Problem Set 7

Dongyu Lang
SID: 24174288

Nov, 15, 2017

Question 1

We can get the standard deviation of the 1000 estimated coefficients from the simulated datasets, and construct a 95% confidence interval. Then, we can check if the standard error of each coefficient lies in the confidence interval to determine the quality of our estimates.

Question 2

We know that A is a symmetric matrix; thus, we can factorize A into orthogonalΓ and diagonal Λ, that is $A = \Gamma\Lambda\Gamma^T$, where the diagonal values are all eigenvalues of A. Thus,
$(AZ)^T(AZ) = Z^TA^TAZ = Z^T(\Gamma\Lambda\Gamma^T)^T(\Gamma\Lambda\Gamma^T)Z = Z^T\Gamma\Lambda^T\Gamma^T\Gamma\Lambda\Gamma^TZ = Z^T\Gamma\Lambda^T\Lambda\Gamma^TZ = (\Gamma^TZ)^T\Lambda^T\Lambda(\Gamma^TZ)$
We know that $(\Gamma^TZ)^T(\Gamma^TZ) = Z^T\Gamma\Gamma^TZ = Z^TI_nZ = 1$, and let $y = \Gamma^TZ$
Let us express $\sqrt{y^T\Lambda^T\Lambda y}$ as a sum, that is $\sqrt{\Gamma_1^2y_1^2 + \Gamma_2^2y_2^2 + \ldots + \Gamma_n^2y_n^2}$, in addition, $\|y\|_2 = 1$; thus, in order to maximize the function, we should let the y_i, which corresponds to the one of the largest abosulte values among the eigenvaues, to be 1. That is, $\|A\|_2$ is the largest of the absolute values of the eigenvalues of A for symmetric A.

Question 3(a)

X can be decomposed as $X = UDV^T$, where U and V are matrices with orthonormal columns and D is diagonal with non-negative values. Thus,
$X^TX = (UDV^T)^T(UDV^T) = VD^TU^TUDV^T = VD^TDV^T = VD^2V^T$
In addition, $(X^TX)^T = X^T(X^T)^T = X^TX$; thus, X^TX is a symmetric square matrix, and can decomposite into an orthogonal matrix V and a diagonal matrix D^2, where the eigenvalues on the diagonal of D^2 are ordered in decreasing value.
Thus, we know that the squares of the sigular values of X are the eigenvalues of X^TX, That is the diagonals of D^2 are the eigenvalues of X^TX.

Thus, $X^TXV = VD^2V^T = VD^2$, **which means that the right sigular vectors of X are the eigenvectors of the matrix X^TX.**
Moreover, it is easy to show that X^TX is positive semidefinite, because, $a^TX^TXa = (Xa)^T(Xa) = \|Xa\|_2^2 \geq 0$. **$X^TX$ is positive semidefinite.**

Question 3(b)

Suppose we knew that $\Sigma = \Gamma\Lambda\Gamma^T$, and $Z = \Sigma + cI$.
Thus, $\Gamma^TZ\Gamma = \Gamma^T(\Sigma + cI)\Gamma = \Gamma^T(\Gamma\Lambda\Gamma^T + cI)\Gamma = \Lambda + c\Gamma^TI\Gamma = \Lambda + cI$
This leads to that $Z = \Gamma(\Lambda + cI)\Gamma^T$, and the diagonal matrix with eigenvalues is $\Lambda + cI$, that is we add a "c" to each of the eigenvalues of Σ, which takes O(n) calculations.

