Assignment 03: Vectors and matrices

Kenji Sato*

30 March, 2017

1 Overview

Purpose

To learn how to use vectors and matrices with R. Write user defined functions.

Instructions

In this assignment, you will

- clone the assignment repository and make a working branch (eg. `solution` branch);
- solve the problems in Section 5;
- write the solutions in `R/matrix.R`, edit and knit `solution.Rmd`;
- commit `R/matrix.R`, `solution.Rmd` and `solution.pdf`; and
- open a Pull Request.

2 R quick course

2.1 Atomic vectors

A numeric vector is a tuple of numbers. In mathematics, when we write such an expression as

$$x = \begin{bmatrix} 2.1 \\ 1.2 \\ 4.3 \end{bmatrix}$$

we understand that this x is a vector in the three-dimensional vector space (it can be $\mathbb{R}^3$, $\mathbb{C}^3$ or $\mathbb{Q}^3$).

*Kobe University. mail@kenjisato.jp

In R, we use `c()` function to concatenate the three numbers.

```
x <- c(2.1, 1.2, 4.3)
x
```

```
## [1] 2.1 1.2 4.3
```

An object like `x` is called an atomic vector. All elements have a common type and are aligned flat and straight.

To read a value in the vector, we use the indexing operator `[`.[1] Note that the index starts from 1.[2]

```
x[1]
```

```
## [1] 2.1
```

Similarly,

```
x[2]
```

```
## [1] 1.2
```

We can use multiple subscripts

```
x[c(2, 3)]
```

```
## [1] 1.2 4.3
```

and negative subscript

```
x[-1]
```

```
## [1] 1.2 4.3
```

Negative multiple subscripts works just fine but you cannot mix positive and negative subscripts.

```
x[c(1, -2)]
```

```
## Error in x[c(1, -2)]: only 0's may be mixed with negative subscripts
```

2.2 Scalars

In R, there is no such thing as a scalar value. A single number is a vector of size one.

```
1 == c(1)
```

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

[1] This is just a R function. Try `` `[`(x, 1) `` in the console.

[2] Notice that the n-th element of x in this example has n after the decimal point. I owe this idea to Hadley Wickham. See http://adv-r.had.co.nz/Subsetting.html.

The following code might seem a bit awkward at first glance.

```
length(0)
```

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

2.3 Arithmetics

Addition and subtraction of two (or more) vectors with same size work as we expect.

```
y <- c(-2, -1, -4)
x + y
```

```
## [1] 0.1 0.2 0.3
```

Multiplication and division are performed element wise.

```
x * y
```

```
## [1]  -4.2  -1.2 -17.2
```

2.4 Column and row vectors

In mathematics,

$$x = \begin{bmatrix} 2.1 \\ 1.2 \\ 4.3 \end{bmatrix}$$

and

$$x = \begin{bmatrix} 2.1 & 1.2 & 4.3 \end{bmatrix}$$

are usually considered different. The first is called a column vector and the latter a row vector. `c(2.1, 1.2, 4.3)` doesn't have row-column distinction.

To get a column vector, use `matrix()` function

```
xcol <- matrix(x)
xcol
```

```
##      [,1]
## [1,]  2.1
## [2,]  1.2
## [3,]  4.3
```

To get a row vector, do the following

```
xrow <- matrix(x, nrow = 1)
xrow
```

```
##      [,1] [,2] [,3]
## [1,]  2.1  1.2  4.3
```

Alternatively, you can do something like this. `t()` function returns the transpose matrix.

```
t(xcol)
```

```
##      [,1] [,2] [,3]
## [1,]  2.1  1.2  4.3
```

```
xcol * xrow
```

```
## Error in xcol * xrow: non-conformable arrays
```

```
xrow * xcol
```

```
## Error in xrow * xcol: non-conformable arrays
```

The %*% operator works as a matrix multiplication.

```
xrow %*% xcol
```

```
##       [,1]
## [1,] 24.34
```

2.5 Euclidean norm

To compute the length (norm) of a vector, you can use

```
sqrt(sum(x ^ 2))
```

```
## [1] 4.933559
```

The above code works for row and column vectors too.

```
sqrt(sum(xrow ^ 2))
```

```
## [1] 4.933559
```

