Laborarator 2 – Probabilități și Statistică Matematică

TIPURI SI STRUCTURI DE DATE (1)

`R` are cinci tipuri de date principale (atomi), după cum urmează:

- **character**: `"a"`, `"swc"`
- **numeric**: `2`, `15.5`
- **integer**: `2L` (sufix-ul `L` îi spune R-ului să stocheze numărul ca pe un întreg)
- **logical**: `TRUE`, `FALSE`
- **complex**: `1+4i` (numere complexe)

R pune la dispoziție mai multe funcții cu ajutorul cărora se pot examina trăsăturile vectorilor sau a altor obiecte, cum ar fi de exemplu

- `class()` - ce tip de obiect este
- `typeof()` - care este tipul de date al obiectului
- `length()` - care este lungimea obiectului
- `attributes()` - care sunt atributele obiectului (metadata)

```
# Exemplu
x <- "curs probabilitati si statistica"
typeof(x)
[1] "character"
attributes(x)
NULL


y <- 1:10
y
 [1]  1  2  3  4  5  6  7  8  9 10
typeof(y)
[1] "integer"
length(y)
[1] 10


z <- as.numeric(y)
```

```
z
 [1]  1  2  3  4  5  6  7  8  9 10
typeof(z)
[1] "double"
```

În limbajul `R` regăsim mai multe **structuri de date**. Printre acestea enumerăm

- vectorii (structuri atomice)
- listele
- matricele
- data frame
- factori

3.1 Scalari și vectori

Cel mai de bază tip de obiect în R este vectorul. Una dintre regulile principale ale vectorilor este că aceștia pot conține numai obiecte de același tip, cu alte cuvinte putem avea doar vectori de tip caracter, numeric, logic, etc.. În cazul în care încercăm să combinăm diferite tipuri de date, acestea vor fi forțate la tipul cel mai flexibil. Tipurile de la cel mai puțin la cele mai flexibile sunt: logice, întregi, numerice și caractere.

3.1.1 Metode de construcție a vectorilor

Putem crea vectori fără elemente (empty) cu ajutorul funcției `vector()`, modul default este *logical* dar acesta se poate schimba în funcție de necesitate.

```
vector() # vector logic gol
logical(0)
vector("character", length = 5) # vector de caractere cu 5 eleme
nte
[1] "" "" "" "" ""
character(5) # acelasi lucru dar in mod direct
[1] "" "" "" "" ""
numeric(5)   # vector numeric cu 5 elemente
[1] 0 0 0 0 0
logical(5)   # vector logic cu 5 elemente
```

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

Putem crea vectori specificând în mod direct conținutul acestora. Pentru aceasta folosim funcția `c()` de concatenare:

```
x <- c(0.5, 0.6)        ## numeric
x <- c(TRUE, FALSE)     ## logical
x <- c(T, F)            ## logical
x <- c("a", "b", "c")   ## character
x <- 9:29               ## integer
x <- c(1+0i, 2+4i)      ## complex
```

Funcția poate fi folosită de asemenea și pentru (combinarea) adăugarea de elemente la un vector

```
z <- c("Sandra", "Traian", "Ionel")
z <- c(z, "Ana")
z
[1] "Sandra" "Traian" "Ionel"  "Ana"
z <- c("George", z)
z
[1] "George" "Sandra" "Traian" "Ionel"  "Ana"
```

O altă funcție des folosită în crearea vectorilor, în special a celor care au repetiții, este funcția `rep()`. Pentru a vedea documentația acestei funcții apelați `help(rep)`. De exemplu, pentru a crea un vector de lungime `5` cu elemente de `0` este suficient să scriem

```
rep(0, 5)
[1] 0 0 0 0 0
```

Dacă în plus vrem să creăm vectorul 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3 sau 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3 atunci

```
rep(c(1,2,3), 5)
 [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
rep(c(1,2,3), each = 5)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
```

- Ce se întâmplă dacă apelăm `rep(c(1,2,3), 1:3)` ?

În cazul în care vrem să creăm un vector care are elementele egal depărtate între ele, de exemplu 1.3, 2.3, 3.3, 4.3, 5.3, atunci putem folosi funcția `seq()`:

```
seq(1, 10, 1)
 [1]  1  2  3  4  5  6  7  8  9 10
1:10 # acelasi rezultat
 [1]  1  2  3  4  5  6  7  8  9 10
seq(1, 10, length.out = 15)
 [1]  1.000000  1.642857  2.285714  2.928571  3.571429  4.214286
4.857143
 [8]  5.500000  6.142857  6.785714  7.428571  8.071429  8.714286
9.357143
[15] 10.000000
```

