Introduction, R installation, basic data types and structures, operations

—

B83128 – Introduction to R scripting language (shortened version)

Lubomír Štěpánek[1, 2]

[1]Department of Biomedical Statistics
Institute of Biophysics and Informatics
First Faculty of Medicine
Charles University, Prague

[2]Department of Biomedical Informatics
Faculty of Biomedical Engineering
Czech Technical University in Prague

Content

Subject organization

- elective subject (3 credit points), shortened, online
- ended by a seminar project making you credit-eligible (data analysis with R, 1–2 pages of your comments about the analysis and results)
- expected syllabus
  - (i) installation, data types and structures, operations
  - (ii) if-else conditions, cycles, warnings, data import and export
  - (iii) in-built and user-defined functions
  - (iv) exploratory data analysis, hypotheses testing
  - (v) analysis of variance, correlation, linear regression
  - (vi) logistic regression and other models

github repo ▸ GitHub

https://github.com/LStepanek/B83128_Introduction_to_R_scripting_language

What is R

- R is an interpreted programming language
- it combines several paradigms
  - imperative one
  - functional one
  - object one
- R is *domain specific language* – it is primarily dedicated for statistical analyses and ongoing graphical visualizations
- R is open-source, both *free-as-in-beer* and *free-as-in-speech*

Download and installation of R

- on webpages of R-project

  https://www.r-project.org/

  search for **download R**, and download to your desktop
- afterward, follow the instructions and install locally

Download and installation of RStudio

- RStudio is a graphical IDE (Integrated Development Environment) for R language
- on webpages of RStudio

https://www.rstudio.com/

follow **Products** > **RStudio** > **Desktop** > **Open Source Edition** > **Free** > **Download**, download to your desktop

- afterward, follow the instructions and install locally

Hello world!

- write in your script or directly into the console

```
1 print("hello world")
```

- you are getting

```
1 [1] "hello world"
```

Usage of help

- we can get a help for each object or function using `help()`, where a name of the object or function is the argument

```
help(print)
```

- or by typing of `?` before a name of the object or function

```
?print
```

- by typing of `??` before a name of the object or function, we search through all help files

```
??print
```

- consequently, as an output, HTML file with a plain-text help is returned

Data types

- a „real“ number value (`numeric`)
- an integer (`integer`)
- a logical value (`logical`)
- a string (`character`)
- `NA`, `NULL`, `NaN`

A „real“ number value

- in R as `numeric` type
- $x \in \mathbb{R}$ limited by a float precision
- similar to `double` double type with 64 bit precision
- for example

```
5; -13.8; 4.5578e15
```

- we can check whether a value is `numeric` using

```
is.numeric(-13.8)     # TRUE
class(-13.8)          # "numeric"
class(Inf)            # "numeric"
```

- ideal for representation of real numbers (real data)

An integer

- in R as `integer` type
- $z \in \mathbb{Z}$ limited by a bit precision
- for example

```
5L; 13L; -5L
```

- we can check whether a value is `integer` using

```
is.integer(-13L)    # TRUE
class(-13L)         # "integer"
is.integer(-13)     # FALSE
class(-13)          # "numeric"
```

- a coercion of an integer to `numeric` by

```
as.numeric(5L)
```

A logical value

- in R as `logical` type
- boolean $x \in \{\texttt{TRUE}, \texttt{FALSE}\}$
- for example

```
1 TRUE; FALSE; T; F
```

- we can check whether a value is `logical` using

```
1 is.logical(TRUE)      # TRUE
2 class(FALSE)          # "logical"
3 class("TRUE")         # "character"
4 class(T)              # "logical"
5 class(F)              # "logical"
```

A string

- in R as `character` type
- a sequence of symbols (extended ASCII) bounded by simple of double quotes
- for example

```
"hello"; 'xweiwogw23425ng'; ""
```

- we can check whether a value is `character` using

```
is.character("ahoj")    # TRUE
class("blah blah")      # "character"
class("123")            # "character"
class(123)              # "numeric"
is.numeric(Inf)         # TRUE
is.numeric("Inf")       # FALSE
```

- a coercion of other data type to `character` by

```
as.character(123)
```

NA, NULL, NaN

- NA is a value of Not Available, i. e. usually a missing value
- NULL is a null object, it is used for initialization of an empty object
- NaN is a value of Not a Number, i. e. usually a result of an incorrect math operation
- it holds that $\{\texttt{NaN}\} \subseteq \{\texttt{NA}\}$
- for example

```
log(-1)              # NaN
is.na(NaN)           # TRUE
is.nan(NA)           # FALSE
is.nan(1 / 0)        # FALSE
1 / 0                # Inf
```

