INIŢIERE în R

Pentru persoane cu pregătire matematică

Maria Miroiu

Viorel Petrehuş

Gheorghiţă Zbăganu

Introducere

Mediul de programare "R" face parte dintr-un "proiect" mai mare de a produce software gratuit. Se poate spune că el este un dialect al altui limbaj, numit „S", creat în 1976 la laboratoarele Bell pentru a se uşura calculele statistice. La început s-a folosit o sintaxă similară Fortran-ului, dar în 1988 a fost rescris în C. Spre deosebire de „R", „S" nu este un software gratuit. Prima variantă de „R" a fost creată în Noua Zeelandă, 1991, de Ross Ihaka şi Robert Gentleman [1]. În 1993 s-a făcut lansarea oficială a proiectului, conceput la început ca un software statistic non – gratuit, aşa cum mai erau şi altele - SAS, SPSS, STATISTICA. În 1995 autorii s-au hotărît să îl facă un software gratuit. Pentru a controla nucleul de bază al limbajului, s-a creat în 1997 aşa numitul „R Core Group", care controlează codul sursă. În 2000 s-a lansat versiunea R.1.0.0 iar în 2008 s-a ajuns la versiunea R.2.7.2. Noi am folosit versiunea R.2.12.2, apărută pe 25 februarie 2011. Există zeci de site-uri de unde se poate descărca sistemul R şi pachete adiţionale..

Părintele limbajului , Ross Ihaka scria în 1998 [2]:

„În august 1993 Robert Gentleman şi cu mine am plasat copii ale R-ului la Statlib şi am făcut un mic grup pe s-news. Unii ne-au descărcat copiile şi s-au oferit să colaboreze. Cel mai de nădejde a fost Martin Maehler de la ETH Zurich care ne-a convins să lansăm codul sursă ca un software gratuit. La început am avut ezitări, dar în iunie 1995 am convenit să punem la dispoziţia tuturor codul sursă în condiţiile fundaţiei FRF [3] *. "*

S-a creat astfel o impresionantă comunitate a utilizatorilor şi dezvoltatorilor limbajului. Toţi voluntari.

Decizia de a face R un software gratuit, continuă Ihaka *„ne-a făcut să ne propunem ţeluri mai mari decît un simplu soft statistic cu care să ne putem preda cursurile mai bine. Ne-a dat acces la o masă de indivizi foarte talentaţi doritori să se implice în proiect. Poate unul din cele mai bune lucruri care ni s-a întîmplat a fost şansa de lucra cu asemenea oameni strălucitori* ". Şi îşi exprima optimismul : „ *Se pare că R a ajuns la punctul în care este destul de stabil pentru a fi folosit de utilizatori – cel puţin sub Unix. Sperăm ca în 1999 să putem lansa varianta completă R.1.0 în cadrul fundaţiei Free Software* "

De ce software gratuit? Ce avantaje avem?

În primul rând, firește, nu dăm bani pe el. Dar mai sunt şi alte avantaje, cum arată un entuziast al R-ului, Roger. D. Peng [4] "*Un software gratuit vă garantează libertatea de a folosi programele lui în orice scop (libertatea 0)*; *libertatea de a studia cum funcţionează programele şi de a le adapta la nevoile voastre. Accesul la codul*

[1] http://www.biostat.jhsph.edu/~rpeng/biostat776/lecture1.pdf
[2] http://cran.r-project.org/doc/html/interface98-paper/paper.html
[3] http://www.fsf.org/
[4] http://www.biostat.jhsph.edu/~rpeng/

sursă (*libertatea 1*). *Libertatea de a redistribui copiile pe care le aveţi oricui, aşa încât să îţi poţi ajuta vecinul* (*libertatea* 2); *libertatea de a distribui oricui îmbunătăţirile făcute de tine*".

Sigur că există şi dezavantaje. Nu există garanţie că nu s-a strecurat pe undeva vreo greşeală. Trebuie să ştii matematică dacă vrei să îl foloseşti în siguranţă. Nu este prietenos, aşa cum este de exemplu Excel-ul. Ca să poţi folosi help-ul există chiar o instrucţiune help.search; nu are un help uşor accesibil. Dar nici matematica nu este uşor accesibilă, iar cartea aceasta este destinată unor cititori cu pregătire matematică – fie ei studenţi la matematică, informatică, politehnică sau construcţii [5].

Cartea este intenţionată a fi o iniţiere în R. Sistemul R este imens – există deja mai mult de 1000 de pachete. Noi vrem să prezentăm doar pachetul de bază, acela care se descarcă de la CRAN (http://cran.r-project.org). Acesta este suficient pentru a putea rula R şi conţine funcţiile fundamentale ale limbajului.

Utilizatorul poate fi descurajat de cantitatea imensă de documentaţie care există în legătură cu R. Numai editura Springer a publicat, începând cu 2004, cîteva zeci de cărţi dedicate acestui limbaj, scrise pentru cititori cu pregătire matematică. O listă aproape completă se poate găsi la http://www.r-project.org/doc/bib/R-books.html[6].

Cunoscătorii apreciază R-ul pentru facilităţile sale grafice[7].

Pentru un începător în R există cel puţin trei mari dificultăţi.

Prima este legată de sintaxă. O paranteză greşit pusă, sau o paranteză [.] în loc de (.) sau de {.} poate face necazuri foarte mari. Mesajele de eroare sunt sibilinice.

A doua este legată de scripturi. După cum se va vedea, utilizatorul poate da instrucţiunile ori în linie de comandă, ori în pachete de instrucţiuni numite scripturi. Un script este un fişier cu extensia .R. Ca să deschizi un script, însă trebuie să ştii unde e salvat. Un script nu este un program sau o singură subrutină, el poate să conţină mai multe asemenea programe. Pentru a-l rula, este suficient să îl deschizi, să cauţi paragraful care te interesează, să îl selectezi şi apoi să tastezi Ctrl+R. Se va executa exact ceea ce doreşti, dar asta se învaţă mai greu. Pe de altă parte, este bine ca proiectele diferite să fie puse în directoare diferite, fiindcă R păstrează în memorie variabilele ultimei sesiuni de lucru şi probabilitatea de a da acelaşi nume unor variabile diferite este foarte mare.

În fine, a treia dificultate este legată de introducerea datelor din fişiere. Începătorul poate fi foarte frustrat încercând să copieze un tabel din Excel într-o matrice din R: dacă tabelul din Excel nu este o matrice – de exemplu daca are celule lăsate necompletate atunci R dă mesaje de eroare neinteligibile cum ar fi „`Error in scan (file, what, nmax, sep, dec, quote, skip, nlines, na.strings, : line 3 did not have 3 elements`". Acelaşi lucru îl putem spune şi despre salvarea în fişiere a rezultatelor obţinute prin calcule: a transfera un vector din R în Excel este o operaţie care poate provoca mare frustrare utilizatorului neavertizat, aşa cum se va vedea la capitolul doi.

---

[5] Chiar şi persoane fără o pregătire matematică specială pot folosi sitemul R dacă au un minimum de pregătire statistică, de exemplu studenţi la biologie, sociologie, agronomie, etc.

[6] Manualul complet "R Reference Manual" care este conţinut în kitul de instalare R are 1884 de pagini din care 60 pagini reprezintă indexul cu peste 4000 de funcţii puse la dispoziţia utilizatorilor care ştiu să le folosească.

[7] Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

Odată depăşite aceste dificultăţi inerente începutului, apar şi satisfacţiile. De exemplu a găsi minimul, maximul, mediana unui şir de date, ai face histograma, a-l sorta devin o joacă de copii.

În primele patru capitole se prezintă principalele posibilităţi de utilizare a mediului R în regim de comandă. În capitolul al cincilea se prezintă posibilităţile de programare în R, iar în capitolul al şaselea sunt prezentate aplicaţii programate în R din diverse domenii care au legătură cu statistica. În ultimul capitol se prezintă teste statistice şi serii de timp.

Maria Miroiu a scris capitolele 1,2 şi 7.1; Viorel Petrehuş a scris capitolele 3,4,5 şi 7.2 iar Gheorghiţă Zbăganu a scris capitolul 6.

Variantele de R utilizate au fost: 2.12 şi 2.14.

Accentuăm: cartea se adresează utilizatorilor începători ai sistemului R, dar care au o minimă pregătire în matematică, îndeosebi în domeniul statisticii. Se presupune că există la cititor o oarecare familiaritate cu termenii de variabilă aleatoare, repartiţie, test statisctic. Este important să accentuăm însă că sistemul R poate fi utilizat în mai multe domenii ale matematicii aplicate, nu numai în statistică, de exemplu pentru rezolvarea ecuaţiilor şi sistemelor de ecuaţii diferenţiale.

În final avem de adăugat următoarele: cartea a fost elaborată în cadrul proiectului POSDRU/56/1.2/S/32768, "Formarea cadrelor didactice universitare şi a studenţilor în domeniul utilizării unor instrumente moderne de predare-învăţare-evaluare pentru disciplinele matematice, în vederea creării de competenţe performante şi practice pentru piaţa muncii".

Finanţat din Fondul Social European şi implementat de către Ministerul Educaţiei, Cercetării, Tineretului şi Sportului, în colaborare cu The Red Point, Oameni şi Companii, Universitatea din Bucureşti, Universitatea Tehnică de Construcţii din Bucureşti, Universitatea „Politehnica" din Bucureşti, Universitatea din Piteşti, Universitatea Tehnică „Gheorghe Asachi" din Iaşi, Universitatea de Vest din Timişoara, Universitatea „Dunărea de Jos" din Galaţi, Universitatea Tehnică din Cluj-Napoca, Universitatea "1 Decembrie 1918" din Alba-Iulia, proiectul contribuie în mod direct la realizarea obiectivului general al Programului Operaţional Sectorial de Dezvoltare a Resurselor Umane – POSDRU şi se înscrie în domeniul major de intervenţie 1.2 Calitate în învăţământul superior.

Proiectul are ca obiectiv adaptarea programelor de studii ale disciplinelor matematice la cerinţele pieţei muncii şi crearea de mecanisme şi instrumente de extindere a oportunităţilor de învăţare.

Obiectivele de interes ale proiectului sunt evaluarea nevoilor educaţionale ale cadrelor didactice şi studenţilor legate de utilizarea matematicii în învăţământul superior, masterate şi doctorate precum şi analizarea eficacităţii şi relevanţei curriculelor actuale la nivel de performanţă şi eficienţă, în vederea dezvoltării de cunoştinţe şi competenţe pentru studenţii care învaţă discipline matematice în universităţi. Obiectivele specifice la care răspunde materialul de faţă sunt: dezvoltarea şi armonizarea curriculelor disciplinelor matematice, conform exigenţelor moderne; elaborarea şi implementarea unui program de formare a cadrelor didactice şi a studenţilor interesaţi din universităţile partenere, bazat pe dezvoltarea şi armonizarea de curriculum; crearea unei baze de resurse inovative şi funcţionale pentru predarea şi învăţarea în disciplinele matematice pentru învăţământul universitar. sunt obiectivele specifice care au ca raspuns materialul de faţă.

Formarea de competenţe cheie de matematică şi informatică presupune crearea de abilităţi de care fiecare individ are nevoie pentru dezvoltarea personală, incluziune socială şi inserţie pe piaţa muncii. Se poate constata însă că programele disciplinelor de matematică nu au întotdeauna în vedere identificarea şi sprijinirea elevilor şi studenţilor potenţial talentaţi la matematică.

În acest context, analiza flexibilităţii curriculei, însoţită de analiza metodelor şi instrumentelor folosite pentru identificarea şi motivarea studenţilor talentaţi la matematică ar putea răspunde deopotrivă cerinţelor de masă, cât şi celor de elită.

Viziunea pe termen lung a acestui proiect preconizează determinarea unor schimbări în abordarea fenomenului matematic pe mai multe planuri: informarea unui număr cât mai mare de membri ai societăţii în legătură cu rolul şi locul matematicii în educaţia de bază în instrucţie şi în descoperirile ştiinţifice menite să îmbunătăţească calitatea vieţii, inclusiv popularizarea unor mari descoperiri tehnice, şi nu numai, în care matematica cea mai avansată a jucat un rol hotărâtor. De asemenea, se urmăreşte evidenţierea a noi motivaţii solide pentru învăţarea şi studiul matematicii la nivelele de bază şi la nivel de performanţă; stimularea creativităţii şi formarea la viitorii cercetători matematicieni a unei atitudini deschise faţă de însuşirea aspectelor specifice din alte ştiinţe, în scopul participării cu succes în echipe mixte de cercetare sau a abordării unei cercetări inter şi multi disciplinare; identificarea unor forme de pregătire adecvată de matematică pentru viitorii studenţi ai disciplinelor matematice, în scopul utilizării la nivel de performanţă a aparatului matematic în construirea unei cariere profesionale.

CUPRINS

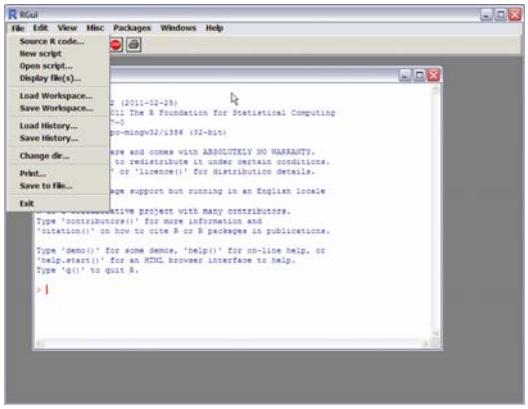
Capitolul 1
Preliminarii

R este un mediu integrat de facilităţi software destinat în special analizei statistice şi reprezentării grafice a datelor. Acesta include un limbaj simplu de programare, limbajul S, care permite utilizarea pachetelor existente şi construirea de pachete noi de programe destinate analizei statistice şi reprezentării grafice a datelor.

R este un mediu de dezvolare gratuit distribuit sub licenţă publică generală GNU. A fost iniţial scris de Ross Ihaka şi Robert Gentleman de la departmentul de statistică de la Universittea din Auckland, New Zealand. Din 1997 s-a format un grup principal ("R Core Team") care poate modifica variante ale R-ului. R-ul s-a dezvoltat rapid şi s-a extins la o mare colecţie de pachete de programe.

Site-ul oficial de unde se pot obţine gratuit versiuni compilate de R pentru OS X, Linux şi Windows este http://www.r-project.org. Tot aici se pot găsi şi resurse pentru instalarea şi utilizarea R-ului. Cea mai recentă versiune este 2.13.2 disponibilă din 30.09.2011. Odată cu R se descarcă şi o serie de pachete de programe (numite şi pachete standard), dar sunt disponibile şi alte pachete de programe R prin intermediul CRAN (Comprehensive R Archive Network – http://CRAN.R-project.org) care conţine diferite variante ale sistemului R şi ale pachetelor de programe R.

Kitul de instalare R vine cu manuale în care este descris în detaliu modul de funcţionare al programului. De asemenea se găsesc cărţi specializate pe diverse domenii unde se utilizează statistica şi unde se arată cum se poate utiliza R pentru calculele necesare.



La lansarea în execuţie a R-ului apare o fereastră ca cea alăturată şi se poate lucra direct cu R în fereastra consolă interioară.

În meniul „Packages" se găsesc opţiuni de instalare separată a unor pachete suplimentare de programe specializate pe diverse domenii prin care posibilităţile lui R se pot extinde foarte mult.

Lucrul cu R se poate face

- fie în regim interactiv: în fereastra Rconsole, după prompter-ul ‚>', se scrie o comandă acceptată de R, apoi se apasă tasta Enter şi rezultatul apare imediat mai jos în aceeaşi ferestră,
- fie prin script-uri, în regim de programare: comenzile de calcul sunt grupate într-un fişier cu extensia .r care poate fi apelat în mai multe moduri. Un fişier script poate fi scris în orice editor de text, dar R are un editor ce se apelează din meniul *„File"* prin *„Open script..." sau „New script"*. Fişierul script poate fi executat după ce se încarcă în mediul R prin *File → Open script...* sau se poate folosi direct funcţia `source()`, de exemplu, presupunând că fişierul script se numeşte *test.r* şi se află în directorul de lucru curent, comanda scrisă la prompt: `>source("test.r")` va avea ca efect încărcarea în spaţiul de lucru a fişierului şi executarea comenzilor conţinute în fişierul „test.r".

R dispune de un mecanism de rechemare a comenzilor executate în linia de comandă în fereastra RConsole. Prin apăsarea tastei de deplasare pe verticală în sus (↑), se pot obţine comenzile scrise anterior. Acestea pot fi reexecutate sau modificate şi apoi executate.

Toate datele citite sau salvate prin comenzi R sunt asociate cu directorul curent care poate fi schimbat prin *File → Change dir...* la calea dorită de utilizator.

Directorul curent (de lucru) poate fi aflat prin comanda `getwd()` şi poate fi fixat şi prin comanda `setwd("directorul de lucru dorit")`.

R dispune de funcţii care permit obţinerea rapidă de informaţii referitoare la funcţiile şi facilităţile sistemului R instalat pe calculatorul propriu. Pentru a obţine informaţii despre facilităţile oferite de functia `help()`, din R se execută comanda

```
> help.start()
```

Se va lansa o pagină web cu o serie link-uri care permit accesarea informaţiilor legate de sistemul R.

Pentru a obţine informaţii despre o anumită funcţie R, de exemplu `plot`, se execută comanda

```
> help(plot)
```

sau comanda alternativă

```
> ?plot
```

Aceasta va deschide un Web browser cu o pagină în care se afişează informaţii referitoare la funcţia `plot`.

Pentru funcţiile specificate prin cuvinte cheie, ca de exemplu, `if`, `while`, `function`, argumentele trebuie incluse între ghilimele:

```
> help("while")
```

Pentru a afişa fişiere, titluri, obiecte, etc. ce conţin un anumit şir de caractere, se foloseşte comanda

```
> help.search(şir de caractere, ….)
```

De exemplu:

```
> help.search("poisson")
```

sau echivalent

```
> ??poisson
```

conduce la afişarea următoarelor rezultate

```
Help files with alias or concept or title matching
'poisson' using fuzzy matching:

boot::poisons           Animal Survival Times
stats::family           Family Objects for Models
stats::Poisson          The Poisson Distribution
stats::poisson.test     Exact Poisson tests
```

Dacă dorim să încărcăm un anumit pachet de programe R, mai întâi trebuie descărcat în subdirectorul library al directorului R (de exemplu în: *C:\Program Files\R\R-2.13.0\library*). Pentru încărcarea de exemplu a pachetului *graphics* (the R graphics package), putem folosi funcţia

```
> library(graphics)
```

Dacă dorim să aflăm informaţii despre un anumit pachet de programe, de exemplu utils (the R Utils package)

```
> library(help="utils").
```

Pentru vedea ce pachete de programe R sunt disponibile în versiunea curentă R, se poate folosi funcţia

```
> library()
```

De exemplu, pahetele disponibile în versiunea RR-2.13.0 sunt:

```
base        The R Base Package
boot        Bootstrap R (S-Plus) Functions (Canty)
class       Functions for Classification
cluster     Cluster Analysis Extended Rousseeuw et al.
codetools   Code Analysis Tools for R
compiler    The R Compiler Package
datasets    The R Datasets Package
foreign     Read Data Stored by Minitab, S, SAS, SPSS,
                  Stata, Systat, dBase, ...
graphics    The R Graphics Package
grDevices   The R Graphics Devices and Support for Colours
                  and Fonts
grid        The Grid Graphics Package
KernSmooth  Functions for kernel smoothing for Wand &
                   Jones (1995)
lattice     Lattice Graphics
MASS        Support Functions and Datasets for Venables
                   and Ripley's MASS
Matrix      Sparse and Dense Matrix Classes and Methods
methods     Formal Methods and Classes
mgcv        GAMs with GCV/AIC/REML smoothness estimation
                  and GAMMs by PQL
nlme        Linear and Nonlinear Mixed Effects Models
nnet        Feed-forward Neural Networks and Multinomial
                 Log-Linear Models
rpart       Recursive Partitioning
spatial     Functions for Kriging and Point Pattern
                  Analysis
splines     Regression Spline Functions and Classes
stats       The R Stats Package
stats4      Statistical Functions using S4 Classes
survival    Survival analysis, including penalised
                  likelihood.
Tcltk       Tcl/Tk Interface
tools       Tools for Package Development
utils       The R Utils Package
```

Pentru a vizualiza example referitoare la anumite funcţii R, de exemplu pentru funcţia `mean`, se poate proceda astfel:

```
> example(mean)
```

iar răspunsul sistemului este următorul:

```
mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50

mean> mean(USArrests, trim = 0.2)
 Murder  Assault UrbanPop     Rape
 7.42   167.60    66.20    20.16
```

Odată cu lansarea **R**-ul se deschide o nouă sesiune **R**. Ieşirea din sesiunea **R** se poate face utilizând comanda:

```
> q()
```

sau echivalent

```
> quit()
```

Pe ecran va apare o întrebare referitoare la memorarea sesiunii de lucru (instrucţiuni lansate la linia de comandă, obiecte cu care s-a operat, etc.) care se poate salva într-un fişier şi redeschide mai târziu sau se poate renunţa definitiv la aceasta.

Dacă se doreşte salvarea spaţiului de lucru (ce cuprinde toate variabilele create în R), într-un fişier anume se poate folosi comanda `save`. Să presupunem de exemplu că într-o nouă sesiune R, în spaţiul de lucru, printre altele, avem şi variabila x:

```
> x=2      #se creeaza si initializeaza varibila x
> save(x,file="C:/Lucru/date.RData")
> # se salveaza variabila x in fisierul date.Rdata
> q()            #părăsirea sesiunii R
```

Funcţia `save` este echivalentă cu **`saveimage`**.

Să mai precizăm că R este **case sensitive** şi deci *x* şi *X* sunt interpretate ca simboluri diferite şi se vor referi la variabile diferite.

De asemenea, să precizăm că orice şir de caractere precedat de simbolul '#' este interpretat ca fiind un **comentariu**.

La redeschiderea unei noi sesiuni R se poate proceda în modul următor:

```
> load("c:/lucru/date.RData")
> #se incarca obiectele din fisierul date
> x        #se afiseaza valoarea variabilei x
[1] 2      #raspunsul sistemului
```

În mod similar, exectând dublu click pe fişierul „date.Rdate", se realizează deschiderea unei noi sesiuni R şi apoi încărcarea fişierului „date.Rdate". De asemenea, salvarea spaţiului de lucru curent se poate realiza prin meniul *File → Save Workspace ...*, respectiv încărcarea unui fişier Rdate se poate realiza prin meniul *File → Load Workspace ...*

Comenzile se pot separa în R fie prin simbolul punct şi virgulă fie se pot scrie pe linii diferite. De exemplu:

```
> x1=1; x2=3.7;
> x3=5.1;
> x1+x2
[1] 4.7
> x1-x3
[1] -4.1
```

Dacă o comandă nu este completă la sfârşitul unei linii de la prompt-ul >, R va lansa un nou prompt, de obicei simbolul '+' în linia sau liniile următoare. Acest simbol dispare când comanda este completă. De exemplu:

```
> x=4
> y=6
> z=x*
+ y
> z
[1] 24
```

Istoria comenzilor introduse la prompt în Rconsole într-o sesiune de lucru R poate fi salvata, iar apoi reîncărcată folosind comenzile **`savehistory`** şi **`loadhistory`**, care se regăsesc şi în meniu: *File* → *Save History*..., respectiv *File* → *Load History*.... Fişierele în care se salvează / de unde se încarcă comenzile au extensia .Rhistory.

Pentru alte comenzi / funcţii R de bază se poate consulta help-ul Pachetul R Utils:

```
> library(help="utils")
```

Capitolul 2
Tipuri de date

Entităţile cu care operează R pentru a memora şi a lucra cu date se numesc **tipuri de date** sau **obiecte**. Acestea pot fi: scalari, vectori, matrice, liste, tablouri, şiruri de caractere, factori, dataframe-urile, etc. De asemenea, se pot defini tipuri noi de obiecte.

Funcţia **typeof**(obiect) sau echivalent mode(obiect) pot fi folosite pentru a afla tipul atributelor intriseci ale unui obiect. Câteva dintre cele mai des folosite tipuri de obiecte: "numeric" (întreg="integer" sau real="double"), "complex", "logical" (logic), "character" (caracter) sau "raw" (vector de biţi, fiecare componentă a acestuia fiind reprezentată separat ca pereche de cifre hexazecimale). De exemplu:

```
> x=3.2
> typeof(x)
[1] "numeric"

> x=1+2i
> typeof(x)
[1] "complex"
```

```
> s="unu"
> typeof(x)
[1] "character"

> e=1<3
> typeof(e)
[1] "logical"
```

```
> s="sir"
> b=charToRaw(s);b
[1] 73 69 72
> typeof(b)
[1] "raw"
```

Accesul la obiecte se face prin intermediul unor simboluri numite **variabile**. În R, variabilele sunt ele însele obiecte şi pot fi manipulate în acelaşi mod ca şi obiectele. Numele variabilele din R încep cu o literă, apoi pot conţine litere, cifre, „.” sau ”_”.

Pentru a atribui o valoare unei variabile se poate folosi **operatorul de atribuire** („=” sau „<−” sau „<<-„ sau „->” sau „->>”) sau se poate folosi funcţia assign din pachetul de programe R „base”:

```
> variabila = valoare
> variabila <- valoare
> variabila <<- valoare
> valoare -> variabila
> valoare ->> variabila
> assign("variabila”,valoare)
```

Se observă faptul că după atribuire, rezultatul nu este afişat automat.

Afişarea valorii unei variabile se face explicit, prin apelarea numelui variabilei sau prin folosirea funcţiei print. De exemplu:

```
> x=1.5
> x
[1] 1.5
> print(x)
[1] 1.5
```

La afişarea valorilor variabilelor se poate folosi funcţia options(nume=valoare) pentru a stabili diverse opţiuni. De exemplu, pentru a stabili numărul de zecimale la 2 se apelează înainte de afişare: **options(digits=2)**. Valorile valide sunt 1..22 cu valoarea implicită 7. Însă acest număr de zecimale este doar sugestie. De exemplu:

```
> options(digits=5)
> x=1.3
```

```
> x
[1] 1.3
> y=1/3; y
[1] 0.33333
```

2.1. Numere

R lucrează cu numere întregi, reale sau complexe. Numerele reale se scriu cu punct zecimal sau cu exponent: 0.001 = 1.e-3.

Pentru a calcula o expresie numerică, se poate edita în RConsole de la prompter expresia corespunzătoare, apoi se tastează <Enter>. De exemplu,

```
>> 2*6
[1]  12          #rezultat
```

Cea de-a doua linie reprezintă răspunsul sistemului R, iar [1] indică numărul de ordine al primei valori afişate pe acest rând. Alt exemplu:

```
>> sqrt(2)
[1]  1.414214
```

Pentru a genera 14 numere aleatoare distribuite uniform în intervalul (0,1) se poate folosi comanda:

```
> runif(n=14, min=0, max=1)
[1] 0.70871023 0.39347162 0.51954422 0.23446865 0.38287326 0.26328696
[7] 0.56719362 0.82779636 0.01660613 0.49667370 0.20340420 0.96901185
[13] 0.25215480 0.94807566
```

Operatorii binari cu care poate lucra R sunt:

Operaţia matematică	Simbolul R	Expresia R
Adunarea $x+y$	+	x+y
Scăderea $x–y$	–	x-y
Înmulţirea xy	*	x*y
Împărţirea x / y	/	x/y
Exponenţiala x^y	^	x^y
Câtul întreg al împărţirii lui x la y: $[x/y]$	%/%	x%/%y
Restul împărţirii lui x la $y = x - y * [x / y]$	%%	x%%y

Expresia matematică se evaluează de la stânga la dreapta, ţinând cont de şi ordinul de prioritate:

Ordinul de prioritate	
1	Paranteza
2	Exponenţiala
3	Înmulţirea, împărţirea, câtul, restul
4	Adunarea şi scăderea

Operatorii %% şi %/% pot fi folosiţi şi pentru operatori neîntregi. De exemplu:

```
> 10.2%/%1.2
[1] 8

> 10.2%%1.2
[1] 0.6
```

În R există constante speciale precum: **`Inf`** (ce ţine locul lui infinit ∞), **`NaN`** ("not-a-number", ce indică un rezultat numeric nedefinit ca 0/0, $(\pm\infty)/(\pm\infty)$, $\infty - \infty$ sau $0 \times (\pm\infty)$), **`NULL`** (indică un obiect gol) şi **`NA`** ("not available"). De exemplu:

```
> 1/0
[1] Inf

> 5^987
[1] Inf

> 0/0
[1] NaN

> Inf/Inf
[1] NaN

>u=c(1,2,NA,4);u #valoare lipsa reprezentata prin NA
[1] 1 2 NA 4

> A = matrix(data=NA,nrow=2,ncol=3); A
     [,1]  [,2]  [,3]
[1,]   NA    NA    NA
[2,]   NA    NA    NA
```

2.2. Vectori

Vectorii sunt structuri de date cu una sau mai multe valori de acelaşi tip: întreg, real, complex, logic, caracter sau raw (şiruri de biţi). Vectorii cu o singură componentă se numesc *scalari*.

