Stat243: Problem Set 7

Qingan Zhao
SID: 3033030808

17 Nov. 2017

Problem 1

I think we could first compute the standard deviation of the 1000 estimated coefficient, and then compute the mean of the 1000 estimated standard error. If **the standard deviation of the 1000 estimated coefficient is close enough to the mean of the 1000 estimated standard error**, then we could determine that the standard error properly characterizes the uncertainty of the estimated coefficient.

Problem 2

$$\begin{aligned}
||A||_2 &= \sup_{z:||z||_2=1} \sqrt{(Az)^T Az} \\
&= sup\sqrt{(\Gamma\Lambda\Gamma^T z)^T(\Gamma\Lambda\Gamma^T z)} \\
&= sup\sqrt{(z^T\Gamma\Lambda\Gamma^T)(\Gamma\Lambda\Gamma^T z)} \\
&= sup\sqrt{z^T\Gamma\Lambda^2\Gamma^T z}
\end{aligned}$$

Set $y = \Gamma^T z$:

$$\begin{aligned}
||A||_2 &= sup\sqrt{z^T\Gamma\Lambda^2\Gamma^T z} \\
&= sup\sqrt{y^T\Lambda^2 y} \\
&= sup\sqrt{\Sigma y_i^2 \times \lambda_i^2}
\end{aligned}$$

Since Γ is orthogonal, $\Gamma^T\Gamma = I$, hence:

$$\begin{aligned}
||y||_2 &= sup\sqrt{(\Gamma z)^T(\Gamma^T z)} \\
&= sup\sqrt{z^T\Gamma\Gamma^T z} \\
&= sup\sqrt{z^T z} \\
&= ||z||_2 \\
&= 1
\end{aligned}$$

Known that $||y||_2 = 1$, the sum is just $\lambda_j^2 \times 1 + \Sigma_{i\neq j} \times 0$ for the jth eigenvalue of A. Hence, the maximum of the sum is the largest of the squared eigenvalues, which means that $||A||_2$ is the largest of the absolute values of the eigenvalues A as we take the square root of the sum.

Problem 3

(a)

The singular value decomposition (SVD) of X is:

$$X = UDV^T$$

where U, V are orthogonal matrices and D is a diagonal matrix. Hence, X^TX can be expressed as:

$$\begin{aligned} X^TX &= (UDV^T)^T(UDV^T) \\ &= VD^TU^TUDV^T \\ &= VD^TDV^T \end{aligned}$$

Since D is a diagonal matrix, D^TD is also a diagonal matrix with diagonal elements d_{ii}^2 (where d_{ii} are diagonal elements of D). So the above equation is just the eigen decomposition of X^TX with eigen values d_{ii}^2:

$$(D^TD)V_i = d_{ii}^2 V_i$$

Hence, the right singular vectors of X (which is V_i) are the eigenvectors of X^TX, and the eigenvalues of X^TX are the squares of the singular values of X. Given that $d_{ii}^2 \geq 0$, all eigenvalues are non-negative, which means X^TX is positive semi-definite.

(b)

Denote the eigenvalues of a matrix by $\lambda()$, and we have already known $\lambda(\Sigma)$. Then $\lambda(Z)$ is:

$$\begin{aligned} \lambda(Z) &= \lambda(\Sigma + cI) \\ &= \lambda(\Sigma) + c\lambda(I) \\ &= \lambda(\Sigma) + c \end{aligned}$$

Since Σ has exactly n eigenvalues, adding c to each eigenvalue would require exactly n calculations (which is $O(n)$).

Problem 4

(a)

The QR decomposition of X is:

$$X = QR$$

where Q is an orthogonal matrix and R is an upper triangular matrix.

Then C can be expressed as:

$$C = X^TX = (QR)^TQR = R^TQ^TQR = R^TR$$

$C^{-1}d$ can be expressed as:

$$C^{-1}d = (R^TR)^{-1}(QR)^TY = R^{-1}(R^T)^{-1}R^TQ^TY = R^{-1}(Q^TY)$$

$AC^{-1}A^T$ can be expressed as:

$$AC^{-1}A^T = AR^{-1}(R^T)^{-1}A^T = (AR^{-1})(AR^{-1})^T$$

Hence, $\hat{\beta}$ can be implemented as follows:

```
1. Compute the QR decomposition of 'X' (denote as 'Q' and 'R')
2. Compute 'C', 'C inverse times d', and 'A times R inverse' using 'R' as above.
3. Compute 'A times C inverse times A transpose' using 'A times R inverse' in step 2.
4. Compute 'beta hat' (the second part of the 'beta hat' can be computed using solve()
                       and crossprod() from right to left).
```

