



R programming for statistics

Aleš Kozubík
University of Žilina



**Project: Innovative Open Source Courses
for Computer Science**



31. 5. 2021



Co-funded by the
Erasmus+ Programme
of the European Union

- 1 Introduction into R environment
- 2 Data structures in R
- 3 Probability distributions in R
- 4 Programming in R
- 5 Elementary graphics
- 6 Descriptive sample characteristics
- 7 Parameter estimates

Innovative Open Source Courses for Computer Science



This teaching material was written as one of the outputs of the project
“Innovative Open Source Courses for Computer Science”,
funded by the Erasmus+ grant no. 2019-1-PL01-KA203-065564.

The project is coordinated by West Pomeranian University of Technology in Szczecin (Poland)
and is implemented in partnership with Mendel University in Brno (Czech Republic)
and University of Žilina (Slovak Republic).

The project implementation timeline is September 2019 to December 2022.

Innovative Open Source Courses for Computer Science

Project was implemented under the Erasmus+.

Project name: “[Innovative Open Source courses for Computer Science curriculum](#)”

Project no.: [2019-1-PL01-KA203-065564](#)

Key Action: [KA2 – Cooperation for innovation and the exchange of good practices](#)

Action Type: [KA203 – Strategic Partnerships for higher education](#)

Consortium: Zachodniopomorski uniwersytet technologiczny w Szczecinie
Mendelova univerzita v Brně
Žilinská univerzita v Žiline

Erasmus+ Disclaimer: This project has been funded with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Copyright Notice: This content was created by the IOSCS consortium: 2019–2022.
The content is Copyrighted and distributed under Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).



R programming for statistics

I. Introduction into R environment

Characteristics of the R language



- it is a language and free software environment specialized in statistical computation and data visualisation,
- it is accessible for all common OS: UNIX platforms, Win or MacOS,
- it is an alternative to the commercial tool S resp. S-plus (language and environment), developed by AT&T.

Why R

- it is free, most of the statistical software platforms cost thousands of dollars,
- the program has a huge collection of available packages,
- R can easily import data from a wide variety of sources,
- R contains numerous advanced statistical routines,
- R provides interactive platform for data analysis,
- R environment offers data visualisation in the form very high quality and aesthetic graphs,
- it is platform independent, it is compatible with all most frequently used operating systems,
- it is highly compatible with the programming languages like C,C++, Python, Java.

Installing R

It is freely available from the Comprehensive R Archive Network (shortly CRAN)

The internet location is <https://cran.r-project.org>

Here are at disposal pre-compiled binaries for all common platforms Linux, Mac OS, and Windows.

You can select the most suitable mirror for downloading the installation package.

Installing R packages

R comes with a huge set of packages extending its base core.

They increase the power of R.

To install package we use the function `install.packages()`

R first run

Once we have installed R, we can try if it works correctly.

We start the R environment interface simply from command prompt typing:

```
username@host:~$ R
```

It displays a short introductory note that is followed by the sign

```
>
```

assigning the R prompt.

Leaving the R environment

The R environment is now ready for an interactive mode of work.

In order to finish work in R environment we simply type

```
> q()
```

R reacts by question:

```
Save workspace image? [y/n/c]:
```

If we select `y`, the whole working history is saved in the file `.Rhistory` saved in the working directory.

Workspace and navigation

We enter all command interactively at the command prompt.

Scrolling through the commands history is enabled by using the up and down arrow keys.

This allows to submit a previous commands without retyping it. We only select the desired and submit it repeatedly using the key.

If we saved the history when leaving R environment, we can return to the commands from previous session.

Communication with the OS

The default working directory is the directory where R was started. In this current working directory R reads and saves files and results. The actual working directory we can find using the `getwd()` function.

The current working directory can be changed using the `setwd()` function.

To run the OS commands we apply the `system()` function.

To create a new directory

```
> system("mkdir new")
```

Getting help

The general function for getting help has a simple form `help()`, or shortly in the operator form `?`.

To get some information about the additional packages, we use

```
> help(package="package name")
```

Some packages include also code demonstrations, we run using function `demo()`

```
> demo(package="stats")
```

Using R as calculator

Console prompt enables interactively compute the operations and functions

```
> 5+3  
[1] 8
```

If we don't see new prompt, it may be because we entered an incomplete command

```
> 5-  
+
```

We have to type the rest of the command and then press `Enter` or cancel the command pressing `Esc` key.

Arithmetic operators in R

- + Addition.
- Subtraction.
- * Multiplication.
- / Division.
- ^ Exponentiation.
- %% Modulus (Remainder from integer division).
- %%/ Integer division.

Relational operators in R

- < Less than
- > Greater than
- <= Less than or equal to
- >= Greater than or equal to
- == Equal to
- != Not equal to

Frequently used math functions

<code>exp()</code>	Exponential	<code>sqrt()</code>	Square root
<code>log()</code>	Logarithm (default natural)	<code>abs()</code>	Absolute value
<code>log10()</code>	Logarithm with base 10		
<code>sin()</code>	Sine	<code>asin()</code>	Arc sine
<code>cos()</code>	Cosine	<code>acos()</code>	Arc cosine
<code>tan()</code>	Tangent	<code>atan()</code>	Arc tangent
<code>round()</code>	Rounding (default to integer)		

Objects

R is object-oriented language

Everything in R is an object and it represents some data that has been stored in memory

Objects can be given any name, rules that must be respected:

- the name consists only of lower or upper case letters, numbers, underscores and dots,
- the name begins with upper or lower case letter,
- R is case sensitive (it means `A` and `a` are two different objects),
- the name must not be any of the R's reserved words (the list of them is visible after entering `help(reserved)`),

Creating objects

We create a new object simply with the assignment operator

The assigning operator has two possible forms: `<-` or `=`.

It is recommended to use `<-` as `=` can sometimes lead to errors:

```
> log(x=25,base=5)
```

```
[1] 2
```

```
> x
```

```
Error: object 'x' not found
```

```
> log(x<-25,base=5)
```

```
[1] 2
```

```
> x
```

```
[1] 25
```

Listing and removing objects

The list of all created object we obtain as an answer of the `ls()` function.

The objects we will not use in the future can be removed from the memory using the `rm()` function.



R programming for statistics

II. Data structures in R

Data types and structures

Here exists 5 elementary data types

- numeric,
- integer,
- complex,
- logical,
- character.

They can be aggregated into the data structures:

- vector,
- matrix,
- list,
- frame.

Data type `numeric`

The data type `numeric` represents the real decimal numbers.

It is the default type of each new object

If we assign to any variable the decimal

The type of any object we recognise using function `class()`

Data type `numeric` – an example

Let us see the example.

```
1 > x<-12.35
2 > class(x)
3 [1] "numeric"
```

Note

Number is represented as vector with length 1. Sign `[1]` means the first position in the vector.

Data type integer

To create the object of the integer type, we use the function `as.integer()`

Example

```
> a <- as.integer(12)
> a
[1] 12
> class(a)
[1] "integer"
> is.integer(a)
[1] TRUE
```

Change of the type

When making any computations, it is important to keep in mind that a variable may be retyped.

Example

```
1 > x<-as.integer(20)
2 > class(x)
3 [1] "integer"
4 > x<-x/3+1
5 > x
6 [1] 7.666667
7 > class(x)
8 [1] "numeric"
```

Data type complex

R environment provides also the possibility to work with complex numbers

The complex value is in R defined via the imaginary unit i

Example

```
> z<-1+2i
> class(z)
[1] "complex"
```

Data type logical

It can have two logical values TRUE or FALSE

It is frequently created via comparison between variables

```
1 > x<-10;y<-20
2 > z<-x<y
3 > z
4 [1] TRUE
5 > class(z)
6 [1] "logical"
```

Data type logical

Here are defined all standard logical operations

- & Logical AND
- | Logical OR
- ! Negation

Data type character

It is used to store the string values, strings are entered using the quotation marks

```
1 > x<-"facina"
2 > class(x)
3 [1] "character"
4 #But as well
5 > x<-as.character(3.1415926)
6 > x
7 [1] "3.1415926"
8 > class(x)
9 [1] "character"
```

Data type character

The character type objects can be concatenated using the `paste()` function

```
1 > name<-"Donald"
2 > surname<-"Knuth"
3 > paste(name,surname)
4 [1] "Donald_Knuth"
5 # To add any separator
6 > paste(name,surname,sep=",")
7 [1] "Donald,Knuth"
```


Vectors

Vector is the simplest data structure

It can be characterised as a sequence of data elements of the same basic type

The single values contained in the vector are referred as components

The number of components of the vector is referred as its length.

Vectors

Vector v is created by combine function `c()`

Its length we get using the `length()` function

```
1 > v<-c(1,3,5,7,9)
2 > length(v)
3 [1] 5
```

Vectors – the arithmetics

The vector arithmetic is implemented component-wise.

The arithmetic operations are performed component-by-component.

- + addition of a number to all components or addition of vectors component-by-component,
- subtracting of a number from all components or subtracting of vectors component-by-component,
- * multiplication of all components by number or multiplication of vectors component by component,
- / dividing all components by number or dividing of vectors component-by-component.

