STAT 243 PS 7

Junyuan Gao(SID:26484653)

November 18, 2017

1 Q1

Suppose the estimates of coefficients of 1000 datasets are $\hat{\beta_1}, ..., \hat{\beta_{1000}}$. We can calculate the standard error of $\hat{\beta_1}, ..., \hat{\beta_{1000}}$ and compare it to the mean of $se(\hat{\beta_1}), ..., se(\hat{\beta_{1000}})$. If the two values are similar, we can say that the standard error properly characterizes the uncertainty of the estimated regression coefficient.

2 Q2

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

Since $A^T A$ is symmetric, we can write eigen decomposition of $A^T A$ as

$$A^T A = U^T \Sigma U = U^T \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} U$$

where $\lambda_i' s$ are the eigenvalues of A.
Plug it into $\|A\|_2$, we can get

$$\|A\|_2 = \sup_{\|z\|_2=1} \sqrt{z^T U^T \Sigma^2 U z} = \sup_{\|z\|_2=1} \sqrt{y^T \Sigma^2 y}$$

where $y = Uz$ and U is orthogonal. Since U orthogonal and $\|z\|_2 = 1$, we have $\|y\|_2^2 = \|Uz\|_2^2 = z^T U^T U z = z^T z = \|z\|_2^2 = 1 \implies \|y\|_2 = 1$.

With this property, we have

$$\begin{aligned}\|A\|_2 = \sup_{\|y\|_2=1} \sqrt{y^T\Sigma^2 y} = \sup_{\|y\|_2=1} \sqrt{\sum_{i=1}^n \lambda_i^2 y_i^2} \\ \le \sqrt{\sum_{i=1}^n \max_i |\lambda_i^2| y_i^2} \\ = \sqrt{\max_i |\lambda_i^2| \sum_{i=1}^n y_i^2} \\ = \sqrt{\max_i |\lambda_i^2|} \\ = \max_i |\lambda_i|\end{aligned}$$

Assume $\lambda_k = \max_i \lambda_i$, then the "=" is obtained when $y = (0, 0...0, \underbrace{1}_{k^{th} element}, 0..., 0)^T$ (i.e. only k^{th} element of y is 1 while other elements are 0).
Thus, $\|A\|_2$ = largest of absolute values of eigenvalues of A for symmetric A.

3 Q3

3.1 a

(1)For a rectangular $n \times p$ matrix X, we have SVD of X as $X = U\Sigma V^T = U \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} V^T$ where V composed of right singular vectors and λ_i's are singular values of X .
Since

$$X^T X = (V\Sigma U^T)U\Sigma V^T = V\Sigma^2 V^T = V \begin{bmatrix} \lambda_1^2 & & \\ & \ddots & \\ & & \lambda_n^2 \end{bmatrix} V^T$$

where V is a matrix of eigenvectors of $X^T X$ and λ_i^2's are eigenvalues of $X^T X$. Thus, the right singular vectors of X are the eigenvectors of the matrix $X^T X$. Moreover, from above we can see that the eigenvalues of $X^T X$ are the squares of singular values of X.

(2)

$$X^T X = V\Sigma^2 V^T = V \begin{bmatrix} \lambda_1^2 & & \\ & \ddots & \\ & & \lambda_n^2 \end{bmatrix} V^T$$

Since the eigenvalues λ_i^2's are non-negative, by property of positive semi-definite matrices, we can see that X^TX is positive semi-definite.

3.2 b

Suppose we have eigendecomposition $\Sigma = UAU^T$, where U is a orthonormal matrix and $A = diag(A_1, ..., A_n)$ is a diagonal matrix of eigenvalues. Since $U(cI)U^T = c(UIU^T) = c(UU^T) = cI$, we have

$$Z = \Sigma + cI = UAU^T + U(cI)U^T = U(A+cI)U^T = U \begin{bmatrix} A_1 + c & & \\ & \ddots & \\ & & A_n + c \end{bmatrix} U^T$$

Thus, eigenvalues of Z are $(A_1 + c, ..., A_n + c)$, for which is n additions in total. In this way, we compute $O(n)$ arithmetic calculations.

4 Q4

Collaborate With Ming Qiu

4.1 a

We can first consider the QR decomposition $X = QR$, then we will get $X^TX = R^TR$. Moreover, assume $AR^{-1} = Q_1R_1$ and will have the following induction:

$$\begin{aligned}
\hat{\beta} &= (X^TX)^{-1}X^TY + (X^TX)^{-1}A^T(A(X^TX)^{-1}A^T)^{-1}(-A(X^TX)^{-1}X^TY + b) \\
&= (R^TR)^{-1}R^TQ^TY + (R^TR)^{-1}A^T(A(R^TR)^{-1}A^T)^{-1}(-A(R^TR)^{-1}R^TQ^TY + b) \\
&= (R^TR)^{-1}R^TQ^TY + (R^TR)^{-1}A^T(AR^{-1}(AR^{-1})^T)^{-1}(-AR^{-1}Q^TY + b) \\
&= (R^TR)^{-1}R^TQ^TY + (R^TR)^{-1}A^TR_1^{-1}(R_1^T)^{-1}(-AR^{-1}Q^TY + b) \\
&= (R^TR)^{-1}[R^TQ^TY + A^TR_1^{-1}(R_1^T)^{-1}(-AR^{-1}Q^TY + b)] \\
&= R^{-1}(R^T)^{-1}[R^TQ^TY + A^TR_1^{-1}(R_1^T)^{-1}(-AR^{-1}Q^TY + b)]
\end{aligned}$$

For each term in the above calculation with inverse, we use backsolve() to get the solutions efficiently.
e.g. get $R^{-1}Q^TY$ using backsolve(R, Q^TY) and get $(R_1^T)^{-1}(-AR^{-1}Q^TY + b)$ using backsolve(R_1, $-AR^{-1}Q^TY + b$, transpose=T).

4.2 b

In the following code, I implement the above algorithm and showed that it will be more efficient to use QR decomposition and backsolve() rather than simply using solve() function.

```r
get_betahat <- function(A,X,Y,b){
  R <- qr.R(qr(X))
  Q <- qr.Q(qr(X))
  R1 <- qr.R(qr(t(A%*%solve(R))))
  backsolve(R, backsolve(R, t(X)%*%Y+t(A) %*% backsolve(R1,
    backsolve(R1,-A%*%backsolve(R, t(Q)%*%Y)+b, transpose = T )),transpose = T))
}


#Generate random matrix to make some tests
m <- 100
n<- 2000
p <- 2000
set.seed(1)
A <- matrix(rnorm(m*p), nrow = m)
set.seed(1)
X <- matrix(rnorm(n*p), nrow = n)
set.seed(1)
Y <- rnorm(n)
set.seed(1)
b <- rnorm(m)
d <- t(X)%*%Y

system.time(C <- crossprod(X))

##    user  system elapsed
##    4.17    0.04    4.30

system.time(solve(C)%*%d +
  solve(C)%*%t(A)%*%solve(A%*%solve(C)%*%t(A))%*%(-A%*%(solve(C)%*%d)+b))

##    user  system elapsed
##   32.24    0.07   32.72

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

##    user  system elapsed
##   20.14    0.11   20.37
```

From above, we can see that the sum of top 2 elapsed time is much greater than the elapsed time of the third one, which proves my claim.

5 Q5

5.1 a

The reason is that we can't even do the first stage of calculating $\hat{X}$ since although we can calculate $(Z^TZ)^{-1}$ beccause Z is sparse and (Z^TZ) is only 630×630, we can't calculate $Z(Z^TZ)^{-1}$ since the result will not be sparse with a extremely huge dimension of 60 million $\times$ 630. When calculating a 60 million$\times$630 matrix, it will arise a lack of memory and even though memory is enough, storing such a huge matrix will be another serious issue.
Thus, we can't calculate $\hat{X}$ in this case and can't do the calculation in two stages.

5.2 b

Since $\hat{X} = Z(Z^TZ)^{-1}Z^TX$ and $\hat{\beta} = (\hat{X}^T\hat{X})^{-1}\hat{X}^Ty$, we can have the following calculation:

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

Since $Z_{(60\text{ million}\times 630)}$ and $X_{(60\text{ million}\times 600)}$ and $y_{(60\text{ million}\times 1)}$, we can see that the dimensions of pairs in () are $(X^TZ)_{600\times 630}, (Z^TZ)_{630\times 630}, (Z^TX)_{630\times 600}$ and $(Z^Ty)_{630\times 1}$. In this way, we can acctually calculate the values inside the () first(and store them) and then multiply those outputs together(in this case it's really computational non-intensive), which will lead to a applicable solution of $\hat{\beta}$.