(b)

```
#this function computes beta hat using the method presented in Problem 4a
beta_hat <- function(X, Y, A, b){
  X_QR <- qr(X)
  Q <- qr.Q(X_QR)
  R <- qr.R(X_QR)
  C <- crossprod(R, R)
  Cinv_d <- backsolve(R, crossprod(Q, Y))
  A_Rinv <- A %*% solve(R)
  A_Cinv_At <- tcrossprod(A_Rinv, A_Rinv)
  beta <- Cinv_d + solve(C, crossprod(A, solve(A_Cinv_At, -A %*% Cinv_d + b)))
  return(beta)
}

#run a example and compare the time with the naive approach
set.seed(123)
X <- matrix(rnorm(1e6), ncol=1000)
Y <- matrix(rnorm(1e6), ncol=1000)
A <- matrix(rnorm(1e6), ncol=1000)
b <- matrix(rnorm(1e6), ncol=1000)
system.time(beta_1 <- beta_hat(X, Y, A, b))

##    user  system elapsed
##  18.415   0.167  18.783

system.time(beta_2 <- solve(t(X) %*% X) %*% (t(X)%*%Y) + solve(t(X) %*% X) %*%
              t(A) %*% solve(A %*% solve(t(X) %*% X) %*% t(A)) %*%
              (-A %*% solve(t(X) %*% X) %*% (t(X) %*% Y)+b))

##    user  system elapsed
##  23.369   0.379  24.535

#check if we got the correct answer
all.equal(beta_1, beta_2, tolerance=1e-5)

## [1] TRUE
```

From the result we can see that the presented method is more efficient than computing $\hat{\beta}$ naively.

Problem 5

(a)

The first stage requires computing $Z(Z^TZ)^{-1}Z^TX$ which is a 60 million $\times$ 60 million matrix. Since it may not be a sparse matrix even if Z is sparse, the computer would run out of memory when storing it. $\hat{X}$ is also too large (60 million $\times$ 600) for ordinary PC to store. Same problems would occur in the second stage.

(b)

First, let's just express $\hat{\beta}$ without using $\hat{X}$:

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

Now let's add some parentheses to make $\hat{\beta}$ computed on a computer without using a large amount of memory:

$$\hat{\beta} = [(X^TZ)(Z^TZ)^{-1}(Z^TX)]^{-1}(X^TZ)(Z^TZ)^{-1}(Z^Ty)$$

Follow the step using the above equation, and $\hat{\beta}$ can be computed given that multiplications of sparse matrices can be done on the computer using the *spam* package in R. Now let's prove it:

First compute X^TZ, $(Z^TZ)^{-1}$, Z^TX, and Z^Ty. Their sizes are 600×630, 630×630, 630×630, and 630×1 which do not cost much memory.

Second compute $[(X^TZ)(Z^TZ)^{-1}(Z^TX)]^{-1}$ of 600×600 size which also does not cost much memory.

Finally $\hat{\beta}$ (of 600×1 size) can be computed by doing the multiplications of the above results.

