STAT243: Problem Set 7

Zicheng Huang

11/17/2017

1.

From the 1000 simulated datasets, we have 1000 1000 estimates of the coefficient and 1000 standard errors of the estimates of the coefficient. In order to determine if the standard error properly characterizes the uncertainty of the estimated regression coefficient, we can compute the standard deviation of the 1000 estimates of the coefficient and see if the result is similar to the 1000 standard errors of the estimates of the coefficient we get from the simulated datasets, by comparing the standard deviation of the 1000 estimates of the coefficient to the mean of the 1000 standard errors of the estimates of the coefficient.

2.

Since A is symmetric, we can do eigendecomposition as $A = \Gamma\Lambda\Gamma^T$ where Γ is orthogonal matrix and $\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$ and λ_i's are the eigenvalues of A.

Then A^TA can be rewritten as $A^TA = (\Gamma\Lambda\Gamma^T)^T\Gamma\Lambda\Gamma^T = \Gamma\Lambda^T\Lambda\Gamma^T$ where $\Lambda^T\Lambda = \begin{bmatrix} \lambda_1^2 & & \\ & \ddots & \\ & & \lambda_n^2 \end{bmatrix}$.

Therefore, we can rewrite $||A||_2$ as follows:

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

Let $\begin{bmatrix} y_1 & \cdots & y_n \end{bmatrix}^T = y = \Gamma^Tz$, where Γ is orthogonal.
Then,

$$||y||_2 = ||\Gamma^Tz||_2 = \sqrt{z^T\Gamma\Gamma^Tz} = \sqrt{z^Tz} = ||z||_2 = 1$$

Therefore, we can further rewrite $||A||_2$ as follows:

$$\begin{aligned}
||A||_2 &= \sup_{z:||z||_2=1} \sqrt{z^T \Gamma \Lambda^T \Lambda \Gamma^T z} \\
&= \sup_{||y||_2=1} \sqrt{y^T \Lambda^T \Lambda y} \\
&= y^T \begin{bmatrix} \lambda_1^2 & & \\ & \ddots & \\ & & \lambda_n^2 \end{bmatrix} y \\
&= \sup_{||y||_2=1} \sqrt{\Sigma_{i=1}^n \lambda_i^2 y_i^2} \\
&\leq \sup_{||y||_2=1} \sqrt{\Sigma_{i=1}^n (\max_i \lambda_i)^2 y_i^2} \\
&= \sup_{||y||_2=1} \sqrt{(\max_i \lambda_i)^2 \Sigma_{i=1}^n y_i^2} \\
&= \sup_{||y||_2=1} \sqrt{(\max_i \lambda_i)^2} \\
&= \sup_{||y||_2=1} \max_i |\lambda_i|
\end{aligned}$$

In other words, we get the following expression:

$$||A||_2 \leq \sup_{||y||_2=1} \max_i |\lambda_i|$$

Suppose $\lambda_j = \max_i |\lambda_i|$, then if $y = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}^T$ where the 1 locates at the jth position, we get

$$||A||_2 = \max_i |\lambda_i|$$

Thus, as $||A||_2 = \max_i |\lambda_i|$, we have shown that $||A||_2$ is the largest of the absolute values of the eigenvalues of A.

3.

(a)

Since X is a rectangular matrix with dimension $n \times p$ and $n > p$, we can do SVD on X as follows:

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

where elements of V are the right singular vectors and λ_i's are the singular values of X.
Then we can rewrite $X^T X$ as follows:

$$X^T X = (UDV^T)^T UDV^T = VDU^T UDV^T = VD^2V^T = V \begin{bmatrix} \lambda_1^2 & & \\ & \ddots & \\ & & \lambda_n^2 \end{bmatrix} V^T$$

where elements of V are the eigenvectors of X^TX and λ_i^2's are the eigenvalues of X^TX. Thus, the right singular vectors of X are the eigenvectors of the matrix X^TX and the eigenvalues of X^TX are the squares of singular values of X.

