ps7

Yihuan Song

11/12/2018

question 1

By knowing that m = 1000 and the result of each simulation giving h(Yi) and $\hat{\phi}$, we can calculate $\hat{Var}(\hat{\phi})$, based on which we could achieve the MC simulation error. To determine if the statistical method properly characterizes the uncertainty of the estimated regression coefficient, we could first calculate the standard error of the simulation, and see how it compares to the effect sizes. If the standard error is less than the effect size, we can say that we have a proper method.

question 2

(a) The dataset will take up 8np = 6.4*10^10 bytes = 64 GB.

(b) We could first create the matrix storing the 10000 unique combinations, which would be a $8 * 10^4$ matrix. Then we could create a vector such that each element is an integer index of one row in the original data set, for which the index corresponds every row in the original dataset to the matrix of the unique combinations.(For each combination, we have a unique integer index). Therefore, the total memory to be taken would be $8 * 8 * 10^4$ + 4*10^9 (one index is 4 bytes since the indexes are integer values), which is approximately 4GB.

(c) If we need to run lm() or glm() on this data, the method in (b) would not work, because in (b) we stored the matrix of the 10000 unique combinations by row, but the lm() would have to calculate the result based on each column, so our method cannot be used to store data if we would like to use it for the lm() function.

(d) We try to calculate $(X^TX)^{-1}X^TY$ estimator, so we need to calculate both $(X^TX)^{-}1$ and X^TY. Therefore, we can create the matrix of the unique combinations and call it U, and create the vector of index and call it N. For M = (X^TX), we could find M by finding the upper triangular part since it is symmetric. To calculate a single element Mij, we could find it by the inner product of each column of X. Our seudo-code could be:

```
for( i in  1:8){
  Xi = U[N,i]
  for(j in i:8){
     Xj = U[N,j]
     Mij = crossprod(Xi,Xj)
  }
}
```

Similarly, for Z = XY, we could find each Zij by the inner product of each column in X and Y.

```
for( i in  1:8){
  Xi = U[N,i]
  Zij = crossprod(Xi,Y)
}
```

Therefore, after we have M and Z, the calculation left is to take the inverse of M and multiply M by Z, since M is $8 * 8$, and Z is $8 * 1$, the calculation would not be costly.

question 3

pseudo-code: To solve for $\hat{\beta} = (X^T(\Sigma)^{-1}X)^{-1}X^T(\Sigma)^{-1}Y$, we need to solve:

$X^T(\Sigma)^{-1}X\hat{\beta} = X^T(\Sigma)^{-1}Y$

Then since we know that Σ is positive definite, we can use the eigen-decomposition to get $(\Sigma) = U(\Lambda)U^T$.

so it follows that $X^TU(\Lambda)^{-1}U^TX\hat{\beta} = X^TU(\Lambda)^{-1}U^TY$

so $X^TU(\Lambda)^{-1/2}(\Lambda)^{-1/2}U^TX\hat{\beta} = X^TU(\Lambda)^{-1/2}(\Lambda)^{-1/2}U^TY$since Λ is diagonal.

Then call $(\Lambda)^{-1/2}U^TX$ as X^*, and call $(\Lambda)^{-1/2}U^TY$ as Y^*.

we have $X^{*T}X^*\hat{\beta} = X^{*T}Y^*$

In this form, we can then apply QR decomposition on X^*, so

$R^TQ^TQR\hat{\beta} = R^TQ^TY^*$

cancelling and get

$R\hat{\beta} = Q^TY^*$

R code:

```
gls = function(X,Y, sig){
# eigen-decomposition for sigma
eigen = eigen(sig)
#get U and $\Lambda$
U = eigen$vectors
Lambda = eigen$values
LambdaSqrtInv = Lambda^(-1/2)
# substitute for new X and Y
Xstar = lambdaSqrtInv * t(U) %*% X
Ystar = lambdaSqrtInv * t(U) %*% Y
# QR decomposition on Xstar
QR = qr(Xstar)
Q = qr.Q(QR)
R = qr.R(QR)
# solve for betahat
betahat = backsolve(R, t(Q) %*% Ystar) }
```

question4

(a) Transforming AZ = I to UZ = I∗: the number of computations is $n^3/3$ to get U, plus $n * n^2/2$ to get I* (since we need $n^2/2$ to get one column); therefore, a total of $5n^3/6$ computation is needed.

(b) Solving for Z given UZ = I*: the number of computations is $n^2/2$ for one column, so a total of $n^3/2$ is needed.

(c) Calculating x = Zb: the number of computation is the order of multiplying $n * n$ matrix with $n * 1$ matrix, so the total computation is $n * n * 1 = n^2$

The total cost is $4n^3/3 + n^2$, which is much larger than the $n^3/3$ cost that we saw in class.

question 5

(a) Order of computation:

(a): $4n^3/3 + n^2$;

(b): $n^3/3 + n^2$;

(c): $n^3/6 + n^2$

Timing(time it took): (a) > (b) > (c) The first method is the slowest and had the largest order of computation, the third method is the fastest. The order of computation is (c) < (b) < (a), so the method having a larger order of computation will take more time.

```
set.seed(2)
n <- 5000
y <- rnorm(n)
W <- matrix(rnorm(n^2),n)
X <- crossprod(W)
system.time(
    out1 <- solve(X) %*% y
)
```

```
##    user  system elapsed
## 108.126   0.210 108.395
```

```
system.time(
    out2 <- solve(X, y)
)
```

```
##    user  system elapsed
##  24.338   0.063  24.412
```

```
system.time(
  {
    U <- chol(X)
    out3 <- backsolve(U, backsolve(U, y, transpose = TRUE))
  }
)
```

```
##    user  system elapsed
##  22.452   0.063  22.525
```

```
options(digits = 16)
print(out2[1])
```

```
## [1] 3.015320615421198
```

```
print(out3[1])
```

```
## [1] 3.015320632207052
```

```
print(out2[2])
```

```
## [1] -4.893017274905444
```

```
print(out3[2])
```

```
## [1] -4.893017273691108
```

(b) The results for b are approximately the same numerically for methods (b) and (c). The elements of b agree to 8 digits. Since our condition number is approximately 10^8, we have accuracy of order 10^8.

```
eigen_x <- eigen(X)
cond <- eigen_x$values[1] / eigen_x$values[length(eigen_x$values)]
cond
```

```
## [1] 101089596.9304507
```