În R, nu sunt permişi indici negativi, iar indicele 0 este ignorat. Aşadar, indicii acceptaţi sunt 1, 2, ...

Vectorii pot fi creaţi interactiv sau se pot importa din fişiere. Crearea interactivă a vectorilor se poate face utilizând funcţia de concatenare „*c*". De exemplu:

```
> u = c(1,2,3)
> p = c(1.25, 0.75)
> v = c(1+1i,1-2i)
```

Pentru vizualizarea unei componente, de exemplu a doua, se poate face astfel:

```
> u[2]
[1] 2
```

Funcţia de concatenare se poate folosi şi pentru a construi vectori de dimensiune mai mare prin concatenarea mai multor vectori de dimensiuni mai mici. În exemplul următor s-a creat un vector numeric, x_1, un vector x_2 cu elementele de tip şir de caractere şi s-au combinat cei doi vectori în unul singur, x_3. Vectorul x_3 trebuie să aibă componentele de acelaşi tip şi tipul este şir de caractere. *Regula de conversie* a datelor este *logice => numere => caractere*.

```
> x1=c(1,2,3)          #vector numeric
> x2=c("unu","doi")    #vector de siruri
> x3=c(x1,x2); x3      #concatenarea celor doi vectori
[1] "1" "2" "3" "unu" "doi"   #conversie la siruri
```

Limbajul R permite efectuarea interactivă a diferitor operaţii cu vectori numerici, precum adunarea (+), scăderea (-), înmulţirea(*), împărţirea (/) şi ridicarea la putere (^). De exemplu, pentru vectorii x = (2 4 6) şi y=(1 2 3) avem:

```
> x=c(2,4,6)      > x*y           > x+2
> y=c(1,2,3)      [1] 2 8 18      [1] 4  6 8
> x+y             > x/y
[1] 3 6 9         [1] 2 2 2       > options(digits=2)
> x-y             > x^y           > 1/x
[1] 1 2 3         [1] 2 16 216    [1] 0.50 0.25 0.17
```

Un operator special este '%*%' care aplicat unor vectori conduce la determinarea produsului scalar, ca în exemplul de mai jos:

```
> x=c(2,4,6)
> y=c(1,2,3)
> x%*%y    #produsul scalar a vectorilor x si y
     [,1]
[1,]   28
```

R dispune de o serie de facilităţi pentru generarea secvenţelor de numere, organizate ca numere sau matrice. De exemplu, pentru a genera secvenţa de numere consecutive de la 1 la 5 şi memorarea acesteia în vectorul *u* se poate proceda astfel:

```
> u=1:5; u
[1] 1 2 3 4 5
```

În mod similar, pentru se poate genera secvenţa de numere consecutive de la 5 la 1 şi apoi memorarea acesteia în vectorul *v* folosind instrucţiunea

```
> v=5:1; v
[1] 5 4 3 2 1
```

Generarea de numere se poate face şi cu funcţia **`seq(from,to,by).`** Primele două argumente specifică valorile de început (from) şi sfârşit (to) ale secvenţei, iar ultimul argument (by) reprezintă pasul. De exemplu:

```
> v1=seq(3,5)    #sau v1=seq(from=3,to=5)
[1] 3 4 5

> v2=seq(from=1, to=2, by=0.2); v2
[1] 1.0 1.2 1.4 1.6 1.8 2.0

> seq(4,16,length=6)
[1] 4.0  5.2  6.4  7.6  8.8 10.0
```

O funcţie înrudită cu `seq()` este `rep()`. Aceasta se utilizează pentru a replica un obiect în diferite moduri.

```
> x=c(1,2,3)
> x1=rep(x, times=2); x1    #x se repeta de 2 ori
[1] 1 2 3 1 2 3

> x2=rep(x, each=2); x2
#fiecare componenta a lui x se va repeta de 2 ori
[1] 1 1 2 2 3 3

> rep(1:3,c(2,4,6))
[1] 1 1 2 2 2 2 3 3 3 3 3 3
#prima componenta s-a multiplicat de  ori, a doua de 4 ori etc
```

```
>rep(0,10)
[1] 0 0 0 0 0 0 0 0 0 0
```

Este important să putem identifica anumite şiruri de numere. Operatorul **„%in%”** care testează existenţa unei secvenţe într-un vector de numere este deosebit de util în acest scop. De exemplu:

```
> x=rep(1:3,rep(2,3)); x
[1] 1 1 2 2 3 3
> x[x %in% c(1,3)]
#se testează existenţa valorilor 1 şi 3
[1]  1 1 3 3
```

Pentru a determina lungimea unui vector se poate folosi funcţia length, ca mai jos:

```
> x=seq(-15,15,by=5); x
[1] -15 -10  -5   0   5  10  15
> length(x)      #lungimea vectorului x
[1] 7
```

Elementele unui **vector logic** sunt succesiuni de TRUE (adevărat), FALSE (fals) şi NA („not available”). Deseori, TRUE se abreviază cu T, iar FALSE cu F, dar este posibil ca variantele abreviate să fie variabile deja definite.

Vectorii logici sunt generaţi atunci când se impun anumite condiţii. De exemplu, dacă punem condiţia ca elementele vectorului v = (2 –1 5) să fie pozitive, putem genera vectorul logic (TRUE, FALSE, TRUE) cu secvenţa de instrucţiuni:

```
> v=c(2,-1,5); v
[1]  2 -1  5
> y=v>0
> y
[1] TRUE FALSE  TRUE
```

Primul şi ultimul element al lui *v* sunt mai mari decât zero şi condiţia este îndeplinită (TRUE), iar al doilea este negativ şi condiţia nu este îndeplinită (FALSE).

Operatorii relaţionali sunt: < (’mai mic), <= (’mai mic sau egal’), > (’mai mare’), >= (’mai mare sau egal’), = = (’egalitate’) şi != (diferit).

Operatorii logici: & (şi), | (sau), ! (negarea).

Vectorii logici pot fi utilizaţi şi în operaţii aritmetice. În acest caz, ei se transformă în vectori numerici, FALSE devenind 0, iar TRUE devenind 1:

```
> p=c(TRUE,FALSE,TRUE)
> q=c(TRUE,FALSE,FALSE)
> p
[1]  TRUE FALSE TRUE
> q
[1]  TRUE FALSE FALSE
> p+q
[1] 2 0 1
```

Vectorii de tip caracter sunt secvenţe de caractere delimitate de ghilimele. Aceştia pot fi concatenaţi utilizând funcţia `c()`. De asemenea, vectorii de tip caracter pot fi concatenaţi folosind funcţia `paste()`, care acceptă un număr arbitrar de argumente pe care le concatenează unul câte unul, separându-le printr-un caracter blank. Orice număr este transformat în caracter.

```
> nume="Popescu"
> prenume="Ion"
> numeintreg=paste(nume,prenume)
> numeintreg
[1] "Popescu Ion"
> n=c(nume,prenume)
> n
[1] "Popescu" "Ion"
> n[1]
[1] "Popescu"
> n[2]
[1] "Ion"
```

Funcţia `strsplit` se poate folosi pentru extragerea unor caractere dintr-un vector utilizat ca argument. De exemplu, pentru eliminarea caracterului 'a' din şirul "Transilvania" se poate proceda astfel

```
> strsplit("Transilvania","a")
[[1]]
[1] "Tr" "nsilv" "ni"
```

Funcţia sort se poate folosi pentru a rearanja / sorta elementele unui vector caracter în ordine alfabetică. De exemplu:

```
> s=c("Iulia","Magda","Ana","Paula")
> s
[1] "Iulia" "Magda" "Ana"   "Paula"
> sort(s)
[1] "Ana"   "Iulia" "Magda" "Paula"
```

2.3. Matrice şi tablouri de numere

Matricele pot fi **create** în diverse moduri. Modalitatea cea mai simplă este de a utiliza vectori de aceeaşi lungime care se pun ca şi coloane ale matricei prin instrucţiunea `cbind()` sau ca şi linii ale matricei prin instrucţiunea `rbind()`, ca mai jos. **Dimensiunile** unei matrice se pot determina prin folosirea funcţiei `dim`.

```
> y1=c(1,2,3)
> y2=c(4,5,6)
> A=cbind(y1,y2) # concatenare pe orizontala
> B=rbind(y1,y2) # concatenare pe verticala
> A
     y1 y2
[1,]  1  4
[2,]  2  5
[3,]  3  6
> B
   [,1] [,2] [,3]
y1    1    2    3
y2    4    5    6
> dim(B)
[1] 2 3
```

Pentru ca doi vectori să se poată concatena pe orizontală aceştia trebuie să aibă acelaşi număr de linii, iar pentru se putea concatena pe verticală aceştia trebuie să aibă acelaşi număr de coloane.

Funcţiile `cbind` şi `cbind` se pot aplica şi la matrice. De exemplu:

```
> a<-c(-1,0,1)
> b<-c(-2,-1,1)
> c<-cbind(a,b); #concatenare vectori pe orizontala
> c
      a  b
[1,] -1 -2
[2,]  0 -1
[3,]  1  1
> d<-cbind(c,c);d #concatenare matrice pe orizontala
      a  b  a  b
[1,] -1 -2 -1 -2
[2,]  0 -1  0 -1
[3,]  1  1  1  1
> d=rbind(d,d); d #concatenare matrice pe verticala
      a  b  a  b
[1,] -1 -2 -1 -2
[2,]  0 -1  0 -1
[3,]  1  1  1  1
[4,] -1 -2 -1 -2
[5,]  0 -1  0 -1
[6,]  1  1  1  1
> e=cbind(5,c) #se adauga o coloana de 5 la matricea c
> e
        a  b
[1,] 5 -1 -2
[2,] 5  0 -1
[3,] 5  1  1
> f=rbind(7,c);f #se adauga o linie de 7 la matricea c
      a  b
[1,]  7  7
[2,] -1 -2
[3,]  0 -1
[4,]  1  1
```

Dacă tipurile vectorilor care intră în matrice sunt diferite atunci se aplică regula de conversie logice=>numere=>caractere. Este posibil ca lungimile vectorilor să fie diferite, dar cea mai mare lungime să fie multiplu întreg al lungimilor mai mici, caz în care vectorii de lungime mai mică se repetă.

O a doua modalitate de a crea matrice, cât şi tablouri multidimensionale (arrays) se poate face şi prin comenzile `matrix` şi `array` plecând de la un vector prin aranjarea elementelor lui într-un tablou de dimensiuni date.

```
> y=1:12
> m=matrix(y, nrow=4, ncol=3) #completare pe coloane
> m
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Se observă că din elementele vectorului y_3 se formează coloanele matricii m_3. Numărul de elemente ale primului parametru trebuie să fie multiplu sau submultiplu al lui `nrow*ncol`. În cazul în care este submultiplu, valorile vor fi ciclate până când toate valorile sunt completate. De exemplu:

```
> m=matrix(1:6, nrow=4, ncol=3); m
> #completare pe coloane
     [,1] [,2] [,3]
[1,]    1    5    3
[2,]    2    6    4
[3,]    3    1    5
[4,]    4    2    6
```

sau

```
> m=matrix(1:14, nrow=4, ncol=3)
Warning message:
In matrix(1:14, nrow = 4, ncol = 3) :
data length [14] is not a sub-multiple or multiple
of the number of rows [4]
```

Dacă dorim ca din elementele lui să formăm liniile**,** mai adăugăm indicaţia `byrow=TRUE` (valoarea implicită fiind `byrow=FALSE`)**.** Astfel se obţine

```
> m4=matrix(1:12, nrow=4, ncol=3, byrow=TRUE)
> #completare pe linii
> m4
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

O altă modalitate de a crea matrice este cea care converteşte efectiv un vector în matrice, folosind funcţia `dim` ca în exemplul de mai jos:

```
> x=seq(1:12); x      #x este vector
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> dim(x)=c(4,3); x    #x este matrice
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Accesul la elementele din vector, matrice sau array se face punând indicii de poziţie ai elementului între paranteze drepte, şi eventual separându-i prin virgulă:

```
> m=matrix(1:6, nrow=2, ncol=3)
> m    # sau m[,]
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m[1,3]
[1] 5
> m[2,] #sau m[2,1:3] (linia a doua)
[1] 2 4 6
> m[2,1:2] #linia a doua, primele doua elemente
[1] 2 4
```

```
> m[,1] #sau m[1:2,1] (coloana 1)
[1] 1 2
> m[,2:3]  #coloanele 2 si 3
     [,1] [,2]
[1,]    3    5
[2,]    4    6
```

Pentru a prezenta **operaţiile cu matrice**, să considerăm întâi matricele $A=\begin{pmatrix} 2 & 4 \\ 1 & 10 \\ 6 & 0 \end{pmatrix}$ şi $B=\begin{pmatrix} 4 & 6 \\ 8 & 5 \\ 1 & 2 \end{pmatrix}$:

```
> A=matrix(c(2,4,1,10,6,0),ncol=2,byrow=TRUE)
> A
      [,1] [,2]
[1,]    2    4
[2,]    1   10
[3,]    6    0
> B=matrix(c(4,6,8,5,1,2),ncol=2,byrow=TRUE)
> B
      [,1] [,2]
[1,]    4    6
[2,]    8    5
[3,]    1    2
```

Apoi

```
> A+B
     [,1] [,2]
[1,]    6   10
[2,]    9   15
[3,]    7    2
> A-B
     [,1] [,2]
[1,]   -2   -2
[2,]   -7    5
[3,]    5   -2
> A*B
> # (aij*bij)i,j = inmultirea element cu element
     [,1] [,2]
[1,]    8   24
[2,]    8   50
[3,]    6    0
> A/B      # (aij/bij)i,j
> # (aij/bij)i,j = impartirea element cu element
      [,1]      [,2]
[1,] 0.500 0.6666667
[2,] 0.125 2.0000000
[3,] 6.000 0.0000000
> A^B      # (aij^bij)i,j
> #(aij^bij)i,j=ridicarea la putere element cu elem.
     [,1]   [,2]
[1,]   16   4096
[2,]    1 100000
[3,]    6      0
```

De observat că înmulţirea matricelor folosind operatorul '*' nu este înmulţirea clasică ci înmulţirea element cu element. Analog, folosirea operatorului '/', respectiv '^' conduce la construirea unei matrice în care fiecare element este de tipul a_{ij} / b_{ij}, respectiv a_{ij} ^ b_{ij}.

Pentru înmulţirea clasică a două matrice se foloseşte operatorul '%*%' aplicat unor matrice de dimensiuni corespunzătoare.

```
> C=matrix(c(2,4,1,10),ncol=2,byrow=TRUE); C
     [,1] [,2]
[1,]    2    4
[2,]    1   10
> D=matrix(c(4,6,8,5),ncol=2,byrow=TRUE); D
     [,1] [,2]
[1,]    4    6
[2,]    8    5
> C%*%D    #inmultirea clasica dintre doua matrice
     [,1] [,2]
[1,]   40   32
[2,]   84   56
```

Dacă unul dintre operanzi este vector şi al doilea matrice, atunci la înmulţirea vectorului cu matricea, vectorul este organizat în vector linie sau coloană astfel încât să se poată realiza înmulţirea dintre vector şi matrice. De exemplu:

```
> x=1:2; x                  #vector
[1] 1 2
> A=matrix(1:4,c(2,2)); A   #matrice
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> x%*%A
     [,1] [,2]
[1,]    5   11
> A%*%x
     [,1]
[1,]    7
[2,]   10
```

$$(1 \quad 2)\cdot\begin{pmatrix}1 & 3\\ 2 & 4\end{pmatrix}=(5 \quad 11)$$

$$\begin{pmatrix}1 & 3\\ 2 & 4\end{pmatrix}\cdot\begin{pmatrix}1\\ 2\end{pmatrix}=\begin{pmatrix}7\\ 10\end{pmatrix}$$

Pentru determina transpusa unei matrice se poate folosi funcţia `t`:

```
> A=matrix(c(2,4,1,10,6,0),ncol=2,byrow=TRUE); A
> A
     [,1] [,2]
[1,]    2    4
[2,]    1   10
[3,]    6    0
> t(A)      # transpusa matricei A
     [,1] [,2] [,3]
[1,]    2    1    6
[2,]    4   10    0
```

Pentru a determina inversa unei matrice se poate folosi funcţia `solve`:

```
> M=matrix(c(1,2,3,4),ncol=2,byrow=TRUE); M
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

```
> solve(M)     # M^(-1)
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
> A=matrix(c(1,2,1,2),c(2,2)); A       #det(A)=0
     [,1] [,2]
[1,]    1    1
[2,]    2    2
> solve(A)
Error in solve.default(A) :
  Lapack routine dgesv: system is exactly singular
```

2.4. Tablouri numerice

Tablourile de numere (`arays`) sunt generalizări ale matricelor la dimensiuni mai mari sau egale cu doi. O matrice este un `array` de dimensiune doi.

Crearea unui tablou de numere se poate face utilizând funcţia **`array(elemente,dimensiuni)`**. De exemplu:

```
> m5=array(1:12, dim=c(2,3,2))
> m5
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

O altă modalitate de a crea tablouri multidimensionale, este de a utiliza funcţia `dim` ca în exemplul de mai jos:

```
> x=1:8
> dim(x)=c(2,2,2)
> x
, , 1
     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2
     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

Astfel, vectorul *x* a fost transformat în tablou de dimensiuni 2×2×2.

Tablourile pot fi utilizate în diferite operaţii aritmetice, multe dintre acestea fiind similare celor descrise pentru matrice. Se pot efectua de asemenea operaţii între tablouri numerice şi alte obiecte R precum ar fi vectorii şi matricele. De exemplu:

```
> x=1:4
> y=5:8
> t1=array(x,c(2,2,2))
> t2=array(y,c(2,2,2))
> t1
, , 1
     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> t2
, , 1
     [,1] [,2]
[1,]    5    7
[2,]    6    8

, , 2
     [,1] [,2]
[1,]    5    7
[2,]    6    8
> t1*t2
, , 1
     [,1] [,2]
[1,]    5   21
[2,]   12   32

, , 2
     [,1] [,2]
[1,]    5   21
[2,]   12   32
> t1%*%t2  #suma_k suma_i,j t1[i,j]*t2[i,j]
     [,1]
[1,]  140
```

Dacă A şi B sunt două tablouri numerice atunci **produsul lor exterior** (engl., *outer product*) va fi un tablou numeric ale cărui elemente se obţin prin formarea tuturor combinaţiilor posibile ale elementelor lui A şi B. Simbolul pentru acest tip de produs este '%0%':

```
> AB = A%o%B
```

sau echivalent

```
> AB = outer(A,B,"*")
```

De remarcat că produsul definit de simbolul '%*%' (înmulţirea clasică dintre matrice) este diferit de produsul reprezentat prin simbolul '%o%'.

Funcţia `aperm()` poate fi utilizată pentru a permuta dimensiunile unui tablou numeric. Al doilea argument este o permutare de numere întregi {1, 2,..., *k*}, unde *k* este numărul de indici al tabloului numeric. Rezultatul acestei funcţii este un tablou numeric de aceeaşi dimensiuni cu dimensiunile, respectiv datele permutate conform

celui de-al doilea argument. Astfel, funcţia aperm poate fi o transpusă generalizată ca în exemplul următor:

```
A=array(1:12, dim=c(2,3,2))
> A
, , 1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

> B = aperm(A,c(2,1,3))
> B
, , 1
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6

, , 2
     [,1] [,2]
[1,]    7    8
[2,]    9   10
[3,]   11   12
```

În cazul particular al unei matrice A, instrucţiunea `B=aperm(A,c(2,1))` este echivalentă cu instrucţiunea `B=t(A)` şi determină transpusa matricei A.

2.5. Factori

Conceptual, factorii în R sunt variabile care iau un număr limitat de valori diferite. Aceştia sunt reprezentanţi intern ca valori întregi 1, 2,..., *k*, unde *k* este numărul de niveluri. Variabilele de tip numeric sau cele de tip caracter pot fi convertite în factori. Nivelurile de factori sunt întotdeauna şiruri de caractere. Factorii pot fi utilizaţi pentru sortarea elementelor vectorului, cât şi pentru ordonarea după o anumită funcţie de distribuţie.

Pentru crearea unui obiect de tip factor se foloseşte funcţia `factor`. Să considerăm un exemplu în care avem 10 studenţi cazaţi la un cămin studentesc, aceştia fiind identificaţi prin identificatorul judeţului din care provin:

```
> orase=c("VL","DB","VL","AG","OT","SB","DB","VL","AG","AG")
> orase
 [1] "VL" "DB" "VL" "AG" "OT" "SB" "DB" "VL" "AG" "AG"
> orasef=factor(orase)           #generarea obiectului factor
> orasef
 [1] VL DB VL AG OT SB DB VL AG AG
Levels: AG DB OT SB VL
> levels(orasef)      #lista judetelor in ordine alfabetica
[1] "AG" "DB" "OT" "SB" "VL"
```

Ordonarea şirurilor de caractere în ordine alfabetică se poate face şi cu funcţia `sort`:

```
> sort(orasef)
 [1] AG AG AG DB DB OT SB VL VL VL
Levels: AG DB OT SB VL
```

Continuând aplicaţia anterioară, să presupunem că mai ştim mediile generale de admitere, respectiv mediile de pe anul universitar anterior:

```
> medii = c(7.9,9,8.5,7.7,8.6,9.3,7.3,9.5,8.8,9.2)
```

Să presupunem că dorim să ştim media generală medie şi erorile standard corespunzătoare fiecărui oraş în parte. Pentru ceasta putem folosi funcţia `tapply`:

```
> media_medie=tapply(medii,orasef,mean)
> media_medie
      AG       DB       OT       SB       VL
8.566667 8.150000 8.600000 9.300000 8.633333
```

Aici, `tapply` foloseşte funcţia `mean` pentru a grupa elementele primului argument, *medii*, definită de nivelurile celei de-a doua componente, *orasef*, ca şi cum ar fi structuri vectoriale separate. Rezultatul este un vector de lungime egală cu numărul nivelurilor. Pentru a calcula şi eroarea standard, determinăm variaţia cu funcţia `var` şi calculăm eroarea standard:

```
> eroare_std=function(x){sqrt(var(x))/length(x)}
> media_medie=tapply(medii,orasef,eroare_std)
> media_medie
    AG        DB     OT  SB        VL
0.2589151 0.6010408  NA  NA 0.2694301 #eroarea std
```

Valorile NA corespund cazului în care calculul erorii standard nu a fost posibil, deoarece numărul studenţilor a fost mai mic sau egal cu 1.

Factorii ordonaţi se obţin practic prin ordonarea nivelurilor. Pentru a crea un factor ordonat sau pentru a ordona un factor existent se poate utiliza funcţia `ordered`. Nivelurile unui factor ordonat specifică poziţia acestora pe o scară ordinară. Pe lângă factorii ordonaţi, mai există şi factorii de contrast.

2.6. Data frame

`Data frame` este o colecţie de date organizate într-un tablou dreptunghiular în care pe fiecare coloană sunt date de acelaşi tip. Un exemplu este tabelul următor:

Nume	Analiză	Algebră
Popescu Nicolae	8	6
Stan Ion	7	7
Vasile Anca	9	8

O variabilă `data frame` se creează prin alipirea unor vectori de aceeaşi dimensiune, ca în exemplele de mai jos:

```
> c1=c("a","b","c")
> c2=c(1,2,3)
> df1=data.frame(c1,c2)     #creare data frame
> df1
```

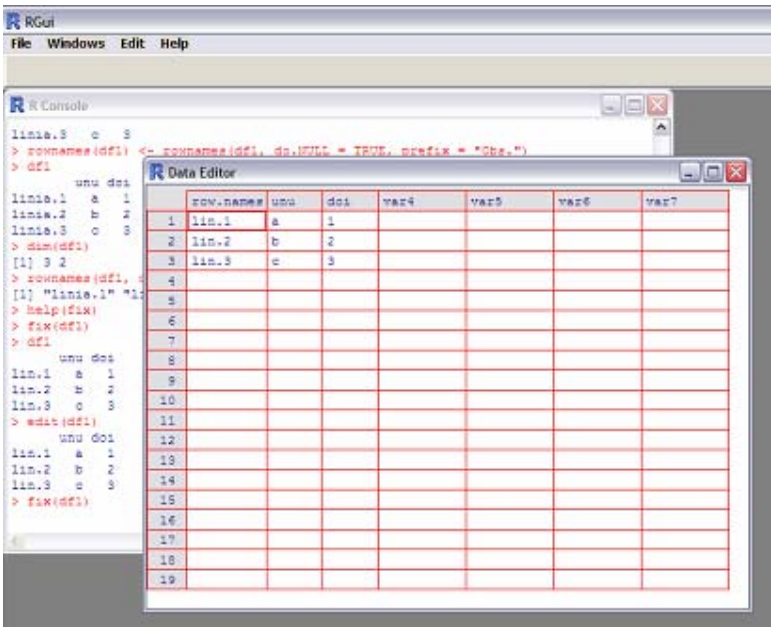
```
  c1 c2
1  a  1
2  b  2
3  c  3
```

sau

```
> clasa=c("9A","9B","9C","9D","9E","9F","9G","9H")
> media=(8.82,9.01,8.45,8/23,8.14,8.01,7.76,9.08)
> catalog=data.frame(clasa,media); catalog
  clasa      media
1    9A  8.8200000
2    9B  9.0100000
3    9C  8.4500000
4    9D  0.3478261
5    9E  8.1400000
6    9F  8.0100000
7    9G  7.7600000
8    9H  9.0800000
```

Numele coloanelor coincide cu numele vectorilor, iar liniile au numele 1, 2, 3, ... Putem schimba aceste nume ca în secvenţa de mai jos:

```
> rownames(df1)=c("linia.1","linia.2","linia.3")
> colnames(df1)=c("unu","doi")
> df1
        unu doi
linia.1   a   1
linia.2   b   2
linia.3   c   3
```



Numele liniilor, coloanelor ca şi valorile din `data.frame` (sau din alt obiect precum matricea sau vectorul) pot fi editate cu ajutorul unui editor propriu R care se apelează prin comenzile **`fix`** sau **`edit`**, ca mai jos:

```
> fix(df1)
> df1
      unu doi
lin.1   a   1
lin.2   b   2
lin.3   c   3
```

În procesul de editare am schimbat numele liniilor.

Se pot crea matrice sau data frames cu ajutorul editorului încorporat prin instrucţiunile

```
> df2=edit(data.frame())
> m5=edit(matrix())
```

Valorile pentru `df2` şi `m5` vor fi cele completate în editor.