Problem 6

In this problem, we first generate matrices A and Γ using random generalized Z. Then create eigenvalues with different magnitudes and vary between all being equal and having a range of values from large to small. Use these created eigenvalues and Gamma to create $\Gamma\Lambda\Gamma^T$ (new A). Compute the eigenvalues, condition numbers and errors of the computed eigenvalues (from the original eigenvalues) of the new generated $\Gamma\Lambda\Gamma^T$. Finally put the results into a data frame and plot some figures to visualize the results.

```
#generate A and Gamma
set.seed(123)
n <- 100
Z <- matrix(rnorm(n^2), n, n)
A <- crossprod(Z)
Gamma <- cbind(eigen(A)$vectors)

#create eigenvalues with different magnitudes and vary between equal & having a range
#magnitude from 1 to 1e12 (intervel is 100)
#so the number of sets of eigenvalues is 7*2=14
num_sets <- 14
eigs_actual <- matrix(0, num_sets, n)
eigs_compute <- matrix(0, num_sets, n)
A_create <- array(0, c(num_sets, n, n))
magnitude <- rep(0, num_sets)
condition_num <- rep(0, num_sets)
```

```r
error <- rep(0, num_sets)
pos_definite <- rep(NA, num_sets)

for (i in seq(1, 13, 2)){
  eigs_actual[i, ] <- rep(10^((i-1)), n)
  magnitude[i] <- 10^((i-1))
}

for (i in seq(2, 14, 2)){
  eigs_actual[i, ] <- seq(10^(-(i-2)), 10^((i-2)), length.out = n)
  magnitude[i] <- 10^((i-2))
}

#employ eigen decomposition & compute the following items
for (i in 1:num_sets){
  A_create[i, , ] <- Gamma %*% diag(eigs_actual[i, ]) %*% solve(Gamma)
  eigs_compute[i, ] <- eigen(A_create[i, , ])$values
  condition_num[i] <- abs(max(eigs_actual[i, ]) / min(eigs_actual[i, ]))
  pos_definite[i] <- all(eigs_compute[i, ]>0)
  error[i] <- sum((eigs_compute[i, ] - eigs_actual[i, ])^2)
}

#create a data frame for the results
data_frame <- data.frame(Magnitude = magnitude, Condition_number = condition_num,
                         Error = error, Positive_definite = pos_definite)
data_frame

##    Magnitude Condition_number        Error Positive_definite
## 1      1e+00            1e+00 7.504039e-28              TRUE
## 2      1e+00            1e+00 7.504039e-28              TRUE
## 3      1e+02            1e+00 2.629570e-24              TRUE
## 4      1e+02            1e+04 3.399993e+05              TRUE
## 5      1e+04            1e+00 3.153543e-20              TRUE
## 6      1e+04            1e+08 3.400673e+09              TRUE
## 7      1e+06            1e+00 3.056501e-16              TRUE
## 8      1e+06            1e+12 3.400673e+13              TRUE
## 9      1e+08            1e+00 2.638334e-12              TRUE
## 10     1e+08            1e+16 3.400673e+17              TRUE
## 11     1e+10            1e+00 2.567685e-08              TRUE
## 12     1e+10            1e+20 3.400673e+21              TRUE
## 13     1e+12            1e+00 2.507865e-04              TRUE
## 14     1e+12            1e+24 3.400673e+25             FALSE
```

```r
#plot magnitude vs error when all eigenvalues are equal
ggplot(data_frame[seq(1, num_sets-1, 2), ], aes(x=Magnitude)) + geom_line(aes(y=Error))
```

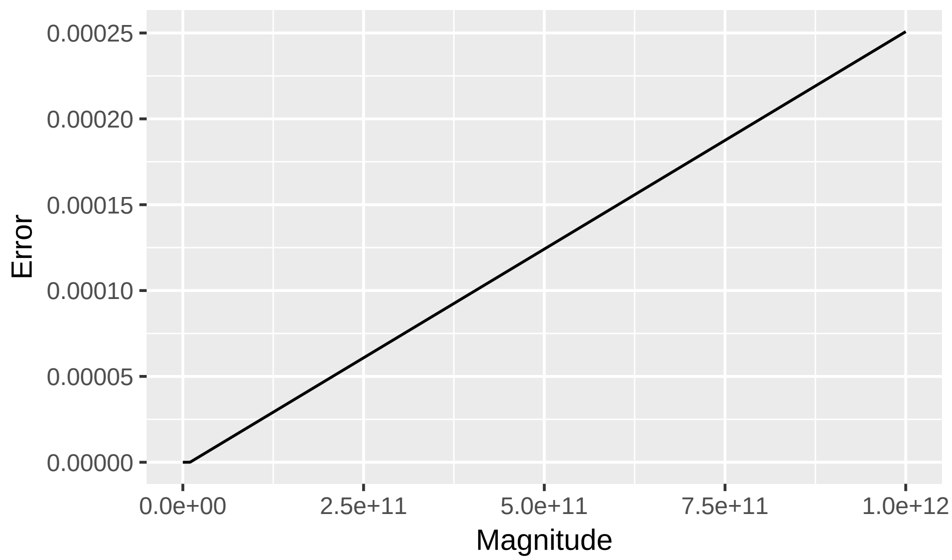

Fig.1: Magnitude vs error when all eigenvalues are equal

```
#plot magnitude vs error when all eigenvalues having a range from large to small
ggplot(data_frame[seq(2, num_sets, 2), ], aes(x=Magnitude)) + geom_line(aes(y=Error))
```

