
  con_num[i]<-Max/Min

  #get the eigenvalues calculated by R
  R_eigenval<-eigen(true_eigenvec%*%Sigma%*%t(true_eigenvec))$values

  #judge if matrix is positive definite, and record it
  if(sum(R_eigenval>0)<100)
    non_positive[i]<-1

  #calculate the error
  error[i]<-sqrt(crossprod(true_eigenval-R_eigenval))
}
```

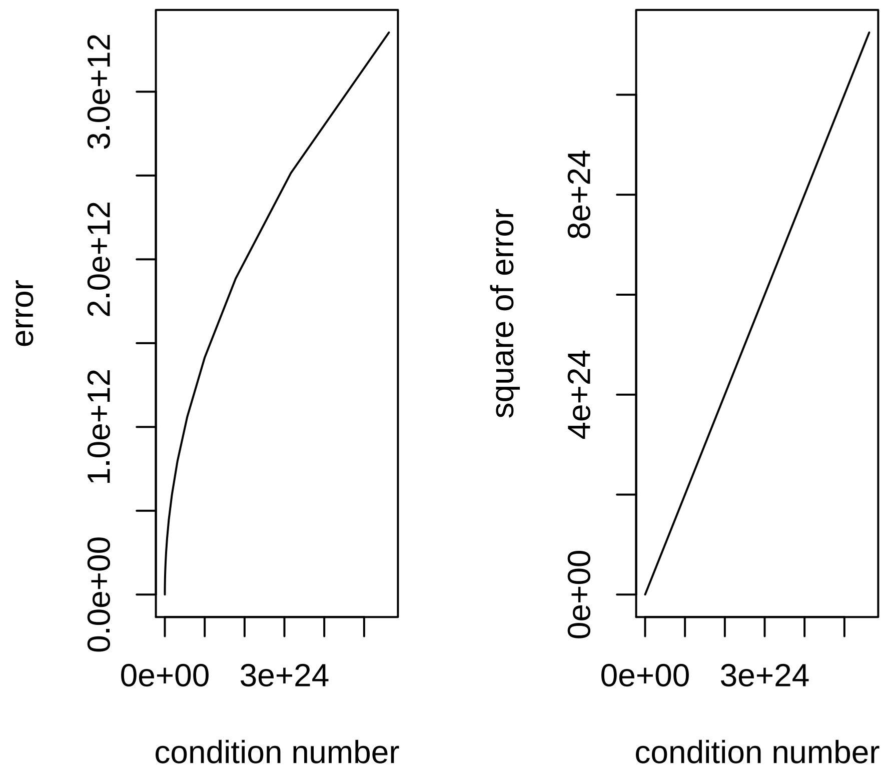
view at waht condition number the matrics would be non-positive definite. We can find that when the condition number be larger than 1e17, the matrix begin to be numerically non-positive definite

```
con_num[which(non_positive==1)]

##  [1] 5.623413e+17 1.778279e+18 3.162278e+18 5.623413e+18 1.000000e+19
##  [6] 1.778279e+19 3.162278e+19 5.623413e+19 1.000000e+21 1.778279e+21
## [11] 1.778279e+22 3.162278e+22 5.623413e+22 1.000000e+23 5.623413e+23
## [16] 1.778279e+24 3.162278e+24 5.623413e+24
```

view the relationship between condition number and error in plot. We can find that the square of error and condition number is near linear relationship.

```
par(mfrow=c(1,2))
plot(con_num,error,type='l',xlab='condition number',ylab='error')
plot(con_num,error^2,type='l',xlab='condition number',ylab='square of error')
```



error
0.0e+00
1.0e+12
2.0e+12
3.0e+12
0e+00
3e+24
condition number
square of error
0e+00
4e+24
8e+24
0e+00
3e+24
condition number