STAT 5014 Homework 6

Chaoran Wang

2017-10-10

Problem 2

```
## Using method a, we could get SST = 81774703374.569 and the time it costs is 106.7 .
##  And when using method b, we could get SST = 81774703374.5981 and the time it costs is
##  0.549999999999997 .
##  Method b is much faster!
```

Problem 3

This method is called gradient descent which I learned from my math class before. First, I create a function to do gradient descent in general. Then, I call the function to compute the intercept and slope. After comparing to the result from lm() function, we can see the results are almost same.

```
##
## Call:
## lm(formula = h ~ 0 + X)
##
## Coefficients:
##     X1      X2
## 0.9696  2.0016
```

```
## [1] "Intercept is  0.969618273035085 . Slope is  2.00155618852984 ."
```

Problem 4

First, I need to create two matrix X and Y using as.matrix function. Then using t() function to transpose the matrix X. Using %*% to do matrix multiplication and solve() to take the inverse of the matrix.

For example, we could use the following codes:

```
X = as.matrix(cbind(1,x_i))
Y = as.matrix(y_i)

beta_hat = round(solve(t(X)%*%X)%*%t(X)%*%Y, digits=2)
```

However, according to John Cook, solving the equation $Ax = b$ is faster than finding A^{-1}. Since $\hat{\beta} = (X'X)^{-1}X'\vec{y} \Rightarrow (X'X)\hat{\beta} = X'\vec{y}$, we can solve this equation to find β.

The following codes could be:

```
X = as.matrix(cbind(1,x_i))
Y = as.matrix(y_i)

beta_hat = round(solve(t(X)%*%X,t(X)%*%Y), digits=2)
```

Problem 5

Since my R keep crashing when creating these large datasets, I choose to half the sizes of all factors in this problem.

(a)

```
## 28086952 bytes
```

```
## 454089448 bytes
```

```
##    user  system elapsed
##   94.01    0.22   94.39
```

(b) & (c)

I break the equation into $y = p + AX$ where $X = B^{-1}(q - r)$. By using the strategy stated in Problem 4, that might cost less time than solve B directly.

```
##    user  system elapsed
##   76.69    0.07   76.77
```

As we can see, the break method does cost less time than the original one but not too much. I look forward to learn better methods from our classmates tomorrow.

Appendix 1: R code

```
############################
# Problem2
############################
set.seed(12345)
y <- seq(from = 1, to = 100, length.out = 1e+08) + rnorm(1e+08)
y_bar <- mean(y)
# Method a, for loop to calculate the summed squared difference
# between the data points and mean of the data
sst_a <- 0
sst_a_time <- system.time(for (i in 1:length(y)){
  sst_a <- sst_a + (y[i]-y_bar)^2
})

# Method b
sst_b_time <- system.time(sst_b <- sum((y-mean(y))^2))
# Conclusion
cat(paste("Using method a, we could get SST =", sst_a, "and the time it costs is",
          sst_a_time[1],".", '\n',"And when using method b, we could get SST =",
          sst_b, "and the time it costs is", '\n', sst_b_time[1],".", '\n',
          "Method b is much faster!"))
############################
# gradientdesc_fun
############################
gradientdesc <- function(x, y, theta, alpha, m, tol){
  theta0 <- theta[1]
  theta1 <- theta[2]
  h_0 <- theta0 + theta1*x
  stopornot <- F
  while(stopornot == F) {
    theta0_new <- theta0 - alpha*sum(h_0-y)/m
    theta1_new <- theta1 - alpha*sum((h_0-y)*x)/m
    if (abs(theta0_new-theta0) < tol && abs(theta1_new-theta1) < tol) {
      stopornot <- T
      return(paste("Intercept is ", theta0_new, ". Slope is ", theta1_new, "."))
    } else{
      theta0 <- theta0_new
      theta1 <- theta1_new
      h_0 <- theta0 + theta1*x
    }
  }
}
############################
# Problem3_compute
############################
set.seed(1256)
theta <- as.matrix(c(1,2),nrow=2)
X <- cbind(1,rep(1:10,10))
h <- X%*%theta+rnorm(100,0,0.2)
lm(h~0+X)
gradientdesc(x=X[,2], y=h, theta, alpha = 3e-5, m = 30, tol = 1e-9)
############################
```

```r
# Problem5
###########################
set.seed(12456)
G <- matrix(sample(c(0, 0.5, 1), size = 16000/2, replace = T), ncol = 10)
R <- cor(G) # R: 10 * 10 correlation matrix of G
C <- kronecker(R, diag(1600/2)) # C is a 16000 * 16000 block diagonal matrix
id <- sample(1:8002, size = 932/2, replace = F)
q <- sample(c(0, 0.5, 1), size = 15068/2, replace = T) # vector of length 15068
A <- C[id, -id] # matrix of dimension 932 * 15068
B <- C[-id, -id] # matrix of dimension 15068 * 15068
p <- runif(932/2, 0, 1)
r <- runif(15068/2, 0, 1)
C <- NULL #save some memory space
y <- p + A%*%solve(B)%*%(q-r)
###########################
# Problem5_size&time
###########################
object.size(A)
object.size(B)
system.time({y <- p + A%*%solve(B)%*%(q-r)})
###########################
# Problem5_break
# break apart: assume y = p+AX where X=inv(B)(q-r)
# Using the strategy in problem 4
###########################
X <- solve(B,q-r)
y_star <- p+A%*%X
system.time({X <- solve(B,q-r)
y_star <- p+A%*%X})
```