

Laborator 1

Introducere în R

Obiectivul acestui laborator este de a prezenta o scurtă introducere în programul **R** (cu ajutorul interfeței grafice **RStudio**). O descriere detaliată a acestui program precum și versiunile disponibile pentru descărcat se găsesc pe site-ul www.r-project.org. Pentru mai multe detalii se pot consulta [Davies, 2016] sau [Matloff, 2011].

1 Introducere

Programul **R** este un program **gratuit** destinat, cu precădere, analizei statistice și prezintă o serie de avantaje:

- rulează aproape pe toate platformele și sistemele de operare
- permite folosirea metodelor statistice clasice cu ajutorul unor funcții predefinite
- este adoptat ca limbaj de analiză statistică în majoritatea domeniilor aplicate
- prezintă capabilități grafice deosebite
- permite utilizarea tehniciilor statistice de ultimă oră prin intermediul pachetelor dezvoltate de comunitate (în prezent sunt mai mult de 10000 de pachete)
- are o comunitate foarte activă și în continuă creștere

Numarul de pachete în funcție de versiunea de R

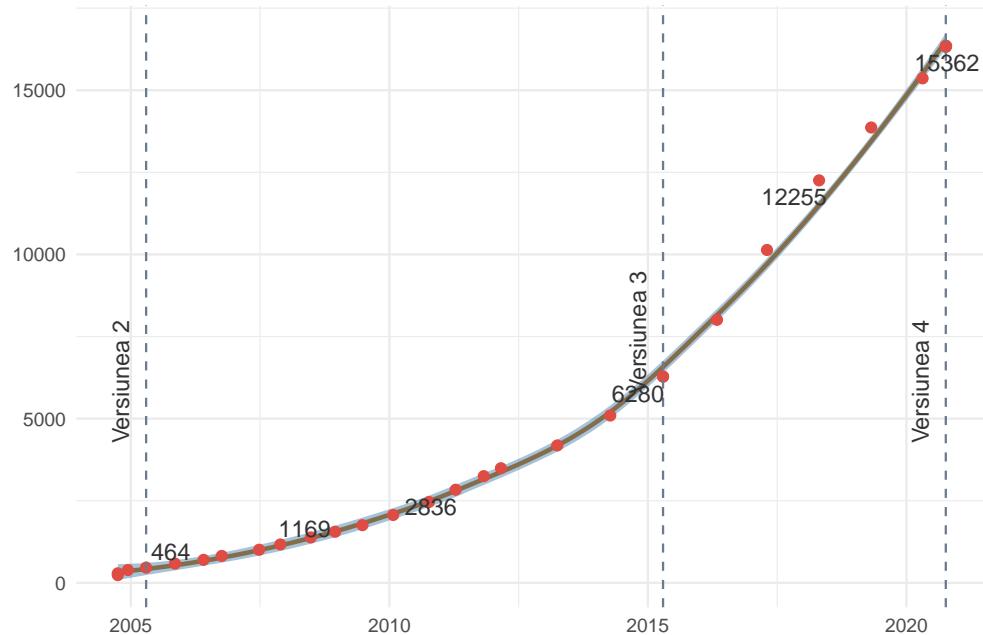


Fig. 1: Numarul de pachete din R

1.1 Interfața RStudio

Interfața RStudio (vezi Figura 1) este compusă din patru ferestre:

- *Fereastra de editare* (stânga sus): în această fereastră apar fișierele, de tip **script**, în care utilizatorul dezvoltă propriile funcții ori script-uri.
- *Fereastra de comandă sau consola* (stânga jos): în această fereastră sunt executate comenzi R
- *Fereastra cu spațiul de lucru/istoricul* (dreapta sus): conține obiectele definite în memorie și istoricul comenzi folosite
- *Fereastra de explorare* (dreapta jos): în această fereastră ne putem deplasa în interiorul repertoriului (tab-ul *Files*), putem vedea graficele trase (tab-ul *Plots*) dar și pachetele instalate (tab-ul *Packages*). De asemenea, tot în această fereastră putem să căutăm documentația despre diferite funcții, folosind fereastra de ajutor (tab-ul *Help*).

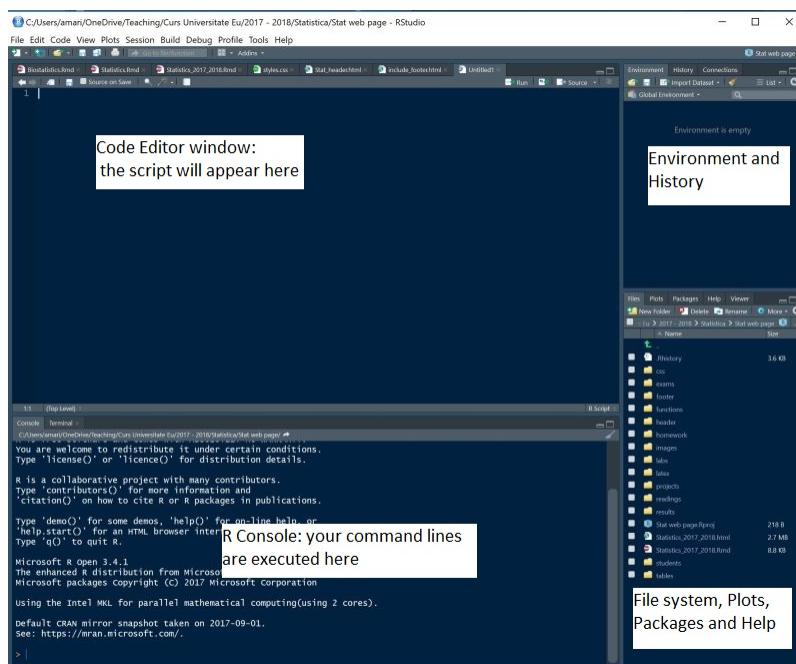


Fig. 2: Interfata RStudio

1.2 Pachetele ajutătoare

Pe lângă diferitele pachete conținute în versiunea de bază a programului R se mai pot instala și pachete suplimentare. Pentru a instala un pachet suplimentar se apelează comanda:

```
install.packages("nume_pachet")
```

Odată ce pachetul este instalat, pentru a încărca pachetul, și prin urmare funcțiile disponibile în acesta, se apelează comanda:

```
library("nume_pachet")
```

Instalarea unui pachet se face o singură dată dar încărcarea acestuia trebuie făcută de fiecare dată când lansăm o sesiune nouă.

2 Primele comenzi în R

2.1 Calcul elementar

Programul R poate fi folosit și pe post de calculator (mai avansat). De exemplu putem face calcule elementare

```
5 - 1 + 10
[1] 14
7 * 10 / 2
[1] 35
exp(-2.19)
[1] 0.1119167
pi
[1] 3.141593
sin(2 * pi/3)
[1] 0.8660254
```

De asemenea, rezultatele pot fi stocate într-o variabilă

```
a = (1+sqrt(5)/2)/2
```

păstrată în memorie (a apare în fereastra de lucru - Environment) și care poate fi reutilizată ulterior

```
asq = sqrt(a)
asq
[1] 1.029086
```

Operațiile binare în R sunt date în tabelul de mai jos:

Tab. 1: Operatii binare in R

Operatorul	Descriere
+	Adunare
-	Scădere
*	Înmulțire
/	Împărțire
^ or **	Exponențiere
%/%	Câtul împărțirii
%%	Restul împărțirii

Pentru a șterge toate variabilele din memorie trebuie să folosim comanda următoare (funcția `ls()` listeză numele obiectelor din memorie iar comanda `rm()` șterge obiectele; de asemenea se poate folosi și comanda `ls.str()` pentru a lista obiectele împreună cu o scurtă descriere a lor)

```
ls.str()
rm(list = ls())
```

2.2 Folosirea documentației

Funcția `help()` și operatorul de ajutor `?` ne permite accesul la paginile de documentație pentru funcțiile, seturile de date și alte obiecte din R. Pentru a accesa documentația pentru funcția standard `mean()` putem să folosim comanda `help(mean)` sau `?mean` în consolă. Pentru a accesa documentația unei funcții dintr-un pachet care nu este în prezent încărcat (dar este instalat) trebuie să adăugăm în plus numele pachetului, de exemplu `help(rlm, package = "MASS")` iar pentru a accesa documentația întregului pachet putem folosi comanda `help(package = "MASS")`.

O altă funcție de căutare des utilizată, în special în situația în care nu știm cu exactitate numele obiectului pe care îl căutăm, este funcția `apropos()`. Aceasta permite căutarea obiectelor (inclusiv funcții), disponibile în pachetele încărcate în sesiunea curentă, după un sir de caractere specificat (se pot folosi și expresii regulate). De exemplu dacă apelăm `apropos("mean")` vom obține toate funcțiile care conțin sirul de caractere `mean`.

```

apropos("mean") # functii care contin mean
[1] ".colMeans"      ".rowMeans"      "colMeans"       "kmeans"
[5] "mean"           "mean.Date"      "mean.default"   "mean.difftime"
[9] "mean.POSIXct"   "mean.POSIXlt"   "mean_cl_boot"   "mean_cl_normal"
[13] "mean_sdl"       "mean_se"        "rowMeans"      "weighted.mean"

apropos("^mean") # functii care incep cu mean
[1] "mean"           "mean.Date"      "mean.default"   "mean.difftime"
[5] "mean.POSIXct"   "mean.POSIXlt"   "mean_cl_boot"   "mean_cl_normal"
[9] "mean_sdl"       "mean_se"

```

Următorul tabel prezintă funcțiile de ajutor, cel mai des utilizate:

Tab. 2: Functii folosite pentru ajutor

Funcție	Acțiune
<code>help.start()</code>	Modul de ajutor general
<code>help("nume") sau ?nume</code>	Documentație privind funcția <i>nume</i> (ghilimelele sunt optionale)
<code>help.search(nume) sau ??nume</code>	Caută sistemul de documentație pentru instanțe în care apare sirul de caractere <i>nume</i>
<code>example("nume")</code> <code>RSiteSearch("nume")</code>	Exemple de utilizare ale funcției <i>nume</i> Caută sirul de caractere <i>nume</i> în manualele online și în arhivă
<code>apropos("nume", mode = "functions")</code>	Listează toate funcțiile care conțin sirul <i>nume</i> în numele lor
<code>data()</code>	Listează toate seturile de date disponibile în pachetele încărcate
<code>vignette()</code> <code>vignette("nume")</code>	Listează toate vignetele disponibile Afisează vignetele corespunzătoare topicului <i>nume</i>

Documentații online:

Tab. 3: O serie de link-uri utile

CheatSheet	Link
R de bază	https://cran.r-project.org/doc/contrib/Short-refcard.pdf
Noțiuni de R avansat	https://www.rstudio.com/wp-content/uploads/2016/02/advancedR.pdf
Noțiuni de R de bază	http://github.com/rstudio/cheatsheets/raw/master/base-r.pdf
Manipularea sirurilor de caractere	https://github.com/rstudio/cheatsheets/raw/master/strings.pdf
Importarea datelor	https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf
Transformarea datelor	https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf
RStudio	https://github.com/rstudio/cheatsheets/raw/master/rstudio-ide.pdf
Pachetul ggplot2	https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf
Pachetul RMarkdown	https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

3 Tipuri și structuri de date

R are cinci tipuri de date principale (atomii), 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 aceştia 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 elemente
[1] "" "" "" ""
character(5) # același lucru dar în 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 sau 1, 1, 1, 1, 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 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 # același 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
```

Tab. 4: Functii utile pentru crearea unui vector

Function	Example	Result
c(a, b, ...)	c(1, 5, 9)	1, 5, 9
a:b	1:5	1, 2, 3, 4, 5
seq(from, to, by, length.out)	seq(from = 0, to = 6, by = 2)	0, 2, 4, 6
rep(x, times, each, length.out)	rep(c(7, 8), times = 2, each = 2)	7, 7, 8, 8, 7, 7, 8, 8

3.1.2 Operații cu vectori

Operațiile elementare pe care le putem 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 întrebunțare 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 2.6456 2.9762
[11] 3.3069 3.6376 3.9683 4.2990 4.6297 4.9604 5.2911 5.6218 5.9525 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 stergem 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 6.40
[13] 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 7.30
[13] 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 modalitate 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
[13] 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 8.65
[13] 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:

Tab. 5: Operatori logici

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

Operator	Descriere
<code> sau </code>	Sau (primul are valori vectoriale al doilea scalare)
<code>& sau &&</code>	Și (primul are valori vectoriale al doilea scalare)
<code>!</code>	Negatie
<code>%in%</code>	În multimea

```
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 (con vorbiri + 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 reprezentarea datelor de același tip în două dimensiuni. Matricele sunt similare tablourilor din Excel și pot fi văzute ca vectori cu două attribute suplimentare: numărul de linii (*rows*) și numărul de coloane (*columns*).

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 metricei 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.

Tab. 6: Functii care permit crearea matricelor

Funcție	Descriere	Exemple
<code>cbind(a, b, c)</code>	Combină vectorii ca și coloane într-o matrice	<code>cbind(1:5, 6:10, 11:15)</code>
<code>rbind(a, b, c)</code>	Combină vectorii ca și linii într-o matrice	<code>rbind(1:5, 6:10, 11:15)</code>

Funcție	Descriere	Exemple
<code>matrix(x, nrow, ncol, byrow)</code>	Crează o matrice dintr-un vector x	<code>matrix(x = 1:12, nrow = 3, ncol = 4)</code>

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

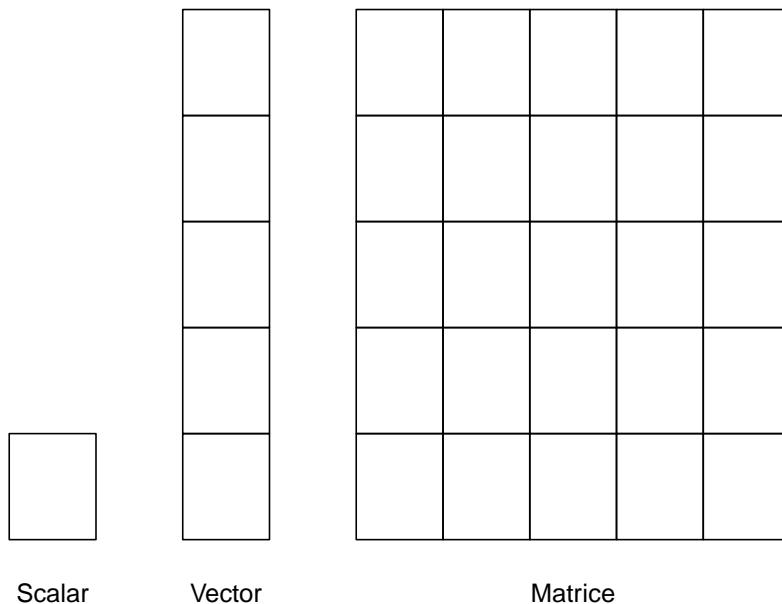


Fig. 3: Scalari, Vectori, Matrice

```

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 aibă 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

# aceeași 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:

```
diag(M) # Diagonala matricei M
dim(M) # Dimensiunile matricei M
nrow(M) # Numarul de linii ale matricei M
ncol(M) # Numarul de coloane ale matricei M
t(M) # Transpusa
colSums(M), rowSums(M) # Suma pe coloane și suma pe linii
```

De exemplu:

```
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
```

Adunarea și scăderea matricelor se face pe componenete:

```
A = matrix(c(1, 3, 2, 2, 2, 1, 3, 1, 3), ncol = 3)
B = matrix(c(4, 6, 4, 5, 5, 6, 6, 4, 5), ncol = 3)
a = 2
```

```
A + a
[,1] [,2] [,3]
[1,] 3 4 5
[2,] 5 4 3
[3,] 4 3 5
A + B
[,1] [,2] [,3]
[1,] 5 7 9
[2,] 9 7 5
[3,] 6 7 8
A - B
[,1] [,2] [,3]
[1,] -3 -3 -3
[2,] -3 -3 -3
[3,] -2 -5 -2
```

Înmulțirea și împărțirea se face tot pe componenete

```
A * a
[,1] [,2] [,3]
[1,] 2 4 6
[2,] 6 4 2
[3,] 4 2 6
A * B
[,1] [,2] [,3]
[1,] 4 10 18
[2,] 18 10 4
[3,] 8 6 15
A / B
[,1] [,2] [,3]
[1,] 0.25 0.4000000 0.50
[2,] 0.50 0.4000000 0.25
[3,] 0.50 0.1666667 0.60
```

Transpusa unei matrice (A^\top) se obține cu ajutorul funcției `t()`

```
t(A)
[,1] [,2] [,3]
[1,] 1 3 2
[2,] 2 2 1
[3,] 3 1 3
```

iar inversa (A^{-1}) cu ajutorul funcției `solve()`

```
solve(A)
[,1] [,2] [,3]
[1,] -0.41666667 0.25 0.3333333
[2,] 0.58333333 0.25 -0.6666667
[3,] 0.08333333 -0.25 0.3333333
```

Înmulțirea (uzuală) a matricelor se face folosind operatorul `%*%`

```
A %*% B # inmultirea matricelor
[,1] [,2] [,3]
[1,] 28 33 29
[2,] 28 31 31
```

```
[3,] 26 33 31
```

iar funcția `crossprod()` calculează produsul $A^T B$ (mai repede decât folosind instrucțiunea `t(A) %*% B`)

```
crossprod(A, B)
[,1] [,2] [,3]
[1,] 30   32   28
[2,] 24   26   25
[3,] 30   38   37
```

Determinantul și urma unei matrice se obțin folosind funcțiile `det()` și respectiv `sum(diag())`

```
det(A) # determinantul
[1] -12
sum(diag(A)) # urma matricei A
[1] 6
```

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 și 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 elementele
# de pe prima coloana sunt > 3
m[m[,1]>3, 3]
[1] 8 13 18
```

3.3 Liste

Spre deosebire de vectori în care toate elementele trebuie să aibă același tip de date, structura de date din R de tip listă (`list`) permite combinarea obiectelor de mai multe tipuri. Cu alte cuvinte, o listă poate avea primul element un scalar, al doilea un vector, al treilea o matrice iar cel de-al patrulea element poate fi o altă listă. Tehnic listele sunt tot vectori, vectorii pe care i-am văzut anterior se numesc *vectori atomici*, deoarece elementele lor nu se pot diviza, pe când listele se numesc *vectori recursivi*.

Ca un prim exemplu să considerăm cazul unei baze de date de angajați. Pentru fiecare angajat, ne dorim să stocăm numele angajatului (șir de caractere), salariul (valoare numerică) și o valoare de tip logic care poate reprezenta apartenența într-o asociație. Pentru crearea listei folosim funcția `list()`:

```
a = list(nume = "Ionel", salariu = 1500, apartenenta = T)
a
$nume
```

```
[1] "Ionel"

$salariu
[1] 1500

$apartenenta
[1] TRUE
str(a) # structura listei
List of 3
$ nume      : chr "Ionel"
$ salariu   : num 1500
$ apartenenta: logi TRUE
names(a) # numele listei
[1] "nume"      "salariu"    "apartenenta"
```

Numele componentelor listei a (nume, salariu, apartenenta) nu sunt obligatorii dar cu toate acestea pentru claritate sunt indicate:

```
a2 = list("Ionel", 1500, T)
a2
[[1]]
[1] "Ionel"

[[2]]
[1] 1500

[[3]]
[1] TRUE
```

Deoarece listele sunt vectori ele pot fi create și prin intermediul funcției `vector()`:

```
z <- vector(mode="list")
z
list()
z[["a"]] = 3
z
$a
[1] 3
```

3.3.1 Indexarea listelor

Elementele unei liste pot fi accesate în diferite moduri. Dacă dorim să extragem primul element al listei atunci vom folosi indexarea care folosește o singură paranteze pătrate []

```
a[1]
$nume
[1] "Ionel"
a[2]
$salariu
[1] 1500

# ce obținem cand extragem un element al listei a ?
str(a[1])
List of 1
$ nume: chr "Ionel"
```

În cazul în care vrem să accesăm structura de date corespunzătoare elementului i al listei vom folosi două perechi de paranteze pătrate `[]` sau în cazul în care lista are nume operatorul `$` urmat de numele elementului i.

```
a[[1]]  
[1] "Ionel"  
a[[2]]  
[1] 1500  
  
a$nume  
[1] "Ionel"  
a[["nume"]]  
[1] "Ionel"
```

Operațiile de adăugare, respectiv ștergere, a elementelor unei liste sunt des întâlnite.

Putem adăuga elemente după ce o listă a fost creată folosind numele componentei

```
z = list(a = "abc", b = 111, c = c(TRUE, FALSE))  
z  
$a  
[1] "abc"  
  
$b  
[1] 111  
  
$c  
[1] TRUE FALSE  
z$d = "un nou element"  
z  
$a  
[1] "abc"  
  
$b  
[1] 111  
  
$c  
[1] TRUE FALSE  
  
$d  
[1] "un nou element"
```

sau indexare vectorială

```
z[[5]] = 200  
z[6:7] = c("unu", "doi")  
z  
$a  
[1] "abc"  
  
$b  
[1] 111  
  
$c  
[1] TRUE FALSE
```

```
$d
[1] "un nou element"

[[5]]
[1] 200

[[6]]
[1] "unu"

[[7]]
[1] "doi"
```

Putem șterge o componentă a listei atribuindu-i valoarea NULL:

```
z[4] = NULL
z
$a
[1] "abc"

$b
[1] 111

$c
[1] TRUE FALSE

[[4]]
[1] 200

[[5]]
[1] "unu"

[[6]]
[1] "doi"
```

Putem de asemenea să concatenăm două liste folosind funcția `c()` și să determinăm lungimea noii liste cu funcția `length()`.

```
11 = list(1:10, matrix(1:6, ncol = 3), c(T, F))
12 = list(c("Ionel", "Maria"), seq(1,10,2))

13 = c(11, 12)
length(13)
[1] 5
str(13)
List of 5
 $ : int [1:10] 1 2 3 4 5 6 7 8 9 10
 $ : int [1:2, 1:3] 1 2 3 4 5 6
 $ : logi [1:2] TRUE FALSE
 $ : chr [1:2] "Ionel" "Maria"
 $ : num [1:5] 1 3 5 7 9
```

3.4 Data frame-uri

La nivel intuitiv, o structură de date de tip *data frame* este ca o matrice, având o structură bidimensională cu linii și coloane. Cu toate acestea ea diferă de structura de date de tip matrice prin faptul că fiecare coloană

poate avea tipuri de date diferite. Spre exemplu, o coloană poate să conțină valori numerice pe când o alta, valori de tip caracter sau logic. Din punct de vedere tehnic, o structură de tip data frame este o listă a cărei componente sunt vectori (atomici) de lungimi egale.

Pentru a crea un dataframe din vectori putem folosi funcția `data.frame()`. Această funcție funcționează similar cu funcția `list()` sau `cbind()`, diferența față de `cbind()` este că avem posibilitatea să dăm nume coloanelor atunci când le unim. Dată fiind flexibilitatea acestei structuri de date, majoritatea seturilor de date din R sunt stocate sub formă de dataframe (această structură de date este și cea mai des întâlnită în analiza statistică).

Să creăm un dataframe simplu numit `survey` folosind funcția `data.frame()`:

```
survey <- data.frame("index" = c(1, 2, 3, 4, 5),
                      "sex" = c("m", "m", "m", "f", "f"),
                      "age" = c(99, 46, 23, 54, 23))

survey
  index sex age
1     1   m  99
2     2   m  46
3     3   m  23
4     4   f  54
5     5   f  23
```

Funcția `data.frame()` prezintă un argument specific numit `stringsAsFactors` care permite convertirea coloanelor ce conțin elemente de tip caracter într-un tip de obiect numit `factor`. Un factor este o variabilă nominală care poate lua un număr bine definit de valori. De exemplu, putem crea o variabilă de tip factor `sex` care poate lua doar două valori: *masculin* și *feminin*. Comportamentul implicit al funcției `data.frame()` (`stringAsFactors = TRUE`) transformă automat coloanele de tip caracter în factor, motiv pentru care trebuie să includem argumentul `stringsAsFactors = FALSE`.

```
# Structura initială
str(survey)

'data.frame': 5 obs. of 3 variables:
 $ index: num 1 2 3 4 5
 $ sex   : chr "m" "m" "m" "f" ...
 $ age   : num 99 46 23 54 23

survey <- data.frame("index" = c(1, 2, 3, 4, 5),
                      "sex" = c("m", "m", "m", "f", "f"),
                      "age" = c(99, 46, 23, 54, 23),
                      stringsAsFactors = FALSE)

# Structura de după
str(survey)

'data.frame': 5 obs. of 3 variables:
 $ index: num 1 2 3 4 5
 $ sex   : chr "m" "m" "m" "f" ...
 $ age   : num 99 46 23 54 23
```

R are mai multe funcții care permit vizualizarea structurilor de tip dataframe. Tabelul de mai jos include câteva astfel de funcții:

Tab. 7: Exemple de funcții necesare pentru înțelegerea structurii dataframe-ului

Funcție	Descriere
<code>head(x)</code> , <code>tail(x)</code>	Printarea primelor linii (sau ultimelor linii).
<code>View(x)</code>	Vizualizarea obiectului într-o fereastră nouă, tabelară.
<code>nrow(x)</code> , <code>ncol(x)</code> , <code>dim(x)</code>	Numărul de linii și de coloane.
<code>rownames()</code> , <code>colnames()</code> , <code>names()</code>	Numele liniilor sau coloanelor.
<code>str(x)</code>	Structura dataframe-ului

```

data() # vedem ce seturi de date există

# Alegem setul de date mtcars
?mtcars
str(mtcars) # structura setului de date
'data.frame': 32 obs. of 11 variables:
 $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num 160 160 108 258 360 ...
 $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num 16.5 17 18.6 19.4 17 ...
 $ vs : num 0 0 1 1 0 1 0 1 1 1 ...
 $ am : num 1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num 4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num 4 4 1 1 2 1 4 2 2 4 ...
head(mtcars)
      mpg cyl disp hp drat    wt  qsec vs am gear carb
Mazda RX4     21.0   6 160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag 21.0   6 160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710    22.8   4 108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive 21.4   6 258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360 175 3.15 3.440 17.02  0  0    3    2
Valiant       18.1   6 225 105 2.76 3.460 20.22  1  0    3    1
tail(mtcars)
      mpg cyl disp hp drat    wt  qsec vs am gear carb
Porsche 914-2 26.0   4 120.3 91 4.43 2.140 16.7   0  1    5    2
Lotus Europa  30.4   4  95.1 113 3.77 1.513 16.9   1  1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5   0  1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5   0  1    5    6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6   0  1    5    8
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6   1  1    4    2

rownames(mtcars)
[1] "Mazda RX4"           "Mazda RX4 Wag"        "Datsun 710"
[4] "Hornet 4 Drive"       "Hornet Sportabout"   "Valiant"
[7] "Duster 360"          "Merc 240D"           "Merc 230"
[10] "Merc 280"             "Merc 280C"            "Merc 450SE"
[13] "Merc 450SL"           "Merc 450SLC"          "Cadillac Fleetwood"
[16] "Lincoln Continental"  "Chrysler Imperial"   "Fiat 128"
[19] "Honda Civic"          "Toyota Corolla"       "Toyota Corona"
[22] "Dodge Challenger"     "AMC Javelin"          "Camaro Z28"

```

```
[25] "Pontiac Firebird"    "Fiat X1-9"        "Porsche 914-2"  
[28] "Lotus Europa"       "Ford Pantera L"   "Ferrari Dino"  
[31] "Maserati Bora"      "Volvo 142E"  
names(mtcars)  
[1] "mpg"   "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"  
[11] "carb"  
  
View(mtcars)
```

3.4.1 Metode de indexare

Indexarea structurilor de tip dataframe se face la fel ca și indexarea listelor.

```
mtcars[1,1:4]  
      mpg cyl disp hp  
Mazda RX4  21   6 160 110  
mtcars[c(1,2),2]  
[1] 6 6  
mtcars$mpg  
[1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4  
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7  
[31] 15.0 21.4
```

La fel ca vectorii, dataframe-urile (dar și listele) pot fi indexate logic

```
mtcars[mtcars$mpg > 25, ]  
      mpg cyl disp hp drat wt qsec vs am gear carb  
Fiat 128  32.4  4 78.7 66 4.08 2.200 19.47 1 1 4 1  
Honda Civic 30.4  4 75.7 52 4.93 1.615 18.52 1 1 4 2  
Toyota Corolla 33.9  4 71.1 65 4.22 1.835 19.90 1 1 4 1  
Fiat X1-9  27.3  4 79.0 66 4.08 1.935 18.90 1 1 4 1  
Porsche 914-2 26.0  4 120.3 91 4.43 2.140 16.70 0 1 5 2  
Lotus Europa 30.4  4 95.1 113 3.77 1.513 16.90 1 1 5 2  
mtcars[(mtcars$mpg > 25) & (mtcars$wt < 1.8), ]  
      mpg cyl disp hp drat wt qsec vs am gear carb  
Honda Civic 30.4  4 75.7 52 4.93 1.615 18.52 1 1 4 2  
Lotus Europa 30.4  4 95.1 113 3.77 1.513 16.90 1 1 5 2
```

O altă metodă de indexare este prin folosirea funcției `subset()`.

Tab. 8: Principalele argumente ale functiei `subset()`

Argument	Descriere
x	Un dataframe
subset	Un vector logic care indică liniile pe care le vrem
select	Coloanele pe care vrem să le păstrăm

```
subset(x = mtcars,  
       subset = mpg < 12 &  
                cyl > 6,  
       select = c(disp, wt))  
               disp   wt  
Cadillac Fleetwood 472 5.250  
Lincoln Continental 460 5.424
```

3.4.2 Metode de manipulare

În această secțiune vom prezenta câteva metode mai avansate de manipulare a seturilor de date (a data frame-urilor).

Vom începe prin introducerea comenzi `order()` care permite sortarea liniilor unui data.frame în funcție de valorile coloanei de interes. De exemplu să considerăm cazul setului de date `mtcars`. Vrem să afisăm primele 10 mașini în funcție de greutatea lor (crescător și descrescător):

```
cars_increasing = rownames(mtcars[order(mtcars$wt),])
# afisarea celor mai usoare 10 masini
cars_increasing[1:10]
[1] "Lotus Europa"    "Honda Civic"      "Toyota Corolla"  "Fiat X1-9"
[5] "Porsche 914-2"   "Fiat 128"        "Datsun 710"      "Toyota Corona"
[9] "Mazda RX4"       "Ferrari Dino"

cars_decreasing = rownames(mtcars[order(mtcars$wt, decreasing = TRUE),])
# afisarea celor mai grele 10 masini
cars_decreasing[1:10]
[1] "Lincoln Continental" "Chrysler Imperial"   "Cadillac Fleetwood"
[4] "Merc 450SE"          "Pontiac Firebird"   "Camaro Z28"
[7] "Merc 450SLC"         "Merc 450SL"        "Duster 360"
[10] "Maserati Bora"
```

Functia `order()` permite ordonarea după mai mult de o coloană, de exemplu dacă vrem să ordonăm mașinile după numărul de cilindri și după greutate atunci apelăm

```
mtcars[order(mtcars$cyl, mtcars$wt), 1:6]
      mpg cyl disp hp drat wt
Lotus Europa 30.4 4 95.1 113 3.77 1.513
Honda Civic 30.4 4 75.7 52 4.93 1.615
Toyota Corolla 33.9 4 71.1 65 4.22 1.835
Fiat X1-9 27.3 4 79.0 66 4.08 1.935
Porsche 914-2 26.0 4 120.3 91 4.43 2.140
Fiat 128 32.4 4 78.7 66 4.08 2.200
Datsun 710 22.8 4 108.0 93 3.85 2.320
Toyota Corona 21.5 4 120.1 97 3.70 2.465
Volvo 142E 21.4 4 121.0 109 4.11 2.780
Merc 230 22.8 4 140.8 95 3.92 3.150
Merc 240D 24.4 4 146.7 62 3.69 3.190
Mazda RX4 21.0 6 160.0 110 3.90 2.620
Ferrari Dino 19.7 6 145.0 175 3.62 2.770
Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875
Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215
Merc 280 19.2 6 167.6 123 3.92 3.440
Merc 280C 17.8 6 167.6 123 3.92 3.440
Valiant 18.1 6 225.0 105 2.76 3.460
Ford Pantera L 15.8 8 351.0 264 4.22 3.170
AMC Javelin 15.2 8 304.0 150 3.15 3.435
Hornet Sportabout 18.7 8 360.0 175 3.15 3.440
Dodge Challenger 15.5 8 318.0 150 2.76 3.520
Duster 360 14.3 8 360.0 245 3.21 3.570
Maserati Bora 15.0 8 301.0 335 3.54 3.570
Merc 450SL 17.3 8 275.8 180 3.07 3.730
Merc 450SLC 15.2 8 275.8 180 3.07 3.780
Camaro Z28 13.3 8 350.0 245 3.73 3.840
```

Pontiac Firebird	19.2	8	400.0	175	3.08	3.845
Merc 450SE	16.4	8	275.8	180	3.07	4.070
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345
Lincoln Continental	10.4	8	460.0	215	3.00	5.424

Sunt multe situațiile în care avem la dispoziție două sau mai multe seturi de date și am vrea să construim un nou set de date care să combine informațiile din acestea. Pentru aceasta vom folosi funcția `merge()`. Principalele argumente ale acestei funcții se regăsesc în tabelul de mai jos:

Tab. 9: Argumentele functiei merge

Argument	Descriere
x, y	Două data frame-uri ce urmează a fi unite
by	Un vector de caractere ce reprezintă una sau mai multe coloane după care se va face lipirea. De exemplu <code>by = "id"</code> va combina coloanele care au valori care se potrivesc într-o coloană care se numește <code>"id"</code> . <code>by = c("last.name", "first.name")</code> va combina coloanele care au valori care se potrivesc în ambele coloane <code>"last.name"</code> și <code>"first.name"</code>
all	Un vector logic care indică dacă vrem să includem sau nu liniile care nu se potrivesc conform argumentului <code>by</code> .

Să presupunem că avem la dispoziție un set de date în care apar 5 studenți și notele pe care le-au obținut la examenul de statistică:

```
stat_course = data.frame(student = c("Ionel", "Maria", "Gigel", "Vasile", "Ana"),
                           note_stat = c(9, 8, 5, 7, 9))
```

și să presupunem că avem notele acestor studenți la examenul de algebră

```
alg_course = data.frame(student = c("Maria", "Ana", "Gigel", "Ionel", "Vasile"),
                           note_alg = c(10, 8, 9, 7, 9))
```

Scopul nostru este să creăm un singur tabel în care să regăsim notele la ambele materii:

```
combined_courses = merge(x = stat_course,
                           y = alg_course,
                           by = "student")
combined_courses
  student note_stat note_alg
1     Ana        9        8
2   Gigel        5        9
3   Ionel        9        7
4   Maria        8       10
5   Vasile       7        9
```

O a treia funcție care joacă un rol important în manipularea data frame-urilor este funcția `aggregate()` care, după cum îi spune și numele, permite calcularea de funcții pe grupe de date din setul initial. Argumentele principale ale acestei funcții sunt date în tabelul de mai jos:

Tab. 10: Argumentele functiei aggregate

Argument	Descriere
formula	O formulă de tipul $y \sim x_1 + x_2 + \dots$ unde y este variabila dependentă iar x_1, x_2, \dots sunt variabilele independente. De exemplu, <code>salary ~ sex + age</code> va agrega o coloană <code>salary</code> la fiecare combinație unică de <code>sex</code> și <code>age</code> .
FUN	O funcție pe care vrem să o aplicăm lui y la fiecare nivel al variabilelor independente. E.g. <code>mean</code> sau <code>max</code> .
data	Data frame-ul care conține variabilele din <code>formula</code>
subset	O submulțime din <code>data</code> pe care vrem să le analizăm. De exemplu, <code>subset(sex == "f" & age > 20)</code> va restrânge analiza la femei mai învârstă de 20 de ani.

Structura generală a functiei `aggregate()` este

```
aggregate(formula = dv ~ iv, # dv este data, iv este grupul
          FUN = fun, # Functia pe care vrem sa o aplicam
          data = df) # setul de date care contine coloanele dv si iv
```

Să considerăm setul de date `ChickWeight` și să ne propunem să calculăm pentru fiecare tip de dietă greutatea medie:

```
# Fara functia aggregate
mean(ChickWeight$weight[ChickWeight$Diet == 2])
[1] 122.6167
mean(ChickWeight$weight[ChickWeight$Diet == 3])
[1] 142.95
mean(ChickWeight$weight[ChickWeight$Diet == 4])
[1] 135.2627

# Cu ajutorul functiei aggregate
aggregate(formula = weight ~ Diet, # DV este weight, IV este Diet
          FUN = mean, # calculeaza media pentru fiecare grup
          data = ChickWeight) # dataframe este ChickWeight
Diet      weight
1     1 102.6455
2     2 122.6167
3     3 142.9500
4     4 135.2627
```

Functia `aggregate()` a întors un data.frame cu o coloană pentru variabila independentă `Diet` și o coloană pentru greutatea medie.

Dacă vrem să calculăm greutățile medii în funcție de dietă pentru găinile care au mai puțin de 10 săptămâni de viață atunci folosim opțiunea `subset`:

```
aggregate(formula = weight ~ Diet, # DV este weight, IV este Diet
          FUN = mean, # calculeaza media pentru fiecare grup
          subset = Time < 10, # gainile care au mai putin de 10 saptamani
          data = ChickWeight) # dataframe este ChickWeight
Diet      weight
1     1 58.03093
2     2 63.40000
3     3 65.94000
4     4 69.36000
```

Putem să includem de asemenea mai multe variabile independente în formula funcției `aggregate()`. De

exemplu putem să calculăm greutatea medie a găinilor atât pentru fiecare tip de dietă cât și pentru numărul de săptămâni de la naștere:

```
aggregate(formula = weight ~ Diet + Time, # DV este weight, IV sunt Diet și Time
          FUN = mean,                      # calculeaza media pentru fiecare grup
          data = ChickWeight)            # dataframe este ChickWeight

  Diet Time     weight
1    1    0  41.40000
2    2    0  40.70000
3    3    0  40.80000
4    4    0  41.00000
5    1    2  47.25000
6    2    2  49.40000
7    3    2  50.40000
8    4    2  51.80000
9    1    4  56.47368
10   2    4  59.80000
11   3    4  62.20000
12   4    4  64.50000
13   1    6  66.78947
14   2    6  75.40000
15   3    6  77.90000
16   4    6  83.90000
17   1    8  79.68421
18   2    8  91.70000
19   3    8  98.40000
20   4    8 105.60000
21   1   10  93.05263
22   2   10 108.50000
23   3   10 117.10000
24   4   10 126.00000
25   1   12 108.52632
26   2   12 131.30000
27   3   12 144.40000
28   4   12 151.40000
29   1   14 123.38889
30   2   14 141.90000
31   3   14 164.50000
32   4   14 161.80000
33   1   16 144.64706
34   2   16 164.70000
35   3   16 197.40000
36   4   16 182.00000
37   1   18 158.94118
38   2   18 187.70000
39   3   18 233.10000
40   4   18 202.90000
41   1   20 170.41176
42   2   20 205.60000
43   3   20 258.90000
44   4   20 233.88889
45   1   21 177.75000
46   2   21 214.70000
47   3   21 270.30000
```

48 4 21 238.55556



Considerați setul de date `mtcars`. Calculați:

- a) Greutatea medie în funcție de tipul de transmisie
- b) Greutatea medie în funcție de numărul de cilindrii
- c) Consumul mediu în funcție de numărul de cilindrii și tipul de transmisie

Referințe

Tilman Davies. *The Book of R. A First Course in Programming and Statistics*. No Starch Press, Inc., 1st edition, 2016. ISBN 978-1-59327-651-5. (Citat la pagina 1.)

Norman Matloff. *The Art of R Programming. A Tour of Statistical Software Design*. No Starch Press, Inc., 1st edition, 2011. ISBN 978-1-59327-384-2. (Citat la pagina 1.)

Laborator 2

Elemente de programare și grafică în R

Obiectivul acestui laborator este de a prezenta succint elementele de programare din programul R, care este structura lor și cum le putem aplica. De asemenea, tot în acest laborator vom introduce și câteva elemente de grafică. Pentru mai multe detalii se pot consulta [Davies, 2016, Matloff [2011]].

1 Elemente de programare în R

1.1 Funcții

O *funcție* este un obiect în R care primește câteva obiecte de intrare (care se numesc *argumentele funcției*) și întoarce un obiect de ieșire. Structura unei funcții va avea următoarele patru părți:

- *Nume*: Care este numele funcției? Aveți grija să nu folosiți nume ale funcțiilor deja existente în R!
- *Argumente*: Care sunt datele de intrare pentru funcție? Puteti specifica oricâte date de intrare doriti!
- *Corp sau acțiune*: Ce vreți să facă această funcție? Să traseze un grafic? Să calculeze o statistică?
- *Rezultat*: Ce vreți să vă întoarcă funcția? Un scalar? Un vector? Un data.frame?

```
# Structura de baza a unei functii
NUME <- function(ARGUMENTE) {
    ACTIUNI
    return(RESULTAT)
}
```

Funcțiile în R sunt *obiecte de primă clasă* (first class objects), ceea ce înseamnă că ele pot fi tratate ca orice alt obiect din R. Este important de reținut că, în R,

- funcțiile pot fi date ca argumente pentru alte funcții (de exemplu familia de funcții `apply()`)
- funcțiile pot fi imbricate (nested), cu alte cuvinte puteți crea funcții în interiorul altor funcții

Mai jos avem un exemplu de funcție care nu are niciun argument și nu întoarce nicio valoare:

```
f <- function() {
    ## Aceasta este o functie goala
}
## Functiile au clasa lor speciala
class(f)
[1] "function"

f()
NULL
```

Următoarea funcție întoarce numărul de caractere al textului dat ca argument:

```
f <- function(mesaj){  
  chars = nchar(mesaj)  
  chars  
}  
  
mes = f("curs de statistica si probabilitati")  
mes  
[1] 35
```

În funcția de mai sus nu am indicat nimic special pentru ca funcția să ne întoarcă numărul de caractere. În R, rezultatul unei funcții este întotdeauna ultima expresie evaluată. De asemenea există funcția `return()` care poate fi folosită pentru a întoarce o valoare explicită, dar de multe ori această funcție este omisă.

