Exercises - Class 3: Vectors

Sander Bossier

2023-10-21

1.

Create a 13x2 matrix `mat1` with the elements of the (built-in) vector `letters`in row-major order.

```
mat1 <- matrix(data = letters, ncol = 2, byrow = TRUE)
mat1
```

```
##       [,1] [,2]
##  [1,] "a"  "b"
##  [2,] "c"  "d"
##  [3,] "e"  "f"
##  [4,] "g"  "h"
##  [5,] "i"  "j"
##  [6,] "k"  "l"
##  [7,] "m"  "n"
##  [8,] "o"  "p"
##  [9,] "q"  "r"
## [10,] "s"  "t"
## [11,] "u"  "v"
## [12,] "w"  "x"
## [13,] "y"  "z"
```

2.

Create another 13x2 matrix mat2 with the elements of letters in column-major order. How can you transform mat2 in order to make it identical to mat1 (from the previous exercise)?

```
mat2 <- matrix(data = letters, nrow = 13)
mat2
```

```
##       [,1] [,2]
##  [1,] "a"  "n"
##  [2,] "b"  "o"
##  [3,] "c"  "p"
##  [4,] "d"  "q"
##  [5,] "e"  "r"
##  [6,] "f"  "s"
##  [7,] "g"  "t"
##  [8,] "h"  "u"
##  [9,] "i"  "v"
## [10,] "j"  "w"
## [11,] "k"  "x"
## [12,] "l"  "y"
## [13,] "m"  "z"
```

```r
identical(mat1, t(matrix(mat2, nrow = 2, ncol = 13)))
```

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

3.

Extract the 2nd column from mat1 as a 13x1 matrix, not a vector. Do this for three different ways of indexing: * positive integers * negative integers * logical vector

```r
mat1[,2, drop = FALSE]
```

```
##       [,1]
##  [1,] "b"
##  [2,] "d"
##  [3,] "f"
##  [4,] "h"
##  [5,] "j"
##  [6,] "l"
##  [7,] "n"
##  [8,] "p"
##  [9,] "r"
## [10,] "t"
## [11,] "v"
## [12,] "x"
## [13,] "z"
```

```r
mat1[,-1, drop = FALSE]
```

```
##       [,1]
##  [1,] "b"
##  [2,] "d"
##  [3,] "f"
##  [4,] "h"
##  [5,] "j"
##  [6,] "l"
##  [7,] "n"
##  [8,] "p"
##  [9,] "r"
## [10,] "t"
## [11,] "v"
## [12,] "x"
## [13,] "z"
```

```r
mat1[,c(FALSE, TRUE), drop = FALSE]
```

```
##       [,1]
##  [1,] "b"
##  [2,] "d"
##  [3,] "f"
##  [4,] "h"
##  [5,] "j"
##  [6,] "l"
##  [7,] "n"
##  [8,] "p"
##  [9,] "r"
## [10,] "t"
```

```
## [11,] "v"
## [12,] "x"
## [13,] "z"
```

4.

Compare the computation time of the three indexing operations from the previous exercise. You can use R's core function for timing but you will find the microbenchmark package more informative.

```r
library(microbenchmark)
exp1 <- expression(mat1[,2, drop = FALSE])
exp2 <- expression(mat1[,-1, drop = FALSE])
exp3 <- expression(mat1[,c(FALSE, TRUE), drop = FALSE])
microbenchmark(exp1, exp2, exp3)
```

```
## Warning in microbenchmark(exp1, exp2, exp3): Could not measure a positive
## execution time for 20 evaluations.
```

```
## Unit: nanoseconds
##   expr min lq mean median uq  max neval
##   exp1   0  0    2      0  0  200   100
##   exp2   0  0   22      0  0 2000   100
##   exp3   0  0    5      0  0  500   100
```

5.

Extract the five vowels from mat1 using:

- a single index vector
- an index matrix

Again, time the computations of both operations.

```r
microbenchmark(
  mat1[c(1, 3, 5, 8, 11)],
  mat1[rbind(c(1, 1), c(3, 1), c(5, 1), c(8, 1), c(11, 1))], unit = "ns"
)
```

```
## Unit: nanoseconds
##                                                         expr  min   lq mean
##                                      mat1[c(1, 3, 5, 8, 11)]  400  500  837
##  mat1[rbind(c(1, 1), c(3, 1), c(5, 1), c(8, 1), c(11, 1))] 3200 3500 4071
##  median   uq   max neval
##     600  800 17500   100
##    3700 4100 31300   100
```

6.