Then we will show that X^TX is positive semi-definite.
Since we already saw that

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

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

(b)
Assume that we have already computed the eigendecomposition of Σ as $\Sigma = \Gamma\Lambda\Gamma^T$ where Γ is an orthogonal matrix and $\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$ is a diagonal matrix of the eigenvalues of Σ.
Since we can see that $cI = c(\Gamma\Gamma^T) = c(\Gamma I\Gamma^T) = \Gamma(cI)\Gamma^T$, we can rewrite Z as follows:

$$Z = \Sigma + cI = \Gamma\Lambda\Gamma^T + \Gamma(cI)\Gamma^T = \Gamma(\Lambda + cI)\Gamma^T = \Gamma \begin{bmatrix} \lambda_1 + c & & \\ & \ddots & \\ & & \lambda_n + c \end{bmatrix} \Gamma^T$$

Therefore, we can see that the eigenvalues of Z are $\{\lambda_1 + c, \cdots, \lambda_n + c\}$, which can be obtained by doing n additions in total. Thus, we compute the eigenvalues of Z in $O(n)$ arithmetic calculations.

4.
(a)
I will implement this by making use of the QR decomposition. Reason to use QR is because eigendecomposition is computational intensive and Cholesky decomposition is less stable than QR. First we need to decompose X into $X = QR$, where Q is an n by n orthogonal matrix and R is an n by p upper triangular matrix, then decompose $(AR^{-1})^T$ into $(AR^{-1})^T = qr$ where q is an n by n orthogonal matrix and r is an n by m upper triangular matrix and R is from the decomposition of X. Then we can obtain $\hat{\beta}$ with the following steps:

First we rewrite the elements in the expression for $\hat{\beta}$:

$$\begin{cases} X = QR \\ (AR^{-1})^T = qr \end{cases}$$

$$\Downarrow$$

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

$$d = X^T Y = (QR)^T Y = R^T Q^T Y$$

$$-AC^{-1}d = -A(R^T R)^{-1}(X^T Y) = -A(R^T R)^{-1}((QR)^T Y) = -AR^{-1}(R^T)^{-1}R^T Q^T Y = -AR^{-1}Q^T Y$$

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

Then we can rewrite the expression for $\hat{\beta}$:

$$\begin{aligned} \hat{\beta} &= C^{-1}d + C^{-1}A^T(AC^{-1}A^T)^{-1}(-AC^{-1}d + b) \\ &= (R^T R)^{-1}(X^T Y) + (R^T R)^{-1}A^T(r^T r)^{-1}(-AR^{-1}Q^T Y + b) \\ &= (R^T R)^{-1}[(X^T Y) + A^T(r^T r)^{-1}(-AR^{-1}Q^T Y + b)] \end{aligned}$$

Now we can solve for $\hat{\beta}$ with following steps:

1. Compute $R^{-1}Q^T Y$ by backsolve(R, t(Q) %*% Y)
2. Compute $(r^T r)^{-1}(-AR^{-1}Q^T Y + b)$ by backsolve(crossprod(r), -A %*% (result from step 1) + b)
3. Compute $\hat{\beta}$ by backsolve(crossprod(R), t(X) %*% Y + t(A) %*% (result from step 2))

(b)

Below is a R function that implement the computation steps above to obtain $\hat{\beta}$.

```
getBetaHat <- function(X, Y, A, b){
  # QR decompose X
  R <- qr.R(qr(X))
  Q <- qr.Q(qr(X))
  # QR decompose (AR^{-1})^T
  r <- qr.R(qr(t(A %*% solve(R))))
  # solve for betahat using algorithm mentioned in the pseudo-code in part (a)
  # step 3
  backsolve(crossprod(R),
                                    # step 2
            t(X) %*% Y + t(A) %*% (backsolve(crossprod(r),
                                             # step 1
                                             -A %*% (backsolve(R, t(Q) %*% Y)) + b)))
}
```

5.

(a)

Even though X and Z are sparse matrix, $\hat{X}$ cannot be obtained due to issues regarding both memory and storage. In the computation of $\hat{X}$, the calculation of $Z(Z^TZ)^{-1}Z^T$ will not necessarily be a sparse matrix. If the result of $Z(Z^TZ)^{-1}Z^T$ is dense, together with its huge dimension (60 million by 60 million), issues in calculation memory will arise and it is to computational costly. Moreover, $\hat{X}$ might be a dense matrix as well with a huge dimension of 60 million by 600 such that the storage of such matrix is problematic.