Dacă utilizatorul nu specifică valoarea argumentului `mesaj` în funcția de mai sus atunci R-ul întoarce o eroare:

```
f()  
Error in nchar(mesaj): argument "mesaj" is missing, with no default
```

Acest comportament al funcției poate fi modificat prin definirea unei valori implicate (de default). Orice argument al funcției poate avea o valoare de default.

```
f <- function(mesaj = "Valoare de default"){  
  chars = nchar(mesaj)  
  chars  
}  
  
# Folosim valoarea implicită  
f()  
[1] 18  
# Folosim o valoare specificată  
f("curs de statistica si probabilitati")  
[1] 35
```



Să presupunem că Jack Sparrow este convins că poate prezice cât aur va găsi pe o insulă folosind următoarea ecuație: $ab - 324c + \log(a)$, unde a este aria insulei (în m^2), b este numărul de copaci de pe insulă iar c reprezintă cât de beat este pe o scală de la 1 la 10. Creați o funcție numită `Jacks.Money` care primește ca argumente a , b și c și întoarce valoare prezisă.

Un exemplu ar fi

```
Jacks.Money(a = 1000, b = 30, c = 7)  
[1] 27738.91
```

Argumentele funcțiilor în R pot fi potrivite după poziția lor sau după numele lor. Potrivirea după poziție înseamnă că R atribuie prima valoare primului argument, a doua valoare celui de-al doilea argument, etc. De exemplu atunci când folosim funcție `rnorm()`,

```
str(rnorm)  
function (n, mean = 0, sd = 1)  
set.seed(1234) # pentru repetabilitate  
mydata <- rnorm(10, 3, 1)  
mydata  
[1] 1.7929343 3.2774292 4.0844412 0.6543023 3.4291247 3.5060559 2.4252600  
[8] 2.4533681 2.4355480 2.1099622
```

valoarea 10 este atribuită argumentului `n`, valoarea 3 argumentului `mean` iar valoarea 1 argumentului `sd`,

toate prin potrivire după poziție.

Atunci când specificăm argumentele funcției după nume, ordinea acestora nu contează. De exemplu

```
set.seed(1234)
rnorm(mean = 3, n = 10, sd = 1)
[1] 1.7929343 3.2774292 4.0844412 0.6543023 3.4291247 3.5060559 2.4252600
[8] 2.4533681 2.4355480 2.1099622
```

întoarce același rezultat cu cel obținut mai sus.

De cele mai multe ori, argumentele cu nume sunt folositoare atunci când funcția are un sir lung de argumente și ne dorim să folosim valorile implicate pentru majoritatea dintre ele. De asemenea aceste argumente pot fi folositoare și atunci când stăm numele argumentului dar nu și poziția în lista de argumente. Un exemplu de astfel de funcție este funcția `plot()`, care are multe argumente folosite în special pentru customizare:

```
args(plot.default)
function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
         log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
         ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first = NULL,
         panel.last = NULL, asp = NA, xgap.axis = NA, ygap.axis = NA,
         ...)
NULL
```

În R există un argument special notat ..., care indică un număr arbitrar de argumente care sunt atribuite altor funcții din corpul funcției. Acest argument este folosit în special atunci când vrem să extindem o altă funcție și nu vrem să copiem întreaga listă de argumente a acesteia. De exemplu, putem crea o funcție de plotare în care specificăm tipul în prealabil

```
myplot <- function(x, y, type = "l", ...) {
  plot(x, y, type = type, ...)          ## Atribuie '...' funcției 'plot'
}
```

Argumentul ... poate fi folosit (și este necesar) și atunci când numărul de argumente pe care îl ia funcția nu este cunoscut în prealabil. De exemplu să considerăm funcțiile `paste()` și `cat()`

```
args(paste)
function (... , sep = " ", collapse = NULL, recycle0 = FALSE)
NULL
args(cat)
function (... , file = "", sep = " ", fill = FALSE, labels = NULL,
          append = FALSE)
NULL
```

Deoarece ambele funcții printează text în consolă combinând mai mulți vectori de caractere împreună, este imposibil ca acestea să cunoască în prealabil câți vectori de caractere vor fi dați ca date de intrare de către utilizator, deci primul argument pentru fiecare funcție este

Este important de menționat că toate argumentele care apar după argumentul ... trebuie explicitate după nume.

```
paste("Curs", "Probabilitati si Statistica", sep = ":")
[1] "Curs:Probabilitati si Statistica"
```

1.2 Structuri de control (`if-else`, `for`, etc.)

Structurile de control, în R, permit structurarea logică și controlul fluxului de execuție al unei serii de comenzi. Cele mai folosite structuri de control sunt:

- `if` și `else`: testează o condiție și acționează asupra ei

- **switch**: compara mai multe opțiuni și execută optiunea pentru care avem potrivire
- **for**: execută o acțiune repetitivă de un număr fix de ori
- **while**: execută o acțiune repetitivă *cât timp* o condiție este adevărată
- **repeat**: execută o acțiune repetitivă de o infinitate de ori (trebuie să folosim **break** pentru a ieși din ea)
- **break**: îintrerupe execuția unei acțiuni repetitive
- **next**: sare peste un pas în executarea unei acțiuni repetitive

1.2.1 if-else

Structura **if-else** este una dintre cele mai folosite structuri de control în R permitând testarea unei condiții și acționând în funcție de valoarea de adevăr a acesteia.

Forma de bază a acestei structuri este

```
if(<conditie>) {  
    ## execută instructiuni atunci cand conditia este adevarata  
}  
else {  
    ## execută instructiuni atunci cand conditia este falsa  
}
```

dar putem să avem și o serie de teste, de tipul

```
if(<conditie1>) {  
    ## execută instructiuni  
} else if(<conditie2>) {  
    ## execută instructiuni  
} else {  
    ## execută instructiuni  
}
```

Avem următorul exemplu

```
## Generăm un numar uniform in [0,10]  
x <- runif(1, 0, 10)  
  
if(x > 3) {  
    y <- 10  
} else {  
    y <- 0  
}
```

1.2.2 Comanda switch

Comanda **switch** este folosită cu precădere atunci când avem mai multe alternative dintre care vrem să alegem. Structura generală a acestei comenzi este

```
switch (Expresie, "Optiune 1", "Optiune 2", "Optiune 3", ..., "Optiune N")
```

sau într-o manieră extinsă

```
switch (Expresie,  
        "Optiune 1" = Executa aceste expresii atunci cand expresia se  
                      potriveste cu Optiunea 1,  
        "Optiune 2" = Executa aceste expresii atunci cand expresia se
```

```
        potriveste cu Optiunea 2,  
    "Optiune 3" = Atunci cand expresia se potriveste cu Optiunea 3,  
                executa aceste comenzi,  
    ...  
    "Optiune N" = Atunci cand expresia se potriveste cu Optiunea N,  
                  executa aceste comenzi  
)
```

Considerăm următorul exemplu

```
number1 <- 30  
number2 <- 20  
# operator <- readline(prompt="Insereaza OPERATORUL ARITMETIC: ")  
  
operator = "*"  
  
switch(operator,  
    "+" = print(paste("Suma celor doua numere este: ",  
                      number1 + number2)),  
    "-" = print(paste("Diferenta celor doua numere este: ",  
                      number1 - number2)),  
    "*" = print(paste("Inmultirea celor doua numere este: ",  
                      number1 * number2)),  
    "^" = print(paste("Ridicarea la putere a celor doua numere este: ",  
                      number1 ^ number2)),  
    "/" = print(paste("Impartirea celor doua numere este: ",  
                      number1 / number2)),  
    "%/" = print(paste("Catul impartirii celor doua numere este: ",  
                      number1 %/ number2)),  
    "%%" = print(paste("Restul impartirii celor doua numere este: ",  
                      number1 %% number2))  
)  
[1] "Inmultirea celor doua numere este: 600"
```

1.2.3 Bucle for

În R, buclele **for** iau o variabilă care se iterează și îi atribuie valori succesive dintr-un sir sau un vector. Buclele **for** au următoarea structură de bază

```
for (<i> in <vector>) {  
    ## executa instructiuni  
}
```

de exemplu

```
for(i in 1:10) {  
    print(i)  
}  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
[1] 6  
[1] 7  
[1] 8
```

```
[1] 9  
[1] 10
```

Următoarele trei bucle prezintă același comportament

```
x <- c("a", "b", "c", "d")  
  
for(i in 1:4) {  
    ## Afiseaza fiecare element din 'x'  
    print(x[i])  
}  
[1] "a"  
[1] "b"  
[1] "c"  
[1] "d"
```

Functia `seq_along()` este des întâlnită atunci când folosim bucle `for` deoarece crează un sir întreg folosind lungimea obiectului (în acest caz al lui `x`)

```
## Genereaza un sir folosind lungimea lui 'x'  
for(i in seq_along(x)) {  
    print(x[i])  
}  
[1] "a"  
[1] "b"  
[1] "c"  
[1] "d"
```

De asemenea putem folosi chiar pe `x` ca vector de indexare

```
for(letter in x) {  
    print(letter)  
}  
[1] "a"  
[1] "b"  
[1] "c"  
[1] "d"
```

Atunci când folosim comenziile din buclele `for` pe o singură linie nu este necesară folosirea parantezelor `{}`

```
for(i in 1:4) print(x[i])  
[1] "a"  
[1] "b"  
[1] "c"  
[1] "d"
```

Putem folosi buclele `for` și imbricat (nested)

```
x <- matrix(1:6, 2, 3)  
  
for(i in seq_len(nrow(x))) {  
    for(j in seq_len(ncol(x))) {  
        print(x[i, j])  
    }  
}
```



Construiți următoarele matrice de dimensiune 10×10 : $M_{i,j} = \frac{1}{\sqrt{|i-j|+1}}$ și $N_{i,j} = \frac{i}{j^2}$. Puteți construi matricea M și matricea N fără a folosi bucle `for`? (Hint: ce face comanda `outer`?)

1.2.4 Bucle de tip `while`

Acțiunile repetitive de tip `while` încep prin testarea unei condiții și în cazul în care aceasta este adevărată atunci se execută corpul comenzi. Odată ce corpul buclei este executat, condiția este testată din nou până când devine falsă (se poate ca bucla `while` să rezulte într-o repetiție infinită!). Structura generală a acestei bucle este

```
while(<conditie>) {  
    ## executa instructiuni  
}
```

Considerăm următorul exemplu

```
count <- 0  
while(count < 4) {  
    print(count)  
    count <- count + 1  
}  
[1] 0  
[1] 1  
[1] 2  
[1] 3
```

Uneori putem testa mai multe condiții (acestea sunt întotdeauna evaluate de la stânga la dreapta)

```
z <- 5  
set.seed(123)  
  
while(z >= 3 && z <= 10) {  
    coin <- rbinom(1, 1, 0.5) # arunc cu banul  
  
    if(coin == 1) { ## random walk  
        z <- z + 1  
    } else {  
        z <- z - 1  
    }  
}  
print(z)  
[1] 11
```



Scrieți un program, folosind bucle de tip `while`, care să permită calcularea radicalului numărului $a \in \mathbb{N}$ plecând de la relația de recurență:

$$2x_{n+1} = x_n + \frac{a}{x_n}, \quad x_1 = \frac{a}{2}$$

De exemplu:

```
# Calculul radicalului  
a = 12223
```

```
# folosind while
x = a/2
while(abs(x^2 - a) > 1e-10){
  x = (x + a/x)/2
}
```

1.2.5 Bucle de tip repeat

Acest tip de acțiuni repetitive nu sunt foarte des întâlnite, cel puțin în statistică sau analiză de date. O situație în care ar putea să apară este atunci când avem un algoritm iterativ în care căutăm o soluție și nu vrem să opriam algoritmul până când soluția nu este suficient de bună.

```
x0 <- 1
tol <- 1e-8

repeat {
  x1 <- o_functie_definita()

  if(abs(x1 - x0) < tol) { ## este suficient de buna solutia
    break
  } else {
    x0 <- x1
  }
}
```

1.2.6 Comezile break și next

Comanda `next` este folosită pentru a sării un pas într-o buclă

```
for(i in 1:10) {
  if(i <= 5) {
    ## sări peste primele 5 iteratii
    next
  }
  print(i)

}
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Comanda `break` este folosită pentru a părăsi o buclă imediat

```
for(i in 1:10) {
  print(i)

  if(i > 5) {
    ## opreste dupa 5 iteratii
    break
  }
}
[1] 1
[1] 2
```

```
[1] 3  
[1] 4  
[1] 5  
[1] 6
```

Calculul radicalului unui număr cu ajutorul buclei de tip `repeat` și a comenzi `break` se scrie

```
# folosind repeat  
a = 12223  
  
x = a/2  
repeat{  
  x = (x + a/x)/2  
  if (abs(x^2 - a) < 1e-10) break  
}
```

2 Elemente de grafică în R

Graficele reprezintă, în general, un instrument folositor pentru vizualizarea rezultatelor unei analize dar și un mod prin care se poate verifica dacă am obținut ce ni s-a cerut. Este important să folosim opțiunile corecte atunci când trasăm un grafic, de multe ori valorile predefinite putând conduce la figuri eronate. În tabelul de mai jos avem listate principalele instrucțiuni grafice

Tab. 1: Functii grafice uzuale in R

Funcția	Scurtă descriere
<code>plot</code>	O funcție generică folosită pentru plotarea unui obiect (puncte, curbe, etc.)
<code>barplot</code>	Trasează un grafic de bare orizontale sau verticale (folosit pentru variabile discrete)
<code>hist</code>	Desenează histogramme cu frecvențe sau probabilități
<code>boxplot</code>	Desenează una sau mai multe boxplot-uri în paralel
<code>pie</code>	Desenează o diagramă de tip plăcintă

Metodele grafice din R nu se limitează la cele din tabelul de mai sus. De exemplu, pachetul `ggplot2`¹ este foarte răspândit la ora actuală deoarece pune la dispoziție o serie de instrumente grafice (folosind o gramatică de grafice) cu ajutorul cărora se pot produce figuri de calitate deosebită (ce se folosesc în special pentru publicații științifice).

În continuare vom prezenta o parte din funcțiile cel mai des folosite pentru trasarea graficelor în R împreună cu proprietățile de bază ale acestora.

2.1 Funcția `plot`

Cea mai folosită funcție (high-level) de plotare în R este funcția `plot()`. Aceasta permite trasarea unei diagrame de împrăștiere (*scatterplot*) între doi vectori `x` și `y`, unde vectorul `x` indică valorile pe axa abscisei iar `y` pe axa ordonatelor.

Argumentele principale ale funcției `plot()` se găsesc în tabelul următor.

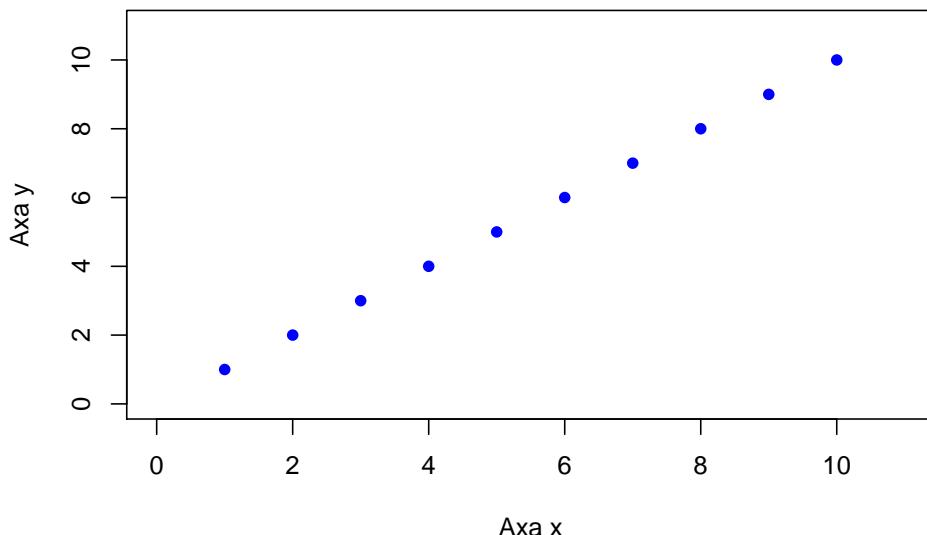
¹Vizitați pagina <http://ggplot2.tidyverse.org/reference/> pentru mai multe detalii

Tab. 2: Argumentele principale ale functiei plot()

Argument	Descriere
x, y	Vectori de aceeași lungime care specifică valorile coordonatelor de pe x și de pe y
type	Tipul de grafic: "l" reprezintă linii, "p" reprezintă puncte, "b" reprezintă și linii și puncte, "n" înseamnă că nu trasează nimic
main, xlab, ylab	String-uri folosite pentru titlul graficului și etichetarea axelor x și y
xlim, ylim	Limitele pe axa x și y. De exemplu, xlim = c(0, 100) va seta valoarea minimă și maximă de pe axa x la 0 și respectiv 100.
pch	Un număr întreg care denotă tipul de simbol folosit pentru trasarea punctelor (vezi ?points), sau un sir de caractere care specifică simbolurile ca text. De exemplu, pch = 21 va crea un cerc colorat cu 2 culori, iar pch = "P" va trasa caracterul "P" pentru fiecare punct.
col	Culoarea principală a simbolurilor desenate. De exemplu col = "blue" va trasa simbolurile în culoarea albastră.
lty	Tipul de linie folosit. De exemplu lty = 1 reprezintă o linie solidă pe când lty = 2 reprezintă o linie întreruptă.
lwd	Grosimea liniei folosite, valoare prestatibila este lwd = 1.
cex	Un vector numeric folosit pentru a specifica mărimea simbolurilor trasate. Valoarea prestatibila este 1. De exemplu cex = 4 va face punctele foarte mari pe când cex = .5 le va face mai mici.

```
plot(x = 1:10,                                     # x-coordonate
      y = 1:10,                                     # y-coordonate
      type = "p",                                    # puncte (nu linii)
      main = "Primul grafic",                      # Titlu
      xlab = "Axa x",                                # Etichetă x
      ylab = "Axa y",                                # Etichetă y
      xlim = c(0, 11),                               # Valorile min si max pe axa x
      ylim = c(0, 11),                               # Valorile min si max pe axa y
      col = "blue",                                 # Culoarea punctelor
      pch = 16,                                    # Tipul simbolului
      cex = 1)                                     # Marimea simbolului
```

Primul grafic



În afară de vectorii x și y toate celelalte argumente sunt opționale, dacă nu sunt specificate atunci R-ul folosește valorile prestablebite. De exemplu dacă nu specificăm limitele $xlim$ și $ylim$, R-ul calculează aceste limite astfel încât toate punctele să fie încadrați în interiorul graficului.



Trasați următoarele diagrame de împrăștiere:

```
x <- seq(0, 1, 0.1); plot(x, x - x * x + 2)    plot(x, x - x * x + 2, type = "l");  
plot(x, x - x * x + 2, type = "b", pch = 19)
```

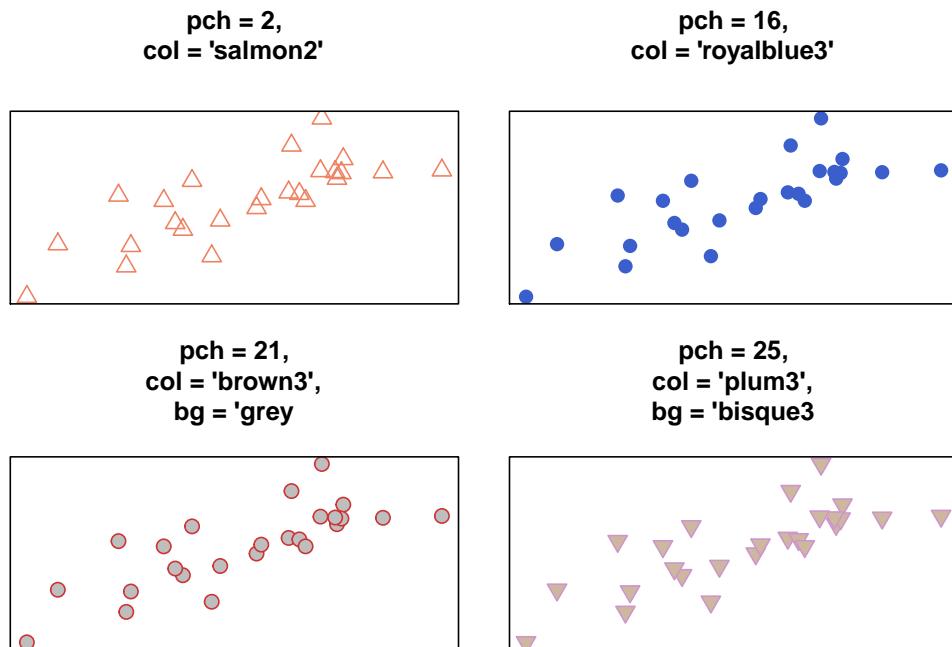
Tipul de simbol pe care vrem să-l folosim atunci trasăm un grafic este specificat prin argumentul `pch`. Figura de mai jos ne prezintă tipurile simboluri pe care atunci când atribuim argumentului `pch` o valoare întreagă.

Următorul exemplu ilustrează câteva tipuri de simboluri:

pch = ...

1 ○ 6 ▽ 11 ♠ 16 ● 21 ○
 2 △ 7 ☐ 12 ☉ 17 ▲ 22 □
 3 + 8 * 13 ☗ 18 ◆ 23 ◇
 4 × 9 ☔ 14 ☑ 19 ● 24 △
 5 ◇ 10 ⊕ 15 ■ 20 • 25 ▽

Fig. 1: Tipurile de simboluri asociate parametrului pch.



Considerăm următoarea funcție $g : \mathbb{R} \rightarrow \mathbb{R}$,

$$g(x) = \begin{cases} \sin^2(x) \log(x), & x > 0 \\ \sin^2(x)x, & x \leq 0 \end{cases}$$

- Definiți funcția folosind comenzi `if-else` și `Vectorize` iar apoi folosind comanda `ifelse`.
- Trasați graficul curbei pe intervalul $[-\pi, \pi]$.

2.2 Culori

Majoritatea funcțiilor de desenare au un argument în care pot specifica culoarea elementelor pe care vrem să le trasăm, de cele mai multe ori acesta este `col`. Cel mai ușor mod de a introduce o culoare este prin specificarea numelui acesteia, de exemplu `col = 'red'` este culoarea roșie. Figura 1 prezintă 144 de culori alese la întâmplare din totalul de 657 câte există în R.

grey77	grey89	pink3	darkslateblue	palegreen4	violetred2	cadetblue1	gray36	gray18	burlywood2	gray63	grey30
darkkhaki	gray97	turquoise	grey20	lightskyblue1	blue4	aquamarine2	lightsteelblue3	grey84	aquamarine4	grey32	grey42
gray35	lightskyblue2	navajowhite3	pink2	royalblue3	gray62	pink4	sienna4	grey35	grey67	goldenrod	darkmagenta
yellow1	magenta	honeydew1	mintcream	lightblue4	indianred3	grey74	gold	lightyellow1	rosybrown	slategray2	maroon2
gray10	mediumspringgreen	gray75	seashell4	grey4	lightsalmon3	sienna3	gray96	snow4	burlywood3	plum	olivedrab4
deeppink	springgreen4	darkseagreen	gray61	lavenderblush2	gray7	darkgoldenrod2	papayawhip	dimgray	lightsteelblue4	gray28	
grey15	gray16	cyan3	sandybrown	wheat3	navajowhite	mediumaquamarine	coral3	linen	purple1	bisque	grey58
peachpuff4	grey36	gray55	tomato	salmon1	paleturquoise3	gray95	honeydew3	slateblue1	lightskyblue4	gray65	orchid
grey97	grey46	tan2	gray68	rosybrown4	grey52	firebrick3	gold3	deepskyblue2	gray81	snow3	olivedrab2
lightsalmon	ivory2	darkslategray	dodgerblue3	grey82	burlywood4	thistle2	grey37	green4	wheat2	gray29	grey44
gray31	seagreen	antiquewhite1	red	gray59	gray42	orchid3	mistyrose	snow	tomato4	seagreen4	orchid2
grey23	darksalmon	tan3	violetred4	lightblue1	darkorchid3	darkseagreen1	slategray3	grey65	darkgoldenrod4	grey9	khaki

Fig. 2: 144 de culori din totalul de 657 din R

Pentru a vedea toate culorile din R putem rula comanda `colors()`.

2.3 Funcția `hist`

Histograma² reprezintă metoda grafică, cea mai comună, de reprezentare a repartiției unui vector numeric. Pentru a crea o histogramă în R folosim funcția `hist()` în care argumentul principal este un vector numeric. Tabelul de mai jos prezintă argumentele principale ale funcției `hist`.

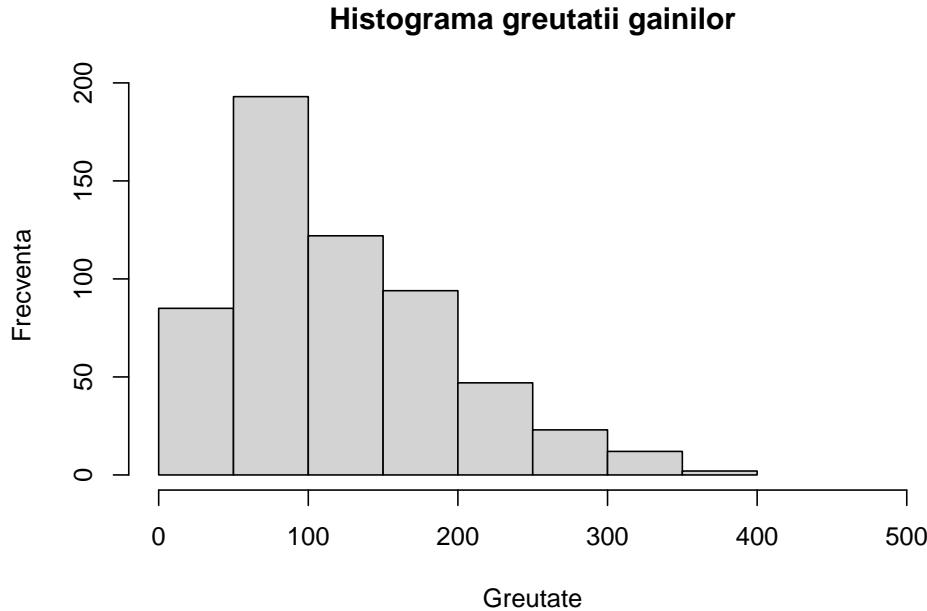
²Histograma este un estimator neparametric al densității.

Tab. 3: Argumentele funcției `hist()`

Argument	Descriere
<code>x</code>	Vector de valori
<code>breaks</code>	Cum să calculăm mărimea bin-urilor (vezi <code>?hist</code>)
<code>freq</code>	Opțiune pentru trasarea histogramei de frecvență și de probabilități, <code>freq = TRUE</code> arată frecvențele, <code>freq = FALSE</code> arată probabilitățile.
<code>col, border</code>	Culoarea interioară a bin-urilor (<code>col</code>) și culoarea conturului lor (<code>border</code>)

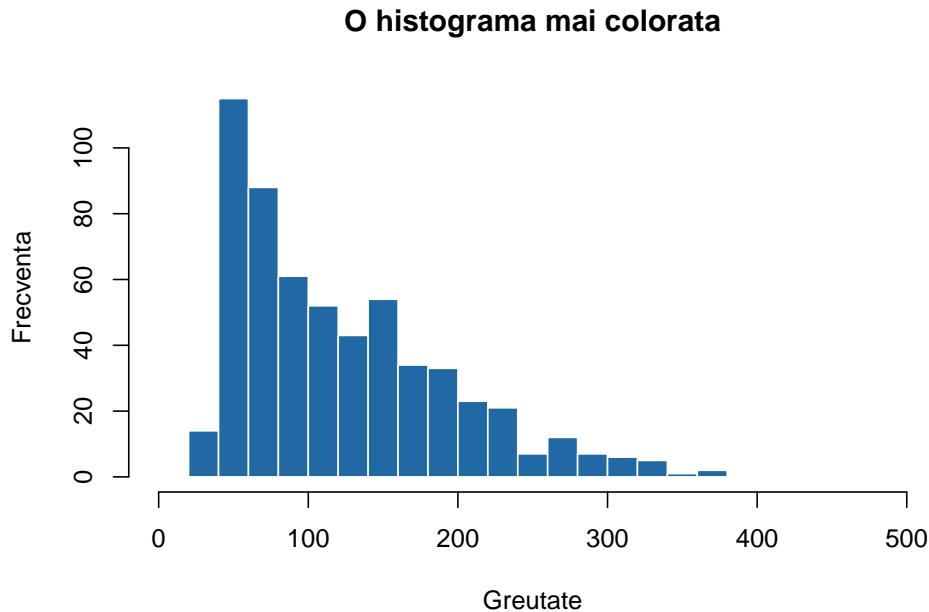
Putem crea o histogramă folosind setul de date `ChickWeight` (`?ChickWeight`)

```
hist(x = ChickWeight$weight,
      main = "Histograma greutatii gainilor",
      xlab = "Greutate",
      ylab = "Frecventa",
      xlim = c(0, 500))
```



Putem modifica histograma de mai sus, schimbând numărul de bin-uri și culoarea acestora:

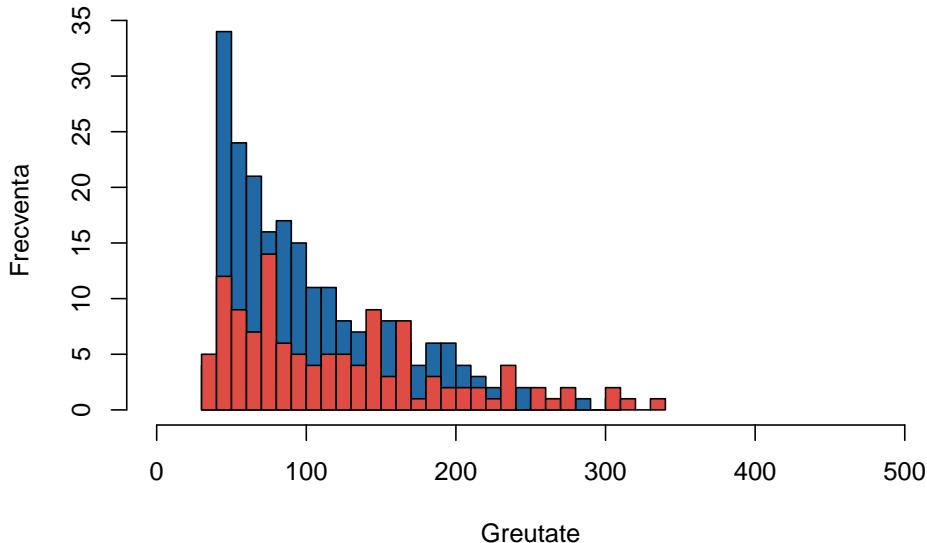
```
hist(x = ChickWeight$weight,
      main = "O histograma mai colorata",
      xlab = "Greutate",
      ylab = "Frecventa",
      breaks = 20, # 20 Bins
      xlim = c(0, 500),
      col = myblue, # Culoarea de umplere
      border = "white") # Culoarea conturului
```



Dacă vrem să ilustrăm două histograme pe aceeași figură, pentru a evidenția repartiția după două clase, putem folosi argumentul `add = TRUE` la cel de-al doilea plot:

```
hist(x = ChickWeight$weight[ChickWeight$Diet == 1],  
      main = "Doua histograme pe acelasi grafic",  
      xlab = "Greutate",  
      ylab = "Frecventa",  
      breaks = 20,  
      xlim = c(0, 500),  
      col = myblue)  
  
hist(x = ChickWeight$weight[ChickWeight$Diet == 2],  
      breaks = 30,  
      add = TRUE, # Adauga graficul la cel de dinainte  
      col = myred)
```

Două histogramme pe același grafic



2.4 Funcția barplot

Funcția `barplot` este folosită în special atunci când avem de-a face cu o variabilă discretă. Argumentul principal al funcției este `height`, un vector numeric care va genera înălțimea fiecărei bare. Pentru a adăuga nume sub fiecare bară putem folosi argumentul `names.arg`.

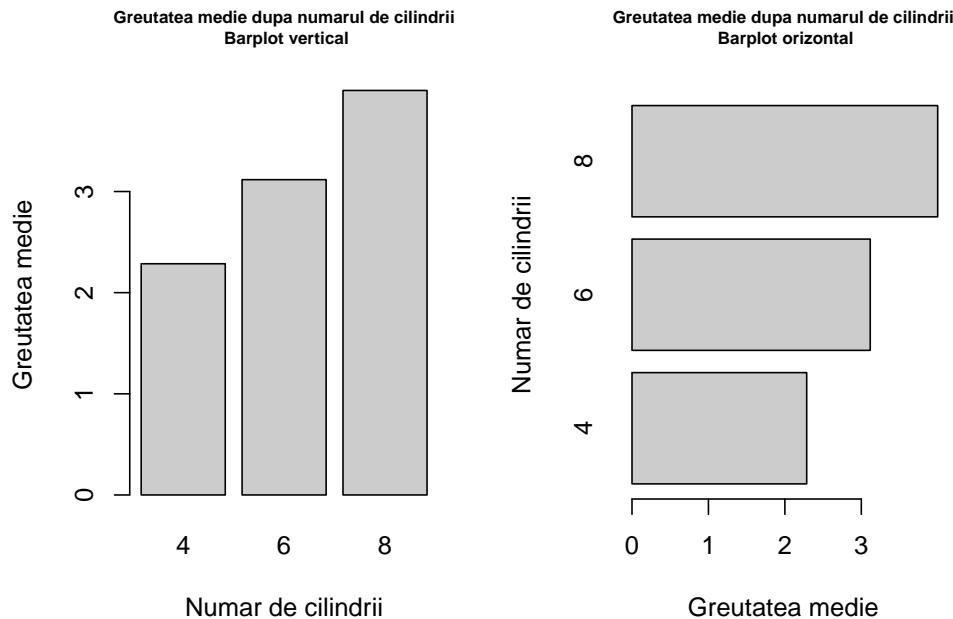
De exemplu, folosind setul de date `mtcars` putem să afișăm greutatea medie a mașinilor în funcție de numărul de cilindri:

```
par(mfrow = c(1, 2))

weight_cars = aggregate(wt ~ cyl,
                       data = mtcars,
                       FUN = mean)

barplot(height = weight_cars$wt,
        names.arg = weight_cars$cyl,
        xlab = "Numar de cilindri",
        ylab = "Greutatea medie",
        main = "Greutatea medie dupa numarul de cilindrii\n Barplot vertical",
        col = "grey80",
        cex.main = 0.7)

barplot(height = weight_cars$wt,
        names.arg = weight_cars$cyl,
        horiz = TRUE,
        ylab = "Numar de cilindri",
        xlab = "Greutatea medie",
        main = "Greutatea medie dupa numarul de cilindrii\n Barplot orizontal",
        col = "grey80",
        cex.main = 0.7)
```



Folosind setul de date `ChickWeight` afișați, cu ajutorul funcției `barplot`, greutatea medie a găinilor în raport cu numărul de zile de la naștere.

De asemenea putem crea un barplot clusterizat în funcție de mai multe grupuri de date. De exemplu să presupunem că vrem să vedem dacă există diferențe între greutatea medie a mașinilor (din setul de date `mtcars`) care au transmisie manuală sau automată și numărul de cilindrii.

```
# calculam greutatea medie după numarul de cilindrii și transmisie
carWeight_cyl_am = aggregate(mtcars$wt, by = list(mtcars$cyl, mtcars$am), FUN = mean)

# transformăm rezultatul sub forma de matrice
carWeight_cyl_am = as.matrix(carWeight_cyl_am)
carWeight_cyl_am

  Group.1 Group.2      x
[1,]      4      0 2.935000
[2,]      6      0 3.388750
[3,]      8      0 4.104083
[4,]      4      1 2.042250
[5,]      6      1 2.755000
[6,]      8      1 3.370000

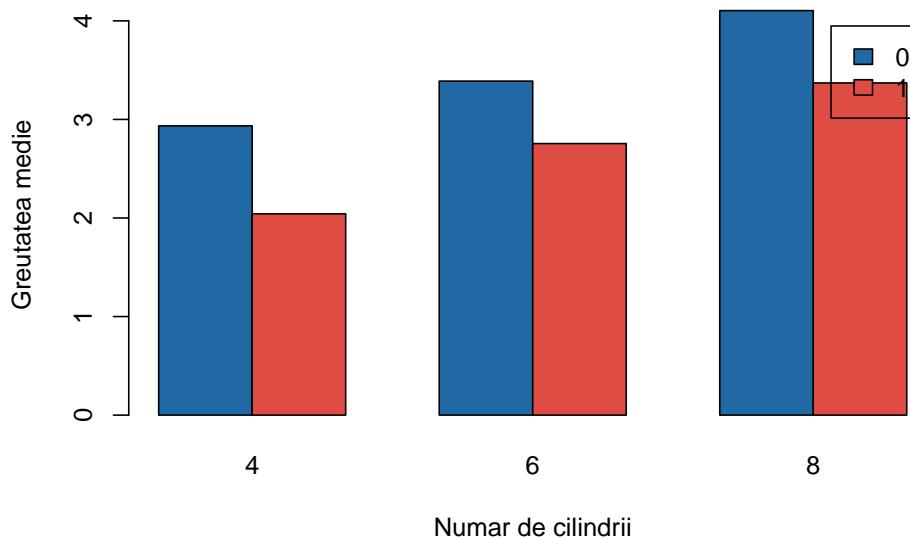
# aducem la forma necesara pentru barplot
carWeight = matrix(carWeight_cyl_am[,3], nrow = 3)
colnames(carWeight) = unique(carWeight_cyl_am[,2])
rownames(carWeight) = unique(carWeight_cyl_am[, 1])

carWeight = t(carWeight)

barplot(carWeight,
        beside = TRUE,
        legend.text = TRUE,
```

```
col = c(myblue, myred),
main = "Greutatea medie a masinilor dupa numarul de cilindri si transmisie",
xlab = "Numar de cilindri",
ylab = "Greutatea medie")
```

Greutatea medie a masinilor dupa numarul de cilindri si transmisie



2.5 Funcția boxplot

Pentru a vedea cât de bine sunt repartizate datele în setul de date putem folosi funcția `boxplot` (box and whisker plot - cutie cu mustăți). Această funcție prezintă într-o manieră compactă modul în care este repartizată o variabilă. Această metodă grafică prezintă principali indicatori de poziție ai variabilei studiate: quartilele de ordin 1 și 3 (Q_1, Q_3) care delimită cutia ($IQR = Q_3 - Q_1$) și quartila de ordin 2 sau mediana (linia din interiorul cutiei). Mustățile sunt calculate în modul următor: mustața superioară este determinată de valoarea celei mai mari observații care este mai mică sau egală cu $Q_3 + 1.5IQR$ iar mustața inferioară este valoarea celei mai mici observații mai mari sau egale cu $Q_1 - 1.5IQR$. Valorile din afara cutiei cu mustăți se numesc valori aberante.

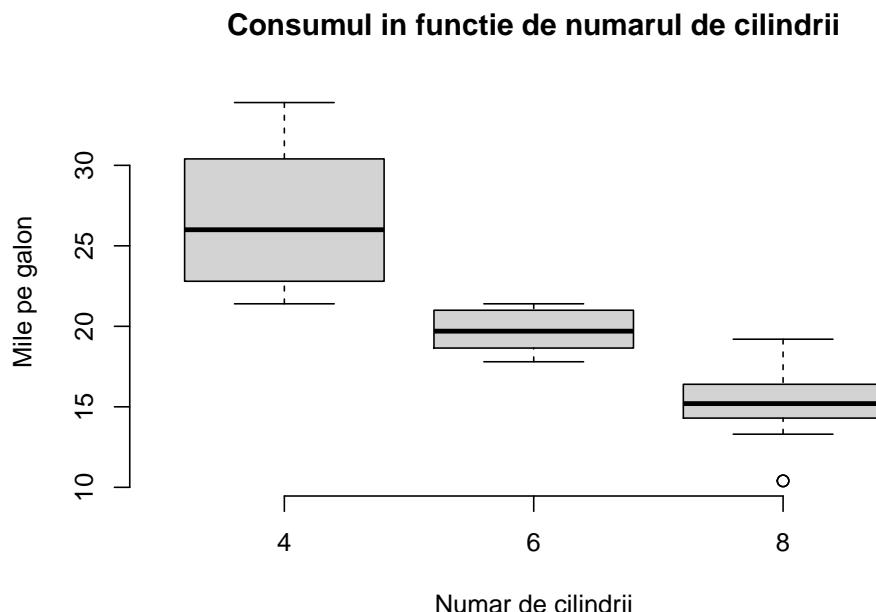
Principalele argumente ale funcției `boxplot` se regăsesc în tabelul următor, pentru mai multe detalii apelați `?boxplot`.

Tab. 4: Principalele argumente ale functiei boxplot.

Argument	Descriere
<code>formula</code>	O formulă de tip <code>y ~ grp</code> , unde <code>y</code> este variabila investigată iar <code>grp</code> este variabila care descrie grupurile după care vrem să trasăm graful
<code>data</code>	Un data frame (sau listă) în care variabilele din formulă sunt definite
<code>subset</code>	Un vector care specifică o submulțime a observațiilor
<code>x</code>	Un vector care specifică valorile ce urmează să fie trasate
<code>horizontal</code>	O valoare logică care indică dacă trasăm boxplot-urile vertical (FALSE) sau orizontal (TRUE)
<code>add</code>	O valoare logică prin care se permite adăugarea graficului la unul deja existent

Următorul exemplu ne prezintă relația dintre consum (mpg) și numărul de cilindrii (cyl) în cazul mașinilor din setul de date mtcars.