What is the most efficient way to raise the first element of the following vector to the power 1, the second element to the power 2 etc. and the last element to the power 10 (you can copy-paste the following code):

```r
vec1 <- 10:1
vec1
```

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

```r
vec1^(1:10)
```

```
##  [1]    10    81   512  2401  7776 15625 16384  6561   512     1
```

7.

What is the most efficient way to raise the first element of the vector vec1 (from the previous exercise) to the power 3, the second element to the power 2, the third element again to the power 3, the fourth element again to the power 2 etc. up until the one-but-last element to the power 3 and the last element to the power 2? In other words, your result should be: [1] 1000 81 512 49 216 25 64 9 8 1

```r
vec1^rep(c(3,2), times = 5)
```

```
##  [1] 1000   81  512   49  216   25   64    9    8    1
```

```r
vec1^c(3,2)
```

```
##  [1] 1000   81  512   49  216   25   64    9    8    1
```

The first elements of the shorter vector are consecutively copied to the end of the vector until its length matches that of the longer vector. **But** R raises a warning about this only when the length of the shorter vector is not a multiple of the length of the longer vector.

8.

Consider the following matrix (you can copy-paste the code):

```r
mat3 <- matrix(seq(30, 270, by = 30), nrow = 3, ncol = 3)
mat3
```

```
##      [,1] [,2] [,3]
## [1,]   30  120  210
## [2,]   60  150  240
## [3,]   90  180  270
```

```r
mat3/c(2,3,5)
```

```
##      [,1] [,2] [,3]
## [1,]   15   60  105
## [2,]   20   50   80
## [3,]   18   36   54
```

```r
mat3/rep(c(2,3,5), each = 3)
```

```
##      [,1] [,2] [,3]
## [1,]   15   40   42
## [2,]   30   50   48
## [3,]   45   60   54
```

```r
t(t(mat3)/c(2,3,5))
```

```
##      [,1] [,2] [,3]
## [1,]   15   40   42
## [2,]   30   50   48
## [3,]   45   60   54
```

```r
sweep(x = mat3, MARGIN = 1, STATS = c(2, 3, 5), FUN = "/")
```

```
##      [,1] [,2] [,3]
## [1,]   15   60  105
## [2,]   20   50   80
## [3,]   18   36   54
```

```r
sweep(x = mat3, MARGIN = 2, STATS = c(2, 3, 5), FUN = "/")
```

```
##      [,1] [,2] [,3]
## [1,]   15   40   42
## [2,]   30   50   48
## [3,]   45   60   54
```

9.

Use a meta-function to generate all possible sums of the sides of two dice. The sides of one die can be represented as a vector (you can copy-paste the code):

```r
vec2 <- 1:6
vec2
```

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

```r
outer(vec2, vec2, FUN = "+")
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    2    3    4    5    6    7
## [2,]    3    4    5    6    7    8
## [3,]    4    5    6    7    8    9
## [4,]    5    6    7    8    9   10
## [5,]    6    7    8    9   10   11
## [6,]    7    8    9   10   11   12
```

10.

The matrix from the previous exercise can be converted to a vector. There is another meta-function which allows you to compute the vector of all (36) sums of the sides of two dice. Compare the computation times of both commands.

```r
vec3 <- as.vector(outer(vec2, vec2, FUN = "+"))
vec3
```

```
##  [1]  2  3  4  5  6  7  3  4  5  6  7  8  4  5  6  7  8  9  5  6  7  8  9 10  6
## [26]  7  8  9 10 11  7  8  9 10 11 12
```

```r
microbenchmark(as.vector(outer(vec2, vec2, FUN = "+")), unit = "ns")
```

```
## Unit: nanoseconds
##                                     expr  min   lq  mean median   uq    max
##  as.vector(outer(vec2, vec2, FUN = "+")) 8200 8500 10314   8800 9200 113600
##  neval
##    100
```

```r
microbenchmark(apply(expand.grid(vec2, vec2), MARGIN = 1, FUN = sum), unit = "ns")
```

```
## Unit: nanoseconds
##                                                   expr    min     lq   mean
##  apply(expand.grid(vec2, vec2), MARGIN = 1, FUN = sum) 173700 180950 239080
##  median     uq     max neval
##  186500 200650 1393600   100
```

11.