Values assigning

- we can assign a value to another simply by an equation mark

```
1         x = 5
```

- or by an oriented arrow

```
1         x <- 5
2         5 -> x        # the same as above
```

- or using a function `assign()`, where the first argument is a name of the new variable, and the second one is its value

```
1         assign("x", 5)   # equivalent to x <- 5 or x = 5
```

Data structures

- a vector (`vector`)
- a factor (`factor`)
- a matrix (`matrix`)
- a data frame (`data.frame`)
- a list (`list`)

Vector initialization and basic commands

- a vector is a one-dimensional sequence of values of the same data type, it does not have a prior column or row orientation
- a vector is a type of *tuple*, i. e. it respects an order of its values (contrary to *set* data type)
- a vector could be created using `c()` function, where *c* stands for *concatenate*
- for example

```
1 c()                # an empty vector
2 length(c())        # 0
3 c(3, 1, 2)         # a vector of length 3 and values 3, 1, 2
4 c("a", "d")        # a vector of length 2 and values "a", "d"
```

- using the `c()` function, we can combine existing vectors and make longer ones

```
1 c(c(3, 1, 2), 4)   # a vector of values 3, 1, 2, 4
2 c(3, 1, 2, 4)      # the same as above
```

Vector initialization and basic commands

- one or more values could be added to a vector

```
1   x <- c(3, 1, 2)
2   length(x)          # 3
3   y <- 1
4   z <- c(2)
5   w <- c(5, 7)
6   x <- c(x, y)       # a vector x extended
7                      # by a value y
8   w <- c(w, z)       # a vector w extended
9                      # by a vector z
10  c <- c(1, 2, 3)
11  c                  # both a vector
12                     # and a function
13                     # could have the same name "c"
14                     # in one R session
```

Vectors of characters

- vectors could consist of `character` values, they could be used e. g. as names for other vectors

```
x <- c(3, 1, 2)
y <- c("a", "b", "c")
names(x) <- y      # values of vector x
                   # are named using y
x
unname(x)          # names of the vector x
                   # are deleted
setNames(x, y)     # again, values of the vector x
                   # are named using y
```

Subvectors, indexing, addressing

- R indexes vectors starting by 1, not by 0 (index of the first value is 1, index of the second value is 2, etc.)
- we address using brackets [ ]

```
x <- c(4, 2, 6, -3)
x[1]                # 4
x[1:2]              # c(4, 2)
x[5]                # NA
x[length(x)]        # -3
x[c(1, 3, 4)]       # c(4, 6, -3)
x[length(x):1]      # c(-3, 6, 2, 4)
rev(x)              # the same as above,
                    # c(-3, 6, 2, 4)
```

Logical vectors

- they could be used for addressing of other vectors

```
y <- c(TRUE, TRUE, FALSE, TRUE) # logical
                                # vector
x <- c(3, 1, 2, 5)
x[y]                        # (sub)vector c(3, 1)
x[c(F, T, F, T)]            # subvector c(1, 5)
```

- sometimes, a *recycling* is very useful

```
z <- c("R", "G", "E", "F", "I")
z[c(T, F)]      # it picks only values
                # with even indices,
                # i. e. "R", "E", "I"
                # or in other words, a vector of
                # c("R", "E", "I")
```

Factors

- vectors of character values so that each value belongs to one of a few categories

```
x <- factor(
    c("male", "female", "male", "male")
)           # an order of the levels (categories)
            # is apriori alphabetical
x <- factor(
    c("male", "female", "male", "male"),
    levels = c("male", "female")
)           # we can manage the order
            # of the levels (categories)
```

- we can simply create a pivot table using factors

```
table(x)    # x
            # male female
            # 3 1
```

Arithmetic operations

operation	operator	example
summation	+	2 + 3
subtraction	-	2 - 3
multiplication	*	2 * 3
division	/	2 / 3
exponentiation	^ or **	2 ^ 3 or 2 ** 3
modulo[1]	%%	7 %% 3
integer division	%/%	7 %/% 3

[1] a remainder by an integer division

Logical operations

- logical operations could be used for predicates, i. e. objects of a data type `logical`
- operation *short* AND (operator `&`)
  - useful for vectors
  - it evaluates the operation one-by-one value pairwisely using both vectors

```
1       c(FALSE, FALSE, TRUE, TRUE) &
2       c(FALSE, TRUE, FALSE, TRUE)
3                       # c(FALSE, FALSE, FALSE, TRUE)
```