Numele coloanelor unei date de tip `data frame` se obţine prin comanda `names(data.frame)`, iar coloana cu un nume dat se poate accesa prin `data$nume` ca în exemplele următoarea:

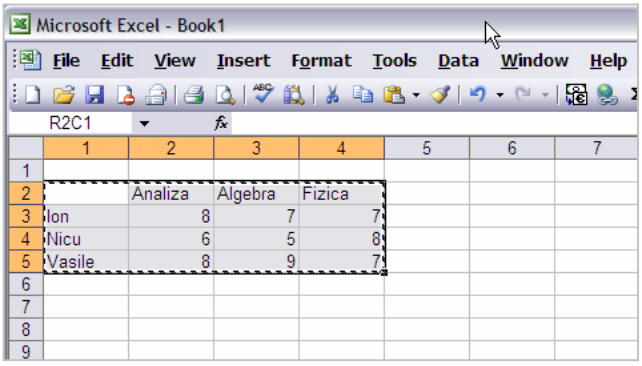
```
> df1
      unu doi
lin.1   a   1
lin.2   b   2
lin.3   c   3
> names(df1)     #numele coloanelor
[1] "unu" "doi"
> df1$unu        #coloana cu numele "unu"
[1] a b c
Levels: a b c
> df1$unu[3]     #elem. al 3-lea de pe coloana "unu"
[1] c
Levels: a b c
```

sau

```
> names(catalog)      #numele coloanelor
[1] "clasa" "media"
> catalog[1]          #prima coloana
  clasa
1    9A
2    9B
3    9C
4    9D
5    9E
6    9F
7    9G
8    9H
> catalog[1,1]  #elem.din prim coloana, prima linie
[1] 9A
Levels: 9A 9B 9C 9D 9E 9F 9G 9H
```

O variabilă de tip `data.frame` este de fapt un tabel. Poate fi gândită ca o matrice care are un cap de tabel în care şi liniile şi coloanele au denumiri. Liniile sunt considerate „nivele”, iar coloanele sunt vectori de acelaşi tip care poarta un nume.

Datele `data frame` pot fi introduse în R şi din fişiere text exterioare prin comanda **`read.table`**. Datele în fişier trebuie să umple o zonă dreptunghiulară. Fişierul text cu date poate fi copiat în clipboard, poate fi pe un suport de memorie, într-o locaţie de internet. Din mulţimea de parametri ai instrucţiunii `read.table` este important de reţinut: numele fişierului, `sep`=semnul separator între date, `header` care poate fi TRUE dacă primul rând este citit ca nume ale variabilelor, altfel FALSE, `dec` ce indică semnul separator pentru zecimale (implicit). Liniile vide ca şi cele care încep cu semnul # sunt ignorate. Datele citite sunt interpretate ca `data frame`. O comandă asemănătoare este **`read.csv`**. În acest caz fişierul trebuie să fie un fişier text cu datele separate prin virgulă.



De exemplu, să presupunem că s-au creat nişte date în Excel ca în figura alăturată, se selectează şi se copiază în clipboard. Datele se citesc în R astfel:

```
> t=read.table("clipboard")
> t
       Analiza Algebra Fizica
Ion          8       7      7
Nicu         6       5      8
Vasile       8       9      7
```

Pentru a vedea ce tip de dată este *t* apelăm funcţia `class`:

```
> class(t)
[1] "data.frame"
```

Dacă am fi selectat doar matricea de date numerice s-ar fi citit în R tot ca `data frame`. O metodă alternativă este să salvăm documentul Excel ca fişier text sau fişier „csv" şi să-l citim cu `read.table` sau `read.csv`.

```
> t2=read.table(file.choose())
> t2
       Analiza Algebra Fizica
Ion          8       7      7
Nicu         6       5      8
Vasile       8       9      7
> t3=read.csv(file.choose())
> t3
       X Analiza Algebra Fizica
1    Ion       8       7      7
2   Nicu       6       5      8
3 Vasile       8       9      7
```

Funcţia **`file.choose`** deschide o fereastră de dialog pentru alegerea fişierului. Dacă se pune numele fişierului atunci fişierul trebuie să fie în directorul curent sau să fie indicat cu cale cu tot. În rezumat avem:

```
t=read.table("clipboard")
 #pentru citirea din clipboard
t=read.table(file.choose())
 #citire dintr-un fişier text
t=read.table("numele.fişierului")
 #citire dintr-un fişier aflat în directorul curent
t=read.table("http://adresa.web/fisier.txt")
 #citire fişier de pe internet.
```

Pentru a scrie un obiect `data frame` într-un fişier se foloseşte instrucţiunea **`write.table`** sau **`write.csv`** care au cam aceeaşi parametri ca `read.table`. De exemplu prin instrucţiunea

```
> write.table(t, file="t.txt")
```

se salvează obiectul `data frame t` în fişierul "t.txt" din directorul curent de unde poate fi citit eventual altă dată cu `read.table`.

Pentru scrierea vectorilor sau matricelor în fişiere text se poate utiliza instrucţiunea `write`. Citirea matricelor din aceste fişiere se face cu instrucţiunea `scan`. Sintaxa instrucţiunii `write` este:

```
write(matricea, file = "fişierul", ncolumns
=nr.coloane, append = FALSE, sep = ",")
```

din care doar matricea (vectorul) şi fişierul sunt obligatorii. Dacă `file=""` atunci se scrie la consolă. Dacă `apend=TRUE` atunci se adaugă datele la sfârşitul fişierului altfel se şterge fişierul şi se recrează cu datele noi. Separatorul poate fi şi alt caracter, de exemplu spaţiu sau tab. Scrierea se face pe linii. Pentru a scrie coloanele la început, trebuie transpusă matricea. Citirea unei matrice dintr-un fişier text cu instrucţiunea `scan` are forma

```
scan(file = "", what = numeric), nmax = -1, n = -1,
sep = "",dec = ".",skip = 0, nlines = 0)
```

`File` este numele fişierului, `what` reprezintă tipul de date care se citesc (logical, integer, numeric, complex, character, raw, list), `nmax` reprezintă numărul maxim de date ce se citesc (dacă lipseşte sau nu e pozitiv atunci se citeşte până la sfârşitul sişierului), `n` are aceeaşi semnificaţie doar că se ignoră valorile invalide, `sep` este separatorul de date (dacă lipseşte se presupune că datele sunt separate prin spaţii), `dec` este separatorul pentru zecimale în numerele reale (implicit), `skip` reprezintă numărul de linii de la început care sunt ignorate, `nlines` este numărul maxim de linii care se citesc.

R include şi un număr de **baze de date** care se pot folosi în diverse aplicaţii. Aceste bazele de date sunt de fapt data frame-uri.

Pentru a vizualiza toate bazele de date disponibile, se poate folosi funcţia `data`:

```
> data()
```

şi se va deschide o fereastră în care sunt enumerate toate bazele de date disponibile. Pentru a afla informaţii suplimentare despre o anume bază de date, se poate apela help-ul sistemului R:

```
> ?cars
```

Se va deschide o fereastra web în care se specifică faltul că baza de date conţine 50 de observaţii luate din anul 1920, iar viteza este numerică şi măsurată în mph, iar disctanţa este de asemenea numerică, măsurată in ft.

Pentru a accesa una dintre aceste baze de date, se poate scrie data(numele bazei de date). De exemplu, pentru a accesa baza de date cars care conţine vitezele şi distanţele de oprire ale unor maşini putem proceda astfel:

```
> data(cars)
> cars[1:5,]   #primele 5 rânduri ale bazei de date
  speed dist   #coloanele bazei de date
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
```

Dacă se doreşte manipularea pe caracteristici, atunci se poate folosi vectorul **cars[,1]** ce ne dă toate vitezele măsurate sau vectorul **cars[,2]** ce ne dă toate distanţele disponibile.

O altă modalitate de a lucra cu o bază de date este de a folosi funcţia `attach`:

```
> attach(cars)
> names(cars)       #numele coloanelor bazei de date
[1] "speed" "dist"
> mean(speed)      #media datelor din prima coloanei
[1] 15.4
> mean(cars[,1])    #o varianta echivalenta
[1] 15.4
> mean(cars$speed) #a doua varianta echivalenta
[1] 15.4
```

2.7. Liste

Lista este o colecţie de obiecte R care pot fi de tipuri diferite şi de mărimi diferite. Crearea unei liste se face simplu prin funcţia `list` urmată între paranteze de numele obiectelor ce o compun. Să presupunem că există o familie cu numele tatălui „Ion”, numele mamei „Maria”, iar numele copiilor sunt: „Andrei” (4 ani), „Alexandra” (7 ani) şi „Bogdan” (9 ani).

```
> lista1=list(sot="Ion",sotie="Maria",nr.copii=3,
           varsta.copii=c(4,7,9))
> lista1   #afisarea continutului listei
$sot
[1] "Ion"
$sotie
[1] "Maria"
$nr.copii
[1] 3
$varsta.copii
[1] 4 7 9
```

iar accesul la un element din listă se face prin `numele_listei[[numarul de ordine]]` al elementului ca în comanda:

```
> lista1[[2]]
[1] "Maria"
```

Alt exemplu:

```
> clasa=c("9A","9B","9C","9D","9E","9F","9G","9H")
> media=c(8.82,9.01,8.45,8/23,8.14,8.01,7.76,9.08)
> catalog=data.frame(clasa,media)
> lista=list(clasa,media,catalog)
> lista[1]
[[1]]
[1] "9A" "9B" "9C" "9D" "9E" "9F" "9G" "9H"
> lista[2]
[[1]]
[1] 8.8200000 9.0100000 8.4500000 9.3478261
[5] 8.1400000 8.0100000 7.7600000 9.0800000
> lista[3]
[[1]]
```

```
  clasa     media
1    9A 8.8200000
2    9B 9.0100000
3    9C 8.4500000
4    9D 9.3478261
5    9E 8.1400000
6    9F 8.0100000
7    9G 7.7600000
8    9H 9.0800000
> lista[[1]][2]
[1] "9B"
```

2.8. Raw-uri

Şirurile de biţi se obţin prin declararea unei variabile ca vector de tip `raw` şi atribuirea fiecărui element din vector a unei valori întregi între 0 şi 255 convertită la tipul `raw` prin instrucţiunea `as.raw`. Dacă nu reuşeşte conversia atunci valoarea este luată implicit 0. Exemplu:

```
> x=raw(2) #creeaza un vector raw gol de dim. 2
> x[1]=as.raw(100)     #codul ASCII 100 (caract. d)
> x[2]=charToRaw("Z") #codul ASCII al caract. Z
> x #afisare a cate o pereche de cifre hexazecimale
[1] 64 5a
> rawToChar(x)   #conversie raw in sir de caractere
[1] "dZ"
> rawToBits(x) #conversie la biti
 [1] 00 00 01 00 00 01 01 00 00 01 00 01 01 00 01 00
```

Vectorii de tip raw pot fi manipulaţi folosind operatorii la nivel de biţi: ! (complement faţă de 1), & (şi), | (sau) and xor (sau exclusiv). Continuând exemplul de mai sus:

```
> y=!x           #complement data de 1
> rawToBits(y)
 [1] 01 01 00 01 01 00 00 01 01 00 01 00 00 01 00 01
> z=charToRaw("AB"); z
[1] 41 42
> rawToBits(z)
[1] 01 00 00 00 00 00 01 00 00 01 00 00 00 00 01 00
> rawToBits(xr&z)      #si la nivel de biti
[1] 00 00 00 00 00 00 01 00 00 01 00 00 00 00 01 00
> rawToBits(xr|z)      #sau la nivel de biti
[1] 01 00 01 00 00 01 01 00 00 01 00 01 01 00 01 00
```

Capitolul 3
Funcţii predefinite în R

Practic totul în R se obţine prin funcţii. Să urmărim comenzile de mai jos:

```
> "+"(2,3)
[1] 5
> 2+3
[1] 5
```

Se vede că operaţia de adunare este implementată ca o funcţie. La fel alte operaţii *x-y*, *x/y*, *x*y*, *x^y* sunt funcţii pentru care utilizăm forma operatorială de notaţie, mai comodă.

R vine cu un mare număr de funcţii predefinite pentru prelucrarea datelor, iar diversele pachete adiţionale vin cu funcţii suplimentare. În cele ce urmează descriem un număr de funcţii curent utilizate. În manualele ce însoţesc distribuţiile de R, în carţile din bibliografie sau în diverse locaţii de pe internet (de exemplu la adresa http://www.statmethods.net/ sau pur şi simplu căutând cu Google documentaţie pentru R) se pot găsi informaţii pentru multe alte funcţii disponibile. Pe lângă funcţiile predefinite putem defini funcţii noi care se pot utiliza exact ca cele predefinite.

Un lucru este important de menţionat. Argumentele funcţiilor au în general nume şi dacă se specifică numele lor atunci nu este importantă ordinea în care se pun, ca în exemplul următor în care se crează un şir de numere cu funcţia `seq`.

```
> seq(from=1, by=2, to=5)
[1] 1 3 5
> seq(by=2, to=5, from=1)
[1] 1 3 5
```

De asemenea, unele funcţii admit un număr mai mare de argumente care nu sunt compatibile decât în anumite combinaţii, ca în exemplul de mai jos :

```
> seq(by=2, to=5, from=1)
[1] 1 3 5
> seq(from=1, by=2, length=3)
[1] 1 3 5
> seq(from=1, by=2, to=5, length=3)
Error in seq.default(from = 1, by = 2, to = 5,
length = 3): too many arguments
```

Numele argumentelor formale este ales intuitiv (în engleză) şi după un pic de practică nu este dificil a le ţine minte. De la o funcţie la alta unele argumente formale au valori predefinite şi funcţiile pot fi apelate fără aceste argumente (se utilizează valorile implicite în calcul). Apar astfel situaţii când aceeaşi funcţie este apelată cu un număr diferit de argumente. O astfel de situaţie este curentă pentru funcţiile grafice de exemplu. Dacă funcţiile sunt apelate fără menţionarea numelui argumentelor atunci ordinea şi semnificaţia argumentelor este strictă, ca în programul de definiţie a funcţiei. De exemplu

```
> seq(1,5,2)
[1] 1 3 5
```

pune în evidenţă că în funcţia `seq` primul argument este `from`, al doilea este `to` şi al treilea este `by`. Pentru foarte multe funcţii din R apelul se face cu primele 1, 2, 3 argumente nedenumite, dar în ordinea şi cu semnificaţia specificată în documentaţie, pe când restul argumentelor sunt parametri opţionali din care unii se introduc prin numele lor iar ceilalţi se utilizează prin valorile lor implicite.

3.1. Funcţii numerice

Aceste funcţii aplicate unui număr dau ca rezultat un alt număr, iar aplicate unui vector sau unei matrice se aplică fiecărui element al vectorului sau matricei respective (sunt vectorizate).

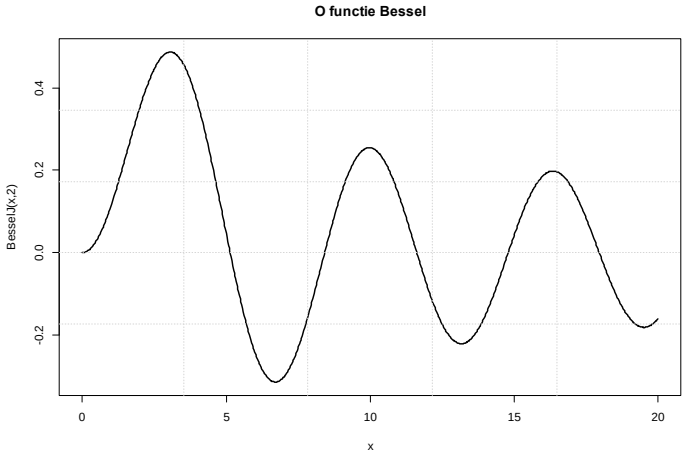
Funcţia	Explicaţii
`abs(x)`	$\|x\|$
`sqrt(x)`	$\sqrt{x}$
`ceiling(x)`	Dacă $x \in (n\text{-}1,n]$ valoarea returnată este n
`floor(x), trunc(x)`	Partea întreagă a lui x
`round(x, digits=n)`	Rotunjire la n zecimale
`signif(x, digits=n)`	Rotunjire la cele mai semnificative n cifre
`cos(x), sin(x), tan(x),` `acos(x), asin(x), tan(x),` `cosh(x), acosh(x), etc.`	Funcţii trigonometrice directe şi inverse precum şi funcţii hiperbolice directe şi inverse
`log(x)`	Logaritmul natural, ln x
`log10(x)`	Logaritmul în baza 10, lg x
`exp(x)`	e^x
`besselI(x, nu)` `besselK(x, nu)` `besselJ(x, nu)` `besselY(x, nu)`	Funcţii Bessel. Parametru nu este ordinul funcţiei. Parametru x este un vector numeric cu valori pozitive.
`gamma(x), lgamma(x)`	Funcţia gamma şi logaritm natural din gamma
`beta(a,b), lbeta(a,b)`	Funcţia beta şi logaritm natural din beta
`digamma(x)`	Calculează derivata lui `log(gamma(x))`
`psigamma(x, deriv)`	Calculează derivata de ordin `deriv` a lui `digamma(x)`
`choose(n,k), lchoose(n,k)`	Combinări de n câte k (n poate fi real); `lchoose` calculează logaritm natural din choose.
`factorial(x)`	Calculează `gamma(x+1)`

`F<-splinefun(x, y, method)`	Valoarea întoarsă este o funcţie, *F*, care poate fi ulterior apelată normal, *F*(*a*) pentru valoarea funcţiei sau *F*(2, deriv=1) ceea ce duce la calculul derivatei funcţiei spline. `deriv` poate lua valorile 0, 1, 2, 3. *x*, *y* sunt vectorii cu coordonatele punctelor de interpolat method = "fmm", "periodic", "natural", "monoH.FC", şi indică tipul de funcţie spline
`spline(x, y , n, method, xmin, xmax, xout)`	Forma aceasta calculează valoarea funcţiei spline determinată de vectorii *x* şi *y* în *n* puncte echidistante între *xmin* şi *xmax* sau în vectorul *xout*.

Exemplul 3.1 Să se reprezinte grafic funcţia Bessel J_2 pentru $x \in [0,20]$. O posibilă rezolvare este:

```
x=seq(0,20,length=1000)
y=besselJ(x,2)
plot(x,y,type="lines",lwd=2,ylab="BesselJ(x,2)",
xlab="x", main="O functie Bessel")
grid(5,5)
```

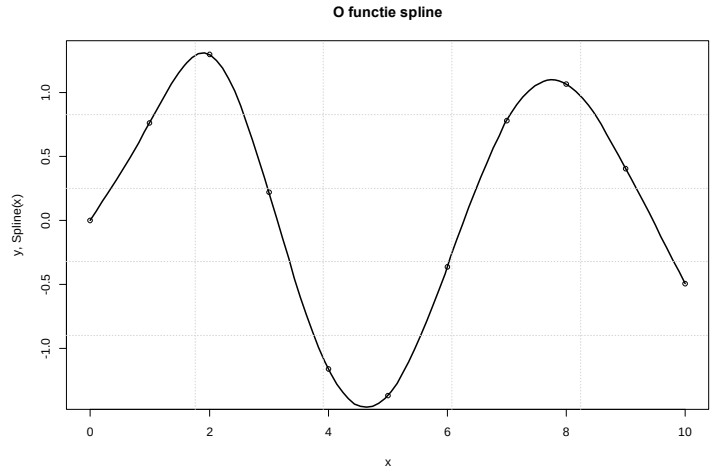
Graficul este :



Exemplul 3.2 Se dau doi vectori şi să se reprezinte grafic punctele care au coordonatele date de cei doi vectori precum şi funcţia spline naturală ce trece prin puncte. O posibilă rezolvare este:

```
x=0:10
y=x*sin(x)/(1+x^2/10)
plot(x,y,main="O functie spline", xlab="x", ylab="y,
Spline(x)")
F<-splinefun(x,y,method="natural")
x1=seq(0,10,length=100)
y1=F(x1)
lines(x1,y1,lwd=2)
grid(5,5)
```

Graficul este:



3.2. Funcţii pentru matrice

Matricele reprezintă un obiect matematic important în statistică şi de aceea R are implementate multe funcţii referitoare la ele. Mai jos indicăm unele din ele.

Operatorul sau funcţia	Explicaţii
`A * B`	Înmulţirea element cu element
`A %*% B`	Înmulţirea matriceală
`outer(a,b,f)` `a %o% b`	Dacă *a* şi *b* sunt vectori generează o matrice *x* cu $x_{i,j} = f(a_i, b_j)$. Dacă *a* şi *b* sunt matrice generează un `array` `x` cu $x_{i,j,k,l} = f(a_{i,j}, b_{k,l})$. Forma `a %o% b` este asemănătoare faţă de funcţia produs.
`crossprod(A,B)` `crossprod(A)`	A^tB, respectiv A^tA
`cbind(A,B,...)`	Combină orizontal (rezultatul va avea liniile din *A*, *B*,...)
`rbind(A,B,...)`	Combină vertical (rezultatul este o matrice cu coloanele din *A*, *B*,...)
`rowMeans(A)`	Se obţine un vector cu mediile liniilor
`rowSums(A)`	Se obţine un vector cu sumele elementelor liniilor
`colMeans(A)`	Se obţine un vector cu mediile coloanelor
`colSums(A)`	Se obţine un vector cu sumele elementelor coloanelor
`min(A), max(A),` `range(A)`	Minimul, maximu, respectiv un vector cu minimul si maximul
`solve(A)`	Inversa matricei *A*
`ginv(A)`	Inversa generalizată Moore-Penrouse (se găseşte în pachetul MASS)
`x<-eigen(A)`	x$values conţine valorile proprii ale lui *A* x$vectors conţine vectorii proprii ai lui *A*

`t(A), det(A)`	Transpusa, respectiv determinantul unei matrice
`x<-svd(A)`	Descompunerea singulară a matricei A, $A=UDV^t$, cu `D=diag(d)` x$d = vector ce conţine valorile singulare ale matricei A x$u = matrice ortogonală cu vectorii singulari la stânga ai matricei A x$v = matrice ortogonală cu vectorii singulari la dreapta ai matricei A
`R <- chol(A)`	Factorizarea Choleski, $A=R^tR$, cu R superior triunghiulară, iar A pozitiv definită.
`x<- qr(A)`	Descompunerea QR a matricei A x$qr are deasupra diagonalei principale matricea R (superior triunghiulară), iar sub diagonala principală informaţii pentru construcţia matricei Q. x$rank este rangul lui A x$qraux este un vector cu informaţii suplimentare despre Q x$pivot conţine informaţii despre strategia utilizată
`qr.Q(x), qr.R(x), qr.X(x)`	Crează dintr-un obiect x de tip qr generat de funcţia `qr` matricile Q, R sau matricea iniţială x.
`v<-sort(A,....)`	v$x conţine elementele vectorului sau matricei A (matricea este convertită în vector punând coloanele una după alta) sortate. v$ix conţine dacă parametrul suplimentar `index.return=FALSE` un vector cu poziţiile originale ale elementelor ordonate. Dacă `decreasing=TRUE` sortarea se face descrescător Dacă `partial=vector de întregi` sortarea se face numai pentru componentele indicate de partial
`rev(x)`	Inversează ordinea elementelor vectorului x

Exemplul 3.3 Se generează aleator un sistem liniar şi se determină soluţia lui.

```
a = matrix(runif(25),ncol=5)
b = runif(5)
x = solve(a,b)
a
b
x
```

Se obţine răspunsul:

```
> a
          [,1]       [,2]      [,3]       [,4]      [,5]
[1,] 0.2368829 0.79721628 0.1265954 0.04369927 0.2061009
[2,] 0.1015411 0.70852984 0.7379991 0.03374915 0.4783938
[3,] 0.4941975 0.28829965 0.2681115 0.92383512 0.3105410
[4,] 0.8037851 0.04331768 0.5082257 0.62702404 0.3799365
[5,] 0.2050783 0.22855288 0.3172556 0.75981321 0.1595518
```

```
> b
[1] 0.4920966 0.5553061 0.3207777 0.2010808 0.5628473
> x
[1]  0.1839354  0.9262001  1.2986354  0.3486333 -2.2779749
```

Exemplul 3.4 Să se determine în cât timp se rezolvă un sistem liniar 1000×1000 generat aleator. Rezolvarea este în secvenţa de cod următoare:

```
ptm <- proc.time()
a = matrix(runif(1000000),ncol=1000)
b = runif(1000)
x = solve(a,b)
proc.time() - ptm
```

Răspunsul pe un calculator cu procesor Q9300 quad la 2.5 Ghz este:

```
user  system elapsed
0.69    0.08    1.36
```

Exemplul 3.5 Să se determine valorile şi vectorii proprii ai unei matrice precum şi descompunerea ei singulară. Se va alege o matrice singulară.

```
x=runif(5); y=runif(5)
a=outer(x,y,"+")
v<-eigen(a)
ds<-svd(a)
```

Ca produs `outer`, matricea *a* este singulară. Valorile şi vectorii proprii sunt:

```
> v
$values
[1] 6.819055e+00 -7.174162e-02 -2.935982e-16 -5.941790e-17
-1.790505e-17

$vectors
          [,1]        [,2]        [,3]       [,4]       [,5]
[1,] 0.3264044  0.73432693 -0.67659694  0.3795791  0.3029054
[2,] 0.4377868  0.04995971  0.68624453 -0.3881873 -0.1576330
[3,] 0.5506698 -0.64362770  0.21134194  0.6398041  0.3044435
[4,] 0.4216151  0.14932348 -0.14367175 -0.5354580 -0.8120513
[5,] 0.4699004 -0.14735628 -0.07731778 -0.0957380  0.3623354
```

Se vede că trei valori proprii sunt aproape zero. În fapt ele sunt zero, matricea având rangul 1 şi doar erorile de rotunjire fac rezultatele de ordinul 10^{-16}. Descompunerea singulară este:

```
> ds

$d
[1] 6.870905e+00 1.077193e-01 1.869400e-16 2.686574e-17 3.540473e-18

$u
           [,1]        [,2]       [,3]       [,4]         [,5]
[1,] -0.3262407  0.77055695 -0.5096937 -0.1995559  0.014098365
[2,] -0.4377668  0.09392287  0.3454248  0.1386439 -0.813017974
[3,] -0.5507955 -0.59182723 -0.3600220 -0.4655446 -0.004147016
[4,] -0.4215743  0.19216386  0.6827627 -0.2779026  0.491887645
[5,] -0.4699219 -0.10116352 -0.1584718  0.8043592  0.311178980
```

```
$v
           [,1]       [,2]          [,3]        [,4]        [,5]
[1,] -0.3932570 -0.5884878  0.674404332 -0.20995153 -0.01141111
[2,] -0.3959388 -0.5602933 -0.706356531  0.03387009  0.17093016
[3,] -0.4729784  0.2496549 -0.138133931 -0.21376690 -0.80572118
[4,] -0.4682765  0.2002217  0.164709931  0.84032560  0.08574326
[5,] -0.4955703  0.4871727  0.005376199 -0.45047726  0.56045887
```

Putem verifica faptul că $A=UDV^t$, unde U şi V sunt matrici ortogonale (date de dsu, dsv), iar D este matricea diagonală cu ds$d pe diagonală. Verificarea este făcută prin comanda următoare:

```
> a-ds$u%*%diag(ds$d)%*%t(ds$v)
              [,1]          [,2]          [,3]          [,4]          [,5]
[1,] -1.110223e-16 -8.881784e-16 -8.881784e-16 -6.661338e-16 -8.881784e-16
[2,]  0.000000e+00 -4.440892e-16 -2.220446e-16 -2.220446e-16 -2.220446e-16
[3,] -2.220446e-16 -4.440892e-16 -2.220446e-16 -2.220446e-16 -4.440892e-16
[4,] -2.220446e-16 -4.440892e-16 -2.220446e-16 -2.220446e-16 -2.220446e-16
[5,] -2.220446e-16 -6.661338e-16 -4.440892e-16 -4.440892e-16 -4.440892e-16
```

Din nou erorile de rotunjire duc la un rezultat doar foarte apropiat de zero, nu exact zero.

3.3. Funcţii pentru şiruri de caractere

Pentru prelucrarea şirurilor şi vectorilor cu elemente şiruri de caractere se pun la dispoziţie mai multe funcţii. Unele sunt în tabelul de mai jos.