```
par(bty = "n")
boxplot(mpg ~ cyl,
        data = mtcars,
        xlab = "Numar de cilindrii",
        ylab = "Mile pe galon",
        main = "Consumul in functie de numarul de cilindrii")
```

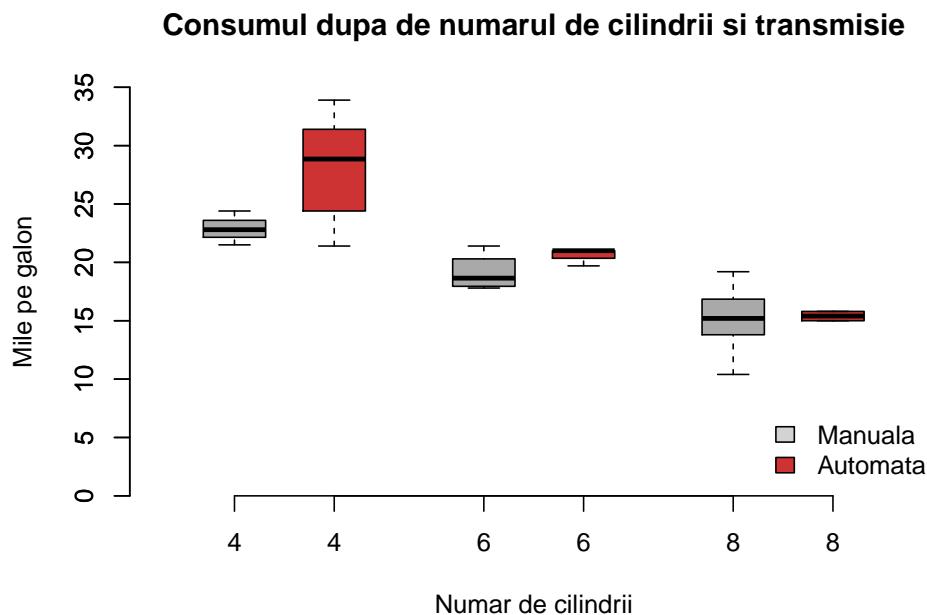


Putem să vedem această relație și în raport cu tipul de transmisie.

```
par(bty = "n")
boxplot(mpg ~ cyl,
        data = mtcars,
        subset = am == 0,
        boxwex = 0.25,
        at = 1:3 - 0.2,
        col = "darkgrey",
        xlab = "Numar de cilindrii",
        ylab = "Mile pe galon",
        main = "Consumul dupa de numarul de cilindrii si transmisie",
        xlim = c(0.5, 3.5),
        ylim = c(0, 35),
        yaxs = "i")

boxplot(mpg ~ cyl,
        data = mtcars,
        subset = am == 1,
        add = TRUE,
        boxwex = 0.25,
        at = 1:3 + 0.2,
        col = "brown3")
```

```
legend("bottomright" ,c("Manuala", "Automata"),
      fill = c("lightgray", "brown3"), bty = "n")
```



2.6 Funcții pentru adăugarea unor elemente la un grafic

Functiile (low-level) din această secțiune sunt folosite pentru a adăuga elemente, de tipul linii, puncte, text la un grafic deja existent.

Tab. 5: Functii low-level uzuale

Funcția	rezultatul
points(x, y)	Adaugă puncte la un grafic.
abline(), segments()	Adaugă linii sau segmente la un grafic existent.
arrows()	Adaugă săgeți.
curve()	Adaugă o curbă care reprezintă graficul unei funcții.
rect(), polygon()	Adaugă un dreptunghi sau un poligon oarecare.
text(), mtext()	Adaugă text la o figură.
legend()	Adaugă legenda.
axis()	Adaugă o axă.

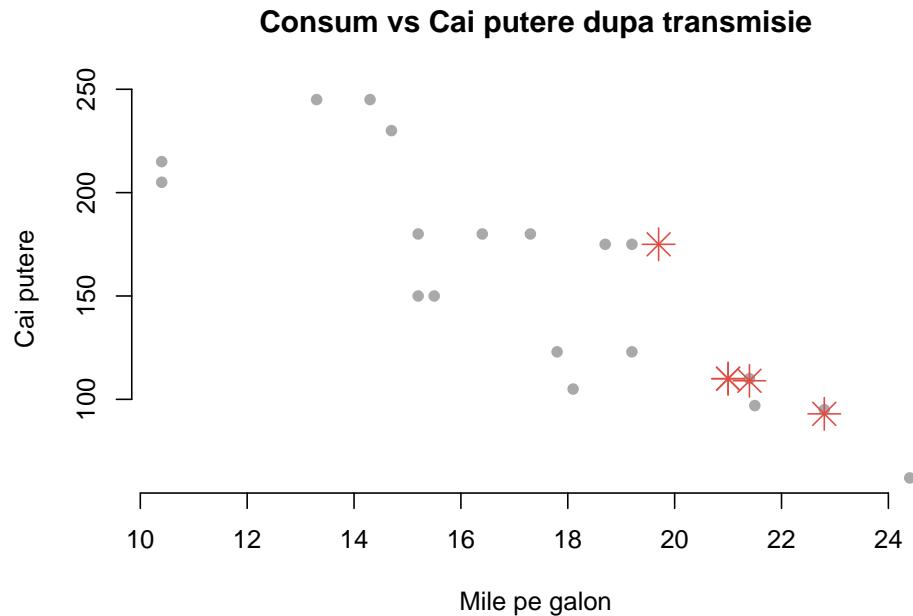
Pentru a adăuga noi puncte la un grafic deja existent putem folosi funcție `points()`. Pentru a vedea toate argumentele acestei funcții apelați `?points`.

Să considerăm următorul exemplu în care trasăm diagrama de împrăștiere după consum (`mpg`) și putere (`hp`) pentru mașinile din setul de date `mtcars` în raport cu tipul de transmisie.

```
plot(x = mtcars$mpg[mtcars$am == 0],
     y = mtcars$hp[mtcars$am == 0],
     xlab = "Mile pe galon",
     ylab = "Cai putere",
```

```
main = "Consum vs Cai putere dupa transmisie",
pch = 16,
col = "darkgrey", bty = "n")

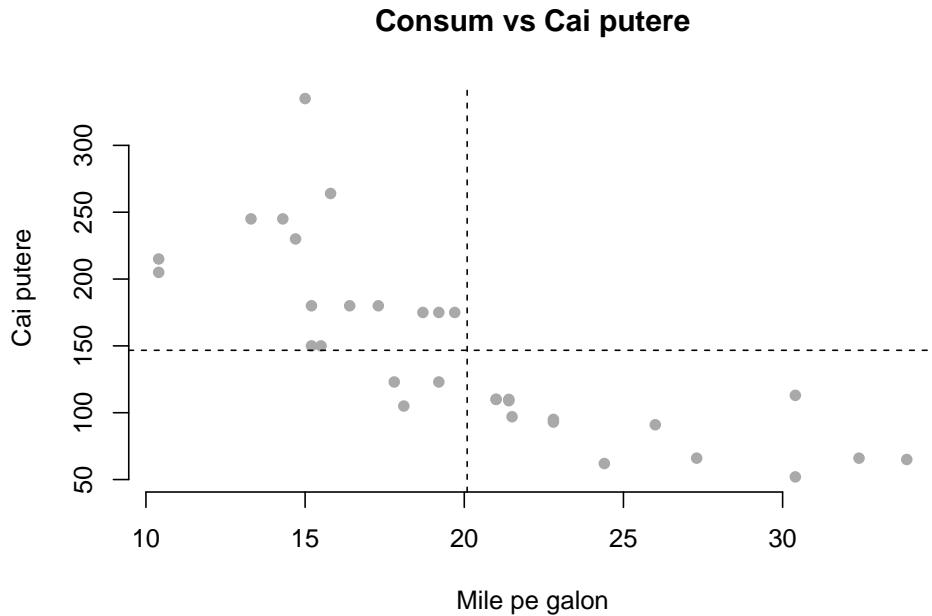
points(x = mtcars$mpg[mtcars$am == 1],
       y = mtcars$hp[mtcars$am == 1],
       pch = 8,
       cex = 2,
       col = myred)
```



Dacă vrem să adăugăm linii drepte la un grafic putem folosi comanda `abline()` sau `segments()`. De exemplu în figura de mai sus vrem să adăugăm o linie verticală și una orizontală care să marcheze media variabilelor de pe axa x și y.

```
plot(x = mtcars$mpg,
      y = mtcars$hp,
      xlab = "Mile pe galon",
      ylab = "Cai putere",
      main = "Consum vs Cai putere",
      pch = 16,
      col = "darkgrey",
      bty = "n")

abline(h = mean(mtcars$hp), lty = 2)
abline(v = mean(mtcars$mpg), lty = 2)
```

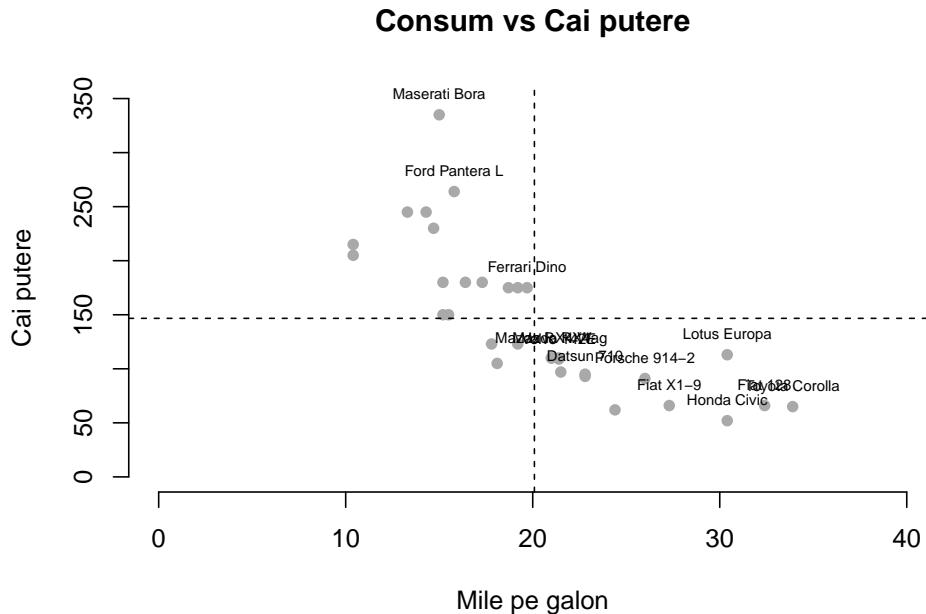


Pentru a adăuga numele mașinilor cu transmisie automată în fiecare punct putem folosi comanda `text()`. Argumentele principale ale acestei funcții sunt `x`, `y` care descriu coordonatele etichetelor și `labels` care reprezintă etichetele.

```
plot(x = mtcars$mpg,
      y = mtcars$hp,
      xlab = "Mile pe galon",
      ylab = "Cai putere",
      main = "Consum vs Cai putere",
      pch = 16,
      col = "darkgrey",
      bty = "n",
      xlim = c(0, 40),
      ylim = c(0, 350))

abline(h = mean(mtcars$hp), lty = 2)
abline(v = mean(mtcars$mpg), lty = 2)

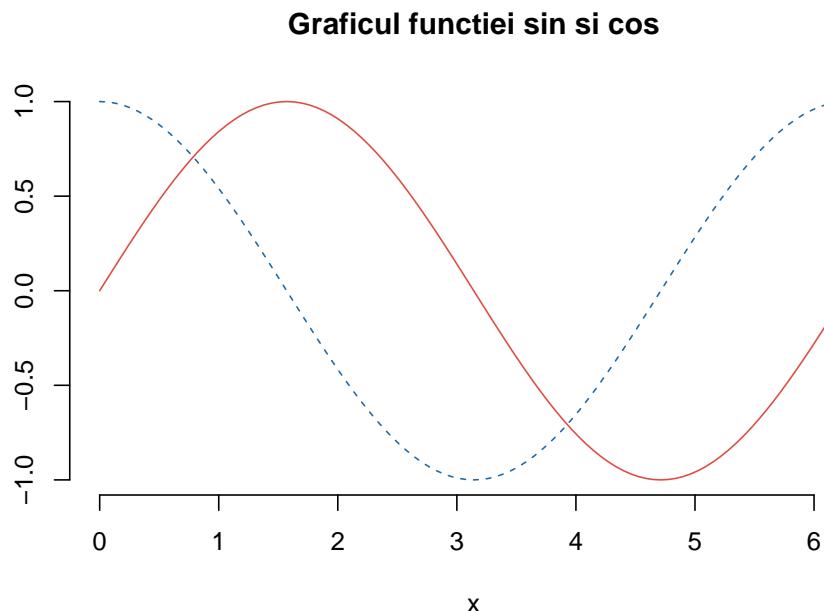
text(x = mtcars$mpg[mtcars$am == 1],
      y = mtcars$hp[mtcars$am == 1],
      labels = rownames(mtcars[mtcars$am == 1, ]),
      pos = 3,
      cex = 0.6)
```



Funcția `curve()` permite trasarea/adăugarea unei linii care descrie o funcție. Printre argumentele funcției regăsim `expr` care reprezintă expresia funcției care depinde de `x` (se pot folosi și funcții customizate), `from`, `to` care reprezintă intervalul de valori pentru `x` și `add` care permite adăugarea unei curbe la un grafic existent.

```
par(bty = "n")
curve(expr = sin(x),
      from = 0,
      to = 2*pi,
      ylab = "",
      main = "Graficul functiei sin si cos",
      col = myred)

curve(expr = cos(x),
      from = 0,
      to = 2*pi,
      add = TRUE,
      col = myblue,
      lty = 2)
```



Atunci când vrem să adăugăm o legendă la un grafic folosim funcția `legend()`. Argumentele acestei funcții se regăsesc în tabelul de mai jos.

Tab. 6: Argumentele functiei `legend()`

Argument	Rezultat
<code>x, y</code>	Coordonatele legendei - de exemplu, <code>x = 0, y = 0</code> va pune legenda la coordonatele $(0, 0)$. Alternativ, putem indica poziția unde vrem legenda (i.e. <code>"topright"</code> , <code>"topleft"</code>).
<code>legend</code>	Un vector de caractere care precizează textul care vrem să apară în legendă.
<code>pch, lty,</code> <code>lwd, col,</code> <code>pt.bg, ...</code>	Argumente grafice adiționale (pentru detalii apelați <code>?legend</code>).

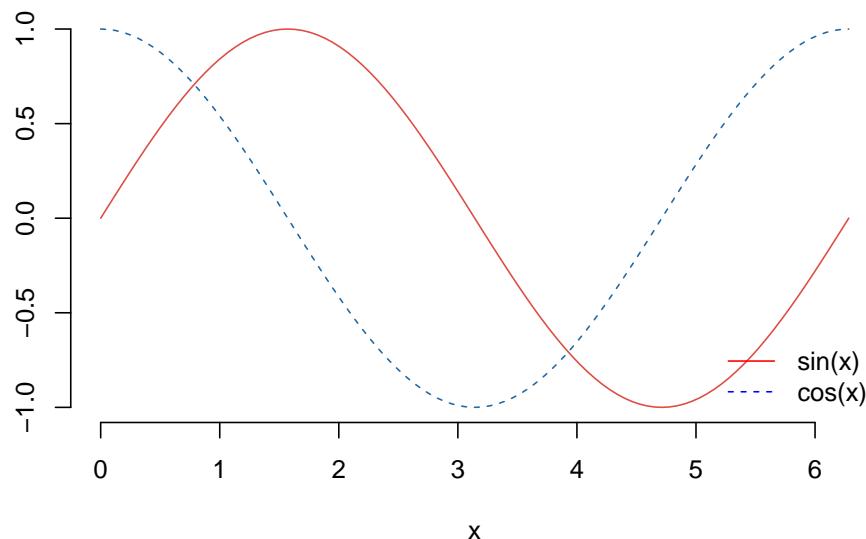
Ca exemplu să considerăm graficele de funcții de mai sus la care vrem să specificăm care grafic corespunde funcției *sin* și care funcției *cos*:

```
par(bty = "n")
curve(expr = sin(x),
       from = 0,
       to = 2*pi,
       ylab = "",
       main = "Graficul functiei sin si cos",
       col = myred)

curve(expr = cos(x),
       from = 0,
       to = 2*pi,
       add = TRUE,
       col = myblue,
       lty = 2)
```

```
legend("bottomright",
       legend = c("sin(x)", "cos(x")),
       col = c("red", "blue"),
       lty = c(1, 2),
       bty = "n")
```

Graficul functiei sin si cos



2.7 Salvarea figurilor

Odată ce am creat un grafic putem să-l salvăm într-un fișier extern. Pentru aceasta folosim funcțiile `pdf()`, `png()` sau `jpeg()`. Aceste funcții vor salva figura ca fișier de tip `.pdf`, `.png` sau `.jpeg`.

Tab. 7: Argumente pentru funcțiile `pdf`, `jpeg` și `png`

Argument	Rezultat
<code>file</code>	Directorul și numele fișierului sub formă de sir de caractere. De exemplu, pentru a salva un grafic pe desktop scriem <code>file = "/Users/.../Desktop/plot.pdf"</code> pentru un fișier pdf.
<code>width</code> , <code>height</code>	Dimensiunea graficului final în inchi.
<code>dev.off()</code>	Acesta nu este un argument al funcțiilor <code>pdf()</code> și <code>jpeg()</code> . Trebuie executat acest cod după ce trasarea graficului a fost efectuată pentru a finaliza crearea imaginii.

Pentru a salva o imagine avem de parcurs următorii pași:

1. Execută funcțiile `pdf()` sau `jpeg()` cu argumentele `file`, `width`, `height`.
2. Execută codul care generează figura (e.g. `plot(x = 1:10, y = 1:10)`)
3. Completează scrierea fișierului prin execuția comenzii `dev.off()`. Această comandă spune R-ului că am finalizat crearea fișierului.

```
# Pasul 1
pdf(file = "/Users/.../Desktop/MyPlot.pdf",      # directorul cu fisierul
```

```
width = 4, # latimea in inchii
height = 4) # inaltimea in inchii

# Pasul 2
plot(x = 1:10,
      y = 1:10)
abline(v = 0)
text(x = 0, y = 1, labels = "Ceva text aleator")

# Pasul 3
dev.off()
```

Referințe

Tilman Davies. *The Book of R. A First Course in Programming and Statistics*. No Starch Press, Inc., 1st edition, 2016. ISBN 978-1-59327-651-5. (Citat la pagina 1.)

Norman Matloff. *The Art of R Programming. A Tour of Statistical Software Design*. No Starch Press, Inc., 1st edition, 2011. ISBN 978-1-59327-384-2. (Citat la pagina 1.)

Laborator 3

Elemente de probabilități în R

Obiectivul acestui laborator este de a prezenta succint câteva funcții utile teoriei probabilităților din programul R, care este structura lor și cum le putem aplica. De asemenea, tot în acest laborator vom prezenta și câteva probleme ce se pot rezolva cu ajutorul algoritmilor aleatori.

1 Familia de funcții apply

Pe lângă buclele `for` și `while`, în R există și un set de funcții care permit scrierea și rularea într-o manieră mai compactă a codului dar și aplicarea de funcții unor grupuri de date.

- `lapply()`: Evaluează o funcție pentru fiecare element al unei liste
- `sapply()`: La fel ca `lapply` numai că încearcă să simplifice rezultatul
- `apply()`: Aplică o funcție după fiecare dimensiune a unui `array`
- `tapply()`: Aplică o funcție pe submulțimi ale unui vector
- `mapply()`: Varianta multivariată a funcției `lapply`
- `split`: Împarte un vector în grupuri definite de o variabilă de tip factor.

1.1 `lapply()`

Funcția `lapply()` efectuează următoarele operații:

1. buclează după o listă, iterând după fiecare element din acea listă
2. aplică o *funcție* fiecărui element al listei (o funcție pe care o specificăm)
3. întoarce ca rezultat tot o listă (prefixul `l` vine de la listă).

Această funcție primește următoarele trei argumente: (1) o listă `X`; (2) o funcție `FUN`; (3) alte argumente via Dacă `X` nu este o listă atunci aceasta va fi transformată într-o listă folosind comanda `as.list()`.

Considerăm următorul exemplu în care vrem să aplicăm funcția `mean()` tuturor elementelor unei liste

```
set.seed(222)
x <- list(a = 1:5, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
lapply(x, mean)
$a
[1] 3

$b
[1] 0.1996044

$c
[1] 0.7881026

$d
[1] 5.064188
```

Putem să folosim funcția `lapply()` pentru a evalua o funcție în moduri repetitive. Mai jos avem un exemplu în care folosim funcția `runif()` (permite generarea observațiilor uniform repartizate) de patru ori, de fiecare dată generăm un număr diferit de valori aleatoare. Mai mult, argumentele `min = 0` și `max = 3` sunt atribuite, prin intermediul argumentului `...`, funcției `runif`.

```
x <- 1:4
lapply(x, runif, min = 0, max = 3)
[[1]]
[1] 0.03443616

[[2]]
[1] 1.267361 1.365441

[[3]]
[1] 1.8084700 2.1902665 0.4139585

[[4]]
[1] 1.5924650 0.7355067 2.1483841 1.6082945
```

1.2 sapply()

Functia `sapply()` are un comportament similar cu `lapply()` prin faptul că funcția `sapply()` apelează intern `lapply()` pentru valorile de input, după care evaluează:

- dacă rezultatul este o listă în care fiecare element este de lungime 1, atunci întoarce un vector
- dacă rezultatul este o listă în care fiecare element este un vector de aceeași lungime (>1), se întoarce o matrice
- în caz contrar se întoarce o listă.

Considerăm exemplul de mai sus

```
set.seed(222)
x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
sapply(x, mean)
      a          b          c          d
2.5000000 0.1996044 0.7881026 5.0641876
```

1.3 split()

Funcția `split()` primește ca argument un vector sau o listă (sau un `data.frame`) și împarte datele în grupuri determinate de o variabilă de tip factor (sau o listă de factori).

Argumentele acestei funcții sunt

```
str(split)
function (x, f, drop = FALSE, ...)
```

unde

- `x` este un vector, o listă sau un `data.frame`
- `f` este un factor sau o listă de factori

Considerăm următorul exemplu în care generăm un vector de date și îl împărțim după o variabilă de tip factor creată cu ajutorul funcției `gl()` (*generate levels*).

```
x <- c(rnorm(10), runif(10), rnorm(10, 1))
f <- gl(3, 10)
split(x, f)
```



```
$`1`  
[1] -2.27414224 -0.11266780  0.61308167  0.07733545  0.57137727  0.11672493  
[7] -0.95685256 -1.90008460 -1.48972089  0.55925676  
  
$`2`  
[1] 0.91159086 0.03291829 0.78368939 0.11852882 0.64443831 0.78790988  
[7] 0.82451477 0.05642366 0.65075027 0.95426854  
  
$`3`  
[1] 2.6666242 2.6634334 1.8106280 -0.7837308 1.6575684 0.1546575  
[7] 0.4930056 -0.9031544 2.4042311 1.4106863
```

Putem folosi funcția `split` și în conjuncție cu funcția `lapply` (atunci când vrem să aplicăm o funcție FUN pe grupuri de date).

```
lapply(split(x, f), mean)  
$`1`  
[1] -0.4795692  
  
$`2`  
[1] 0.5765033  
  
$`3`  
[1] 1.157395
```

1.4 tapply()

Funcția `tapply()` este folosită pentru aplicarea unei funcții FUN pe submulțimile unui vector și poate fi văzută ca o combinație între `split()` și `sapply()`, dar doar pentru vectori.

```
str(tapply)  
function (X, INDEX, FUN = NULL, ..., default = NA, simplify = TRUE)
```

Argumentele acestei funcții sunt date de următorul tabel:

Tab. 1: Argumentele functiei tapply

Argument	Descriere
X	un vector
INDEX	este o variabilă de tip factor sau o listă de factori
FUN	o funcție ce urmează să fie aplicată
...	argumente ce vor fi atribuite funcției FUN
simplify	dacă vrem să simplificăm rezultatul

Următorul exemplu calculează media după fiecare grupă determinată de o variabilă de tip factor a unui vector numeric.

```
x <- c(rnorm(10), runif(10), rnorm(10, 1))  
f <- gl(3, 10)  
f  
[1] 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3  
Levels: 1 2 3  
tapply(x, f, mean)  
      1           2           3  
 0.0000000  0.5000000  0.0000000
```

```
-0.0007774025 0.3736457792 0.5789436983
```

Putem să aplicăm și funcții care întorc mai mult de un rezultat. În această situație rezultatul nu poate fi simplificat:

```
tapply(x, f, range)
$`1`
[1] -2.1904113 0.9249901

$`2`
[1] 0.004445296 0.998309704

$`3`
[1] -0.3379675 1.9327099
```

1.5 apply()

Funcția `apply()` este folosită cu precădere pentru a aplica o funcție liniilor și coloanelor unei matrice (care este un `array` bidimensional). Cu toate acestea poate fi folosită pe tablouri multidimensionale (`array`) în general. Folosirea funcției `apply()` nu este mai rapidă decât scrierea unei bucle `for`, dar este mai compactă.

```
str(apply)
function (X, MARGIN, FUN, ...)
```

Argumentele funcției `apply()` sunt

- `X` un tablou multidimensional
- `MARGIN` este un vector numeric care indică dimensiunea sau dimensiunile după care se va aplica funcția
- `FUN` este o funcție ce urmează să fie aplicată
- ... alte argumente pentru funcția `FUN`

Considerăm următorul exemplu în care calculăm media pe coloane într-o matrice

```
x <- matrix(rnorm(200), 20, 10)
apply(x, 2, mean) ## media fiecărei coloane
[1] 3.745002e-02 1.857656e-01 -2.413659e-01 -2.093141e-01 -2.562272e-01
[6] 8.986712e-05 7.444137e-02 -7.460941e-03 6.275282e-02 9.801550e-02
```

precum și media după fiecare linie

```
apply(x, 1, sum) ## suma fiecărei linii
[1] 2.76179139 2.5317581 0.87923177 1.80480589 0.98225832 -3.06148753
[7] -1.40358820 -0.65969812 -1.63717046 -0.29330726 -2.41486442 -3.15698523
[13] 2.27126822 -3.88290287 -3.15595194 5.41211963 2.32985530 -3.05330574
[19] -0.02110926 -1.34909559
```

2 Exerciții pregătitoare

2.1 Aruncarea cu banul

În acest exemplu vrem să simulăm aruncarea unei monede (echilibrată) folosind funcția `sample()`. Această funcție permite extragerea, cu sau fără întoarcere (`replace = TRUE` sau `replace = FALSE` - aceasta este valoarea prestabilită), a unui eșantion de volum dat (`size`) dintr-o mulțime de elemente `x`.

Spre exemplu dacă vrem să simulăm 10 aruncări cu banul atunci apelăm:

```
sample(c("H", "T"), 10, replace = TRUE)
[1] "H" "H" "H" "H" "H" "T" "T" "T" "H" "T"
```

Pentru a estima probabilitatea de apariție a stemei (H) repetăm aruncarea cu banul de 10000 de ori și calculăm raportul dintre numărul de apariții ale evenimentului $A = \{H\}$ și numărul total de aruncări:

```
# atunci cand moneda este echilibrata
a = sample(c("H", "T"), 10000, replace = TRUE)
p = sum(a == "H")/length(a)
p
[1] 0.5015
```

și pentru cazul în care moneda nu este echilibrată

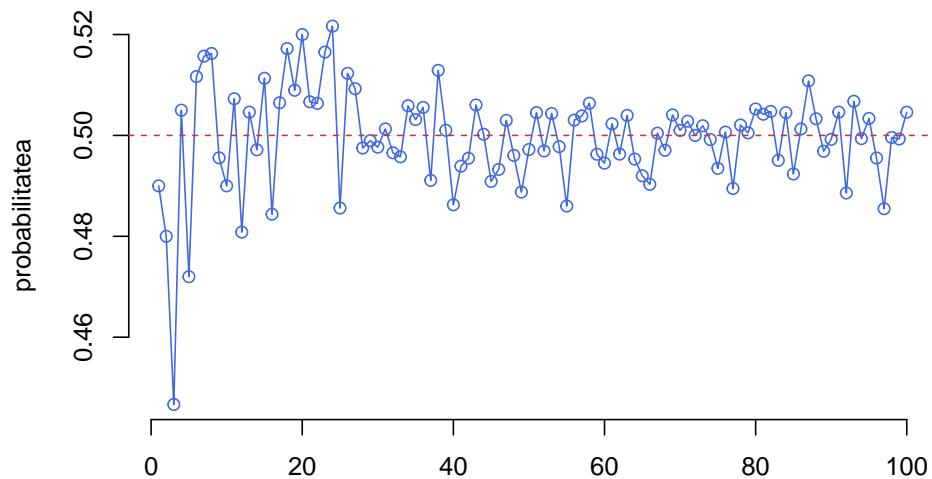
```
a = sample(c("H", "T"), 10000, replace = TRUE, prob = c(0.2, 0.8))
p = sum(a == "H")/length(a)
p
[1] 0.2012
```

Putem vedea cum evoluează această probabilitatea în funcție de numărul de repetări

```
y = rep(0,100)

for (i in 1:100){
  a = sample(c("H", "T"), i*100, replace = TRUE)
  y[i] = sum(a == "H")/length(a)
}

plot(1:100, y, type = "o", col = "royalblue", bty = "n",
      xlab = "", ylab = "probabilitatea")
abline(h = 0.5, lty = 2, col = "brown3")
```



2.2 Numărul de băieți dintr-o familie cu doi copii



O familie are doi copii. Care este probabilitatea ca ambii copii să fie băieți știind că cel puțin unul

dintre copii este băiat? Care este probabilitatea ca ambii copii să fie băieți știind că cel mai Tânăr este băiat?

Pentru a răspunde la cele două întrebări să observăm că cei doi copii (copilul mai mare și cel mai mic) pot fi ambii de sex masculin sau feminin, prin urmare avem patru combinații de sexe, pe care le presupunem egal probabile. Putem reprezenta spațiul stărilor prin

$$\Omega = \{BB, BF, FB, FF\}$$

unde $\mathbb{P}(BB) = \mathbb{P}(BF) = \mathbb{P}(FB) = \mathbb{P}(FF) = \frac{1}{4}$.

Pentru a răspunde la prima întrebare avem:

$$\begin{aligned}\mathbb{P}(BB \mid \text{cel puțin unul este băiat}) &= \mathbb{P}(BB \mid BF \cup FB \cup BB) \\ &= \frac{\mathbb{P}(BB \cap (BF \cup FB \cup BB))}{\mathbb{P}(BF \cup FB \cup BB)} = \frac{\mathbb{P}(BB)}{\mathbb{P}(BF \cup FB \cup BB)} = \frac{1}{3}.\end{aligned}$$

Iar pentru cea de-a doua întrebare:

$$\begin{aligned}\mathbb{P}(BB \mid \text{cel mai Tânăr este băiat}) &= \mathbb{P}(BB \mid FB \cup BB) \\ &= \frac{\mathbb{P}(BB \cap (FB \cup BB))}{\mathbb{P}(FB \cup BB)} = \frac{\mathbb{P}(BB)}{\mathbb{P}(FB \cup BB)} = \frac{1}{2}.\end{aligned}$$

Vom încerca să răspundem la aceste întrebări și cu ajutorul limbajului R, prin simulare. Din abordarea *frecvenționistă* am văzut că prin repetarea de N ori a unui experiment în condiții identice,

$$\mathbb{P}(A|B) \approx \frac{N(A \cap B)}{N(B)}$$

unde $N(A \cap B)$ este numărul de realizări (din N) a evenimentului $A \cap B$ iar $N(B)$ este numărul de realizări a evenimentului B . Să considerăm $N = 10^5$ și fie

$N = 10^5$

```
copil1 = sample(c("băiat", "fata"), N, replace = TRUE)
copil2 = sample(c("băiat", "fata"), N, replace = TRUE)
```

Aici `copil1` este un vector de lungime N care reprezintă sexul primului copil și în mod similar `copil2` reprezintă sexul celui de-al doilea copil.

Fie A evenimentul ca ambii copii să fie băieți și B evenimentul prin care cel mai Tânăr este băiat.

```
nB = sum(copil2 == "băiat")
nAB = sum(copil1 == "băiat" & copil2 == "băiat")

p2 = nAB/nB
```

Prin urmare probabilitatea (simulată) ca familia să aibă cei doi copii băieți știind că cel Tânăr este băiat este 0.4970443.

Considerând acum C evenimentul prin care familia are cel puțin un copil băiat, avem

```
nC = sum(copil1 == "băiat" | copil2 == "băiat")

p1 = nAB/nC
```

de unde probabilitatea ca familia să aibă doar băieți știind că cel puțin unul este băiat este 0.3300092.

2.3 Monty Hall



Sunteți participant într-un joc televizat în care gazda vă prezintă trei uși închise. Acesta vă spune că în spatele unei uși se află o mașină iar în spatele celorlalte două se află câte o capră. Jocul decurge în felul următor: trebuie să alegeti una dintre cele trei uși; gazda, care știe în spatele cărei uși se află mașina, deschide una dintre celelalte două uși, în spatele căreia se află o capră apoi vă întreabă dacă vreți să rămâneți la alegerea inițială sau vreți să alegeti cealaltă ușă rămasă închisă. Presupunând că vreți să câștigați o mașină, ce alegere preferați ?

Considerăm evenimentele C_1 , C_2 și C_3 ca indicând ușa în spatele căreia se află mașina. Aceste evenimente au probabilitatea de $1/3$ fiecare.

Evenimentul prin care jucătorul alege ușa cu numărul 1 este notat cu X_1 și evenimentul prin care gazda deschide ușa cu numărul 3 este notat cu H_3 . Avem că $\mathbb{P}(C_i|X_1) = 1/3$ iar $\mathbb{P}(H_3|C_1, X_1) = 1/2$, $\mathbb{P}(H_3|C_2, X_1) = 1$ și $\mathbb{P}(H_3|C_3, X_1) = 0$.

Avem că probabilitatea ca jucătorul să câștige adoptând schimbarea ușilor este (în condițiile în care a ales inițial poarta 1 și gazda a deschis poarta 3)

$$\begin{aligned}\mathbb{P}(C_2|H_3, X_1) &= \frac{\mathbb{P}(H_3|C_2, X_1)\mathbb{P}(C_2 \cap X_1)}{\mathbb{P}(H_3 \cap X_1)} \\ &= \frac{\mathbb{P}(H_3|C_2, X_1)\mathbb{P}(C_2 \cap X_1)}{\mathbb{P}(H_3|C_1, X_1)\mathbb{P}(C_1 \cap X_1) + \mathbb{P}(H_3|C_2, X_1)\mathbb{P}(C_2 \cap X_1) + \mathbb{P}(H_3|C_3, X_1)\mathbb{P}(C_3 \cap X_1)} \\ &= \frac{\mathbb{P}(H_3|C_2, X_1)}{\mathbb{P}(H_3|C_1, X_1) + \mathbb{P}(H_3|C_2, X_1) + \mathbb{P}(H_3|C_3, X_1)} = \frac{1}{1/2 + 1 + 0} = \frac{2}{3}\end{aligned}$$

Intuiție (să presupunem că ne aflăm în situația în care am ales ușa cu numărul 1):

Ușa 1	Ușa 2	Ușa 3	Nu schimbăam	Schimbăam
Mașină	Capră	Capră	✓	✗
Capră	Mașină	Capră	✗	✓
Capră	Capră	Mașină	✗	✓

Vom încerca să simulăm jocul descris de problema noastră:

```
monty = function(random = TRUE) {
  doors = 1:3

  if (random){
    # alege aleator unde se află mașina
    cardoor <- sample(doors,1)
  }else{
    print("Scrieti numarul ușii în spatele căreia se află mașina:")
    cardoor = scan(what = integer(), nlines = 1, quiet = TRUE)
  }

  # Monty îi spune jucătorului să aleaga ușa
  print("Monty Hall spune 'Alege o ușă, orice ușă!'")
```

```
# alegerea jucatorului (1,2 sau 3)
chosen = scan(what = integer(), nlines = 1, quiet = TRUE)

# Monty intoarce o usa cu capra (nu poate fi usa aleasa
# de jucator si nici usa cu masina)
if (chosen != cardoor){
  montydoor = doors[-c(chosen, cardoor)]
} else{
  montydoor = sample(doors[-chosen], 1)
}

# jucatorul schimba sau...
print(paste("Monty deschide usa ", montydoor, "!", sep=""))
print("Doresti sa schimbi usa (y/n)?")

reply = scan(what = character(), nlines = 1, quiet = TRUE)

# ce incepe cu "y" este da
if (substr(reply,1,1) == "y"){
  chosen = doors[-c(chosen, montydoor)]
}

# Rezultatul jocului
if (chosen == cardoor){
  print("Bravo! Ai castigat !")
} else{
  print("Pacat! Ai pierdut!")
}
```

2.4 Jocul de loto



Construiți în R o funcție care să simuleze jocul de loto 6/49. Acest joc consistă din extragerea aleatoare a 6 numere dintr-o urnă cu 49 de numere posibile, fără întoarcere. Fiecare extragere se face de manieră uniformă din numerele rămase în urnă (la a i-a extragere fiecare bilă din urnă are aceeași șansă să fie extrasă). De exemplu putem avea următorul rezultat: 10, 27, 3, 45, 12, 24.

Notă: Funcția `sample()` poate face această operație, ceea ce se cere este de a crea voi o funcție care să implementeze jocul fără a folosi funcția `sample`. Binenteles că puteți folosi funcții precum: `runif`, `floor`, `choose`, etc.

Începem prin a construi o funcție care ne permite generarea unei variabile aleatoare uniform repartizate pe mulțimea $\{1, 2, \dots, n\}$ (această funcție este cea care simulează procesul de extragere de la fiecare pas):

```
myintunif = function(n){
  # dunctia care genereaza un numar uniform intre 1 si n
  r = n*runif(1)
  u = floor(r)+1
  return(u)
}
```

Funcția care realizează extragerea fără întoarcere a k numere aleatoare din n , este:

```
myrandsample = function(n,k){  
  #  
  x = 1:n  
  q = rep(0,k)  
  
  for(i in 1:k){  
    l = length(x)  
    u = myintunif(l)  
    q[i] = x[u]  
    x = x[x!=q[i]]  
  }  
  return(q)  
}
```

Pentru a vedea ce face această funcție putem scrie:

```
n = 49  
k = 6  
  
myrandsample(n,k)  
[1] 28 19 43 1 41 39
```



Să presupunem acum că la extragerea loto de Duminică seara au fost alese numerele: `r set.seed(1223); myrandsample(49, 6)` și că toți cei 100000 de locuitori ai unui orașel și-au cumpărat un bilet. Care este probabilitatea ca o persoană care poate completa o singură grilă să piardă? (o persoană pierde dacă a nimerit cel mult două numere din cele extrase) Comparați rezultatul teoretic cu cel empiric.

Pentru $0 \leq k \leq 6$, fie A_k evenimentul ca jucătorul să fi nimerit exact k numere din cele câștigătoare. Atunci, cum putem alege cele k numere din cele 6 în $\binom{6}{k}$ moduri iar pe celelalte $6 - k$ în $\binom{43}{6-k}$ moduri, găsim că

$$\mathbb{P}(A_k) = \frac{\binom{6}{k} \binom{43}{6-k}}{\binom{49}{6}}.$$

Prin urmare probabilitatea ca jucătorul să piardă este

$$\mathbb{P}(A_0) + \mathbb{P}(A_1) + \mathbb{P}(A_2) = \frac{\binom{6}{0} \binom{43}{6} + \binom{6}{1} \binom{43}{5} + \binom{6}{2} \binom{43}{4}}{\binom{49}{6}}$$

adică $\mathbb{P}(A_0) + \mathbb{P}(A_1) + \mathbb{P}(A_2) \approx 0.9813625$.

Să testăm rezultatul empiric

```
a = c(31, 7, 17, 15, 10, 42)  
  
n = 100000  
u = replicate(n, sum(myrandsample(49, 6) %in% a))  
  