2.6 Matrices

Column and row vectors are matrices with a single column and row, respectively. General matrices can be defined similarly.

```
elm_colwise <- c(1.11, 3.21, -5.31, 2.12, -6.22, 0.32)
matrix(elm_colwise, nrow = 3)
```

```
##       [,1]  [,2]
## [1,]  1.11  2.12
## [2,]  3.21 -6.22
```

```
## [3,] -5.31  0.32
```

Notice that by default `matrix()` function fills elements of the passed atomic vector by column. To change this behavior, pass `byrow = TRUE` as parameter.

```
elm_rowwise <- c(
     1.11,  2.12,
     3.21, -6.22,
    -5.31,  0.32
  )
matrix(elm_rowwise, nrow = 3, byrow = TRUE)
```

```
##       [,1]  [,2]
## [1,]  1.11  2.12
## [2,]  3.21 -6.22
## [3,] -5.31  0.32
```

Addition, subtraction and multiplication are defined for conformal matrices.

```
set.seed(1)
m1 <- matrix(rnorm(9), nrow = 3)
m2 <- matrix(rnorm(9), nrow = 3)
```

Addition:

```
m1 + m2
```

```
##            [,1]       [,2]      [,3]
## [1,] -0.9318422  0.9740402 0.4424954
## [2,]  1.6954245 -1.8851921 0.7221344
## [3,] -0.4457854  0.3044625 1.5196176
```

Subtraction:

```
m1 - m2
```

```
##            [,1]      [,2]       [,3]
## [1,] -0.3210654  2.216521  0.5323627
## [2,] -1.3281378  2.544208  0.7545150
## [3,] -1.2254718 -1.945399 -0.3680549
```

Multiplication:

```
m1 %*% m2
```

```
##            [,1]        [,2]      [,3]
## [1,]  2.7930481 -2.59556567 0.4623740
## [2,]  0.7298920 -0.01328322 0.6832710
## [3,] -0.7607129  2.98393189 0.5942747
```

2.7 Comparison

You must not use `==` to check if two non-integer values are identical.

While you can safely write

```
1 == 2 - 1
```

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

you cannot expect that the following comparison returns `TRUE`.

```
0.3 == 0.1 + 0.1 + 0.1
```

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

When you compare decimal numbers (floating point numbers to be precise), alway use functions such as `all.equal()`.

```
all.equal(0.3, 0.1 + 0.1 + 0.1)
```

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

This approach works just fine for matrices and vectors.

```
all.equal(m1, m2) == TRUE
```

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

3 Exercise 1: User defined functions

If you find yourself writing an identical chuck of code over and over again, you should start writing a function. You can reduce the possibility of making bugs by abstracting common tasks because maintenance of codes becomes easier.

As an exercise, write a user defined function `norm()` to compute Euclidean norm computed above. Ensure that the following code returns the norm:

```
norm(x)  #> Should return 4.933559
```

Solution

```
norm <- function(x) {
  sqrt(sum(x ^ 2))
}
```

4 Exercise 2: `source()` external file

Start writing functions in a Rmd file is not always advisable. Since reporting is usually the final stage of your research, you want to start experimentation in a simpler and handier format. The places you might want to write your codes are files with the extension `.R`.

Look into `R/vector.R` and you can find the `norm()` function defined therein. You can use this function in `R/vector.R` by

```
source('R/vector.R')
```

and then

```
norm(x)
```

```
## [1] 4.933559
```

I recommend to put all `.R` files in the folder named `R` with no subfolders below.[3]

[3] When you write a R package, you need to follow this rule.

5 Problems

Edit two files

- `solution.Rmd`
  - Write your name. That's it.
- `matrix.R`
  - Write `is_symmetric()` function that meets the specifications given below.

and knit `solution.Rmd`. If you have a rendered PDF with no warnings, you are done. Commit, push and send a Pull Request as always.

Specifications

Function `is_symmetric()`

- receives a square matrix `x` as the unique parameter,
- returns `TRUE` for a length-1 vector (scalar),
- returns `TRUE` if `x` is a symmetric matrix and `FALSE` otherwise.

Although you may assume that the parameter `x` is alway square (probably because input validation is done outside of the function), you can make an extra effor to validate that `x` is really square within this function.

This function is similar to `base::isSymmetric()`.