Vectors – the arithmetics

```
1 > v<-c(1,3,5,7,9)
2 > u<-c(10,20,30,40,50)
3 > u+v
4 [1] 11 23 35 47 59
5 > u-v
6 [1] 9 17 25 33 41
7 > 5*v
8 [1] 5 15 25 35 45
9 > u*v
10 [1] 10 60 150 280 450
11 > u/5
12 [1] 2 4 6 8 10
13 > u/v
14 [1] 10.000000 6.666667 6.000000 5.714286 5.555556
```

Matrix

Matrix is a two dimensional collection of data of the same type arranged in rectangular layout.

We create it using the function `matrix()` with following arguments

`vector` contains the elements of the matrix,

`nrow` is an integer value, it specifies number of rows in the matrix,

`ncol` is an integer value, it specifies number of columns in the matrix,

`byrow` is a logical value, it indicates, if the matrix should be filled by rows (`byrows=TRUE`) or by columns (`byrows=FALSE`), its default value is `FALSE`,

`dimnames` is a list of character vectors that contain optional row and columns labels.

Matrix – entering examples

```
1 > A<-matrix(3:8,nrow=3,ncol=2,byrow=TRUE)
2 > A
3      [,1] [,2]
4 [1,]    3    4
5 [2,]    5    6
6 [3,]    7    8
7 > B<-matrix(3:8,nrow=3,ncol=2,byrow=FALSE)
8 > B
9      [,1] [,2]
10 [1,]    3    6
11 [2,]    4    7
12 [3,]    5    8
```

Matrix – extracting submatrices

We define the rows and columns using the `c()` function.

```
1 > C<-matrix(1:12,nrow=3)
2 > C
3      [,1] [,2] [,3] [,4]
4 [1,]    1    4    7   10
5 [2,]    2    5    8   11
6 [3,]    3    6    9   12
7 > C[c(1,3),c(2,4)]
8      [,1] [,2]
9 [1,]    4   10
10 [2,]    6   12
```

Matrix – assigning names

We assign names to rows and columns using the `dimnames()` and `list()` functions

```
1 > dimnames(A) <- list(c("row1", "row2", "row3"),
2 + c("col1", "col2"))
3 > A
4      col1 col2
5 row1    3    4
6 row2    5    6
7 row3    7    8
8
9 > A["row2", "col1"]
10 [1] 5
```


Matrix – transposing

We can transpose the matrix using the function `t()`

```
1 > B<-matrix(3:8,nrow=3,ncol=2,byrow=FALSE)
2 > t(B)
3      [,1] [,2] [,3]
4 [1,]    3    4    5
5 [2,]    6    7    8
```

Other function are defined in package `matlib`.

Matrix – the operations

Defined component-wise

It is important when multiplying matrices. The common operation `*` means multiplication of the elements on the same positions.

The standard multiplication is defined as operation `%*%`.

Array

Arrays are generalizations of the matrix data structure.

They are more than two dimensional matrices

An array we can create using the `array()` function.

Its syntax is

```
name<-array(vector, dimensions,dimnames)
```

Array – creation

Now we illustrate creating of the $3 \times 4 \times 3$ array.

For greater clarity of the array, we create at first the names of single dimensions.

```
1 > dim1<-c("A1","A2","A3")
2 > dim2<-c("B1","B2","B3","B4")
3 > dim3<-c("C1","C2","C3")
```

Data structure frame

Frame is the most common structure for storing the data.

It enables storing the column vectors of the different data types.

Data frames are created using the function `data.frame()`, general syntax

```
1 > name<-data.frame(col1,col2,col3, ...)
```

Data frame – creation

We create a short data frame including data about scoring of the basketball players

```
1 > playerID<-c(1,2,3,4)
2 > position<-c("forward","guard","forward","center")
3 > attempted<-c(12,6,10,15)
4 > made<-c(7,4,6,12)
5 > players<-data.frame(playerID,position,attempted,made)
6 > players
7   playerID position attempted made
8 1         1 forward      12     7
9 2         2   guard       6     4
10 3         3 forward     10     6
11 4         4  center     15    12
```

Data frame – merging the datasets

We frequently need to merge data from two or more datasets.

We use function `merge()`.

The arguments are the names of two data frames to be merged.

Third argument `by='column_name'` defines the key variable for joining the data.

Data frame – merging the datasets

To demonstrate the merging, we create new frame rebounds at first.

```
1 > offensive<-c(5,2,3,10)
2 > defensive<-c(6,3,8,12)
3 > rebounds<-data.frame(playerID,defensive,offensive)
4 > row.names(rebounds)<-c("Player1","Player2","Player3",
5 + "Player4")
```


Lists

Lists represent the most complex data structure.

Lists are ordered collections of objects.

To create a list, we use the function `list()`. Its syntax is simple:

```
\list(object1,object2,...)
```

Its arguments are names of existing objects

Lists

Optionally we can name the object in the created list:

```
\list(name1=object1,name2=object2,...)
```

Entering data from the keyboard

The simplest method (but also the most time consuming for large samples)

We work in two steps

- Create the empty data frame with the variable names and types we want to store in the dataset.
- Invoke the simple data editor using the function `edit()`, whose argument is the name of the data frame we want to edit.

Entering data from the keyboard

We create empty data frame named `mydata` with four variables: `name` that has type `character` and three numeric variables `age`, `height` and `weight`.

```
1 > mydata<-data.frame(name=character(0),age=numeric(0),
2 + height=numeric(0),weight=numeric(0))
3 > mydata<-edit(mydata)
```

Note

Let us note, that assigning like `numeric(0)` and `character(0)` create a variable of the given type but without any data.

Entering data from the .csv file

Options of the `read.csv()` function.

- `header` logical value, indicates whether the input file contains the names of variables as the first line, default value `TRUE`.
- `sep` defines the field separator character, the default value is comma,
- `dec` defines the character used in the file for decimal points, default value is `.`, we mention also `read.csv2()` function, which adopts using the comma for decimal numbers and semicolon as delimiter.
- `skip=n` specify the number of lines to skip before the data starts. This option is useful for data tables with blank rows or text padding at the top of files.
- `stringsAsFactors` which is a logical value that indicates whether the strings are converted to factors, to prevent converting we set it to `FALSE`.
- `row.names` a vector of row names

Writing data into the .csv file

R can create csv file from existing data frame.

We use the `write.csv()` function, or alternatively the `write.csv2()` function, that uses a comma for the decimal point and a semicolon for the separator.

Common syntax

```
write.csv(object,file="file_name",...options)
```

`object` is obligatory argument containing the name of the data frame we want to save and `file_name` is the name (or full path) of the file

Writing data into the .csv file

Selected options of the `write.csv()` function

- `append` which is a logical value that indicates whether the output is appended to existing file. The default value is `FALSE` and any existing file of the given name is destroyed.
- `sep` defines the field separator character. Values within each row of object are separated by this character.
- `dec` the string to use for decimal points in numeric or complex columns, must be a single character. The default value is decimal point.
- `row.names` a logical value indicating whether the row names of object are to be written.

Entering data from the Excel files

Here are several packages that allow us to import data directly from Excel files. Let us mention some of them:

- `xlsx`,
- `XLconnect`
- `readxl`

Excel 2007 and newer versions use an `xlsx` format., therefore we introduce here the `xlsx` package.

Entering data from the Excel files

We install the package by standard command:

```
install.packages("xlsx")
```

To use it in actual workspace, we load it by the standard way:

```
library("xlsx")
```

Entering data from the Excel files

This package provides two functions for reading the contents of an Excel worksheet into a R data.frame: `read.xlsx()` and `read.xlsx2()`.

The difference between these two functions is:

- `read.xlsx()` preserves the data type, the type of the variable corresponds to each column in the worksheet, but it is slow for large data sets (worksheet with more than 100 000 cells).
- `read.xlsx2()` is faster on big files.

Entering data from the Excel files

Both functions have similar syntax:

```
read.xlsx(file, sheetIndex, header=TRUE, colClasses=NA)
read.xlsx2(file, sheetIndex, header=TRUE, colClasses="character")
```

Their arguments have the following meaning:

- `file` is the name of the file containing the spreadsheet. If the file is not in the working directory, it has to be declared with the full path.
- `sheetIndex` a number indicating the index of the sheet to read. We can replace it by the `sheetname` argument given as character string with the sheet name.
- `header` logical value. If `header=TRUE`, the first row is used as the names of the variables.
- `colClasses` a character vector that represents the class of each column.
- `startRow`, `endRow` numbers specifying the index of starting row and the last row to read.

