ps7

Ramon Crespo

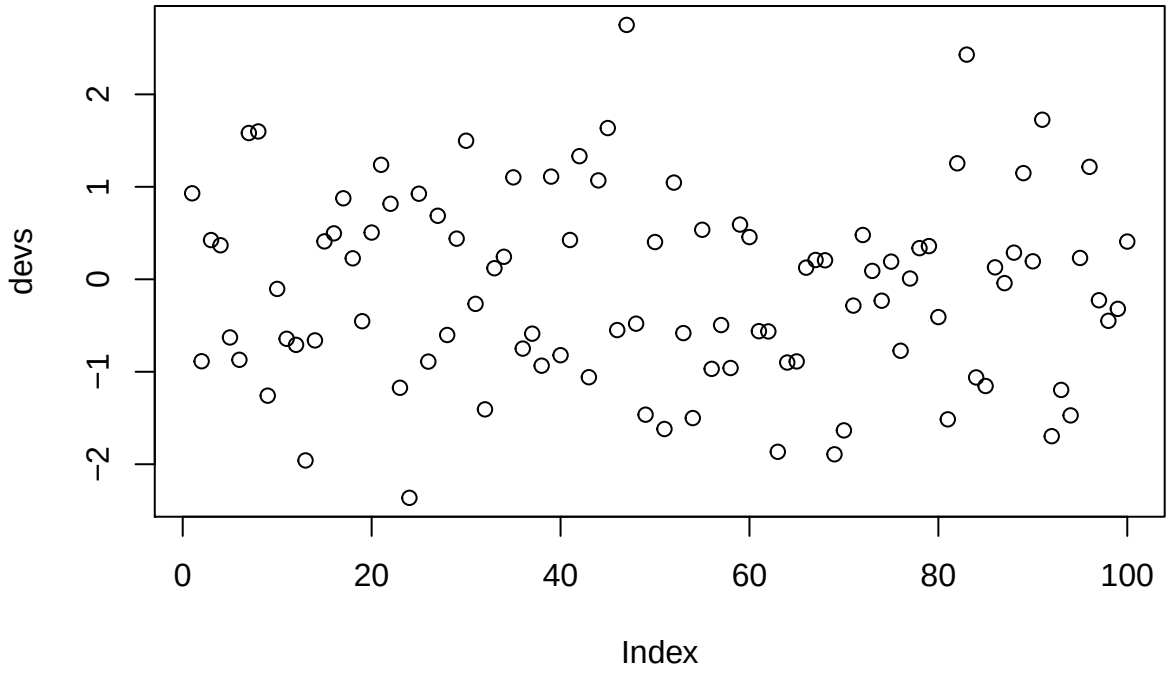
11/15/2018

Notes: Worked on this problem by myself.

Problem 1 Suppose I have a statistical method that estimates a regression coefficient and its standard error. I develop a simulation study and have m = 1000 simulated datasets that each give me an estimate of the coefficent and its standard error. How would I determine if the statistical method properly characterizes the uncertainty of the estimated regression coefficient?

A key element in determining the validity of the regression coeficients is to check for bias results. In the context of linear regression, this would be manifested in the linear regression model fitting a curve that is center slightly of the expected mean.

```
devs <- rnorm(100)
tdevs <- qt(pnorm(devs), df = 1)
plot(devs)
```



Problem 2 (a) Complete matrix = n = 1e9, d=8 –> 8e9 numbers x 8 bytes per number = 64 GB

(b) You can store the 10000 unique combinations, 10000 x 8numbers x 8bytes = 640,000 KB, and store an array of numbers that map the unique combinations to the values in the original matrix. The total space this takes is: 1e9 x 1 x 8bytes + 640,000KB = 8.64GB

(c) The functions would have to build back the matrix to be able to compute the calculations. In other words the original matrix will need to be recomputed for the lm() function to fit the linear model.

(d) Problem statement: Performing calculations on datasets are specific to the structure and content of the dataset. For the purpose of this problem I will introduce some structure to the dataset so that my formulation makes sense. The context is the dataset consists of observations of the state of a system, and the entries in the dataset consist of how far the observations are from a reference state 0. You are mapping this observations to a rewar Y that is a function of the state d and some error e (Y is iid).