3.1.2 Operații cu vectori

Operațiile elementare pe care le puteam face cu scalari (adunarea `+`, scăderea `-`, înmulțirea `*`, împărțirea `/` și ridicarea la putere `^`) putem să le facem și cu vectori (între vectori sau între vectori și scalari).

```
a = 1:4
b = c(5,5,6,7)


a+b  # adunarea
[1]  6  7  9 11
a+10 # adunarea cu scalari
[1] 11 12 13 14
```

```
a-b  # scaderea
[1] -4 -3 -3 -3
a-15 # scaderea cu scalari
[1] -14 -13 -12 -11
a*b # inmultirea
[1]  5 10 18 28
a*3 # inmultirea cu scalari
[1]  3  6  9 12
a/b # impartirea
[1] 0.2000000 0.4000000 0.5000000 0.5714286
a/100 # impartirea la scalari
[1] 0.01 0.02 0.03 0.04
a^b # ridicarea la putere
[1]     1    32   729 16384
a^7 # ridicarea la putere cu scalari
[1]     1   128  2187 16384
```

Observăm că atunci când facem o operație cu scalar, se aplică scalarul la fiecare element al vectorului.

Functiile elementare,
`exp()`, `log()`, `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()`, etc. sunt funcții vectoriale în R, prin urmare pot fi aplicate unor vectori.

```
x = seq(0, 2*pi, length.out = 20)


exp(x)
 [1]   1.000000   1.391934   1.937480   2.696843   3.753827   5.
225078
 [7]   7.272963  10.123483  14.091217  19.614041  27.301445  38.
001803
```

```
[13]  52.895992  73.627716 102.484902 142.652193 198.562402 276.
385707

[19] 384.710592 535.491656

sin(x)

 [1]  0.000000e+00  3.246995e-01  6.142127e-01  8.371665e-01  9.
694003e-01

 [6]  9.965845e-01  9.157733e-01  7.357239e-01  4.759474e-01  1.
645946e-01

[11] -1.645946e-01 -4.759474e-01 -7.357239e-01 -9.157733e-01 -9.
965845e-01

[16] -9.694003e-01 -8.371665e-01 -6.142127e-01 -3.246995e-01 -2.
449213e-16

tan(x)

 [1]  0.000000e+00  3.433004e-01  7.783312e-01  1.530614e+00  3.
948911e+00

 [6] -1.206821e+01 -2.279770e+00 -1.086290e+00 -5.411729e-01 -1.
668705e-01

[11]  1.668705e-01  5.411729e-01  1.086290e+00  2.279770e+00  1.
206821e+01

[16] -3.948911e+00 -1.530614e+00 -7.783312e-01 -3.433004e-01 -2.
449294e-16

atan(x)

 [1] 0.0000000 0.3193732 0.5843392 0.7814234 0.9234752 1.0268631
1.1039613

 [8] 1.1630183 1.2094043 1.2466533 1.2771443 1.3025194 1.3239406
1.3422495

[15] 1.3580684 1.3718664 1.3840031 1.3947585 1.4043537 1.4129651
```

Alte funcții utile des întâlnite în manipularea vectorilor numerici sunt: `min()`, `max()`, `sum()`, `mean()`, `sd()`, `length()`, `round()`, `ceiling()`, `floor()`, `%%` (operația modulo), `%/%` (div), `table()`, `unique()`. Pentru mai multe informații privind modul lor de întrebuințare apelați `help(nume_functie)` sau `?nume_functie`.

```
length(x)
```

```
[1] 20
min(x)
[1] 0
sum(x)
[1] 62.83185
mean(x)
[1] 3.141593


round(x, digits = 4)
 [1] 0.0000 0.3307 0.6614 0.9921 1.3228 1.6535 1.9842 2.3149 2.6
456 2.9762
[11] 3.3069 3.6376 3.9683 4.2990 4.6297 4.9604 5.2911 5.6218 5.9
525 6.2832


y =  c("M", "M", "F", "F", "F", "M", "F", "M", "F")
unique(y)
[1] "M" "F"
table(y)
y
F M
5 4
```

3.1.3 Metode de indexare a vectorilor

Sunt multe situațiile în care nu vrem să efectuăm operații pe întreg vectorul ci pe o submulțime de valori ale lui selecționate în funcție de anumite proprietăți. Putem, de exemplu, să ne dorim să accesăm al 2-lea element al vectorului sau toate elementele mai mari decât o anumită valoare. Pentru aceasta vom folosi operația de *indexare* folosind parantezele pătrate `[]`.