Writing data into the Excel files

Package `xlsx` provides two functions for writing `write.xlsx()` and `write.xlsx2()`

General syntax

```
write.xlsx(x, file, sheetName="Sheet1", col.names=TRUE,  
row.names=TRUE, append=FALSE)
```

```
write.xlsx2(x, file, sheetName="Sheet1", col.names=TRUE,  
row.names=TRUE, append=FALSE)
```

Writing data into the Excel files

Their arguments have the following meaning:

- `x` a `data.frame` to be written into the workbook.
- `file` the path to the output file.
- `sheetName` the character string with the sheet name.
- `col.names` logical value, it indicates if the column names of `x` are to be written along with `x` to the file.
- `row.names` logical value, it indicates if the row names of `x` are to be written along with `x` to the file.
- `append` logical value, it indicates if `x` should be appended to an existing file, if `FALSE`, it overwrites the existing file with the same path.

Reading data from the JSON files

JSON (JavaScript Object Notation) is a lightweight data-interchange format

To get JSON files into R, we first need to install or load the `rjson` package.

We can use the `fromJSON()` function

Usage depends on the location of the `.json` file

```
data<-fromJSON(file = "filename.json")  
data<-fromJSON(file = "URL to the json file")
```

In both cases, the object data is stored as the list. For the further analysis we can convert the data using the `as.data.frame()` function.

a

Writing data into the JSONI files

It has to be done in two phases.

In the first step we must prepare the JSON object and in the second step we write it in the file.

To create a JSON object we use `toJSON()` function:

```
dataJSON<-toJSON(data)
```

Then we use the `write()` function

```
write(dataJSON, "filename.json")
```




R programming for statistics

III. Probability distributions in R

Functions implemented

R enables working with a big variety of the probability distributions

For each of the distributions that R handles are implemented four functions

Their names composed from the root name and prefix:

- p for the cumulative distribution function,
- d for the density or probability function,
- q for the quantile function, the inverse to the cumulative distribution function,
- r random variable having the specified distribution (random values generator).

Discrete distribution

The probabilities are here encoded by a discrete list of the probabilities of the outcomes, known as the probability mass function

If we assign as H the set of all possible values of the discrete random variable X , we can introduce the probability mass function $p(x)$ by formula

$$p(x) = \mathbb{P}(X = x), x \in H. \quad (1)$$

Discrete distribution

We mention some of them:

- Bernoulli distribution,
- binomial distribution,
- geometric distribution,
- hypergeometric distribution,
- negative binomial distribution,
- Poisson distribution.

Bernoulli distribution

The simplest probability distribution

This is a distribution with only two possible values.

It can be interpreted as an indicator variable, if some random event occurs or not.

Let A is a random event, $\mathbb{P}(A) = p$ and the random variable $X = 1$ if A occurs and $X = 0$ otherwise. Then the probability mass function has the form

$$p(x) = p^x(1 - p)^{1-x} \text{ for } x = 0 \text{ or } 1$$

It is implemented in package `Rlab` as `bern` with parameter `prob`.

Bernoulli distribution

We have at disposal four functions:

- `rbern(n,prob)`, where `n` is a number of observations and `prob` is a probability of occurring the random event A (success in the trial). It generates a vector of 0 and 1 selected from the Bernoulli distribution with given probability.
- `pbern(q, prob, lower.tail = TRUE, log.p = FALSE)`
- `dbern(x, prob, log = FALSE)`
- `qbern(p, prob, lower.tail = TRUE, log.p = FALSE)`

Binomial distribution

The binomial distribution is a discrete distribution that describes number of successes in the series of trials with two possible outcomes

Formally, let us assign p the probability of success in one trial, n the number of independent trials and x the number of successes in a sequence of n independent experiments. The random variable X follows the binomial distribution, if its probability mass function has the form:

$$\mathbb{P}(X = x) = \binom{n}{x} p^x q^{n-x}, \quad x = 0, 1, 2, \dots, n, \quad (2)$$

where $q + p = 1$.

Binomial distribution

Four functions for handling binomial distribution in R:

- `rbinom(n, prob)`, where `n` is numbers of observations, `p` is the probability of success. This function generates n random variables of a particular probability.
- `pbinom(x, n, k)`, where `n` is total number of trials, `p` is probability of success, `x` is the value at which the probability has to be found out.
- `dbinom(x, n, p)`, where `n` is total number of trials, `p` is probability of success, `x` is the value at which the probability has to be found out.
- `qbinom(prob, n, p)`, where `prob` is the probability, `n` is the total number of trials and `p` is the probability of success in one trial. This function is used to find the n -th quantile, that is if $P(X \leq k)$ is given, it finds k .

Hypergeometric distribution

The hypergeometric distribution is a discrete probability distribution that describes the probability of x successes in n draws without replacement, from a finite population of size N that contains exactly K objects with that feature.

Let us denote as N the population size, K the number of success states in the population, n the number of draws and x the number of observed successes. The random variable X follows the hypergeometric distribution, if its probability mass function has the form

$$\mathbb{P}(X = x) = \frac{\binom{K}{x} \binom{N-K}{n-x}}{\binom{N}{n}} \quad x = 0, 1, 2, \dots, n. \quad (3)$$

Hypergeometric distribution

Four functions for handling hypergeometric distribution in R:

- `rhyper(N, m, n, k)`, generally refers to generating random numbers function by specifying a seed and sample size,
- `phyper(x, m, n, k)`, defines the cumulative distribution function of the hypergeometric distribution,
- `dhyper(x, m, n, k)`, defines the probability mass function of the hypergeometric distribution,
- `qhyper(N, m, n, k)`, is basically hypergeometric quantile function used to specify a sequence of probabilities between 0 and 1.

Here x represents the data set of values, m size of the population, n number of samples drawn, k number of items in the population, and N hypergeometrically distributed values.

Negative binomial distribution

The negative binomial distribution is a discrete probability distribution that models the number of successes in a sequence of independent and identically distributed Bernoulli trials before a specified (non-random) number of failures (denoted n) occurs.

If we denote x the number of successes and the probability of the success as p , the probability mass function of the negative binomial distribution has the form:

$$\mathbb{P}(X = x) = \binom{n+x-1}{n-1} p^x q^n, \quad x = 0, 1, 2, \dots \quad (4)$$

where $q + p = 1$, $p > 0$, $q > 0$.

Negative binomial distribution

Four functions for handling negative binomial distribution in R:

- `rnbinom(N, n, prob)`, where `n` is numbers of trials, `N` is the sample size, `prob` is the probability of success. This function generates N random variables of a particular probability.
- `pnbinom(x, n, p)`, is used to compute the value of negative binomial cumulative distribution function. Here `x` is number of failures prior to the `n`-th success, and `p` is probability of success.
- `dnbinom(x, n, p)`, is the probability of `x` failures prior to the `n`-th success (note the difference) when the probability of success is `p`.
- `qnbinom(x, n, p)`, is used to compute the value of negative binomial quantile function. Here `x` is the vector of quantile levels, `n` is the total number of trials and `p` is the probability of success in one trial.

Poisson distribution

The Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time or space if these events occur with a known constant mean rate and independently of the time since the last event.

If random variable X follows the Poisson distribution with parameter $\lambda > 0$ (the average number of events), its probability mass function has the form

$$\mathbb{P}(X = x) = \frac{e^{-\lambda} \lambda^x}{x!}, \quad x = 0, 1, 2, \dots \quad (5)$$

Poisson distribution

Four functions for handling Poisson distribution in R:

- `dpois(x,1)` calculates the probability mass function value $\mathbb{P}(X = x)$ of the Poisson distribution with the parameter λ implemented as argument `1`.
- `ppois(x,1)` calculates the cumulative distribution function of a random variable that follows the Poisson distribution. It returns the probability, $\mathbb{P}(X \leq x)$ the argument `1` is the parameter of the distribution. Stating the additional argument `lower.tail=FALSE` we get the probability $\mathbb{P}(X > x)$.
- `rpois(k,1)` is used for generating random numbers from a given Poisson distribution, `k` is number of random numbers needed and `1` is the parameter of the distribution.
- `qpois(q,1)` is used for generating quantile of a given Poisson's distribution, `q` is a vector of the quantile levels required and `1` is the parameter of the distribution.

Continuous distribution

It is a probability distribution whose support is an uncountable set.

They are uniquely characterized by a cumulative distribution function

$$F(x) = \mathbb{P}(X \leq x) = \int_{-\infty}^x f(t) dt.$$

Continuous distribution

We mention some of them:

- uniform distribution,
- exponential distribution,
- normal distribution,
- Student t distribution,
- Chi square distribution,
- Fisher F distribution.

Many other are implemented in R.

Continuous distribution

We mention some of them:

- uniform distribution,
- exponential distribution,
- normal distribution,
- Student t distribution,
- Chi square distribution,
- Fisher F distribution.

Many other are implemented in R.

We will concern in three “blue” distributions.

Uniform distribution

The continuous uniform distribution describes an experiment where there is an arbitrary outcome that lies between certain bounds.

More exactly, random variable X follows the uniform distribution with parameters a and b , $a < b$, if its density function has the form:

$$f(x) = \begin{cases} \frac{1}{b-a} & x \in \langle a; b \rangle \\ 0 & \text{elsewhere} \end{cases} \quad (6)$$

Uniform distribution

Four functions for handling uniform distribution in R:

- `dunif()` that gives the density function, its arguments are vector `x` and parameters `min` and `max` of the distribution,
- `pnif()` that gives the cumulative distribution function, its arguments are vector `x` and parameters `min` and `max` of the distribution,
- `qunif()` that gives the quantile function, its arguments are quantiles `q` and parameters `min` and `max` of the distribution,
- `runif()` that generates the random values of the variable, its arguments are size of the sample `n` and parameters `min` and `max` of the distribution.

