Question 4

Let $\{(Y_i, X_i)\}_{i=1}^n$ be an i.i.d. sequence of random vectors. Suppose that $\mathbb{E}[X_i X_i']$ and $\mathbb{E}[X_i Y_i]$ exists. Suppose further that there is no perfect colinearity in X_i, Hence $\mathbb{E}[X_i X_i']$ is invertible.

a

Does it also follow that

$$\frac{1}{n}\sum_{i=1}^n X_i X_i'$$

is invertible?

No. As a trivial case, consider when $n = 1, k = 2$ and $X_2 \sim \mathcal{N}(1, 1)$. Let a be any realization of X_2.

$$\frac{1}{n}\sum_{i=1}^n X_i X_i' = (1, a)'(1, a) = \begin{pmatrix} 1 & a \\ a & a^2 \end{pmatrix}$$

We can see that the second column is a times the first column, and the matrix is not invertible. This occurs because for any vector $x \in \mathbb{R}^k$, xx' always has rank 1.

b

For any $\lambda_n > 0$ show that

$$\frac{1}{n}\sum_{i=1}^n (X_i X_i' + \lambda_n \mathbb{I})$$

is invertible.

Note that this can be rewritten as

$$\left[\frac{1}{n}\sum_{i=1}^n X_i X_i'\right] + \lambda_n \mathbb{I}$$

For any given i, $X_i X_i'$ is positive semi-definite. The sum of positive semi-definite matrices is also positive semi-definite. This tells us that the first matrix is always positive semi-definite.

$$\frac{1}{n}\sum_{i=1}^n X_i X_i' \succeq 0$$

It is obvious that $\lambda_n \mathbb{I}$ is a positive definite matrix. The sum of a positive definite matrix and a positive semi-definite matrix is positive definite.

Proof: Let A be a positive semi-definite matrix, and B be a positive definite matrix. Then $\forall x \in \mathbb{R}^k, x'Bx > 0$ and $x'Ax \geq 0$. Consider two cases:

Case 1: $x \in \mathbb{R}^k, x'Ax > 0, x'Bx > 0$. Then:

$$\begin{aligned}(x'A + x'B)\, X &> 0 \\ x'(A+B)x &> 0\end{aligned}$$

Case 2: $x \in \mathbb{R}^k, x'Ax = 0, x'Bx > 0$ Then:

$$\begin{aligned}x'Ax + x'Bx &> 0 \\ (x'A + x'B)\, X &> 0 \\ x'(A+B)x &> 0\end{aligned}$$

This tells us that:

$$\left[\frac{1}{n}\sum_{i=1}^{n} X_i X_i'\right] + \lambda_n \mathbb{I} \succ 0$$

Any positive definite matrix has strictly positive eigenvalues, and therefore has a strictly positive determinant. This implies that the matrix is invertible.

C

Suppose that $\lambda_n \to 0$ as $n \to \infty$. Find the limit in probability of

$$\tilde{\beta}_n = \left(\frac{1}{n}\sum_{i=1}^{n}(X_i X_i' + \lambda_n \mathbb{I})\right)^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} X_i Y_i\right)$$

From the weak law of large numbers, we know that $\frac{1}{n}\sum_{i=1}^{n} X_i X_i' \xrightarrow{p} \mathbb{E}\left[XX'\right]$ and $\frac{1}{n}\sum_{i=1}^{n} X_i Y_i \xrightarrow{p} \mathbb{E}\left[XY\right]$.

We wish to show that

$$\frac{1}{n}\sum_{i=1}^{n} X_i X_i' + \lambda_n \mathbb{I} \xrightarrow{p} \frac{1}{n}\sum_{i=1}^{n} X_i X_i'$$

Applying the definition of convergence in probability.

$$\lim_{n\to\infty} \Pr\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i X_i' + \lambda_n \mathbb{I} - \frac{1}{n}\sum_{i=1}^{n} X_i X_i'\right| < \epsilon\right) = \lim_{n\to\infty} \Pr(|\lambda_n \mathbb{I}| < \epsilon)$$

We will consider this on an element-wise basis. Note that if we are not on a diagonal, $(\lambda_n \mathbb{I})_{ij} = 0$. So we may restrict ourselves to the diagonal elements of this matrix. However all the diagonal elements are the same, so this question amounts to the convergence of $|\lambda_n|$. Since λ_n is non-random:

$$\lim_{n\to\infty} \Pr(|\lambda_n| < \epsilon) = 1$$

As we have assumed that $\lambda_n \to 0$ above.

