2017 Big Data – R programming Homework 3

2017/10/28

范真瑋

Part I: Implement function “apply”

```
my_apply <- function(Data, Margin, Func){
  FUN <- match.fun(Func)  #extract a function specified by name
  dl <- length(dim(Data)) #get the dimension
  d <- dim(Data)          #d is a vector that stores Data's each dimension
  ds <- seq_len(dl)       #generate a vector from 1 to dl
  if(Margin!=1 && Margin!=2)
    stop("Margin error")
  s.call <- ds[-Margin]   #if Margin=1, s.call=2
  s.ans <- ds[Margin]
  d.call <- d[-Margin]
  d.ans <- d[Margin]
  newData <- aperm(Data, c(s.call, s.ans))  #transposition
  ans <- vector("list", d.ans)
  for(i in 1L:d.ans){
    tmp <- forceAndCall(1, FUN, newData[, i])  #call a function with some arguments
    if(!is.null(tmp))
      ans[[i]] <- tmp
  }
  ans.list <- is.recursive(ans[[1L]])
  len.a <- if (ans.list)
    d2
  else length(ans <- unlist(ans, recursive = FALSE))
  if(Func %in% c("range", "sort", "rev")) {
    array(ans, c(len.a%/%d.ans, d.ans))
  }
  else return(ans)
}
```

```
> x <- cbind(3, c(4:1, 2:5))
> x
     [,1] [,2]
[1,]    3    4
[2,]    3    3
[3,]    3    2
[4,]    3    1
[5,]    3    2
[6,]    3    3
[7,]    3    4
[8,]    3    5
```

```
> col.sum <- my_apply(x, 2, "sum")
> col.sum
[1] 24 24
> row.sum <- my_apply(x, 1, "sum")
> row.sum
[1] 7 6 5 4 5 6 7 8
```

```
> col.sort <- my_apply(x, 2, "sort")
> col.sort
     [,1] [,2]
[1,]    3    1
[2,]    3    2
[3,]    3    2
[4,]    3    3
[5,]    3    3
[6,]    3    4
[7,]    3    4
[8,]    3    5
> row.sort <- my_apply(x, 1, "sort")
> row.sort
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    3    3    2    1    2    3    3    3
[2,]    4    3    3    3    3    3    4    5
```

Part II: Create Data & Read/Write

1. 使用 random 函數，產生 25 筆資料，且由小而大排列，並存入 my_rand 變數中

```
> my_rand <- sort(runif(25))
> my_rand
 [1] 0.05524984 0.09316368 0.09686369 0.11777632 0.18211228 0.27235339 0.28708282 0.37874881 0.40042551 0.42308558
[11] 0.44546256 0.46984261 0.47726600 0.50634750 0.52894411 0.59426784 0.72427050 0.76698642 0.80113934 0.83901311
[21] 0.87394218 0.89025670 0.90897453 0.92584064 0.93628716
```

2. 將 my_rand 的變數轉化為 5X5 的矩陣並存入 my_matrix 變數中

```
> my_matrix <- matrix(my_rand, nrow = 5, ncol = 5)
> my_matrix
           [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.05524984 0.2723534 0.4454626 0.5942678 0.8739422
[2,] 0.09316368 0.2870828 0.4698426 0.7242705 0.8902567
[3,] 0.09686369 0.3787488 0.4772660 0.7669864 0.9089745
[4,] 0.11777632 0.4004255 0.5063475 0.8011393 0.9258406
[5,] 0.18211228 0.4230856 0.5289441 0.8390131 0.9362872
```

3. 將 my_matrix 的資料寫入 matrix.txt 中

```
> write.table(my_matrix, "matrix.txt")
```

matrix.txt - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

```
"V1" "V2" "V3" "V4" "V5"
"1" 0.0552498374599963 0.272353389300406 0.445462562143803 0.594267835374922 0.873942178208381
"2" 0.0931636777240783 0.287082820665091 0.469842611346394 0.724270503036678 0.890256703132764
"3" 0.0968636942561716 0.3787488094531 0.477266001980752 0.766986420378089 0.908974527381361
"4" 0.117776321014389 0.400425505125895 0.506347495829687 0.801139336079359 0.925840643933043
"5" 0.182112277951092 0.423085580114275 0.528944112826139 0.839013105724007 0.936287164222449
```

4. 從 matrix.txt 中讀入資料，並放入 read_matrix 變數中

```
> read_matrix <- as.matrix(read.table("matrix.txt", header = T))
> read_matrix
          V1        V2        V3        V4        V5
1 0.05524984 0.2723534 0.4454626 0.5942678 0.8739422
2 0.09316368 0.2870828 0.4698426 0.7242705 0.8902567
3 0.09686369 0.3787488 0.4772660 0.7669864 0.9089745
4 0.11777632 0.4004255 0.5063475 0.8011393 0.9258406
5 0.18211228 0.4230856 0.5289441 0.8390131 0.9362872
```

5. 使用 partI 中的 my_apply，將 read_matrix 當成 data，並用 loop 跑 5 次，每次所需使用的 margin 以及 func 須由亂數產生

```
> for(i in 1:5) {
+   margin <- sample(1:2, 1, rep=T)
+   func <- sample(c("sum", "max", "min", "range", "mean", "median", "var", "sd", "sort", "rev", "prod"), 1, rep = T)
+   print(paste0("my_apply(read_matrix, ", margin, ", ", func, ")"))
+   print(my_apply(read_matrix, margin, func))
+ }
[1] "my_apply(read_matrix, 1, sort)"
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.05524984 0.09316368 0.09686369 0.1177763 0.1821123
[2,] 0.27235339 0.28708282 0.37874881 0.4004255 0.4230856
[3,] 0.44546256 0.46984261 0.47726600 0.5063475 0.5289441
[4,] 0.59426784 0.72427050 0.76698642 0.8011393 0.8390131
[5,] 0.87394218 0.89025670 0.90897453 0.9258406 0.9362872
[1] "my_apply(read_matrix, 2, max)"
[1] 0.1821123 0.4230856 0.5289441 0.8390131 0.9362872
[1] "my_apply(read_matrix, 1, min)"
[1] 0.05524984 0.09316368 0.09686369 0.11777632 0.18211228
[1] "my_apply(read_matrix, 2, sum)"
[1] 0.5451658 1.7616961 2.4278628 3.7256772 4.5353012
[1] "my_apply(read_matrix, 2, mean)"
[1] 0.1090332 0.3523392 0.4855726 0.7451354 0.9070602
```