Exponential distribution

The exponential distribution describes the waiting time between events in a Poisson point process, i.e., a process in which events occur continuously and independently at a constant average rate.

The random variable X follows the exponential distribution with parameter $\lambda > 0$, (usually reported as the rate), if its density function has the form:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases} \quad (7)$$

When the parameter λ is interpreted as the rate, the mean waiting time is $\frac{1}{\lambda}$.

Exponential distribution

Four functions for handling exponential distribution in R:

- `dexp()` that gives the density function, its arguments are vector `x` and parameter `rate` of the distribution,
- `pexp()` that gives the cumulative distribution function, its arguments are vector `x` and parameter `rate` of the distribution,
- `qexp()` that gives the quantile function, its arguments are quantiles `q` and parameter `rate` of the distribution,
- `rexp()` that generates the random values of the variable, its arguments are size of the sample `n` and parameter `rate` of the distribution.

Normal distribution

A normal (or Gauss) distribution is a type of continuous probability distribution, its probability density function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}. \quad (8)$$

The parameter μ is the mean the distribution (and also its median and mode), while the parameter σ is its standard deviation.

The normal distribution is important because of the Central Limit Theorem, which states that the population of all possible samples of size n from a population with mean μ and variance σ^2 approaches a normal distribution with mean μ and $\frac{\sigma^2}{n}$ when n approaches infinity.

Normal distribution

Four functions for handling normal distribution in R:

- `dnormf()` that gives the density function, its arguments are vector `x` and parameters `mean` and `sd` of the distribution,
- `pnorm()` that gives the cumulative distribution function, its arguments are vector `x` and parameters `mean` and `sd` of the distribution,
- `qnorm()` that gives the quantile function, its arguments are quantiles `q` and parameters `mean` and `sd` of the distribution,
- `rnorm()` that generates the random values of the variable, its arguments are size of the sample `n` and parameters `mean` and `sd` of the distribution.



R programming for statistics

IV. Programming in R

Functions

Almost all actions in R run through functions.

Here is a rich collection of built-in function

Other functions can be defined by user

The built-in functions we can divide into

- math functions,
- string functions,
- specialized statistical and probability functions,
- other useful functions.

Math functions

Some of them we have already mention in lesson 1.

Here we introduce some additional details

The logarithmic function `log()` computes as the default value the natural logarithm.

To get logarithm with any base, we must declare the base argument of the function `log()`.

```
1 > log(4)
2 [1] 1.386294
3 > log(4, base=2)
4 [1] 2
```

String functions

Function `nchar()` determines the size of each elements of an character vector

```
1 > z<-c("yellow","black","white")
2 > nchar(z)
3 [1] 6 5 5
4 > str<-"This is a long string"
5 > nchar(str)
6 [1] 21
```

Elementary statistical functions

- `mean()` Sample mean.
- `median()` Sample median.
- `sd()` Standard deviation.
- `var()` Sample variance.
- `mad()` Median absolute deviation.
- `quantile()` Quantiles, default returns quartiles.
- `range()` Range of the values.
- `sum()` Sum of the vector elements.
- `min()` Minimum.
- `max()` Maximum.

Elementary statistical functions – mean() optional arguments

trim that state the percentage of the highest and lowest values being dropped from the computation and so it returns the trimmed mean.

Second optional argument na.rm is a logical value indicating whether NA values should be stripped before the computation proceeds.

```
1 > x<-c(1,3,5,10,12)
2 > mean(x)
3 [1] 6.2
4 > mean(x,trim=0.2)
5 [1] 6
```

```
1 > x<-c(1,5,2,12,NA,3,6)
2 > mean(x)
3 [1] NA
4 > mean(x,na.rm=TRUE)
5 [1] 4.833333
6 > mean(x,na.rm=TRUE,trim=0.17)
7 [1] 4
```

Elementary statistical functions – mad()

Median absolute deviation is a robust measure of the variability of a univariate sample of quantitative data.

For the sample X_1, \dots, X_n it is defined by formula:

$$\text{MAD}(X) = \text{median}\{|X_i - \bar{X}|\}$$

```
1 > mad(delay)
2 [1] 13.3434
```

Useful functions – seq()

The function `seq()` generates the sequence of numbers starting by `from` value and ends in `to` value. The last argument by defines the step of the sequence.

```
1 > seq(10)
2 [1] 1 2 3 4 5 6 7 8 9 10
3 > seq(5,15)
4 [1] 5 6 7 8 9 10 11 12 13 14 15
5 > seq(5,15,2)
6 [1] 5 7 9 11 13 15
```

Useful functions – rep()

Function `rep()` has two arguments, the vector `x` to be repeated and number `n` of the repeating cycles

```
1 > rep(1,10)
2 [1] 1 1 1 1 1 1 1 1 1 1
3 > rep(c(1,3),4)
4 [1] 1 3 1 3 1 3 1 3
5 > rep("hello",3)
6 [1] "hello" "hello" "hello"
```


Useful functions – `sort()` and `order()`

The functions `sort()` and `order()` are joined with sorting the vector `x`

`sort()` gives sorted values while `order()` gives the indices of ordered the components in the original vector.

```
1 > x<-c(5,2,10,3,7,8)
2 > sort(x)
3 [1]  2  3  5  7  8 10
4 > order(x)
5 [1] 2 4 1 5 6 3
```

Useful functions – `rev()`

Gives vector `x` in reverse order

```
1 > rev(x)
2 [1]  8  7  3 10  2  5
3 > rev(sort(x))
4 [1] 10  8  7  5  3  2
```

Conditional statements – if statement

if() statement performs operations based on a simple condition

```
if(condition){commands to be performed if condition holds}
```

More than one commands need to be braced

```
1 > x<-5
2 > if(x%%2){print("Odd number")}
3 [1] "Odd number "
4 > x<-6
5 > if(x%%2){print("Odd number")}
6 >
```

Conditional statements – switch

`switch()` tests an expression against elements of a list. Each value in the list is called case

The syntax of the `switch()` function:

```
switch (expression, list)
```

```
1 > x<-10
2 > switch(x%%2+1,"even","odd")
3 [1] "even"
4 > x<-9
5 > switch(x%%2+1,"even","odd")
6 [1] "odd"
```

Conditional statements – switch

If the expression is a character string, `switch()` will return the value based on the name of the element.

```
1 > x <- "a"
2 > switch(x, "a"="apple", "b"="banana", "c"="cherry")
3 [1] "apple"
4 > x <- "c"
5 > switch(x, "a"="apple", "b"="banana", "c"="cherry")
6 [1] "cherry"
```

Loops – for

for loop allows us to repeat a command or a block of commands a fixed number of times

The general syntax of the for loop is like this:

```
for (val in sequence)
{
statement
}
```

where `sequence` is a vector and `val` takes on each of its value during the loop

User defined function

The general structure of a function is given below.

```
myfunction_name <- function(arg1, arg2, ... ){  
  statements  
  return(object)  
}
```

User defined function

The different components of a function are:

- **Function Name** which is the actual name of the function. It is stored in R environment as an object with this name.
- **Arguments** which are placeholders. When a function is invoked, we pass values to the arguments. Arguments are optional, that is, a function may contain no arguments. Also arguments can have default values.
- **Function Body** which contains a collection of statements that defines what the function does. The body of the function goes inside the curly brackets `{}`.
- **Return Value** which is the last expression in the function body to be evaluated.

Running R scripts

An R script is simply a text file containing (almost) the same commands that you would enter

We can create it in any simple text editor and save with extension `.R`

There are basically two Linux commands to run the script

```
Rscript filename.R
```

and is preferred. The older command is

```
R CMD BATCH filename.R
```



R programming for statistics

V. Elementary graphics

Scatter plots

Scatter Plot represents values for two different numeric variables.

The position of each dot on the horizontal and vertical axis indicates values for an individual data point.

The most common applications and uses of the scatter plots are:

- 1 Demonstration of the relationship between two variables.
- 2 Identification of correlation relationships.
- 3 Identification of data patterns.

Scatter plots

We create it simply using `plot()` function.

In the simplest use, the function has two arguments `x` and `y`

These variables are vectors that contain the values we want to plot.

The length of the vectors must be the same.

How to save plot

Alternatively, we can redirect the output from screen to the file.

We can use the functions

<code>pdf()</code>	Vector pdf format, best choice when used with <code>pdflatex</code>
<code>svg()</code>	Vector svg format, easily resizable.
<code>postscript()</code>	Vector postscript format <code>ps</code> , easily resizable.
<code>png()</code>	Bitmap format with high resolution, does not resize.
<code>jpeg()</code>	Compressed bitmap format, does not resize.
<code>bmp()</code>	High resolution bitmap format, does not resize.
<code>tiff()</code>	High resolution bitmap format, does not resize.


