# rezultatul  
res = table(u)/n  
  
# probabilitatea empirica de pierdere
```

```
sum(res[c("0", "1", "2")])  
[1] 0.98168
```

2.5 Problema potrivirilor



Să ne imaginăm că 20 de persoane merg la operă și că fiecare persoană poartă o pălărie. În momentul în care ajung la intrare își lasă pălăria la garderobă. Pe parcursul reprezentării artistice, persoana responsabilă cu garderoba se încurcă, pierde lista cu numerele locațiilor și returnează în mod aleator pălăriile persoanelor la plecare. Care este probabilitatea ca cel puțin o persoană să fi primit pălăria cu care a venit?

Să presupunem că persoanele și pălăriile sunt numerotate de la 1 la $n = 20$ și inițial persoana i a venit cu pălăria i . Fie $\Omega = S_n$ mulțimea permutărilor cu n elemente, $\mathcal{F} = \mathcal{P}(\Omega)$ și \mathbb{P} echiprobabilitatea pe (Ω, \mathcal{F}) . Să notăm cu E_i evenimentul prin care a i -a persoană primește pălăria cu numărul i , adică primește pălăria cu care a venit. Evenimentul A prin care cel puțin o persoană a primit pălăria cu care a venit este

$$A = E_1 \cup E_2 \cup \dots \cup E_n.$$

Cum E_i reprezintă mulțimea permutărilor $\sigma \in \Sigma_n = \Omega$ pentru care $\sigma(i) = i$ avem că

$$\mathbb{P}(E_i) = \frac{(n-1)!}{n!} = \frac{1}{n}$$

deoarece $|\Omega| = n!$ iar cele $n - 1$ valori diferite de i pot fi așezate în $(n - 1)!$ moduri. În mod similar,

$$\mathbb{P}(E_i \cap E_j) = \frac{(n-2)!}{n!}$$

și în general

$$\mathbb{P}(E_{i_1} \cap E_{i_2} \cap \dots \cap E_{i_k}) = \frac{(n-k)!}{n!}.$$

Formula lui Poincaré permite calcularea probabilității dorite

$$\mathbb{P}(A) = \mathbb{P}(E_1 \cup E_2 \cup \dots \cup E_n) = \sum_{k=1}^n (-1)^{k-1} \sum_{1 \leq i_1 < \dots < i_k \leq n} \mathbb{P}(E_{i_1} \cap E_{i_2} \cap \dots \cap E_{i_k})$$

de unde

$$\mathbb{P}(A) = \sum_{k=1}^n (-1)^{k-1} \binom{n}{k} \frac{(n-k)!}{n!} = \sum_{k=1}^n \frac{(-1)^{k-1}}{k!} \rightarrow 1 - \frac{1}{e} \approx 0.63212$$

Să vedem că același rezultat îl obținem și prin simulare:

```
# repetam experimentul de un nr mare de ori  
m = 10000  
  
# nr de persoane  
n = 20
```

```
n.potriviri = rep(0, m)

persoane = 1:n

for (i in 1:m){
  palarii = sample(1:n, n)
  n.potriviri[i] = sum(palarii == persoane)
}

# proporția persoanelor cu cel puțin o potrivire
sum(n.potriviri > 0) / m
[1] 0.6245
```

2.6 Ruina jucătorului



Un bărbat vrea să își cumpere un obiect (de exemplu o mașină sau o casă) care costă N unități monetare. Să presupunem că el are economisit un capital de $0 < k < N$ unități monetare și însearcă să câștige restul jucând un joc de noroc cu managerul unei bănci. Jocul este următorul: bărbatul aruncă o monedă echilibrată în mod repetat dacă moneda pică cap (H) atunci managerul îi dă o unitate monetară, în caz contrar bărbatul plătește o unitate monetară bancii. Jocul continuă până când unul din două evenimente se realizează: sau câștigă suma necesară și își cumpără obiectul dorit sau pierde banii și ajunge la faliment. Ne întrebăm care este probabilitatea să ajungă la faliment?

Fie A evenimentul ca bărbatul să ajungă la ruină și B evenimentul ca la prima aruncare moneda a picat cap. Atunci din formula probabilității totale avem

$$\mathbb{P}_k(A) = \mathbb{P}_k(A|B)\mathbb{P}(B) + \mathbb{P}_k(A|B^c)\mathbb{P}(B^c)$$

unde \mathbb{P}_k este probabilitatea calculată în funcție de valoarea k a capitalului inițial al jucătorului. Să observăm că $\mathbb{P}_k(A|B)$ devine $\mathbb{P}_{k+1}(A)$ deoarece dacă la prima aruncare avem cap atunci capitalul inițial a crescut la $k+1$. În mod similar, dacă la prima aruncare am obținut coadă atunci $\mathbb{P}_k(A|B^c) = \mathbb{P}_{k-1}(A)$. Notând cu $p_k = \mathbb{P}_k(A|B)$ obținem următoarea ecuație

$$p_k = \frac{1}{2}p_{k+1} + \frac{1}{2}p_{k-1},$$

cu valorile inițiale $p_0 = 1$ (dacă jucătorul a pornit cu un capital inițial nul atunci el este în faliment) și respectiv $p_N = 0$ (dacă jucătorul are din start suma necesară pentru a achiziționa obiectul dorit atunci nu mai are loc jocul).

O simulare a jocului pentru $N = 50$ și $k = 5$ este prezentată de următoarea funcție:

```
ruina = function(N, k){
  flag = TRUE

  joc = 0
  capital = k
  y = capital

  while(flag){
    x = 2*rbinom(1,1,0.5)-1
```

```
capital = capital + x
y = c(y, capital)

joc = joc + 1

if (capital == 0 || capital == N){
  flag = FALSE
}
}

return(y) # daca am 0 este ruina altfel este succes
}
```

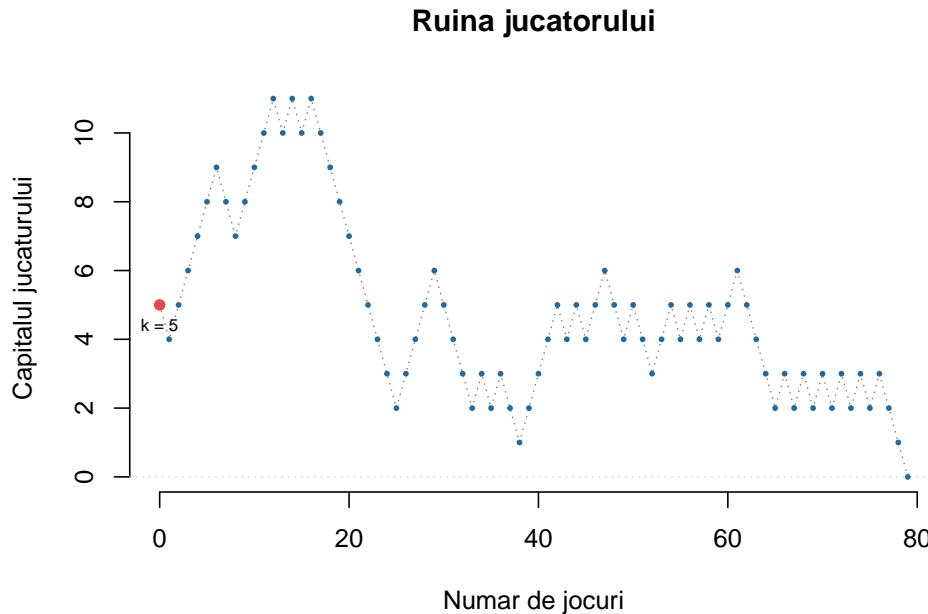
Putem ilustra grafic jocul după cum urmează:

```
N = 50
k = 5

set.seed(1234)

y = ruina(N, k)
joc = length(y) - 1 # nr de jocuri

plot(0:joc, y, type = "l",
      main = "Ruina jucatorului",
      xlab = "Numar de jocuri",
      ylab = "Capitalul jucaturului",
      bty = "n",
      lty = 3, col = "grey50")
abline(h = c(0,N), col = "lightgrey", lty = 3)
points(0:joc, y, col = myblue,
       pch = 16,
       cex = 0.5)
points(0, k, col = myred, pch = 16)
text(0, k, labels = paste0("k = ", k), pos = 1, cex = 0.7)
```



Dacă definim $b_k = p_k - p_{k-1}$ pentru $k \geq 1$ atunci $b_k = b_{k-1}, \forall k \geq 2$. Prin urmare $b_k = b_1$ și $p_k = b_1 + p_{k-1} = kb_1 + p_0$. Observând că $b_1 + \dots + b_N = p_N - p_0 = -1$ deducem că $b_1 = -\frac{1}{N}$ iar $p_k = 1 - \frac{k}{N}$.

Dorim să repetăm experimentul de $M = 1000$ de ori (pentru valorile inițiale $N = 50$ și $k = 5$) și ne interesăm de câte ori jucătorul a ajuns la faliment.

```
N = 50
k = 5
M = 1000
# Obs. - rezultatul functiei ruina trebuie modificat
joc = replicate(M, ruina(N, k)) # repeta functia de M ori
proba_ruina = sum(joc == 0)/M
```

Am obținut că probabilitatea empirică de faliment este 0.9 iar cea teoretică este 0.9.

3 Aplicația 1: Verificarea egalității a două polinoame

În această secțiune ne propunem să abordăm următoarea problemă:



Având date două polinoame $F(x)$ și $G(x)$, cu $F(x) = \prod_{i=1}^d (x - a_i)$ și $G(x)$ dat sub forma canonică $(\sum_{i=0}^d c_i x^i)$, vrem să verificăm dacă are loc identitatea

$$F(x) \stackrel{?}{=} G(x).$$

Stim că două polinoame sunt egale atunci când coeficienții lor în descompunerea canonică sunt egali. Observăm că dacă am transforma polinomul $F(x)$ la forma sa canonică prin înmulțirea consecutivă a monomului i cu produsul primelor $i-1$ monoame atunci am avea nevoie de $\Theta(d^2)$ operații. În cele ce urmează presupunem că fiecare operație de înmulțire se poate face în timp constant, ceea ce nu corespunde în totalitate cu realitatea în special în cazurile în care vrem să înmulțim coeficienți mari.

Ne propunem să construim un algoritm randomizat care să verifice această egalitate într-un număr mai mic de operații ($O(d)$ operații). Să presupunem că d este gradul maxim al lui x (exponentul cel mai mare) în $F(x)$ și $G(x)$. Algoritmul poate fi descris astfel: alegem uniform un număr r din mulțimea $\{1, 2, \dots, 100d\}$ (prin uniform înțelegem că cele 100d numere au aceeași șansă să fie alese) și calculăm valorile lui $F(r)$ și $G(r)$. Dacă $F(r) \neq G(r)$ atunci algoritmul întoarce că cele două polinoame nu sunt egale iar dacă $F(r) = G(r)$ atunci algoritmul întoarce că cele două polinoame sunt egale. Algoritmul poate greși doar dacă cele două polinoame nu sunt egale dar $F(r) = G(r)$. Vrem să evaluăm această probabilitate.

Experimentul nostru poate fi modelat cu ajutorul tripletului $(\Omega, \mathcal{F}, \mathbb{P})$ unde $\Omega = \{1, 2, \dots, 100d\}$, $\mathcal{F} = \mathcal{P}(\Omega)$ iar \mathbb{P} este echirepartiția pe Ω . Fie E evenimentul că algoritmul greșește, acest eveniment se realizează doar dacă numărul aleator r este o rădăcină a polinomului $F(x) - G(x)$, de grad cel mult d . Cum din *Teorema Fundamentală a Algebrei* acest polinom nu poate avea mai mult de d rădăcini rezultă că

$$\mathbb{P}(\text{algoritmul greseste}) = \mathbb{P}(E) \leq \frac{d}{100d} = \frac{1}{100}.$$

Putem să îmbunătățim această probabilitate? O variantă ar fi să mărim spațiul stărilor la $\Omega = \{1, 2, \dots, 1000d\}$ și atunci șansa ca algoritmul să greșească ar fi de $\frac{1}{1000}$. O altă variantă ar fi să repetăm procedeul de mai multe ori, iar în această situație algoritmul ar fi eronat doar dacă ar întoarce că cele două polinoame sunt egale când în realitate ele nu sunt. Atunci când repetăm procesul, alegerea numărului aleator r se poate face în două moduri diferite: cu întoarcere (nu ținem cont de numerele ieșite) sau fără întoarcere (ținem cont de numerele ieșite).

Dacă luăm E_i evenimentul prin care la a i-a rulare a algoritmului am extras o rădăcină r_i astfel încât $F(r_i) = G(r_i)$, atunci probabilitatea ca algoritmul să întoarcă un răspuns greșit după k repetări, este

$$\mathbb{P}(E_1 \cap E_2 \cap \dots \cap E_k).$$

În cazul în care alegerea se face cu întoarcere (evenimentele sunt independente) obținem

$$\mathbb{P}(E_1 \cap E_2 \cap \dots \cap E_k) = \prod_{i=1}^k \mathbb{P}(E_i) \leq \prod_{i=1}^k \frac{d}{100d} = \left(\frac{1}{100}\right)^k,$$

iar dacă extragerea s-a făcut fără întoarcere (evenimentele nu mai sunt independente), atunci folosind regula de multiplicare avem

$$\begin{aligned} \mathbb{P}(E_1 \cap E_2 \cap \dots \cap E_k) &= \mathbb{P}(E_1)\mathbb{P}(E_2|E_1) \dots \mathbb{P}(E_k|E_1 \cap \dots \cap E_{k-1}) \\ &\leq \prod_{i=1}^k \frac{d - (i - 1)}{100d - (i - 1)} \leq \left(\frac{1}{100}\right)^k. \end{aligned}$$

```

Fx = function(x){
  return((x+1)*(x-2)*(x+3)*(x-4)*(x+5)*(x-6))
}

Gx = function(x){
  return(x^6 - 7*x^3 + 25)
}

Gx2 = function(x){
  return(x^6 - 3*x^5 - 41*x^4 + 87*x^3 + 400*x^2 - 444*x - 720)
}

```

```
comparePols = function(Fx, Gx){  
  k = 3 # repetam algoritmul de 3 ori  
  d = 6 # gradul polinomului  
  
  for (i in 1:k){  
    r = floor(100*d*runif(1)) + 1  
  
    f1 = Fx(r)  
    g1 = Gx(r)  
  
    if (f1!=g1){  
      return(cat("Cele doua polinoame sunt diferite ! \nPentru r =",  
                r, "avem ca F(r)!=G(r) (", f1, "!=" , g1, ")"))  
    }  
  
  }  
  
  return(cat("Polinoamele sunt egale. Eroarea este de", 100^(-k)))  
}  
  
comparePols(Fx, Gx)  
Cele doua polinoame sunt diferite !  
Pentru r = 438 avem ca F(r)!=G(r) ( 7.010787e+15 != 7.060649e+15 )  
  
comparePols(Fx, Gx2)  
Polinoamele sunt egale. Eroarea este de 1e-06
```

4 Aplicația 2: Verificarea produsului a două matrice

În cele ce urmează vom prezenta o altă aplicație în care algoritmul randomizat este mult mai eficient decât orice algoritm determinist cunoscut până în acest moment.



Să presupunem că avem trei matrice pătratice \mathbf{A} , \mathbf{B} și \mathbf{C} de dimensiune $n \times n$. Pentru simplitate considerăm că avem de-a face cu matrice cu elemente de 0 și 1 iar operațiile se fac mod 2. Vrem să construim un algoritm randomizat care să verifice egalitatea:

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{C} \pmod{2}.$$

O modalitate ar fi să calculăm elementele matricii $\mathbf{C}' = \mathbf{A} \cdot \mathbf{B}$, folosind relația $C'_{i,j} = \sum_{k=1}^n A_{i,k}B_{k,j}$, și să le comparăm cu elementele matricii \mathbf{C} . Metoda *naivă* de multiplicare a celor două matrice necesită cel mult $O(n^3)$ operații. Știm că există algoritmi (netriviali) care permit multiplicarea matricelor într-un număr mai mic de operații, ca de exemplu:



(Strassen 1969) Este posibil să înmulțim două matrice în aproape $n^{\log_2 7} \approx n^{2.81}$ operații.

iar o versiune îmbunătățită a algoritmului lui Strassen (și cea care deținea recordul până în 2014¹)

¹Actualul record este $O(n^{2.3728})$ dat de Vassilevska Williams



(Coppersmith-Winograd 1987) Este posibil să înmulțim două matrice în aproape $n^{2.376}$ operații.

Noi nu vom vorbi despre algoritmi de multiplicare a două matrice, ci de algoritmi de verificare a acestei operații. Pentru aceasta vom considera un algoritm randomizat care verifică înmulțirea în aproximativ n^2 operații ($O(n^2)$) cu precizarea că acest algoritm poate conduce la un răspuns eronat.



(Freivalds 1979) Există un algoritm probabilist care poate verifica dacă $\mathbf{A} \cdot \mathbf{B} = \mathbf{C} \pmod{2}$ în $O(n^2)$ operații și având o eroare de 2^{-200} .

Algoritmul lui de bază a lui Freivalds este următorul:

1. Alegem uniform un vector $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_n) \in \{\mathbf{0}, \mathbf{1}\}^n$ (unde prin ales uniform înțelegem că fiecare r_i este ales de manieră independentă cu probabilitatea de 0.5 să ia valoarea 0 sau 1²)
2. Calculăm $y = \mathbf{A}\mathbf{Br}$ și $z = \mathbf{Cr}$. Dacă $y = z$ atunci algoritmul întoarce *multiplicarea este corectă* altfel *multiplicarea matricelor este eronată*.

Observăm că acest algoritm necesită trei operații de multiplicare între o matrice și un vector, prin urmare acesta necesită $O(n^2)$ pași. De asemenea, remarcăm că algoritmul întoarce un răspuns eronat atunci când $\mathbf{A} \cdot \mathbf{B} \neq \mathbf{C}$ dar $\mathbf{A}\mathbf{Br} = \mathbf{Cr}$. Probabilitatea ca algoritmul să întoarcă un răspuns eronat verifică

$$\mathbb{P}(\text{algoritmul întoarce răspuns gresit}) = \mathbb{P}(\mathbf{A}\mathbf{Br} = \mathbf{Cr}) \leq \frac{1}{2}.$$

Pentru a verifica acest rezultat, fie $\mathbf{D} = \mathbf{A} \cdot \mathbf{B} - \mathbf{C} \neq \mathbf{0}$. Evenimentul $\{\mathbf{A}\mathbf{Br} = \mathbf{Cr}\}$ implică $\mathbf{Dr} = \mathbf{0}$ și cum $\mathbf{D} \neq \mathbf{0}$ putem presupune că elementul $d_{1,1} \neq 0$. Deoarece $\mathbf{Dr} = \mathbf{0}$ rezultă că $\sum_{j=1}^n d_{1,j}r_j = 0$ sau, echivalent

$$r_1 = -\frac{\sum_{j=2}^n d_{1,j}r_j}{d_{1,1}}.$$

Avem

$$\begin{aligned} \mathbb{P}(\mathbf{A}\mathbf{Br} = \mathbf{Cr}) &= \sum_{(x_2, \dots, x_n) \in \{0,1\}^n} \mathbb{P}(\{\mathbf{A}\mathbf{Br} = \mathbf{Cr}\} \cap \{(\mathbf{r}_2, \dots, \mathbf{r}_n) = (\mathbf{x}_2, \dots, \mathbf{x}_n)\}) \\ &\leq \sum_{(x_2, \dots, x_n) \in \{0,1\}^n} \mathbb{P}\left(\left\{r_1 = -\frac{\sum_{j=2}^n d_{1,j}r_j}{d_{1,1}}\right\} \cap \{(r_2, \dots, r_n) = (x_2, \dots, x_n)\}\right) \\ &\leq \sum_{(x_2, \dots, x_n) \in \{0,1\}^n} \mathbb{P}\left(\left\{r_1 = -\frac{\sum_{j=2}^n d_{1,j}r_j}{d_{1,1}}\right\}\right) \mathbb{P}(\{(r_2, \dots, r_n) = (x_2, \dots, x_n)\}) \\ &\leq \sum_{(x_2, \dots, x_n) \in \{0,1\}^n} \frac{1}{2} \mathbb{P}(\{(r_2, \dots, r_n) = (x_2, \dots, x_n)\}) \\ &= \frac{1}{2}. \end{aligned}$$

În relațiile de mai sus am folosit faptul că r_1 și (r_2, \dots, r_n) sunt independente.

Pentru a îmbunătății eroarea algoritmului putem să repetăm procedeul de k ori³ și în acest caz probabilitatea de eroare devine 2^{-k} iar numărul de operații $O(kn^2)$.

²Putem să ne imaginăm că aruncăm cu un ban echilibrat de n ori și convertim capul în 1 și pajura în 0.

³Algoritmul lui Freivalds presupune $k = 200$.

```
FreivaldsAlg = function(A, B, C){  
  # Algoritmul Freivalds  
  n = dim(A)[1] # dimensiunea matricelor  
  
  k = ceiling(log(n)) # numarul de repetari ale algoritmului  
  
  for (i in 1:k){  
    r = rbinom(n, 1, prob = 0.5)  
  
    y = B%*%r  
    y = A%*%y  
    y = y%/%2  
  
    z = C%*%r  
    z = z%/%2  
  
    if (any(y != z)){  
      return(print("Multiplicarea matricelor este incorecta!"))  
    }  
  
  }  
  
  return(cat("Multiplicarea matricelor este corecta!  
            \nProbabilitatea de eroare a algoritmului este", 2^{ -k}))  
}  
  
# Exemplul 1  
set.seed(1234)  
A = matrix(rbinom(225, 1, 0.5), nrow = 15)  
B = matrix(rbinom(225, 1, 0.5), nrow = 15)  
  
C = A%*%B  
C = C%/%2  
  
FreivaldsAlg(A, B, C)  
Multiplicarea matricelor este corecta!  
  
Probabilitatea de eroare a algoritmului este 0.125  
  
# Exemplul 2  
set.seed(5678)  
A1 = matrix(rbinom(625, 1, 0.5), nrow = 25)  
B1 = matrix(rbinom(625, 1, 0.5), nrow = 25)  
  
C1 = A1%*%t(B1)  
C1 = C1%/%2  
  
C2 = A1%*%B1  
C2 = C2%/%2  
  
FreivaldsAlg(A1, B1, C1)  
[1] "Multiplicarea matricelor este incorecta!"  
FreivaldsAlg(A1, B1, C2)
```

Multiplicarea matricelor este corecta!

Probabilitatea de eroare a algoritmului este 0.0625

```
# Exemplul 3
set.seed(5678910)
A3 = matrix(rbinom(1000000, 1, 0.5), nrow = 1000)
B3 = matrix(rbinom(1000000, 1, 0.5), nrow = 1000)

C3 = A3%*%B3
C3 = C3%/%2
```

```
FreivaldsAlg(A3, B3, C3)
```

Multiplicarea matricelor este corecta!

Probabilitatea de eroare a algoritmului este 0.0078125

Laborator 4

Variabile aleatoare discrete în R

Obiectivul acestui laborator este de a prezenta succint câteva funcții utile teoriei probabilităților din programul R, care este structura lor și cum le putem aplica. De asemenea, tot în acest laborator vom prezenta și câteva probleme ce se pot rezolva cu ajutorul algoritmilor aleatori.

Obiectivul acestui laborator este de a introduce variabilele discrete cu ajutorul limbajului R.

1 Repartiții și elemente aleatoare în R

R pune la dispoziție majoritatea repartițiilor uzuale. Tabelul de mai jos prezintă numele și parametrii acestora:

Tab. 1: Numele și parametrii repartițiilor uzuale în R

Repartiția	Nume	Parametrii	Valori prestatibile
Beta	beta	shape1, shape2	
Binomial	binom	size, prob	
Cauchy	cauchy	location, scale	location = 0, scale = 1
Chi-Squared	chisq	df	
Exponential	exp	rate (=1/mean)	rate = 1
Fisher	f	df1, df2	
Gamma	gamma	shape, rate (=1/scale)	rate = 1
Hypergeometric	hyper	m, n, k	
Log-Normal	lnorm	mean, sd	mean = 0, sd = 1
Logistic	logis	location, scale	location = 0, scale = 1
Normal	norm	mean, sd	mean = 0, sd = 1
Poisson	pois	lambda	
Student	t	df	
Uniform	unif	min, max	min = 0, max = 1
Weibull	weibull	shape	

Pentru fiecare repartitie, există patru comenzi în R prefixate cu literele **d**, **p**, **q** și **r** și urmate de numele repartitiei (coloana a 2-a). De exemplu **dbinom**, **pbinom**, **qbinom** și **rbinom** sunt comenzi corespunzătoare repartitiei binomiale pe când **dgeom**, **pgeom**, **qgeom** și **rgeom** sunt cele corespunzătoare repartitiei geometrice.

- **dname**: calculează densitatea atunci când vorbim de o variabilă continuă sau funcția de masă atunci când avem o repartitie discretă ($\mathbb{P}(X = k)$)
- **pname**: calculează funcția de repartitie, i.e. $F(x) = \mathbb{P}(X \leq x)$
- **qname**: reprezintă funcția cuantilă, cu alte cuvinte valoarea pentru care funcția de repartitie are o anumită probabilitate; în cazul continuu, dacă **pname(x) = p** atunci **qname(p) = x** iar în cazul discret întoarce cel mai mic întreg u pentru care $\mathbb{P}(X \leq u) \geq p$.
- **rname**: generează observații independente din repartitia dată

Avem următoarele exemple:

```
# Functia de repartitie pentru binomiala
pbinom(c(3,5), size = 10, prob = 0.5)
[1] 0.1718750 0.6230469

# Genereaza observatii din repartitia binomiala
rbinom(5, size = 10, prob = 0.5)
[1] 6 4 5 4 6

# Calculeaza functia de masa in diferite puncte
dbinom(0:7, size = 10, prob = 0.3)
[1] 0.028247525 0.121060821 0.233474440 0.266827932 0.200120949 0.102919345
[7] 0.036756909 0.009001692

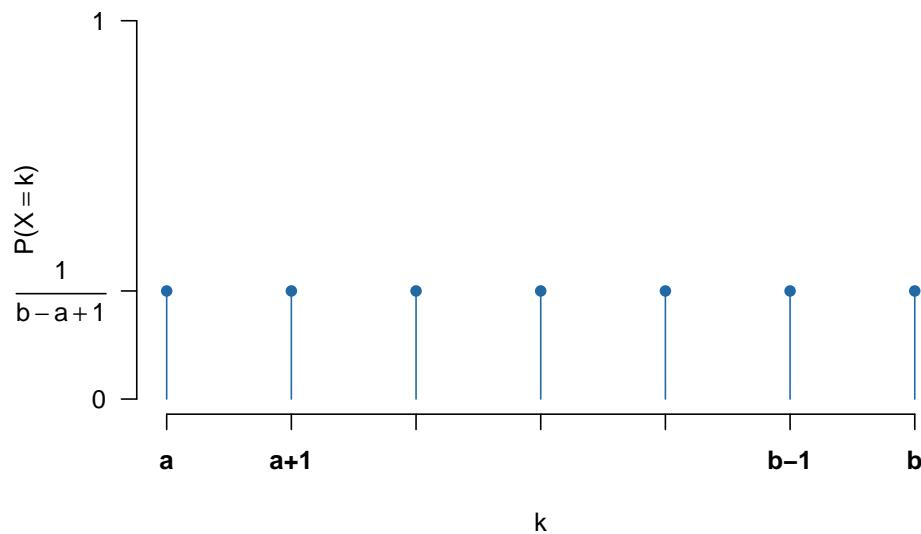
# pentru repartitia Poisson
dpois(1:5, lambda = 3)
[1] 0.1493612 0.2240418 0.2240418 0.1680314 0.1008188
rpois(10, lambda = 3)
[1] 1 2 2 7 1 1 3 2 2 2
```

1.1 Repartiția uniformă discretă

O variabilă aleatoare X este repartizată *uniform* pe mulțimea $\{a, a + 1, \dots, b\}$, și se notează $X \sim \mathcal{U}(\{a, a + 1, \dots, b\})$, are funcția de masă (*PMF - probability mass function*) dată de

$$\mathbb{P}(X = k) = \frac{1}{b - a + 1}, \quad k \in \{a, a + 1, \dots, b\}$$

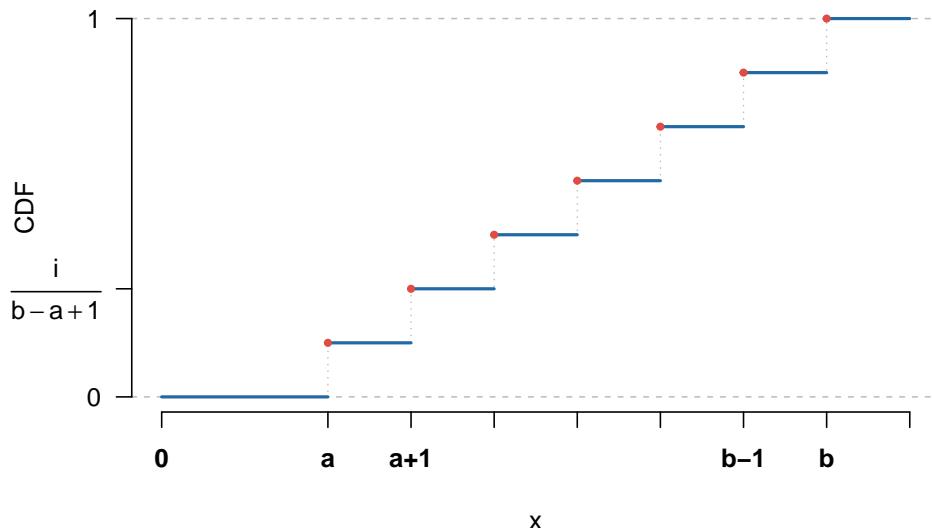
PMF a repartitiei uniformei



Funcția de repartitie a repartitiei uniforme $\mathcal{U}(\{a, a + 1, \dots, b\})$ este dată de

$$F_X(x) = \mathbb{P}(X \leq x) = \frac{\lfloor x \rfloor - a + 1}{b - a + 1}, \quad x \in [a, b].$$

Functia de repartitie a uniformei

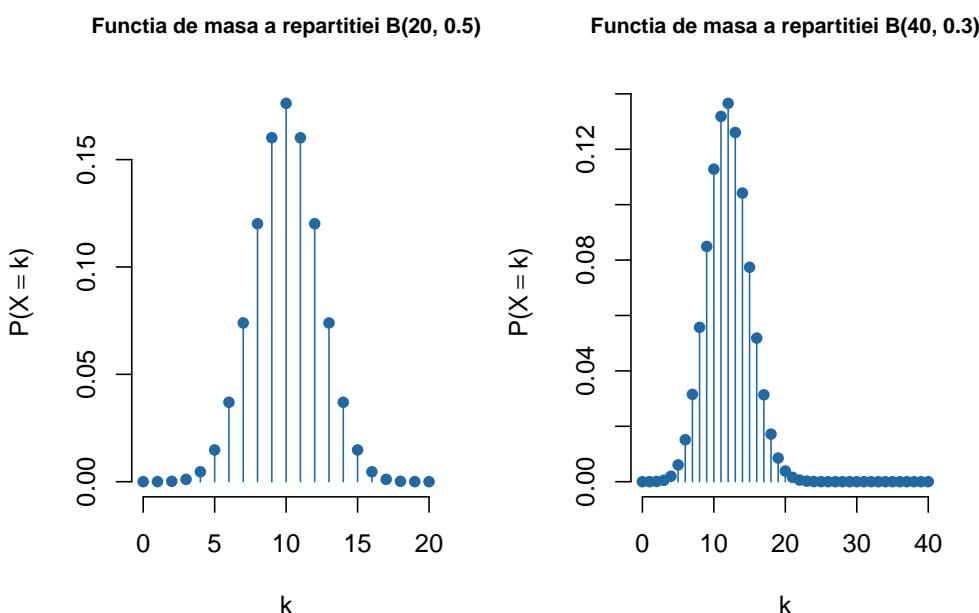


1.2 Repartiția binomială $\mathcal{B}(n, p)$

Spunem că variabila aleatoare X este repartizată binomial de parametrii $n \geq 1$ și $p \in [0, 1]$, și se notează cu $X \sim \mathcal{B}(n, p)$, dacă funcția de masă este

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

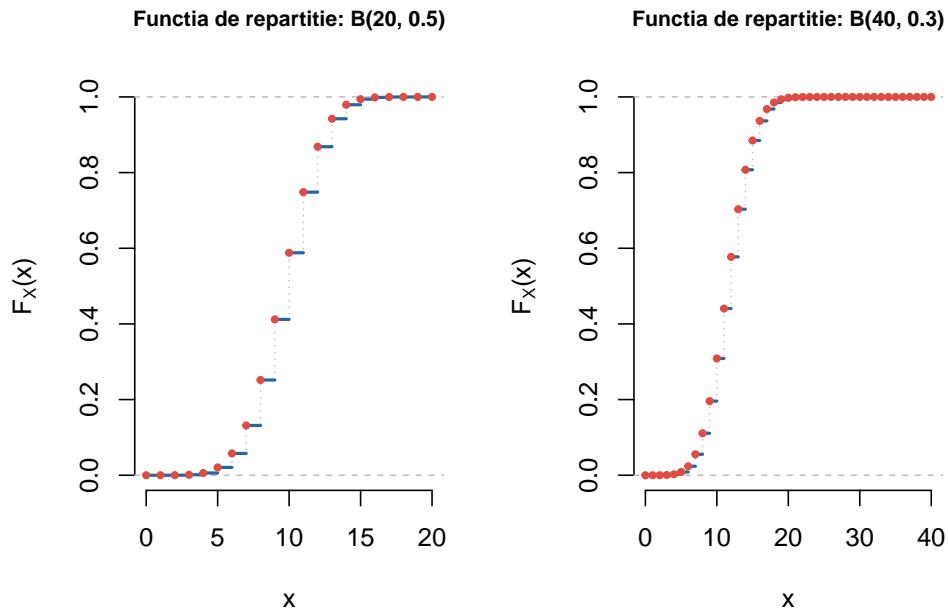
Pentru a ilustra funcția de masă vom considera repartițiile $\mathcal{B}(20, 0.5)$ și $\mathcal{B}(40, 0.3)$:



Funcția de repartiție a repartiției binomiale $\mathcal{B}(n, p)$ este dată de

$$F_X(x) = \mathbb{P}(X \leq x) = \sum_{k=0}^{\lfloor x \rfloor} \binom{n}{k} p^k (1-p)^{n-k}$$

care în cazul celor două exemple date devine

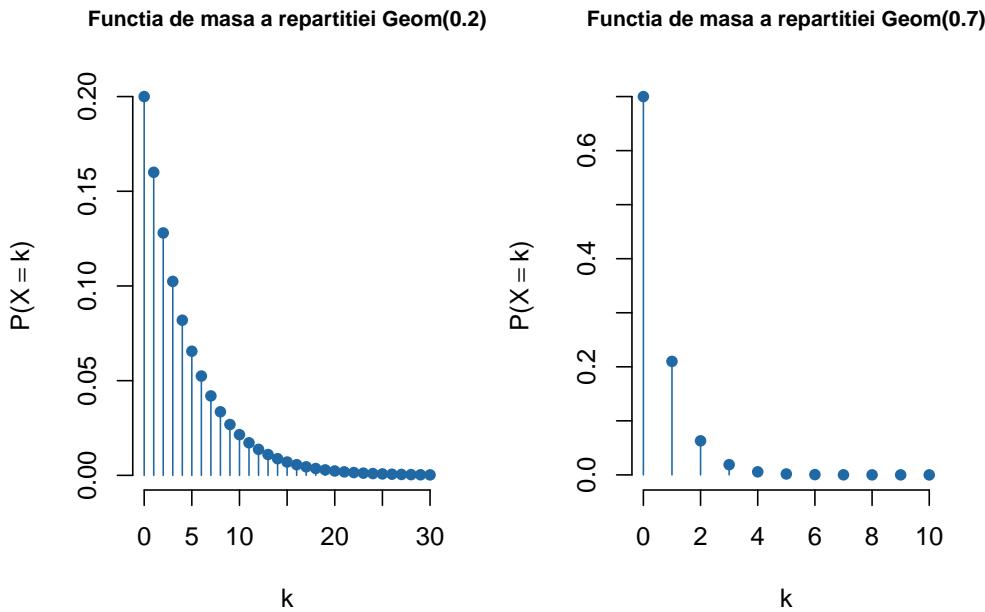


1.3 Repartiția geometrică $Geom(p)$

Variabila aleatoare X repartizată geometric de parametru $p \in (0, 1)$, $X \sim Geom(p)$, are funcția de masă

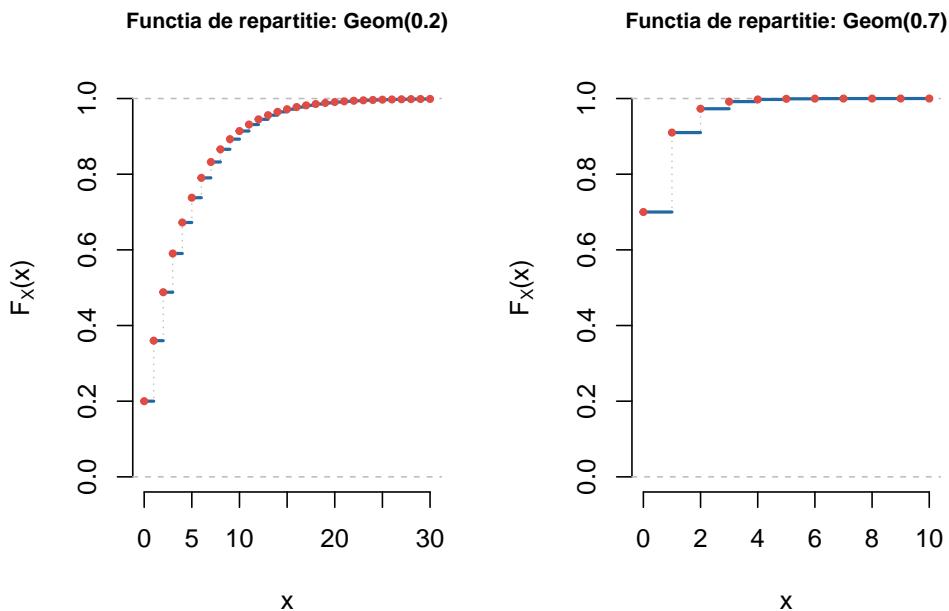
$$\mathbb{P}(X = k) = (1 - p)^{k-1} p, \quad k \in \{1, 2, \dots\}$$

Pentru $p = 0.2$ și respectiv $p = 0.7$ avem



Funcția de repartiție a variabilei $X \sim Geom(p)$ este egală cu

$$F_X(k) = \mathbb{P}(X \leq k) = 1 - (1 - p)^k$$

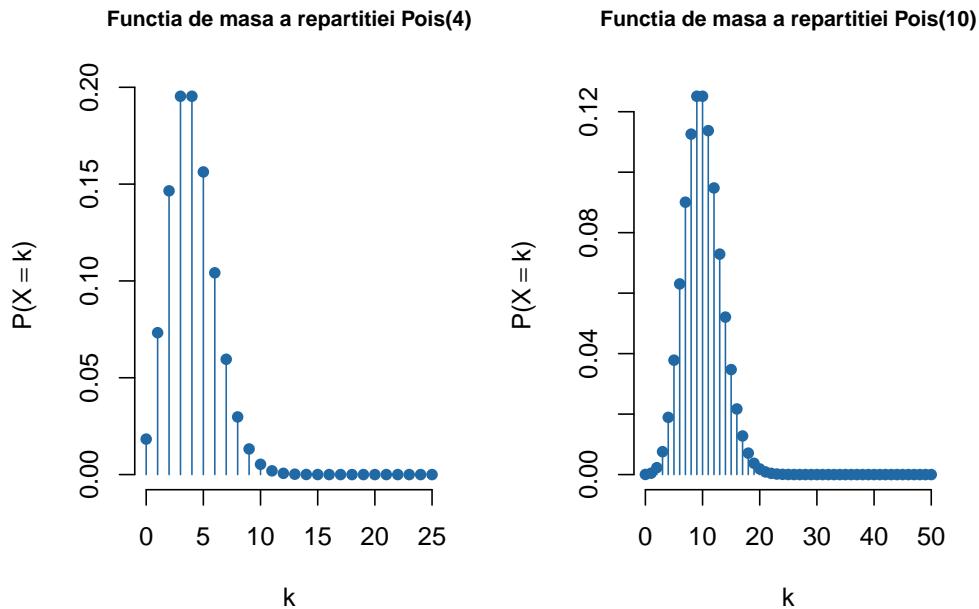


1.4 Repartiția Poisson $Pois(\lambda)$

O variabilă aleatoare X repartizată Poisson de parametru $\lambda > 0$, $X \sim Pois(\lambda)$, are funcția de masă data de

$$\mathbb{P}(X = k) = e^{-\lambda} \frac{\lambda^k}{k!}, \quad k \geq 0.$$

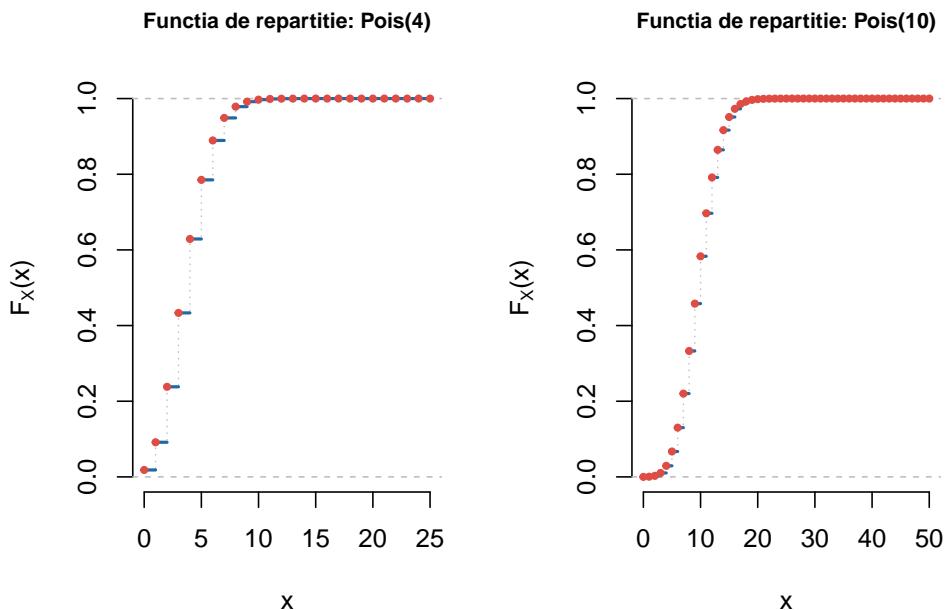
Pentru $\lambda = 4$ și $\lambda = 10$ avem



Functia de repartitie a lui $X \sim Pois(\lambda)$ este

$$F_X(x) = \mathbb{P}(X \leq x) = e^{-\lambda} \sum_{k=0}^{\lfloor x \rfloor} \frac{\lambda^k}{k!}$$

care în cazul exemplelor considerate mai sus devine



2 Generarea unei variabile aleatoare discrete



Definiți o funcție care să genereze un eșantion de talie n dintr-o distribuție discretă definită pe mulțimea $\{x_1, \dots, x_N\}$ cu probabilitățile $\{p_1, \dots, p_N\}$. Pentru început încercați cu v.a. de tip Bernoulli.

Avem următoarea funcție:

```
GenerateDiscrete = function(n = 1, x, p, err = 1e-15){  
  # talia esantionului  
  # x alfabetul  
  # p probabilitatile  
  lp = length(p)  
  lx = length(x)  
  
  # verify if x and p have the same size  
  if(abs(sum(p)-1)>err | sum(p>=0)!=lp){  
    stop("suma probabilitatilor nu este 1 sau probabilitatile sunt mai mici decat 0")  
  }else if(lx!=lp){  
    stop("x si p ar trebui sa aiba aceeasi marime")  
  }else{  
    out = rep(0, n)  
  
    indOrderProb = order(p, decreasing = TRUE) # index  
    pOrdered = p[indOrderProb] # rearrange the values of the probabilities  
    xOrdered = x[indOrderProb] # rearrange the values of x
```

```
# u = runif(n) # generate n uniforms
p0OrderedCS = cumsum(p0Ordered)

for (i in 1:n){
  u = runif(1)

  k = min(which(u<=p0OrderedCS))
  out[i] = x0Ordered[k]
}

return(out)
}
```

Pentru a testa această funcție să considerăm următoarele două exemple:

1. Ne propunem să generăm observații din $X \sim \begin{pmatrix} 1 & 2 & 3 \\ 0.2 & 0.3 & 0.5 \end{pmatrix}$, în acest caz: $x = [1, 2, 3]$ și $p = [0.2, 0.3, 0.5]$. Începem prin generarea a $n = 10$ observații din repartiția lui X

```
GenerateDiscrete(10, c(1,2,3), c(0.2,0.3,0.5))
[1] 3 2 2 3 1 3 3 3 1 3
```

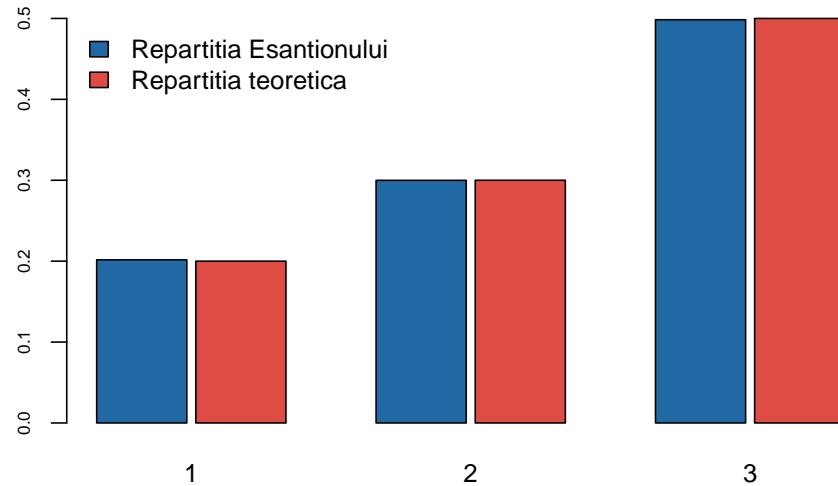
Plecând de la un eșantion de $n = 10000$ de observații vrem să comparăm, cu ajutorul *diagramei cu bare verticale* (`barplot`), repartiția eșantionului cu cea teoretică :

```
n = 10000
x = GenerateDiscrete(n, c(1,2,3), c(0.2,0.3,0.5))

# cate observatii din fiecare valoare unica a lui x
pX = table(x)/n

pT = c(0.2,0.3,0.5)
indX = c(1,2,3)

barplot(rbind(pX, pT),
        beside = T,
        space = c(0.1, 1),
        col = c(myblue, myred),
        names.arg = indX,
        cex.axis = 0.7,
        legend.text = c("Repartitia Esantionului", "Repartitia teoretica"),
        args.legend = list(x = "topleft", bty = "n"))
```



2. În acest caz considerăm variabila aleatoare $X \sim \begin{pmatrix} a & b & c & d \\ 0.15 & 0.25 & 0.15 & 0.45 \end{pmatrix}$, deci $x = [a, b, c, d]$ și $p = [0.15, 0.25, 0.15, 0.45]$. Mai jos generăm $n = 15$ observații din repartitia variabilei aleatoare X :

```
GenerateDiscrete(15, c('a','b','c','d'), c(0.15,0.25,0.15,0.45))
[1] "d" "b" "d" "b" "d" "c" "d" "d" "d" "b" "d" "a" "d" "c"
```

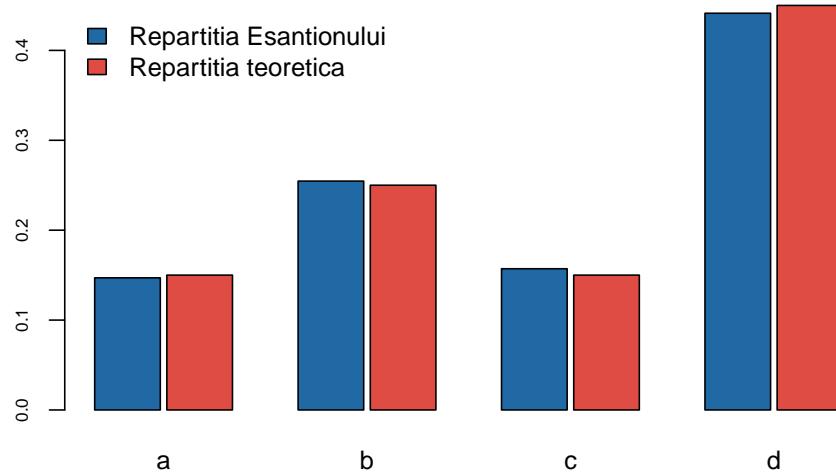
Ca și în cazul primului exemplu, vom compara repartitia teoretică cu cea a unui eșantion de $n = 10000$ de observații:

```
n = 10000
x = GenerateDiscrete(n, c('a','b','c','d'), c(0.15,0.25,0.15,0.45))

# cate observatii din fiecare valoare unica a lui x
pX = table(x)/n

pT = c(0.15,0.25,0.15,0.45)
indX = c('a','b','c','d')

barplot(rbind(pX, pT),
        beside = T,
        space = c(0.1, 1),
        col = c(myblue, myred),
        names.arg = indX,
        cex.axis = 0.7,
        legend.text = c("Repartitia Esantionului", "Repartitia teoretica"),
        args.legend = list(x = "topleft", bty = "n"))
```



Vom testa funcția și pentru repartițiile discrete de bază:

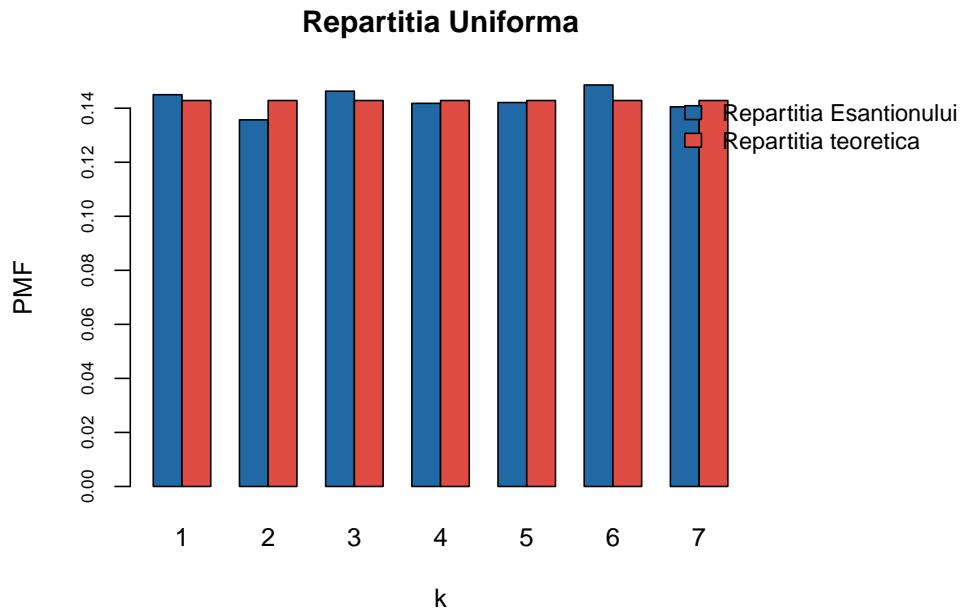
a) Uniformă

```
# Uniforma
n = 10000

pX = table(GenerateDiscrete(n, x = 1:7,
                             p = rep(1/7, 7))/n
pT = rep(1/7, 7)
indX = names(pX)

par(mar=c(5.15, 4.15, 4.15, 8.15), xpd=TRUE)

barplot(rbind(pX, pT),
        beside = T,
        # space = c(0.1, 1),
        col = c(myblue, myred),
        names.arg = indX,
        cex.axis = 0.7,
        main = "Repartitia Uniforma",
        xlab = "k",
        ylab = "PMF",
        legend.text = c("Repartitia Esantionului", "Repartitia teoretica"),
        args.legend = list(x = "topright", bty = "n",
                           inset=c(-0.35,0), cex = 0.9))
```



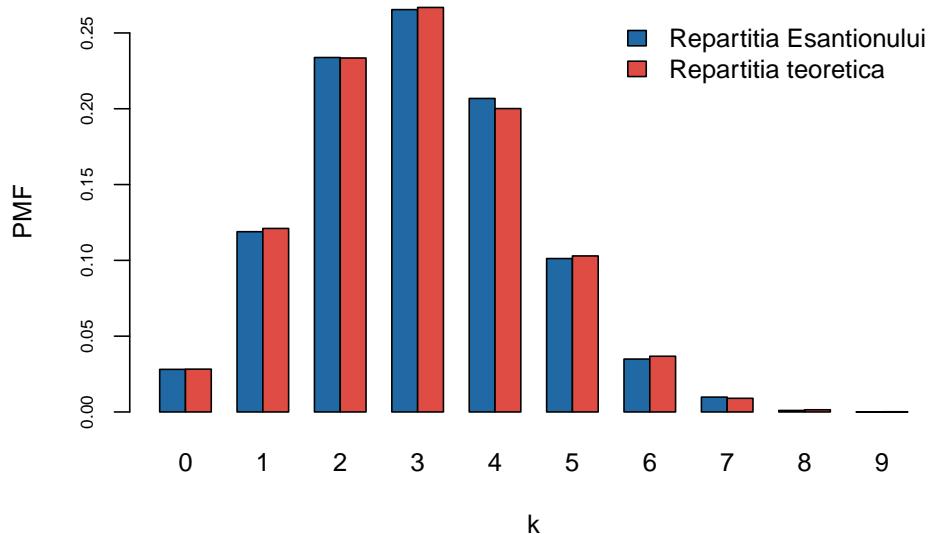
b) Binomiala

```
# Binomiala
n = 10000

pX = table(GenerateDiscrete(n, x = 0:10,
                             p = dbinom(0:10, 10, 0.3))/n
pT = dbinom(as.numeric(names(pX)), 10, 0.3)
indX = names(pX)

barplot(rbind(pX, pT),
        beside = T,
        # space = c(0.1, 1),
        col = c(myblue, myred),
        names.arg = indX,
        cex.axis = 0.7,
        main = "Repartitia Binomiala: B(10, 0.3)",
        xlab = "k",
        ylab = "PMF",
        legend.text = c("Repartitia Esantionului", "Repartitia teoretica"),
        args.legend = list(x = "topright", bty = "n"))
```

Repartitia Binomială: $B(10, 0.3)$



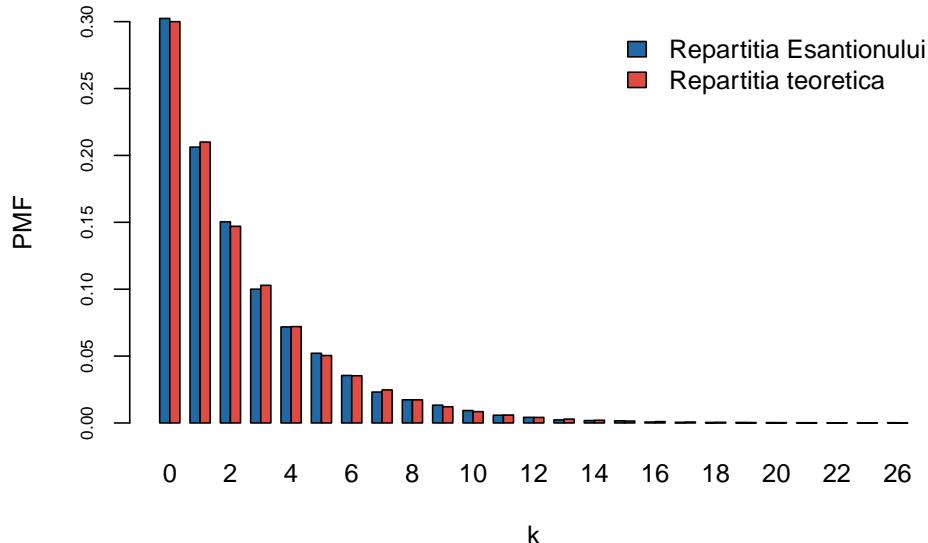
c) Geometrică

```
# Geometrica
n = 10000

pX = table(GenerateDiscrete(n, x = 0:100,
                             p = dgeom(0:100, 0.3))/n
pT = dgeom(as.numeric(names(pX)), 0.3)
indX = names(pX)

barplot(rbind(pX, pT),
        beside = T,
        # space = c(0.1, 1),
        col = c(myblue, myred),
        names.arg = indX,
        cex.axis = 0.7,
        main = "Repartitia Geometrică: Geom(0.3)",
        xlab = "k",
        ylab = "PMF",
        legend.text = c("Repartitia Esantionului", "Repartitia teoretica"),
        args.legend = list(x = "topright", bty = "n"))
```

Repartitia Geometrică: Geom(0.3)



d) Hipergeometrică

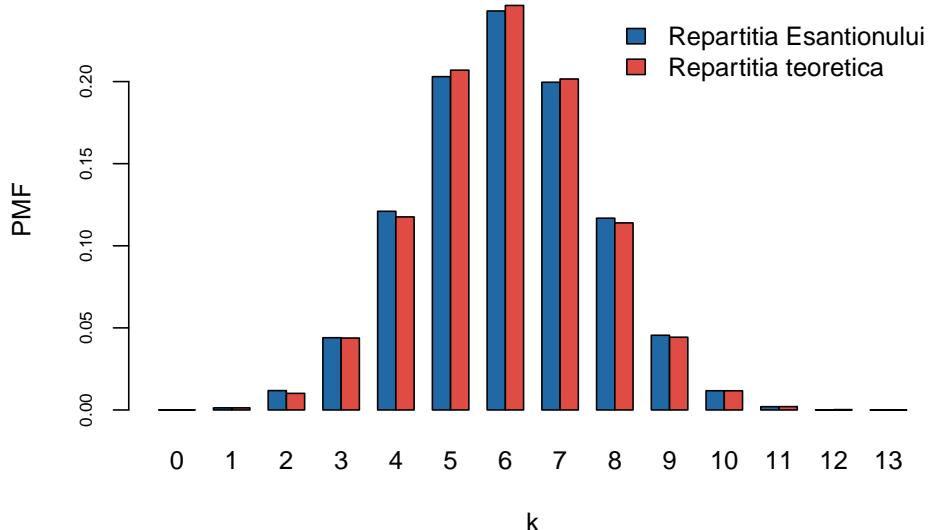
```
# Hipergeometrica
n = 10000

M = 20 # nr de bile albe
N = 30 # nr de bile negre
K = 15 # nr de bile extrase

pX = table(GenerateDiscrete(n, x = 0:15,
                             p = dhyper(0:15, M, N, K))/n
pT = dhyper(as.numeric(names(pX)), M, N, K)
indX = names(pX)

barplot(rbind(pX, pT),
        beside = T,
        # space = c(0.1, 1),
        col = c(myblue, myred),
        names.arg = indX,
        cex.axis = 0.7,
        main = "Repartitia Hipergeometrica: HG(20, 30, 15)",
        xlab = "k",
        ylab = "PMF",
        legend.text = c("Repartitia Esantionului", "Repartitia teoretica"),
        args.legend = list(x = "topright", bty = "n"))
```

Repartitia Hipergeometrica: HG(20, 30, 15)



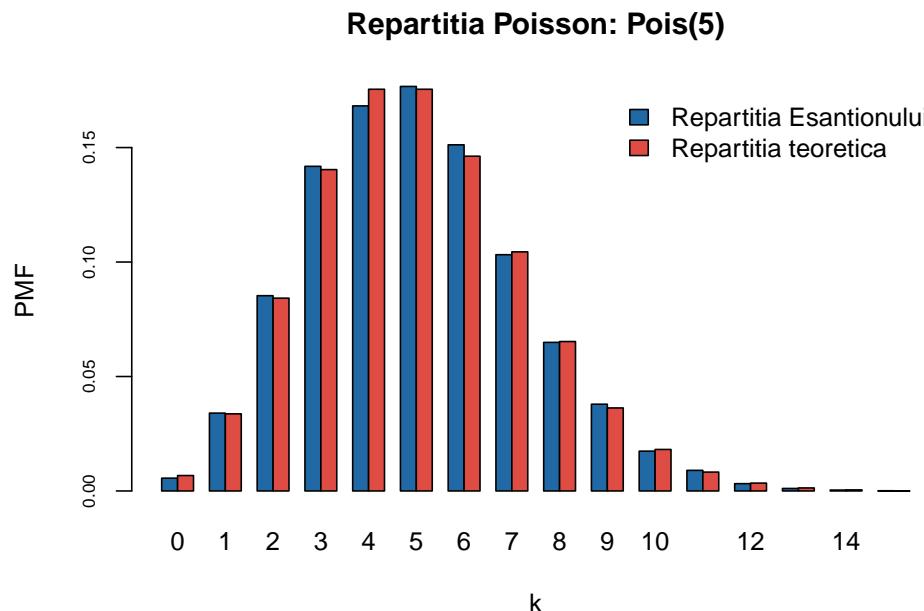
e) Poisson

```
# Poisson
n = 10000

pX = table(GenerateDiscrete(n, x = 0:50,
                             p = dpois(0:50, 5))/n

pT = dpois(as.numeric(names(pX)), 5)
indX = names(pX)

barplot(rbind(pX, pT),
        beside = T,
        # space = c(0.1, 1),
        col = c(myblue, myred),
        names.arg = indX,
        cex.axis = 0.7,
        main = "Repartitia Poisson: Pois(5)",
        xlab = "k",
        ylab = "PMF",
        legend.text = c("Repartitia Esantionului", "Repartitia teoretica"),
        args.legend = list(x = "topright", bty = "n"))
```



3 Funcția de repartitie pentru variabile aleatoare

Scrieți o funcție în R care să traseze graficul funcției de repartitie a unei distribuții date. Verificați și documentația funcției `ecdf`.

Definim următoarea funcție:

```
cdfPlot = function(dist, title, err = 1e-10){
  # dist - repartitia discreta (sau discretizata)
  lp = length(dist)

  if (abs(sum(dist)-1)>err | sum(dist)>=0)!=lp){
    stop("Eroare: vectorul de probabilitati nu formeaza o repartitie")
  }else{
    x = 0:(lp-1) # ia valori in 1:lp
    cp = cumsum(dist)

    plot(x, cp, type = "s", lty = 3,
          xlab = "x",
          ylab = "F",
          main = paste("Functia de repartitie:", title),
          ylim = c(0,1),
          col = "grey",
          bty = "n")
    abline(h = 0, lty = 2, col = "grey")
    abline(h = 1, lty = 2, col = "grey")
    for(i in 1:(lp-1)){
      lines(c(x[i], x[i+1]), c(cp[i], cp[i]),
            col = myblue,
            lwd = 2)
    }
  }
}
```

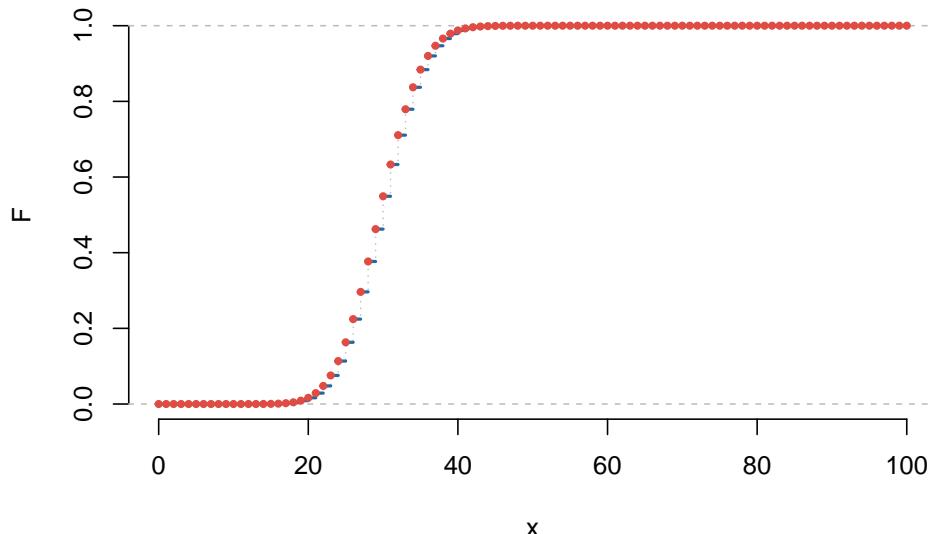
```
    }
    points(x, cp, col = myred, pch = 20, cex = 0.85)
}
}
```

Pentru a testa această funcție să considerăm repartițiile discrete:

a) Binomiala: $B(100, 0.3)$

```
cdfPlot(dist = dbinom(0:100, 100, 0.3), title = "B(100,0.3)")
```

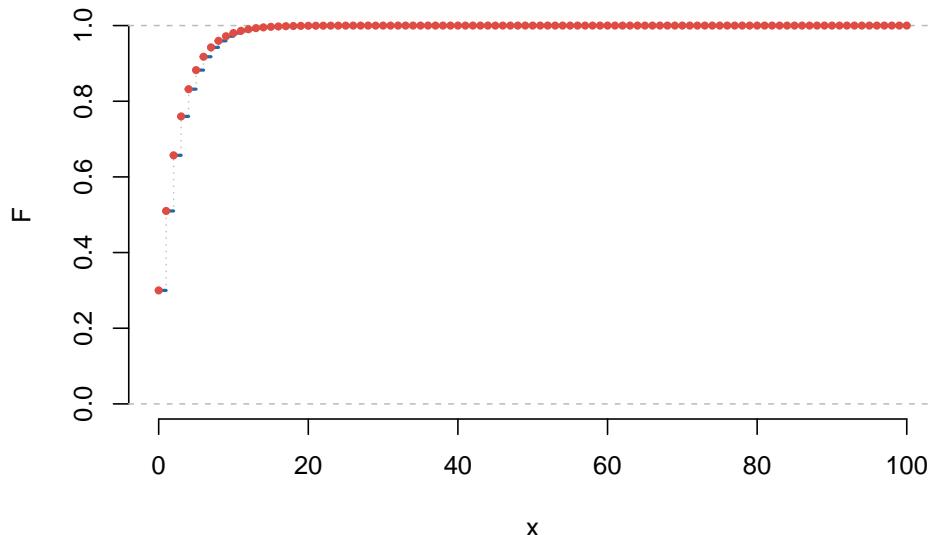
Functia de repartitie: B(100,0.3)



b) Geometrică: $Geom(0.3)$

```
cdfPlot(dist = dgeom(0:100, 0.3), title = "Geom(0.3)")
```

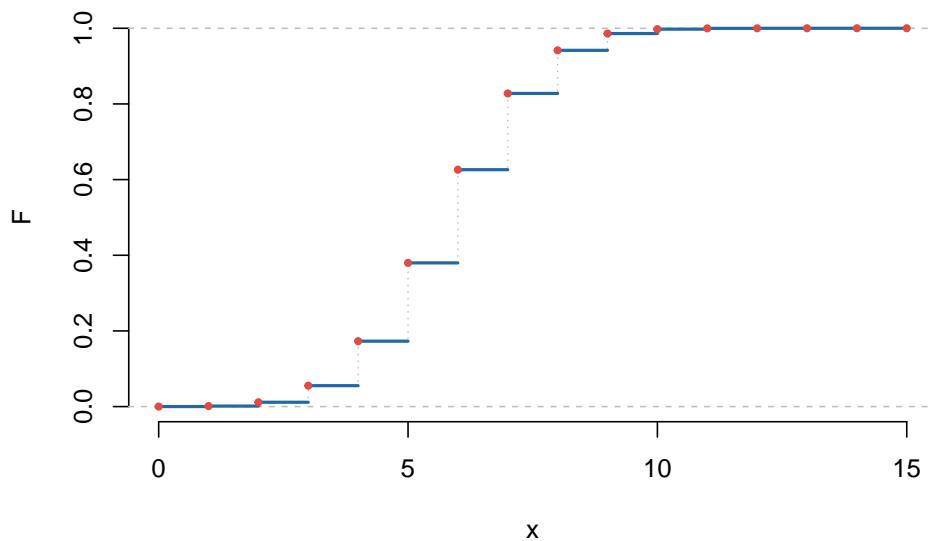
Functia de repartitie: Geom(0.3)



c) Hipergeometrică: $HG(20, 30, 15)$

```
cdfPlot(dist = dhyper(0:15, 20, 30, 15), title = "HG(20, 30, 15)")
```

Functia de repartitie: HG(20, 30, 15)

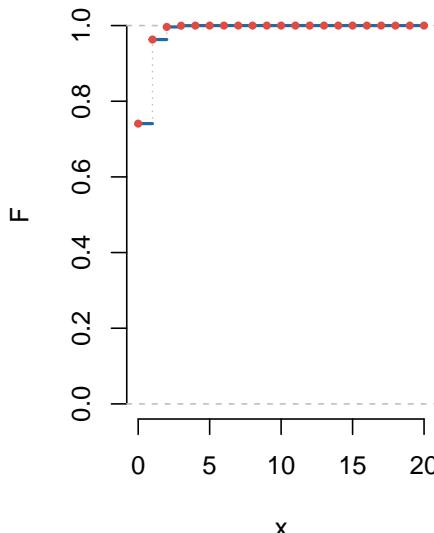


d) Poisson: $Pois(0.3)$ și $Pois(5)$

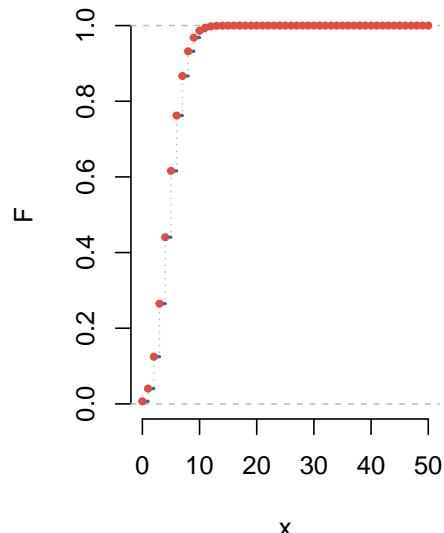
```
par(mfrow = c(1, 2))

cdfPlot(dist = dpois(0:20, 0.3), title = "Pois(0.3)")
cdfPlot(dist = dpois(0:50, 5), title = "Pois(5)")
```

Functia de repartitie: Pois(0.3)



Functia de repartitie: Pois(5)



4 Aproximarea Poisson și Normală a Binomialei



Ilustrați grafic aproximarea Poisson și normală a repartiției binomiale.

Scopul acestei probleme este de a ilustra grafic aproximarea legii binomiale cu ajutorul repartiției Poisson și a normalei.

Pentru o v.a. X repartizată binomial de parametrii n și p ($q = 1 - p$) funcția de masă este

$$f_{n,p}(k) = \mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

iar funcția de repartitie este

$$F_{n,p}(k) = \mathbb{P}(X \leq k) = \sum_{x=0}^k \binom{n}{x} p^x (1-p)^{n-x}.$$

4.1 Aproximarea Poisson

Dacă $n \rightarrow \infty$ (n este mare) și $p \rightarrow 0$ (p este mic, evenimentele sunt rare) aşa încât $np \rightarrow \lambda$ atunci se poate verifica cu usurință că

$$f_{n,p}(k) \approx f_\lambda(k) = e^{-\lambda} \frac{\lambda^k}{k!}.$$

Mai exact, avem că dacă k este mic în comparație cu n atunci

$$\begin{aligned} \binom{n}{k} p^k &= \frac{n(n-1)\cdots(n-k+1)}{k!} \left(\frac{\lambda}{n}\right)^k \\ &= 1 \times \left(1 - \frac{1}{n}\right) \times \cdots \times \left(1 - \frac{k-1}{n}\right) \frac{\lambda^k}{k!} \\ &\approx \frac{\lambda^k}{k!} \end{aligned}$$

și

$$\log(1-p)^{n-k} = (n-k) \log\left(1 - \frac{\lambda}{n}\right) \approx n \left(-\frac{\lambda}{n}\right)$$

ceea ce conduce la $(1-p)^{n-k} \approx e^{-\lambda}$. Combinând cele două aproximări obținem

$$\binom{n}{k} p^k (1-p)^{n-k} \approx \frac{\lambda^k}{k!} e^{-\lambda}.$$

Pentru a ilustra acuratețea acestei aproximări vom folosi instrucțiunile R `dbinom` și `dpois` care permit calcularea funcțiilor de masă $f_{n,p}(k)$ și $f_\lambda(k)$.

```
AppBP = function(n,p,a,b){
  lambda = n*p
  x = matrix(numeric((b-a+1)*3),ncol=3,
             dimnames = list(a:b,c("Binomiala","Poisson","Eroarea Absoluta")))
  x[,1] = dbinom(a:b,n,p)
  x[,2] = dpois(a:b,lambda)
  x[,3] = abs(x[,1]-x[,2])
  error = max(abs(x[,3]))

  return(list(x = as.data.frame(x), error = error, param = c(n, p, lambda)))
}
```

Functie care ilustreaza aproximarea Binomial vs. Poisson

```
pl = function(n,p,a,b){
  clr = c(myblue, myred)# culori
  lambda = n*p
  mx = max(dbinom(a:b,n,p))

  plot(c(a:b,a:b), c(dbinom(a:b,n,p), dpois(a:b,lambda)), type="n",
       main = paste("Approx. Poisson pentru binomiala\n n=",
                   n, ", p = ", p, ", lambda = ",lambda),
       ylab = "Probabilitatea", xlab="x",
       bty = "n")

  points((a:b)-.15, dbinom(a:b,n,p), type = "h",
         col = clr[1], lwd = 8)
  points((a:b)+.15, dpois(a:b,lambda), type = "h",
         col = clr[2], lwd = 8)

  legend(b-b/2, mx, legend = c(paste0("Binomiala(",n,",",p,")"),
                                "Poisson"))}
```

```

        paste0("Poisson(",lambda,")"),
fill = clr, bg="white",
bty = "n")
}

```

Pentru setul de parametrii $n = 10$ și $p = 0.1$ avem următorul tabel și următoarea figură

Tab. 2: Aproximarea Poisson la binomiala $n = 10$ $p = 0.1$ $\lambda = 1$. Eroarea (Diferența în valoare absolută maximă) = 0.01954 .

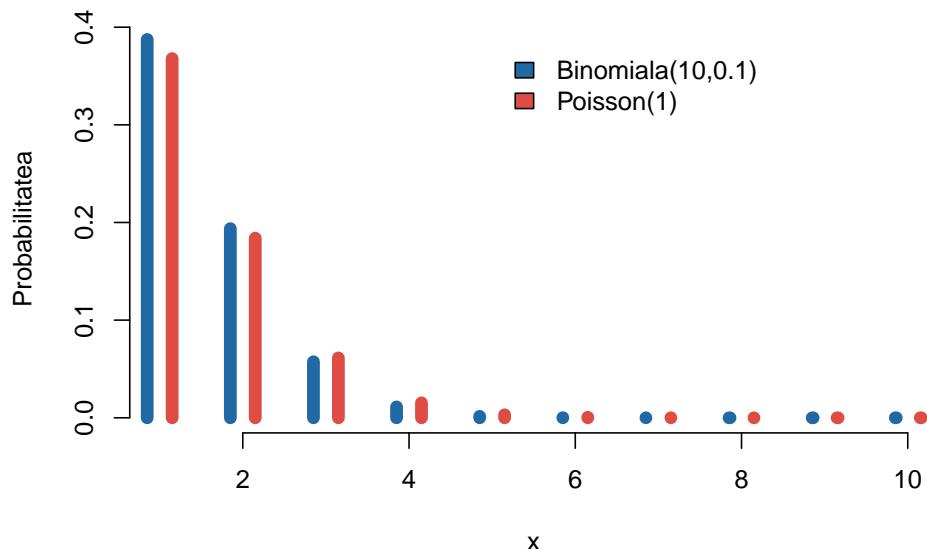
k	Binomiala	Poisson	Eroarea Absolută
1	0.3874205	0.3678794	0.0195410
2	0.1937102	0.1839397	0.0097705
3	0.0573956	0.0613132	0.0039176
4	0.0111603	0.0153283	0.0041680
5	0.0014880	0.0030657	0.0015776
6	0.0001378	0.0005109	0.0003732
7	0.0000087	0.0000730	0.0000642
8	0.0000004	0.0000091	0.0000088
9	0.0000000	0.0000010	0.0000010
10	0.0000000	0.0000001	0.0000001

```

# pentru n = 10, p = 0.1
pl(10,.1,1,10)

```

Approx. Poisson pentru binomiala $n = 10$, $p = 0.1$, $\lambda = 1$



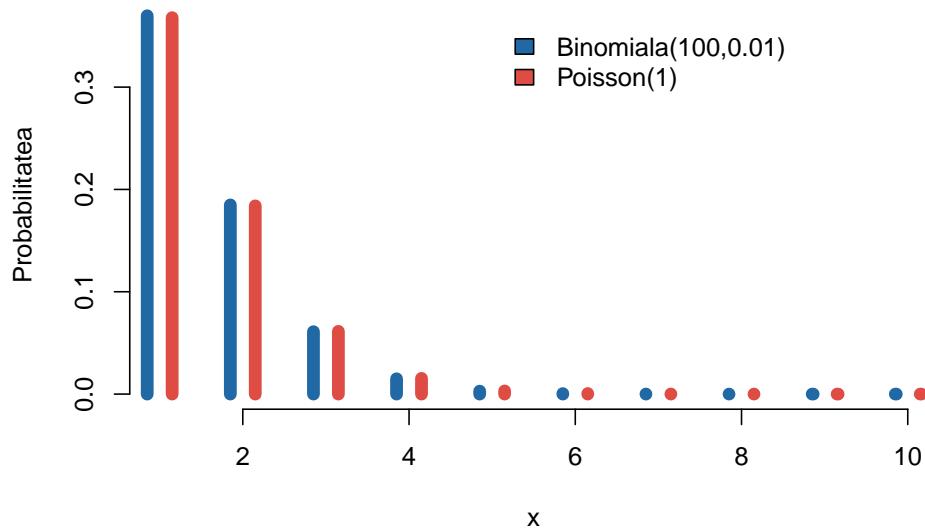
iar pentru parametrii $n = 100$ și $p = 0.01$ obținem

Tab. 3: Aproximarea Poisson la binomiala $n = 100$ $p = 0.01$ lambda $= 1$. Eroarea (Diferenta in valoare absoluta maxima) = 0.00185 .

k	Binomiala	Poisson	Eroarea Absoluta
1	0.3697296	0.3678794	0.0018502
2	0.1848648	0.1839397	0.0009251
3	0.0609992	0.0613132	0.0003141
4	0.0149417	0.0153283	0.0003866
5	0.0028978	0.0030657	0.0001679
6	0.0004635	0.0005109	0.0000475
7	0.0000629	0.0000730	0.0000101
8	0.0000074	0.0000091	0.0000017
9	0.0000008	0.0000010	0.0000003
10	0.0000001	0.0000001	0.0000000

