ps7

Baoyue Liang

11/11/2018

Problem 1

We should calculate the simulation variance(covariance) of the coefficient vector β, so taht we could get the Monte Carlo Simulation error.

$$var(\hat{\beta}) = \frac{1}{m}\sum_{i=1}^{n}(\hat{\beta}_i - \bar{\hat{\beta}}_i)^2$$

We need to check the if the sample standard deviation of all coefficient estimates is close to the average of the 1000 standard errors. If so, we can conclude that the statistical method properly characterized the uncertainty of the esimated regression coefficient.

Problem 2

(a)

We have 9 variables and 10^9 observation, the memory takes should be

$$(8+1) \times 10^9 \times 8 = 72 \times 10^9 btyes = 72GB$$

(b)

For R, if we save same combination of X next to each other, it would point to the same address, so we only need $10^4 \times 8 \times 8 = 64 \times 10^4 bytes = 0.64MB$ to save X.
Since there is only 10000 new row index, we need only 14 bits to save each index (2^{14} = 16384 > 10000).To map the new 10000 row index in binary format to original row index, we need need $14 \times 10^9 bits = 1.75GB$. And also, an extra$10^9 \times 8bytes = 8GB$ to save corresponding Y.
In total, 9.75064GB are used.

(c)

When the lm() or glm() function is called, x is going to form a matrix and all the calculation is based on the matrix in columnwise way. Information is extracted for each column, therefore, we still need to form the original matrix X.

(d)

Suppose we have matrix X^*, storing 10,000 unique combinations of the p covariates. And vector a^* storing the index.

First, we need to solve $X^\top X$. For the ij entry of $X^\top X$, we have $[X^\top X]_{ij} =< X_i, X_j >$. Also, $X^\top X$ is symmetric. Thus, we just need to compute the upper triangle entries of $X^\top X$ and use the use forceSymmetric function to get the entire matrix $X^\top X$. After getting the $X^\top X$, we inverse it to get $(X^\top X)^{-1}$

Below is the pseudo-code to get $(X^\top X)^{-1}$

```
p <- as.integer(8)
M <- matrix(data = NA, nrow=p, ncol=p)

for (i in 1:p){
  for (j in i:p){
    M[i,j] <- crossprod(X*[a*, i], X*[a*, j])  # get the ijth entry
  }
}

## use the property of symmetric matrix
M <-forceSymmetrix(M)

## find the inverse
solve(M)
```

Finally, we need to calculate $(X^\top X)^{-1}X^\top Y$. We can change the order of matrix multiplication by first calculating $X^\top Y$ to reduce the computational complexity.

```
B <- matrix(data = NA, nrow=p, ncol=1)

## compute the inner product of X and Y
for (f in 1:p){
 B[f, 1] <- crossprod(X*[a*, i], Y)
}

## get the final result
result <- M %*% B
```

Problem 3

To solve $\hat{\beta}$, we have

$$X^\top \Sigma^{-1} X\hat{\beta} = X^\top \Sigma^{-1} Y$$

Σ is positive definite, and thus we can write

$$\Sigma = \Gamma\Lambda\Gamma^{-1}$$

Then, we need to inverse Γ. Note that $\Gamma^{-1} = \Gamma^\top$ because Γ is an orthogonal matrix. And thus, we have

$$\Sigma^{-1} = (\Gamma\Lambda\Gamma^{-1})^{-1} = \Gamma\Lambda^{-1}\Gamma^{-1} = \Gamma\Lambda^{-1}\Gamma^\top$$

$$X^\top\Gamma\Lambda^{-1}\Gamma^\top X\hat{\beta} = X^\top\Gamma\Lambda^{-1}\Gamma^\top Y$$

Since Λ is a diagonal matrix of eigenvalues, $\Lambda^{-1} = \Lambda^{-1/2}\Lambda^{-1/2}$. And thus,

$$X^\top\Gamma\Lambda^{-1/2}\Lambda^{-1/2}\Gamma^\top X\hat{\beta} = X^\top\Gamma\Lambda^{-1/2}\Lambda^{-1/2}\Gamma^\top Y$$

Let $X^* = \Lambda^{-1/2}\Gamma^\top X$ and $Y^* = \Lambda^{-1/2}\Gamma^\top Y$, we have

$$X^{*\top} X^* \hat{\beta} = X^{*\top} Y^*$$

We can apply QR decomposition to X^*; that is, let $X^* = QR$, with Q orthogonal matrix and R upper triangular matrix.