Options for saving the graphs

<code>filename</code>	Name of the saved file, with full path if necessary.
<code>width</code>	Width of the resulting graph, default value 7 in.
<code>height</code>	Height of the resulting graph, default value 7 in.
<code>res</code>	resolution of the picture, applicable for bitmap formats, default 72 dpi.
<code>units</code>	Units of measure.
<code>bg</code>	Background colour.
<code>fg</code>	Foreground colour.
<code>family</code>	The fonts used (default Helvetica).

Modifying the plot – dot characters

The dot character is given by value of `pch` argument of the `plot()` function

Possible values

 <code>pch=0</code>	 <code>pch=1</code>	 <code>pch=2</code>	 <code>pch=3</code>	 <code>pch=4</code>
 <code>pch=5</code>	 <code>pch=6</code>	 <code>pch=7</code>	 <code>pch=8</code>	 <code>pch=9</code>
 <code>pch=10</code>	 <code>pch=11</code>	 <code>pch=12</code>	 <code>pch=13</code>	 <code>pch=14</code>
 <code>pch=15</code>	 <code>pch=16</code>	 <code>pch=17</code>	 <code>pch=18</code>	 <code>pch=19</code>
 <code>pch=20</code>	 <code>pch=21</code>	 <code>pch=22</code>	 <code>pch=23</code>	 <code>pch=24</code>

Modifying the plot – line type

The line type is set by the `type` argument of the `plot()` function

Possible values

- p Point graph, the default value.
- l Continuous line.
- b Continuous line with the points.
- c Parts of the continuous lines, with the points omitted.
- o Parts of the continuous lines, with the points over-plotted.
- h Histogram-like graph.
- s stair steps graph.

Modifying the plot – line style

The line style is set by the `lty` argument of the `plot()` function

Possible values

- | | | | |
|---|-----------------------|---|------------------------------------|
| 1 | Solid line (default). | 2 | Dashed line. |
| 3 | Dotted line. | 4 | Dot-dashed line. |
| 5 | Long dashed line. | 6 | Long and short double dashed line. |

The line width is set by the `lwd` argument of the `plot()` function

Modifying the plot – colouring

Setting the colours we can do by

- name of the colour, for example `col=red`
- by number of the colour, for example `col=636`
- by hexadecimal code, for example `col="#FFCC00"`

The list of available colours we get as an answer of the function `colors()`.

Modifying the plot – colouring

Other colouring options are

<code>col.axis</code>	Axis annotation colour.	<code>col.lab</code>	Axes labels colour.
<code>col.main</code>	Main title colour.	<code>col.sub</code>	Subtitle colour.
<code>bg</code>	Character filling color.	<code>fg</code>	Foreground colour.

Modifying the plot – colouring

Colours as vectors

We can set the value of the `col` argument as vector.

The colours from the vector are regularly varying

We can also use the `rainbow()` function, with predefined colour sequence.

Modifying the plot – titles and subtitles

Base R plotting functions come with an argument named `main` that allows adding a title to the plot.

One can also add a subtitle, that will be positioned under the plot making use of the `sub` argument.

Alternative way, how to add the title and subtitle to the graph is using the function `title()`.

Modifying the plot – adding text into the plot

We can add some texts into the plotted graph using the functions `text()` and `mtext()`.

The difference `text()` places the given text on any position in the plotting area , `mtext()` function places the text into the margins.

The function `text()` has two additional arguments:

- `location` defines the `x` and `y` coordinates, where the text will be placed. The coordinates must be submitted as the first two arguments of the function.
- `pos` defines the position according to the actual place, 1=bottom, 2=left, 3=top and 4=right. Defining position as `locator(1)` enables placing the text using the mouse.

Modifying the plot – axes customization

To remove the plot box we set the option `axes=FALSE` inside the plotting function.

New axes we add using the `axis()` function.

Argument of the `axis()` function defines the side of the plot, where the axis will be added.

As usually, the numbers define the sides 1=bottom, 2=left, 3=top and 4=right.

Let us try

```
1 > plot(sort(x), y[order(x)], pch=17, type="b",  
2     col=30, axes=FALSE)  
3 > axis(1)  
4 > axis(2)
```

Further axes customization

We are able to:

- set the number of the tick marks with specified start end values,
- modify the length and orientation of the tick marks,
- rotate the tick marks labels,
- custom the tick mark labels,
- remove tick marks,
- add the minor ticks using the `Hmisc` .

Further axes customization—ticks marks length and orientation

The argument `tck` allows to modify the length and orientation of the tick marks.

Its positive value sets the marks inside the plotting area while the negative values define the marks outside from the plotting area. The greater the absolute value, the longer the ticks. the default value is `tck=-0.05`.

The rotation is enabled using the `las` argument that can take one of four values:

- `las=0` the labels are parallel to axis (default),
- `las=1` all labels are horizontal,
- `las=2` the labels are perpendicular to axis,
- `las=3` all labels are vertical.

Limits and scaling

The limits for the axis we can define using the optional arguments `xlim` and `ylim` of the `plot()` function

The limits are submitted as vectors in the form `c(start, end)`

We can also transform the axes into the logarithmic scale by setting the argument `log` to be equal to axis that we plan to scale.

`log="x"` sets the logarithmic scale to the x-axis,
`log="y"` sets the logarithmic scale to the y-axis and
`log="xy"` transforms both axis into the logarithmic scale.

Two dual vertical axes

Example

We want to plot in the same graph two characteristics of the patients' health condition, the temperature and the blood pressure.

Two dual vertical axes

Example

We want to plot in the same graph two characteristics of the patients' health condition, the temperature and the blood pressure.

We have data of 100 patients stored in variables y and z , while the variable x contains the sequence of the patient identifiers, numbers from 1 to 100.

Plotting the curves

One of the many handy functions in R is `curve()`.

It is a neat little function that provides mathematical plotting, e.g., to plot functions.

The `curve()` function takes, as its first argument, an R expression.

For example

```
curve(x^2)
curve(x^2,xlim=c(-2,2),col="red",lwd=2)
```

Plotting two or more curves in one plot

We use the function `curve()` with argument `add=TRUE`.

For example

```
curve(x^2)
curve(sqrt(x), col="red", lwd=2, add=TRUE)
```

Adding a legend

The function `legend()` enables adding a legend to the plots in R.

Some of the arguments:

- `x,y` position in the plotting area defined by coordinates in the graph,
- legend vector of strings for description in the legend,
- `col` vector of colours used in the graph,,
- `pch` vector of the mark shapes used in the graph,
- `lty` vector of the line types used in the graph,
- `ncol` number of columns used in the legend, default value is one column.

Bar graphs

Bar graph presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent.

To produce the bar graphs, R uses the function

```
barplot(H,xlab,ylab,title, names.arg,col)
```

The parameters used in the function are as follows:

- `H` is a vector or matrix containing numeric values used in bar chart,
- `xlab` is the label for x axis,
- `ylab` is the label for y axis,
- `title` is the title of the bar chart,
- `names.arg` is a vector of names appearing under each bar,
- `col` is used to give colors to the bars in the graph.

Bar graphs – colouring and labels

We use these the `names.arg` parameter of the bar plot to assign these names to the columns

Further we define the values of the parameters

- `xlab` and `ylab` for the axes names,
- `col` and `border` for colouring the bars and
- `main` to define the title of the graph

It is similar like in the case of `plot()` function.

Histograms

A histogram is an approximate representation of the distribution of numerical data.

They represent the frequencies of values of a variable bucketed into ranges.

Histogram is similar to bar graph but the difference is it groups the values into continuous ranges.

Histograms give a rough sense of the density of the underlying distribution of the data

Histograms

Histogram can be created using the `hist()` function in R

```
hist(H,xlab,ylab,title, names.arg,col)
```

The parameters used in the function are as follows:

- `data` is a vector containing numeric values used in histogram,
- `main` indicates title of the chart,
- `col` is used to set colour of the bars,
- `border` is used to set border colour of each bar,
- `xlab` is used to give description of x-axis,
- `xlim` is used to specify the range of values on the x-axis,
- `ylim` is used to specify the range of values on the y-axis,
- `breaks` is used to mention the width of each bar.

Pie graphs

A pie chart is a plot for a single categorical variable and it is an alternative to bar chart.

A pie chart (or a circle chart) is a circular statistical graphic, which is divided into slices to illustrate numerical proportion.

In a pie chart, the arc length of each slice (and consequently its central angle and area), is proportional to the quantity it represents.

Pie graphs

The basic syntax for creating a pie-chart using the R is:

```
pie(data, labels, radius, main, col, clockwise)
```

The meaning of the arguments:

- `data` is a vector containing the numeric values used in the pie chart,
- `labels` is used to give description to the slices,
- `radius` indicates the radius of the circle of the pie chart, (value between -1 and $+1$),
- `main` indicates the title of the chart,
- `col` indicates the colour palette,
- `clockwise` is a logical value indicating if the slices are drawn clockwise or anti clockwise.

Pie graph–colours modifying

To change the colours in the graph we apply the function `rainbow()`, which defines the colour palette.