Funcţia	Explicaţii
`character(length)`	Crează un şir de blank-uri de lungime `length`
`nchar(x,...)`	Numărul de caractere din şir
`substr(x,start=n1,stop=n2)`	Extrage un subşir dintr-un şir dat sau înlocuieşte o parte dintr-un subşir cu un şir dat
`grep(model,x,fixed=TRUE)`	Caută şirul `model` in *x* (*x* este convertit la un vector de şiruri). Rezultatul este o mulţime de indici care specifică indicii şirurilor unde s-a găsit `model`. Printre parametrii opţionali se numără: `fixed=TRUE,FALSE` care dacă este `FALSE` căutarea se face după `model`=expresie regulară, `ignore.case=TRUE,FALSE`, care specifică dacă se ignoră sau nu faptul că sunt litere mari sau mici.
`sub(sir1,sir2,x,fixed=TRUE)` `gsub(sir1,sir2,x,fixed=TRUE)`	Înlocuieşte `şir1` cu `şir2` în `x`. Dacă `fixed=FALSE` atunci `şir1` este o expresie regulată. Parametrul `ignore.case=TRUE, FALSE` controlează dacă se ţine seama de caractere majuscule şi minuscule. Funcţia `sub` înlocuieşte doar prima apariţie a lui `şir1` iar funcţia `gsub` înlocuieşte toate apariţiile şirului `şir1`.

`strsplit(x, split)`	Scindează şirul *x* în subşiruri. Scindarea se face acolo unde este întâlnit şirul `split`. De exemplu şirul "as de frt yt uutyh" este scindat prin comanda `sir1=strsplit(sir,"t")` în şirurile: "as de fr", " y", " uu", "yh". `sir1` este o listă ce conţine un singur vector `sir1[[1]]` ce are ca elemente subşirurile rezultat.
`paste(..., sep=" ", colapse=NULL)`	Converteşte argumentele în şiruri de caractere pe care le concatenează utilizând separatorul sep. Dacă unele argumente sunt vectori se face concatenarea termen cu termen repetând ciclic elementele vectorilor mai scurţi. Dacă De exemplu, `paste("a",1:3,1:4, sep="")` are ca rezultat "a11" "a22" "a33" "a14". Dacă `collapse=""` atunci elementele apar ca un singur şir de caractere. Comanda `paste("a",1:3,1:4,sep="",collapse="")` dă ca rezultat: "a11a22a33a14".
`toupper(x)`	Converteşte la majuscule
`tolower(x)`	Converteşte la minuscule
`LETTERS, letters`	Sunt vectori cu caracterele majuscule (minuscule) ale alfabetului latin.

Exemplul 3.6 Urmărind comenzile de mai jos se pot vedea unele variante de aplicare a operaţiilor cu şiruri.

```
> x="avxfdrwyudbcbdgfjavsjhdhgderwrwy"
> nchar(x)
[1] 32

> x1=strsplit(x,split=NULL)
> x1
[[1]]
[1]  "a" "v" "x" "f" "d" "r" "w" "y" "u" "d" "b" "c" "b"
[14] "d" "g" "f" "j" "a" "v" "s" "j" "h" "d" "h" "g" "d"
[27] "e" "r" "w" "r" "w" "y"

> x2=unlist(x1)
> x2
[1]  "a" "v" "x" "f" "d" "r" "w" "y" "u" "d" "b" "c" "b"
[14] "d" "g" "f" "j" "a" "v" "s" "j" "h" "d" "h" "g" "d"
[27] "e" "r" "w" "r" "w" "y"

> x3=rev(x2)
> x3
[1]  "y" "w" "r" "w" "r" "e" "d" "g" "h" "d" "h" "j" "s"
[14] "v" "a" "j" "f" "g" "d" "b" "c" "b" "d" "u" "y" "w"
[27] "r" "d" "f" "x" "v" "a"

> x4=paste(x3,collapse="")
```

```
> x4
[1] "ywrwredghdhjsvajfgdbcbduywrdfxva"
> x5=substr(x4,3,7)
> x5
[1] "rwred"
> x6=sub("d","D",x)
> x7=gsub("d","D",x)
> x
[1] "avxfdrwyudbcbdgfjavsjhdhgderwrwy"
> x6
[1] "avxfDrwyudbcbdgfjavsjhdhgderwrwy"
> x7
[1] "avxfDrwyuDbcbDgfjavsjhDhgDerwrwy"
> x8=unlist(strsplit(x,split="w"))
> x8
[1] "avxfdr"   "yudbcbdgfjavsjhdhgder"    "r"
[4] "y"
> i=grep("r",x8)
> i
[1] 1 2 3
> x8[i]
[1] "avxfdr"   "yudbcbdgfjavsjhdhgder"     "r"
```

Constatăm că şirul de comenzi care duc de la *x* la *x*4 inversează caracterele într-un şir.

3.4. Repartiţii de probabilitate

R pune la dispoziţia utilizatorilor un mare număr de distribuţii de probabilitate. Numele pentru densitatea de probabilitate începe cu *d* (de exemplu, `dnorm`), pentru funcţia de repartiţie începe cu *p* (de exemplu, `pnorm`), pentru cuantile începe cu *q* (de exemplu, `qnorm`), pentru vectori aleatori începe cu *r* (de exemplu, `rnorm`).

Densitatea de probabilitate are argumentul opţional `log=FALSE`. Dacă `log=TRUE` atunci se calculează logaritm din densitate.

Funcţia de repartiţie are parametrii opţionali `lower.tail = TRUE`, `log.p = FALSE`. Dacă `lower.tail=TRUE` atunci funcţia de repartiţie returnează valoarea $P(X \le x)$, altfel $P(X > x)$. Dacă `log.p=TRUE` se interpretează că probabilităţile sunt date prin logaritmii lor.

Funcţia care calculează cuantilele are parametrii opţionali `lower.tail = TRUE`, `log.p = FALSE`. Dacă `lower.tail = TRUE`, atunci funcţia aplicată lui *q* întoarce valoarea minimă *x* astfel ca $P(X \le x) \ge q$.

În tabelul de mai jos sunt prezentate câteva distribuţii disponibile în R.

Distribuţia	Modul de apelare

Normală
$\rho(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-m)^2}{2\sigma^2}}$
	`dnorm(x,mean=0,sd=1,log=FALSE)`
	`pnorm(x,mean=0,sd=1,lower.tail=TRUE, log.p=FALSE)`
	`qnorm(q,mean=0,sd=1,lower.tail=TRUE, log.p=FALSE)`
	`rnorm(n,mean,sd)`, *n* este numărul de valori aleatoare

	`mean = m, sd=σ`

Log-normal
$\rho(x)=\frac{1}{\sqrt{2\pi}\sigma x}e^{-\frac{(\log x-\mu)^2}{2\sigma^2}}$
	`dlnorm(x,meanlog=0,sdlog=1,log=FALSE)`
	`plnorm(q, meanlog=0, sdlog=1, lower.tail=TRUE, log.p=FALSE)`
	`qlnorm(p, meanlog=0, sdlog=1, lower.tail=TRUE, log.p=FALSE)`
	`rlnorm(n, meanlog = 0, sdlog = 1)`
	`meanlog = μ, sdlog = σ`

Binomială
$\mathbf{P}(X=x) = C_n^x p^x (1-p)^{n-x}$
$x = 0, 1, 2, ..., n$
	`dbinom(x, size, prob, log=FALSE)`
	`pbinom(q,size,prob, lower.tail=TRUE, log.p = FALSE)`
	`qbinom(p,size,prob,lower.tail=TRUE, log.p = FALSE)`
	`rbinom(nr, size, prob)`, *nr* este numărul de valori aleatoare
	`size=n, prob=p`

Binomială negativă
$\mathbf{P}(X=x) = \frac{\Gamma(x+n)}{\Gamma(n)x!}p^n(1-p)^x$
$x\in N$, $n>0$
	`dnbinom(x, size, prob, log = FALSE)`
	`pnbinom(q,size,prob,lower.tail=TRUE, log.p = FALSE)`
	`qnbinom(p,size,prob,lower.tail=TRUE, log.p = FALSE)`
	`rnbinom(n, size, prob)`
	`size=n, prob=p`

Student
$\rho(x)=\frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$
	`dt(x, df,ncp=0, log=FALSE)`
	`pt(q, df,ncp=0, lower.tail = TRUE, log.p = FALSE)`
	`qt(p, df, ncp=0, lower.tail = TRUE, log.p = FALSE)`
	`rt(n, df, ncp=0)`, *n* este numărul de valori aleatoare
	`df =` ν, `ncp` nu este de obicei utilizat

Hi pătrat
$\rho(x)=\frac{1}{2^{\nu/2}\Gamma(\nu/2)}x^{\nu/2-1}e^{-x/2}$,
$x>0$
	`dchisq(x, df, ncp=0, log=FALSE)`
	`pchisq(q,df,ncp=0,lower.tail=TRUE, log.p = FALSE)`
	`qchisq(p,df,ncp=0,lower.tail=TRUE, log.p = FALSE)`
	`rchisq(n, df, ncp=0)`
	`df =` ν
	ncp nu se utilizează de obicei, implicit e zero.

Cauchy
$\rho(x)=\frac{1}{\pi\gamma\left(1+\left(\frac{x=x_0}{\gamma}\right)^2\right)}$
	`dcauchy(x, location=0, scale=1, log=FALSE)`
	`pcauchy(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)`
	`qcauchy(p, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)`
	`rcauchy(n, location = 0, scale = 1)`
	`location =` x_0, `scale =` γ

Exponenţială
$\rho(x)=\lambda e^{-\lambda x}$, $x>0$, $\lambda>0$
	`dexp(x, rate = 1, log=FALSE)`
	`pexp(q, rate = 1, lower.tail = TRUE, log.p = FALSE)`
	`qexp(p, rate = 1, lower.tail = TRUE, log.p = FALSE)`
	`rexp(n, rate = 1)`
	`rate=`λ

Gamma
$\rho(x)=\frac{x^{k-1}}{\Gamma(k)a^k}x^{-\frac{x}{a}}$, $x\geq 0$, $k>0$, $a>0$
	`dgamma(x, shape, scale=1, log=FALSE)`
	`pgamma(q, shape, scale = 1, lower.tail = TRUE, log.p = FALSE)`
	`qgamma(p, shape, scale = 1, lower.tail = TRUE, log.p = FALSE)`
	`rgamma(n, shape, scale = 1)`
	`shape=k, scale=a`

Beta
$\rho(x)=\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}x^{a-1}(1-x)^{b-1}$
$\alpha>0$, $\beta>0$, $x\in(0, 1)$
	`dbeta(x, shape1, shape2, ncp = 0, log=FALSE)`
	`pbeta(q, shape1, shape2, ncp = 0, lower.tail = TRUE, log.p = FALSE)`
	`qbeta(p, shape1, shape2, ncp = 0, lower.tail = TRUE, log.p = FALSE)`
	`rbeta(n, shape1, shape2, ncp = 0)`
	`shape1=`α , `shape2=`β

F
$\rho(x)=\frac{1}{\beta\left(\frac{a}{2},\frac{b}{2}\right)}\left(\frac{a}{b}\right)^{a/2}x^{\frac{a}{2}-1}\left(1+\frac{a}{b}x\right)^{-\frac{a+b}{2}}$
$a>0$, $b>0$, $x\geq 0$, β este funcţia beta
	`df(x, df1, df2, ncp=0, log = FALSE)`
	`pf(q,df1,df2,ncp=0,lower.tail=TRUE, log.p = FALSE)`
	`qf(p,df1,df2,ncp=0,lower.tail=TRUE, log.p=FALSE)`
	`rf(n, df1, df2, ncp=0)`
	`df1=a, df2=b`

Geometrică
$\mathbf{P}(X=x)=p(1-x)^{x-1}$, $x\in N$, $p\in[0,1]$
	`dgeom(x, prob, log = FALSE)`
	`pgeom(q, prob, lower.tail = TRUE, log.p = FALSE)`
	`qgeom(p, prob, lower.tail = TRUE, log.p = FALSE)`
	`rgeom(n, prob)`
	`prob=p`

Hipergeometrică
$\mathbf{P}(X=x)=\frac{C_m^x C_n^{k-x}}{C_{m+n}^k}$
	`dhyper(x, m, n, k, log = FALSE)`
	`phyper(q, m, n, k, lower.tail = TRUE, log.p = FALSE)`
	`qhyper(p, m, n, k, lower.tail = TRUE, log.p = FALSE)`
	`rhyper(nn, m, n, k)`

Poisson
$\mathbf{P}(X=x)=\frac{\lambda^x e^{-\lambda}}{k!}$, $\lambda>0$, $x\in N$
	`dpois(x, lambda, log = FALSE)`
	`ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)`
	`qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)`
	`rpois(n, lambda)`
	`lambda =` λ

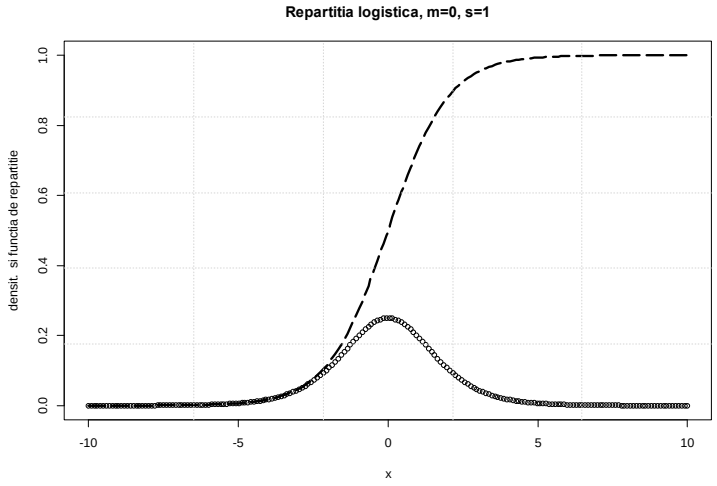
Uniformă
$\rho(x)=\frac{1}{b-a}$, $x\in[a, b]$
	`dunif(x, min=0, max=1, log = FALSE)`
	`punif(q,min=0,max=1,lower.tail=TRUE, log.p = FALSE)`
	`qunif(p,min=0,max=1,lower.tail=TRUE, log.p = FALSE)`
	`runif(n, min=0, max=1)`
	`min=a, max=b`

Weibull
$\rho(x)=\frac{a}{b}\left(\frac{x}{b}\right)^{a-1} e^{-\left(\frac{x}{b}\right)^{a}}$, $x>0$, $a>0$, $b>0$
	`dweibull(x,shape,scale=1,log=FALSE)`
	`pweibull(q, shape, scale = 1, lower.tail = TRUE, log.p = FALSE)`
	`qweibull(p, shape, scale = 1, lower.tail = TRUE, log.p = FALSE)`
	`rweibull(n, shape, scale = 1)`
	`shape=a, scale=b`

Logistică
$\rho(x)=\frac{1}{s}\frac{e^{\frac{x-m}{s}}}{\left(1+e^{\frac{x-m}{s}}\right)^{2}}$
	`dlogis(x, location = 0, scale = 1, log = FALSE)`
	`plogis(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)`
	`qlogis(p, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)`
	`rlogis(n, location = 0, scale = 1)`
	`location=m, scale=s`

Exemplul 3.7. Să realizăm graficul densităţii şi funcţia de repartiţie pentru distribuţia logistică, $m = 0$, $s = 1$.

```
x=seq(-10,10,length=200)
y=dlogis(x, location = 0, scale = 1, log = FALSE)
y1=plogis(x, location = 0, scale = 1, log = FALSE)
plot(x, y, ylim=range(y,y1), ylab="densitatea si functia
de repartitie", main="Repartitia logistica, m=0,s=1")
lines(x,y1, lwd=3, lty=5), grid(5,5)
```



3.5. Funcţii statistice elementare

Funcţia	Explicaţii
`mean(x, trim=0, na.rm=FALSE)`	Media obiectului *x*. `trim` indică ce fracţie din valorile dinspre capete se neglijează. Parametrul `na.rm` indică dacă se elimină `NA` şi `NaN` din calcul.
`sd(x,na.rm=FALSE)`	Deviaţia standard

`var(x, y = NULL, na.rm = FALSE, use)`	Varianţa (dispersia) vectorului, matricei sau `data.frame x`. Parametrul `use` este unul din şirurile: "everything", "all.obs", "complete.obs", "na.or.complete" sau "pairwise.complete.obs" şi specifică felul în care se tratează valorile lipsă. Vectorul *y* trebuie să aibă aceeaşi dimensiune ca *x*. Implicit `y=NULL` înseamnă *y*=*x*.
`median(x,na.rm=FALSE)`	Valoarea mediană
`cov(x, y = NULL, use = "everything", method = "kendall")`	Covarianţa vectorilor *x* şi *y* sau a matricei ori `data.frame x` (în acest caz se face covarianţă între coloane). Parametrul method poate fi `"pearson"`, `"kendall"`, `"spearman"`.
`cor(x, y = NULL, use = "everything",method = "kendall")`	Coeficientul de corelaţie al vectorilor *x* şi *y* sau a matricei ori `data.frame x` (în acest caz se face corelaţia între coloane). Parametrul method poate fi `"pearson", "kendall", "spearman"`.
`quantile(x,prob, na.rm=FALSE,type=7)`	Se determină cuantilele din vectorul de probabilitaţi prob pentru datele numerice din *x*. Parametrul `na.rm` controlează dacă se exclud valorile `NA` şi `NaN`. Parametrul `type` ia valori întregi între 1 şi 9 în funcţie de metoda dorită de calcul a cuantilelor.
`range(x)`	Intervalul [minim, maxim]
`sum(x,y), prod(x,y,...)`	Suma elementelor vectorilor *x*, *y*,.. respectiv produsul.
`diff(x, lag=1)`	Diferenţe finite $x_{i+lag} - x$, cu pasul `lag`. Implicit este lag=1, iar rezultatul are lungimea cu 1 mai mică. Dacă *x* este matrice se fac diferenţele pe componente.
`min(x)`	Minimul
`max(x)`	Maximul
`summary(obiect,...)`	Este o funcţie generică al cărei rezultat depinde de obiectul căreia i se aplică. Pentru un vector numeric se dau minimul, maximul, media, mediana, prima şi a treia cvartilă. Pentru o matrice sau data frame se face acelaşi lucru pentru coloane.
`scale(x, center=TRUE, scale=TRUE)`	Se înlocuieşte x_i cu $\frac{x_i - center}{scale}$. Dacă `center=TRUE` atunci se ia center egal cu `mean(x)`, iar dacă `scale=TRUE` se ia `scale=sd(x)`. Dacă *x* este o matrice se aplică procedura pentru fiecare coloană. `Center` şi `scale` trebuie să fie în acest caz un vector de centre, respectiv un vector de parametri de scalare.

Exemplul 3.8. Se dă un vector numeric *x* şi se cere un sumar al caracteristicilor statistice elementare.

```
> x<-c(1,2,-1,3,4,10,6,4,-2,6,8,3,5)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -2.000   2.000   4.000   3.769   6.000  10.000
```

3.6. Funcţii de mapare

Prin aceste funcţii se evită folosirea unor instrucţiuni de ciclare. O anumită funcţie este aplicată repetat unor date extrase din argumentele instrucţiunii.

Funcţia	Explicaţii
`apply(x, MARGIN, FUN, ...)`	*x* este un array, iar MARGIN un vector cu poziţiile pe care sunt indicii care se ciclează, FUN este funcţia care se aplică acelui array cu indicii din MARGIN fixaţi, ... sunt parametrii adiţionali pentru FUN. De exemplu, dacă *x* este un array cu mai mult de două dimensiuni, să zicem trei, comanda y=apply(x, c(1,2), sum) ne dă o matrice unde y[i,j]=sum(x[i.j,]), pentru orice i,j între limitele permise, iar comanda y=apply(x,c(1,3),sum) ne dă y[i,j]=sum(x[i,j]).
`lapply (x, FUN, ...)`	Aplică funcţia FUN fiecărui element al unui vector sau listă.
`sapply (x, FUN, ..., simplify = TRUE,...)`	La fel ca `lapply` doar că rezultatul se converteşte (simplifică) la un array.
`replicate(n, expr, simplify = "array")`	Evaluarea de *n* ori a expresiei `expr`. Dacă *x* este un vector atunci y=replicate(3,mean(x)) ne dă un vector cu trei componente egale cu media lui *x*.
`tapply(x, INDEX, FUN = NULL, ..., simplify = TRUE)`	Se aplică funcţia FUN elementelor din vectorul *x* grupate după lista de factori INDEX. Dacă `symplify=TRUE` se încearcă convertirea la array a rezultatului.
`mapply(FUN, ..., MoreArgs = NULL, simplify = TRUE, use.names = TRUE)`	Se aplică funcţia FUN, funcţie de mai multe argumente, primelor elemente ale argumentelor...., pe urmă elementelor de pe locurile doi ale argumentelor, etc. ... poate fi o înşiruire de vectori da aceeaşi lungime, data frame cu acelaşi număr de coloane sau liste cu acelaşi număr de componente. De exemplu dacă z1=c(1,2,3,4), z2=c(3,4,5,6) atunci mapply(sum,z1,z2) ne dă 4 6 8 10.

Exemplul 3.9. Să aplicăm funcţiile `apply`, `lapply`, `sapply` şi `mapply` unor matrice sau vectori. Rezultatele se pot vedea după fiecare comandă.

```
> x=matrix(1:12,ncol=4)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
> x1=apply(x,MARGIN=1,FUN=sum)
> x1
[1] 22 26 30

> x2=apply(x,MARGIN=2,FUN=sum)
> x2
[1]  6 15 24 33

> x3=lapply(x,FUN=sqrt)
> x3
[[1]]
[1] 1

[[2]]
[1] 1.414214

[[3]]
[1] 1.732051

[[4]]
[1] 2

[[5]]
[1] 2.236068

[[6]]
[1] 2.44949

[[7]]
[1] 2.645751

[[8]]
[1] 2.828427

[[9]]
[1] 3

[[10]]
[1] 3.162278

[[11]]
[1] 3.316625

[[12]]
[1] 3.464102

> x4=sapply(x,FUN=sqrt)
> x4
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068
 [6] 2.449490 2.645751 2.828427 3.000000 3.162278
[12] 3.316625 3.464102

> y=matrix(seq(from=0,by=10,length=12),ncol=4);
> y
     [,1] [,2] [,3] [,4]
[1,]    0   30   60   90
[2,]   10   40   70  100
[3,]   20   50   80  110

> z=mapply(sum,x,y)
> z
```

```
 [1]   1  12  23  34  45  56  67  78  89 100 111 122
> z1=matrix(z,ncol=ncol(x))
> z1
     [,1] [,2] [,3] [,4]
[1,]    1   34   67  100
[2,]   12   45   78  111
[3,]   23   56   89  122
```

Se constată că funcţia `lapplay` are ca rezultat o listă. Pentru a obţine un vector trebuie utilizată funcţia `sapplay` în loc de `lapplay` sau aplicată funcţia `unlist` rezultatului întors de `lapplay`.

Exemplul 3.10. Să calculăm produsul de convoluţie p al vectorilor x şi y. Prin definiţie $p_k = \sum_{i+j=k} x_i y_j$. Indicii vectorului x apar în vectorul `ix`, ai vectorului y în `iy`, iar ai vectorului p în `ip`.

```
> x=1:5;y=c(1,2,2);ix=-2:2; iy=0:2
> x
[1] 1 2 3 4 5
> ix
[1] -2 -1  0  1  2
> y
[1] 1 2 2
> iy
[1] 0 1 2
> r=as.vector(outer(x,y,"*"))
> ir=as.vector(outer(ix,iy,"+"))
> r
 [1]  1  2  3  4  5  2  4  6  8 10  2  4  6  8 10
> ir
 [1] -2 -1  0  1  2 -1  0  1  2  3  0  1  2  3  4
> fir=factor(ir)
> p=tapply(r,fir,sum)
> ip=as.numeric(unlist(dimnames(p)))
> p
-2 -1  0  1  2  3  4
 1  4  9 14 19 18 10
> ip
[1] -2 -1  0  1  2  3  4
> dimnames(p)<-NULL
> p
[1]  1  4  9 14 19 18 10
```

3.7. Funcţii pentru citire şi scriere

Descrierile care urmează sunt incomplete, dar suficiente în aplicaţii uzuale. Prin apel la `help` se pot obţine informaţii suplimentare.

Funcţia	Explicaţii
`print`	`print(x, ...)` Se tipăresc la consolă datele *x*. Printre opţiuni se numără `digits`=numărul maxim de cifre semnificative ce se tipăresc, `justify` = `"left", "right", "centre", "none".`
`cat`	`cat(...,file="",sep="",fill=FALSE,append=FALSE)` Vectorii şi numele din ... sunt scrise în fişierul `file` (dacă nu apare în instrucţiune atunci se scrie la consolă) utilizând separatorul `sep`. `fill` este un număr pozitiv care dă lungimea unei linii în fişier sau `FALSE` când se crează linie nouă doar la apariţia caracterului "\n". Dacă `append=TRUE` se adaugă datele la fişierul existent.

`read.csv,`
`read.table,`
`read.csv2`
`read.delim`
`read.delim2`
	Prin aceste instrucţiuni se citesc date de tip `data.frame` din fişiere text. Pe fiecare linie din fişier trebuie să fie o linie din tabel. Dacă `header=TRUE` atunci prima linie conţine numele coloanelor. Parametrul `row.names` este fie un număr ce indică pe ce coloană este numele liniilor fie un sir de caractere cu numele coloanei.
	`read.csv(file,header=TRUE,sep=",",quote="\"", dec=".")`
	`read.table(file,header=FALSE,sep="",quote="\"'", dec=".",row.names,col.names,,nrows=-1,skip=0, blank.lines.skip=TRUE, comment.char="#",)`
	Parametrii din instrucţiuni se explică singuri prin nume (`dec` este semnul pentru separarea zecimalelor). Dacă lipsesc se pun valorile implicite. `read.csv2, read.delim, read.delim2` sunt la fel ca `read.csv` cu alţi separatori între câmpuri şi alţi delimitatori pentru zecimale.

`readLines`	Se utilizează pentru a citi linii de la o conexiune (de exemplu dintr-un fişier)

`write,`
`write.table,`
`write.csv,`
`write.csv2`
	`write(x,file,ncolumns,append=FALSE,sep=" ")`
	`write.table(x,file="",append=FALSE,quote=TRUE, sep="",eol="\n",na="NA",dec=".",row.names=TRUE, col.names = TRUE)`
	Parametrii care nu se pun efectiv în instrucţiuni se iau implicit. Cu instrucţiuni se scriu date de tip data.frame în fişiere text.

`scan`	`scan(file="",what,nmax=-1,n=-1,sep="",skip=0, nlines=0)`, unde `file` este fişierul, `what` este tipul de date din fişier: logical, integer, numeric, complex, character, raw, list (implicit numeric), `sep` este separatorul de date din fişier (implicit blanc), `skip` este numărul de linii de la începutul fişierul de ignorat, `nlines` este numărul max.de linii ce se citesc.

`format`	`format(x, digits, nsmall, justify, width)` Se specifică felul în care se va tipări *x*, `digits` este numărul de cifre, `nsmall` este numărul de cifre după virgulă (≤20), `justify`="left","right","centre","none", `width` este lăţimea min.
`save`	`save(...,list,file,ascii=FALSE)`, unde ... este lista de variabile ce se salvează, `list` este la fel liste de variabile ce se salvează, `file` este fişierul în care se salvează, `ascii` este `TRUE` sau `FALSE`.
`save.image`	`save.image(file=".RData", ascii=FALSE, compress, safe = TRUE)` `Compress` poate fi "gzip", "bzip2" or "xz" şi specifică tipul de compresie al datelor. Prin această comandă se salvează toate datele din spaţiul de lucru.
`load`	`load(file)`, unde `file` este fişierul de unde se restaurează datele salvate cu `save` sau `save.image`
`ls()`	Rezultatul este o listă cu variabilele active în programul curent
`rm(list)`	Prin această funcţie sunt şterse din memorie variabilele din lista dată de parametrul list. Instrucţiunea `rm(list=ls())` şterge toate variabilele din memorie.

În capitolul despre tipuri de date s-au dat mai multe exemple de citire sau scriere a datelor. În exemplul următor ilustrăm utilizarea funcţiilor `paste`, `cat`, `print`.