```
# pentru n = 100, p = 0.01
pl(100,.01,1,10)
```

Approx. Poisson pentru binomiala
n= 100 , p = 0.01 , lambda = 1

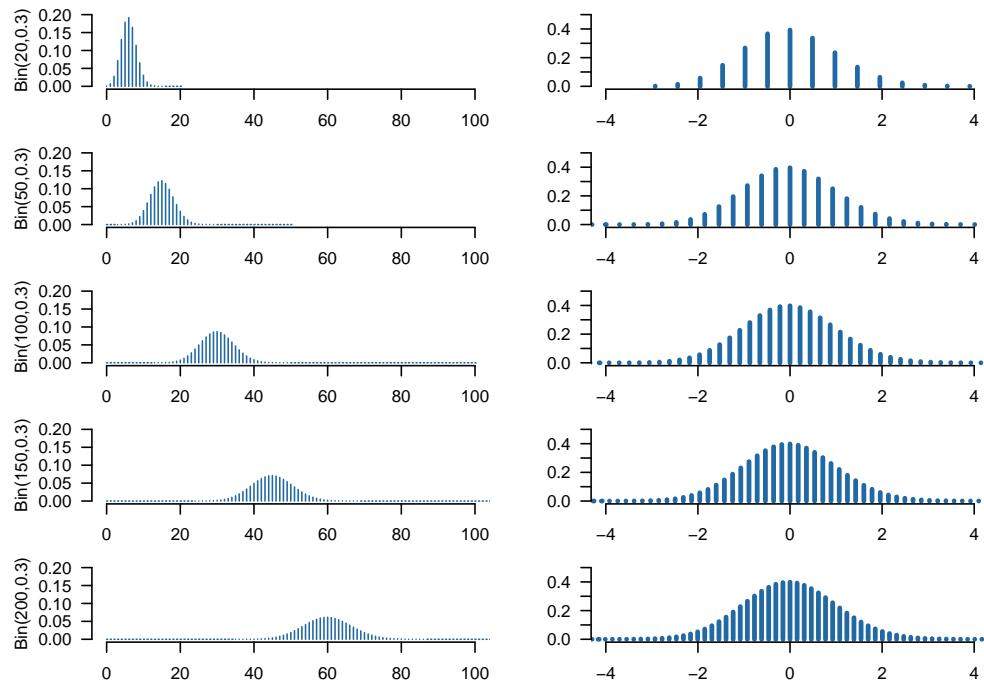


Pentru funcția de repartiție $F_{n,p}(k)$, folosind aproximarea Poisson avem că

$$F_{n,p}(k) \approx F_\lambda(k) = \sum_{x=0}^k e^{-\lambda} \frac{\lambda^x}{x!}.$$

4.2 Aproximarea Normală

Să considerăm repartiția binomială $\mathcal{B}(n,p)$ pentru $p = 0.3$ și $n \in \{20, 50, 100, 150, 200\}$ și să trasăm histogramele variabilelor aleatoare care au aceste repartiții (X_n) precum și a variabilelor standardizate $Z_n = \frac{X_n - np}{\sqrt{npq}}$.



Observăm, pentru graficele din partea stângă, că valoarea maximă se atinge în jurul punctului $n \times 0.3$ pentru fiecare grafic în parte. De asemenea se observă că odată cu creșterea lui n crește și gradul de împrăștiere, cu alte cuvinte crește și abaterea standard ($\sigma_n = \sqrt{npq}$).

Pe de altă parte putem remarca că figurile din partea dreaptă au o formă simetrică, de tip *clopot*, concentrate în jurul lui 0, fiind translate în origine și scalate pentru a avea o varianță egală cu 1. Abraham de Moivre¹ a justificat acest efect (pentru $p = 0.5$) încă din 1756 observând că raportul

$$\frac{f_{n,p}(k)}{f_{n,p}(k-1)} = \frac{\frac{n!}{k!(n-k)!} p^k q^{n-k}}{\frac{n!}{(k-1)!(n-k+1)!} p^{k+1} q^{n-k+1}} = \frac{(n-k+1)p}{kq}$$

pentru $k = 1, 2, \dots, n$. Astfel $f_{n,p}(k) \geq f_{n,p}(k-1)$ dacă și numai dacă $(n+1)p \geq k$ de unde, pentru n fixat, deducem că $f_{n,p}(k)$ atinge valoarea maximă pentru $k_{\max} = \lfloor (n+1)p \rfloor \approx np$ (acesta este motivul pentru care fiecare grafic din partea stângă are vârful în jurul punctului np).

Să observăm ce se întâmplă în jurul lui k_{\max} . Avem

$$\frac{f_{n,p}(k_{\max} + i)}{f_{n,p}(k_{\max} + i - 1)} = \frac{(n - k_{\max} - i + 1)p}{(k_{\max} + i)q} \approx \frac{(np - i)p}{(np + i)q} = \frac{1 - \frac{i}{np}}{1 + \frac{i}{np}}$$

și cum (folosind relația $\log(1 + x) \approx x$, pentru x în jurul lui 0)

$$\log\left(1 - \frac{i}{np}\right) - \log\left(1 + \frac{i}{np}\right) \approx -\frac{i}{np} - \frac{i}{np} = -\frac{2i}{np}$$

deducem, pentru $m \geq 1$ și $k_{\max} + m \leq n$, că

¹de Moivre, A. (1756). *The Doctrine of Chances: or, A Method of Calculating the Probabilities of Events in Play* (Third ed.). New York: Chelsea.

$$\begin{aligned} \log \frac{f_{n,p}(k_{\max} + m)}{f_{n,p}(k_{\max})} &= \log \left(\frac{f_{n,p}(k_{\max} + 1)}{f_{n,p}(k_{\max})} \times \frac{f_{n,p}(k_{\max} + 2)}{f_{n,p}(k_{\max} + 1)} \times \cdots \times \frac{f_{n,p}(k_{\max} + m)}{f_{n,p}(k_{\max} + m - 1)} \right) \\ &= \log \frac{f_{n,p}(k_{\max} + 1)}{f_{n,p}(k_{\max})} + \log \frac{f_{n,p}(k_{\max} + 2)}{f_{n,p}(k_{\max} + 1)} + \cdots + \log \frac{f_{n,p}(k_{\max} + m)}{f_{n,p}(k_{\max} + m - 1)} \\ &\approx \frac{-1 - 2 - \cdots - m}{npq} = -\frac{1}{2} \frac{m^2}{npq}. \end{aligned}$$

Sumarizând avem, pentru m nu foarte mare,

$$\mathbb{P}(X = k_{\max} + m) \approx f_{n,p}(k_{\max}) e^{-\frac{1}{2} \frac{m^2}{npq}}.$$

Folosind formula lui Stirling²

$$n! \approx \sqrt{2\pi n}^{n+\frac{1}{2}} e^{-n}$$

pentru $k = k_{\max} \approx np$, avem

$$f_{n,p}(k) \approx \frac{1}{\sqrt{2\pi}} \frac{n^{n+\frac{1}{2}}}{(np)^{np+\frac{1}{2}} (nq)^{nq+\frac{1}{2}}} p^{np} q^{nq} = \frac{1}{\sqrt{2\pi npq}}.$$

Astfel aproximarea de Moivre devine

$$\mathbb{P}(X = k_{\max} + m) \approx \frac{1}{\sqrt{2\pi npq}} e^{-\frac{1}{2} \frac{m^2}{npq}}$$

și scriind k pentru $k_{\max} + m$ și înlocuind k_{\max} cu np obținem

$$\mathbb{P}(X = k) \approx \frac{1}{\sqrt{2\pi npq}} e^{-\frac{1}{2} \frac{(k-np)^2}{npq}} = \frac{1}{\sigma_n \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{k-np}{\sigma_n}\right)^2}.$$

Astfel $\mathbb{P}(X = k)$ este aproximativ egală cu aria de sub curba

$$f(x) = \frac{1}{\sigma_n \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{x-np}{\sigma_n}\right)^2}$$

pe intervalul $k - \frac{1}{2} \leq x \leq k + \frac{1}{2}$.

În mod similar, pentru $0 \leq a < b \leq n$, avem

$$\mathbb{P}(a \leq X \leq b) = \sum_{k=a}^b f_{n,p}(k) \approx \sum_{k=a}^b \int_{k+\frac{1}{2}}^{k-\frac{1}{2}} f(x) dx = \int_a^b f(x) dx$$

de unde prin schimbarea de variabilă $y = \frac{x-np}{\sigma_n}$ obținem

$$\mathbb{P}(a \leq X \leq b) \approx \frac{1}{\sqrt{2\pi}} \int_{\alpha}^{\beta} e^{-\frac{y^2}{2}} dy = \Phi(\beta) - \Phi(\alpha)$$

²A se vedea cartea lui Feller, W. (1968). *An Introduction to Probability Theory and Its Applications* (third ed.), Volume 1. New York: Wiley. pag. 52-53 pentru o derivare a formulei lui Stirling.

unde $\alpha = \frac{a-np-\frac{1}{2}}{\sigma_n}$, $\beta = \frac{b-np+\frac{1}{2}}{\sigma_n}$ și $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{y^2}{2}} dy$.

Aplicând rezultatele de mai sus, în cele ce urmează vom considera două aproximări pentru funcția de repartiție $F_{n,p}(k)$:

a) aproximarea normală

$$F_{n,p}(k) \approx \Phi\left(\frac{k - np}{\sqrt{np(1-p)}}\right).$$

b) aproximarea normală cu coeficient de corecție de continuitate

$$F_{n,p}(k) \approx \Phi\left(\frac{k + 0.5 - np}{\sqrt{np(1-p)}}\right).$$

În practică această ultimă aproximare se aplică atunci când atât $np \geq 5$ cât și $n(1-p) \geq 5$.

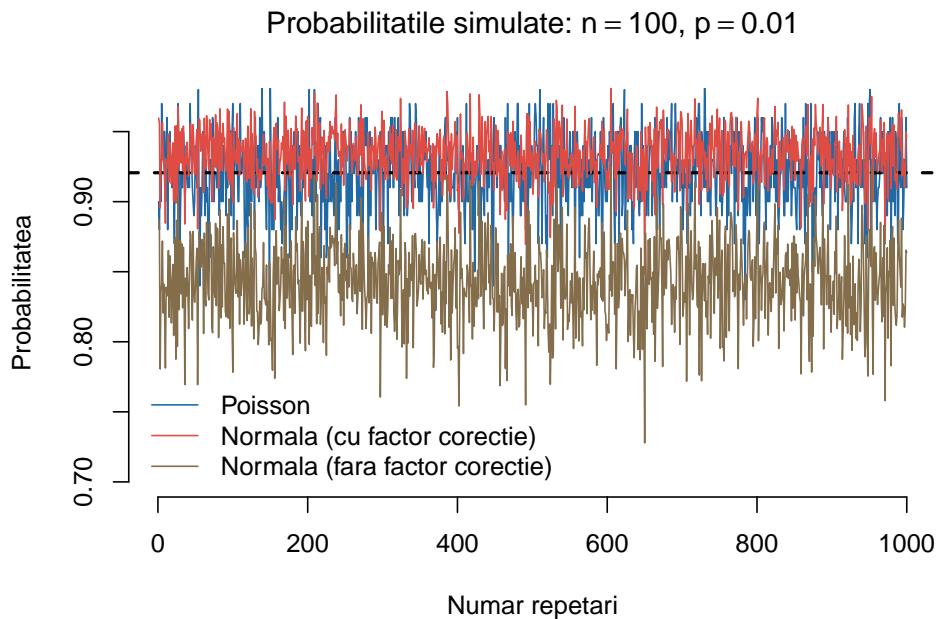
Următorul cod crează o funcție care calculează cele trei aproximări pentru funcția de repartiție binomială

```
appBNP = function(n, p, R = 1000, k = 6) {
  trueval = pbinom(k, n, p) # adevarata valoare a functiei de repartitie in k
  prob.zcc <- prob.zncc <- prob.pois <- NULL # initializare
  q = 1-p
  for (i in 1:R) {# repetam procesul de R ori
    x = rnorm(n, n * p, sqrt(n * p * q)) # generare n v.a. normale de medie np
    z.cc = ((k + .5) - mean(x))/sd(x) # cu coeficient de corectie
    prob.zcc[i] = pnorm(z.cc)
    z.ncc = (k - mean(x))/sd(x) # fara coeficient de corectie
    prob.zncc[i] = pnorm(z.ncc)
    y = rpois(n, n * p)
    prob.pois[i] = length(y[y <= k])/n # approximate Poisson
  }
  list(prob.zcc = prob.zcc, prob.zncc = prob.zncc,
       prob.pois = prob.pois, trueval = trueval)
}
```

Avem următoarea ilustrație grafică a diferitelor metode de aproximare

```
# Plot
R = 1000
set.seed(10)
out = appBNP(n = 100, p = .01, k = 2, R = 1000)

plot(1:R, out$prob.pois, type = "l", col = myblue, xlab = "Numar repetari",
      main = expression(paste("Probabilitatile simulate: ",
                             n==100, ", ", p==0.01, sep="")),
      ylab = "Probabilitatea", ylim = c(.7, .97),
      bty = "n")
abline(h = out$trueval, col="black", lty=2, lwd=2)
lines(1:R, out$prob.zcc, lty = 1, col = myred)
lines(1:R, out$prob.zncc, lty = 1, col = mybrown)
legend("bottomleft", c("Poisson", "Normala (cu factor corectie)",
                       "Normala (fara factor corectie)"),
       lty = c(1), col = c(myblue, myred, mybrown),
       bty = "n")
```

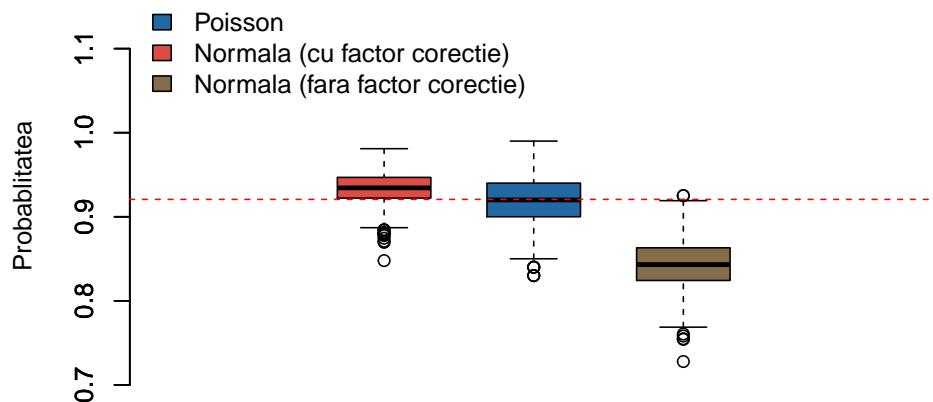


Avem și următorul boxplot (discuție ce reprezintă un boxplot) care ne permite să evidențiem care dintre aproximări este mai bună pentru valorile selectate

```
# n = 200
set.seed(10)
out = appBNP(n = 100, p = .01, k = 2, R = 1000)

par(bty = "n")
boxplot(out$prob.pois, boxwex = 0.25, xlim = c(0.5, 1.5),
        ylim = c(0.7, 1.15),
        col = myblue,
        main = expression(paste("Aproximarea Binomialei: ",
                               n==100, ", ", p==0.01, sep="")),
        ylab = "Probabilitatea",
        ylim = c(out$trueval - 0.1, out$trueval + 0.15),
        bty = "n")
boxplot(out$prob.zcc, boxwex = 0.25, at = 1:1 - 0.2, add = T,
        col = myred)
boxplot(out$prob.zncc, boxwex = 0.25, at = 1:1 + 0.2, add = T,
        col = mybrown)
abline(h = out$trueval, col = "red", lty=2)
legend("topleft", c("Poisson", "Normala (cu factor corectie)",
                    "Normala (fara factor corectie)"),
       fill = c(myblue, myred, mybrown),
       bty = "n")
```

Aproximarea Binomialei: $n = 100$, $p = 0.01$



Laborator 5

Problema colecționarului de cupoane și algoritmul Quicksort

Obiectivul acestui laborator este de prezenta problema Colecționarului de cupoane și algoritmul randomizat QuickSort și de a determina timpul mediu de execuție a acestuia.

1 Problema colecționarului de cupoane



Să presupunem că fiecare cutie de cereale conține unul dintre cele n cupoane diferite existente. Odată ce o persoană a colecționat toate cele n cupoane poate să le trimită pentru a revendica un premiu. De asemenea, presupunem că fiecare cupon este ales uniform și independent din cele n posibilități existente și colecționarul nu colaborează cu alte persoane pentru a completa colecția. Întrebarea care se pune este câte cutii de cereale trebuie cumpărate, în medie, pentru a obține cel puțin unul din fiecare cupon?

Pentru a rezolva această problemă, să notăm cu X variabila aleatoare care descrie numărul de cutii de cereale cumpărate până când obținem toate cupoanele. Vrem să determinăm $\mathbb{E}[X]$.

Să notăm cu X_i numărul suplimentar de cutii de cereale pe care trebuie să le cumpărăm atunci când avem $i - 1$ cupoane colecționate pentru a avea i cupoane diferite, $i \geq 1$. Astfel putem scrie

$$X = \sum_{i=1}^n X_i,$$

de unde $\mathbb{E}[X] = \sum_{i=1}^n \mathbb{E}[X_i]$. Rămâne să determinăm $\mathbb{E}[X_i]$ pentru $i \in \{1, 2, \dots, n\}$.

Să remarcăm faptul că atunci când avem colectate $i - 1$ cupoane ne mai rămân $n - i + 1$ cupoane de colecționat, prin urmare probabilitatea de a alege un nou cupon este $p_i = \frac{n-i+1}{n}$ și cum $X_i \sim \text{Geom}(p_i)$ deducem că $\mathbb{E}[X_i] = \frac{1}{p_i}$.

Avem

$$\mathbb{E}[X] = \sum_{i=1}^n \mathbb{E}[X_i] = \sum_{i=1}^n \frac{1}{p_i} = \sum_{i=1}^n \frac{n}{n-i+1} = n \sum_{i=1}^n \frac{1}{i} = nH_n$$

ceea ce implica $\mathbb{E}[X] = n(\log(n) + \mathcal{O}(1))$ deoarece $H_n = \log(n) + O(1)^1$.

Următorul cod R simulează problema colecționarului de cupoane:

```
simcollect = function(n) {  
  
  coupons = 1:n # multimea cupoanelor  
  collect = numeric(n)  
  
  nums = 0
```

¹A se vedea pagina de wikipedia [armonic_number](#)

```
while (sum(collect)<n)
{
  # extragere cu intoarcere
  i = sample(coupons, 1)
  collect[i] = 1
  nums = nums + 1
}
return(nums)
```

Pentru calculul mediei vom considera $n = 20$ și vom repeta procedeul $N = 10000$ de ori. Vom compara rezultatul empiric cu cel teoretic:

```
## Calculul mediei
n = 20
Nrep = 10000
simlist = replicate(Nrep, simcollect(n))

# media empirica
mean(simlist)
[1] 71.8905

# media teoreтика
n*sum(1/(1:n))
[1] 71.95479
```

Pentru calculul varianței avem că $Var(X_i) = \frac{1-p_i}{p_i^2}$ și cum X_i sunt independente rezultă că

$$Var(X) = \sum_{i=1}^n Var(X_i) = \sum_{i=1}^n \frac{1-p_i}{p_i^2} = n \sum_{i=1}^n \frac{i-1}{(n-i+1)^2}.$$

Pentru compararea rezultatului empiric cu cel teoretic avem:

```
# varianta empirica
var(simlist)
[1] 559.212

# varianta teoreтика
n*sum((0:(n-1))/((n:1)^2))
[1] 566.5105
```

O aplicație a problemei colecționarului de cupoane este următoarea: să presupunem că într-un parc național din India, o cameră video automată fotografiază n tigrii care trec prin dreptul ei pe parcursul unui an și analizând fotografiile se constată că t dintre aceștia sunt diferenți. Ne propunem să estimăm (aproximăm) numărul total de tigrii din parc. În contextul problemei colecționarului de cupoane, presupunem că avem n cupoane diferențiate și că am cumpărat t cutii de cereale și ne întrebăm care este numărul mediu de cupoane distincte din cele t .

Fie Y variabila aleatoare care ne dă numărul de cupoane distincte după t cutii cumpărate. Atunci putem scrie

$$Y = I_1 + I_2 + \cdots + I_n$$

unde pentru $j \in \{1, 2, \dots, n\}$

$$I_j = \begin{cases} 1, & \text{cuponul } j \text{ se află printre cele } t \\ 0, & \text{altfel} \end{cases}$$

Astfel,

$$\mathbb{E}[I_j] = \mathbb{P}(\text{cuponul } j \text{ se află printre cele } t) = 1 - \mathbb{P}(\text{cuponul } j \text{ nu se află printre cele } t) = 1 - \left(1 - \frac{1}{n}\right)^t$$

ceea ce conduce la

$$\mathbb{E}[Y] = \sum_{j=1}^n \mathbb{E}[I_j] = n \left[1 - \left(1 - \frac{1}{n}\right)^t \right].$$

Pentru a determina numărul de tigrii din parc trebuie să găsim valoarea lui n știind numărul t de tigrii fotografiati de camera automată și de numărul mediu de tigrii distincți d determinați manual, deci trebuie să rezolvăm ecuația:

$$d \approx \mathbb{E}[Y] = n \left[1 - \left(1 - \frac{1}{n}\right)^t \right]$$

Ecuația nu are o soluție explicită dar poate fi determinată numeric folosind funcția `uniroot` care permite găsirea unei rădăcini într-un interval dat a unei funcții reale:

```
f = function(n, t = 100, d = 50){  
  n*(1 - (1 - 1/n)^t) - d  
}  
  
# pentru 100 de tigrii fotografiati dintre care 50 diferiti  
ans = uniroot(f, c(50, 200), t = 100, d = 50)  
ans$root  
[1] 62.40844
```

Pentru $t = 100$ și $d = 50$ estimăm în jur de 62 tigrii în parc.

2 Timpul mediu de execuție al algoritmului Quicksort

În practică, algoritmul *Quicksort* este unul din cei mai rapizi și mai populari algoritmi de sortare. Unul dintre motivele acestui fapt este că algoritmul nu necesită memorie de stocare suplimentară. Cu toate acestea, algoritmul *Quicksort* este un algoritm destul de slab atunci când luăm în calcul scenariile teoretice de tipul *cel mai rău caz*. Vom vedea, în cele ce urmează, că versiunea randomizată a acestui algoritm are, în medie, o performanță foarte bună.

Algoritmul primește ca input o listă $S = \{x_1, \dots, x_n\}$ de n numere care, pentru ușurință, vor fi presupuse diferite. Pseudocodul algoritmului este:



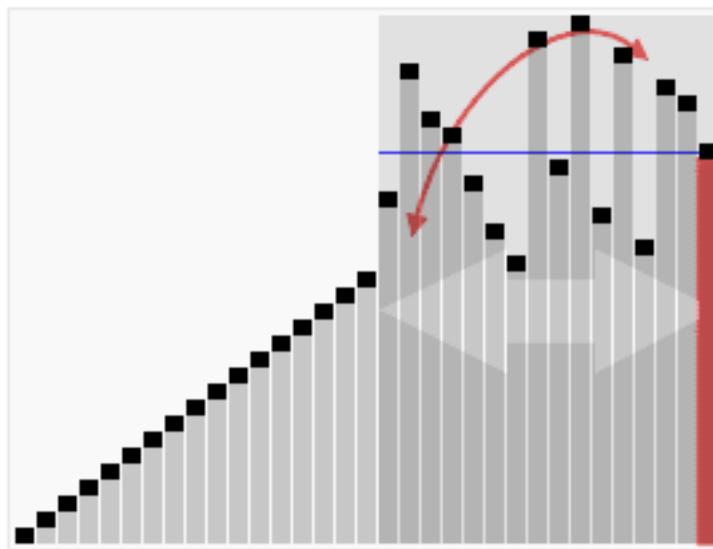
Input: O listă $S = \{x_1, \dots, x_n\}$ de n elemente distincte pe o mulțime total ordonată.

Output: Elementele listei S sortate crescător.

1. Dacă S nu are niciun element sau are un element atunci întoarce S . Altfel continuă.
2. Alege un element din S ca pivot; să-l numim x

3. Compara toate elementele din S cu x pentru a împărți lista în două subliste: a) S_1 conține toate elementele din S mai mici decât x . b) S_2 conține toate elementele din S mai mari decât x .
4. Folosește recursiv Quicksort pentru a sorta crescător sublistele S_1 și S_2 .
5. Întoarce lista $\{S_1, x, S_2\}$

Următoarea animație este ilustrativă (doar în versiunea HTML):



Să observăm că există situații (de tipul *cel mai rău caz*) în care algoritmul Quicksort necesită $\Omega(n)^2$ operații de comparare. De exemplu să presupunem că lista de input este $S = \{x_1 = n, x_2 = n - 1, \dots, x_n = 1\}$ și să presupunem că pentru alegerea pivotului adoptăm regula ca acesta să fie primul element din listă. Prin urmare primul pivot ales este n și algoritmul necesită $n - 1$ comparații. În urma diviziunii, rezultă două subliste, una de lungime 0 (care nu necesită nicio operație suplimentară) și una de lungime $n - 1$ (ce elementele $n - 1, n - 2, \dots, 1$). La pasul doi, următorul pivot ales este $n - 1$ iar algoritmul necesită $n - 2$ comparații și întoarce sublista cu elemente $n - 2, \dots, 1$. Continuând procedeul deducem că algoritmul Quicksort efectuează

$$(n - 1) + (n - 2) + \dots + 1 = \frac{n(n - 1)}{2} \text{ operații.}$$

Din exemplul de mai sus, este clar că alegerea pivotului influențează puternic numărul de operații pe care le efectuează algoritmul. O alegere mai bună a pivotului ar consta în determinarea unui element, la fiecare pas, care să împartă lista în două subliste cam de aceeași mărime ($\lceil n/2 \rceil$ elemente).

Întrebarea care se pune este cum putem garanta că algoritmul alege un pivot bun suficient de des? O modalitate ar fi să alegem pivotul aleator, de manieră uniformă între elementele disponibile. Această abordare face ca algoritmul Quicksort să devină randomizat.



Să presupunem că ori de câte ori un pivot este ales pentru algoritmul *Quicksort randomizat*, acesta este ales independent și uniform din mulțimea elementelor posibile. Arătați că numărul mediu de comparații ale algoritmului este de $2n \log(n) + O(n)$. Scrieți o funcție care implementează algoritmul *Quicksort randomizat* cu pivot ales uniform.

²A se vedea pagina de wikipedia [Big_O_notation](#)

Fie y_1, \dots, y_n elementele x_1, \dots, x_n ordonate crescător. Pentru $i < j$, fie X_{ij} variabila aleatoare care ia valoarea 1 dacă elementele y_i și y_j au fost comparate pe parcursul rulării algoritmului și valoarea 0 altfel. Atunci numărul total de comparații X satisface relația

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}$$

și din proprietatea de liniaritate a mediei

$$\mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \mathbb{E}[X_{ij}].$$

Cum X_{ij} este o variabilă aleatoare de tip Bernoulli care ia doar valoarea 0 și 1, $\mathbb{E}[X_{ij}] = \mathbb{P}(X_{ij} = 1)$ prin urmare trebuie să determinăm probabilitatea ca elementele y_i și y_j să fie comparate pe parcursul algoritmului. Să observăm că elementele y_i și y_j sunt comparate dacă și numai dacă oricare dintre cele două elemente sunt alese ca pivot din mulțimea $A_{ij} = \{y_i, y_{i+1}, \dots, y_j\}$. Acest lucru se datorează faptului că dacă y_i (sau y_j) a fost primul pivot ales din mulțimea A_{ij} atunci elementele y_i și y_j rămân în aceeași sublistă, deci vor fi comparate ulterior. În mod similar, dacă niciunul din elementele y_i și y_j nu este primul pivot ales din mulțimea A_{ij} atunci cele două elemente vor face parte din subliste separate și nu vor mai fi comparate.

Cum pivoții sunt aleși de manieră independentă și uniform din fiecare sublistă de elemente, prima dată când un pivot este ales din mulțimea A_{ij} acesta are aceeași șansă să fie oricare element. Prin urmare probabilitatea ca y_i sau y_j să fie primul pivot ales este $\frac{2}{j-i+1}$. Astfel obținem

$$\begin{aligned} \mathbb{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \mathbb{E}[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} = \sum_{k=2}^n \sum_{i=1}^{n+1-k} \frac{2}{k} \\ &= \sum_{k=2}^n (n+1-k) \frac{2}{k} = (2n+2) \sum_{k=1}^n \frac{1}{k} - 4n. \end{aligned}$$

Prin urmare $\mathbb{E}[X] = 2(n+1)H_n - 4n = 2n \log(n) + O(n)$ (am folosit faptul că $H_n = \log(n) + O(1)$).

Următorul cod implementează algoritmul Quicksort randomizat:

```
quickSort <- function(vect) {
  # Args:
  #   vect: Vector numeric

  # daca lungimea este <= 1 stop
  if (length(vect) <= 1) {
    return(vect)
  }

  # alege pivotul
  ide = sample(1:length(vect), 1)
  element = vect[ide]
  partition = vect[-ide]

  # Imparte elementele in doua subliste (< pivot si >= pivot)
}
```

```
v1 = partition[partition < element]
v2 = partition[partition >= element]

# Aplica recursiv algoritmul
v1 = quickSort(v1)
v2 = quickSort(v2)
return(c(v1, element, v2))
}

n = 25
S = sample(1:n, n, replace = FALSE)
# lista neordonata
S
[1]  9 23  7  4  3 13 17 10 15 24 12 19  2 20  5 22  8  1 21  6 18 11 14 16 25
# lista ordonata
quickSort(S)
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

Numărul mediu de comparații pe care le efectuează algoritmul *Quicksort randomizat*, versiunea empirică versus cea teoretică de mai sus, este ilustrat în figura următoare:

```
countQuickSort <- function(vect) {
  # Args:
  #   vect: Vector numeric

  # daca lungimea este <= 1 stop
  if (length(vect) <= 1) {
    return(vect)
  }

  count <- count + length(vect) - 1 # imi intoarce numarul de comparatii efectuate

  # alege pivotul
  ide = sample(1:length(vect), 1)
  element = vect[ide]
  partition = vect[-ide]

  # Imparte elementele in doua subliste (< pivot si >= pivot)
  v1 = partition[partition < element]
  v2 = partition[partition >= element]

  # Aplica recursiv algoritmul
  v1 = countQuickSort(v1)
  v2 = countQuickSort(v2)
  return(c(v1, element, v2))
}

N = 1000
y = rep(0, N)

for (i in 1:N){
  S = sample(1:i, i, replace = FALSE)

  count = 0
```

```
S_sort = countQuickSort(S)

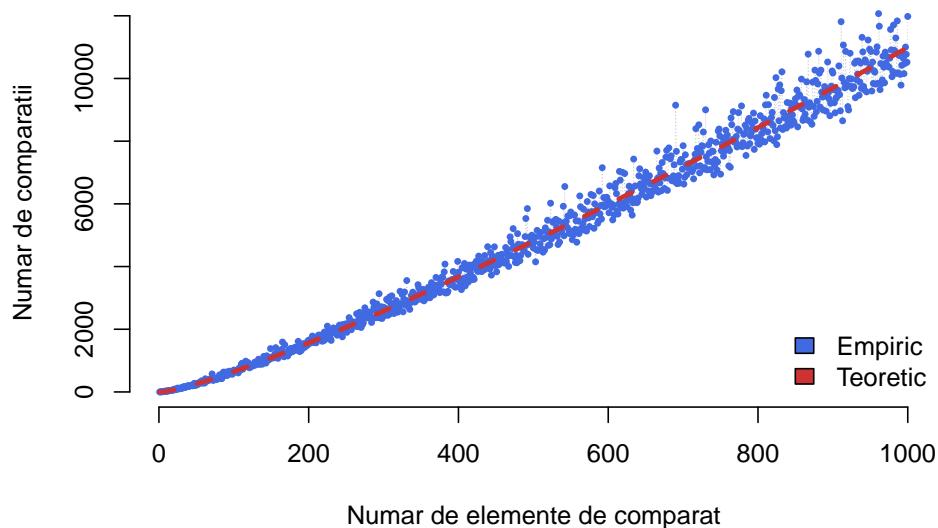
y[i] = count
}

# Functia care calculeaza numarul de operatii teoretice
T_n = function(n){
  return(2*(n+1)*sum(1/(1:n))-4*n)
}

theo_T = sapply(1:N, function(x){T_n(x)})

# Graficul
plot(1:N, y, type = "l",
      col = "grey80",
      bty = "n",
      main = "Algoritmul Quicksort",
      xlab = "Numar de elemente de comparat",
      ylab = "Numar de comparatii",
      lty = 3,
      lwd = 0.5)
points(1:N, y,
       col = "royalblue",
       pch = 16,
       cex = 0.6)
lines(1:N, theo_T, col = "brown3", lwd = 3, lty = 2)
legend('bottomright',
       legend = c("Empiric", "Teoretic"),
       fill = c("royalblue", "brown3"),
       bty = "n")
```

Algoritmul Quicksort



Laborator 6

Repartiții continue (univariate)

Obiectivul acestui laborator este de a prezenta o parte din cele mai cunoscute repartiții continue¹ și de a rezolva câteva probleme cu ajutorul lor.

R pune la dispozitie majoritatea repartițiilor uzuale. Tabelul de mai jos prezintă numele și parametrii acestora:

Tab. 1: Numele și parametrii repartițiilor uzuale în R

Repartiția	Nume	Parametrii	Valori prestabilite
Uniformă	<code>unif</code>	<code>min, max</code>	<code>min = 0, max = 1</code>
Normală	<code>norm</code>	<code>mean, sd</code>	<code>mean = 0, sd = 1</code>
Log-Normală	<code>lnorm</code>	<code>mean, sd</code>	<code>mean = 0, sd = 1</code>
Exponentială	<code>exp</code>	<code>rate (=1/mean)</code>	<code>rate = 1</code>
Cauchy	<code>cauchy</code>	<code>location, scale</code>	<code>location = 0, scale = 1</code>
Gamma	<code>gamma</code>	<code>shape, rate (=1/scale)</code>	<code>rate = 1</code>
Beta	<code>beta</code>	<code>shape1, shape2</code>	
Student	<code>t</code>	<code>df</code>	
Chi-Squared	<code>chisq</code>	<code>df</code>	
Fisher	<code>f</code>	<code>df1, df2</code>	
Weibull	<code>weibull</code>	<code>shape</code>	

Pentru fiecare repartitie, există patru comenzi în R prefixate cu literele `d`, `p`, `q` și `r` și următoare de numele repartiției (coloana a 2-a). De exemplu `dnorm`, `pnorm`, `qnorm` și `rnorm` sunt comenzi corespunzătoare repartiției normale pe când `dunif`, `punif`, `qunif` și `runif` sunt cele corespunzătoare repartiției uniforme.

- `dnume`: calculează densitatea atunci când vorbim de o variabilă continuă sau funcția de masă atunci când avem o repartitie discretă ($\mathbb{P}(X = k)$)
- `pnume`: calculează funcția de repartitie, i.e. $F(x) = \mathbb{P}(X \leq x)$
- `qnume`: reprezintă funcția cuantilă, cu alte cuvinte valoarea pentru care funcția de repartitie are o anumită probabilitate; în cazul continuu, dacă `pnume(x) = p` atunci `qnume(p) = x` iar în cazul discret întoarce cel mai mic întreg u pentru care $\mathbb{P}(X \leq u) \geq p$.
- `rnume`: generează observații independente din repartitia dată

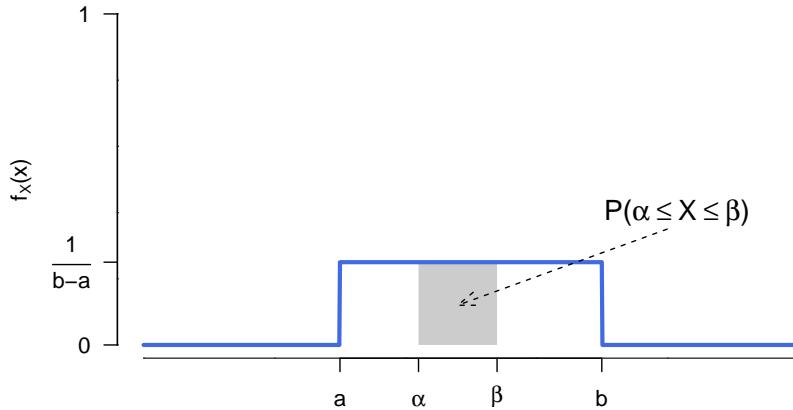
1 Repartitie Uniformă

O variabilă aleatoare X repartizată *uniform* pe intervalul $[a, b]$, notată $X \sim \mathcal{U}[a, b]$, are densitatea dată de

$$f_X(x) = \begin{cases} \frac{1}{b-a}, & x \in [a, b] \\ 0, & \text{altfel} \end{cases}$$

¹Pentru mai multe informații, se poate consulta monografia Johnson, N., Kotz, S. și Balakrishnan, N. *Continuous Univariate Distributions*, (Volumul 1, Ediția a 2-a), John Wiley & Sons, New York (1994), 756 pp., ISBN 0-471-58495-9

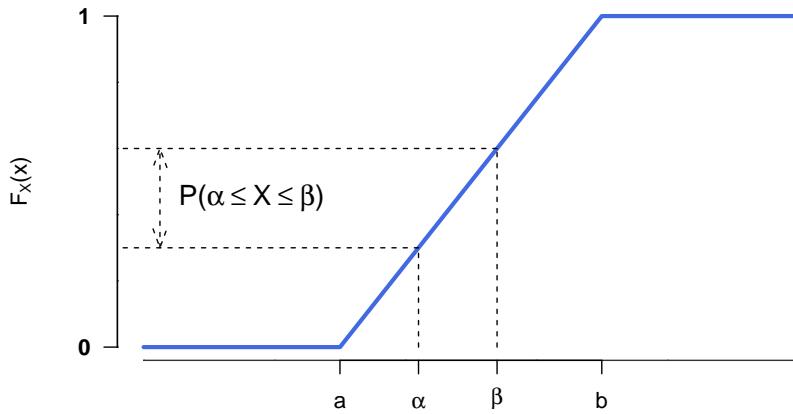
Densitatea repartitiei uniforme pe $[a,b]$



Funcția de repartiție a repartiției uniforme este

$$F_X(x) = \int_{-\infty}^x f_X(t) dt = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & x \in (a, b) \\ 1, & x \geq b \end{cases}$$

Functia de repartitie a uniformei pe $[a,b]$



Media și varianța variabilei aleatoare X repartizate uniform pe $[a, b]$ sunt egale cu

$$\mathbb{E}[X] = \frac{a+b}{2}, \quad Var(X) = \frac{(a-b)^2}{12}.$$

Variabilele aleatoare repartizate uniform joacă un rol important în teoria simulării variabilelor aleatoare datorită următorului rezultat datorat lui Paul Levy și numit *teorema de universalitate a repartitiei uniforme*:



Fie X o variabilă aleatoare reală cu funcția de repartiție F , U o variabilă aleatoare repartizată uniform pe $[0, 1]$ și fie funcția *cuantilă* (inversa generalizată) asociată lui F , $F^{-1} : (0, 1) \rightarrow \mathbb{R}$ definită prin

$$F^{-1}(u) = \inf\{x \in \mathbb{R} \mid F(x) \geq u\}, \quad \forall u \in (0, 1).$$

Atunci X și $F^{-1}(U)$ sunt repartizate la fel.

În R putem să

- generăm observații independente din repartiția $\mathcal{U}([a, b])$ (e.g. $a = 3$ și $b = 5$)

```
runif(10, 3, 5)
```

```
[1] 4.960250 4.403519 3.731753 3.632980 4.317326 4.866822 3.297264 4.662233
[9] 3.099080 3.664218
```

- calculăm densitatea unei variabile aleatoare repartizate uniform pe $[a, b]$ în diferite puncte

```
dunif(c(3.1, 3.7, 3.95, 4.86), 3, 5)
```

```
[1] 0.5 0.5 0.5 0.5
```

- calculăm funcția de repartiție a unei variabile repartizate uniform pe $[a, b]$ pentru diferite valori

```
punif(c(3.1, 3.7, 3.95, 4.86), 3, 5)
```

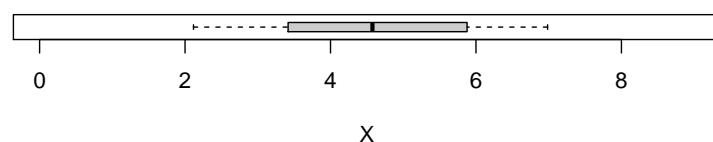
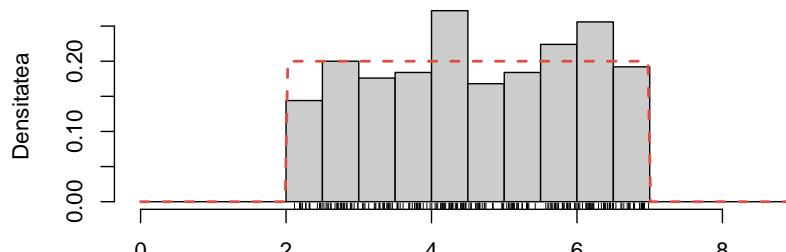
```
[1] 0.050 0.350 0.475 0.930
```



Fie X o variabilă aleatoare repartizată uniform pe $[2, 7]$. Determinați:

- $\mathbb{P}(X \in \{1, 2, 3, 4, 5, 6, 7\})$
- $\mathbb{P}(X < 3)$ și $\mathbb{P}(X \leq 3)$
- $\mathbb{P}(X \leq 3 \cup X > 4)$
- Generați 250 de observații din repartiția dată, trasați histograma acestora și suprapuneți densitatea repartiției date (vezi figura de mai jos).

Repartitia uniformă pe $[2,7]$





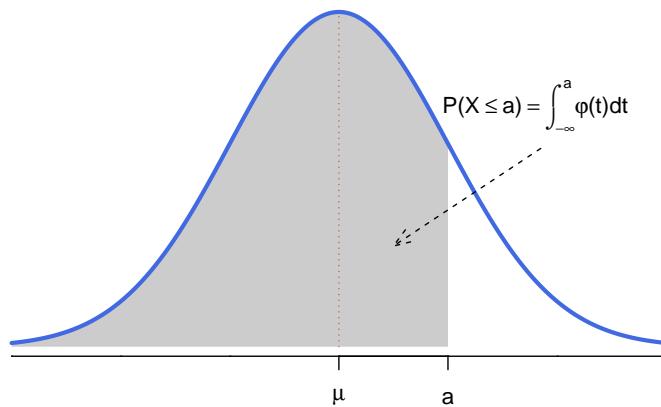
Dacă X o variabilă aleatoare repartizată uniform pe $[a, b]$ și $[c, d] \subset [a, b]$ este un subinterval, atunci repartiția condiționată a lui X la $X \in [c, d]$ este $\mathcal{U}[c, d]$.

2 Repartiția Normală

Spunem că o variabilă aleatoare X este repartizată *normal* sau *Gaussian* de medie μ și varianță σ^2 , și se notează cu $X \sim \mathcal{N}(\mu, \sigma^2)$, dacă densitatea ei are forma

$$f_X(x) \stackrel{\text{not}}{=} \varphi(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R}.$$

Densitatea repartitiei normale $N(\mu, \sigma^2)$



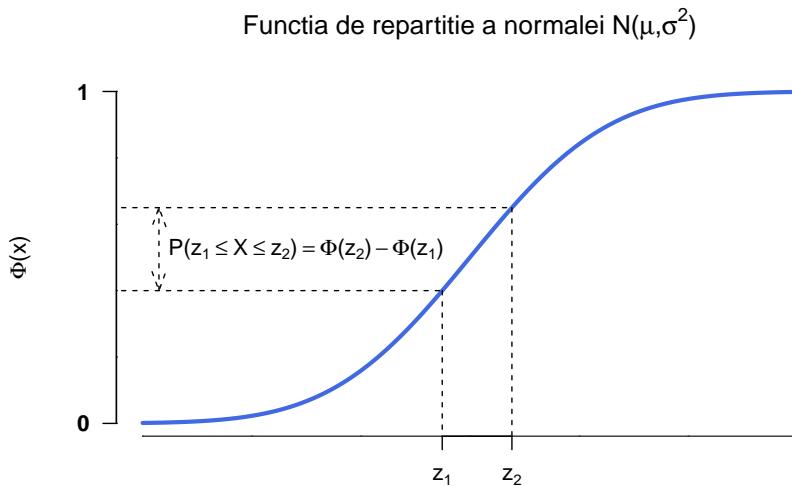
Funcția de repartiție a unei variabile $X \sim \mathcal{N}(\mu, \sigma^2)$ este dată de

$$F_X(x) \stackrel{\text{not}}{=} \Phi(x) = \int_{-\infty}^x \varphi(t) dt = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt.$$

Pentru funcția de repartiție nu avem o formulă explicită de calcul, ea poate fi aproximată cu ajutorul descompunerii în serie. În cazul variabilelor *normale standard* ($X \sim \mathcal{N}(0, 1)$) avem proprietățile

- a) $\Phi(x) = 1 - \Phi(-x)$ pentru toate valorile $x \in \mathbb{R}$
- b) $1 - \Phi(a) \leq \frac{1}{2}e^{-\frac{a^2}{2}}$ pentru $a > 0$ ²

²Pentru mai multe astfel de inegalități se poate consulta cartea (capitolul 2): Lin, Z. și Bai, Z. *Probability Inequalities*, Springer, 2010.