Its arguments are:

- `n` the number of colours (≥ 1) to be in the palette,
- `s, v` the “saturation” and “value” to be used to complete the colour descriptions
- `start` the (corrected) hue in $\langle 0; 1 \rangle$ at which the rainbow begins,
- `end` the (corrected) hue in $\langle 0; 1 \rangle$ at which the rainbow ends,
- `gamma` the gamma correction, for each colour, (r, g, b) in RGB space (with all values in $\langle 0; 1 \rangle$), the final colour corresponds to $(r^\gamma, g^\gamma, b^\gamma)$,
- `alpha` the alpha transparency, a number in $\langle 0; 1 \rangle$, (0 means transparent and 1 means opaque).

Fan plot

Useful alternative to the pie charts is `fun.plot()` defined in the `plotrix` package.

It allows to compare visually the pie sectors of the chart.

We can customize the fun plot setting the additional arguments:

- `max.span` the angle of the maximal sector in radians. The default is to scale data so that it sums to 2π .
- `ticks` the number of ticks that would appear if the sectors were on a pie chart. Default is no ticks.

Box plot

In descriptive statistics, a box plot or boxplot is a type of chart often used in explanatory data analysis.

Box plots visually show the distribution of numerical data and skewness through displaying the data quartiles (or percentiles) and averages.

It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

Box plot

A boxplot is constructed of two parts, a box and a set of whiskers.

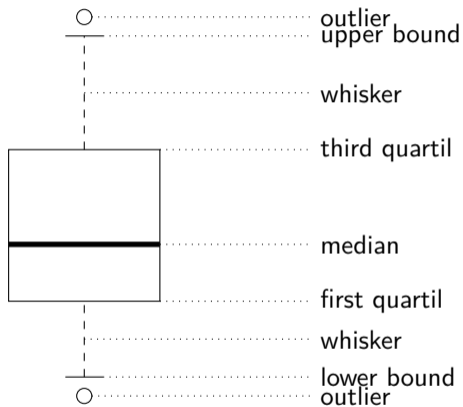
The box is drawn from the lower quartile to the upper quartile with a horizontal line drawn inside the box to denote the median.

However, the whiskers can represent several alternative values, among the observed data:

- the minimum and maximum of all of the data,
- one standard deviation above and below the mean of the data,
- the 9-th percentile and the 91-st percentile,
- the 2-nd percentile and the 98-th percentile.

Any data not included between the whiskers should be plotted as an outlier with a dot, small circle, or star, but occasionally this is not done.

Box plot



Q-Q plot

The quantile-quantile plot (or shortly Q-Q plot), is a graphical tool to help us assess if a set of data plausibly came from some theoretical distribution such as a normal or exponential.

For example, if we run a statistical analysis that assumes our dependent variable is normally distributed, we can use a normal Q-Q plot to check that assumption.

It is just a visual check, not an exact proof, but it allows us to see at-a-glance if our assumption is plausible, and if not, how the assumption is violated and what data points contribute to the violation.

Q-Q plot

A Q-Q plot is essentially a scatterplot created by plotting two sets of quantiles against one another.

If both sets of quantiles came from the same distribution, the points form a roughly straight line.

Q-Q plots take our sample data, sort it in ascending order, and then plot them against quantiles of the suggested theoretical distribution.

The number of quantiles is selected to match the size of our sample data.

More graphs in one plot

In base R we can combine the graph with `mfrow` and `mfcol` graphical parameters.

We just need to specify a vector that determines the number of rows and the number of columns we plan to create.

The decision of which graphical parameter we should use depends on how do we want our plots to be arranged:

- `mfrow` the plots will be arranged by rows,
- `mfcol` the plots will be arranged by columns.

This setting is used as argument of the `par()` function, that defines parameters of the graphical device



R programming for statistics

VI. Descriptive sample characteristics

The mean

The **sample mean** of the given set of numbers x_1, x_2, \dots, x_n is defined by formula:

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n} = \frac{\sum_{i=1}^n x_i}{n}. \quad (9)$$

In the R environment it is implemented as the `mean()` function and its use is very simple.

The mean

On the website of the Slovak central bank

<https://www.nbs.sk/en/monetary-policy/macroeconomic-database> we can find macroeconomic database.

We can download for example the `.csv` file containing the numbers of new passenger cars registrations in selected time period.

This file is implicitly saved as `macrostat.csv`.

Due to the use of comma as the decimal separator, we read the data using the `read.csv2()` function.

The mean

To compute the mean, we have to use `as.numeric()` because `cars[1,]` gives the values in the list format.

So to obtain the average value of the monthly registered new passenger cars (in thousands) in years 2017-18 we can use the code:

```
1 cars<-read.csv2("macrostat.csv",header=FALSE,sep=";")
2 mean(as.numeric(cars[1,]))
3 [1] 8.090125
```

The mean

It is often the case that the values of the statistical trait of interest are ordered in a sequence of absolute frequencies.

In this case, we modify the relation (9) for calculating the sample mean to the form:

$$\bar{x} = \frac{x_1 \cdot n_1 + x_2 \cdot n_2 + \cdots + x_k \cdot n_k}{n_1 + n_2 + \cdots + n_k} = \frac{\sum_{i=1}^k x_i \cdot n_i}{\sum_{i=1}^k n_i}. \quad (10)$$

where x_i 's denote the values of the variable and n_i denotes the absolute frequency of x_i .

The mean

In this case, we need to define a custom function to calculate the mean value.

As input values, we will specify two vectors. The first vector contains the values that the random variable takes, and the second is a vector of their multiplicities.

Before performing the calculation according to the relation (10), it is necessary to verify that both vectors have the same length.

The median

We can characterise median as the middle value if the observed values are sorted from smallest to largest.

Formally we can say, that variable value is larger then median equals to the probability, that its value is less then median and consequently this probability equals to $\frac{1}{2}$.

Having a sample of numbers x_1, x_2, \dots, x_n the median \tilde{x} is given as

$$\tilde{x} = \begin{cases} \left(\frac{n+1}{2}\right)\text{-th ordered value} & n \text{ is odd} \\ \frac{1}{2} \left(\left(\frac{n}{2}\right)\text{-th} + \left(\frac{n}{2}\right)\text{-th ordered value} \right) & n \text{ is even} \end{cases} \quad (11)$$

Quantiles

To find the quantiles, in R is implemented the `quantile()` function. Without any optional parameters it gets the minimum of the sample, the first quartile, median, the third quartile and the maximal value of the sample.

We can illustrate it on the COVID-19 data, downloaded from the official website of the Slovak government <https://korona.gov.sk>.

```
1 data<-read.csv("https://mapa.covid.chat/export/csv",
2   header=T,sep=";")
3 > quantile(data[,4])
4   0%    25%    50%    75%   100%
5     0     30    232   1737  15278
6 >
```

The variation range

The **variation range** depends only on two values in the sample – the greatest and the smallest values.

It is defined as the difference

$$R = \max\{x_1, \dots, x_n\} - \min\{x_1, \dots, x_n\} \quad (12)$$

It is well applicable for a quick orientation on the extent of variability of a given sample.

The variation range

The `range()` function determines the range of variation in the R language environment.

Its outputs are two values – the greatest and the smallest value in the sample.

In order to express the variation range as a single value by definition ([12](#)), we use the `max()` and `min()` functions.

Interquartile range

The **interquartile range** is a measure of statistical dispersion.

It is defined as the difference between the upper and lower quartiles of the data.

Assigning these quartiles as Q_3 and Q_1 , we can express the interquartile range IQR as

$$IQR = Q_3 - Q_1. \quad (13)$$

Mean absolute deviation

The mean absolute deviation *MAD* of a dataset is the average distance between each data point and the mean.

For the sample x_1, \dots, x_n , it is defined by formula

$$MAD = \frac{1}{n} \sum_{i=1}^n |x_i - \bar{x}|. \quad (14)$$

Variance and standard deviation

Variance is the most popular and the most frequently used measure of the sample variation.

If we have the sample x_1, \dots, x_n , we define the sample variation s^2 by formula

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2. \quad (15)$$

Variance and standard deviation

In the case of the data ordered in the sequence absolute frequencies n_1, \dots, n_k , $\sum_{i=1}^k n_i = n$, of the values x_1, \dots, x_k , we modify the formula (15) into the form

$$s^2 = \frac{1}{n} \sum_{i=1}^k n_i (x_i - \bar{x})^2. \quad (16)$$

Variance and standard deviation

The standard deviation is then defined as the square root from the variance, means:

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}, \quad (17)$$

or in the case of the ordered data set

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n} \sum_{i=1}^k n_i (x_i - \bar{x})^2}. \quad (18)$$

Variance and standard deviation

We must use be the `var()` and `sd()` functions carefully.

Their results are an unbiased estimates of the variance and standard deviation of the whole population.

If we want to compute the sampling variance according to the relation (15), we have to define our own function, which we illustrate in the following source code.

Coefficient of variation

The coefficient of variation is a statistical measure of the relative dispersion of data points in a data series around the mean.