Exemplul 3.11 Un vector este scris la consolă cu componentele numite detaliat.

```
> x=1:10
> y=sqrt(x)

> et=paste("y",x,sep="")
> et
 [1] "y1" "y2" "y3" "y4" "y5" "y6" "y7" "y8" "y9" "y10"
> cat(paste(et,"=",format(y,digits=3),";",sep=""),"\n")
y1=1.00; y2=1.41; y3=1.73; y4=2.00; y5=2.24; y6=2.45;
y7=2.65; y8=2.83; y9=3.00; y10=3.16;
> print(y, digits=3)
 [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

3.8. Funcţii de creare de date

Printre funcţiile de creare de date discutate a capitolul despre tipuri de date amintim: `rep`, `seq`, `c`, `matrix`, `array`, `table`, `cbind`, `rbind`. Aceste funcţii au fost utilizate în mai multe rânduri în cpitolul despre tipurile de date.

Funcţia	Exemple

`seq (from, to, by, length.out, along.with=NULL, ...)`	`X=seq(0,1,length=100)`
	`Y=seq(from=1, to=4, by=0.1)`
	`Z=seq(from=10,by=0.3,along=X)`

`rep(x, ...)`. La opţiuni avem `times`, `length.out`, `each` (la `each` se repetă fiecare element al lui *x*)	`rep(x,10); rep(x,times=10);`
	`rep(1:3,length=5);`
	`rep(1:5,each=2)`

	`x<-c(2,3,4)`
`cbind, rbind` combina pe coloanele (liniile) unor vector, ale unei unor matrice sau data.frame	`x=1:3; y=4:6; z<-cbind(x,y);` `u=rbind(x,y)`
`matrix, array` permit crearea de matrice sau blocuri multidimensionale	`x=matrix(1:10, ncol=2)` `y=array(1:24, dim<-c(2,3,4))`

Exemplul 3.12. Ilustrăm cum apar rezultatele la unele modele de funcţii din tabelul de mai sus.

```
> rep(1:3,length=5)
[1] 1 2 3 1 2
> rep(1:5,each=2)
 [1] 1 1 2 2 3 3 4 4 5 5
> x=1:3; y=4:6; z<-cbind(x,y);
> z
     x y
[1,] 1 4
[2,] 2 5
[3,] 3 6
> t=array(1:24, dim<-c(2,3,4))
> t
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 4

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
```

3.9. Funcţii de conversie

Conversia datelor dintr-un tip în altul se poate face prin comenzi ca `as.numeric` `as.integer`, `as.character`, `as.factor`, `as.function`, `as.vector`, `cut` (converteşte un vector numeric într-un factor), `strtoi` (converteşte şiruri de caractere în întregi), `toString` (converteşte un obiect R într-un şir de caractere, `data.matrix` (converteşte un obiect data.frame într-o matrice) etc. Dacă nu e posibilă conversia atunci rezultatul rămâne nemodificat.

Exemplul 3.13. Din doi vectori de aceeaşi dimensiune se construieşte o `data.frame` care se încearcă a se converti în mai multe feluri dintre care unele sunt imposibile.

```
> x=1:5
> y=letters[1:5]
> a=data.frame(x,y)
> a
  x y
1 1 a
2 2 b
3 3 c
4 4 d
5 5 e
> class(a)
[1] "data.frame"
> b=as.character(a)
> b
[1] "1:5" "1:5"
> c=as.matrix(a)
> c
     x   y
[1,] "1" "a"
[2,] "2" "b"
[3,] "3" "c"
[4,] "4" "d"
[5,] "5" "e"
> as.numeric(c)
 [1]  1  2  3  4  5 NA NA NA NA NA
Warning message:
NAs introduced by coercion
```

3.10. Funcţii pentru teste statistice

Pentru teste statistice se poate apela în R la funcţii specializate ca: *t.test, binom.test, prop.test, var.test, wilcox.test, shapiro.test, bartlett.test, Box.test, chisq.test, fisher.test, friedman.test, ks.test,* etc. Informaţii despre aceste funcţii, parametrii de intrare şi rezultatele întoarse, se pot găsi în paginile de ajutor din R. Valoarea de bază întoarsă de un test este `p.value`, notată în exterior ca p-value. Dacă este mai mare decât pragul de risc definit 1-nivelul de încredere (în teste nivelul de încredere este numit `conf.level`) atunci ipoteza este acceptabilă cu o eroare de tipul I sub pragul de risc.

Exemplul 3.14. Din 1000 aruncări cu o monedă, faţa a apărut de 490 ori. Să se testeze dacă probabilitatea p=1/2 de apariţie a feţei este admisibilă cu un prag de risc de 0.05. Următoarea comandă în R face acest test.

```
> rez=binom.test(490, 1000, p=0.5, conf.level=0.95)
> rez
        Exact binomial test
data:  490 and 1000
number of successes = 490, number of trials = 1000,
p-value = 0.548
alternative hypothesis: true probability of success is not
equal to 0.5
95 percent confidence interval:
 0.4585849 0.5214742
sample estimates:
probability of success
                  0.49
```

În cazul nostru `p-value`=0.548 > 0.05=pragul de risc=`1-conf.level`, deci ipoteza că p=0.5 este admisibilă. În plus, avem şi o estimare a lui p sub forma unui interval de încredere cu încrederea `conf.level=0.95`.

Exemplul 3.15. Utilizând testul Kolmogorov-Smirnov să se decidă dacă datele dintr-un vector sunt repartizate conform unei distribuţii date (în cazul de faţă N(1,2)).

```
> x <- runif(500,-1,1)
> rez=ks.test(x, "pnorm",1,2)
> rez
        One-sample Kolmogorov-Smirnov test
data:  x
D = 0.5005, p-value < 2.2e-16
alternative hypothesis: two-sided
```

Am generat un vector aleator uniform cu valori în [-1,1]. Testul `ks` a detectat că datele nu provin dintr-o distribuţie normală cu media 1 şi abaterea standard 2. Acest lucru se reflectă prin faptul că `p-value` este foarte mic, mult mai mic decât pragul de risc uzual 0.05. Dacă schimbăm datele de testare cu un vector repartizat $N(1,2)$, testul `ks` artă că ipoteza este acceptabilă (`p-value` este 0.7422 > 0.05=pragul uzual de risc)

```
> x <- rnorm(500,1,2)
> rez=ks.test(x, "pnorm",1,2)
> rez
        One-sample Kolmogorov-Smirnov test
data:  x
D = 0.0304, p-value = 0.7442
alternative hypothesis: two-sided
```

3.11. Funcţii pentru determinarea duratei de execuţie a unor porţiuni de program

Funcţia	Exemple

`system.time(expr)`	Este determinată durata de execuie a expresiei `expr`.
	De exemplu
	`> system.time(for(i in 1:10000) x <- mean(rnorm(1000, mean=4, sd=5)))`
	`user system elapsed`
	`1.72 0.00 1.72`

`proc.time()`	Este determinat timpul cât calculatorul cheltuie cu anumite proceduri. Se utilizează ca de exemplu:
	`ptm <- proc.time()`
	`a=matrix(runif(1000000),ncol=1000)`
	`b=runif(1000)`
	`x=solve(a,b)`
	`proc.time() - ptm`
	`user system elapsed`
	`0.69 0.08 1.36`

Numărul funcţiilor predefinite în R este mult mai mare dacât am descris noi aici şi pot apare funcţii noi în versiunile ulterioare. Funcţiile descrise în tabelele de mai sus fac parte din pachetul *base* şi din pachetul *stats*, şi foarte probabil vor rămâne şi în versiunile ulterioare de R. Pachetele adiţionale conţin multe funcţii suplimentare care sunt orientate spre rezolvarea unor probleme specifice. Vom mai decrie ulterior unele funcţii din pachetul *tseries*, funcţii orientate spre lucrul cu serii temporale.

Capitolul 4
Funcţii grafice în R

În R se pot executa grafice de calitate pentru datele admise. Aceste grafice sunt trimise pe ecran sau în fişiere de diverse tipuri. Pentru a realiza acest lucru trebuie deshis un "device". Comanda *windows()* deschide ecranul ca dispozitiv grafic, comanda *jpeg()* deschide ca dispozitiv grafic un fişier în care se va scrie informaţia grafică în format jpeg, *postscript()* deschide un fişier de tip postscript, *pdf(), png(), tiff(), bitmap(), win.metafile(),* etc. Aceste comenzi au ca parametru *file* numele fişierului unde se salvează graficul, *width*=lăţimea în pixeli, *height*=înălţimea în pixeli. Numai un dispozitiv este activ la un moment, trecerea de la unul la altul se face *dev.next(), dev.prev(), dev.set(which=k)*. Listarea dispozitivelor grafice se face cu comanda *dev.list()*, închiderea unui dispozitiv cu *dev.off(k)*, iar închiderea tuturor cu *graphics.off()*. Când s-a închis un dispozitiv diferit de ecran, imaginea din el este salvată în fişierul specificat în parametrul *file*. După comanda *graphics.off()* trebuie deschis un nou dispozitiv pentru a putea tipări grafic, de regulă *windows()* pentru ecran. Un clic dreapta pe fereastra cu reprezentarea grafică de pe ecran deschide un meniu cu opţiuni de salvare sau copiere a conţinutului în clipboard.

Rutinele grafice sunt cuprinse în modulul de bază: "base" precum şi în două pachete suplimentare "grid" şi "lattice". Rutinele de bază se împart în trei categorii: rutine de nivel înalt "high level" în urma cărora se crează o fereastră grafică nouă în care se pun graficele specificate prin argumentele rutinelor; rutine de nivel jos "low level" prin care la graficul existent se adaugă informaţii suplimentare ca text, puncte, linii; grafice interactive prin care utilizatorul poate interacţiona cu graficul prin intermediul mausului pentru a obţine informaţii despre valorile din diverse poziţii de pe grafic sau pentru a adăuga informaţii în diverse poziţii de pe grafic.

Rutinele de nivel înalt din modulul de bază sunt:

Instrucţiunea	Explicaţii
plot	*plot(x,y,...), plot(xy,...),* tipăreşte punctele de coordonate (*x*[i],*y*[i]) utlizând opţiunile grafice ... (vezi mai jos). *plot(f,x)* unde *f* este un factor pentru *x* tipăreşte un *boxplot* pentru *x* grupat după factorul *f*. *plot(x~y)* face graficele coloanelor $y^{<i>}$ în funcţie de *x*. *plot(x)* pentru o variabilă care are metoda plot produce un grafic specific acelui tip de variabilă.
pairs	*pairs (x)* unde *x* este o matrice face toate graficele $\left(x^{<i>}, x^{<j>}\right)$ unde $x^{<i>}$ şi $x^{<j>}$ sunt coloane ale lui *x*
coplot	*coplot(x~y,f)* execută graficele lui *y* depinzând de *x*, pentru fiecare nivel al factorului *f*
hist	*hist(x,...)* tipăreşte histograma frecvenţelor valorilor din vectorul *x*. Opţiunea frecvent utilizată este *nclass=n* pentru a specifica în câte clase se împart datele din *x*. Cu opţiunea *probability=TRUE* histograma va reprezenta probabilităţile nu frecvenţele.

boxplot	*boxplot(x,...)* sau *boxplot(x~y,data=z,...)*. În primul caz se execută un *boxplot* pentru valorile din vectorul sau lista de vectori *x*, iar în al doilea caz se execută un box plot pentru valorile din coloana *x* grupate după factorul *y*, date ce se află în variabila *z* ce trebuie să aibă date numite *x* şi *y* (de exemplu *z* este data.frame iar *x*, *y* sunt două coloane în *z*).
qqnorm, qqplot, qqline	*qqnorm(x)* reprezintă grafic cuantilele lui *x* faţă de cele ale distribuţiei normale standard, *qqplot(x,y)* reprezintă grafic cuantilele lui *x* faţă de cele ale lui *y*, *qqline(x)* reprezintă pe lângă graficul ca în *qqnorm(x)* şi linia ce aproximează graficul.
barplot	*barplot(x,...)*, unde ... sunt opţiuni, tipăreşte sub formă de bare verticale valorile lui *x*
dotchart	*dotchart(x,...)* la fel ca *barplot(x,...)* face graficul valorilor lui *x* sub formă de puncte
image	*image(x,y,z,...), image(z,...)* reprezintă matricea *z* prin culori; *x*[*i*] şi *y*[*j*] reprezintă coordonatele centrului dreptunghiului unde se reprezintă prin culoare valoarea *z*[*i,j*]
contour, filled.contour	*contour(x,y,z, nlevels=n,...)* reprezintă grafic liniile de contur ale valorilor *z*[*i,j*] din matricea *z*, având pe axe gradaţiile date de vectorii *x*, respectiv *y*. Opţiuni suplimentare se pot afla cu help(contour). Analog pentru *filled.contour*, unde spaţiul între două linii de contur e colorat.
persp	*persp(z), persp(x,y,z), persp(x,y,z,theta=a,phi=b,...)* desenează o suprafaţă pe care sunt punctele de coordonate (*x*[*i*], *y*[*j*], *z*[*i,j*]). Direcţia de vedere a suprafeţei se ajustează prin parametrii *theta* şi *phi* (in grade), iar *ltheta* şi *lphi* dau direcţia de unde vine lumina pe suprafaţă, *r* dă distanţa de la care se vede (*r* mare e ca vederea de la infinit). Culoarea este controlată prin parametrul opţional *col*. Pentru o variaţie continuă a culorii *col* trebuie să fie un vector de culori de dimensiune egală cu numărul de faţete (vezi exemplul de mai jos). Pentru a obţine o senzaţie de relief mai puternică se utilizează opţiunea *shade=număr* de obicei între 0 şi 1, *thicktype* poate fi “simple” sau “detailed”, *axes* este TRUE sau FALSE după cum se pun sau nu marcajele pe axe, *xlim, ylim, zlim* dau valorile limită pe axe, valori care sunt cuprinse în grafic, *main* şi *sub* sunt titlul şi subtitlul, *xlab,ylab, zlab* sunt titlurile axelor. Dacă *scale* = TRUE atunci nu se păstrează proporţia pe cele trei axe.

Prin tipărirea cu rutine de nivel jos se adaugă elemente noi la graficul existent. Mai jos sunt descrise unele rutine grafice şi unii parametri ai acestora. Restul se pot afla prin comanda `help`.

Rutina grafică	Explicaţii
points	*points(x,y,...)* tipăreşte puncte în coordonatele specificate de *x* şi *y*
lines	*lines(x,y)* adaugă linii între punctele de coordonate specificate în vectorii *x*, *y*.
text	*text(x,y,text,...)* determină scrierea începând din poziţia (*x*[*i*],*y*[*i*]) a caracterelor din *text*[*i*]. Un parametru auxilar numit *pos* ce poate avea valorile 1, 2, 3, 4 controlează dacă textul este sub, la stânga, deasupra sau la dreapta punctului (*x*, *y*).

mtext	*mtext(text,...)* scrie un text pe o margine a graficului. Printre parametrii opţionali : *side* pentru a controla marginea (1=jos, 2=stânga, 3=sus, 4=dreapta), *col* este culoarea pentru text, *line*=întreg controlează depărtarea faţă de marginea corespunzătoare a graficului (în linii).
abline	*abline(a=valoare1, b=valoare2)* trasează graficul lui y=a+bx, *abline(h=valoare)* trasează linia orizontală prin y=h, *abline(v=valoare)* trasează linia verticală prin x=v., *abline(coef=vector)* trasează linia care are în *coef* ordonata la origine şi panta, *abline(obiect)* trasează o linie de parametrii coef din obiectul *obiect* (dacă există).
axis	*axis(side,...)* trasează o axă în funcţie de parametrul *side* (1=jos, 2=stânga, 3=sus şi 4=dreapta). Există mai mulţi parametri opţionali.
segments	*segments(xo,yo,xd,yd,...)* trasează segmente de la (xo,yo) la (xd,yd). Dacă xo, yo, xd, yd sunt vectori atunci se trasează mai multe segmente. Printre parametrii opţionali amintim *col* care este egal cu indicele culorii segmetului (*col* poate fi un nume de culoare).
arrows	*arrows(xo,yo,xd,yd,...)* este ca şi rutina *segments* de mai sus doar că trasează săgeţi de la (xo,yo) la (xd,yd)
rect	*rect(xleft, ybottom, xright, ytop,...)* traează o mulţime de dreptunghiuri în care *xleft* reprezintă coordonatele lor x stânga, *ybottom* reprezintă coordonatele lor y jos, etc. Printre parametrii opţionali menţionăm *col* un vector cu indicii de culoare din paleta curentă, *density* un vector cu numărul de linii de haşurare, *lty* un vector cu tipurile de linie de haşură, *angle* un vector cu unghiurile faţă de Ox aleliniilor de hşurare.
polygon	*polygon(x,y,...)* trasează un poligon cu vârfurile consecutive date de vectorii x şi y. Printre opţiuni sunt importante *col*=culoarea cu care se umple interiorul poligonului, *density*=numărul de linii cu care se haşurează poligonul, *angle*=unghiul cu axa Ox al liniilor de haşurare (în grade)
box	*box(lty=tip, col=culoare)* determină trasarea unui dreptunghi în jurul graficului cu tipul de linie tip şi culoarea culoare.
grid	*grid(nx,ny, col=culoare, lty=tiplinie)* datermină trasarea unui caroiaj cu culoarea culoare şi tipul de linie *tiplinie*. Se mai poate introduce parametrul *lwd* care dă grosimea liniilor de caroiaj. Pe axa Ox sunt *nx* diviziuni, iar pe axa Oy sunt *ny* diviziuni.
legend	*legend(x,y, legenda, col=vector, text.col=vector, lty=vector, pch=vector*, *bg=,...)* unde *(x,y)* este poziţia pe grafic al colţului stânga sus al legendei (se poate indica poziţia prin cuvinte "topleft" sau "bottomright" sau prin instrucţiunea *locator()* caz în care este aleasă cu mausul), *legenda* este un vector cu elemente text (se poate da textul din legendă prin *legend*=textul , *col* este un vector cu indicii culorilor din paleta curentă de culoare pentru culorile marcajelor din legendă, *text.col* este un vector cu indicii de culoare pentru textele din legendă, *lty* este un vector de întregi cu indicii tipurilor de linie din grafice, *pch* este un vector cu indicii tipurilor de linie din legendă, *bg* este indicele culorii de fundal din legendă. Culorile, tipurile de linie sau tipurile de puncte pot fi specificate şi prin cuvinte)

title	*title(main=text1,sub=text2,xlab=text3,ylab=text4,...)* determină scrierea unui titlu(main), subtitlu(sub), etichete pentru axe(xlab, ylab) care sunt luate din text1, text2, etc.

Pentru diverşi parametri grafici R menţine o listă numită *par* cu valorile acestora. Cu comanda *par()* putem afla valorile acestor parametri. Cei mai utilizaţi parametri grafici sunt în tabelul următor:

Parametrul grafic	Valoare implicită	Explicaţii
xlog	FALSE	Dacă este TRUE pe axa x se utilizează scara logaritmica
ylog	FALSE	Analog cu xlog dar pentru axa y
pch	1	*pch* este un vector valori întregi care specifică tipul de punct de pe grafic. Valorile întregi sunt de regulă între 0 şi 255. După epuizarea tipurilor din pch ele se se repetă.
col	"black"	*col* este un vector de întregi care dă indexul culorii curente din paleta curentă de culori. Culoarea se poate exprima şi prin cuvinte (există 667 de culori care au nume care se pot afla cu comanda *colors()*). Culorile se repetă ciclic atunci când se reprezintă diverse obiecte grafice.
col.axis	"black"	Culoarea axelor
col.lab	"black"	Culoarea pentru etichete pe axe
col.main	"black"	Culoarea pentru titlu
col.sub	"black"	Culoarea pentru subtitlu
cex, cex.axis, cex.lab, cex.main, cex.sub	1	Un factor de amplificare pentru mărimea textului faţă de mărimea implicită. Doar *cex.main* are valoarea implicită 1.2
adj	-1	Se controlează cum apare textul justificat faţă de punctul de referină: 0 aliniat la stânga, 1 aliniat la dreapta , 0.5 centrat orizontal
font, font.axis, font.lab, font.main font.sub	1	Un întreg ce reprezintă fontul pentru text, axe, etichete, titlu, subtitlu.
fg, bg	"black", "white"	Culorile pentru foreground respectiv background
lty	"solid"	Tipul de linie: 0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash
lwd	1	Grosimea liniei. Depinde de tipul de dispozitiv grafic pe care se tipăreşte
mai	1.02 0.82 0.82 0.42	Un vector ce conţine dimensiunile în inci ale marginilor în ordinea: jos, stânga, sus, dreapta
mar	5.1 4.1 4.1 2.1	La fel ca la *mai* dar dimensiunea este în linii
mex	1	Factor de amplificare pentru mărimea fontului cu

		care se scrie pe margine
mfcol	1 1	Vector de tipul c(nr,nc). Ecranul va fi împărţit în $nr \times nc$ regiuni. Figurile consecutive vor fi scrise pe coloană.
mfrow	1 1	Vector de tipul c(nr,nc). Ecranul va fi împărţit în $nr \times nc$ regiuni. Figurile consecutive vor fi scrise pe linie.
mfg	1 1 1 1	Un vector de tipul c(i,j,nr,nc) prin care se specifică în ce regiune de pe ecran se scrie. Se acceptă şi valori de tipul c(i,j)
fig		Printr-un vector de patru numere între 0 şi 1 se controlează poziţia figurii în pagină faţă de colţul stânga-jos.
xaxs, yaxs		Controlează stilul marcajelor pe axe şi pot fi "r", "i", "e", "s", "d".
xaxp, yaxp	0 1 5	Au forma c(n1,n2,n) unde n1 şi n2 reprezintă coordonatele extreme ale punctelor de marcaj (tick marks) de pe axe, iar n este numărul dorit de marcaje
lab	5 5 7	Vector de tipul c(n1,n2,n): n1, n2 reprezintă numerele dorite de marcaje pe axe, iar nu e implementat.
las		Este un întreg 0, 1, 2, sau 3, şi controlează felul în care se trasează marcajele pe axe.
xaxt, yaxt	"s"	Controlează tipul axelor "n", "l" , "t", "s".
usr	0 1 0 1	Vector de tipul c(x1,x2,y1,y2) pentru valorile extreme pe coordonate
tcl	-0.5	Lungimea marcajelor pe axe ca fracţie din lăţimea unui rând de text.
srt	0	Unghiul de rotaţie al şirurilor de caractere

Unii parametri sunt admişi ca date opţionale de unele rutine grafice. La apelul rutinei respective, parametrul considerat se introduce sub forma *rutina(..., parametru=valoare,...)*. După execuţia comenzii grafice valoarea parametrului revine la cea dinainte de instrucţiunea grafică. Dacă vrem ca modificarea să fie permanentă (până la o nouă modificare) atunci vom utiliza comada *par(parametru=valoare,...)*. Unii parametrii pot fi modificaţi doar prin instrucţiunea par. Aceştia sunt: "ask", "fig", "fin", "lheight", "mai", "mar", "mex", "mfcol", "mfrow", "mfg", "new", "oma", "omd", "omi", "pin", "plt", "ps", "pty", "usr", "xlog", "ylog", "ylbias".

Înainte de a da exemple de instrucţiuni grafice, câteva cuvinte despre culori. Culorile implicite pe care le utilizează R la un moment dat pentru reprezentări grafice sunt stocate într-o paletă, un vector de tip caracter cu numele culorilor. Le putem vedea cu comanda *palette()*.

```
> palette()
[1]   "black"   "red"   "green3"   "blue"   "cyan"   "magenta"
[7] "yellow" "gray"
```

Pentru culorile care nu au nume se utilizează descrierea lor hexazecimală. Putem crea palete de culori prin *palette(rainbow(n)), palette(heat.colors(n)), palette (terrain.colors(n)), palette(topo.colors(n)), palette(cm.colors(n))*, unde *n* este numărul culorilor pe care le vrem în paletă. O altă modalitate de a obţine palete este prin interpolarea culorilor cu funcţia c*olorRampPalette* ca mai jos:

```
> culorile.mele=colorRampPalette(c("blue","yellow","red"))
> culorile.mele
function (n)
{
    x <- ramp(seq.int(0, 1, length.out = n))
    rgb(x[, 1], x[, 2], x[, 3], maxColorValue = 255)
}
<bytecode: 0x01aa3ae0>
<environment: 0x02c36ab8>
> paleta=palette(culorile.mele(20))
> paleta
[1] "blue"    "#1A1AE4" "#3535C9" "#5050AE" "#6B6B93"
[6] "#868678" "#A1A15D" "#BBBB43" "#D6D628" "#F1F10D"
[11]"#FFF100" "#FFD600" "#FFBB00" "#FFA100" "#FF8600"
[16]"#FF6B00" "#FF5000" "#FF3500" "#FF1A00" "red"
```

Se vede că funcţia c*olorRampPalette* ne întoarce ca rezultat o funcţie de calcul a culorilor. Această funcţie aplicată unui număr natural n crează o paletă de lungime n. Putem vedea culorile din paletă tipărind de exemplu:

```
> plot(1:20, col=paleta, pch=1:20, main="Puncte si culori")
```

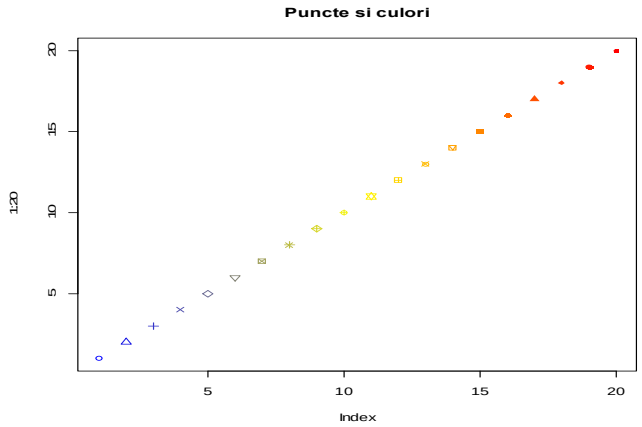


Fig 4.1

Un clic dreapta pe grafic face să apară un meniu contextual unde apar opţiunile de salvare a graficului în format metafile sau postscript sau de copiere în clipboard în format metafile sau bitmap (în windows). De aici este uşor să-l includem într-un document.

Pe lângă parametrii din vectorul *par* în comenzile grafice de nivel înalt apar şi alţi parametri opţionali:

Parametrul	Explicaţii
add	Dacă este TRUE atunci noul grafic se suprapune peste cel existent. Funcţionează uneori.
axes	Dacă este FALSE nu se trasează axele
log=”x”, log=”y”, log=”xy”	Se specifică ce axe sunt gradate logaritmic. Acest lucru se poate face şi prin parametrii grafici, *xlog, ylog*.
type	Poate lua lua valorile "p" pentru a tipări puncte, “l” pentru a tipări linii, “b” pentru a tipări puncte şi linii între ele, "o" pentru a tipări puncte şi linii peste ele, "s" sau "S" pentru a tipări grafice sub formă pas (step) în sus sau jos, "n" pentru a se seta sistemul de coordonate, fără a trasa graficul
xlab, ylab	Sunt şiruri de caractere pentru etichetarea axelor
main, sub	Sunt şiruri de caractere pentru titlu şi subtitlu
xlim, ylim, zlim	Sunt vectori numerici cu câte două valori pentru a seta limitele pe cele trei axe

În cele ce urmează exemplificăm utilizarea în parte a rutinelor grafice, cu comentarii.

Exemplul 4.1. Facem graficele a trei funcţii, conform scriptului următor:

```
x=seq(-pi,pi,length=20)
y1=sin(x); y2=cos(x); y3=2*x/(1+x^2)
plot(x,y1,type="b", pch="+", xlab="", ylab="", lwd=3)
lines(x,y2, type="o", pch=10, col="red", lty="dotted",
lwd=3)
lines(x,y3,type="l", lty="dashed", col="blue", lwd=3)
title(main="Un exemplu de grafic", sub="Graficele a trei
functii",xlab="axa X",ylab="axa Y", cex.main=2,
cex.lab=1.5, cex.sub=2)
grid(5,5,col="cyan")
legend("topleft", legend=c("sin","cos","alta")
,col=c("black","red","blue"),
lty=c("solid","dotted","dashed")
,bg="yellow1")
```

Rezultatul este în Fig 4.2.

Observaţii:

a) Este bine când se reprezintă grafic mai multe funcţii, la apelul rutinei plot să se specifice *xlim=range(x1,x2,x3...), ylim=range(y1,y2,y3,..)* unde *x1, x2, x3,* .. sunt vectorii cu abscisele coordonatelor punctelor de pe grafic iar *y1, y2, y3,* .. conţin ordonatele punctelor. Acest lucru ne asigură că toate graficele vor fi complet reprezentate. Dacă la reprezentările din figura de mai sus mai adăugăm un grafic ce are puncte în afara zonei delimitate (în cazul nostru [-3,3]×[-1,1]) acestea nu vor apare.

b) Primul grafic se execută cu *plot* iar celelalte cu *lines*, *ponts* sau alte comenzi de nivel inferior pentru a se adăuga elementele la ceea ce este deja reprezentat.

c) Titlurile se pot adăuga într-o instrucţiune *plot,* sau prin comanda *title.* Atenţie la faptul că adăugarea ulterioară a unui titlu nu schimbă altul existent ci se suprapune celui existent.

d) Poziţionarea legendei se poate face prin cuvintele “topleft”, “topright”, “top”, bottomleft”, “bottomright”, “bottom”, “lft”, “right”, sau prin coordonatele *x*, *y*

ale colţului stânga sus, coordonate ce pot fi alese interactiv prin *locator()*. Instrucţiunea *legend* din exemplul precedent ar fi putut fi

```
legend(locator(),legend=c("sin","cos","alta"),
col=c("black","red","blue"),
lty=c("solid","dotted","dashed"),
bg="yellow1")
```

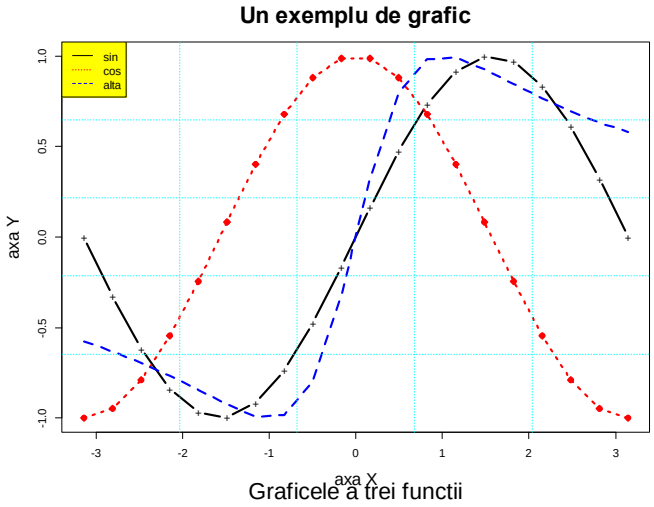


Fig 4.2

Exemplul 4.2. Graficul unei suprafeţe văzută din diverse direcţii. Graficele sunt grupate într-o singură figură divizată în 2x2 zone prin comanda *par(mfrow(2,2))*.