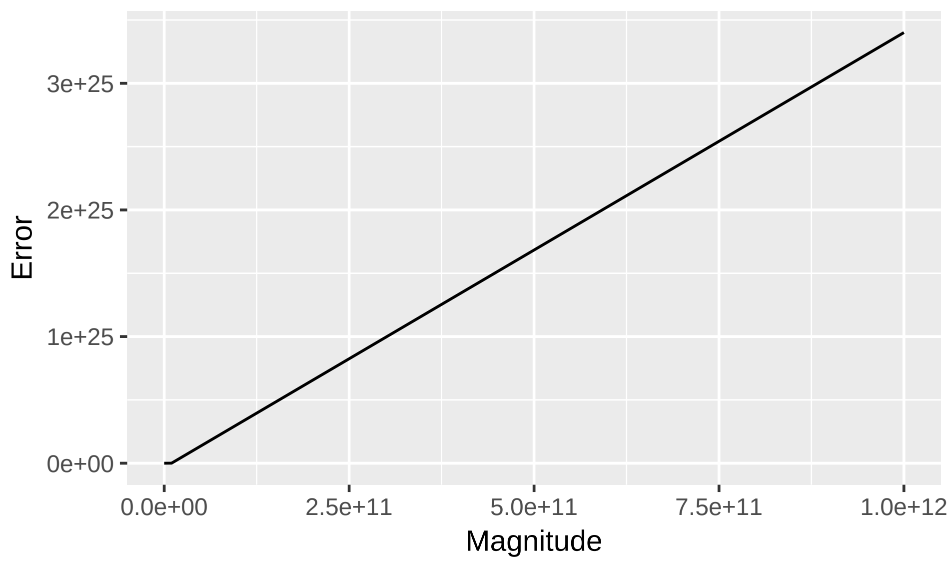

Fig.2: Magnitude vs error when all eigenvalues having a range from large to small

```
#since condition numbers are the same when all eigenvalues are equal,
#hence we do not plot condition number vs error under this condtion

#plot condition number vs error when all eigenvalues having a range from large to small
ggplot(data_frame[seq(2, num_sets, 2), ], aes(x=Condition_number)) + geom_line(aes(y=Error))
```

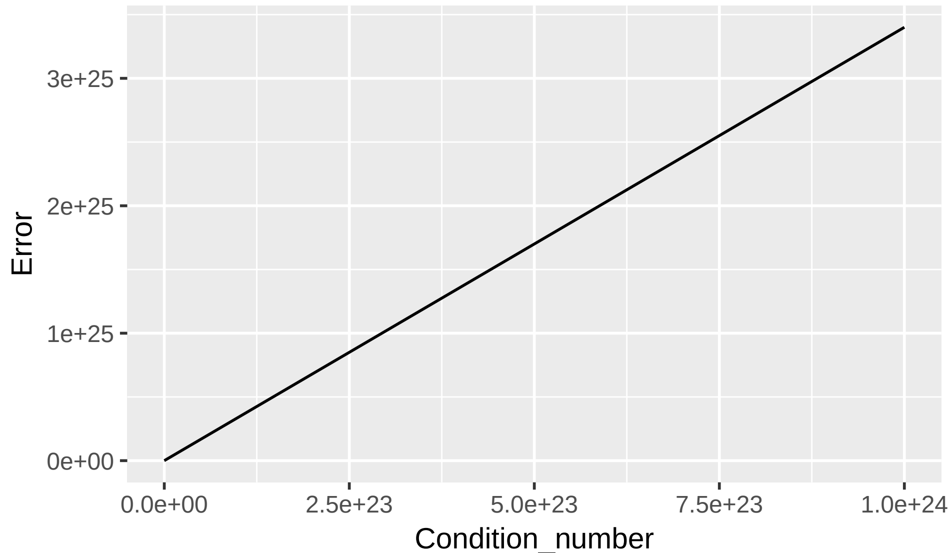

Fig.3: Condition number vs error

From the data frame we can see that our matrix is not numerically positive definite when the condition number is 10^{20}. From Figure 1-3 we can see that the error goes up either with the condition number increases or with the magnitude of the eigenvalues increases.