The **coefficient of variation** CV is defined as the ratio of the standard deviation s to the mean \bar{x}

$$CV = \frac{s}{\bar{x}}. \quad (19)$$

The coefficient of variation is frequently expressed as a percentage.

Skewness

Skewness measures the asymmetry of the distribution or data set.

We define the skewness γ_1 as

$$\gamma_1 = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^3}{s^3}. \quad (20)$$

Skewness

Depending on the value of γ_1 there exist 3 types of skewness.

If $\gamma_1 > 0$ we speak about positive skewness. It means that majority of the data are less than the average.

On the other hand there are the negative skewed data, when $\gamma_1 < 0$ In this case is the majority of the values in the dataset greater than the mean.

Finally zero skewness $\gamma_1 = 0$ represents the symmetric distribution of the data.

Kurtosis

Kurtosis measures the sharpness of the peak in the data distribution.

We define the kurtosis γ_2 as

$$\gamma_2 = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^4}{s^4}. \quad (21)$$

Kurtosis

Similarly like in the case of skewness, here are also three types of kurtosis.

If $\gamma_2 = 3$ we speak about mesokurtic distribution. The value γ_2 is compared with 3, because the kurtosis of the normal distribution equals 3. So we compare the kurtosis of the sample with the Gaussian curve.

In the case $\gamma_2 < 3$ is the distribution more flat than normal distribution and we speak about platykurtic distribution.

In the opposite case, when $\gamma_2 > 3$, the analysed distribution has a greater peak in the mean than normal distribution. This kind of distributions is said to be leptokurtic.



R programming for statistics

VII. Parameter estimates

Parameter estimates

One of the goals of the statistical analysis is to estimate the parameters of the original distribution from which the random selection comes.

We distinguish two types of estimates:

- **the point estimates** that provide the estimate of the exact value of the parameter,
- **confidence intervals** that give the intervals, that contain the real value of the parameter with given probability (reported as the confidence level).

Point estimates

To estimate the value of the parameter we aim to choose the sample characteristic that approximate the parameter Θ with the best quality.

The quality of the estimator is guaranteed by its properties:

- **Unbiased estimate** of the parameter θ is such estimate T that equality $\mathbb{E}(T) = \theta$ holds.
- **Consistent estimate** can be characterised by increasing accuracy with increasing sample size. Formally we can write $\lim_{n \rightarrow \infty} T_n = \Theta$, where indices represent the size of the sample used in estimation.
- **Efficient estimate** can be interpreted as the best possible estimate. The notion best possible relies on the loss measured by the mean squared error of T . It is the value $MSE(T) = \mathbb{E}((T - \Theta)^2)$. The efficient estimate minimizes this value.

Point estimates

These properties influence implementation of some sample characteristics in the R language.

Let us look on the sample mean. Having the random sample X_1, \dots, X_n from the distribution with the mean μ , we can calculate:

$$\mathbb{E}(\bar{x}) = \frac{1}{n} \sum_{i=1}^n \mathbb{E}(X_i) = \frac{1}{n} n\mu = \mu.$$

So we have shown that the average is the unbiased estimate of the mean.

Point estimates

Now let us look on the sample variance. Using the identity

$$\sum_{i=1}^n (X_i - \bar{X})^2 = \sum_{i=1}^n (X_i - \mu + \mu - \bar{X})^2 = \sum_{i=1}^n (X_i - \mu)^2 - n(\bar{X} - \mu)^2,$$

we get

$$\begin{aligned}\mathbb{E} \left(\sum_{i=1}^n (X_i - \bar{X})^2 \right) &= \mathbb{E} \left(\sum_{i=1}^n (X_i - \mu)^2 - n(\bar{X} - \mu)^2 \right) \\ &= \sum_{i=1}^n \mathbb{D}(X_i) - n\mathbb{D}(\bar{X}) \\ &= n\sigma^2 - \frac{n\sigma^2}{n} = (n-1)\sigma^2,\end{aligned}$$

Point estimates

Now let us look on the sample variance. Using the identity

$$\sum_{i=1}^n (X_i - \bar{X})^2 = \sum_{i=1}^n (X_i - \mu + \mu - \bar{X})^2 = \sum_{i=1}^n (X_i - \mu)^2 - n(\bar{X} - \mu)^2,$$

and therefore

$$\mathbb{E}(s^2) = \mathbb{E}\left(\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2\right) = \frac{n-1}{n} \sigma^2.$$

So we see the sample variance is not unbiased estimate of the random variable variance.

Point estimates

Now let us look on the sample variance. Using the identity

$$\sum_{i=1}^n (X_i - \bar{X})^2 = \sum_{i=1}^n (X_i - \mu + \mu - \bar{X})^2 = \sum_{i=1}^n (X_i - \mu)^2 - n(\bar{X} - \mu)^2,$$

In order to get the unbiased estimate we have to multiply the sample variance by $\frac{n}{n-1}$. So we obtain the unbiased estimate of the variance

$$s_{n-1}^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2.$$

This result explains, why the function `var()` is implemented differently from sample variance. It represents the unbiased estimate of the original random variable.

Point estimates

Methods

In this course we introduce two methods of the point estimates constructing:

- the method of moments,
- the method of maximal likelihood.

We will suppose, we have the sample X_1, \dots, X_n from the distribution that depends on the vector of parameters $\theta = (\theta_1, \dots, \theta_m)$.

The method of moments

Let us further suppose that there exist all moments $\nu_k = \mathbb{E}(X_i^k)$, $k = 1, \dots, m$.

The sample moments v_k are defined as $v_k = \frac{1}{n} \sum_{i=1}^n X_i^k$ for $k = 1, \dots, m$.

The principle of the method of moments is the equality of the theoretical and sample moments.

The method of moments

It means the method of moments estimator for $\theta_1, \theta_2, \dots, \theta_k$ denoted by $\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_k$ is defined as the solution (if there is one) of the equations:

$$\nu_k = v_k, \quad k = 1, \dots, m.$$

Alternatively to any r -th moment we can use the r -th central moment defined as $\mu_r = \mathbb{E}((X - \mathbb{E}(X))^r)$ and r -th sample central moment $m_r = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^r$.

The method of moments

We illustrate the method on the case of the uniform distribution with parameters a and b . It is known, that the moments of the uniformly distributed random variable X are

$$\mathbb{E}(X) = \frac{a+b}{2} \quad \text{and} \quad \mathbb{D}(X) = \frac{(b-a)^2}{12}.$$

To find the estimates of the parameters a and b we have to solve the equations:

$$\begin{aligned}\bar{x} &= \frac{a+b}{2}, \\ s^2 &= \frac{(b-a)^2}{12}.\end{aligned}$$

The method of moments

Solving these equations we obtain:

$$a = \bar{x} - \sqrt{3}s,$$

$$b = \bar{x} + \sqrt{3}s.$$

Now we are ready to implement this estimates in R language. We take advance the previously defined functions `variation()` resp. `stdev()`.

The method of moments

```
1 > x<-runif(1000,1,3) #generating the random sample
2 > a<-mean(x)-sqrt(3)*stdev(x)
3 > b<-mean(x)+sqrt(3)*stdev(x)
4 > a
5 [1] 1.018323 #can differ for other samples
6 > b
7 [1] 3.033395 #can differ for other samples
```

The maximum likelihood method

This method is based on maximizing a likelihood function.

The point in the parameter space that maximizes the likelihood function is said to be the maximum likelihood estimate.

Formally, let us assume that X_1, \dots, X_n are independent identically distributed random variables from the distribution with density $f(x, \theta)$. The joint density is the product of these univariate density functions:

$$L(x_1, \dots, x_n; \theta) = \prod_{i=1}^n f(x_i, \theta).$$

The maximum likelihood method

Just introduced function $L(x_1, \dots, x_n; \theta)$ is then called the **likelihood function**.

To get the estimates of the parameters, we maximize this function by standard process known from the mathematical analysis.

To make the work easier, we maximize the function $\ln L(x_1, \dots, x_n; \theta)$ instead of direct maximization of $L(x_1, \dots, x_n; \theta)$.

The natural logarithm is increasing function, therefore it save the extremes and moreover convert the product to the sum of functions.

The maximum likelihood method

We illustrate the method on the problem of estimating the probability p of some random event A .

We can interpret this situation as an results of the alternative random variable that is indicator of the random event A .

Therefore we have $\mathbb{P}(X = 1) = p$ and $\mathbb{P}(X = 0) = 1 - p$.

We perform n random trials and observe the occurrence of event A .

So we get a sample X_1, \dots, X_n of random variables with the densities $f(x_i, p) = p^{x_i}(1 - p)^{1-x_i}$, where $x_i \in \{0, 1\}$.