```
f<-function(x,y){z=x-sin(2*x)^2+2*y^2}
x=seq(-1,1,length=20); y=x;
z=outer(x,y,f);

f<-function(x,y){z=x-sin(2*x)^2+2*y^2}
x=seq(-1,1,length=20); y=x;
z=outer(x,y,f);

par(mfrow=c(2,2))

persp(x,y,z,col="lightblue", shade=0.5, theta=45, phi=30,
r=50,
xlim=1.1*range(x),ylim=1.1*range(y),zlim=1.1*range(z),
main="Primul grafic", sub="Graficul functiei z=x-
sin(2*x)^2+2*y^2",
xlab="axa X",ylab="axa Y", zlab="Z",
ticktype="detailed", cex.main=2, cex.sub=2)

paleta=palette(rainbow(32))
clase=cut(z,32)
par(mfg=c(1,2))
persp(x,y,z,col=paleta[clase], shade=0.5, theta=30,
phi=15, r=50,
xlim=1.1*range(x),ylim=1.1*range(y),zlim=1.1*range(z),
```

```
main="Al doilea grafic", sub="Graficul functiei z=x-
sin(2*x)^2+2*y^2",
xlab="axa X",ylab="axa Y", zlab="Z",
ticktype="detailed",cex.main=2, cex.sub=2)

paleta1=palette(cm.colors(32))
clase=cut(z,32)
persp(x,y,z,col=paleta1[clase],    shade=0.5,    theta=20,
phi=45, r=50,
xlim=1.1*range(x),ylim=1.1*range(y),zlim=1.1*range(z),
main="Al treilea grafic", sub="Graficul functiei z=x-
sin(2*x)^2+2*y^2",
xlab="axa X",ylab="axa Y", zlab="Z",
ticktype="detailed",cex.main=2, cex.sub=2)

paleta2=palette(colorRampPalette(c("maroon", "gold"))(32))
clase=cut(z,32)
persp(x,y,z,col=paleta2[clase], shade=1, theta=30, phi=45,
r=50,
xlim=1.1*range(x),ylim=1.1*range(y),zlim=1.1*range(z),
main="Al patrulea grafic", sub="Graficul functiei z=x-
sin(2*x)^2+2*y^2",
xlab="axa X",ylab="axa Y", zlab="Z",
ticktype="detailed",cex.main=2, cex.sub=2)

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

Rezultatul este în Fig 4.3. Se constată că nu se respectă paletele de culori utilizate la fiecare figură. Când se utilizează o singură culoare pentru suprafaţă ea este imediat luată imediat în consideraţie pe când dacă se variază culoarea de la o faţetă la alta atunci paletele utilizate se permută cumva între grafice.

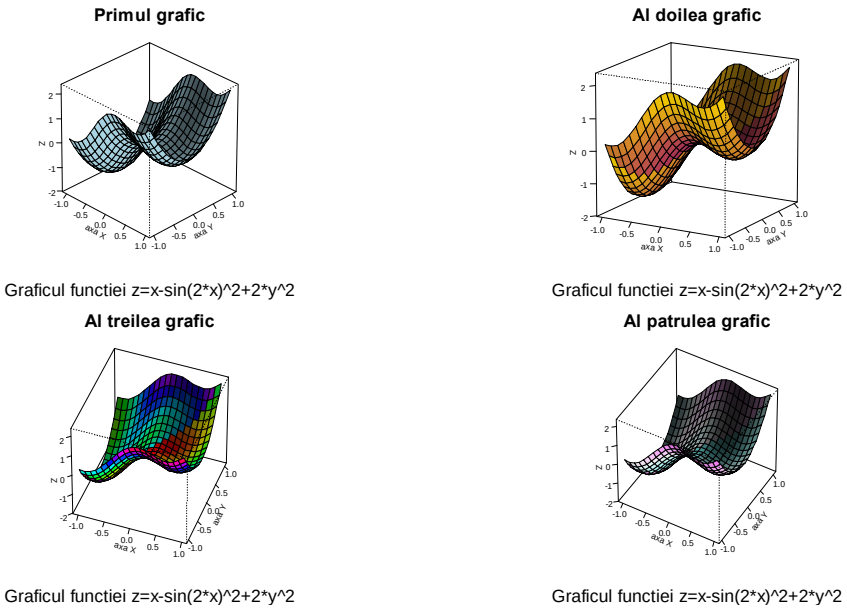


Fig 4.3

Exemplul 4.3. Pentru adnotări matematice pe grafice se pot obţine informaţii din help prin comanda *help(plotmath)*. Expresiile matematice se alcătuiesc cu comanda *expression(...)* în care se introduce sub forma unei expresii apropiate de cea latex formula matematică dorită. Pentru a lipi între ele mai multe şiruri se utilizează comanda *paste(sir1, sir2,...)*. În tabelul de mai jos avem câteva astfel de expresii.

Expresia în R	**Forma pe grafic după executia comenzii *text(locator(), ex)***
ex=expression(paste(x*y, ", ", x%*%y,", ", x/y, ", ", x%/%y,", ",x%+-%y))	$xy,\ x \times y,\ x/y,\ x \div y,\ x \pm y$
ex=expression(paste(x^2,", ",x[2],", ",sqrt(x),", ",sqrt(x,3)))	$x^2,\ x_2,\ \sqrt{x},\ \sqrt[3]{x}$
ex=expression(paste(x == y,", ",x!=y,", ",x<=y,", ",x %~~% y,", ",x %==% y))	$x = y,\ x \neq y,\ x \leq y,\ x \approx y,\ x \equiv y$
ex=expression(paste(cdots,", ",x %subset% y,", ",ldots,", ",x %notsubset% y,", ",x %in% y,", ",x %notin% y))	$\cdots,\ x \subset y,\ \ldots,\ x \not\subset y,\ x \in y,\ x \notin y$
ex=expression(paste(hat(x),", ", tilde(x),", ",dot(x),", ",ring(x),", ",bar(xyzu),", ",widehat(xyzu)))	$\hat{x},\ \tilde{x},\ \dot{x},\ \mathring{x},\ \overline{xyzu},\ \widehat{xyzu}$
ex=expression(paste(x %->% y,", ",x %up% y,", ",x %=>% y,", ",alpha,", ",Alpha,", ",infinity,", ",nabla))	$x \to y,\ x \uparrow y,\ x \Rightarrow y,\ \alpha,\ A,\ \infty,\ \nabla$
ex=expression(paste(underline(x),", ", frac(x, y),", ",sum(x[i], i==1, n),", ",integral(f(x)*dx, a, b),", ",union(A[i], i==1, n),", ",lim(f(x), x %->% 0)))	$\underline{x},\ \frac{x}{y},\ \sum_{i=1}^{n} x_i,\ \int_a^b f(x)dx,\ \bigcup_{i=1}^{n} A_i,\ \lim_{x \to 0} f(x)$
ex=expression(paste(y=sqrt(frac(x^2-1,x^2+1))%*%sin(x)))	$y = \sqrt{\frac{x^2-1}{x^2+1}} \times \sin(x)$
ex=expression(paste(bgroup("(",frac(x+y,x-y)+frac(x^2+y^2,x^2-y^2),")")^2))	$\left(\frac{x+y}{x-y}+\frac{x^2+y^2}{x^2-y^2}\right)^2$

Utilizarea comenzii *text(locator(), ex)* pentru a pune expresia pe grafic se justifică prin aceea că *locator()* aşteaptă un clic stânga pe o poziţie din grafic pentru a se stabili unde va fi pus textul. După aceea se face un clic dreapta şi din meniul derulant se alege opţiunea stop. Mai jos avem două grafice cu adnotări.

```
f<-function(x) ((x^2-1)/(x^2+1)+sin(x))^2
g<-function(x) (ln(2+sin(x))+1)/(2+cos(2*x))
x=seq(-1,1,length=21); y1=f(x); y2=sin(x);
plot(x,y1,ylim=range(y1,y2),type="l", lty=3, col="blue",
lwd=4, xlab="x",
ylab="f(x),g(x)", cex.lab=1.5)
lines(x,y2,type="l", lty=4, col="red",lwd=4)
grid(4,4,col="black")
title(main="Adnotari matematice pe grafice",
sub="Graficul a doua functii", cex.sub=1.5)
```

```
ex1=expression(paste(f(x)%==% bgroup("(",frac(x^2-
1,x^2+1)+sin(x),")")^2))
text(locator(),ex1,cex=1.5)
ex2=expression(paste(g(x)%==%
bgroup("(",frac(log(2+sin(x))+1,2+cos(2*x)),")")))
text(locator(),ex2, cex=1.5)
```

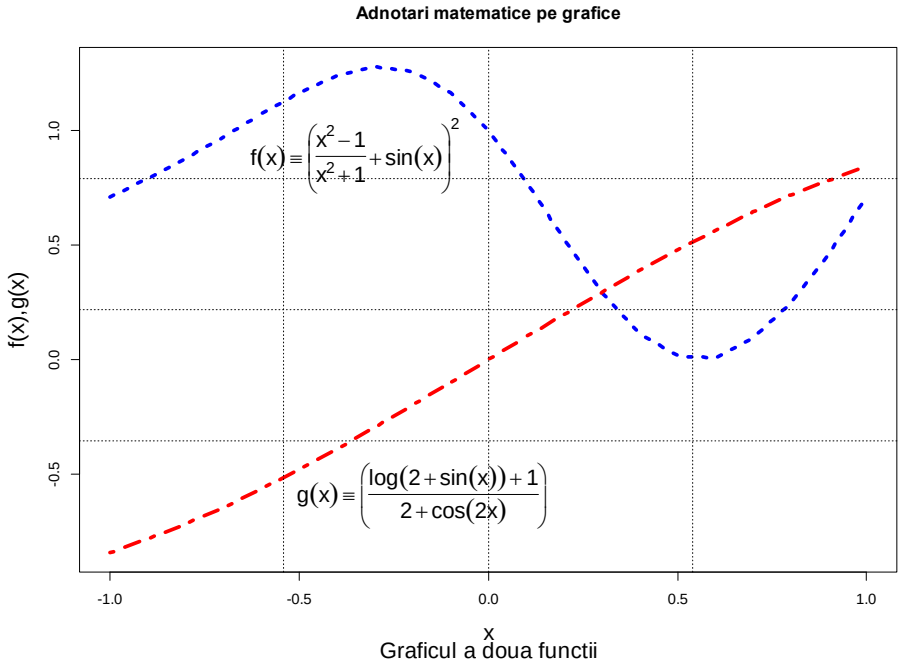


Fig 4.4

Exemplul 4.4. Histograma valorilor generate aleator după o distribuţie normală este comparată cu graficul distribuţiei.

```
media=2; devs=4;
y=rnorm(nr, mean=2, sd=4)
hist(y, probability=TRUE, col="lightgray",
main="Histograma valorilor y=rnorm(200, mean=2, sd=4)",
xlab="Valorile lui y",
ylab="Densitatea valorilor lui y", cex.axis=1.5,
cex.lab=1.5 )
x1=seq(-6,10,length=100)
y1=dnorm(x1,mean=2, sd=4)
lines(x1,y1, lwd=3)
ex=expression(paste(rho, "(", x, ")=",frac(1,
4*sqrt(2*pi)), " ",
plain(e)^{frac(-(x-2)^2, 2%*%4^2)}))
text(locator(),ex, cex=1.5)
```

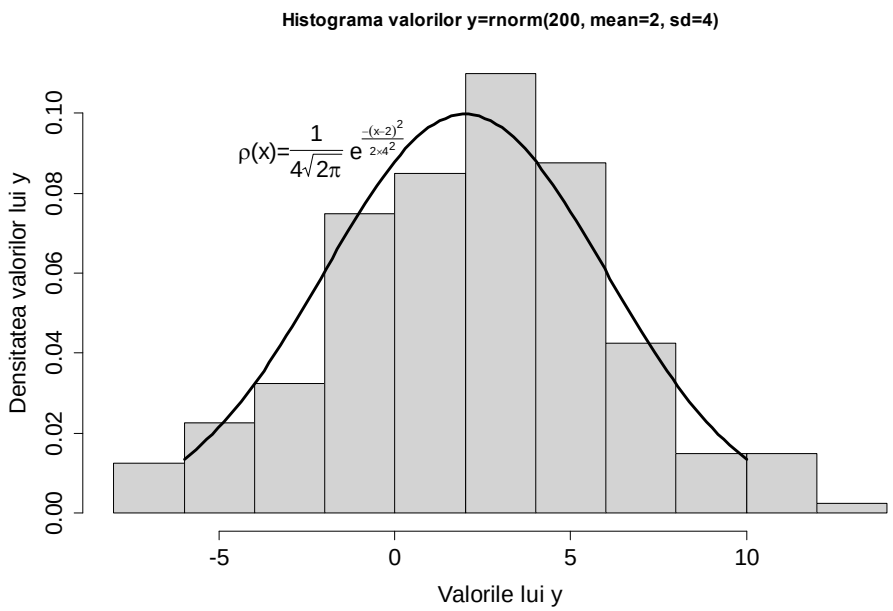


Fig 4.5

Exemplul 4.5. Utilizăm *qqnorm* pentru a detecta dacă o selecţie este normală (graficul ar trebui să fie o dreaptă) şi *qqplot* pentru a detecta dacă două selecţii empirice au aceeşi distribuţie (graficul ar terbui să fie prima bisectoare).

```
x1=runif(1000);x2=rnorm(1000, mean=1, sd=1)
par(mfrow=c(1,2));
qqnorm(x2, main="Q-Q norm pentru o selectie normala",
xlab="Cuantilele teoretice",
ylab="x2=rnorm(100, mean=1, sd=1)", pch="+");
qqplot(x1,x2, main="Q-Q plot pentru doua selectii",
pch="+",
xlab="x1=runif(100)", ylab="x2=rnorm(100, mean=1, sd=1)");
par(mfrow=c(1,1))
```

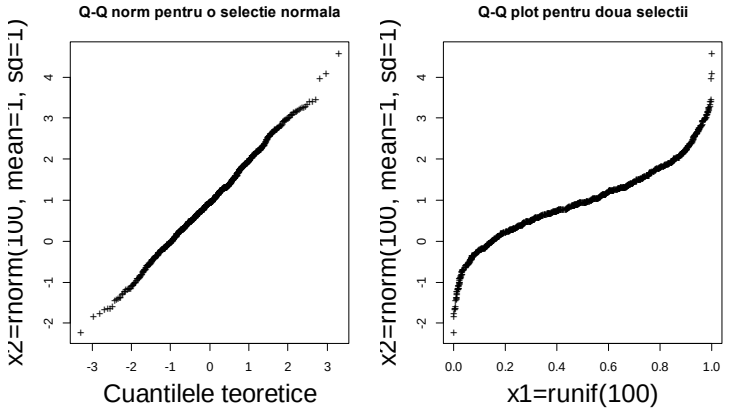


Fig 4.6

Capitolul 5
Programarea în R

În principiu, un limbaj de programare ne pune la dispoziţie un anumit număr de tipuri de date şi un anumit număr de operaţii predefinite asupra lor, modalităţi de introducere a datelor într-un program precum şi de extragere a rezultatelor (instrucţiuni read/write), instrucţiuni de control al fluxului de operaţii executate (în principal instrucţiuni de decizie şi repetiţie), instrucţiuni de creare de subprograme (funcţii). În funcţie de limbaj există şi alte aspecte importante care pot interesa: posibilitatea de a utiliza rutine scrise în alte limbaje, posibilitatea de utilizare recursivă a funcţiilor, programarea pe obiecte, etc. În cele ce urmează descriem succint elementele de bază ale programării în limbajul R.

Codul scris în limbajul R este cod interpretat, deci instrucţiunile sunt compilate la un stadiu intermediar (bytecode) de unde sunt transformate în cod depenedent de maşină în momentul execuţiei. Acest proces de interpretare face ca timpul de execuţie să fie mai lung decât pentru programele compilate de la început în cod-maşină executabil.

Scrierea unui program R se poate face în orice editor text. Se poate utiliza editorul propriu (apelabil cu opţiunea "New script" din meniul File) care are avantajul că din el se poate rula o porţiune din program selectată în prealabil prinr-un clic dreapta şi alegerea opţiunii "Run line or selection" din meniul de context. Dacă programul a fost salvat pe disc în fişierul prog.r atunci rularea lui se face cu comanda source("prog.r") dacă este în directorul curent, altfel trebuie indicată şi calea.

Toate instrucţiunile care se scriu la consolă într-o sesiune de lucru pot fi salvate la final în fişierul text .Rhistory ("Save history" din meniul File) de unde se pot rula din nou cu comanda "Load history" (din meniul File). În acest fel se poate repeta prelucrarea unor date într-o sesiune nouă sau se poate scrie un program după ce se execută pas cu pas instrucţiunile lui ca şi comenzi la consolă.

5.1. Tipurile de date în R. Citirea şi scrierea lor. Operaţii predefinite

Capitolul 2 este dedicat tipurilor de date, operaţiilor asupra lor şi modalităţilor de a le crea, citi sau extrage din mediul R. În capitolul 3 se da o listă mai extinsă a operaţiilor predefinite asupra datelor recunoscute de R. Aceste operaţii sunt prezentate sub forma unor funcţii. Aici menţionăm doar că datele simple cu care lucrează R sunt alcătuite din numere, valori booleene, caractere (şiruri de caractere), date binare (raw). Tipurile fundamentale de date compuse din date simple sunt vectorii, matricele, blocurile multidimensionale (array), data.frame, listele. Se utilizează de asemenea NA pentru date indisponibile şi NaN pentru valori nenumerice. Pot fi create tipuri noi de date.

Un lucru important care trebuie avut în vedere este posibilitatea de grupare a instrucţiunilor prin acolade. Astfel {instr1; instr2; instr3} este un grup. Aceste grupuri apar în general în instrucţiuni de decizie sau repetiţie. Dacă sunt pe acelaşi rând instrucţiunile se separă prin ";" iar dacă sunt pe rânduri diferite nu e nevoie de vreun semn de separaţie.

Pentru comentarea unor secţiuni de program se utilizează caracterul # la începutul liniei de comentariu.

După efectuare unui calcul dacă vrem să păstrăm rezultatele ele trebuie atribuite unei variabile. Atribuirea se face prin (variabila care primeşte valoarea atribuită este A):

```
A<-expresie
A<<-expresie
A=expresie
expresie->A
expresie->>A
```

Atribuirile <- sau -> sunt locale iar <<- şi ->> sunt globale (vezi observaţia *f*) de la secţiunea "definirea funcţiilor proprii"). Atribuirea prin = este echivalentă cu atribuirea -> (există unele situaţii când nu sunt echivalente dar nu le utilizăm)

5.2. Instrucţiunea de decizie

Instrucţiunea de decizie este IF. Forma ei este la fel ca în alte limbaje:

```
if (conditie) instr_1 else instr_2
```

i. `conditie` este o expresie care dă o valoare logică (TRUE, FALSE)
ii. `instr_1` este o instrucţiune sau un grup de instrucţiuni şi la fel `instr_2`
iii. `else` nu poate să apară singur pe o linie. Este de preferat să fie pus după instrucţiunea (sau grupul) `instr_1`.

Exemplul 5.1.

```
a=5; b=3
if((a==5)&(b!=a))
{s<-a+b^2
s<-s^2} else
{s=a-b^2; s<-s^2}
s
[1] 196
```

Există şi o funcţie `ifelse` de forma:

```
ifelse (conditie, da, nu)
```

Dacă este adevărată afirmaţia *condiţie* atunci rezultatul lui *ifelse* este *da* altfel rezultatul este *nu*. În *conditie* putem avea un vector care se evaluează pe fiecare componentă la TRUE sau FALSE. Rezultatul funcţiei este un vector de aceeaşi lungime ca vectorul *condiţie*, cu valori din vectorii da sau nu. Dacă vectorii *da* sau *nu* nu au aceeaşi lungime ca şi vectorul *condiţie* atunci sunt trunchiaţi sau extinşi prin repetare.

Exemplul 5.2.

```
> a=c(1,-2,3,0,-4,-5,0,2)
> da=c(1,2,3)
> nu=c(-1,-2,-3,-4,-5,-6,-7,-8,-9,-10)
> ifelse(a>0,da,nu)
[1]  1 -2  3 -4 -5 -6 -7  2
```

3. Instrucţiuni de repetiţie

Instrucţiunile de repetiţie (ciclare) sunt în R: *for, while, repeat*. Aceste instrucţiuni au formele:

for *(var1 in var2) instr*

unde *var2* este o variabilă ce se poate converti la un şir de valori (ca de exemplu vector sau listă), iar *instr* este o instrucţiune sau un grup de instrucţiuni.

while *(cond) instr*

unde *cond* este o expresie ce se evaluează la TRUE sau FALSE, iar *instr* este o instrucţiune sau un grup de instrucţiuni. Se va avea desigur grijă ca variabila *cond* să se modifice în grupul de instrucţiuni *instr* pentru a deveni la un moment FALSE şi a se opri repetiţia.

repeat *instr*

este o variantă de repetiţie în care *instr* este un bloc de instrucţiuni. Pentru a se asigura că se termină la un moment aceste repetiţii trebuie întreţinută în blocul *instr* o condiţie care se evaluează la TRUE sau FALSE. Ieşirea din bloc se face la execuţia unei instrucţiuni de forma: if(conditie) break, dacă conditie=TRUE.

In blocurile de repetiţie se poate folosi comanda ***break*** ce determină ieşirea imediată din ciclu sau comanda ***next*** care determină trecerea la următoarea repetiţie din ciclu. În cazul ciclurilor incluse unul în altul *break* şi *next* se utilizeaza doar pentru ciclul cel mai interior.

Exemplul 5.3. Se dă o matrice *a* de dimensiune 3x4 şi se cere să se calculeze vectorul *s* de dimensiune egală 3 pentru care *s*[i] este suma elementelor din linia *i*.
Rezolvare prin repetiţie cu *for*:

```
a=matrix(c(1:12),nrow=3)
s=rep(0,3)
for (i in 1:3){
for (j in 1:4) s[i]<-s[i]+a[i,j]
}

> s
[1] 22 26 30
```

Rezolvare prin repetiţie cu *while*

```
a=matrix(c(1:12),nrow=3)
s=rep(0,3)
i=1
while(i<=3){
j=1
while (j<=4) {s[i]=s[i]+a[i,j];j=j+1}
i=i+1
}

> s
[1] 22 26 30
```

Rezolvare prin repetiţie cu *repeat*

```
a=matrix(c(1:12),nrow=3)
s=rep(0,3)
i=1
repeat{
 j=1
 repeat {s[i]=s[i]+a[i,j];j=j+1; if(j==5) break}
i=i+1
if(i==4) break
}

> s
[1] 22 26 30
```

În toate aceste programe ciclul interior se poate înlocui cu funcţia *sum*. Prin aceasta se execută mai repede programul pentru că repetiţia din funcţia *sum* este compilată direct în cod maşină. Programul ar arăta astfel (în varianta cu *for*).

```
a=matrix(c(1:12),nrow=3)
for (i in 1:3)s[i]=sum(a[i,])

> s
[1] 22 26 30
```

Multe repetiţii pot fi realizate cu funcţiile de mapare *apply, lapply, sapply, tapply, mapply*, descrise în capitolul 3. Programul precedent poate fi scris de exemplu cu funcţia *apply* astfel:

```
a=matrix(c(1:12),nrow=3)
s=apply(a,MARGIN=1,sum)

> s
[1] 22 26 30
```

Exemplul 5.4. Date matricele *a*, *b*, *c* de dimensiune 3x4 să se determine matricea *d* de dimensiune 3x4 pentru care *d*[*i*, *j*] este media a vectorului format de *a*[*i*, *j*], *b*[*i*, *j*], c[*i*, *j*].
Repetiţia cu *for* ne dă două cicluri:

```
a=matrix(1:12,nrow=3);
b=matrix(seq(1,2,length=12),nrow=3)
c=matrix(11:22, nrow=3)

d=matrix(rep(0,12),nrow=3)
for (i in 1:3)
for (j in 1:4)
d[i,j]=mean(a[i,j],b[i,j],c[i,j])

> d
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Repetiţia prin funcţia *mapply* se realizează astfel:

```
a=matrix(1:12,nrow=3);
b=matrix(seq(1,2,length=12),nrow=3)
c=matrix(11:22, nrow=3)

d=mapply(mean,a,b,c)
d=matrix(d,nrow=3)

> d
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

5.4. Definirea funcţiilor proprii

În limbajul R definirea funcţiilor se face sub forma

nume_functie<-function(argumente) {instructiuni de calcul}

iar apelul funcţiei se face simplu *nume_functie(argumente)*. Rezultatul întors de funcţie trebuie să apară ultimul, fără atribuire. O altă posibilitate este instrucţiunea *return(val)*. În acest caz, execuţia instrucţiunilor din funcţie se încheie imediat şi *val* este returnată ca valoare a funcţiei.