Thus
$$\frac{1}{n}\sum_{i=1}^{n} X_i X_i' + \lambda_n \mathbb{I} \xrightarrow{p} \frac{1}{n}\sum_{i=1}^{n} X_i X_i' \xrightarrow{p} \mathbb{E}\left[XX'\right]$$

As multiplication and inverting a matrix are continuous functions, we may apply the continuous mapping theorem to get that

$$\tilde{\beta_n} = \left(\frac{1}{n}\sum_{i=1}^{n}(X_i X_i' + \lambda_n \mathbb{I})\right)^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} X_i Y_i\right) \xrightarrow{p} \mathbb{E}\left[XX'\right]^{-1}\mathbb{E}\left[XY\right] = \beta$$

Question 8

a

```r
data <- read.csv( "ps4.csv" )

k <- ncol(data)
N <- nrow(data)

## Since we are not calling lm, we want to do matrix algebra, we need
## R to not store this stuff as a data frame. What a terrible language.

Y <-  as.matrix(data$y)
X <- as.matrix(cbind( rep(1,N), data[,2:3] ))

## Remember that matrix multiplication uses the %*%
mat <-  t(X)%*%X

## Rather than using inverses, let's be numerically stable and use the
## Cholesky decomp and forward/back substitution for legitimate answers
F <- chol(mat)

## We now have $X'X\beta = X'Y$
## This is equivalent to $F'F\beta = X'Y$
## Thus $\beta = F^{-1}F'^{-1}X'Y$

## Note that F' is lower triangular so we use forward substitution.
beta <-  backsolve( F, forwardsolve( t(F), t(X)%*%Y ) )
```

Our estimated values of β are: $(0.1680066, 1.0843565, 0.9203671)'$.

b

```r
## Now lets build our variance estimates.

outerproduct <- function( row ){
    row%*%t(row)
}

## We are interested in estimating $\left(\frac{1}{n}\sum_{i=1}^{n} X_i X_i'\right)^{-1}$


## The inner apply() forms the outer product matrices, the outer
## averages over them The matrix() reforms them as a matrix since
## apply flattens them.  This is equivalent to just doing
## $\frac{1}{n}X'X$, I just wanted some R practice.
outerProductGradient <- matrix( apply( apply( X, 1, outerproduct ), 1,
                                       mean ), nrow = k, ncol = k )

## Mama told me to never invert a matrix on a computer
varF <- chol( outerProductGradient )
informationEstimate <- backsolve( varF, forwardsolve( t(varF), diag(k) ) )

## Now lets get the heteroskedasticity-robust version of this bad boy.
## We multiply the matrix of $X_i X_i'$ by $\hat{u}_i^2$ component wise, hence no %
monstronsity <- matrix( apply(
    matrix( rep( (Y - X%*%beta)^2, k*k ), nrow=k*k, ncol = N, byrow = TRUE )
     * apply( X, 1, outerproduct ), 1, mean ), nrow = k, ncol = k )

## This is what are interested in: $\mathbb{V}\left(\widehat{\beta}_N | X\right)$
condVarHetero <- informationEstimate%*%monstronsity%*%informationEstimate


## Note that it's  possible to just use matrix operations to get there
## I just chose this way for practice and to have it look like the notes.
## One could always do $(X'X)^{-1} X' \widehat{\Sigma}_N X (X'X)^{-1}$
```

Our estimated Variance-Covariance Matrix of $\widehat{\beta}_N$ is:

$$\mathbb{V}\left(\widehat{\beta}_N | X\right) = \begin{pmatrix} 4.8905355 & 0.4493318 & -1.6478739 \\ 0.4493318 & 0.4517238 & -0.3702895 \\ -1.6478739 & -0.3702895 & 0.7567006 \end{pmatrix}$$

c

```r
## Now we face multiple linear restrictions in the form of $R\beta = r$

## We don't really know anything about the nature of $R\mathbb{V}\left(\widehat{\beta}_N\right)$
## So we can't rely on any decompositions, and we'll let solve() work here
multipleLinearTest <- function( R, r, N, beta, Var ){
    N*t(R%*%beta - r )%*%solve(R%*%Var%*%t(R))%*%(R%*%beta -r  )
}


R <-  matrix( c( 0, 0, 1, 0 ,0,1 ), nrow = 2, ncol = 3 )
r <- c(  1, 1 )

## This is free to be changed.
alpha <- .05

## This c is the critical value used in a hypothesis test
c <- qchisq( alpha, df = 2, lower.tail = FALSE )


testStat <- multipleLinearTest( R, r, N, beta, condVarHetero )
pValue <-  pchisq( testStat, df = 2, lower.tail = FALSE )
```

Our test statistic value is 1.599558 and our p-value is: 0.4494283

d

```r
## Testing: $f(\beta) = (\beta_1 - \beta_2)^2 = 0$
## However we need the rows of the total derivative to be linearly independent.
## $\nabla f(\beta) = (0, 2(\beta_1 - \beta_2), -2(\beta_1 - \beta_2))'$
## The rows are not linearly independent - The standard nonlinear test will not work.

## Worse yet, if we attempt to simply take the square root of both
## sides we lose the reliability as this is a Wald-Test. Wald Tests
## are not invariant to non-linear Transforms. This means we want to
## use a likelihood-ratio test, which is. However if we do not want to
## assume normality of Y and then the GLM framework to get a
## likelihood-ratio test, we can just stand for the errors in the Wald Test.

```

```r
## Our test is simply testing if β1 − β2 = 0

R <- matrix( c( 0, 1, -1 ), nrow = 1, ncol = 3 )
r <- c(0)

## I just copy and pasted the previous code
c <- qchisq( alpha, df = 1, lower.tail = FALSE )
testStat <- fischerFTest( R, r, N, beta, condVarHetero )
pValue <-  pchisq( testStat, df = 1, lower.tail = FALSE )
```

Our test statistic value is 1.379809 and our p-value is: 0.2401337