În general, sunt două tehnici principale de indexare: *indexarea numerică și indexarea logică.*

Atunci când folosim indexarea numerică, inserăm între parantezele pătrate un vector numeric ce corespunde elementelor pe care vrem să le accesăm sub forma `x[index]` (`x` este vectorul inițial iar `index` este vectorul de indici):

```
x = seq(1, 10, length.out = 21) # vectorul initial


x[1] # accesam primul element
[1] 1
x[c(2,5,9)] # accesam elementul de pe pozitia 2, 5 si 9
[1] 1.45 2.80 4.60
x[4:10] # accesam toate elementele deintre pozitiile 4 si 9
[1] 2.35 2.80 3.25 3.70 4.15 4.60 5.05
```

Putem folosi orice vector de indici atât timp cât el conține numere întregi. Putem să accesăm elementele vectorului `x` și de mai multe ori:

```
x[c(1,1,2,2)]
[1] 1.00 1.00 1.45 1.45
```

De asemenea dacă vrem să afișăm toate elementele mai puțin elementul de pe poziția `i` atunci putem folosi indexare cu numere negative (această metodă este folositoare și în cazul în care vrem să ștergem un element al vectorului):

```
x[-5] # toate elementele mai putin cel de pe pozitia 5
 [1]  1.00  1.45  1.90  2.35  3.25  3.70  4.15  4.60  5.05  5.50
5.95
[12]  6.40  6.85  7.30  7.75  8.20  8.65  9.10  9.55 10.00
x[-(1:3)] # toate elementele mai putin primele 3
 [1]  2.35  2.80  3.25  3.70  4.15  4.60  5.05  5.50  5.95  6.40
6.85
[12]  7.30  7.75  8.20  8.65  9.10  9.55 10.00
x = x[-10] # vectorul x fara elementul de pe pozitia a 10-a
```

A doua modilitate de indexare este cu ajutorul vectorilor logici. Atunci când indexăm cu un vector logic acesta trebuie să aibă aceeași lungime ca și vectorul pe care vrem să îl indexăm.

Să presupunem că vrem să extragem din vectorul `x` doar elementele care verifică o anumită proprietate, spre exemplu sunt mai mari decât 3, atunci:

```
x>3 # un vector logic care ne arata care elemente sunt mai mari
decat 3
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
[12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
x[x>3] # elementele din x care sunt mai mari decat 3
 [1]  3.25  3.70  4.15  4.60  5.50  5.95  6.40  6.85  7.30  7.75
8.20
[12]  8.65  9.10  9.55 10.00
```

Pentru a determina care sunt toate elementele din `x` cuprinse între 5 și 19 putem să folosim operații cu operatori logici:

```
x[(x>5)&(x<19)]
 [1]  5.50  5.95  6.40  6.85  7.30  7.75  8.20  8.65  9.10  9.55
10.00
```

O listă a operatorilor logici din R se găsește în tabelul următor:

Tabelul 2. Operatori logici

Operator	Descriere
==	Egal
!=	Diferit
<	Mai mic
<=	Mai mic sau egal
>	Mai mare
>=	Mai mare sau egal