Exemplul 5.5. Să definim funcţia $f(x,y,z)=x^2+xyz$ şi apoi să o aplicăm argumentelor x=3, y=5, z=9.

```
f<-function(x,y,z){r<-x^2+x*y*z;r}
f(3,5,9)

[1] 144
```

În continuare am strâns câteva observaţii referitoare la funcţii

a) Deoarece în definiţia funcţiei am folosit numele *x*, *y*, *z*, dacă la apelul funcţiei vom utiliza argumentele cu numele lor, atunci ordinea în care apar poate fi schimbată.

```
> f<-function(x,y,z){r<-x^2+x*y*z;r}
> f(3,5,9)
[1] 144
> f(y=5,z=9,x=3)
[1] 144
> f(3, z=9,y=5)
[1] 144
> f(3,y=5,9)
[1] 144
> f(3,9,y=5)
[1] 144
```

b) Dacă unele argumente sunt prezente cu numele lor şi altele fără nume, atunci cele fără nume trebuie să apară în ordinea în care apar în definiţia funcţiei.

c) Este posibil ca în definiţia unei funcţii unele argumente să aibă valori implicite prestabilite. În acest caz la utilizarea funcţiei nu este neapărat necesar ca ele să fie specificate decât dacă se doreşte utilizarea lor cu alte valori decât cele implicite. Mai jos avem un exemplu:

```
> f<-function(x,y=5,z){r<-x^2+x*y*z;r}
> f(3,5,9)
[1] 144
> f(x=3, z=9)
[1] 144
```

Un apel *f*(3,9) pentru funcţia precedentă generează un mesaj de eroare pentru că se face identificarea *x*<-3, *y*<-9 şi nu este definită valoarea pentru *z*.

```
> f(3,9)
Error in x * y * z : 'z' is missing
```

d) Putem utiliza în definiţia unei funcţii şi alte funcţii definite anterior. De asemenea putem utiliza în definiţia unei funcţii argumentul "..." (trei puncte) care reprezintă argumente opţionale. De obicei aceste argumente sunt transferate funcţiei interioare.

```
> g<-function(a=4,b=6) a+b/2
>
> f<-function(x,y=5,z,...){r1<-x^2+x*y*z; r2<-g(...)
+ r<-r1+r2; r}
>
> f(1,2,3)
[1] 14
> f(1,2,3,4); # a=4 ; b=6 implicit
[1] 14
> f(z=3,x=1,y=2,4,6); #a=4, b=6
[1] 14
> f(z=3,x=1,y=2,4,8); #a=4, b=8
[1] 15
> f(z=3,x=1,y=2,b=8,a=4); #a=4, b=8
[1] 15
> f(z=3,x=1,y=2,b=8); #a=4 (implicit), b=8
[1] 15
```

e) În anumite cazuri avem nevoie de o funcţie doar într-un loc. În acest caz funcţia se poate defini anonim (fără nume) şi în consecinţă poate fi utilizată doar în acel loc. De exemplu să calculăm pentru trei matrice *a*, *b*, *c* de aceleaşi dimensiuni expresia *a*^2+*a***b***c* (nu uităm că operaţiile sunt vectorizate, se aplică elementelor în poziţii similare în matricele *a*, *b*, *c*). Scriptul următor realizează acest lucru în două feluri, în unul din feluri se utilizeză o funcţie anonimă.

```
a=matrix(1:12, nrow=3)
b=matrix(seq(from=1,by=2, length=12), nrow=3)
```

```
c=matrix(rep(2,12), nrow=3)
rez1=a^2+a*b*c
rez2=matrix(mapply(function(x,y,z)x^2+x*y*z,a,b,c),nrow=3)
rez1
rez2
```

Rezultatele sunt aceleaşi pentru cele două metode:

```
> rez1
     [,1] [,2] [,3] [,4]
[1,]    3   72  231  480
[2,]   16  115  304  583
[3,]   39  168  387  696
> rez2
     [,1] [,2] [,3] [,4]
[1,]    3   72  231  480
[2,]   16  115  304  583
[3,]   39  168  387  696
```

f) Variabilele definite în corpul unei funcţii, să zicem *f*, au caracter local. Ele sunt vizibile doar în corpul funcţiei *f*. De asemenea în corpul funcţiei *f* se pot utiliza date sau funcţii definite în funcţia în care este definită *f*, sau date ori funcţii globale. Atunci când se caută valoarea unei variabile (sau definiţia unei funcţii) se caută printre parametrii locali (inclusiv argumentele funcţiei), apoi printre parametrii locali funcţiei superioare (în interiorul căreia este definită) s.a.m.d. Modificările parametrilor locali nu modifică parametrii globali cu acelaşi nume. Pentru a modifica în corpul unei funcţii un parametru global, să zicem A, trebuie să utilizăm instrucţiunea de atribuire `A<<-expresie`. La crearea unei funcţii se crează un context de vizibilitate (environment) cu ceea ce este vizibil de către funcţia respectivă.

Comentăm acum (şi modificăm) un exemplu din manual: *An introduction to R* , pg. 47 (manualul vine cu distribuţia standard de R).

Apelul funcţiei open.account produce o listă ce are ca elemente trei funcţii (deposit, withdraw, balance). Crearea unui cont se face printr-o instrucţiune de tipul `client=open.account(sumadepusa)`. Variabila `data` este produsă de program. Variabila `dob` reprezintă rata anuală a dobînzii (compuse) şi poate fi modificată de utilizatorul scriptului: este aceeaşi pentru orice client

`open.account <-`
`function(total=0,data,dob=0.10){`
 	
`data=Sys.time()`
 	
`list(`
 	
`deposit = function(amount) {`
	La crearea unui nou cont se apelează o singură dată funcţia sub forma `client=open.account(…)`
	Sistemul generează un context de vizibilitate pentru rezultatul funcţiei (în cazul nostru lista `client`). Funcţiile din lista `client` au în contextul lor de vizibilitate parametrii de intrare ai funcţiei generatoare `open.account` . În acest context vor fi văzute variabilele `total, dob,data;` `data` este data creării contului
	
	Prima funcţie: `deposit` modifică

`if(amount <= 0)`
`stop("Deposits must be positive!\n")`
`dataa=Sys.time()`
`durata=as.numeric(dataa) - as.numeric(data)`
`durata=durata/86400`
`durata=durata/365`
`total <<- total*(1+dob)^durata + amount`
`cat(amount, "deposited. Your balance is", total, "\n\n")`
`data<<-dataa`
`},`
 	
`withdraw <- function(amount) {`
`dataa=Sys.time()`
`durata<-as.numeric(dataa) - as.numeric(data)`
`durata<-durata/86400`
`durata<-durata/365`
`total1<-total*(1+dob)^durata`
`if(amount > total1)`
`stop("You don't have that much money!\n")`
`total <<- total1 - amount`
`data<<-dataa`
`cat(amount, "withdrawn. Your balance is", total, "\n\n")`
`},`
 	
`balance <- function() {`
`dataa=Sys.time()`
`durata=as.numeric(dataa) - as.numeric(data)`
`durata=durata/86400`
`durata=durata/365`
`total1=total*(1+dob)^durata`
`cat("Your balance is", total1, "last change ",as.character(data),"\n\n")`
`}`
 	
`)`
`}`
	valoarea contului ţinînd seama de dobîndă şi valoarea depusă
	`dataa` este data alimentării contului
	Se convertesc datele în secunde, apoi în zile, apoi în ani pentru a se calcula durata de la ultima operaţie în cont
	Se actualizează suma după formula total = total$(1+dob)^{durata}$ + amount
	Se afişează informaţiile pentru client
	Actualizează data
	
	A doua funcţie: `withdraw` modifică valoarea contului ţinînd seama de dobîndă şi valoarea retrasă
	
	Se actualizează totalul după formula Total1 = total$(1+dob)^{durata}$
	Dacă total1 este prea mic, se dă mesaj de eroare
	Dacă nu, se modifică valoarea contului
	Actualizează data
	Mesaj de confirmare
	
	A treia funcţie: `balance` nu modifică nimic în cont, nici data, doar informează clientul asupra sumei disponibile şi asupra datei ultimei operaţii în cont.

După ce se rulează scriptul ce conţine funcţia open.account se poate lucra în flul următor cu conturile:

```
> gh<-open.account(10000000)
> gh$deposit(1000000)
1e+06 deposited. Your balance is 1.1e+07

> gh$withdraw(100000)
1e+05 withdrawn. Your balance is 10900001

> gh$balance()
Your balance is 10900002 last change  2011-12-19 01:23:47
```

Se vede cum cu trecerea timpului se adaugă dobânda.

g) Printre parametrii unei funcţii pot fi alte funcţii. De asemenea în corpul unei funcţii pot fi apeluri la ea însăşi ceea ce duce la aplicarea recursivă a funcţiei. În acest caz e nevoie de condiţii care să asigure că numărul de aplicări recusive e finit. Ca exemplu avem următorul script simplu ce calculează recursiv termenii şirului Fibonacci.

```
fib<-function(n){
if(n==1) return(c(1))
if(n==2)return(c(1,1))
s1=fib(n-1)
urm=s1[n-1]+s1[n-2]
return (c(s1,urm))
}
> fib(10)
 [1]  1  1  2  3  5  8 13 21 34 55
```

5.5. Apelul din R al funcţiilor scrise în C

Apar situaţii când nu avem în R rutine pentru anumite operaţii. Soluţia este scrierea de rutine (funcţii) proprii. Utilizarea într-o măsură mare a instrucţiunilor de decizie sau repetiţie în R face ca timpul de rulare să crească mult datorită faptului că R este un limbaj interpretat. O ieşire o reprezintă uneori utilizarea de rutine scrise într-un limbaj în care se face compilarea la nivel de cod maşină, cum ar fi C, Fortran, Pascal, etc. În R există mecanisme de apel a funcţiilor scrise în C ori Fortran precum şi mecanisme de apel într-un program extern a unor rutine din R. Ne vom ocupa de metoda de apel în R a unor rutine scrise în C.

Atunci când se apelează într-un program funcţii scrise în alt program trebuie avute în vedere corespondenţa tipurilor de date, modul de transfer al datelor spre rutina apelată, modul de preluare a rezultatelor produse de rutina apelată. În documentaţia standard pentru R găsim următoarea corespondenţă între tipurile de date din R, C, Fortran :

R storage mode	C type	FORTRAN type
logical	int *	INTEGER
integer	int *	INTEGER
double	double *	DOUBLE PRECISION
complex	Rcomplex *	DOUBLE COMPLEX
character	char **	CHARACTER*255
raw	unsigned char *	none

Pentru cei care doresc să scrie pachete de programe pentru R utilizând limbajul C, în subdirectorul « include » din directorul unde este instalat R se găsesc fişiere de definiţie pentru diverse tipuri de date şi diverse funcţii din librării larg utilizate. Pentru noi corespondenţa de tipuri din tabelul de mai sus este suficientă.

Câteva observaţii sunt de avut în vedere:

a) Datele între R şi rutine externe se schimbă doar prin adrese.

b) Rutinele externe nu trebuie să întoarcă vreo valoare (în C ele trebuie să fie funcţii de tip void iar în Fortran trebuie să fie subprograme de tip subroutine)

c) Schimbul de date (parametri de intrare şi rezultate) se face doar prin argumentele funcţiilor (subrutinelor în Fortran)

d) Rutinele externe (în C ori Fortran) se pun în biblioteci dinamice (.dll în windows, .so în linux) care se încarcă în R cu comanda *dyn.load(nume_librarie, ...)* şi se deconectează de la R prin comanda *dyn.unload(nume_librarie)*.

e) În principiu se pot utiliza în R şi rutine scrise în Pascal dacă acestea sunt declarate de tip C. Declararea în Pascal a unor rutine ca fiind de tip C este importantă pentru a se face corect corespondenţa între parametri (în Pascal transmiterea parametrilor către o subrutină se face în ordine inversă faţă de C). De asemenea trebuie avut grijă la corespondenţa tipurilor de date (de exemplu double din R corespunde cu ^double în Pascal)

f) Apelul funcţiilor C se poate face cu comanda

.C(NAME, ..., NAOK = FALSE, DUP = TRUE, PACKAGE, ENCODING)

unde NAME este numele funcţiei, dacă NAOK este TRUE atunci valorile NA, NaN, Inf sunt transmise funcţiei C, altfel se generează o eroare dacă apar, DUP=TRUE determină o duplicare a datelor înainte de a fi transmise subrutinei, PACKAGE specifică în ce pachete să se caute subrutina NAME. Este obligatorie doar prezenţa parametrului NAME în apelul .C(NAME,….)

g) Pentru comoditate se poate introduce comanda .C(NAME…) într-o funcţie R care face apelul mult mai comod şi se pot prelua din parametrii întorşi doar pe cei care interesează (vezi exemplul).

În cele ce urmează ne mărginim la utilizarea unor rutine C care au ca parametri doar numere, reale sau întregi. Pentru aplicaţii matematice pare cea mai frecventă situaţie (ar mai fi eventual şi utilizarea de numere complexe ca perechi de numere reale).

Exemplul 5.6. Sa se scrie o funcţie C numită „rot” care plecând de la o matrice pătratică „a” să producă un vector care conţine elementele matricei în ordinea din figura următoare, apoi să se apeleze această funcţie din R.

$$\begin{pmatrix} 1 & 12 & 11 & 10 \\ 2 & 13 & 16 & 9 \\ 3 & 14 & 15 & 8 \\ 4 & 5 & 6 & 7 \end{pmatrix}$$

Ordinea de extragere a elementelor din matrice

Programul de extragere îl scriem în C. Am utilizat mediul de programare C++ gratuit Code:Blocks 8.02 (pentru windows) în care am ales un proiect nou de tip .dll numit pr1, care va avea ca efect crearea librăriei dinamice pr1.dll. In cadrul acestui proiect au fost create automat două fişiere main.h şi main.dll de mai jos în care partea pe care am adăugat-o este evidenţiată (se recunoaşte rutina de extragere a elementelor din matricea a în vectorul b în ordinea specificată, precum şi declaraţia că această rutină să aparţină librăriei numită aici pr1.dll)

Fişierul main.h

```
#ifndef __MAIN_H__
#define __MAIN_H__

#include <windows.h>

/*   To use this exported function of dll, include this
header
 *  in your project.  */

#ifdef BUILD_DLL
    #define DLL_EXPORT __declspec(dllexport)
#else
    #define DLL_EXPORT __declspec(dllimport)
#endif

#ifdef __cplusplus
extern "C"
{
#endif

void DLL_EXPORT SomeFunction(const LPCSTR sometext);
void DLL_EXPORT rot(int *nlin, double *a,double *b);

#ifdef __cplusplus
}
#endif

#endif // __MAIN_H__
```

Fişierul main.cpp

```
#include "main.h"

// a sample exported function
void DLL_EXPORT SomeFunction(const LPCSTR sometext)
{
    MessageBoxA(0,   sometext,   "DLL   Message",   MB_OK   |
MB_ICONINFORMATION);
}


void DLL_EXPORT rot(int *nlin, double *a,double *b){
 int i,j,k,cnt=0;
 int n=*nlin,mi=n/2;

for (k=0;k<mi;k++){
  for(i=k;i<n-k-1;i++){b[cnt]=a[i+n*k];cnt++;}
  for(j=k;j<n-k-1;j++) {b[cnt]=a[n-k-1+n*j];cnt++;}
  for(i=n-k-1;i>k;i--){b[cnt]=a[i+n*(n-k-1)];cnt++;}
  for(j=n-k-1;j>k;j--) {b[cnt]=a[k+n*j];cnt++;}
}
if(n%2==1) b[cnt]=a[mi+n*mi];
}
BOOL  WINAPI  DllMain(HINSTANCE  hinstDLL,  DWORD  fdwReason,
LPVOID lpvReserved)
```

```
{
    switch (fdwReason)
    {
        case DLL_PROCESS_ATTACH:
            // attach to process
            // return FALSE to fail DLL load
            break;

        case DLL_PROCESS_DETACH:
            // detach from process
            break;

        case DLL_THREAD_ATTACH:
            // attach to thread
            break;

        case DLL_THREAD_DETACH:
            // detach from thread
            break;
    }
    return TRUE; // succesful
}
```

După compilare şi linkeditare (am optat pentru compilatorul GCC) se obţine librăria pr1.dll pe care am mutat-o în directorul curent din R.

Programul R în care am utilizat funcţia *rot* este următorul:

```
dyn.load("pr1.dll")
x=1:25
a=matrix(x,ncol=5)

b=rep(0,length(a))
rez<-
.C("rot",as.integer(ncol(a)),as.double(a),as.double(b))

dyn.unload("pr1.dll")
```

Rezultatele le vedem mai jos. Parametrii întorşi sunt preluaţi de R sub forma unei liste în ordinea în care au fost declaraţi. Avem trei parametri în rutina *rot* deci şi rezultatul întors *rez* va fi o listă cu trei intrări. Ultima intrare (a treia rez[[3]]) este vectorul b pe care îl căutăm.

```
> a
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
> rez[[3]]
 [1]  1  2  3  4  5 10 15 20 25 24 23 22 21 16 11  6  7  8
9 14 19 18 17 12 13
```

Pentru apelul funcţiei *rot* care se găseşte în lbrăria *pr1.dll* putem scrie o funcţie R care să facă acest apel mai uşor. In fişierul următor se face acest lucru. Din cei trei parametri îl luăm ca parametru de ieşire din funcţia *rot* doar pe al treilea, b.

```
dyn.load("pr1.dll")
x=1:25
a=matrix(x,ncol=5)

rot<-function(a){
  if(!is.matrix(a))stop("intrarea in rot trebuie sa
fie matrice")
  if(!is.numeric(a)) stop("matricea trebuie sa fie
numerica")
  if(!(ncol(a)==nrow(a))) stop("matricea trebuie sa
fie patratica")
  b=rep(0,length(a));
  rez<-
C("rot",as.integer(ncol(a)),as.double(a),as.double(b)
)
  return(rez[[3]])
}
aa<-rot(a)
dyn.unload("pr1.dll")
```

Rezultatul se vede mai jos:

```
> a
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
> aa
 [1]  1  2  3  4  5 10 15 20 25 24 23 22 21 16 11  6  7  8
9 14 19 18 17 12 13
>
```

Apelul rutinelor scrise în Fortran este asemanator. Ele sunt puse în librii dinamice de unde se încarcă cu comanda dyn.load(nume_librarie). Apelul unei funcţii din librărie se face prin instrucţiunea:

```
.Fortran(NAME, ..., NAOK = FALSE, DUP = TRUE, PACKAGE,
                        ENCODING)
```

unde parametrii au aceeaşi semnificaţie ca la apelul funcţiilor C.

Capitolul 6

Aplicaţii diverse programate în R

Poate cel mai uşor mod de a învăţa un limbaj este de a urmări cîteva programe simple scrise în el.

6.1. Rambursări de credite.

Un client face un credit în valoare de S_0 UM la o bancă. În urma negocierilor, obţine o rată a dobînzii egală cu i. Doreşte să îşi facă un plan de rambursare în care să plătească ratele la momentele de timp $t_1,...,t_n$; ar mai dori să ştie şi ce suma mai are de plătit după fiecare rată.

Să presupunem că ratele plătite la momentele t_i au valoarea r_i. Partea efectivă din datorie plătită atunci va fi egală cu $a_i = r_i - D_i$ unde D_i este dobînda plătită la suma restantă pe intervalul de timp (t_{i-1},t_i). Fie S_i suma restantă la momentul t_i, după achitarea ratei r_i.

Axioma este că dacă dobînda unitară pe intervalul (t_{i-1},t_i) este d_i, atunci dobînda plătită la momentul t_i va fi egală cu $D_i = S_{i-1}d_i$. Să notăm $q_i = 1 + d_i$.

Cum prima dobîndă se aplică la suma totală S_0 obţinem următoarea recurenţă pentru sumele restante

$$a_1 = S_0 - S_1 = r_1 - d_1S_0 \Leftrightarrow S_1 = (1 + d_1)S_0 - r_1 \Leftrightarrow S_1 = q_1S_0 - r_1$$
$$a_2 = S_1 - S_2 = r_2 - d_2S_1 \Leftrightarrow S_2 = (1 + d_2)S_2 - r_2 \Leftrightarrow S_2 = q_2S_1 - r_2$$
$$.....$$
$$S_k = (1 + d_k)S_{k-1} - r_k \Leftrightarrow S_k = q_kS_{k-1} - r_k \qquad (6.1.1)$$

Putem scrie relaţia (6.1.1) şi sub forma $S_k + r_k = q_kS_{k-1}$. Să notăm $\alpha_k = 1/q_k$. Atunci relaţia (6.1.1) devine

$$S_{k-1} = \alpha_kS_k + \alpha_kr_k \qquad (6.1.2)$$

Se propun mai multe scenarii.

6.1.1. Rată constantă, dobîndă simplă

Dobînda fiind simplă, dobînda unitară pe intervalul (s,t) este $d(s,t) = i(t\text{-}s)$, unde i este rata anuală a dobînzii. Acum $q_j = 1 + d_j = 1 + i(t_j - t_{j-1})$ pentru $j \geq 2$, $q_1 = 1 + it_1$. Scenariul presupune rate egale, adică $r_1 = ... = r_n = r$.

Rata se calculează punînd condiţia ca $S_n = 0$.

Relaţia (6.1.2) implică, din aproape în aproape $S_{n-1}=r\alpha_n \Rightarrow S_{n-2}=r(\alpha_{n-1}+\alpha_{n-1}\alpha_n)$ $\Rightarrow S_{n-3} = r(\alpha_{n-2} + \alpha_{n-2}\alpha_{n-1} + \alpha_{n-2}\alpha_{n-1}\alpha_n) \Rightarrow$ ..... de unde găsim rata

$$r = \frac{S_0}{\alpha_1 + \alpha_1\alpha_2 + ... + \alpha_1\alpha_2...\alpha_n} \qquad (6.1.3)$$

Scriptul următor calculează rata r după formula (6.1.3) şi amortismentele $(a_k)_{1\leq k\leq n}$. Atenţie: comentariile scrise cu font Times New Roman nu fac parte din script, ci sunt comentarii.

Se apelează prin instrucţiunea

```
a = ratadosi(s0,i,t)
```

Variabilele funcţiei sunt suma împrumutată s_0, rata anuală a dobînzii, i şi vectorul t cu momentele cînd se fac plăţile.

Rezultatul se dă în forma unei liste. Aşadar rata este `a[[1]]` iar amortismentele sunt `a[[2]]`.

```
#functia ratadosi calculeaza rata constanta daca se da
dobinda
#simpla cu rata i, suma de platit
#si daca se da t=(t1,...,tn) momentele de plata a ratelor
ratadosi<-function(s0,i,t)
{n=length(t);q=t;alfa=t;bet=t;s=t;amo=t # iniţializări
q[1]=1+i*t[1];for (k in 2:n) {q[k]=1+i*(t[k]-t[k-1])}
## q1 = 1 + it1, k ≥ 2 ⇒ qk = 1 + i(tk – tk-1)
alfa=1/q;for (k in 1:n) {bet[k]=prod(alfa[1:k])}
##αk = 1/qk, βk=α1...αk
r=s0/sum(bet)              ## r = s0/(β1 + β2 +...+ βn)
s[1]=s0*q[1]-r;for (k in 2:n){s[k]=s[k-1]*q[k]-r} ## Relaţia (1)
amo[1]=s0-s[1];for (k in 2:n){amo[k]=s[k-1]-s[k]}## ak=Sk-1 - Sk
rata=list(r,amo)                ## r = rata[[1]], amortismente = rata[[2]]
rata}
```

Exemplu de aplicare

Se face un credit de 1200 de lei, plătibil în 12 rate lunare egale, în regim de dobîndă simplă cu termen de graţie de 1 an şi dobîndă 12%. Care este rata şi cum evoluează amortismentele?

Soluţie. Avem s0 = 1200, i = 0.12 şi t = $(k/12)_{13\le k\ \le\ 24}$. Deci succesiunea de instrucţiuni este

```
>t=13:24/12
> a=ratadosi(1200,.12,t)
> rata=a[[1]];rata
[1] 119.2861
> amortismente=a[[2]];amortismente
 [1] -36.7139 106.9190 107.9881 109.0680 110.1587 111.2603
112.3729 113.4966
 [9] 114.6316 115.7779 116.9357 118.1050
```

De remarcat că primul amortisment este negativ (s-a acumulat dobînda!) , celelalte cresc în progresie geometrică.

6.1.2. Rată constantă, dobîndă compusă

Singura deosebire este că acum $q_1 = (1+i)^{t_1}, k>1 \Rightarrow q_k = (1+i)^{t_k - t_{k-1}}$,

```
#functia ratadoco calculeaza rata constanta daca se da
#dobinda compusa cu rata i, suma de platit
#si daca se da t=(t1,...,tn) momentele de plata a ratelor
ratadoco<-function(s0,i,t)
{n=length(t);q=t;alfa=t;bet=t;s=t;amo=t
```

```
q[1]=(1+i)^t[1];for (k in 2:n) {q[k]=(1+i)^(t[k]-t[k-1])}
## q_k = (1+i)^{t_k - t_{k-1}},
alfa=1/q;for (k in 1:n) {bet[k]=prod(alfa[1:k])}
r=s0/sum(bet)
s[1]=s0*q[1]-r;for (k in 2:n){s[k]=s[k-1]*q[k]-r}
amo[1]=s0-s[1];for (k in 2:n){amo[k]=s[k-1]-s[k]}
rata=list(r,amo)
rata}
```

Exemplu de aplicare

Se face un credit de 1200 de lei, plătibil în 12 rate lunare egale,în regim de dobîndă compusă cu termen de graţie de 1 an şi dobîndă 12%. Care este rata şi cum evoluează amortismentele?

Soluţie.

```
>  a=ratadoco(1200,.12,t);a
[[1]]
[1] 119.0274

[[2]]
 [1] -37.72552 107.28290 108.30088 109.32853 110.36592
111.41316 112.47034
 [8] 113.53755 114.61488 115.70244 116.80031 117.90861
```

Rata este putin mai mică, deoarece dacă s-ar calcula corect rata lunară a dobînzii, ea nu ar fi egală cu $i/12$, ci cu $(1+i)^{\frac{1}{12}} - 1$. Pentru $i = 0{,}12$ ar rezulta că rata lunară ar trebui să fie 0.95% şi nu 1%, aşa cum se ia în mod real. [8]

6.1.3. Amortismente constante, dobîndă simplă

Relaţia folosită este $r_k = a + d_k S_{k-1}$ cu amortismentul $a = S_0/n$. Dobînzile unitare sunt $d_1 = it_1$ şi $k \geq 2 \Rightarrow d_k = i(t_k - t_{k-1})$

```
#functia amordosi calculeaza ratele in ipoteza
#amortismentelor constante
# daca se da dobinda simpla cu rata i, suma de platit s0
#si daca se da t=(t1,...,tn) momentele de plata a ratelor
amordosi<-function(s0,i,t)
{n=length(t);d=t;r=t;amo=s0/n
d[1]=i*t[1];for (k in 2:n) {d[k]=i*(t[k]-t[k-1])}
for (k in 1:n) {r[k]=amo+d[k]*s0*(1 - (k-1)/n)}
# d_k = i(t_k - t_{k-1})
r}
```

Exemplu de aplicare

Se face un credit de 1200 de lei, plătibil în 12 rate lunare, în regim de dobîndă simplă cu termen de graţie de 1 an şi dobîndă 12%. Care sunt ratele dacă se planifică 12 amortismente egale a cîte 1200 lei?

[8] Ca o curiozitate, rata zilnică a dobînzii se calculează de unele bănci după formula $i/360$. Ca şi cum anul ar avea 360 de zile.

Soluţie.

```
t=13:24/12
> a=amordosi(1200,.12,t);a
 [1] 256 111 110 109 108 107 106 105 104 103 102 101
```

Cu excepţia primeia, ratele formează o progresie artitmetică descrescătore.