Question 4(a)

We can perform a QR decomposition on X, that is $X = QR$, with Q orthogonal and R upper triangular.
Thus, $X^TX = R^TQ^TQR = R^TR$ and $C = (R^TR)^{-1}$
And $(AC^{-1}A^T)^{-1} = (A(R^TR)^{-1}A^T)^{-1} = (AR^{-1}(R^T)^{-1}A^T)^{-1}$
Now let $AR^{-1} = \hat{Q}\hat{R}$.
$(AC^{-1}A^T)^{-1} = (\hat{R}\hat{R}^T)^{-1}$
Thus, $\hat{\beta} = (R^TR)^{-1}d + (R^TR)^{-1}A^T(\hat{R}\hat{R}^T)^{-1}(-A(R^TR)^{-1}d + b)$

```
C = crossprod(R,R)
d = crossprod(X,Y)
R_hat_square = t(crossprod(R_hat,R_hat))
beta_hat = backsolve(C, d + crossprod(A, (backsolve(R_hat_square,
          (-A %&% backsolve(C,d) + b)))))
```

Question 4(b)

```
solvebeta = function(X, Y, A, b){
  R = qr.R(qr(X))
  R_hat = qr.R(qr(t(A %*% solve(R))))
  C = crossprod(R,R)
  d = crossprod(X,Y)
  R_hat_square = t(crossprod(R_hat,R_hat))
  beta_hat = backsolve(C, d + crossprod(A, (backsolve(R_hat_square,
             (-A %*% backsolve(C,d) + b)))))
  return (beta_hat)
}
```

```
A = matrix(rnorm(100*50), nrow = 100)
b = rnorm(100)
X = matrix(rnorm(200*50), nrow = 200)
Y = rnorm(200)
## Run time
system.time(beta <- solvebeta(X,Y,A,b))

##    user  system elapsed
##   0.005   0.001   0.008

dim(beta)

## [1] 50  1
```

Question 5(a)

It is hard to store $\hat{X}$, because the size of the matrix is large, which is 60 million by 600, and even though both Z and X are sparse, we cannot guarantee that $\hat{X}$ is sparse.

Question 5(b)

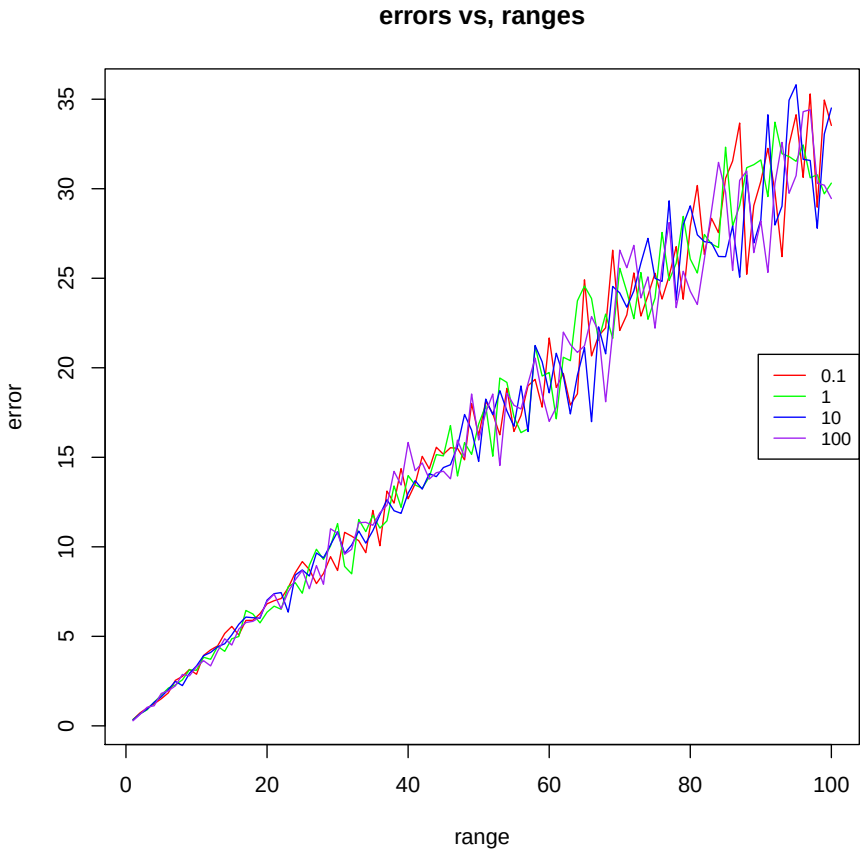
We can plug $\hat{X}$ back into $\hat{\beta}$, that is,
$(\hat{X^T})^{-1} = ((Z(Z^TZ)^{-1}Z^TX)^TZ(Z^TZ)^{-1}Z^TX)^{-1} = (X^TZ(Z^TZ)^{-1}Z^TZ(Z^TZ)^{-1}Z^TX)^{-1} = ((X^TZ)(Z^TZ)^{-1}(X^TZ)^T)^{-1}$
$X^Ty = (Z(Z^TZ)^{-1}Z^TX)^Ty = X^TZ(Z^TZ)^{-1}Z^Ty$
$\hat{\beta} = ((X^TZ)(Z^TZ)^{-1}(X^TZ)^T)^{-1}X^TZ(Z^TZ)^{-1}Z^Ty$
Thus, we only need to compute X^TZ, Z^TZ and Z^Ty, and the dimensions are 600×630, 630×630 and 630×1. And, the computation cost much less.