(b) We can rewrite the equation by plug in the expression for $\hat{X}$ into the equation of $\hat{\beta}$ so as to avoid obtaining $\hat{X}$ in the first place.
We first need to simplify $\hat{X}^T\hat{X}$ as follows:

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

Therefore, we can rewrite the expression for $\hat{\beta}$ as follows:

$$\begin{aligned}
\hat{\beta} &= (\hat{X}^T\hat{X})^{-1}\hat{X}^Ty \\
&= (X^TZ(Z^TZ)^{-1}Z^TX)^{-1}(Z(Z^TZ)^{-1}Z^TX)^Ty \\
&= (X^TZ(Z^TZ)^{-1}Z^TX)^{-1}(X^TZ(Z^TZ)^{-1}Z^T)y \\
&= \underbrace{\underbrace{\underbrace{\underbrace{(\underbrace{(X^TZ)}_{600\times 630}\underbrace{(Z^TZ)^{-1}}_{630\times 630}\underbrace{(Z^TX)}_{630\times 600})^{-1}}_{600\times 600}\underbrace{(X^TZ)}_{600\times 630}}_{600\times 630}\underbrace{(Z^TZ)^{-1}}_{630\times 630}}_{600\times 630}\underbrace{(Z^Ty)}_{630\times 1}}_{600\times 1}
\end{aligned}$$

Notice that X is a 60 million by 600 sparse matrix, Z is a 60 million by 630 sparse matrix, and y is 60 million by 1, then the results of (X^TZ), $(Z^TZ)^{-1}$, (Z^TX), (Z^Ty) are matrices with small dimension denoted in the expression above. Since the input to these matrix multiplications are all sparse, except for (Z^Ty) but still computable by Z being sparse and y being 60 million by 1, all of them can be calculated. Therefore, we can obtain $\hat{\beta}$ through computations involving all small matrix, without facing problems regarding memory and storage.

7.

Generate a set of eigenvectors

```
set.seed(1)
## create arbitrary Z
Z <- matrix(rnorm(100*100), nrow = 100, ncol = 100)
## Compute A as cross product of Z, then A is symmetric
A <- crossprod(Z)
## eigendecompose A to obtain its eigenvectors
eigenVectorsA <- eigen(A)$vectors
```

Now we want to compare the computed eigenvalues and the actual eigenvalues
First we start with eigenvalues all being equal

```
## create a matrix with eigenvalues that are all the same, with value 3
eigenValues0 <- diag(x = rep(3, 100), nrow = 100, ncol = 100)
## numerical calculate the computed eigenvalues
computedEigenValues0 <- eigen(eigenVectorsA %*% eigenValues0 %*% t(eigenVectorsA))$values
## difference between computed eigenvalues and actual eigenvalues
computedEigenValues0 - diag(eigenValues0)
```

```
##   [1]  5.995204e-14  5.595524e-14  5.018208e-14  3.685940e-14  3.463896e-14
##   [6]  3.419487e-14  3.197442e-14  2.620126e-14  2.264855e-14  2.087219e-14
##  [11]  2.042810e-14  1.909584e-14  1.731948e-14  1.687539e-14  1.554312e-14
##  [16]  1.376677e-14  1.199041e-14  1.199041e-14  1.110223e-14  1.065814e-14
##  [21]  1.065814e-14  9.769963e-15  9.769963e-15  9.325873e-15  8.881784e-15
##  [26]  8.437695e-15  7.993606e-15  7.549517e-15  7.549517e-15  7.105427e-15
##  [31]  7.105427e-15  6.661338e-15  6.217249e-15  5.773160e-15  5.329071e-15
##  [36]  5.329071e-15  4.440892e-15  4.440892e-15  3.996803e-15  3.552714e-15
##  [41]  3.552714e-15  3.108624e-15  2.664535e-15  2.220446e-15  1.776357e-15
##  [46]  1.776357e-15  1.332268e-15  8.881784e-16  8.881784e-16  4.440892e-16
##  [51]  4.440892e-16  0.000000e+00 -8.881784e-16 -8.881784e-16 -1.332268e-15
##  [56] -1.332268e-15 -1.776357e-15 -2.220446e-15 -2.220446e-15 -2.664535e-15
##  [61] -2.664535e-15 -3.108624e-15 -3.552714e-15 -3.996803e-15 -4.440892e-15
##  [66] -4.440892e-15 -4.884981e-15 -4.884981e-15 -5.773160e-15 -5.773160e-15
##  [71] -6.217249e-15 -6.661338e-15 -7.105427e-15 -7.549517e-15 -7.993606e-15
##  [76] -8.437695e-15 -9.769963e-15 -9.769963e-15 -1.021405e-14 -1.065814e-14
##  [81] -1.110223e-14 -1.199041e-14 -1.287859e-14 -1.332268e-14 -1.376677e-14
##  [86] -1.421085e-14 -1.509903e-14 -1.687539e-14 -1.776357e-14 -1.909584e-14
##  [91] -2.087219e-14 -2.176037e-14 -2.220446e-14 -2.531308e-14 -2.842171e-14
##  [96] -3.508305e-14 -3.552714e-14 -5.329071e-14 -5.728751e-14 -6.661338e-14
```

