PM566-lab09-Tiansheng-Jin

Tiansheng Jin

2022-10-26

```
library(parallel)
```

Problem 2

1.This function generates a n x k dataset with all its entries distributed poission with mean lambda.

```
set.seed(156)

fun1 <- function(n = 100, k = 4, lambda = 4) {
  x <- NULL

  for (i in 1:n)
    x <- rbind(x, rpois(k, lambda))

  return(x)
}
f1 <- fun1(100, 4)
mean(f1)
```

```
## [1] 4.02
```

```
f1 <- fun1(1000, 4)
f1 <- fun1(10000, 4)
f1 <- fun1(50000, 4)

fun1alt <- function(n = 100, k = 4, lambda = 4) {
  x <- matrix(rpois(n*k, lambda) , ncol = 4)
}

# Benchmarking
microbenchmark::microbenchmark(
  fun1(),
  fun1alt()
)
```

```
## Warning in microbenchmark::microbenchmark(fun1(), fun1alt()): less accurate
## nanosecond times to avoid potential integer overflows
```

```
## Unit: microseconds
##       expr     min       lq      mean   median       uq     max neval
##     fun1() 164.656 187.7185 199.91477 197.3535 207.9110 323.982   100
##  fun1alt()  13.161  14.0425  25.29946  14.7190  15.5185 973.422   100
```

2.Find the column max (hint: Checkout the function max.col()).

```r
# Data Generating Process (3 x 4 matrix)
set.seed(156)
M <- matrix(runif(12), nrow=4)

# Find each column's max value
fun2 <- function(y) {
  apply(y, 2, max)
}
fun2(y=M)
```

```
## [1] 0.7061012 0.5870727 0.9189349
```

```r
fun2alt <- function(y) {
  idx <- max.col(t(y))
  idx
  # y[cbind(idx,1:4)]
}
fun2alt(y=M)
```

```
## [1] 4 1 1
```

```r
y <- matrix(rnorm(1e4), ncol = 10)

# Benchmarking
microbenchmark::microbenchmark(
  fun2(y),
  fun2alt(y)
)
```

```
## Unit: microseconds
##        expr    min      lq      mean  median       uq     max neval
##     fun2(y) 89.134 93.3980 107.27978 99.6095 106.9280 639.641   100
##  fun2alt(y) 45.305 50.2045  59.71896 52.7055  55.1655 612.663   100
```

Problem 3: Parallelize everything

```r
my_boot <- function(dat, stat, R, ncpus = 1L) {

  # Getting the random indices
  n <- nrow(dat)
  idx <- matrix(sample.int(n, n*R, TRUE), nrow=n, ncol=R)

  # Making the cluster using `ncpus`
  # STEP 1: GOES HERE
  cl <- makePSOCKcluster(4)
  clusterSetRNGStream(cl, 123)

  # STEP 2: GOES HERE
  clusterExport(cl, c("stat", "dat", "idx"), envir = environment())

  # STEP 3: THIS FUNCTION NEEDS TO BE REPLACES WITH parLapply
  ans <- lapply(seq_len(R), function(i) {
    stat(dat[idx[,i], , drop=FALSE])
```

```r
  })

  # Coercing the list into a matrix
  ans <- do.call(rbind, ans)

  # STEP 4: GOES HERE

  ans

}
```

```r
# Bootstrap of an OLS
my_stat <- function(d) coef(lm(y ~ x, data=d))

# DATA SIM
set.seed(1)
n <- 500; R <- 1e4

x <- cbind(rnorm(n)); y <- x*5 + rnorm(n)

# Checking if we get something similar as lm
ans0 <- confint(lm(y~x))
ans1 <- my_boot(dat = data.frame(x, y), my_stat, R = R, ncpus = 2L)

# You should get something like this
t(apply(ans1, 2, quantile, c(.025,.975)))
```

```
##                   2.5%      97.5%
## (Intercept) -0.1386903 0.04856752
## x            4.8685162 5.04351239
```

```r
##                   2.5%      97.5%
## (Intercept) -0.1372435 0.05074397
## x            4.8680977 5.04539763
ans0
```

```
##                  2.5 %     97.5 %
## (Intercept) -0.1379033 0.04797344
## x            4.8650100 5.04883353
```

```r
##                  2.5 %     97.5 %
## (Intercept) -0.1379033 0.04797344
## x            4.8650100 5.04883353
```

```r
system.time(my_boot(dat = data.frame(x, y), my_stat, R = 4000, ncpus = 1L))
```

```
##    user  system elapsed
##   1.199   0.032   1.359
```

```r
system.time(my_boot(dat = data.frame(x, y), my_stat, R = 4000, ncpus = 2L))
```

```
##    user  system elapsed
##   1.179   0.030   1.334
```