Media și varianța variabilei aleatoare X repartizate normal de parametrii $\mathcal{N}(\mu, \sigma^2)$ sunt egale cu

$$\mathbb{E}[X] = \mu, \quad \text{Var}(X) = \sigma^2.$$

Mai mult, momentele de ordin se pot calcula cu ușurință și avem că

$$\mathbb{E}[X^k] = \begin{cases} \sigma^k (k-1)!! & k \text{ este par} \\ 0 & k \text{ este impar.} \end{cases}$$

Pentru o variabilă aleatoare repartizată normal, avem următoarea regulă numită și regula 68 – 95 – 99.7%:



Fie X o variabilă aleatoare repartizată $\mathcal{N}(\mu, \sigma^2)$. Atunci

$$\begin{aligned} \mathbb{P}(|X - \mu| < \sigma) &\approx 0.68 \\ \mathbb{P}(|X - \mu| < 2\sigma) &\approx 0.95 \\ \mathbb{P}(|X - \mu| < 3\sigma) &\approx 0.997 \end{aligned}$$

În R putem să

- generăm observații independente din repartiția $\mathcal{N}(\mu, \sigma^2)$ (e.g. $\mu = 0$ și $\sigma^2 = 2$ - în R funcțiile `rnorm`, `dnorm`, `pnorm` și `qnorm` primesc ca parametrii media și abaterea standard, σ nu varianța σ^2)

```
rnorm(10, mean = 0, sd = sqrt(2))
[1] 0.0798922 -2.4044042 -2.3017202 1.0751185 -0.1834820 1.0632273
[7] -0.1228208 0.1467213 -1.2199174 -0.5529398
```

- calculăm densitatea unei variabile aleatoare repartizate normal $\mathcal{N}(\mu, \sigma^2)$ în diferite puncte

```
dnorm(seq(-2, 2, length.out = 15), mean = 3, sd = 5)
[1] 0.04839414 0.05115647 0.05390019 0.05660592 0.05925368 0.06182308
[7] 0.06429362 0.06664492 0.06885700 0.07091058 0.07278734 0.07447021
[13] 0.07594361 0.07719368 0.07820854
```

- calculăm funcția de repartiție a unei variabile repartizate normal $\mathcal{N}(\mu, \sigma^2)$ pentru diferite valori

```
pnorm(seq(-1, 1, length.out = 15), mean = 3, sd = 1)
[1] 3.167124e-05 5.736006e-05 1.018892e-04 1.775197e-04 3.033834e-04
[6] 5.086207e-04 8.365374e-04 1.349898e-03 2.137367e-03 3.320943e-03
[11] 5.063995e-03 7.579219e-03 1.113549e-02 1.606229e-02 2.275013e-02
```

- calculăm cuantilele de ordin $\alpha \in (0, 1)$ (i.e. valoarea z_α pentru care $\Phi(z_\alpha) = \alpha$ sau altfel spus $z_\alpha = \Phi^{-1}(\alpha)$)

```
qnorm(c(0.01, 0.025, 0.05, 0.25, 0.5, 0.75, 0.95, 0.975, 0.99), mean = 0, sd = 1)
[1] -2.3263479 -1.9599640 -1.6448536 -0.6744898 0.0000000 0.6744898 1.6448536
[8] 1.9599640 2.3263479
```



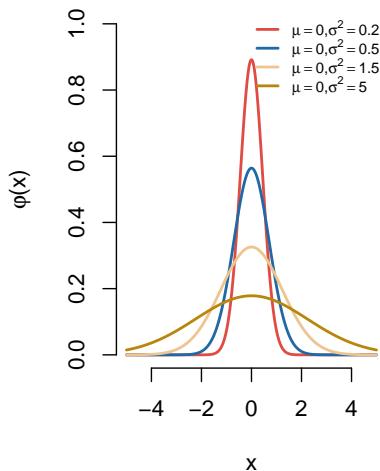
Fie X o variabilă aleatoare repartizată $\mathcal{N}(\mu, \sigma^2)$. Atunci pentru $\mu = 1$ și $\sigma = 3$ calculați:

- 1) $\mathbb{P}(X \text{ este par})$
- 2) $\mathbb{P}(X < 3.4)$ și $\mathbb{P}(X > 1.3)$
- 3) $\mathbb{P}(1 < X < 4)$
- 4) $\mathbb{P}(X \in [2, 3] \cup [3.5, 5])$
- 5) $\mathbb{P}(|X - 3| > 6)$

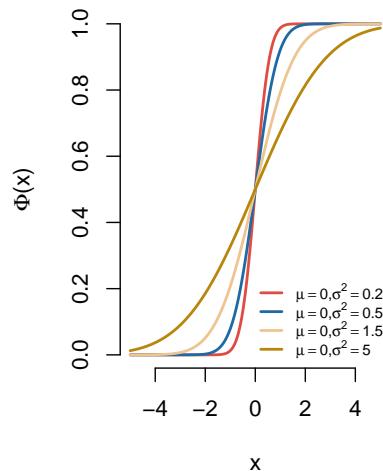


Fie X o variabilă aleatoare repartizată $\mathcal{N}(\mu, \sigma^2)$. Pentru $\mu = 0$ și $\sigma^2 \in \{0.2, 0.5, 1.5, 5\}$ trasați pe același grafic densitățile repartițiilor normale cu parametrii $\mathcal{N}(\mu, \sigma^2)$. Adăugați legendele corespunzătoare. Aceeași cerință pentru funcțiile de repartiție.

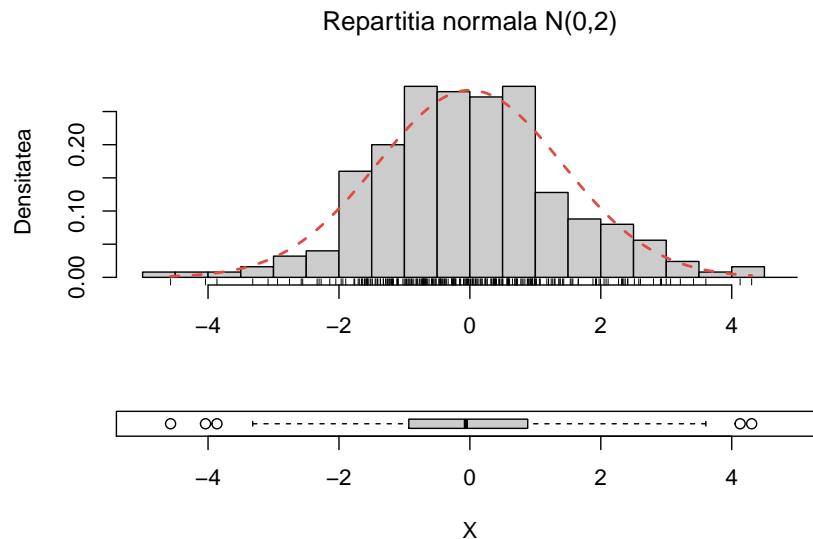
Densitatea



Functia de repartitie

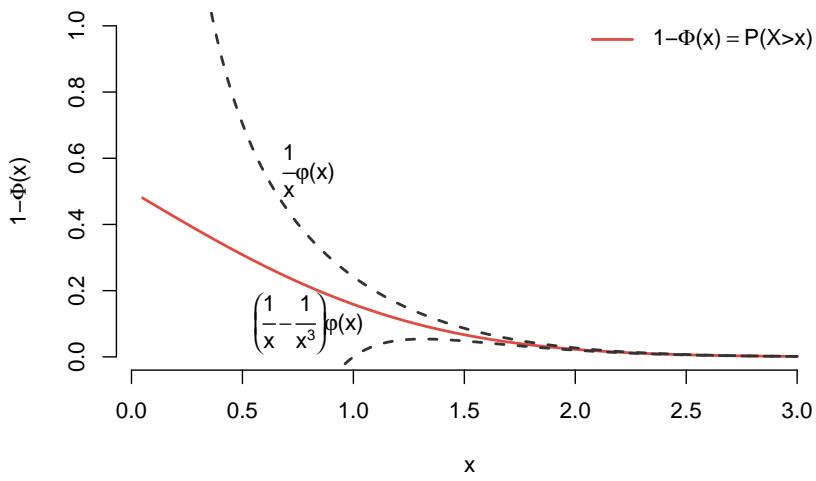


Generați 250 de observații din repartiția $\mathcal{N}(0, 2)$, trasați histograma acestora și suprapuneți densitatea repartiției date (vezi figura de mai jos).



Fie X o variabilă aleatoare repartizată normal de parametrii μ și σ^2 . Ilustrați grafic pentru $\mu = 0$ și $\sigma = 1$ că are loc următoarea inegalitate:

$$\left(\frac{1}{x} - \frac{1}{x^3}\right)\phi(x) < 1 - \Phi(x) < \frac{1}{x}\phi(x), \quad x > 0.$$

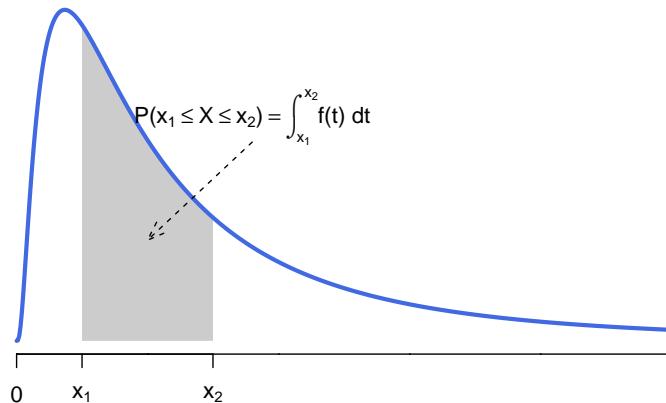


3 Repartiția Log-Normală

Spune că o variabilă aleatoare X este repartizată log-normal de parametrii μ și σ^2 , și notăm $X \sim LN(\mu, \sigma^2)$, dacă $\ln(X)$ este repartizată normal de parametrii μ și σ^2 . Cu alte cuvinte dacă $Y \sim \mathcal{N}(\mu, \sigma^2)$ atunci $X = e^Y \sim LN(\mu, \sigma^2)$. Densitatea repartiției log-normale $LN(\mu, \sigma^2)$ este

$$f_X(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}}, \quad x \in (0, +\infty).$$

Densitatea repartitiei log-normale $LN(\mu, \sigma^2)$

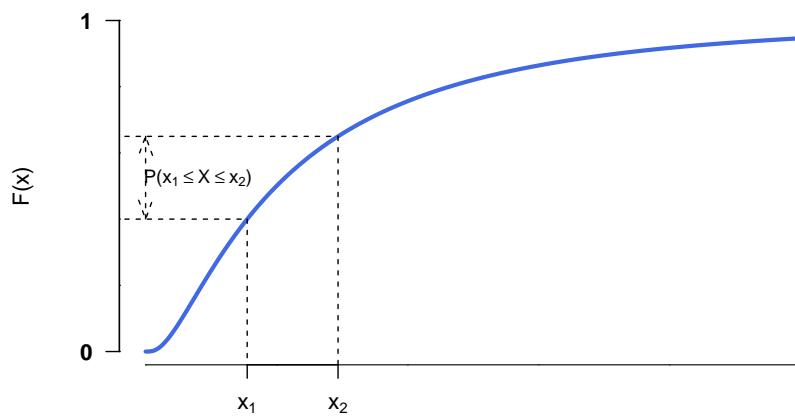


Funcția de repartiție a unei variabile aleatoare $X \sim LN(\mu, \sigma^2)$ este dată de

$$F_X(x) = \int_{-\infty}^x f_X(t) dt = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x \frac{1}{t} e^{-\frac{(\ln(t)-\mu)^2}{2\sigma^2}} dt$$

și, ca și în cazul repartiției normale, nu are o formulă explicită de calcul.

Functia de repartitie a log-normalei $LN(\mu, \sigma^2)$



Media și varianța variabilei aleatoare X repartizate log-normal de parametrii $LN(\mu, \sigma^2)$ sunt egale cu

$$\mathbb{E}[X] = e^{\mu + \frac{\sigma^2}{2}}, \quad Var(X) = (e^{\sigma^2} - 1) e^{2\mu + \sigma^2}.$$



Arătați că media și varianța unei variabile aleatoare repartizate log-normal de parametrii μ și σ^2 sunt egale cu

$$\mathbb{E}[X] = e^{\mu + \frac{\sigma^2}{2}}, \quad \text{Var}(X) = (e^{\sigma^2} - 1) e^{2\mu + \sigma^2}.$$

În R putem să

- generăm observații independente din repartiția $LN(\mu, \sigma^2)$ (e.g. $\mu = 0$ și $\sigma^2 = 3$ - ca și în cazul repartiției normale, funcțiile `rlnorm`, `dlnorm`, `plnorm` și `qlnorm` primesc ca parametrii media și abaterea standard, σ pentru $\ln(X)$ - variabila normală)

```
rlnorm(15, meanlog = 0, sdlog = sqrt(3))
[1] 2.13141475 6.27258447 2.18850080 3.15407005 0.13970018 0.52638598
[7] 12.91237780 0.12004802 1.56359485 2.01674623 5.42024453 0.54647199
[13] 1.31619806 0.04716763 1.79762358
```

- calculăm densitatea unei variabile aleatoare repartizate log-normal $LN(\mu, \sigma^2)$ în diferite puncte

```
dlnorm(seq(0, 5, length.out = 20), meanlog = 3, sdlog = 5)
[1] 0.00000000 0.20820751 0.11627647 0.08196427 0.06370023 0.05226715
[7] 0.04440086 0.03864103 0.03423291 0.03074580 0.02791546 0.02557044
[13] 0.02359456 0.02190618 0.02044622 0.01917084 0.01804680 0.01704845
[19] 0.01615564 0.01535234
```

- calculăm funcția de repartiție a unei variabile repartizate log-normal $LN(\mu, \sigma^2)$ pentru diferite valori

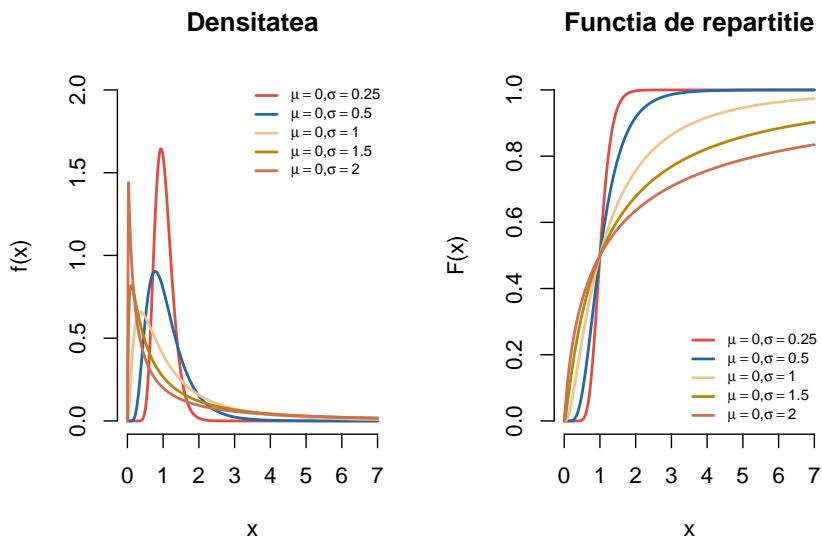
```
plnorm(seq(0, 15, length.out = 25), meanlog = 3, sdlog = 1)
[1] 0.0000000000 0.0002602257 0.0027443707 0.0088606283 0.0185933103
[6] 0.0314027650 0.0466497221 0.0637426806 0.0821791298 0.1015482283
[11] 0.1215206945 0.1418356830 0.1622882185 0.1827183180 0.2030019832
[16] 0.2230439002 0.2427715876 0.2621307274 0.2810814477 0.2995953616
[21] 0.3176532076 0.3352429649 0.3523583472 0.3689975944 0.3851625036
```

- calculăm cuantilele de ordin $\alpha \in (0, 1)$

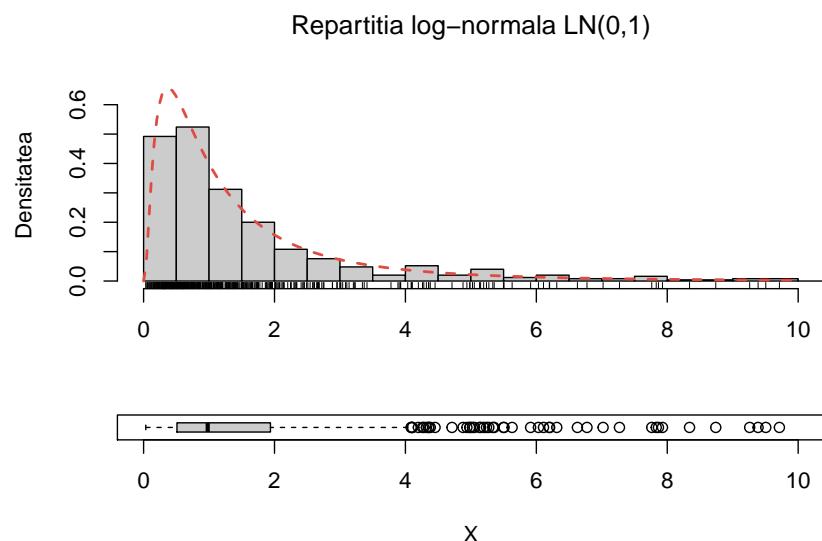
```
qlnorm(c(0.01, 0.025, 0.05, 0.25, 0.5, 0.75, 0.95, 0.975, 0.99), meanlog = 0, sdlog = 1)
[1] 0.09765173 0.14086349 0.19304082 0.50941628 1.00000000 1.96303108
[7] 5.18025160 7.09907138 10.24047366
```



Fie X o variabilă aleatoare repartizată $LN(\mu, \sigma^2)$. Pentru $\mu = 0$ și $\sigma \in \{0.25, 0.5, 1.5, 5\}$ trasați pe același grafic densitățile repartițiilor log-normale cu parametrii $LN(\mu, \sigma^2)$. Adăugați legendele corespunzătoare. Aceeași cerință pentru funcțiile de repartiție.



Generați 500 de observații din repartiția $LN(0, 2)$, trasați histograma acestora și suprapuneți densitatea repartiției date (vezi figura de mai jos).



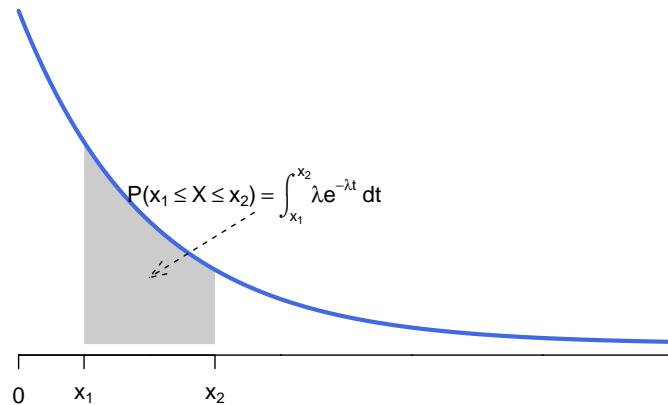
Printre fenomenele care pot fi modelate cu ajutorul repartiției log-normale se numără: cantitatea de lapte produsă de vaci, cantitatea de ploaie dintr-o perioadă dată, repartiția mărimii picăturilor de ploaie, volumul de gaz dintr-o rezervă petrolieră, etc. Pentru mai multe aplicații se poate consulta lucrarea lui Limpert, E., Stajel, W. și Abbt, M. *Log-normal Distributions across the Sciences: Keys and Clues*, *BioScience*, Vol. 51, Nr. 5, 2001.

4 Repartiția Exponențială

Spunem că o variabilă aleatoare X este repartizată *exponential* de parametru λ , și se notează cu $X \sim \mathcal{E}(\lambda)$, dacă densitatea ei are forma

$$f_X(x) = \lambda e^{-\lambda x} \mathbb{1}_{\mathbb{R}_+}(x), \quad \forall x \in \mathbb{R}.$$

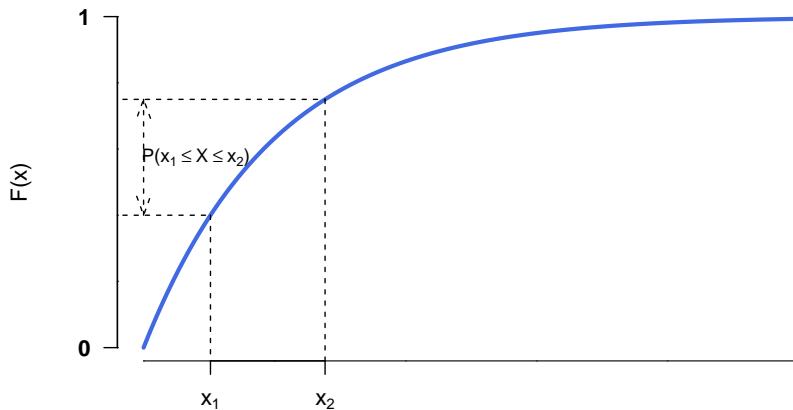
Densitatea repartitiei exponentiale $E(\lambda)$



Funcția de repartiție a unei variabile aleatoare $X \sim E(\lambda)$ este dată de

$$F_X(x) = 1 - e^{-\lambda x} \mathbb{1}_{\mathbb{R}_+}(x), \quad x \in \mathbb{R}.$$

Functia de repartitie a exponentialei $E(\lambda)$



Media și varianța variabilei aleatoare X repartizate exponențial de parametru λ sunt egale cu

$$\mathbb{E}[X] = \frac{1}{\lambda}, \quad Var(X) = \frac{1}{\lambda^2}.$$



Arătați că momentul de ordin k , $k \geq 1$, al unei variabile aleatoare repartizate exponentiațial $X \sim \mathcal{E}(\lambda)$ este egal cu

$$\mathbb{E}[X^k] = \frac{k!}{\lambda^k}.$$



Fie X o variabilă repartizată exponentiațial de parametru λ . Atunci are loc următoarea proprietate numită și *lipsa de memorie*:

$$\mathbb{P}(X > s + t | X > s) = \mathbb{P}(X > t), \quad \forall s, t \geq 0.$$

Mai mult, dacă o variabilă aleatoare continuă³ X verifică proprietatea de mai sus atunci ea este repartizată exponentiațial.

Variabilele aleatoare repartizate exponentiațial sunt utilizate în modelarea fenomenelor care se desfășoară în timp continuu și care satisfac (aproximativ) proprietatea lipsei de memorie: de exemplu timpul de așteptare la un ghișeu, durata de viață a unui bec sau timpul până la următoarea con vorbire telefonică.

În R putem să

- generăm observații independente din repartiția $\mathcal{E}(\lambda)$ (e.g. $\lambda = 5$)

```
rexp(15, rate = 5)
[1] 0.13505357 0.15392539 0.25036131 0.15351051 0.00878456 0.07362396
[7] 0.07543271 0.18981181 0.05540771 0.05649451 0.15878039 0.39847262
[13] 0.05191221 0.07776034 0.22483594
```

- calculăm densitatea unei variabile aleatoare repartizate exponentiațial $\mathcal{E}(\lambda)$ în diferite puncte

```
dexp(seq(0, 5, length.out = 20), rate = 5)
[1] 5.000000e+00 1.341312e+00 3.598237e-01 9.652719e-02 2.589462e-02
[6] 6.946555e-03 1.863500e-03 4.999070e-04 1.341063e-04 3.597568e-05
[11] 9.650925e-06 2.588981e-06 6.945263e-07 1.863153e-07 4.998141e-08
[16] 1.340814e-08 3.596899e-09 9.649130e-10 2.588499e-10 6.943972e-11
```

- calculăm funcția de repartiție a unei variabile repartizate exponentiațial $\mathcal{E}(\lambda)$ pentru diferite valori

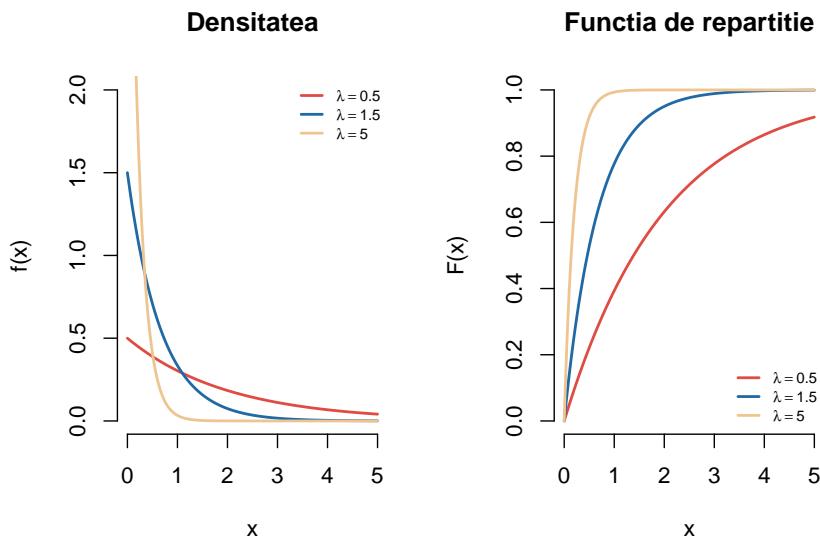
```
pexp(seq(0, 5, length.out = 15), rate = 5)
[1] 0.0000000 0.8323228 0.9718843 0.9952856 0.9992095 0.9998675 0.9999778
[8] 0.9999963 0.9999994 0.9999999 1.0000000 1.0000000 1.0000000
[15] 1.0000000
```

- calculăm cuantilele de ordin $\alpha \in (0, 1)$

```
qexp(c(0.01, 0.025, 0.05, 0.25, 0.5, 0.75, 0.95, 0.975, 0.99), rate = 5)
[1] 0.002010067 0.005063562 0.010258659 0.057536414 0.138629436 0.277258872
[7] 0.599146455 0.737775891 0.921034037
```



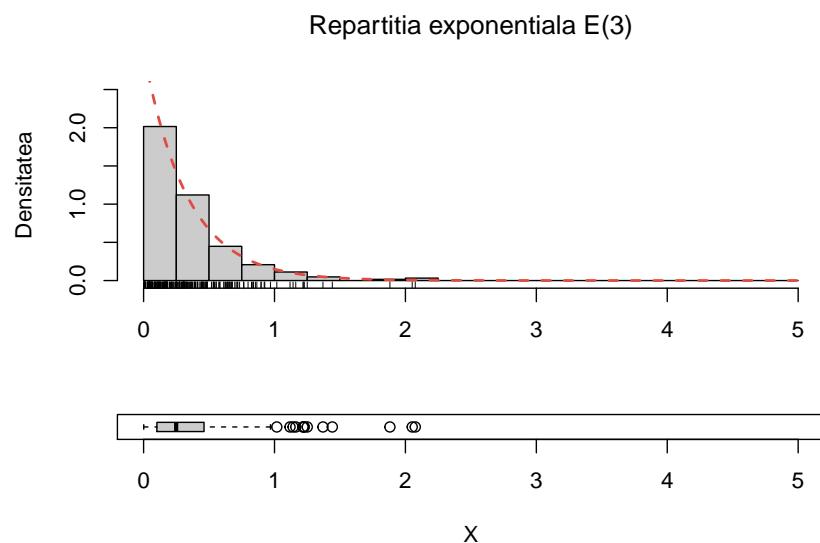
Fie X o variabilă aleatoare repartizată $\mathcal{E}(\lambda)$. Pentru $\lambda \in \{0.5, 1.5, 5\}$ trasați pe același grafic densitățile repartițiilor exponențiale de parametru λ . Adăugați legendele corespunzătoare. Aceeași cerință pentru funcțiile de repartiție.



Folosind rezultatul de universalitate de la repartiția uniformă, descrieți o procedură prin care puteți simula o variabilă aleatoare repartizată exponențială $\mathcal{E}(\lambda)$ și construiți o funcție care permite generarea de n observații independente dintr-o variabilă repartizată $X \sim \mathcal{E}(\lambda)$.



Generați 250 de observații din repartiția $\mathcal{E}(3)$, trasați histograma acestora și suprapuneți densitatea repartiției date (vezi figura de mai jos).



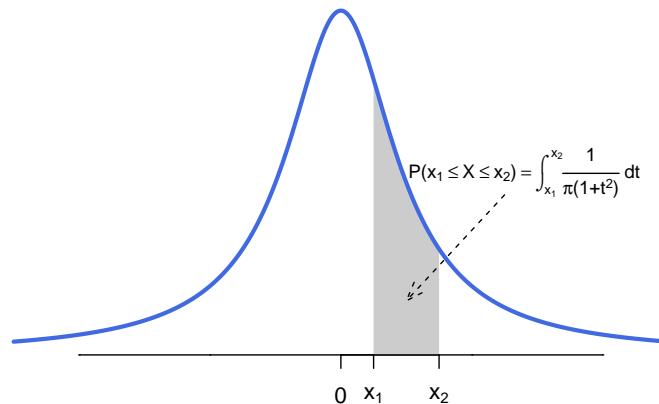
5 Repartiția Cauchy

Spunem că o variabilă aleatoare X este repartizată *Cauchy* de parametrii $(0, 1)$, și se notează cu $X \sim C(0, 1)$, dacă densitatea ei are forma

$$f_X(x) = \frac{1}{\pi} \frac{1}{1+x^2}, \quad \forall x \in \mathbb{R}.$$

Observăm că graficul densității repartiției Cauchy este asemănător cu cel al repartiției normale. Parametrul $M = 0$ reprezintă mediana (de fapt $\mathbb{P}(X \leq 0) = \mathbb{P}(X \geq 0) = \frac{1}{2}$) variabilei aleatoare X și nu media iar prima și a treia quartilă sunt $Q_1 = -1$ și respectiv $Q_3 = 1$ (avem $\mathbb{P}(X \leq -1) = \mathbb{P}(X \geq 1) = \frac{1}{4}$).

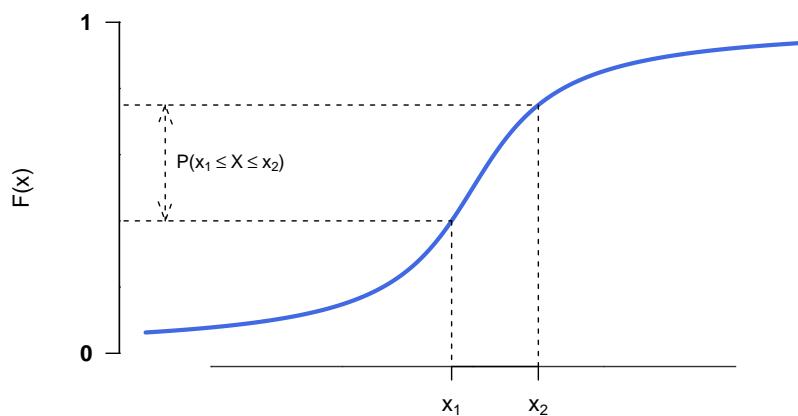
Densitatea repartitiei Cauchy



Funcția de repartiție a unei variabile aleatoare $X \sim C(0, 1)$ este dată de

$$F_X(x) = \frac{1}{2} + \frac{1}{\pi} \arctan(x), \quad x \in \mathbb{R}.$$

Functia de repartitie a repartitiei Cauchy



Media și varianța variabilei aleatoare $X \sim C(0, 1)$ **nu există**.



Arătați că o variabilă aleatoare repartizată Cauchy $C(0, 1)$ nu are medie.

Fie $Y \sim C(0, 1)$ și $\alpha, \beta \in \mathbb{R}$ cu $\beta > 0$. Spunem că variabila aleatoare $X = \alpha + \beta Y$ este repartizată Cauchy de parametrii (α, β) , $X \sim C(\alpha, \beta)$. Densitatea ei este

$$f_X(x) = \frac{1}{\pi\beta} \frac{1}{1 + \left(\frac{x-\alpha}{\beta}\right)^2}, \quad \forall x \in \mathbb{R}.$$

Parametrii α și β se interpretează în modul următor: $M = \alpha$ este mediana lui X iar $Q_1 = \alpha - \beta$ și $Q_3 = \alpha + \beta$ reprezintă prima și a treia quartilă.

În R putem să

- generăm observații independente din repartiția Cauchy $C(\alpha, \beta)$ (e.g. $\alpha = 0, \beta = 2$)

```
rcauchy(15, location = 0, scale = 2)
[1] -0.5966228  3.7627987  0.6864597 -0.4316018  1.4524446  0.3427032
[7]  8.4285326  3.6056089  2.3506764 -3.5453329 -1.6137218 10.4304800
[13] -0.4449169  2.3005176 -3.6644199
```

- calculăm densitatea unei variabile aleatoare repartizate Cauchy $C(\alpha, \beta)$ în diferite puncte

```
dcauchy(seq(-5, 5, length.out = 20), location = 1, scale = 3)
[1] 0.02122066 0.02450975 0.02852541 0.03345265 0.03951056 0.04693392
[7] 0.05591721 0.06648594 0.07825871 0.09012539 0.10006665 0.10558334
[13] 0.10494052 0.09835367 0.08782920 0.07584810 0.06425529 0.05399054
[19] 0.04532934 0.03819719
```

- calculăm funcția de repartiție a unei variabile repartizate Cauchy $C(\alpha, \beta)$ pentru diferite valori

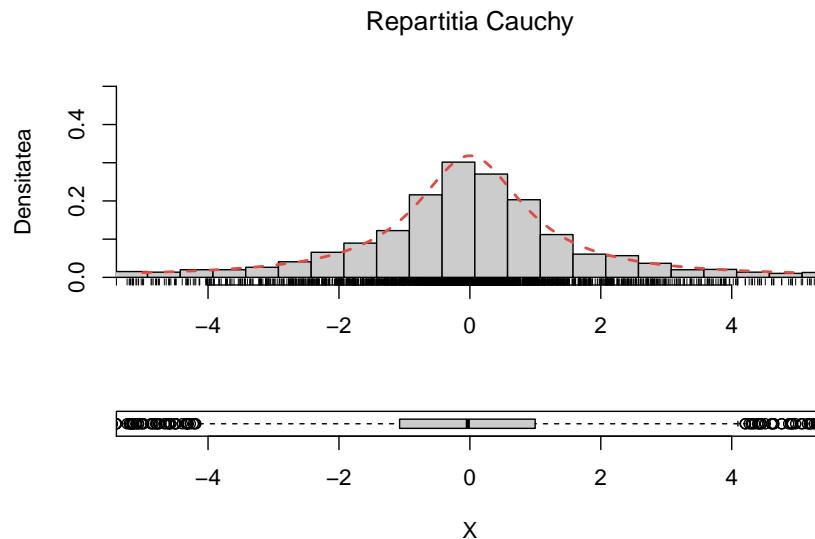
```
pcauchy(seq(-5, 5, length.out = 15), location = 1, scale = 3)
[1] 0.1475836 0.1643213 0.1848605 0.2104166 0.2425988 0.2833834 0.3347507
[8] 0.3975836 0.4697759 0.5451672 0.6158581 0.6764416 0.7255627 0.7644587
[15] 0.7951672
```

- calculăm cuantilele de ordin $p \in (0, 1)$

```
qcauchy(c(0.01, 0.025, 0.05, 0.25, 0.5, 0.75, 0.95, 0.975, 0.99), location = 1, scale = 3)
[1] -94.46155 -37.11861 -17.94125 -2.00000  1.00000  4.00000 19.94125
[8]  39.11861  96.46155
```

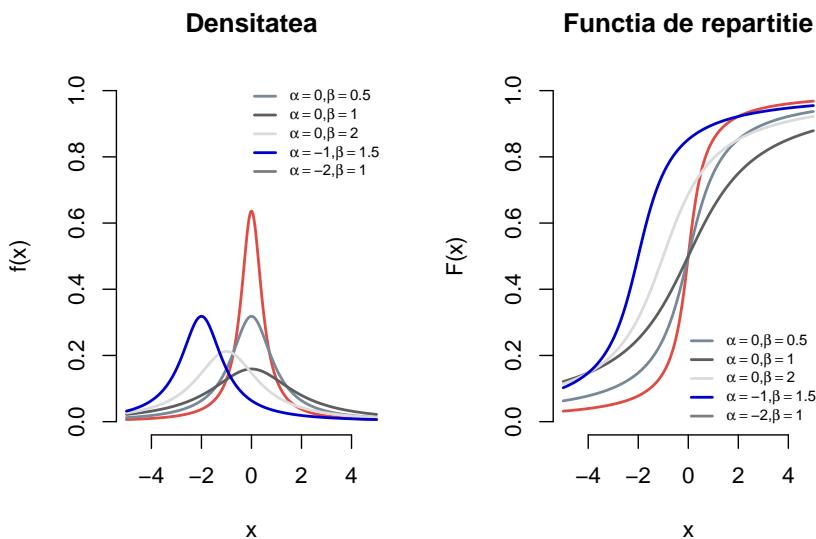


Generați 2500 de observații din repartiția Cauchy, trasați histograma acestora și suprapuneți densitatea repartiției date pentru intervalul $[-5, 5]$ (vezi figura de mai jos).



Fie X și Y două variabile aleatoare independente repartizate $\mathcal{N}(0, 1)$. Arătați că variabila aleatoare $\frac{X}{Y}$ este repartizată Cauchy $C(0, 1)$.

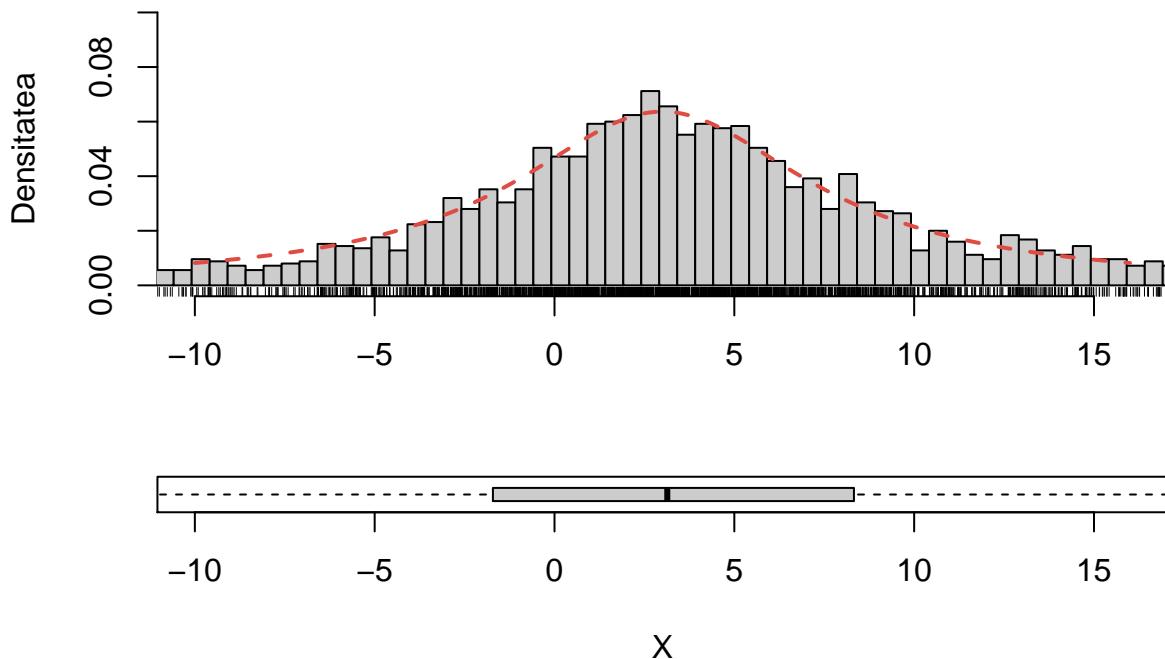
Fie X o variabilă aleatoare repartizată Cauchy $C(\alpha, \beta)$. Pentru fiecare pereche de parametrii (α, β) din mulțimea $\{(0, 0.5), (0, 1), (0, 2), (-1, 1.5), (-2, 1)\}$ trasați pe același grafic densitățile repartițiilor Cauchy cu parametrii (α, β) . Adăugați legendele corespunzătoare. Aceeași cerință pentru funcțiile de repartiție.



Folosind rezultatul de universalitate de la repartiția uniformă, descrieți o procedură prin care puteți simula o variabilă aleatoare repartizată Cauchy $C(0, 1)$ și construiți o funcție care permite

generarea de n observații independente dintr-o variabilă repartizată $X \sim C(\alpha, \beta)$. Verificați pentru parametrii $\alpha = 3$ și $\beta = 5$ (a se vedea figura de mai jos).

Repartitia Cauchy $C(3,5)$



6 Repartiția Gama

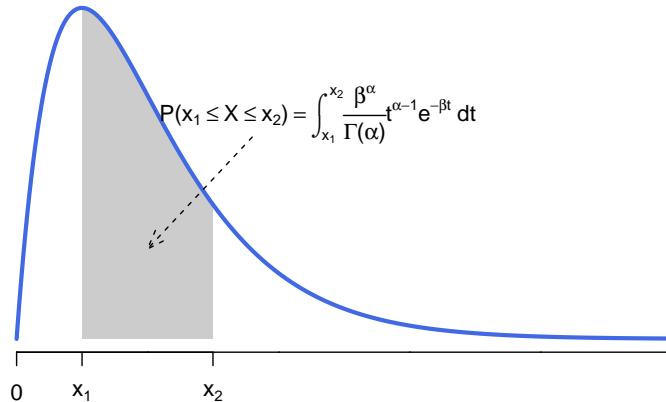
Spunem că o variabilă aleatoare X este repartizată *Gama* de parametrii (α, β) , cu $\alpha, \beta > 0$, și se notează cu $X \sim \Gamma(\alpha, \beta)$, dacă densitatea ei are forma

$$f_X(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}, \quad \forall x > 0.$$

unde $\Gamma(\alpha)$ este funcția (Gama, numită și integrală Euler de al doilea tip) definită prin

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} dx, \quad \forall \alpha > 0.$$

Densitatea repartitiei $\Gamma(\alpha, \beta)$



Arătați că funcția $\Gamma(\alpha)$ verifică⁴:

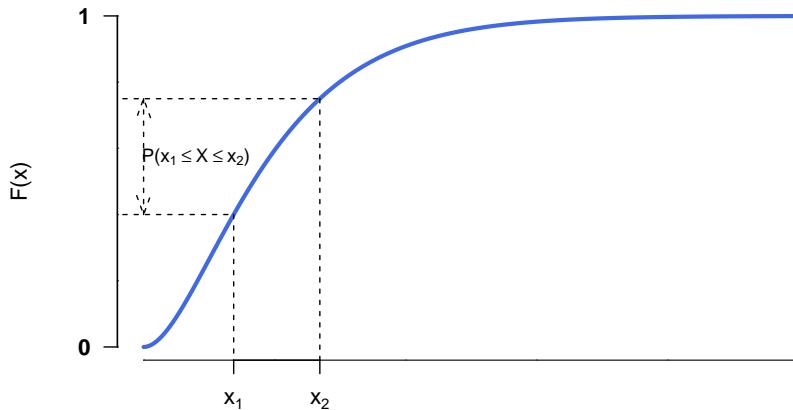
- 1) $\Gamma(1) = 1$
- 2) $\Gamma(\alpha + 1) = \alpha\Gamma(\alpha), \quad \forall \alpha > 0$
- 3) $\Gamma(\alpha) = \beta^\alpha \int_0^\infty x^{\alpha-1} e^{-\beta x} dx, \quad \forall \alpha, \beta > 0$
- 4) $\Gamma(n) = (n - 1)!, \quad n = 1, 2, \dots$
- 5) $\Gamma(1/2) = \sqrt{\pi}$

Funcția de repartiție a unei variabile aleatoare $X \sim \Gamma(\alpha, \beta)$ este dată de

$$F_X(x) = \int_{-\infty}^x f_X(t) dt = \frac{\beta^\alpha}{\Gamma(\alpha)} \int_{-\infty}^x t^{\alpha-1} e^{-\beta t} dt$$

și nu are o formulă explicită de calcul.

