Basic concepts with R (part 3)

Rodrigo Esteves de Lima-Lopes
State University of Campinas
rll307@unicamp.br

Contents

1 Introduction

In this tutorial we are going to discuss some data data structure in R. Naturally, vectors are very important, but we need other kind of data if we are going to study language data. Here we are going to discuss matrices.

2 Matrices

A matrix is a bi-dimensional rectangular data structure, like a more complex vector. In a matrix, all data have to be on the same format: a matrix contains only characters, or numbers etc. The command we usually create a matrix is:

`matrix(data, nrow, ncol, byrow, dimnames)`

These parameters are:

- **data**: an input vector.
- **nrow**: the number of rows.
- **ncol**: the number of columns.
- **byrow**: logical clue, *TRUE*: input is arranged by row.
- **dimname**: names assigned to rows columns.

Let us see an example of matrix:

```
columns.names <- c('col1','col2', 'col3')
rows.names <- c('row1','row2','row3','row4','row5')
My.Matrix <- matrix(c(1:15), nrow = 5, byrow = TRUE, dimnames = list(rows.names, columns.names))
My.Matrix.2 <- matrix(c(1:15), nrow = 5, byrow = FALSE)
My.Matrix.3 <-matrix(c(1:15), nrow = 5, ncol=5, byrow = TRUE)
typeof(My.Matrix)
```

```
## [1] "integer"
```

```
str(My.Matrix)
```

```
##  int [1:5, 1:3] 1 4 7 10 13 2 5 8 11 14 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:5] "row1" "row2" "row3" "row4" ...
##   ..$ : chr [1:3] "col1" "col2" "col3"
```

We learnt a couple of things from the code above:

1. The operator ":" makes a numerical sequence in *R*
2. The option `byrow = TRUE` distributes the elements by lines

- While `byrow = FALSE` distributes the elements by columns

1. If I create a matrix which needs more that than my input, *R* recycles the data.
2. Vectors were used as input in a function

We can also create a character matrix:

```
columns.names <- c('col1','col2', 'col3')
rows.names <- c('row1','row2','row3','row4','row5')
my.data.1 <- rep("test", 15)
my.data.2<-1:15
my.data.3 <- paste0(my.data.1,my.data.2)
My.Matrix.4 <- matrix(my.data.3, nrow = 5, byrow = TRUE, dimnames = list(rows.names, columns.names))
str(My.Matrix.4)
```

```
##  chr [1:5, 1:3] "test1" "test4" "test7" "test10" "test13" "test2" "test5" ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:5] "row1" "row2" "row3" "row4" ...
##   ..$ : chr [1:3] "col1" "col2" "col3"
```

```
typeof(My.Matrix.4)
```

```
## [1] "character"
```

As vectors, we can also access elements in a matrix, but each element is now identified for two indexes:

```
My.Matrix[3,1]
```

```
## [1] 7
```

```
My.Number<-My.Matrix[3,1]
My.Matrix.4[3,1]
```

```
## [1] "test7"
```

The first index refers to the row, while the second to the column. If I leave the number before the coma blank, I will be extracting all elements of a row. If I leave the number after the coma blank, I will be extracting all elements of a column.

```
My.Matrix[,1]
```

```
## row1 row2 row3 row4 row5
##    1    4    7   10   13
```

```
My.Matrix[1,]
```

```
## col1 col2 col3
##    1    2    3
```

It is easier to understand if I print this:

```
My.Matrix.3
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]    1    2    3    4    5
## [5,]    6    7    8    9   10
```

```
My.Matrix.3[,1]
```

```
## [1]  1  6 11  1  6
```

```
My.Matrix.3[1,]
```

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

I can do all sort of operations with matrices, just as I do with vectors:

```
print(My.Matrix - My.Matrix.2)
```

```
##      col1 col2 col3
## row1    0   -4   -8
## row2    2   -2   -6
## row3    4    0   -4
## row4    6    2   -2
## row5    8    4    0
```

```
print(My.Matrix + My.Matrix.2)
```

```
##      col1 col2 col3
## row1    2    8   14
## row2    6   12   18
## row3   10   16   22
## row4   14   20   26
## row5   18   24   30
```

```
sqrt(My.Matrix)
```

```
##          col1     col2     col3
## row1 1.000000 1.414214 1.732051
## row2 2.000000 2.236068 2.449490
## row3 2.645751 2.828427 3.000000
## row4 3.162278 3.316625 3.464102
## row5 3.605551 3.741657 3.872983
```

Note that such operations are recursively. Operations on matrices of different size might through an error.