HW4_tanner95

Tanner Glass

10/13/2020

Problem 2: Using the dual nature to our advantage

Sometimes using a mixture of true matrix math plus component operations cleans up our code giving better readibility. Suppose we wanted to form the following computation:

- $while(abs(\Theta_0^i - \Theta_0^{i-1})$ AND $abs(\Theta_1^i - \Theta_1^{i-1}) > tolerance)$ {

$$\begin{aligned}\Theta_0^i &= \Theta_0^{i-1} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_0(x_i) - y_i)\\ \Theta_1^i &= \Theta_1^{i-1} - \alpha\frac{1}{m}\sum_{i=1}^{m}((h_0(x_i) - y_i)x_i)\end{aligned}$$

}

Where $h_0(x) = \Theta_0 + \Theta_1 x$.

Given $\mathbf{X}$ and $\vec{h}$ below, implement the above algorithm and compare the results with lm(h~0+$\mathbf{X}$). State the tolerance used and the step size, α.

```
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)
```

we will build the following algorithm using the started values of $\Theta_0 = 1$ and $\Theta_1 = 2$.

```
y <- X%*%theta
thetamat <-cbind(c(0,0),theta)
alpha<-0.05
i=2
while(abs(thetamat[1,i]-thetamat[1,i-1])>.0001 & abs(thetamat[2,i]-thetamat[2,i-1])>.0001) {
    newtheta0 <- thetamat[1,i]-alpha*mean(h[i-1]-y[i-1])
    newtheta1 <- thetamat[2,i]-alpha*mean((h[i-1]-y[i-1])*X[i-1,2])
    thetamat<-cbind(thetamat,c(newtheta0,newtheta1))
    i<-i+1
}
thetamat[,ncol(thetamat)]
```

```
## [1] 1.049571 2.054243
```

```
lm(h~0+X)
```

```
##
## Call:
## lm(formula = h ~ 0 + X)
```

```
##
## Coefficients:
##     X1      X2
## 0.9696  2.0016
```

We state our tolerance to be .0001 and our alpha to be 0.05. Compared to lm function, we get satisfactory results.

Problem 3

The above algorithm is called Gradient Descent. This algorithm, like Newton's method, has "hyperparameters" that are determined outside the algorithm and there are no set rules for determing what settings to use. For gradient descent, you need to set a start value, a step size and tolerance.

Part a.

Using a step size of $1e^{-7}$ and tolerance of $1e^{-9}$, try 10000 different combinations of start values for β_0 and β_1 across the range of possible β's +/-1 from true determined in Problem 2, making sure to take advantages of parallel computing opportunities. In my try at this, I found starting close to true took 1.1M iterations, so set a stopping rule for 5M. Report the min and max number of iterations along with the starting values for those cases. Also report the average and stdev obtained across all 10000 β's.

We use the algorithm with the requested step size and tolerance. we first construct 10000 betas that we will use.

```
#build-a-beta
beta0<-runif(10000,min=1-.9696,max=1+.9696)
beta1<-runif(10000,min=1.0016,max=3.0016)
beta<-rbind(beta0,beta1)


#set tolerance as .000000001 and alpha as .0000001


#implement algorithm with addition stop roll of iteration stop at 5m
alpha<- .0000001
beta0store<-c()
beta1store<-c()
iterstore<-c()
for (kc in 1:10000){
    i=2
    yp3 <- X%*%beta[,kc] #define under new betas at kth step
    betamat <-cbind(c(0,0),beta[,kc])
    hp3 <- X%*%beta[,kc]+rnorm(100,0,0.2)

while(abs(betamat[1,i]-betamat[1,i-1])>.000000001 & abs(betamat[2,i]-betamat[2,i-1])>.000000001) {



    newbeta0 <- betamat[1,i]-alpha*mean(hp3-yp3)
    newbeta1 <- betamat[2,i]-alpha*mean((hp3-yp3)*X[,2])
    betamat<-cbind(betamat,c(newbeta0,newbeta1))
if (i>501){
    break
```

```
}
    else(i<-i+1)
}
    beta0store<-c(beta0store,betamat[1,ncol(betamat)])
    beta1store<-c(beta1store,betamat[2,ncol(betamat)])
    iterstore<-c(iterstore,i)

}


min(iterstore) #min iterations
beta[,which.min(iterstore)] #starting values for min
max(iterstore) #max iterations
beta[,which.max(iterstore)] #starting values for max
mean(beta0store)
sd(beta0store)
mean(beta1store)
sd(beta1store)
```

Part b.

What if you were to change the stopping rule to include our knowledge of the true value? Is this a good way to run this algorithm? What is a potential problem? Because we know the true value, even though we set up tolerance and step levels, it would potentially causes problems of never actually meeting the number exaclty. therefore we would get an infinite loop.

Part c.

What are your thoughts on this algorithm? This algorithm is cool but taxing. Although i was able to succeed in making the algorithm work. i ended up having to create the stopping rule after around 500 otherwise cycling through all the iterations would have taken too long to run.

Problem 4: Inverting matrices

Ok, so John Cook makes some good points, but if you want to do:

$$\hat{\beta} = (X'X)^{-1}X'\underline{y}$$

what are you to do?? Can you explain what is going on?

You are achieving the result of finding $\hat{\beta}$ by solving for it directly in the equation. we note that $(X'X)^{-1}$ is always considered an invertible matrix since its square 2x2. The values of the matrix contain our sum of squares operations. Thus when multiplying by X' and Y, we construct the result for what we are looking for.