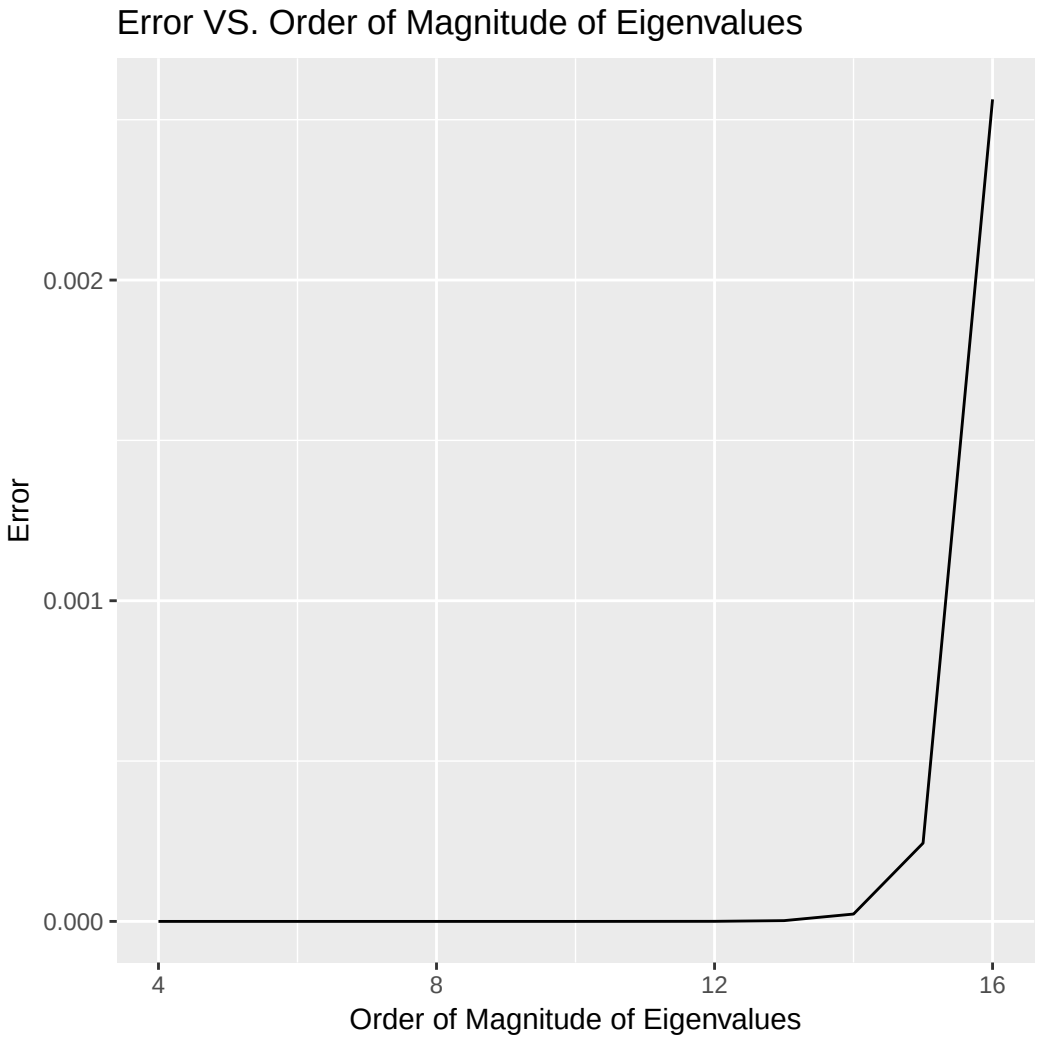
When the generated eigenvalues are all the same, we see that the difference between the computed eigenvalues and the actual eigenvalues are basically 0.