The linear regression, would then be a predictor of rewards you can expect from visiting a specific state. In general, given that X = n x p matrix of and Y = p, it makes more sense to compute the operations in the following order = (X.T x X)-1 x (X.T x Y)

Pseudo code -import matrix D and S where S is the matrix encoding the 10000 possible states and D is the map between S and the real data set R -fit a model to the dataset D. -Step1: obtain E(Y), E((Y-E(Y)2) for each possible state by using the observed rewards. This is computed using the reduced dataset containing the codes for the states visited, not the actual states. -Step2: Now that you have a distribution for the expected rewards in each state, you can sample acual states and rewards by using the distributions. Sample from the obtained distribution E(Y), E((Y-E(Y)2) by using rnorm(E(Y),E((Y-E(Y)2). Choose a proper size for the dataset, lets say 100000. -Step3: compute w = (X.T x X)-1 x (X.T x Y) by using the sampled dataset. -Step 3.1 = instead of solving the full problem above, use : QR decomposition. We could try to use Cholesky decomposition but we depend on X.T x X being pd which is highly unlikely because states are discrete thus some repetition is highly likely): -QR Decomposition x X.T x X x w = X.T x Y x R.T x Q.T x .Q x R x w = Q.T x Y x R x w = Q.T x Y (this will be a backsolve since R is upper triangular)

-Step4: Check is results are consistent acros different simulations of the above procedure, specially making sure that you have not created a biased simulator that would lead to bias estimations of w.

The main point here is that in this context the actual data did not have to be accessed, and distributions could be obtained by just using the map (reduced dataset). Then expoit this structure to sample a smaller dataset and fit a linear regression to the reduced dataset, that is representative of the larger dataset.

3.

Basic Idea: 1) Separate the equation into efficient matrix and vector operations 2) Avoid doing the naive inverse, solve a linear system of equations instead 3) Avoid explicitly doing the transpose of matrices, use code that computes that directly.

pseudo-code - compute the inverse of epsilon - a - compute the crossproduct of X,epsilon - a - compute the crossproduct of a,X - a - compute the inverse of a - b - compute the crossproduct of epsilon inverse and Y - b - compute the crossproduct of X,b - c - compute the crossproduct of a,b

```
n <- 500
x <- matrix(rnorm(n*n,mean=100,sd=2),nrow=n,ncol=n)
y <- matrix(rnorm(n,mean=80,sd=1),nrow=n,ncol=1)
gls <- function(X,Y,epsilon) {
  eps_inv = inv(epsilon)
  a = inv(crossprod(tcrossprod(X,epsilon),X))
  b = tcrosspro(X,crossprod(eps_inv,Y))
  c =crossprod(a,b)
  return(c)
}
```

4.

(a) transforming AZ = I to UZ = I (where I is no longer a diagonal matrix), This requires 2/3(n3). This corresponds to transforming matriz A into an upper triangular matrix
(b) for solving for Z given UZ = I O(n2)
(c) for calculating x = Zb. O(n2)

total complexity 2/3(n3)+2(n2)

5.

(a) The computational time are as follows:

- solve(X)% x %y = 189s
- solve(X,y) = 41s
- Cholesky Decomposition = 26s

The cholesky decomposition is the fasted as expected. From class notes an inclass work we know that LU O(n3) and Cholesky is also O(n3) but involves only half as many calculations. This follows the results where the Cholesky decomposition takes almost half of the time is takes solve(X,y). The naive way is summarized in solve(X)% x %y where the extra time and computational burdain comes from computing two separate calculations, one is solving the system of linear equations and solving for the identity matrix (solve(X)), and then using this solution times the right hand side of the linear system to obtain the result of the operations. From the previous questions we see that the computational complexity of the naive method is quite higher that the computational complexity of the more efficient methods, this follows the results in this part of the problem

(b)

The results are not the same for b and c. As seen in the code below there is a distance between the two solutions. This is a result of the rounding error in the computation of the matrices. The from the dataframe we conclude that the solutions are similar to the 5th decimal point. The condition number in the matrices make the solutions different, and specifically Cholesky decomposition is sensisitive to the values in the matrix.

```
n <- 5000
w <- matrix(rnorm(n*n,mean=10,sd=2),nrow=n,ncol=n)
X <- t(w) %*% w
y <- matrix(rnorm(n,mean=100,sd=10),nrow=n,ncol=1)

time_a <-system.time(
a <- solve(X)%*%y
)

time_b <-system.time(
b <- solve(X,y)
)

time_c_1 <-system.time(
U <- chol(X)
)
time_c_2 <-system.time(
c_ <- backsolve(U, backsolve(U, y, transpose = TRUE))
)
print(time_a)
```

```
##    user  system elapsed
## 158.847   1.592 162.808
```

```
print(time_b)
```

```
##    user  system elapsed
##  34.591   0.203  34.982
```

```
print(time_c_1)
```

```
##    user  system elapsed
##  23.774   0.147  24.041
```

```
print(time_c_2)
```

```
##    user  system elapsed
##   0.032   0.000   0.032
```

```
df <- data.frame(b,c_)
df[0:10,]
```

```
##              b          c_
## 1   -36.561959  -36.561970
## 2   -50.695215  -50.695182
## 3   -23.144755  -23.144727
## 4     4.575191    4.575188
## 5   -31.052891  -31.052908
## 6    -5.505648   -5.505631
## 7   113.047797  113.047842
## 8   -56.060246  -56.060229
## 9  -106.698627 -106.698624
## 10  -39.624987  -39.624989
```