- operation *short* OR (operator `|`)
  - useful for vectors, it evaluates the operation one-by-one value pairwisely using both vectors

```
1       c(FALSE, FALSE, TRUE, TRUE) |
2       c(FALSE, TRUE, FALSE, TRUE)
3              # c(FALSE, TRUE, TRUE, TRUE)
```

Logical operations

- operation NOT (operator `!`)
  - it returns an opposite boolean value to the given value

```
! TRUE                    # FALSE
! 2 > 3                   # TRUE
```

- function `all()`
  - it returns TRUE if and only if all the values are TRUE

```
all(c(3 > 2, 7 %% 3 == 1, 1 == 0))   # FALSE
all(c(3 > 2, 7 %% 3 == 1, 1 >= 0))   # TRUE
```

- funkce `any()`
  - it returns TRUE if at least one of the values is TRUE

```
any(c(3 < 2, 7 %% 3 <= 0, FALSE))    # FALSE
any(c(3 < 2, 7 %% 3 >= 1, FALSE))    # TRUE
```

Operation of comparation

- we can compare two objects of any length and values order
- as an output, we get a value of a data type `logical`
- a comparison of (==, `all.equal()`, `identical()`)

```
2 == 3                          # FALSE
all.equal(c(1, 2), c(1, 2 + 1e-13),
          tolerance = 1e-12)
                                # TRUE; it respects
                                # the given numerical
                                   tolerance
identical(c(1, 2), c(1, 2 + 1e-13))
                                # FALSE, it returns
                                # TRUE if and only if
                                # both the objects are
                                # totally the same
```

Operation of comparation

- a comparison of *is less than*, *is less than or equal to*, *is greater than*, *is greater than or equal to* (<, <=, >, >=)

```
2 < 3                    # TRUE
"b" <= "a"               # FALSE; it compares
                         # an order within English
                           alphabet
FALSE >= TRUE            # FALSE; it compares
                         # values in boolean algebra
                         # (TRUE := 1, FALSE := 0)
```

- a comparison of *not equal* (!=)

```
2 != 3                   # TRUE
TRUE != FALSE            # TRUE
```

Operation of comparation

- a comparison of type *is in* (%in%)

```
c(2, 6) %in% c(1:5)          # c(TRUE, FALSE)
"k" %in% LETTERS             # FALSE
"J" %in% letters             # FALSE
"May" %in% month.name        # TRUE
"a" %in% "alphabet"          # FALSE
```

- an equivalent (a wrapper) to an operation %in% is a function `is.element()`

```
is.element(c(2, 6), c(1:5))
                             # c(TRUE, FALSE)
is.element(c(1:5), c(2, 6))
                             # c(FALSE, TRUE,
                             # FALSE, FALSE, FALSE)
```

In-built math functions

- functions of packages base, stats and others
- for example

```
1  abs(), sign()
2  acos(), asin(), atan()
3  sin(), cos(), tan()
4  ceiling(), floor(), trunc()
5  exp(), log(), log10(), log2(), sqrt()
6  max(), min(), prod(), sum()
7  cummax(), cummin(), cumprod(), cumsum(),
       diff()
8  pmax(), pmin()
9  range()
10 mean(), median(), cor(), sd(), var()
11 rle()
```

Rounding and formatting of numbers

- rounding of a number `x` using `round(x, digits)` with respect to `digits` of decimal digits

```
round(1.4, digits = 0)      # 1
round(-146.655, 2)          # -146.66
```

- rounding of a number `x` using `signif(x, digits)` with respect to `digits` of significant digits

```
signif(1.458, digits = 1)   # 1
signif(1.458, digits = 2)   # 1.5
signif(1.458, digits = 3)   # 1.46
signif(1.458, digits = 4)   # 1.458
```

- formatting of a number `x` using `format(x, nsmall)` with respect to `nsmall` of fixed decimal digits

```
format(1.45, nsmall = 1)    # "1.45"
format(1.45, nsmall = 2)    # "1.45"
format(1.45, nsmall = 3)    # "1.450"
```

Matrices' initialization and basic commands

- a matrix (`matrix`) is a two-dimensional array containing values of (only) one data type
- all columns of a matrix are of one length, and all rows of a matrix are of one length
- let

$$\boldsymbol{A} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

- in R, we get the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ by

```
1  A <- matrix(c(1, 2, 3, 4), nrow = 2,
2              ncol = 2)
3  B <- matrix(c(1, 3, 2, 4), nrow = 2,
4              ncol = 2)
5  B <- matrix(c(1, 2, 3, 4), nrow = 2,
6              ncol = 2, byrow = TRUE)
7  # only one of the arguments "nrow" and "ncol" is necessary
```