Now we consider generating eigenvalues vary from a range of values from very large to very small. Then we calculate the corresponding condition number and error between computed eigenvalues and actual eigenvalues by median of absolute difference.

```
## loop index
i <- 1
## min set to 1 to start the loop
min <- 1
## vector storing condition numbers
condNum <- c()
## vector storing errors
error <- c()
## use the eigen vectors obtained from matrix A
eigenVectors <- eigenVectorsA
## the loop will break when then min of the computed eigenvalues is less than 0,
## this means that we get a matrix that is not numerically positive definite
while (min > 0){
  set.seed(666)
  ## generate eigen values with different magnitudes
  eigenValues <- diag(x = sort(seq(from = 0.001, to = (10^i), by = ((10^i)-0.001)/99), decreasing = T),
                      nrow = 100, ncol = 100)
  ## numerically calculate the computed eigen values
  computedEigenValues <- eigen(eigenVectors %*% eigenValues %*% t(eigenVectors))$values
  ## compute the condition number
  condNum <- c(condNum, max(computedEigenValues)/min(computedEigenValues))
  ## compute the error by median of the absolute difference
  error <- c(error, median(abs(computedEigenValues - diag(eigenValues))))
  ## set min equals the minimum of the computed eigenvalues
  min = min(computedEigenValues)
  i = i + 1
}

## at this condition number we empirically see that the matrix is not numerically positive definite
abs(condNum[i-1])

## [1] 2.56e+15

## the value of tis condition number without absolute is negative, shows that the min of eigenvlaues is
condNum[i-1]

## [1] -2.56e+15
```

At the above condition numer, we empirically see that the matrix is not numerically positive definite since the loop breaks at this point. The negative value above also shows this observation.

