HW 2 Computing in Statistics

Elias Washor

2024-09-13

1

```
## vector of 500_000
cat("First Method:\n")
```

```
## First Method:
```

```
system.time(sum(rnorm(500000)))
```

```
##    user  system elapsed
##    0.01    0.00    0.03
```

```
answer <- 0
cat("Second Method:\n")
```

```
## Second Method:
```

```
system.time(for (i in 1:500000) answer <- answer + rnorm(1))
```

```
##    user  system elapsed
##    0.99    0.22    1.27
```

Method 1: vector of length 500,000 random numbers worked much more efficiently (~36x faster) at 0.03 seconds, compared to method 2, which ran in appx. 1.07 seconds. There is some variability in the timings for the second method.

2

```
(M <- matrix(c(1,2,3,3,4,6,5,6,9), 3, 3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
## [3,]    3    6    9
```

```r
(v <- c(17, 46, 181))
```

```
## [1]  17  46 181
```

```r
(Mv <- M * v)
```

```
##      [,1] [,2] [,3]
## [1,]   17   51   85
## [2,]   92  184  276
## [3,]  543 1086 1629
```

```r
(MTranspose <- t(M))
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    3    4    6
## [3,]    5    6    9
```

```r
v[v < 50]
```

```
## [1] 17 46
```

3

```r
#install.packages('cumstats')
library(cumstats)
## my example
skewness(c(1,2,3,5))
```

```
## [1] 0.4346508
```

a) The function name is skewness() from the 'cumstats' package.

b) The function is fivenum() from the 'stats' package

```r
library(cumstats)

skewFunc <- function(x) {
  if (class(x) == "numeric") {
    y <- na.omit(x)
    sk <- skewness(y)
    if (abs(sk) < 1) {
      return (list('skewness' = sk, 'descstats' = c(mean(y), sd(y) )))
      }
    else{ return(list('skewness' = sk, 'descstats' = fivenum(y)))
    }
  }
  print("Vector must be numeric and exist")
}

skewFunc(c("stat", "actuarial", "2022"))
```

```
## [1] "Vector must be numeric and exist"
```

```
skewFunc(rnorm(100))
```

```
## $skewness
## [1] 0.2030662
##
## $descstats
## [1] 0.06141652 1.06794595
```

```
skewFunc(rexp(5))
```

```
## $skewness
## [1] 1.262184
##
## $descstats
## [1] 0.05044857 0.06092610 0.27088066 0.40148021 1.43438345
```

4

```
### generate data
set.seed(1)
m <- 1000
n <- 50
X <- matrix(rnorm(m * n, mean=10, sd=3), nrow=m)
grp <- rep(1:2, each=n/2)

### first row
a <- X[1, grp==1]
b <- X[1, grp==2]

n1 <- length(a)
n2 <- length(b)
x_1 <- mean(a)
x_2 <- mean(b)
x_delta <- x_1 - x_2

Sp_sq <- (((n1-1)* var(a) ) + ((n2-1)* var(b))) / (n1 + n2 - 2)
Sp <- sqrt(Sp_sq)

(test <- x_delta / (Sp * sqrt(1/n1 + 1/n2)))
```

```
## [1] -0.5284632
```

```
### compare with
(t.test(X[1, grp==1], X[1, grp==2])$stat)
```

```
##          t
## -0.5284632
```

```r
### all 1000 samples
ComputeTest <- function(M) {
  tStats <- rep(NA, length = NROW(M))
  for (i in 1:nrow(M)) {
    a <- M[i, grp==1]
    b <- M[i, grp==2]

    n1 <- length(a)
    n2 <- length(b)
    x_1 <- mean(a)
    x_2 <- mean(b)
    x_D <- x_1 - x_2

    Sp_sq <- (((n1-1)* var(a) ) + ((n2-1)* var(b))) / (n1 + n2 - 2)
    Sp <- sqrt(Sp_sq)

    test <- x_D / (Sp * sqrt(1/n1 + 1/n2))
    #print(test)
    tStats[i] <- test
    ## or tStats <- c(tStats,test)
  }
  #print(tStats)
  return (tStats)
}

### Given Code
rowtstat <- function(X, grp){
  t_stat <- function(X) {
    m <- rowMeans(X)
    n <- ncol(X)
    var <- rowSums((X - m) ^ 2) / (n - 1)

    list(m = m, n = n, var = var)
  }

  g1 <- t_stat(X[, grp == 1])
  g2 <- t_stat(X[, grp == 2])

  se_total <- sqrt(g1$var / g1$n + g2$var / g2$n)
  (g1$m - g2$m) / se_total
}
cat('\n')
```

```r
## my function timing
system.time(ComputeTest(X))
```

```
##    user  system elapsed
##    0.08    0.00    0.06
```

```r
## Webpage example
system.time(rowtstat(X, grp))
```

```
##    user  system elapsed
```

```
##    0.01    0.00    0.02
```

```
all.equal(ComputeTest(X), rowtstat(X, grp))
```

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

My code is slightly slower at 0.06 seconds compared with 0.01 because I calculated row means, row by row, whereas the faster method uses rowMeans(), which I assume is optimized. I believe by also using ncol() and rowSums() in the manual variance calculation and storing the results in a vector there was a faster computation of the 1000 test statistics. Lastly, I used the function var(), which may have been a slower way to get the two variances.