The maximum likelihood method

The corresponding likelihood function has the form

$$L(x, p) = \prod_{i=1}^n p^{x_i} (1-p)^{1-x_i} = p^{\sum_{i=1}^n x_i} (1-p)^{n - \sum_{i=1}^n x_i}.$$

In order to state the estimate of p we will maximize the function $L(x, p)$ with respect to the parameter p . To do so, we take natural logarithm of the likelihood function

$$\ln(L(x, p)) = \sum_{i=1}^n x_i \ln p + \left(n - \sum_{i=1}^n x_i \right) \ln(1-p),$$

The maximum likelihood method

We set its derivative with respect to p equal to 0:

$$\frac{d \ln L(x, p)}{dx} = \frac{\sum_{i=1}^n x_i}{p} - \frac{n - \sum_{i=1}^n x_i}{1 - p} = 0.$$

Multiplying the equation by $\frac{1}{n}$ we have

$$\bar{x} \cdot \frac{1}{p} - (1 - \bar{x}) \cdot \frac{1}{1 - p} = 0,$$

and its solution is

$$p = \bar{x}.$$

The maximum likelihood method

To confirm that $p = \bar{x}$ really maximizes the likelihood function, we have to verify the second order condition.

$$\frac{d^2 \ln L(x, p)}{dx^2} = -\frac{\bar{x}}{p} + (1 - \bar{x}) \cdot \frac{1}{(1 - p)^2}.$$

Substituting $p = \bar{x}$ we get

$$\left(\frac{d^2 \ln L(x, p)}{dx^2} \right)_{p=\bar{x}} = -\frac{1}{1 - \bar{x}} < 0.$$

So we have the most likely estimate $p = \bar{x}$.

The maximum likelihood method

We can use this approach to determine just how biased an unfair coin is.

At first we generate the sample of 0 and 1 which indicates tossing head or tail. To get sample of unfair coin, we declare the vector of probabilities `prob`, how we can see in the source code:

```
1 > x<-sample(c(0,1),1000,replace=TRUE,prob=c(2/3,1/3))
2 > mean(x)
3 [1] 0.304
```

Confidence intervals

The **confidence interval** can be defined as the range of estimates for an unknown parameter, that contains the real value of the parameter with given probability.

This probability that the parameter is within the given interval is reported as the **confidence level**.

The most common confidence level used in practice is 95 %, but other levels (such as 90 % or 99 %) are also frequently used.

Confidence intervals

Formally, let $\mathbf{X} = (X_1, \dots, X_n)$ is a random sample from distribution that depends on the unknown parameter θ .

A confidence interval for the parameter θ , with confidence level α , is an interval with random endpoints $(u(\mathbf{X}); v(\mathbf{X}))$, determined by the pair of random variables $u(\mathbf{X})$ and $v(\mathbf{X})$, with the property:

$$\mathbb{P}(u(\mathbf{X}) < \theta < v(\mathbf{X})) = \alpha.$$

Confidence intervals

The derivation of the confidence interval we illustrate on the normal distribution.

Let us assume, at first, that X_1, \dots, X_n is a random sample from the normal distribution $N(\mu, \sigma^2)$, whose standard deviation σ is known.

We want to find the confidence interval for the mean μ . Because \bar{X} has the normal distribution $N\left(\mu, \frac{\sigma^2}{n}\right)$, we have

$$\mathbb{P}\left(\left|\frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}}\right| < c\right) = 2\Phi(c) - 1, \text{ for all } c > 0,$$

Confidence intervals

It is equivalent to

$$\mathbb{P} \left(\bar{X} - c \frac{\sigma}{\sqrt{n}} < \mu < \bar{X} + c \frac{\sigma}{\sqrt{n}} \right) = 2\Phi(c) - 1, \text{ for all } c > 0.$$

From the last relation, it follows that the confidence interval for the mean with the confidence level $\alpha = 2\Phi(c) - 1$ has the form

$$\left(\bar{X} - c \frac{\sigma}{\sqrt{n}}; \bar{X} + c \frac{\sigma}{\sqrt{n}} \right).$$

If we take in account, that $c = \Phi^{-1} \left(\frac{\alpha+1}{2} \right)$, we can write the confidence interval in the form

$$\left(\bar{X} - \Phi^{-1} \left(\frac{\alpha+1}{2} \right) \frac{\sigma}{\sqrt{n}}; \bar{X} + \Phi^{-1} \left(\frac{\alpha+1}{2} \right) \frac{\sigma}{\sqrt{n}} \right).$$

Confidence intervals

We use the quantile function `qnorm()` to find the bounds of the confidence interval.

```
1 x<-rnorm(100,1,2) # we generate some sample
2 > n<-length(x)
3 > sample.mean<-mean(x)
4 > sample.sd<-2 # standard deviation is known
5 > alpha<-0.95 # setting the confidence level 95\%
6 > c<-qnorm((alpha+1)/2,0,1)
7 > margin<-c*sample.sd/sqrt(n)
8 > lower.bound<-sample.mean-margin
9 > upper.bound<-sample.mean+margin
10 > print(c(lower.bound,upper.bound))
11 [1] 0.8149884 1.5989740
```

Confidence intervals

Now let us see, how is the confidence interval changed, if the standard deviation is unknown.

Then we have to use the unbiased estimate of the standard deviation $s^2 = \frac{1}{n-1} \sum_{i=1}^n (\bar{X} - X_i)^2$, $s = \sqrt{s^2}$. Then the random variable

$$T = \frac{\bar{X} - \mu}{s} \sqrt{n},$$

follows the Student's t -distribution with $n - 1$ degrees of freedom. The confidence interval is then given

$$\left(\bar{X} - c \frac{s}{\sqrt{n}}; \bar{X} + c \frac{s}{\sqrt{n}} \right).$$

Confidence intervals

The value c is the corresponding quantile of the Student distribution, so we apply the `qt()` function in R.

```
1 > n<-length(x) # we use previously generated sample
2 > sample.mean<-mean(x)
3 > sample.sd<-sd(x) # estimate of the standard deviation
4 > alpha<-0.95
5 > c<-qt((alpha+1)/2,df=n-1)
6 > margin<-c*sample.sd/sqrt(n)
7 > lower.bound<-sample.mean-margin
8 > upper.bound<-sample.mean+margin
9 > print(c(lower.bound,upper.bound))
10 [1] 0.8118763 1.6020861
```

Confidence intervals

Let us now look at the confidence interval for the variance of the normal distribution.

If s^2 is the unbiased estimate of the variance, then the random variable

$$Y = \frac{(n-1)s^2}{\sigma^2} = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{\sigma^2} = \sum_{i=1}^n \left(\frac{x_i - \bar{x}}{\sigma} \right)^2$$

follows the $\chi^2(n-1)$ distribution.

Confidence intervals

For the confidence interval we get

$$\begin{aligned}\mathbb{P}(c_1 < \chi^2 < c_2) &= \alpha \\ \mathbb{P}\left(c_1 < \frac{(n-1)s^2}{\sigma^2} < c_2\right) &= \alpha \\ \mathbb{P}\left(\sigma \in \left(\frac{(n-1)s^2}{c_2}; \frac{(n-1)s^2}{c_1}\right)\right) &= \alpha,\end{aligned}$$

where c_1 and c_2 are the critical values of the $\chi^2(n-1)$ distribution.

Confidence intervals

When computing, we have to respect that χ^2 distribution is not symmetric, what is important for stating the critical values c_1 and c_2 .

```
1 > n<-length(x) # the sample already generated sooner
2 > sample.var<-var(x)
3 > c1<-qchisq(1-(alpha+1)/2,df=n-1)#consequent of asymmetry
4 > c2<-qchisq((alpha+1)/2,df=n-1)#consequent of asymmetry
5 > lower.bound<-sample.var*(n-1)/c2
6 > upper.bound<-sample.var*(n-1)/c1
7 > print(c(lower.bound,upper.bound))
8 [1] 3.056626 5.350767
```


Confidence intervals

As the last example, we will show the confidence interval for the probability of a random event.

The unbiased estimate of the probability p of a random event occurring is $\hat{p} = \frac{m}{n}$, where m is the number of occurrences of the observed event in a series of n random trials.

We know, that $\mathbb{E}(\hat{p}) = p$ and $\mathbb{D}(\hat{p}) = \frac{pq}{n}$, where $q = 1 - p$.

The approximative equality $\frac{pq}{n} \approx \frac{\hat{p}\hat{q}}{n}$ holds for the large n .

Confidence intervals

Therefore the random variable

$$Z = \frac{\frac{m}{n} - p}{\sqrt{\frac{\hat{p}\hat{q}}{n}}}$$

follows the standardized normal distribution $N(0, 1)$. Then we can write the confidence interval for the probability p with confidence level α as

$$\left(\hat{p} - \Phi^{-1} \left(\frac{\alpha + 1}{2} \right) \cdot \sqrt{\frac{\hat{p}\hat{q}}{n}}; \hat{p} + \Phi^{-1} \left(\frac{\alpha + 1}{2} \right) \cdot \sqrt{\frac{\hat{p}\hat{q}}{n}} \right).$$

Confidence intervals

Example

Suppose 250 randomly selected people are surveyed to determine if they own a tablet. Of the 250 surveyed, 98 reported owning a tablet. Using a 95 % confidence level, compute a confidence interval estimate for the true proportion of people who own tablets.

Thanks for your attention.



R programming for statistics

Aleš Kozubík