Manipulation with matrices

- let

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- in R using

```
C <- matrix(letters[1:12], nrow = 3,
            byrow = T)
```

- some useful commands are

```
is.matrix(C)    # TRUE
class(C)        # "matrix"
mode(C)         # "character"; data type of matrix
                # values
str(C)          # chr [1:3, 1:4] "a" "e" "i" ...
dim(C)          # c(3, 4); dimensions of matrix C
```

Manipulation with matrices

- let

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- other bunch of useful commands

```
colnames(C) <- c("c1", "c2", "c3", "c4")
rownames(C) <- c("r1", "r2", "r3")
                # adds labels to columns and rows
C <- unname(C)
                # deletes the labels of columns
                # and rows
dimnames(C) <- list(
                    c("r1", "r2", "r3"),
                    c("c1", "c2", "c3", "c4")
                )
                # also adds labels to columns and rows
```

Manipulation with matrices

- let's have

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- another bunch of useful commands

```
1  rbind(C, c("x", "x", "x", "x"))
2                  # adds a row of c("x", "x", "x", "x")
3                  # to the matrix C
4  cbind(C, c("x", "x", "x"))
5                  # adds a column of c("x", "x", "x")
6                  # to the matrix C
7  C[-1, ]         # deletes the 1-st row of the matrix C
8  C[, -2]         # deletes the 2-nd column
9                  # of the matrix C
```

Submatrices, indexing, addressing

- let

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- in R by

```
1  C <- matrix(letters[1:12], nrow = 3,
2              byrow = T, dimnames = list(
3              c("r1", "r2", "r3"),
4              c("c1", "c2", "c3", "c4")))
5  C[2, 3]          # "g"; a value of the 2-nd row
6                   # and the 3-st column
7  C["r2", "c3"]    # "g"; a value of the 2-nd row
8                   # and the 3-st column
9  C[1, ]           # c("a", "b", "c", "d");
10                  # a vector of the 1-st row of the matrix C
11                  # with labels
```

Submatrices, indexing, addressing

- still let's have

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- addressing

```
C[, 3]              # c("c", "g", "k");
                    # a vector of the 3-rd column
                    # of the matrix C with labels
C[c(1, 3), c(2, 4)]
                    # matrix(c("b", "j", "d", "l"), 2)
                    # a submatrix of the 1-st and 3-rd rows,
                    # 2-nd and 4-th column of the matrix C
                    # with labels
C["r2", ]           # c("e", "f", "g", "h");
                    # a vector of the 2-nd row
                    # of the matrix C with labels
```

Submatrices, indexing, addressing

- let's have

$$\boldsymbol{C} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

- addressing

```
C[dim(C)[1], dim(C)[2]]
                # "l"; a general addressing
                # of the right bottom page
C[5]            # "f"; major-column ordering
C[c(8, 9)]      # c("g", "k")
C[13]           # NA
diag(C)         # c("a", "f", "k"); main diagonal
diag(C[, dim(C)[2]:1])
                # c("d", "g", "j"); opposite diagonal
```

Matrix algebra

- let

$$\boldsymbol{A} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} 5 & 7 \\ 6 & 8 \end{pmatrix}$$

- in R using

```
A <- matrix(c(1, 2, 3, 4), nrow = 2)
B <- matrix(c(5, 6, 7, 8), nrow = 2)
```

- Hadamard's product (*element-wise*, *pairwise*) $\boldsymbol{A} \circ \boldsymbol{B} = \begin{pmatrix} 5 & 21 \\ 12 & 32 \end{pmatrix}$

```
A * B       # matrix(c(5, 12, 21, 32), 2)
```

- matrix product $\boldsymbol{A} \cdot \boldsymbol{B} = \begin{pmatrix} 23 & 31 \\ 34 & 46 \end{pmatrix}$

```
A %*% B     # matrix(c(23, 34, 31, 46), 2)
```

Matrix algebra

- let

$$\boldsymbol{A} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} 5 & 7 \\ 6 & 8 \end{pmatrix}$$

- in R using

```r
A <- matrix(c(1, 2, 3, 4), nrow = 2)
B <- matrix(c(5, 6, 7, 8), nrow = 2)
```

- transposition $\boldsymbol{A}^T = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

```r
t(A)          # matrix(c(1, 3, 2, 4), 2)
```

Thank you for your attention!

lubomir.stepanek@lf1.cuni.cz
lubomir.stepanek@fbmi.cvut.cz

▸ GitHub

https://github.com/LStepanek/B83128_Introduction_to_R_scripting_language