6 Q6

In this question, I define error in the estimated eigenvalues relative to the known true values as error=$\|V_1 - V_2\|_2$, where V_1= sorted vector of true eigenvalues and V_2 = sorted vector of estimated eigenvalues.

```r
set.seed(2)
z <- matrix(rnorm(100*100), nrow = 100)
eigen_vecs <- eigen(crossprod(z))$vectors

#generate a matrix of eigenvalues that are all the same
lambda0 <- diag(x= rep(1.5, 100), nrow=100, ncol=100)

#numerical estimate of above matrix
numerical_lambda0 <- eigen(eigen_vecs %*% lambda0 %*% t(eigen_vecs))$values
```

```
numerical_lambda0 - diag(lambda0)

##   [1]  1.150191e-13  4.130030e-14  3.996803e-14  2.198242e-14  1.776357e-14
##   [6]  1.421085e-14  1.332268e-14  1.310063e-14  1.132427e-14  1.065814e-14
##  [11]  9.769963e-15  9.103829e-15  7.993606e-15  7.327472e-15  6.661338e-15
##  [16]  6.439294e-15  5.995204e-15  5.551115e-15  5.551115e-15  5.107026e-15
##  [21]  4.662937e-15  4.662937e-15  4.440892e-15  4.440892e-15  4.218847e-15
##  [26]  3.996803e-15  3.774758e-15  3.774758e-15  3.552714e-15  3.330669e-15
##  [31]  2.886580e-15  2.886580e-15  2.442491e-15  2.442491e-15  2.442491e-15
##  [36]  2.220446e-15  2.220446e-15  1.998401e-15  1.998401e-15  1.776357e-15
##  [41]  1.554312e-15  1.554312e-15  1.332268e-15  1.332268e-15  1.110223e-15
##  [46]  6.661338e-16  6.661338e-16  4.440892e-16  4.440892e-16  2.220446e-16
##  [51]  0.000000e+00 -2.220446e-16 -2.220446e-16 -2.220446e-16 -4.440892e-16
##  [56] -6.661338e-16 -8.881784e-16 -1.110223e-15 -1.110223e-15 -1.332268e-15
##  [61] -1.332268e-15 -1.332268e-15 -1.554312e-15 -1.554312e-15 -1.776357e-15
##  [66] -1.776357e-15 -1.998401e-15 -2.220446e-15 -2.442491e-15 -2.664535e-15
##  [71] -3.108624e-15 -3.108624e-15 -3.552714e-15 -3.774758e-15 -3.774758e-15
##  [76] -3.996803e-15 -4.218847e-15 -4.218847e-15 -4.662937e-15 -4.884981e-15
##  [81] -5.329071e-15 -5.329071e-15 -5.551115e-15 -6.217249e-15 -6.439294e-15
##  [86] -7.105427e-15 -7.327472e-15 -8.881784e-15 -9.547918e-15 -9.769963e-15
##  [91] -1.021405e-14 -1.154632e-14 -1.287859e-14 -1.332268e-14 -1.554312e-14
##  [96] -1.820766e-14 -2.109424e-14 -3.907985e-14 -4.041212e-14 -1.130207e-13

numerical_lambda0

##   [1] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
##  [18] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
##  [35] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
##  [52] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
##  [69] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5
##  [86] 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5

#generate matrices of eigenvalues vary between range of values from
#very large to very small, and find the magnitude of condition number
#that the matrix is not numerically PSD
i=1
min=1
cond_num= c()
error=c()
while (min >0){
  set.seed(2)
  # increase the magnitude of eigenvalues and generate a random vector
  # of eigenvalues for each magnitude
  eigen_vals = c(10^i, sort(runif(98, min =1e-4, max =10^i),
          decreasing= TRUE), 1e-4)
  lambda1 <- diag(x = eigen_vals, nrow=100,ncol=100)
  num_lbda1 <- eigen(eigen_vecs %*% lambda1 %*% t(eigen_vecs))$values
  num_lbda1 <- sort(num_lbda1, decreasing = TRUE)

  error <- c(error, sqrt(sum((eigen_vals - num_lbda1)^2)) )
  cond_num<- c(cond_num, max(num_lbda1)/min(num_lbda1))
  min= min(num_lbda1)
  i= i+1
}

# Print Error Values
error

##  [1] 3.870296e-14 4.163431e-13 3.775483e-12 3.424479e-11 4.426547e-10
##  [6] 3.834080e-09 3.826148e-08 3.851280e-07 3.783608e-06 3.522364e-05
## [11] 3.761453e-04 3.598560e-03 3.681613e-02 3.495847e-01 2.866103e+00
## [16] 3.070514e+01 3.113717e+02 2.907646e+03

# Print Condition Numbers
cond_num

##  [1]  1.000000e+05  1.000000e+06  1.000000e+07  1.000000e+08  9.999997e+08
##  [6]  9.999947e+09  9.999644e+10  9.996851e+11  1.002223e+13  9.892226e+13
## [11]  7.281778e+14  4.096000e+15  2.560000e+15  2.124985e+15  1.472002e+16
## [16]  1.034188e+16  5.549190e+15 -9.951082e+15

# emperical conditional number for the matrix that is not numerically psd
abs(cond_num[i-1])

## [1] 9.951082e+15

# True condition number
10^(i+4-1)

## [1] 1e+22
```

When emperical condition number is at 9.951082e+15 magnitude and true condition number is at 1e+22 magnitude, the matrix is not numerically positive semi-definite.

Moreover, from above values of errors and condition numbers, we can see that (1)the error approximately increases in 10^1 magnitude as the eigenvalues increase in 10^1 magnitude, and (2) the error increases as condition number increases until condition number become negative(at which point the matrix is no more numerically positive semi-definite).