Tabelul 2. Operatori logici

Operator	Descriere
`|` sau `||`	Sau (primul are valori vectoriale al doilea scalare)
`&` sau `&&`	Și (primul are valori vectoriale al doilea scalare)
!	Negație
`%in%`	În mulțimea

```
x = seq(1,10,length.out = 8)
x == 3
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
x != 3
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
x <= 8.6
[1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
(x<8) & (x>2)
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
(x<8) && (x>2)
[1] FALSE
(x<7) | (x>3)
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
(x<7) || (x>3)
[1] TRUE
x %in% c(1,9)
[1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

- Să presupunem că am înregistrat în fiecare zi, pe parcursul a 4 săptămâni (de Luni până Duminică), numărul de minute petrecute la telefonul mobil (convorbiri + utilizare) și am obținut următoarele valori: 106, 123, 123, 111, 125, 113, 130, 113, 114, 100, 120, 130, 118, 114, 127, 112, 121, 114, 120, 119, 127, 114, 108,

127, 131, 157, 102, 133. Ne întrebăm: care sunt zilele din săptămână în care am vorbit cel mai mult? dar cel mai puțin? dar zilele în care am vorbit mai mult de 120 de minute?

3.2 Matrice

Matricele sunt structuri de date care extind vectorii și sunt folosite la representarea datelor de același tip în două dimensiuni. Matricele sunt similare tablourilor din Excel și pot fi văzute ca vectori cu două atribute suplimentare: numărul de linii (*rows*) și numărul de coloane (*columns*).

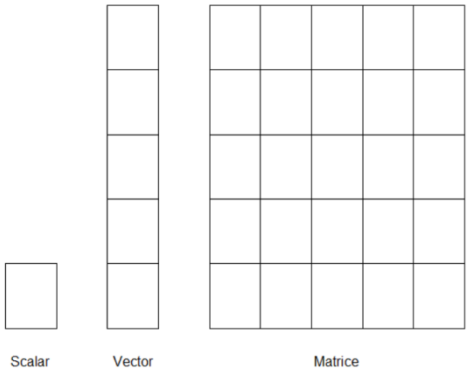

Figura 3. Scalari, Vectori, Matrice

Indexarea liniilor și a coloanelor pentru o matrice începe cu 1. De exemplu, elementul din colțul din stânga sus al unei matrice este notat cu `x[1,1]`. De asemenea este important de menționat că stocarea (internă) a metricelor se face pe coloane în sensul că prima oară este stocată coloana 1, apoi coloana 2, etc..

Există mai multe moduri de creare a unei matrici în R. Funcțiile cele mai uzuale sunt prezentate în tabelul de mai jos. Cum matricele sunt combinații de vectori, fiecare funcție primește ca argument unul sau mai mulți vectori (toți de același tip) și ne întoarce o matrice.

Tabelul 3. Functii care permit crearea matricelor

Funcție	Descriere	Exemple
`cbind(a, b, c)`	Combină vectorii ca și coloane	`cbind(1:5, 6:10, 11:15)`

Tabelul 3. Functii care permit crearea matricelor

Funcție	Descriere	Exemple
	într-o matrice	
`rbind(a, b, c)`	Combină vectorii ca și linii într-o matrice	`rbind(1:5, 6:10, 11:15)`
`matrix(x, nrow, ncol, byrow)`	Crează o matrice dintr-un vector `x`	`matrix(x = 1:12, nrow = 3, ncol = 4)`

Pentru a vedea ce obținem atunci când folosim funcțiile `cbind()` și `rbind()` să considerăm exemplele următoare:

```
x <- 1:5
y <- 6:10
z <- 11:15

# Cream o matrice cu x, y si z ca si coloane
cbind(x, y, z)
     x  y  z
[1,] 1  6 11
[2,] 2  7 12
[3,] 3  8 13
[4,] 4  9 14
[5,] 5 10 15

# Cream o matrice in care x, y si z sunt linii
rbind(x, y, z)
  [,1] [,2] [,3] [,4] [,5]
x    1    2    3    4    5
y    6    7    8    9   10
z   11   12   13   14   15
```

Funcția `matrix()` crează o matrice plecând de la un singur vector. Funcția are patru valori de intrare: `data` – un vector cu date, `nrow` – numărul de linii pe care le vrem în

matrice, `ncol` – numărul de coloane pe care să le aibe matricea și `byrow` – o valoare logică care permite crearea matricei pe linii (nu pe coloane cum este default-ul).

```
# matrice cu 5 linii si 2 coloane
matrix(data = 1:10,
       nrow = 5,
       ncol = 2)
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
# matrice cu 2 linii si 5 coloane
matrix(data = 1:10,
       nrow = 2,
       ncol = 5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
# aceeasi matrice cu 2 linii si 5 coloane, umpluta pe linii
matrix(data = 1:10,
       nrow = 2,
       ncol = 5,
       byrow = TRUE)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
```

Operațiile uzuale cu vectori se aplică și matricelor. Pe lângă acestea avem la dispoziție și operații de algebră liniară clasice, cum ar fi determinarea dimensiunii acestora, transpunerea matricelor sau înmulțirea lor:

```
m = matrix(data = 1:10,
       nrow = 2,
       ncol = 5)
m
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
dim(m) # dimensiunea matricei
[1] 2 5
nrow(m) # numarul de linii
[1] 2
ncol(m) # numarul de coloane
[1] 5
tpm = t(m) # transpusa
tpm
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    9   10
m %*% tpm # inmultirea matricelor
     [,1] [,2]
[1,]  165  190
[2,]  190  220
```

Metodele de indexare discutate pentru vectori se aplică și în cazul matricelor (`[,]`) numai că acum în loc să folosim un vector să indexăm putem să folosim doi vectori. Sintaxa are structura generală `m[linii, coloane]` unde `linii` și `coloane` sunt vectori cu valori întregi.

```
m = matrix(1:20, nrow = 4, byrow = TRUE)
# Linia 1
m[1, ]
[1] 1 2 3 4 5
# Coloana 5
m[, 5]
[1]  5 10 15 20
# Liniile 2, 3 si coloanele 3, 4
m[2:3, 3:4]
     [,1] [,2]
[1,]    8    9
[2,]   13   14
# Elementele din coloana 3 care corespund liniilor pentru care e
lementele
# de pe prima coloana sunt > 3
m[m[,1]>3, 3]
[1]  8 13 18
```