HW 3 Elias Washor Comp in Stats

Elias Washor

2024-09-19

Q1) map_dbl

```r
library(purrr)
```

```
## Warning: package 'purrr' was built under R version 4.2.3
```

```r
#Prof. Wang -- question intention is to find factorial of a vector of numbers

take_fact <- function(x) {
  map_dbl(c(1:x), factorial )
}

## same result
take_fact(5)
```

```
## [1]   1   2   6  24 120
```

```r
factorial(1:5)
```

```
## [1]   1   2   6  24 120
```

```r
library(microbenchmark)
```

```
## Warning: package 'microbenchmark' was built under R version 4.2.3
```

```r
microbenchmark(
  take_fact(5),
  factorial(1:5))
```

```
## Unit: nanoseconds
##            expr   min    lq   mean median    uq      max neval cld
##    take_fact(5) 19500 20000 151787  20450 23250 12847800   100   a
##  factorial(1:5)   500   600   1175    700   800    15000   100   a
```

My function take_fact() was slower than the native factorial function by a factor of approximately 30. Median time for my function was 23000 ns while only 800 for factorial().

Q2) moments::skewness

```r
library(moments)

## nested function
compute_skew <- function(x, drop_na= FALSE) {
  if (drop_na) {
    x[!is.na(x)]}
  n <- length(x)
  dif <- x - mean(x)
  dif_sq <- dif * dif
  psum2 <- sum(dif_sq / n)
  return ((sum(dif * dif_sq) /n) / (sqrt(psum2 * psum2 * psum2 )) )
}

### MAIN Function
get_skewness <- function(x, drop_na= FALSE) {
  if (is.vector(x)) {compute_skew(x)}
  else if (is.matrix(x)) {
    apply(x, compute_skew, na.rm = drop_na)}
  else if (is.data.frame(x)) {
    sapply(x, compute_skew, na.rm = drop_na)
    }
  else {compute_skew(as.vector(x))}
}

library(microbenchmark)

tests <- rexp(10000)
microbenchmark(
  moments::skewness(tests),
  get_skewness(tests)
)
```

```
## Unit: microseconds
##                      expr   min     lq    mean median     uq    max neval cld
##  moments::skewness(tests) 481.8 519.05 706.409 530.15 560.85 9908.3   100  a
##       get_skewness(tests)  63.1 127.35 231.038 135.90 148.25 9552.1   100   b
```

a) My function was faster than moment's skewness() function. I ran both on vectors of size 10000 and my function was almost twice as fast: 362 ns compared to 641 mean ns. Also the median time was 4 times faster on my function: 614 ns compared to 156 ns.

Q3) Monty Hall Problem

a) The contestant should switch curtains. If the contestant switches the probability of winning is now 2/3 instead of 1/3.

b) Simulation yields 0.663, which is approximately 2/3

```r
#simulation
monty_vector = rep(NA, 10000)

monty_hall <- function() {
  (doors <- c(1:3))
  (prize <- sample.int(3,1))
  (goats <- doors[doors != prize])
  (chosen <- sample.int(3,1))
  (temp <- goats[goats != chosen])

  ## if prize chosen
  if (length(temp) > 1) {reveal <- sample(temp, 1)}
  else {reveal <- temp}

  (switch_door <- doors[-c(reveal, chosen)])
  (as.integer(switch_door == prize))
}

for (i in 1:10000) {
  monty_vector[i] <- monty_hall()
}

(win_probability <- mean(monty_vector))
```

```
## [1] 0.6648
```

c) The probability of winning when switching with m prizes and n doors is

   (m(n - 1)) / (n(n - 2))

   For $m = 2$ and $n = 4$, we get 0.75 for the theoretical probability and in the simulation, we find it to be 0.743, which is close to 0.75.

```r
## probability monty hall problem for m prizes and n doors
gen_monty_hall <- function(m, n) {
  (doors <- 1:n )
  (prize <- sample.int(n,m))
  (goats <- doors[-c(prize)])
  (chosen <- sample.int(n,1))
  (temp <- goats[goats != chosen])

  if (length(temp) > 1) {reveal <- sample(temp, n - 2)
  } else {reveal <- temp}

  (reveal)
  (switch_door <- doors[-c(reveal, chosen)])
  any(switch_door == prize)
}

## 2 prizes and 4 doors
gen_monty_vector <- rep(NA, 10000)
for (i in 1:(length(gen_monty_vector))) {
  gen_monty_vector[i] <- gen_monty_hall(2,4)
}
```

```
(two_four_prb <- mean(gen_monty_vector))
```

```
## [1] 0.7516
```

Q4) Coding Practice

```
foo <- function (n) {
  log_of <- log(n, base = 4)
  return (round(log_of) == log_of)
  # input: foo(n)
  # output: TRUE or FALSE
  # example foo(16) outputs TRUE and foo(31) outputs FALSE
}

foo(16)
```

```
## [1] TRUE
```

```
foo(31)
```

```
## [1] FALSE
```

```
### verify function w test cases. It works
library(purrr)
map_dbl(c(4,16,64,256, 20), foo)
```

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

```
map_dbl(c(1,4,10, 16, 100), foo)
```

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

Q5) Coding Practice

```
set.seed(5400)

foo2 <- function(a1, a2)
{
  m <- length(a1)
  n <- length(a2)
  result <- rep(NA, m + n )
  i <- 1
  j <- 1
  k <- 1
  while (i <= m & j <= n) {
    if (a1[i] < a2[j]) {
```

```r
      result[k] <- a1[i]
      i <- i + 1
      k <- k + 1
    }
    else if (a1[i] > a2[j]) {
      result[k] <- a2[j]
      j <- j + 1
      k <- k + 1
    }
    else {
      result[k] <- a1[i]
      i <- i + 1
      k <- k + 1
      }
  }
  if (i > m) {result[k:(m+n)] <- a2[j:n]
  }
  else {result[k:(m+n)] <- a1[i:m]}
  return (result)
}
a1 <- sort(sample(8, 8, TRUE))
a2 <- sort(sample(10, 8, TRUE))

## test case
foo2(a1, a2)
```

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

```r
## check with 1000 test cases for seeds
lgl <- rep(NA, 1000)

for (i in 1000:2000) {
  set.seed(i)
  lgl[i - 999] <- (all.equal(sort(c(a1,a2)), foo2(a1,a2)))
}
### all equal to sort function in test cases??
all(lgl)
```

```
## [1] TRUE
```