Functia de repartitie a repartitiei $\Gamma(\alpha, \beta)$



Observăm că repartiția $\Gamma(1, \lambda)$ coincide cu repartiția $\mathcal{E}(\lambda)$.

Media și varianța variabilei aleatoare X repartizate Gama de parametrii $\Gamma(\alpha, \beta)$ sunt egale cu

$$\mathbb{E}[X] = \frac{\alpha}{\beta}, \quad \text{Var}(X) = \frac{\alpha}{\beta^2}.$$



Arătați că media și varianța unei variabile aleatoare repartizate Gama de parametrii α și β sunt egale cu

$$\mathbb{E}[X] = \frac{\alpha}{\beta}, \quad \text{Var}(X) = \frac{\alpha}{\beta^2}.$$

În R putem să

- generăm observații independente din repartiția $\Gamma(\alpha, \beta)$ (e.g. $\alpha = 2, \beta = 2$)

```
rgamma(15, shape = 2, rate = 2)
[1] 0.6207897 1.6546379 0.4210210 0.8476985 0.2928765 0.6798413 1.1393160
[8] 1.0763898 1.4411221 0.9500644 0.7387296 0.4159926 0.8942659 0.8366199
[15] 0.9733579
```

- calculăm densitatea unei variabile aleatoare repartizate $\Gamma(\alpha, \beta)$ în diferite puncte

```
dgamma(seq(0, 5, length.out = 20), shape = 1, rate = 3)
[1] 3.000000e+00 1.362251e+00 6.185761e-01 2.808853e-01 1.275455e-01
[6] 5.791632e-02 2.629886e-02 1.194188e-02 5.422615e-03 2.462321e-03
[11] 1.118100e-03 5.077110e-04 2.305433e-04 1.046860e-04 4.753619e-05
[16] 2.158541e-05 9.801583e-06 4.450739e-06 2.021008e-06 9.177070e-07
```

- calculăm funcția de repartitie a unei variabile repartizate $\Gamma(\alpha, \beta)$ pentru diferite valori

```
pgamma(seq(0, 5, length.out = 15), shape = 1, rate = 3)
[1] 0.0000000 0.6574811 0.8826808 0.9598160 0.9862362 0.9952856 0.9983852
[8] 0.9994469 0.9998106 0.9999351 0.9999778 0.9999924 0.9999974 0.9999991
[15] 0.9999997
```

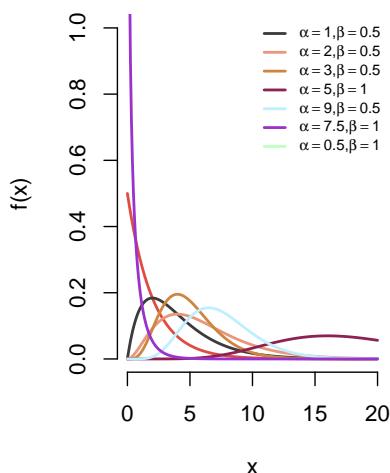
- calculăm cuantilele de ordin $p \in (0, 1)$

```
qgamma(c(0.01, 0.025, 0.05, 0.25, 0.5, 0.75, 0.95, 0.975, 0.99), shape = 1, rate = 3)
[1] 0.003350112 0.008439269 0.017097765 0.095894024 0.231049060 0.462098120
[7] 0.998577425 1.229626485 1.535056729
```

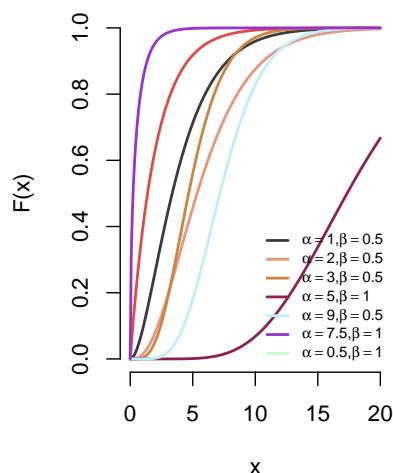


Fie X o variabilă aleatoare repartizată $\Gamma(\alpha, \beta)$. Pentru fiecare pereche de parametrii (α, β) din mulțimea $\{(1, 0.5), (2, 0.5), (3, 0.5), (5, 1), (9, 0.5), (7.5, 1), (0.5, 1)\}$ trasați pe același grafic densitățile repartițiilor Gama cu parametrii (α, β) . Adăugați legendele corespunzătoare. Aceeași cerință pentru funcțiile de repartitie.

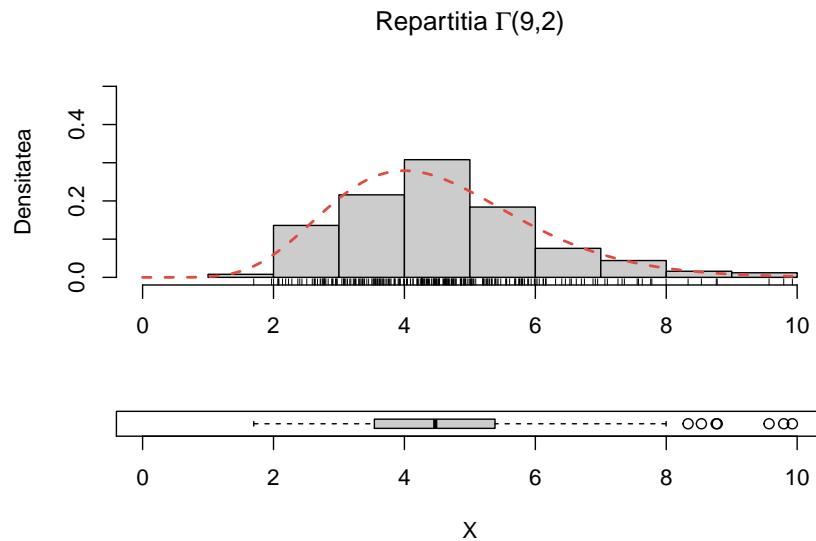
Densitatea



Functia de repartitie



Generați 250 de observații din repartitia $\Gamma(9, 2)$, trasați histograma acestora și suprapuneți densitatea repartiției date (vezi figura de mai jos).



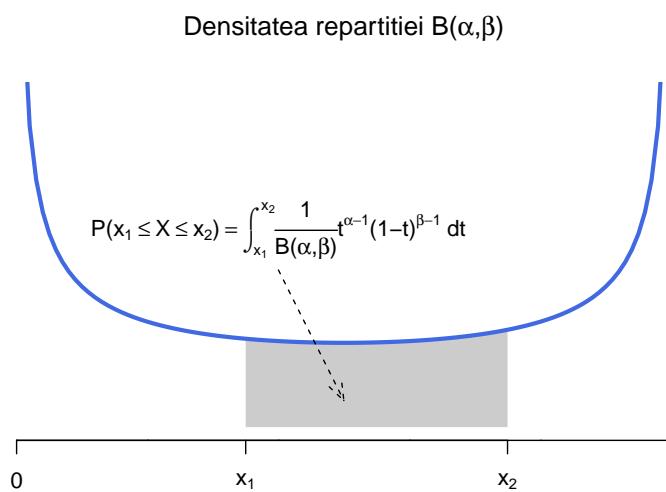
7 Repartitia Beta

Spunem că o variabilă aleatoare X este repartizată *Beta* de parametrii (α, β) , cu $\alpha, \beta > 0$, și se notează cu $X \sim B(\alpha, \beta)$, dacă densitatea ei are forma

$$f_X(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1}, \quad 0 \leq x \leq 1.$$

unde $B(\alpha, \beta)$ este funcția (Beta, numită și integrală Euler de primul tip) definită prin

$$B(\alpha, \beta) = \int_0^\infty x^{\alpha-1} (1-x)^{\beta-1} dx, \quad \forall \alpha, \beta > 0.$$





Arătați că funcția Beta $B(\alpha, \beta)$ verifică următoarele proprietăți:

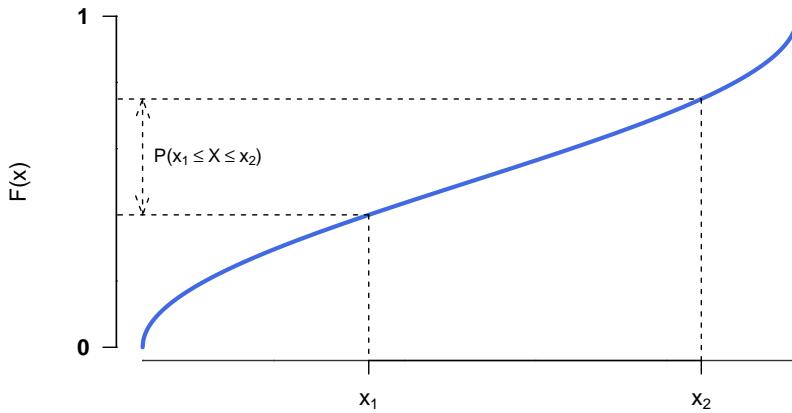
- 1) $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$
- 2) $B(\alpha, \beta) = B(\beta, \alpha)$
- 3) $B(\alpha, \beta) = B(\alpha, \beta+1) + B(\alpha+1, \beta)$
- 4) $B(\alpha+1, \beta) = B(\alpha, \beta) \frac{\alpha}{\alpha+\beta}$ și $B(\alpha, \beta+1) = B(\alpha, \beta) \frac{\beta}{\alpha+\beta}$.

Funcția de repartiție a unei variabile aleatoare $X \sim B(\alpha, \beta)$ este dată de

$$F_X(x) = \int_{-\infty}^x f_X(t) dt = \frac{1}{B(\alpha, \beta)} \int_{-\infty}^x t^{\alpha-1} (1-t)^{\beta-1} dt$$

și nu are o formulă explicită de calcul.

Functia de repartitie a repartitiei $B(\alpha, \beta)$



Observăm că repartiția $B(1, 1)$ coincide cu repartiția $\mathcal{U}([0, 1])$.

Media și varianța variabilei aleatoare X repartizate Gamma de parametrii $B(\alpha, \beta)$ sunt egale cu

$$\mathbb{E}[X] = \frac{\alpha}{\alpha + \beta}, \quad Var(X) = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}.$$

Observăm că $Var(X) \leq \mathbb{E}[X](1 - \mathbb{E}[X])$.



Arătați că media și varianța unei variabile aleatoare repartizate Beta de parametrii α și β sunt egale cu

$$\mathbb{E}[X] = \frac{\alpha}{\alpha + \beta}, \quad Var(X) = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}.$$

În R putem să

- generăm observații independente din repartiția $B(\alpha, \beta)$ (e.g. $\alpha = 2.5, \beta = 1$)

```
rbeta(15, shape1 = 2.5, shape2 = 1)
[1] 0.7945436 0.7609136 0.9265073 0.9309420 0.5621874 0.3664261 0.9694945
[8] 0.5804873 0.9504669 0.9115169 0.8457509 0.6717780 0.7213322 0.9738473
[15] 0.9791769
```

- calculăm densitatea unei variabile aleatoare repartizate $B(\alpha, \beta)$ în diferite puncte

```
dbeta(seq(0, 1, length.out = 20), shape1 = 1, shape2 = 3)
[1] 3.000000000 2.692520776 2.401662050 2.127423823 1.869806094 1.628808864
[7] 1.404432133 1.196675900 1.005540166 0.831024931 0.673130194 0.531855956
[13] 0.407202216 0.299168975 0.207756233 0.132963989 0.074792244 0.033240997
[19] 0.008310249 0.000000000
```

- calculăm funcția de repartiție a unei variabile repartizate $B(\alpha, \beta)$ pentru diferite valori

```
pbeta(seq(0, 1, length.out = 15), shape1 = 1, shape2 = 3)
[1] 0.0000000 0.1993440 0.3702624 0.5149417 0.6355685 0.7343294 0.8134111
[8] 0.8750000 0.9212828 0.9544461 0.9766764 0.9901603 0.9970845 0.9996356
[15] 1.0000000
```

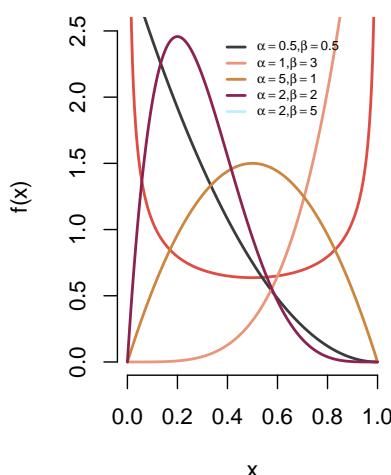
- calculăm cuantilele de ordin $p \in (0, 1)$

```
qbeta(c(0.01, 0.025, 0.05, 0.25, 0.5, 0.75, 0.95, 0.975, 0.99), shape1 = 1, shape2 = 3)
[1] 0.003344507 0.008403759 0.016952428 0.091439704 0.206299474 0.370039475
[7] 0.631596850 0.707598226 0.784556531
```

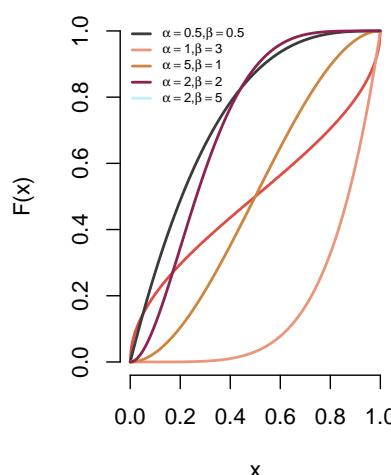


Fie X o variabilă aleatoare repartizată $B(\alpha, \beta)$. Pentru fiecare pereche de parametrii (α, β) din mulțimea $\{(0.5, 0.5), (1, 3), (5, 1), (2, 2), (2, 5)\}$ trasați pe același grafic densitățile repartițiilor Beta cu parametrii (α, β) . Adăugați legendele corespunzătoare. Aceeași cerință pentru funcțiile de repartiție.

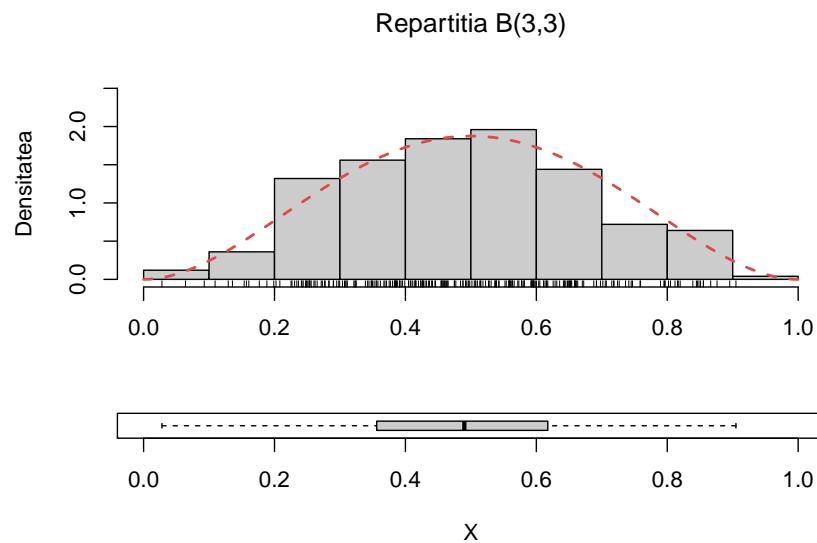
Densitatea



Functia de repartitie



Generați 250 de observații din repartiția $B(3, 3)$, trasați histograma acestora și suprapuneți densitatea repartiției date (vezi figura de mai jos).



Laborator 7

Legea Numerelor Mari și Teorema Limită Centrală

Obiectivul acestui laborator este de a prezenta noțiunea de convergență în probabilitate și convergență în repartiție precum și a *Legii Numerelor Mari* (versiunea slabă) și a *Teoremei Limită Centrale*.

1 Ilustrarea Legii Numerelor Mari

1.1 Convergență în probabilitate

Fie $X_n, n \geq 1$ și X variabile aleatoare definite pe câmpul de probabilitate $(\Omega, \mathcal{F}, \mathbb{P})$. Spunem că un sirul de variabile aleatoare $(X_n)_n$ converge în probabilitate la variabila aleatoare X , și notăm $X_n \xrightarrow{\mathbb{P}} X$, dacă pentru orice $\epsilon > 0$ are loc

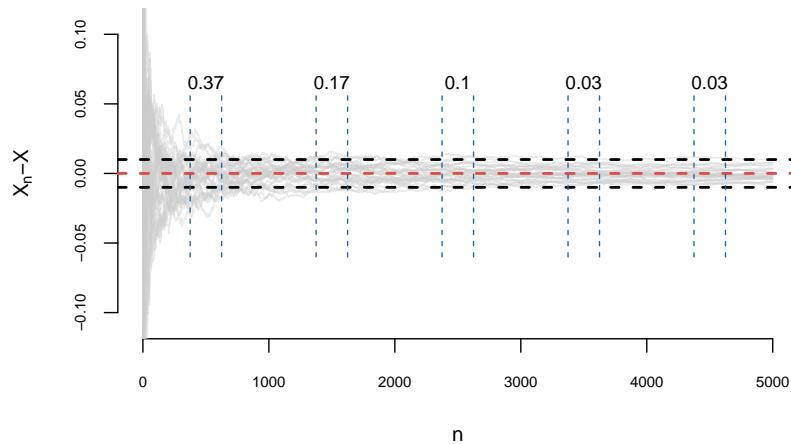
$$\mathbb{P}(|X_n - X| > \epsilon) \xrightarrow{n \rightarrow \infty} 0.$$

De asemenea putem observa că $X_n \xrightarrow{\mathbb{P}} X$ dacă și numai dacă $X_n - X \xrightarrow{\mathbb{P}} 0$. Pentru a ilustra grafic acest tip de convergență¹ vom aproxima probabilitatea $\mathbb{P}(A_n)$, unde $A_n = \{\omega \in \Omega \mid |X_n(\omega) - X(\omega)| > \epsilon\}$, folosind abordarea frecvenționistă. Aceasta presupune ca pentru n dat să considerăm $\omega_1, \dots, \omega_M \in \Omega$, M realizări ale experimentului (repetat în condiții identice) și să folosim aproximarea

$$\mathbb{P}(A_n) \approx p_n(M) = \frac{\#\{j \in \{1, \dots, M\} \mid |X_n(\omega_j) - X(\omega_j)| > \epsilon\}}{M}.$$

Concret, în figura de mai jos, considerăm $M = 30$ de repetiții ale experimentului (avem M curbe) cu $n = 5000$ de realizări ale unui sir de variabile aleatoare $X_k = \frac{Y_1 + \dots + Y_k}{k}$, cu Y_i independente și repartizate $\mathcal{U}[0, 1]$, $X = 0.5$ și $\epsilon = 0.01$. Pentru $i \in \{500, 1500, 2500, 3500, 4500\}$ am calculat și afisat frecvența de realizarea a evenimentului A_i (câte din cele M curbe sunt în afara benzii $[-\epsilon, \epsilon]$ pentru i , fixat). Observăm că $p_{500}(30) = 0.37$ și $p_{3500}(30) = 0.03$, convergența lui $p_n(M) \xrightarrow{n \rightarrow \infty} 0$ implicând convergență în probabilitate.

¹Pentru alte moduri de convergență și ilustrarea lor grafică se poate consulta lucrarea: Pierre LAFAYE DE MICHEAUX și Benoit LIQUET *Understanding Convergence Concepts: A Visual-Minded and Graphical Simulation-Based Approach*, The American Statistician, Vol. 63, No. 2, 2009



1.2 Legea numerelor mari (versiunea slabă)



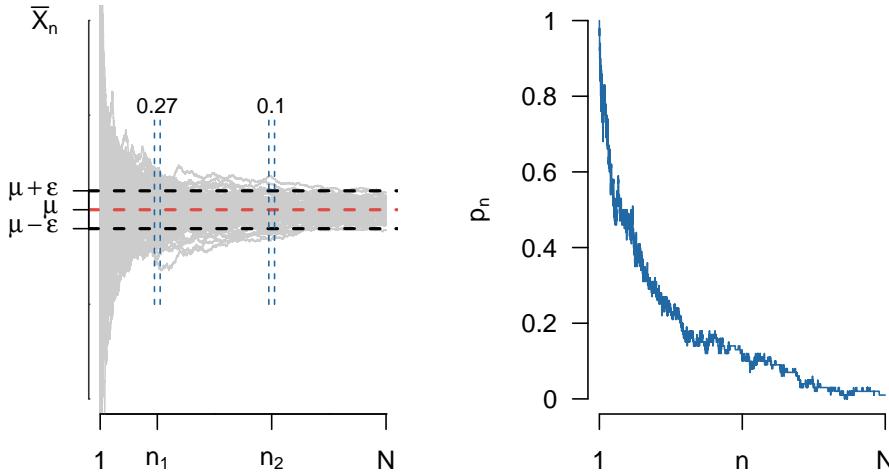
Fie X_1, X_2, \dots un sir de variabile aleatoare independente și identic repartizate, de medie $\mathbb{E}[X_1] = \mu < \infty$ și varianță $Var(X_1) = \sigma^2 < \infty$. Atunci $\forall \epsilon > 0$ avem

$$\mathbb{P} \left(\left| \frac{X_1 + \dots + X_n}{n} - \mu \right| > \epsilon \right) \xrightarrow{n \rightarrow \infty} 0$$

sau echivalent

$$\mathbb{P} \left(\left| \frac{X_1 + \dots + X_n}{n} - \mu \right| \leq \epsilon \right) \xrightarrow{n \rightarrow \infty} 1$$

Notând $\bar{X}_n = \frac{X_1 + \dots + X_n}{n}$, Legea numerelor mari (versiunea slabă) afirmă că $\bar{X}_n \xrightarrow{\mathbb{P}} \mu$. Figura de mai jos ilustrează această convergență pentru $M = 100$ de traекторii. În figura din dreapta este ilustrată evoluția probabilității p_n pentru $n \in \{1, 2, \dots, N\}$.



Să presupunem că primim o monedă și ni se spune că aceasta aterizează pe față cap în 48% din cazuri. Vrem să testăm această afirmație. Folosind *Legea numerelor mari* și știind că vrem să fim siguri în 95% din cazuri, ne întrebăm de câte ori trebuie să aruncăm moneda pentru a verifica afirmația?

Să presupunem că aruncăm moneda, independent, de n ori și fie X_i rezultatul obținut la cea de-a i -a aruncare: $X_i = 1$ dacă la i -a aruncare am obținut cap și $X_i = 0$ dacă am obținut pajură. Avem că variabilele aleatoare X_1, X_2, \dots, X_n sunt independente și repartizare $\mathcal{B}(p)$, cu $p = 0.48$ din ipoteză.

De asemenea, observăm că $\mathbb{E}[X_1] = \mu = 0.48$ și $Var(X_1) = \sigma^2 = p(1 - p) = 0.2496$. Pentru testarea monedei permitem o eroare $\epsilon = 0.02$ ceea ce înseamnă că probabilitatea ca moneda să aterizeze cap se află în intervalul $(0.46, 0.5)$. Din *Inegalitatea lui Cebîșev* avem că

$$\mathbb{P} \left(\left| \frac{X_1 + \dots + X_n}{n} - 0.48 \right| > 0.02 \right) \leq \frac{Var(X_1)}{n \times (0.02)^2},$$

de unde, având un grad de încredere de 95%, vrem să determinăm pe n pentru care

$$\frac{0.2496}{n \times (0.02)^2} = 0.05$$

ceea ce implică $n = 12480$.



Fie X_1, X_2, \dots, X_N, N v.a. i.i.d. de lege $\mathcal{U}([0, 1])$. Pentru $1 \leq n \leq N$, notăm cu $S_n = X_1 + X_2 + \dots + X_n$ sirul sumelor parțiale și μ media legii $\mathcal{U}([0, 1])$. Trasați pe același grafic funcția $n \rightarrow \bar{X}_n = \frac{S_n}{n}$ pentru $n = 1, \dots, N$ și dreapta de ecuație $y = \mu$. Faceți același lucru pentru legea normală $\mathcal{N}(2, 1)$.

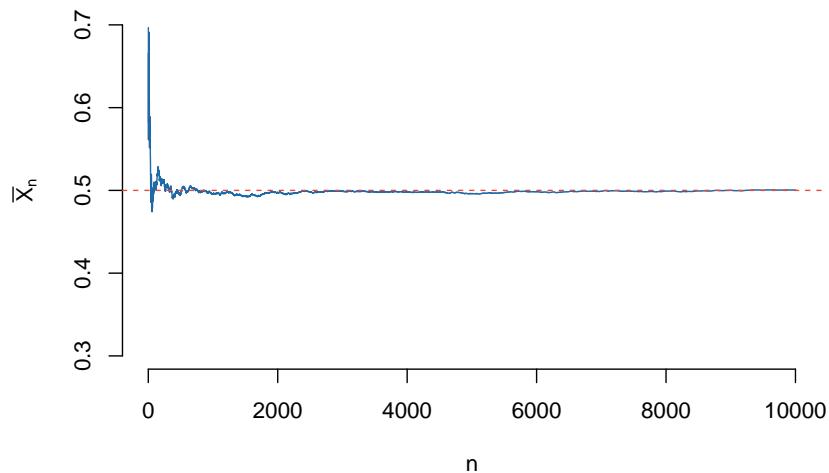
În cazul în care v.a. X_1, X_2, \dots, X_N sunt repartizate uniform $\mathcal{U}([0, 1])$ (deci media este $\mu = \frac{1}{2}$) avem:

```
n = 10000

# Pentru legea uniformă folosim comanda runif
# Pentru calculul sumelor parțiale putem folosi functia cumsum

y1 = cumsum(runif(n))
y1 = y1/(1:n)
mu1 = 1/2 # media uniformei pe [0,1]

# trăsăm graficul
mar.default <- c(5,4,4,2) + 0.1
par(mar = mar.default + c(0, 0.3, 0, 0))
plot(1:n, y1, type = "l",
      col= myblue, xlab = "n",
      ylab = expression(bar(X)[n]),
      bty = "n",
      ylim = c(0.3,0.7))
abline(h = mu1, col = myred, lty= "dashed") # adaugam linia orizontală
```



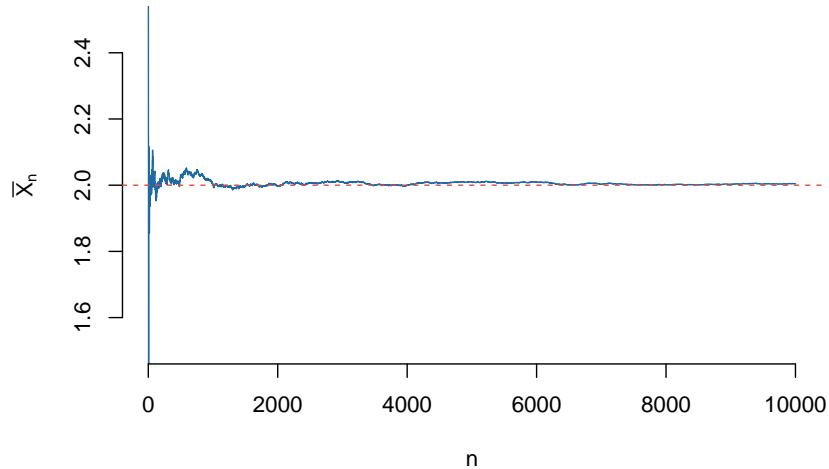
În cazul în care v.a. X_1, X_2, \dots, X_N sunt normale de parametrii $\mathcal{N}(2, 1)$ (deci media este $\mu = 2$) avem:

```
# Folosim același număr de variabile n

# Pentru legea normală folosim comanda rnorm
# Pentru calculul sumelor parțiale putem folosi functia cumsum
y2 = cumsum(rnorm(n, mean = 2, sd = 1))
y2 = y2/(1:n)
mu2 = 2 # media normalei  $N(2, 1)$ 

# facem graficul
mar.default <- c(5,4,4,2) + 0.1
par(mar = mar.default + c(0, 0.3, 0, 0))
plot(1:n, y2, type = "l",
      col= myblue, xlab = "n",
      ylab = expression(bar(X)[n]),
```

```
bty = "n",
ylim = c(1.5, 2.5)
abline(h = mu2, col = myred, lty= "dashed") # adaugam linia orizontala
```



Construiți o funcție care să vă permită generarea a m esantioane de volum n dintr-o populație normală de medie μ și varianță σ^2 dată. Ilustrați grafic cu ajutorul unui boxplot cum variază diferența dintre media aritmetică (media esantionului \bar{X}_n) și media teoretică pentru $m = 100$ și diferite volume ale esantionului $n \in \{10, 100, 1000, 10000\}$. Se consideră $\mu = 1$ și $\sigma^2 = 1$.

Următoarea funcție verifică cerința din problema (normal.mean = μ , normal.sd = σ , num.samp = m și samp.size = n). Să observăm că am folosit funcția rowMeans pentru a calcula media fiecărui esantion (media pe liniile matricii de observații).

```
normalSampleMean <- function(normal.mean, normal.sd, num.samp, samp.size) {
  # generam matricea de observatii
  x = matrix(rnorm(n = num.samp * samp.size, mean = normal.mean, sd = normal.sd),
             nrow = num.samp, ncol = samp.size)

  # calculam media esantionului pentru fiecare esantion
  x.mean = rowMeans(x)

  return(x.mean)
}
```

Pentru a ilustra grafic să considerăm o populație $\mathcal{N}(1,1)$ și pentru talie a esantionului, $n \in \{10, 100, 1000, 10000\}$, să calculăm \bar{X}_n corespunzător (aici am folosit funcția sapply - a se vedea laboratorul 3).

```
# date de intrare
normal.mean = 1
normal.sd = 1
true.mean = normal.mean

# marimea esantioanelor
```

```
samp.sizes = c(10, 100, 1000, 10000)

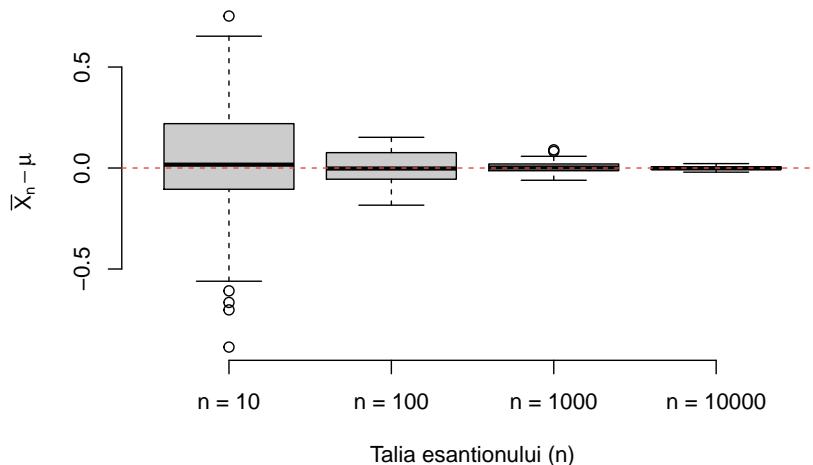
names(samp.sizes) = paste0("n = ", samp.sizes)

# numarul de esantioane
num.samp = 100

# calculul mediei de selectie pentru fiecare esantion
x.mean = sapply(samp.sizes, normalSampleMean, num.samp = num.samp,
                 normal.mean = normal.mean, normal.sd = normal.sd)

# ilustrarea grafica
mar.default <- c(5,4,4,2) + 0.1
par(mar = mar.default + c(0, 0.2, 0, 0), bty = "n")
boxplot(x.mean - true.mean,
        xlab = "Talia esantionului (n)",
        ylab = expression(bar(X)[n] - mu),
        col = "gray80",
        bty = "n")

abline(h = 0, lty = 2, col = myred)
```



Din boxplot-ul de mai sus observăm că pe măsură ce creștem talia eșantionului media boxplot-ului se duce spre 0 ceea ce justifică enunțul *Legii Numerelor Mari*, și anume că media eșantionului converge la media populației (media teoretică). De asemenea putem observa că și varianta scade (gradul de împrăștie scade) odată cu creșterea numărului de observații.



Utilizați *Legea Numerelor Mari* pentru a aproxima integrala următoare

$$I = \int_0^1 e^x \sin(2x) \cos(2x) dx.$$

Calculați de asemenea valoarea exactă I a acesteia și comparați-o cu aproximarea găsită.

Fie U_1, U_2, \dots, U_n un sir de v.a. i.i.d. repartizare uniform pe $[0, 1]$. Cum g este o funcție continuă atunci $g(U_1), g(U_2), \dots, g(U_n)$ sunt variabile aleatoare i.i.d. și aplicând *Legea Numerelor Mari* obținem

$$g_n = \frac{1}{n} \sum_{i=1}^n g(U_i) \xrightarrow{\mathbb{P}} \mathbb{E}[g(U_1)] = \int_0^1 g(x)dx.$$

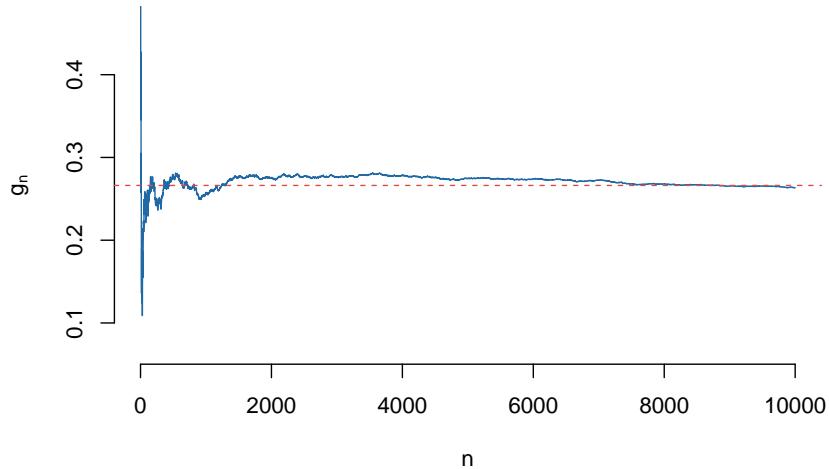
Pentru a calcula integrala numeric vom folosi funcția **integrate** (trebuie observat că această integrală se poate calcula ușor și exact prin integrare prin părți). Următorul script ne dă valoare numerică și aproximarea obținută cu ajutorul metodei Monte Carlo pentru integrale $\int_0^1 g(x)dx$:

```
myfun=function(x){  
  y = exp(x)*sin(2*x)*cos(2*x);  
  return(y);  
}  
  
# calculul integralei cu metode numerice  
I = integrate(myfun,0,1) # raspunsul este o lista si oprim prima valoare  
I = I[[1]]  
  
# calculul integralei cu ajutorul metodei Monte Carlo  
n = 10000  
  
u = runif(n) # generarea sirului U_n  
z = myfun(u) # calcularea sirului g_n  
  
I2 = sum(z)/n # aproximarea MC
```

Obținem că valoarea numerică a lui I este 0.2662 iar cea obținută cu ajutorul metodei Monte Carlo este 0.2673.

Avem următoarea ilustrare grafică a convergenței metodei Monte Carlo:

```
# graficul  
gn = myfun(runif(n))  
gn = cumsum(gn)/(1:n) # calculul lui g_n  
  
plot(1:n, gn, type = "l",  
      col = myblue, xlab = "n",  
      ylab = expression(g[n]),  
      bty = "n",  
      ylim = c(I-0.2, I+0.2))  
abline(h = I, lty = "dashed", col = myred)
```



2 Ilustrarea Teoremei Limită Centrală



Fie X_1, X_2, \dots un sir de variabile aleatoare independente și identic repartizate, de medie $\mathbb{E}[X_1] = \mu < \infty$ și varianță $Var(X_1) = \sigma^2 < \infty$. Atunci, notând $S_n = X_1 + \dots + X_n$, avem

$$\mathbb{P}\left(\frac{S_n - \mathbb{E}[S_n]}{\sqrt{Var(S_n)}} \leq x\right) = \mathbb{P}\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} \leq x\right) \xrightarrow{n \rightarrow \infty} \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt, \quad \forall x \in \mathbb{R}.$$

Echivalent, dacă notăm media eșantionului cu $\bar{X}_n = \frac{S_n}{n}$, atunci

$$\mathbb{P}\left(\sqrt{n}\frac{\bar{X}_n - \mu}{\sigma} \leq x\right) \xrightarrow{n \rightarrow \infty} \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt, \quad \forall x \in \mathbb{R}.$$



Să presupunem că primim o monedă și ni se spune că aceasta aterizează pe față cap în 48% din cazuri. Vrem să testăm această afirmație. Folosind *Teorema Limită Centrală* și știind că vrem să fim siguri în 95% din cazuri, ne întrebăm de câte ori trebuie să aruncăm moneda pentru a verifica afirmația? Comparați răspunsul cu cel din exercițiul în care am folosit *LNM*, de mai sus.

Folosind aceleași notări ca și în exercițiul din secțiunea de mai sus și notând în plus $S_n = X_1 + \dots + X_n$, avem

$$\begin{aligned} \mathbb{P}\left(\frac{S_n}{n} < 0.5\right) &= \mathbb{P}\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} < \frac{(0.5 - \mu)\sqrt{n}}{\sigma}\right) = \mathbb{P}\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} < \frac{0.02\sqrt{n}}{\sqrt{0.2496}}\right) \\ &= \mathbb{P}\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} < 0.04\sqrt{n}\right) \approx \Phi(0.04\sqrt{n}) \geq 0.95 \end{aligned}$$

Prin urmare, $(0.04\sqrt{n} \geq 1.645)$ de unde $n = 1692$. Putem observa că rezultatul obținut prin aplicarea *Teoremei Limită Centrală* este mai precis decât cel obținut prin aplicarea *Legii numerelor mari*.



Fie $(X_n)_{n \geq 1}$ un sir de v.a. i.i.d. de lege $\mathcal{E}(1)$. Pentru toti n , notăm cu $S_n = X_1 + X_2 + \cdots + X_n$ sirul sumelor parțiale, μ și σ^2 reprezentând media și respectiv varianța legii $\mathcal{E}(1)$. Teorema Limită Centrală afirmă că dacă n este mare atunci v.a.

$$\frac{S_n - n\mu}{\sqrt{n}\sigma}$$

are aproximativ aceeași distribuție ca și legea normală $\mathcal{N}(0, 1)$. Ilustrați această convergență în distribuție cu ajutorul unei histograme. Suprapuneți peste această histogramă densitatea legii $\mathcal{N}(0, 1)$.

Stim că media unei v.a. distribuite exponențial de parametru λ , $\mathcal{E}(\lambda)$ este $\mu = \frac{1}{\lambda}$ iar varianța acesteia este $\sigma^2 = \frac{1}{\lambda^2}$. Pentru fiecare valoare a lui i de la 1 la N calculăm raportul $\frac{S_n - n\mu}{\sigma\sqrt{n}}$ (cu alte cuvinte repetăm experimentul de N ori):

```
N = 1000 # alegem numarul de repetitii ale experimentului
n = 1000 # alegem n pentru care folosim aproximarea normala

lambda = 1 # parametrul legii E(1)

mu = 1/lambda # media
sigma = 1/lambda # abaterea standard

s = rep(0,N) # initializam sirul sumelor parțiale

for (i in 1:N){
  x = rexp(n, rate = lambda) # generam variabilele exponentiale
  s[i] = (sum(x)-n*mu)/(sigma*sqrt(n)) # calculam raportul
}

}
```

Continuăm prin trasarea histogramei cerute și adăugăm la grafic densitatea legii normale $\mathcal{N}(0, 1)$:

```
# trasam histograma
# pentru mai multe opțiuni latex: ?plotmath
hist(s, main = expression(paste("Histograma raportului ",
                                frac(S[n]-n%*%mu,sigma%*%sqrt(n)))),
      prob = TRUE,
      col = "grey80", # Culoarea de umplere
      border = "grey20",
      xlim = c(-4,4),
      cex.main=0.75,
      cex.lab = 0.75,
      cex.axis = 0.75,
      xlab = "",
      ylab = "Densitatea")

# adaugam densitatea normalei N(0, 1)
x1 = seq(-4,4,by=0.1)
y1 = dnorm(x1, mean = 0, sd = 1)
lines(x1, y1, col = myred, lwd = 2, lty = 2)
```