Question 6

```
library(matrixcalc)
set.seed(10000)
## Create the eigenvectors
Z = matrix(rnorm(100*100), nrow = 100)
A = crossprod(Z)
eigenvectors = eigen(A)$vectors

eigen_error = function(magnitude, range) {
  ## Create the eigenvalues
```

```
  values = rep(magnitude,100)
  value_range = runif(100,min = -magnitude, max = range-magnitude)
  values = values + value_range
  Lambda = diag(x = values,100,100)
  ## Construct the matrix
  create_matrix = eigenvectors %*% Lambda %*% t(eigenvectors)
  ## Get the eigenvalues
  create_eigenvalues = eigen(create_matrix)$values
  ## Get the errors between true eigenvalues and calculated ones
  error = mean(abs(values-create_eigenvalues))
  ## Check if the matrix is positive definite
  all_positive = sum((create_eigenvalues <= 0))
  pos_def = if(sum(all_positive) == 0) T else F
  return (c(error,pos_def))
}

## Create the error list with magnitude 0.1 and
## range of eigenvalues from 1 to 100
error_list_0.1 = c()
pos_def_0.1 = c()
for (i in 1:100) {
  error_list_0.1[i] = eigen_error(0.1,i)[1]
  pos_def_0.1[i] = eigen_error(0.1,i)[2]
}

## Create the error list with magnitude 1 and
## range of eigenvalues from 1 to 100
error_list_1 = c()
pos_def_1 = c()
for (i in 1:100) {
  error_list_1[i] = eigen_error(1,i)[1]
  pos_def_1[i] = eigen_error(1,i)[2]
}

## Create the error list with magnitude 10 and
## range of eigenvalues from 1 to 100
error_list_10 = c()
pos_def_10 = c()
for (i in 1:100) {
  error_list_10[i] = eigen_error(10,i)[1]
  pos_def_10[i] = eigen_error(10,i)[2]
}

## Create the error list with magnitude 100 and
## range of eigenvalues from 1 to 100
```

```
error_list_100 = c()
pos_def_100 = c()
for (i in 1:100) {
  error_list_100[i] = eigen_error(10,i)[1]
  pos_def_100[i] = eigen_error(10,i)[2]
}

## Make the plot with different magnitudes and ranges
plot(x = 1:100, y=error_list_0.1, type = "l",col = "red",
     xlab = "range", ylab = "error", main = "errors vs, ranges")
lines(x = 1:100, y=error_list_1,type = "l", col = "green")
lines(x = 1:100, y=error_list_10,type = "l", col = "blue")
lines(x = 1:100, y=error_list_100,type = "l",col = "purple")
legend("right", legend=c("0.1", "1", "10","100"),
       col=c("red", "green","blue","purple"), lty=1:1, cex=0.8)
```



From the graph, we can notice that the errors do not vary too much as the

magnitudes of eigenvalues increase. However, as the condition number increases, the error also increases.

Collaboration

In this problem set, I worked with Gaowei Chen.