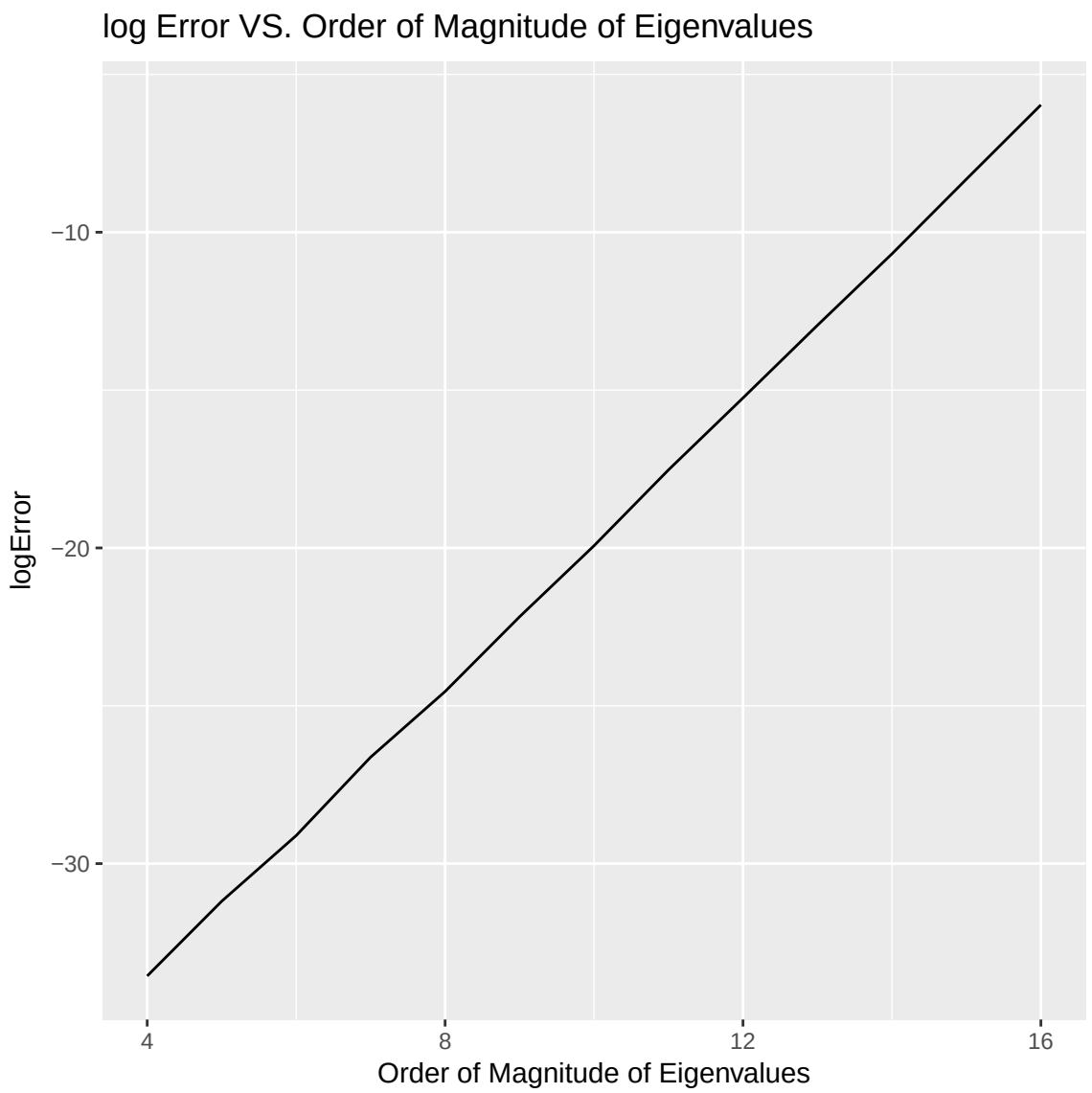
```r
library(ggplot2)
## plot to see the relationship between the error and the condition number
## Order measures the order of magnitude, starting at 4, Error measures the median
## of absolute difference logError measures the log of Error
df <- data.frame(Order = 4:(i-1+3), Error = error, logError = log(error),
                 CondNum = abs(condNum[1:i-1]))
# relationship between error and order of magitude
g1 <- ggplot(df) +
  geom_line(aes(x = Order, y = Error)) +
  ggtitle("Error VS. Order of Magnitude of Eigenvalues") +
  xlab("Order of Magnitude of Eigenvalues")
g1
```



We can see that there is a positive relationship between errors and the order of magnitude.