$$(QR)^\top QR\hat{\beta} = (QR)^\top Y^*$$

$$R^\top Q^\top QR\hat{\beta} = R^\top Q^\top Y^*$$

Since Q is an orthogonal matrix, $Q^\top Q = I$.

$$R^\top R\hat{\beta} = R^\top Q^\top Y^*$$
$$R\hat{\beta} = Q^\top Y^*$$

We can solve $\hat{\beta}$ by backslove() because R is a upper triangular matrix.

```
gls <- function(X, Y, sigma){
  ## eigen decomposition
  e = eigen(sigma)
  gamma = e$vectors
  lambda = e$values
  sqrti_lambda = lambda^(-1/2)


  X_star = sqrti_lambda * t(gamma) %*% X
  Y_star = sqrti_lambda * t(gamma) %*% Y

  ## QR decomposition
  X_qr = qr(X_star)
  Q = qr.Q(X_qr)
  R = qr.R(X_qr)

  ## backsolve
  beta.hat = backsolve(R, crossprod(Q, Y.star))

  return(beta.hat)
}
```

Problem 4

(a)

To get the U, we zero out the first column of A except for the first row, and then proceed to zero out values below the diagonal for the other columns of A. The computational complexity to get U is

$$\sum_{i=1}^{n-1}(i+1)i = \sum_{i=1}^{n-1}(i^2+i) = \frac{n(n+1)}{2} + \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + O(n^2)$$

To get the I^*, The computational complexity is

$$n \times [(n-1) + (n-2) + ... + 1] = \frac{n^2(n+1)}{2} = \frac{n^3}{2} + O(n^2)$$

In total, the computational complexity to transformation is $\frac{5n^3}{6} + O(n^2)$.

(b)

For solving for Z, we can use backsolve(U, I*). The computational complexity is $\frac{n^3}{2}$

(c)

For calculating x=Zb, we use matrix multiplication. Z is a $n \times n$ matrix, and b is $n \times 1$ vector. Thus, the computational complexity to get x is n^2.

Problem 5

(a)

Method One:

Order of computations:

$\frac{4n^3}{3} + O(n^2)$

Reason:

It takes approximately $\frac{5n^3}{6} + \frac{n^3}{2}$ operations to solve X.

To compute $x^{-1}\%*\%y$, we need n^2 operations.

In total, we need $\frac{4n^3}{3} + O(n^2)$ operations.

Method Two:

Order of computations:

$n^3/3 + O(n^2)$

Using solve(X,y), we are actual using the LU decomposition

To get the upper triangle U, the operation is

$$\sum_{i=1}^{n-1}(i+1)i = \sum_{i=1}^{n-1}(i^2+i) = \frac{n(n+1)}{2} + \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + O(n^2)$$

To get the b*, the operation is

$$(n-1) + (n-2) + ... + 1 = \frac{n(n+1)}{2} = \frac{n^2}{2} + O(n)$$

To solve $UX = b^*$, we can use backsolve(U, b*). The operation is $\frac{n^2}{2}$.

In total, LU involves $n^3/3 + O(n^2)$ computations.

Method Three:

Order of computations:

Cholesky involves $n^3/6 + O(n^2)$ computations, which is half as many computations as LU Decomposition.

```
W <- matrix(rnorm(25000000), 5000, 5000)
X <- crossprod(W,W)
y <- rnorm(5000)

system.time(solve(X)%*%y)
```

```
##    user  system elapsed
## 243.022   2.323 255.610
```

```
system.time(LU <- solve(X,y))
```

```
##    user  system elapsed
##  29.846   0.271  30.960
```

```
system.time({
  U <- chol(X)
  Cholesky <- backsolve(U, backsolve(U, y, transpose = TRUE))
})
```

```
##    user  system elapsed
##  19.719   0.153  20.195
```

(b)

Since cond(A)$\approx \times 10^7$, we have accuracy of order 10^{-9}. We still have 9 digits accuracy relative to our original 16 on standard systems. Using formatC(), we can see that methods (b) and (c) generally agree to 9 digits.

```
## compute the condition number
e <- eigen (X)
cond.A <- max(e$values)/min(e$values)
print(cond.A)
```

```
## [1] 84230812
```

```
## check whether identical
identical(LU, Cholesky)
```

```
## [1] FALSE
```

```
## see how many digits are agreed
print(formatC(LU[1], 20, format = 'f'))
```

```
## [1] "-83.09847488110371216408"
```

```
print(formatC(Cholesky[1], 20, format = 'f'))
```

```
## [1] "-83.09847488745762689177"
```