Problem 5: Need for speed challenge

In this problem, we are looking to compute the following:

$$y = p + AB^{-1}(q - r) \tag{1}$$

Where A, B, p, q and r are formed by:

```
    set.seed(12456)

    G <- matrix(sample(c(0,0.5,1),size=16000,replace=T),ncol=10)
    R <- cor(G) # R: 10 * 10 correlation matrix of G
    C <- kronecker(R, diag(1600)) # C is a 16000 * 16000 block diagonal matrix
    id <- sample(1:16000,size=932,replace=F)
    q <- sample(c(0,0.5,1),size=15068,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(932,0,1)
    r <- runif(15068,0,1)
    C<-NULL #save some memory space
```

Part a.

How large (bytes) are A and B? Without any optimization tricks, how long does the it take to calculate y?

112347224 bytes for A and 1816357208 bytes for B. Rstudio cloud will not compile the data. crashes before it can every time. I booted up my local R to find these values. My local computer has CPU/memory issues as its an older computer, and will not actually allocate the value of y. To receive credit for the problem, i have attached my code.

```
object.size(A)
object.size(B)
memory.limit(size = 40000000)
 time1<-Sys.time()
    y<-p+A*solve(B)*(q-r)

time2<-Sys.time()
time2-time1
```

Part b.

How would you break apart this compute, i.e., what order of operations would make sense? Are there any mathmatical simplifications you can make? Is there anything about the vectors or matrices we might take advantage of?

We could break this apart using apply family of functions, since p and q-r are vectors, we could just apply them to the matrix A*inverse(B). As far as efficiency in matrix operations, as discussed in lecture, there are way to improve the results of matrix operations to make this computation faster.

Part c.

Use ANY means (ANY package, ANY trick, etc) necessary to compute the above, fast. Wrap your code in "system.time({})", everything you do past assignment "C <- NULL".

Although i am not able to run it on Rstudio cloud, i am not able to check if this significantly increases the speed. There exists packages out there such as gpuR that given you have the right equipment, you may use those GPU cores in parallel to compute much faster

```
#library("gpuR")
ab1<-A*solve(B)
mapply(ab1,2,*(q-r)+p)
```

Problem 3

a. Create a function that computes the proportion of successes in a vector. Use good programming practices.

Success are defined as 0=failure 1= success. We define a functions as follows:

```
p3fun_a<- function (vec){
  prop<-mean(vec)
  return(prop) }

p3fun_a(c(1,1,1,0,0,0,0))
```

```
## [1] 0.4285714
```

We have tested the function and see that it works.

b. Create a matrix to simulate 10 flips of a coin with varying degrees of "fairness" (columns = probability) as follows:

```
set.seed(12345)
P4b_data <- matrix(rbinom(10, 1, prob = (31:40)/100), nrow = 10, ncol = 10, byrow = FALSE)
P4b_data
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    1    1    1    1    1    1    1    1     1
##  [2,]    1    1    1    1    1    1    1    1    1     1
##  [3,]    1    1    1    1    1    1    1    1    1     1
##  [4,]    1    1    1    1    1    1    1    1    1     1
##  [5,]    0    0    0    0    0    0    0    0    0     0
##  [6,]    0    0    0    0    0    0    0    0    0     0
##  [7,]    0    0    0    0    0    0    0    0    0     0
##  [8,]    0    0    0    0    0    0    0    0    0     0
##  [9,]    1    1    1    1    1    1    1    1    1     1
## [10,]    1    1    1    1    1    1    1    1    1     1
```

c. Use your function in conjunction with apply to compute the proportion of success in P4b_data by column and then by row. What do you observe? What is going on?

```
apply(P4b_data,1,p3fun_a) #rows
```

```
##  [1] 1 1 1 1 0 0 0 0 1 1
```

```
apply(P4b_data,2,p3fun_a)#colomns
```

```
##  [1] 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6
```

They seem to be constant throughout, even with the change in probability.

d. You are to fix the above matrix by creating a function whose input is a probability and output is a vector whose elements are the outcomes of 10 flips of a coin. Now create a vector of the desired probabilities. Using the appropriate apply family function, create the matrix we really wanted above. Prove this has worked by using the function created in part a to compute and tabulate the appropriate marginal successes.

```
p3fun_d<- function(prop){
  result<-rbinom(10,1,prop)
  return(result)
}
ans<-p3fun_d(31:40/100)
```

```r
p3mat<- matrix(0,nrow = 10,ncol = 10)
#mapply(p3mat,1,+ans)
#little confused on appply function, just want to add results per row
```

Problem 4

In Homework 4, we had a dataset we were to compute some summary statistics from. The description of the data was given as "a dataset which has multiple repeated measurements from two devices by thirteen Observers". Where the device measurements were in columns "dev1" and "dev2". Reimport that dataset, change the names of "dev1" and "dev2" to x and y and do the following:

1. create a function that accepts a dataframe of values, title, and x/y labels and creates a scatter plot
2. use this function to create:
   (a) a single scatter plot of the entire dataset
   (b) a seperate scatter plot for each observer (using the apply function)

```r
HW4_data <- readRDS("/cloud/project/HW3_data.rds")
names(HW4_data)<-c("Observer","x","y")

p4fun<-function(df){
  return(plot(df[,2],df[,3]))
}
p4fun(HW4_data)
```