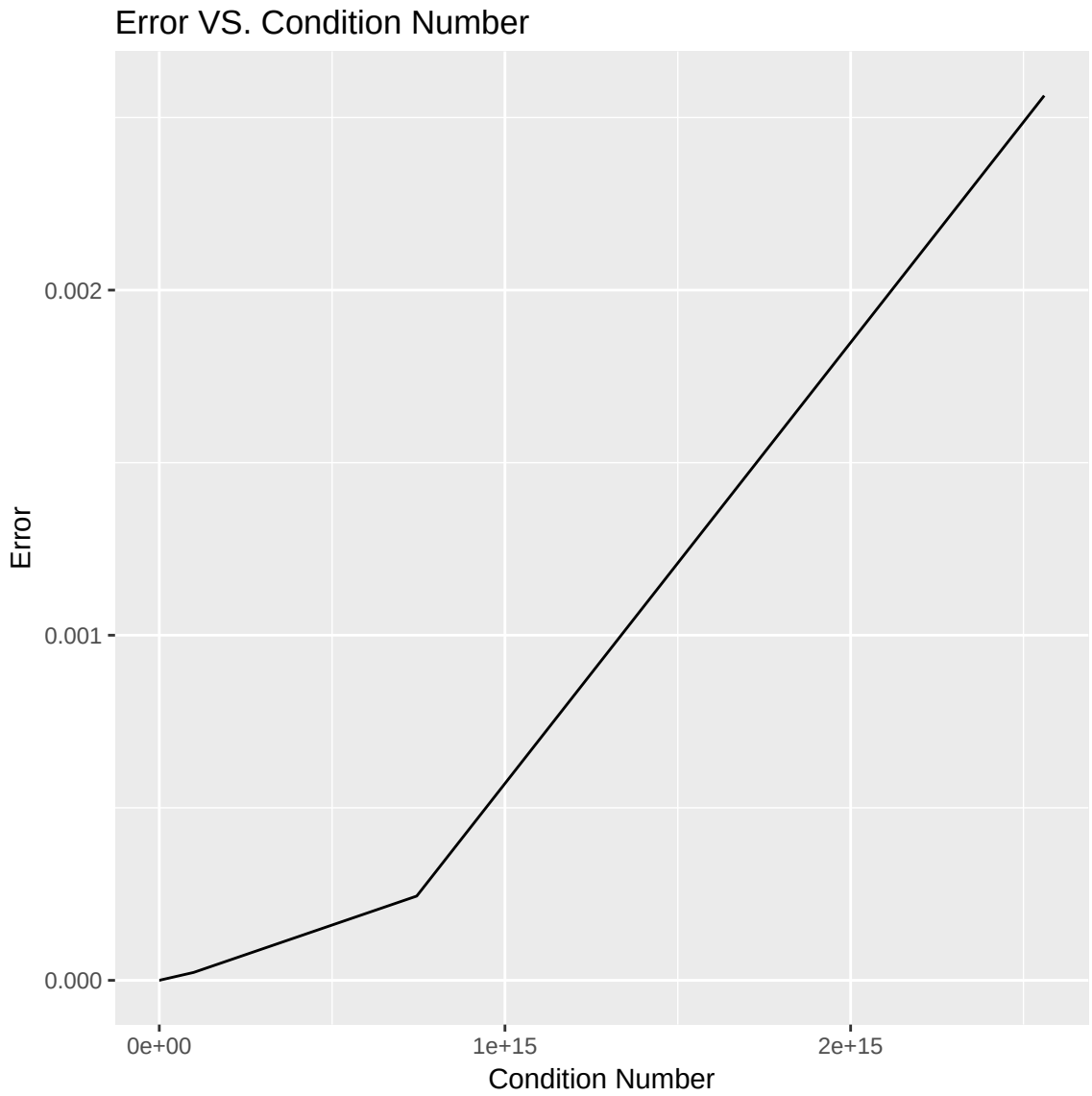
To have a better idea, we plot the log error with respect to the order of magnitudes.

```
# relationship between log error and order of magitude
g2 <- ggplot(df) +
  geom_line(aes(x = Order, y = logError)) +
  ggtitle("log Error VS. Order of Magnitude of Eigenvalues") +
  xlab("Order of Magnitude of Eigenvalues")
g2
```



We can see that there is a positive relationship between log errors and the order of magnitude.

```r
# relationship between error and condition number
g3 <- ggplot(df) +
  geom_line(aes(x = CondNum, y = Error)) +
  ggtitle("Error VS. Condition Number") +
  xlab("Condition Number")
g3
```



We can see that there is a positive relationship between errors and the condition numbers.

This is the dataframe for the plots.

```
df

##    Order        Error   logError      CondNum
## 1      4 2.664535e-15 -33.558747 1.000000e+04
## 2      5 2.842171e-14 -31.191623 1.000000e+05
## 3      6 2.273737e-13 -29.112182 1.000000e+06
## 4      7 2.728484e-12 -26.627275 1.000000e+07
## 5      8 2.182787e-11 -24.547833 1.000000e+08
## 6      9 2.328306e-10 -22.180710 9.999998e+08
## 7     10 2.211891e-09 -19.929418 9.999942e+09
## 8     11 2.444722e-08 -17.526749 9.999831e+10
## 9     12 2.384186e-07 -15.249238 9.997149e+11
## 10    13 2.384186e-06 -12.946653 9.986438e+12
## 11    14 2.288818e-05 -10.684890 9.929697e+13
## 12    15 2.441406e-04  -8.317766 7.447273e+14
## 13    16 2.563477e-03  -5.966391 2.560000e+15
```