Compare the computation time of R's core `cumsum()` function with `Reduce( . , accumulate = TRUE)`. Do this on this vec3 (you can copy-paste the code):

```r
set.seed(123)
vec3 <- sample(vec1, size = 100, replace = TRUE)
head(vec3, n = 10)
```

```
##  [1] 8 8 1 9 5 6 7 5 2 1
```

```r
tail(vec3, n = 10)
```

```
##  [1]  5  8  3  8  3 10  4  4  4  1
```

```r
microbenchmark(cumsum(vec3), times = 100, unit = "ns")
```

```
## Unit: nanoseconds
##          expr min  lq mean median  uq   max neval
##  cumsum(vec3) 200 300  741    300 500 23900   100
```

```r
microbenchmark(Reduce(f = "+", x = vec3, accumulate = TRUE), times = 100, unit = "ns")
```

```
## Unit: nanoseconds
##                                          expr   min    lq  mean median    uq
##  Reduce(f = "+", x = vec3, accumulate = TRUE) 59500 60850 66431  62500 65250
##     max neval
##  178600   100
```

12.

Compare the computation times of three ways to sum every two consecutive elements in vector `vec3` (from the previous exercise):

- a for loop
- the vapply() meta-function
- another meta-function of the apply family, closely related to vapply()

In other words, the first element of your result 16 should the sum of the first two elements 8 and 8 in `vec3`, the second element of the result 9 should be the sum of the second element 8 and third element 1, the third element of the result 10 needs to be the sum of the third element 1 and fourth element 9 in vec3, and so on:

[1] 16 9 10 14 11 13 12 7 3 7 14 10 4 4 10 11 4 5 5 3 10 15 17 14 10

[26] 7 5 6 4 3 5 10 10 10 11 7 11 15 9 12 19 12 4 7 11 8 3 8 12 8

[51] 8 10 9 14 15 14 19 19 16 13 11 13 10 9 12 7 4 6 12 11 5 10 12 12 12

[76] 9 6 7 12 9 11 9 10 10 10 10 6 12 17 15 13 11 11 11 13 14 8 8 5

This exercise assumes some basic familiarity with creating functions in R.

```r
consecsumloop <- function(vec) {
  result <- numeric(length(vec) - 1)
  for (i in 1:(length(vec) - 1)) {
    result[i] <- vec[i] + vec[i+1]
  }
  result
}
```

```r
microbenchmark({result <- numeric(length(vec3) - 1)
  for (i in 1:(length(vec3) - 1)) {
    result[i] <- vec3[i] + vec3[i+1]
```

```r
  }
  result}, unit = "us")
```

```
## Unit: microseconds
##
##  {     result <- numeric(length(vec3) - 1)     for (i in 1:(length(vec3) - 1)) {         result[i] <-
##     min      lq     mean  median     uq     max neval
##  2985.9 3195.55 4163.804 3344.25 3867.3 20144.2   100
```

```r
microbenchmark(
  vapply(seq_len(length(vec3) - 1), FUN = function(i) sum(vec3[i:i + 1]), numeric(1)), unit = "us"
)
```

```
## Unit: microseconds
##                                                                                       expr
##  vapply(seq_len(length(vec3) - 1), FUN = function(i) sum(vec3[i:i +      1]), numeric(1))
##    min    lq   mean median     uq  max neval
##  104.1 107.7 154.57 111.45 120.85 2508   100
```

```r
microbenchmark(
  sapply(seq_len(length(vec3) - 1), function(i) sum(vec3[i:i + 1])), unit = "us"
)
```

```
## Unit: microseconds
##                                                                   expr   min
##  sapply(seq_len(length(vec3) - 1), function(i) sum(vec3[i:i +      1])) 116.5
##   lq    mean median    uq  max neval
##  121 161.271 123.05 129.8 2348   100
```