6.1.4. Amortismente constante, dobîndă compusă

Singura diferenţă faţă de scenariul anterior este că dobînzile unitare sunt acum $d_1 = (1+i)^{t_1} - 1$ şi $k \geq 2 \Rightarrow (1+i)^{t_k - t_{k-1}} - 1$.

```
#functia amordoco calculeaza ratele in ipoteza
#amortismentelor constante
# daca se da dobinda compusa de rata i, suma de platit s0
#si daca se da t=(t1,...,tn) momentele de plata a ratelor
amordoco<-function(s0,i,t)
{n=length(t);d=t;r=t;amo=s0/n
d[1]=(1+i)^t[1]-1;for (k in 2:n) {d[k]=(1+i)^(t[k]-t[k-
1])-1}
for (k in 1:n) {r[k]=amo+d[k]*s0*(1 - (k-1)/n)}
r}
```

Exemplu de aplicare

Se face un credit de 1200 de lei, plătibil în 12 rate lunare, în regim de dobîndă compusă cu termen de graţie de 1 an şi dobîndă 12%. Care sunt ratele dacă se planifică 12 amortismente egale a cîte 1200 lei?

Soluţie.

```
> a=amordoco(1200,.12,t);a
 [1] 256.7529 110.4377 109.4888 108.5399 107.5910 106.6422
105.6933 104.7444
 [9] 103.7955 102.8466 101.8978 100.9489
```

6.1.5. Compararea celor patru scenarii

Dacă dorim să comparăm cele patru scenarii din punctul de vedere al sumei totale pe care o avem de plată, putem apela funcţia comprate.

```
#functia comprate compara cele patru scenarii prin suma
totala
comprate<-function(s0,i,t)
{n=length(t)
a1=ratadosi(s0,i,t);a1=a1[[1]]
a2=ratadoco(s0,i,t);a2=a2[[1]]
a3=amordosi(s0,i,t)
a4=amordoco(s0,i,t)
S1=n*a1;S2=n*a2;S3=sum(a3);S4=sum(a4)
scenarii<-c("rate egale ds","rate egale dc","amo egale
ds","amo egale dc")
sume<-c(S1,S2,S3,S4)
```

```
rez=data.frame(scenarii,sume)
rez}
```

Exemplu de aplicare

În cazul nostru suma cea mai mică este, după cum rezultă din tabelul de mai jos, de 1419.38 lei, în regim de dobîndă compusă, dacă s-ar accepta calculul normal al dobînzii.

```
t=13:24/12; comprate(1200,0.12,t)
                  scenarii     sume
1 rate egale dob simpla  1431.433
2 rate egale dob compusa 1428.329
3 amo egale dob simpla   1422.000
4 amo egale dob compusa  1419.379
```

6.1.6. Rată fixă de valoare determinată

Revenim la relaţia de recurenţă a sumelor restante, $S_k = q_k S_{k-1} - r_k$ şi punem altă întrebare: dacă dorim să plătim o rată fixă, r, cîte rate avem de plătit? Ultima rată va fi eventual mai mică. Deci problema este de a calcula numărul $N = \min\{n \mid S_n \le 0\}$. De exemplu, în scenariu de dobîndă compusă, i = 12%, cu termen de gratie de 1 an, cîte rate de 100 de lei ne trebuie pentru a achita creditul de 1200 lei?

Presupunem că plăţile se fac la momentele $t_0 + nh$. Este posibil ca rata să fie prea mică pentru a putea achita rambursa creditul.

Dacă $q_1 = (1+i)^{t_1} - 1$ şi $k \ge 2 \Rightarrow q_k = (1+i)^{t_k - t_{k-1}} - 1 = (1+i)^h - 1 = q$, avem

$S_1 = S_0 q_1 - r$, $S_2 = S_1 q - r = (S_0 q_1 - r)q - r = S_0 q_1 q - r(1 + q)$, $S_3 = S_0 q_1 q^2 - r(1 + q + q^2)$,....şi, în general, $S_{n+1} = S_0 q_1 q^n - r(1 + q + ... + q^n)$.

Atunci $S_{n+1} \le 0 \Leftrightarrow S_0 q_1 q^n \le r(1 + q + ... + q^n) \Leftrightarrow r(1 + \alpha + ... + \alpha^n) \ge S_0 q_1$, cu $\alpha = 1/q$.

Creditul se poate rambursa într-un număr finit de rate egale cu r dacă există un n *ca* $S_{n+1} \le 0$. Trebuie deci ca $r > \dfrac{S_0 q_1}{1 + \alpha + \alpha^2 + ...}$, adică deducem condiţia

$$r > S_0 q_1 (1 - \alpha) \tag{6.1.4}$$

Funcţia care calculează numărul de rate şi arată şi evoluţia sumelor restante este nrate. Funcţia e calculată în ipoteza dobînzii compuse, dar cititorul o poate modifica uşor înlocuind q_1 şi q. De asemenea, am pus o limită numărului de rate (pînă la 1000), care poate fi şi ea modificată. Sumele restante sunt puse în vectorul s, iniţializat cu zerouri. Dacă rata este prea mică, funcţia prevede un mesaj în care se dă rata minimă cu care se poate acoperi creditul.

```
#Cite rate in valoare de "r"UM am de platit daca dobinda
este
#compusa, plata se incepe la t0 si se face la intervale de
h ani.
nrrate<-function(r,s0,t0,h,i)
{s=c(rep(0,1000));q1=(1+i)^t0;q=(1+i)^h;alfa=1/q
if(r<=q1*s0*(1-alfa)) {print("rata prea mica,
n=infinit");return()}
s[1]=s0*q1-r;rest=s[1];k=2
while((rest>0)&&(k<1000)){s[k]=s[k-1]*q-r;rest=s[k];k=k+1}
```

```
rez=c(k-2,s[k-2])
rez=list(rez,s[1:k-1])
rez}
```

Exemplu de aplicare
În cazul nostru avem $r = 100$, $S_0 = 1200$, $t_0 = 1$, $h = 1/12$, $i = 0.12$

```
a=nrrate(100,1200,1,1/12,0.12);a
[[1]]
[1] 14.00000 29.85075

[[2]]
 [1] 1244.00000 1155.80406 1066.77124  976.89362  886.16316
794.57178
 [7]  702.11130  608.77349  514.55002  419.43248  323.41238
226.48118
[13]  128.63021   29.85075  -69.86600
```

Interpretare: este nevoie de 14 rate, ultima incompletă (de 29.85 lei).După prima rată, suma restantă este de 1244 lei, etc; dacă am plăti 14 rate a 100 lei, ar trebui să primim înapoi 69,87 lei – acesta este sensul ultimului număr din tabelul de mai sus.

Dacă dorim să plătim în rate de 10 lei scriem

```
> a=nrrate(10,1200,1,1/12,0.12);a
```

şi obţinem reacţia

```
[1] "rata prea mica, n=infinit"
[1] 12.63307
```

Ultimul număr ne spune că rata minimă este de 12.64 lei. Dacă am încerca cu rate de 13 lei am obţine

```
a=nrrate(13,1200,1,1/12,0.12);a[[1]]
[1] 377.000000   9.707553
```

E nevoie de 33 de ani!

6.1.7. Rată fixă de valoare determinată, dobîndă compusă aleatoare

Funcţia `nralrate(r,s0,i0,i1)` face o simulare în ipoteza că rata este *r,* suma împrumutată este s0 iar rata dobînzii este aleatoare. Mai precis, aici se presupune că ratele se plătesc la intervale constante de timp $\Delta t = h$ şi pe fiecare din aceste momente de timp dobînzile unitare formează un şi de variabile aleatoare independente şi identic repartizate uniform pe intervalul (i_0, i_1). Pentru a nu risca o buclă infinită, am ales un prag (n = 1000) peste care nu mai verificăm evoluţia sumelor restante. Acum $N = \min\{n \mid S_n \le 0\}$ devine o variabilă aleatoare.

```
#presupun ca rata dobinzii este aleatoare. Atunci
#S[n]=S[n-1]qn-r
#dar q[n] sunt presupuse i.i.d. repartizate uniform pe
#intervalul i1,i2
#functia nralrate are ca argumente r (rata), s0 (suma
#imprumutata)
```

```
#i0 (rata minima a dobinzii)
#i1(rata maxima). qn = 1 + i[n] sunt presupusi i.i.d
nralrate<-function(r,s0,i0,i1)
{q1=1+(i1-i0)*runif(1);s[1]=s0*q1-r;rest=s[1];k=2
while((rest>0)&&(k<1000)){q=1+(i1-i0)*runif(1)
s[k]=s[k-1]*q-r;rest=s[k];k=k+1}
rez=c(k-2,s[k-2])
rez=list(rez,s[1:k-1])
rez
```

Exemplu de aplicare

Ca să fim consecvenţi, luăm r = 100, S_0 = 1200, i_0 = 0.05, i_1 = 0.15.

```
a=nralrate(100,1200,.05,.15);a
[[1]]
[1] 19.000000  7.846417

[[2]]
 [1] 1164.531376 1139.006774 1072.575349 1001.426395  993.777986
963.953179
 [7]  917.006879  848.128914  749.050657  714.409097  673.291059
585.682507
[13]  504.665078  424.788245  355.531910  284.929749  199.222488
106.591050
[19]    7.846417  -91.942226
```

Interpretare: trebuie 19 rate, din care 18 complete. Vectorul `a[[2]]` dă evoluţia sumelor restante.

Putem face o **analiză statistică** a numărului necesar de rate pentru rambursarea creditului cu ajutorul funcţiei nralratesim(n,r,s_0,i_0,i_1), care prezintă un vector cu *n* simulări

```
##rata aleatoare, uniform repartizata pe (i0,i1), n simulari
nralratesim<-function(n,r,s0,i0,i1)
{nr=c(rep(0,n))
for (k in 1:n) {nr[k]=nralrate(r,s0,i0,i1)[[1]][1]}
nr}
```

A se remarca rîndul al treilea: funcţia nralrate(...) returnează o listă cu două elemente: primul este un vector cu două componente (n, S_{n-1}) cu *n* numărul de rate şi S_{n-1} ultima sumă restantă iar a doua componentă este şirul sumelor restante. Daca lista este *a*, atunci *a*[[1]] este primul vector iar *a*[[1]][1] este numărul *n*.

Exemplu:

```
nr=nralratesim(10,100,1200,0.05,.15);nr
[1] 16 18 24 20 16 15 20 19 18 16
```

Dacă dorim să estimăm repartiţia variabilei aleatoare *N* , putem folosi funcţia „table”. Iată rezultatul a 10.000 de simulări:

```
> nr=nralratesim(10000,100,1200,0.05,.15)
> a=table(nr);x=as.numeric(names(a));m=length(x);p=x;for (k
in 1:m) {p[k]=a[[k]]}
> x
 [1] 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 36
```

```
> p
 [1]    4   89  479 1165 1809 2106 1721 1128  731  387  200
97   48   18   10
[16]    6    1    1
> pp=p;for (k in 1:m) {pp[k]=sum(p[1:k])/sum(p)}
> b=rbind(x,pp);b

x  13.0000 14.0000 15.0000 16.0000 17.0000 18.0000 19.0000
20.0000 21.0000
pp  0.0004  0.0093  0.0572  0.1737  0.3546  0.5652  0.7373
0.8501  0.9232

x  22.0000 23.0000 24.0000 25.0000 26.0000 27.0000 28.0000
29.0000    36
pp  0.9619  0.9819  0.9916  0.9964  0.9982  0.9992  0.9998
0.9999     1
```

De exemplu, probabilitatea ca $N \le 20$ este cam 85%

6.1.8. Probabilitatea ca *n* rate în valoare de *r* UM să fie suficiente

Revenim la relaţia de recurenţă a sumelor restante, $S_k = q_k S_{k-1} - r_k$. Presupunem, la fel ca în paragraful anterior, că ratele sunt egale între ele, $r_k = r$. Ne punem întrebarea : care este probabilitatea ca n asemenea rate să fie suficiente pentru rambursarea creditului?

Să precizăm.

Lucrurile nu sunt lămurite complet din punct de vedere matematic nici măcar ăn cazul cel mai simplu. Acesta este următorul: presupunem că ratele anuale ale dobînzilor formează un şir de variabile aleatoare independente şi identic repartizate $(i_n)_n$. Atunci factorii de fructificare q_n vor forma şi ei un şir de variabile aleatoare independente şi identic repartizate, şi la fel şi coeficienţii de actualizare $\alpha_n = 1/q_n$.

O rată este suficientă dacă $S_1 \le 0 \Leftrightarrow r \ge q_1 S_0 \Leftrightarrow \alpha_1 \ge S_0/r$

Două rate sunt suficiente dacă $S_2 \le 0 \Leftrightarrow S_0 q_1 q_2 - r(1+q_2) \le 0 \Leftrightarrow \alpha_1 + \alpha_1\alpha_2 \ge S_0/r$

Trei rate sunt suficiente dacă $S_3 \le 0 \Leftrightarrow \alpha_1 + \alpha_1\alpha_2 + \alpha_1\alpha_2\alpha_3 \ge S_0/r$

În general, n rate sunt suficiente dacă

$$S_n \le 0 \Leftrightarrow \alpha_1 + \alpha_1\alpha_2 + ... + \alpha_1\alpha_2...\alpha_n \ge S_0/r \qquad (6.1.5)$$

Probabilitatea ca aşa ceva să se întîmple este $F_n(S_0/r)$, unde F_n este funcţia de repartiţie a variabilei aleatoare $X_n = \alpha_1 + \alpha_1\alpha_2 + ... + \alpha_1\alpha_2...\alpha_n$.

Fie $F = F_1$ funcţia de repartiţie a coeficientului de actualizare α_1.

Observăm că $X_2 = X_1(1 + \alpha_2)$. Dacă notăm $Y_2 = 1 + \alpha_2$, atunci Y_2 este independentă de X_1 , are funcţia de repartiţie $G(x) = F(x - 1)$ şi $X_2 = X_1 Y_2$.

Cum găsim repartiţia lui X_2 ?

Dacă X şi Y sunt două variabile aleatoare independente, cu repartiţiile F şi G, atunci XY are o repartiţie H care se obţine din F şi G printr-o operaţie numită ***convoluţie multiplicativă*** **.** Să o notăm cu „•”. Deci $H = F \bullet G$. Dacă F şi G sunt repartiţii discrete, atunci H se poate calcula astfel

$$F \sim \begin{pmatrix} x_1 & x_2 & \dots & x_m \\ p_1 & p_2 & \dots & p_m \end{pmatrix}, G \sim \begin{pmatrix} y_1 & y_2 & \dots & y_n \\ q_1 & q_2 & \dots & q_n \end{pmatrix} \Rightarrow F \bullet G \sim \begin{pmatrix} x_1y_1 & x_1y_2 & \dots & x_my_n \\ p_1q_1 & p_1q_2 & \dots & p_mq_n \end{pmatrix}$$

Scriptul convm face exact acest lucru.

```
##Calculeaza convolutia multiplicativa intre (x,p) si
(y,q)
convm<-function(x,p,y,q)
{xmy=outer(x,y,"*");pmq=outer(p,q,"*")
a=tapply(pmq,xmy,sum)
x1=as.numeric(names(a))
p1=x1;m=length(x1)
for (i in 1:m) {p1[i]=a[[i]]}
a=list(x1=x1,p1=p1)
a}
```

Variabilele sunt vectorii $x=(x_i)_{1\le i\le m}$, $p=(p_i)_{1\le i\le m}$, $y=(y_j)_{1\le j\le n}$, $q=(q_j)_{1\le j\le n}$.
Scriptul nu verifică dacă p şi q sunt vectori de probabilitate, adică dacă sumele sunt 1. Cititorul este invitat să îl modifice în aşa fel încît să se dea un mesaj de eroare în caz că p sau q nu sunt de sumă 1.
În primul rînd se calculează matricile $\texttt{xmy}_{i,j}=x_iy_j$ şi $\texttt{pmq}_{i,j}=p_iq_j$. Aici se vede forţa operaţiei `outer` din „R”. În rîndul următor se foloseşte o instrucţiune şi mai puternică, `tapply(pmq,xmy,sum)`. Ea face următorul lucru: aplică matricii `pmq` funcţia `sum` depinzînd de `xmy`. Explicit, al doilea argument (adică `xmy`) este văzut ca o colecţie de etichete. Se face o listă cu aceste etichete şi se adună (funcţia `sum`) elementele din `pmq` care au această etichetă.
De exemplu, dacă $x=(-1,0,1,2)$, $y=-x=(1,0,-1,-2)$, $p=q=(\,¼\,,¼\,,¼\,,¼\,)$, avem

```
> xmy=outer(x,y,"*");xmy
     [,1] [,2] [,3] [,4]
[1,]   -1    0    1    2
[2,]    0    0    0    0
[3,]    1    0   -1   -2
[4,]    2    0   -2   -4
```

Deci $\texttt{xmy}_{i,j}=x_iy_j$. Apar 7 valori: -4,-2,-1,0,1,2

```
> pmq=outer(p,q,"*");pmq
       [,1]   [,2]   [,3]   [,4]
[1,] 0.0625 0.0625 0.0625 0.0625
[2,] 0.0625 0.0625 0.0625 0.0625
[3,] 0.0625 0.0625 0.0625 0.0625
[4,] 0.0625 0.0625 0.0625 0.0625
> a=tapply(pmq,xay,sum);a
    -4     -2     -1      0      1      2
0.0625 0.1250 0.1250 0.4375 0.1250 0.1250
```

Cea mai frecventă valoare a fost 0: apare în 7 locuri. Deci probabilitatea sa este 7/16 =0.4375.
Apelarea funcţiei se face prin comanda
`a =convm<-function(x,p,y,q)`
şi rezultatul este o listă, a. Primul element este format din vectorul cu elementele x_iy_j, sortat în ordine crescătoare iar al doilea cu probabilităţile $p\bullet q$

Exemplu

```
> a=convm(x,p,y,q);a
```

```
[[1]]
[1] -4 -2 -1  0  1  2

[[2]]
[1] 0.0625 0.1250 0.1250 0.4375 0.1250 0.1250
```

Să presupunem acum că rata dobînzii este $i \sim \begin{pmatrix} 5 & 10 & 15 & 20 \\ .2 & .3 & .3 & .2 \end{pmatrix}$. Ratele se văd în primul rînd, exprimate în procente iar probabilităţile lor în al doilea rînd. Atunci $q_n \sim \begin{pmatrix} 1.05 & 1.1 & 1.15 & 1.2 \\ .2 & .3 & .3 & .2 \end{pmatrix}$ iar coeficienţii de actualizare $\alpha_n \sim \begin{pmatrix} \frac{1}{1.05} & \frac{1}{1.1} & \frac{1}{1.15} & \frac{1}{1.2} \\ .2 & .3 & .3 & .2 \end{pmatrix}$

Vrem să găsim repartiţia lui $\alpha_1 + \alpha_1\alpha_2 = F \bullet G$ unde $F = \begin{pmatrix} \frac{1}{1.05} & \frac{1}{1.1} & \frac{1}{1.15} & \frac{1}{1.2} \\ .2 & .3 & .3 & .2 \end{pmatrix}$ şi $G = \begin{pmatrix} 1+\frac{1}{1.05} & 1+\frac{1}{1.1} & 1+\frac{1}{1.15} & 1+\frac{1}{1.2} \\ .2 & .3 & .3 & .2 \end{pmatrix}$. Atunci instrucţiunile sunt

```
x=c(1/1.05,1/1.1,1/1.15,1/1.2);y=x+1;p=c(2,3,3,2)/10;q=p
convm(x,p,x+1,p)
$x1
 [1] 1.527778 1.557971 1.590909 1.594203 1.625709 1.626984
1.660079 1.666667 1.697723 1.699605 1.735537
[12] 1.746032 1.774892 1.780538 1.818182 1.859410

$p1
 [1] 0.04 0.06 0.06 0.06 0.09 0.04 0.09 0.06 0.06 0.09 0.09
0.04 0.06 0.06 0.06 0.04
```

Sau, scris şi mai explicit: $(x,p) \bullet (y,q) = (z,r)$ cu z un vector cu 16 valori şi r de asemenea cu 16 valori. S-a întîmplat că toate produsele $x_i y_j$ au fost diferite.

Probabilitatea $P(X \geq x)$ este calculată de scriptul tailfrep

```
##coada unei repartitii discrete (x,p)calculata in t
tailfrep<-function(t,x,p)
{tailfrep=1-sum(p[x<t]);tailfrep}
```

De exemplu, dacă dorim să ştim care este probabilitatea ca o sumă $S_0 = 1200$ lei să poată fi achitată în două rate $r = 700$ lei, calculăm

```
a=convm(x,p,y,q);prob=tailfrep(1200/700,a[[1]],a[[2]]);prob
```

şi răspunsul este prob = 0.35. Dacă mărim rata la 800,

```
a=convm(x,p,y,q);prob=tailfrep(1200/800,a[[1]],a[[2]]);prob
prob = 1.
```

Era şi de aşteptat.

Cum calculăm celelalte probabilităţi?

De exemplu $X_3 = \alpha_1 + \alpha_1\alpha_2 + \alpha_1\alpha_2\alpha_3 = \alpha_1(1 + \alpha_2(1 + \alpha_3))$ Observăm că, deoarece variabilele α_i au fost presupuse independente şi identic repartizate, $\alpha_2(1 + \alpha_3)$ are

aceeaşi repartiţie ca şi X_2 iar α_1 are aceeaşi repartiţie ca şi α_3. Ca atare X_3 are aceeaşi repartiţie ca $\alpha_3(1 + X_2)$.
Mai general, $X_n = \alpha_1(1 + (\alpha_2 + \alpha_2\alpha_3 + ...+ \alpha_2...\alpha_n))$ are aceeaşi repartiţie cu $\alpha_n(1 + \alpha_1 +...+\alpha_1...\alpha_{n-1})$, adică cu $\alpha_n(1 + X_{n-1})$. Dacă notăm faptul că două variabile aleatoare au aceeaşi repartiţie cu $X \overset{D}{=} Y$, putem scrie

$$X_n \overset{D}{=} \alpha_n(1 + X_{n-1}) \tag{6.1.6}$$

Sau, în termeni de repartiţii,

$$F_n = F \bullet h(F_{n-1}) \tag{6.1.7}$$

unde $h(F_{n-1})$ este repartiţia variabilei aleatoare $X_{n-1} + 1$.

Scriptul `convmrep` calculează repartiţia lui X_n

```
##convolutia multiplicativa pentru alfa: F_n+1 =
F_n*delta(1)convmultF
convmrep<-function(n,x,p)
{y=1;q=1
for (i in 1:n) {a=convm(x,p,y,q);y=a[[1]]+1;q=a[[2]]}
a}
```

De exemplu, pentru a afla repartiţia lui X_4 scriem

```
a=convrep(4,x,p);x1=a[[1]];p1=a[[2]]
```

Vectorii x1 şi p1 vor avea lungimea de 256. Valoarea minimă pentru x1 este 2.588735 iar cea maximă este 3.545951. Probabilitatea ca 4 rate în valoare de 400 de lei să fie suficiente este dată de comanda

```
a=convmrep(4,x,p);x1=a[[1]];p1=a[[2]];prob=tailfrep(1200/400,x
1,p1);prob
```

şi este egală cu 0.5361
Dacă dorim să mărim probabilitatea, mărim puţin rata la 450 lei

```
a=convmrep(4,x,p);x1=a[[1]];p1=a[[2]];prob=tailfrep(1200/450,x1,p1);
prob
[1] 0.9836
```

Deja putem paria că este suficient.
Problema este că nu putem aplica acest algoritm de prea multe ori. Deja la 4 convoluţii, şirul de valori pe care îl poate lua X_4 este de 256; în principiu este de 4^n valori. De exemplu, la $n = 10$

```
a=convrep(10,x,p);x1=a[[1]];p1=a[[2]]
```

vectorul `x1` are `n= 1048576` componente. Calculul durează 5 minute. Nu o să afişăm toate valorile, ci doar un sumar prin instrucţiunea summary

```
summary(x1)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
4.192   5.251   5.574   5.599   5.921   7.722
```

Deci minimul este 4.192 iar maximul este 7.722.

Probabilitatea ca acest credit să fie acoperit în 10 rate a 200 de lei este

```
prob=tailfrep(1200/200,x1,p1);prob = 0.178917
```

iar în rate de 250 lei este

```
prob=tailfrep(1200/250,x1,p1);prob = 0.9737461
```

Funcţia `gfrep(n,x,p)` face graficul funcţiei de repartiţie a repartiţiei (x,p) în n puncte echidistante calculate între min x_1 şi max x_1.Iată graficul funcţiei de repartiţie al lui X_{10}

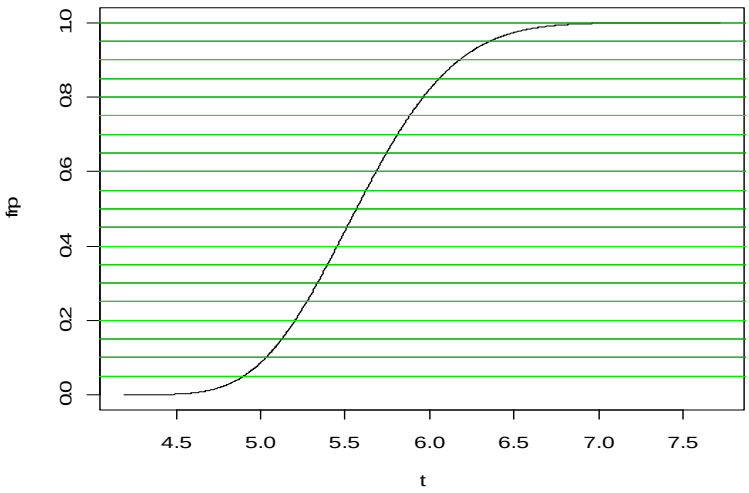


Fig 6.1

Pe axa Ox sunt valorile lui x, între minim şi maxim. În graficul următor prezentăm diferenţele $[F(t+h) - F(t)]/h$ pentru h = (max x_1 – min x_1)/998 care, în ipoteza că repartiţia lui F_n ar avea o limită absolut continuă, ar mima densitatea sa. Dar nu ştim dacă există o asemenea limită: este o problemă nerezolvată de matematicieni

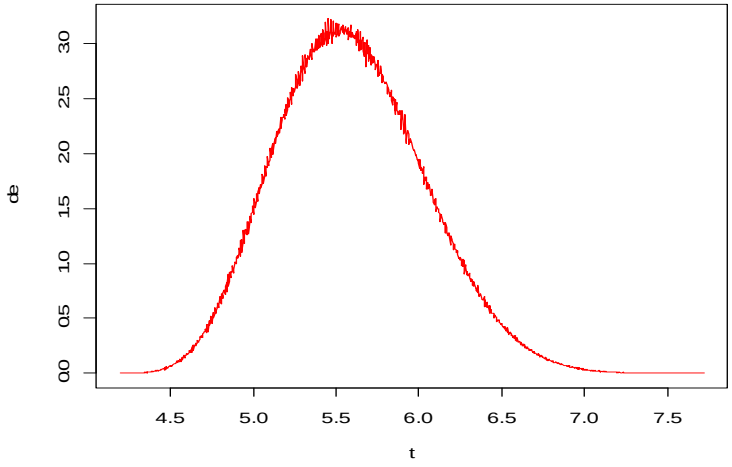


Fig 6.2

Pentru valori mai mari ale lui n nu este realist să aplicăm acest algoritm: dacă la n = 10 X_n are `1048576` de componente, la n = 20 o să aibă 10^{12} . Am putea mări numărul de iteraţii dacă am modifica problema: dacă rata dobînzii nu are decît două valori $i \sim \begin{pmatrix} 0.10 & 0.15 \\ .5 & .5 \end{pmatrix}$. Ratele se văd în primul rînd, exprimate în procente iar probabilităţile lor în al doilea rînd. Atunci $q_n \sim \begin{pmatrix} 1.10 & 1.15 \\ .5 & .5 \end{pmatrix}$ iar coeficienţii de actualizare $\alpha_n \sim \begin{pmatrix} \frac{1}{1.10} & \frac{1}{1.15} \\ .5 & .5 \end{pmatrix}$. Atunci am putea calcula repartiţia chiar şi pentru X_{20}

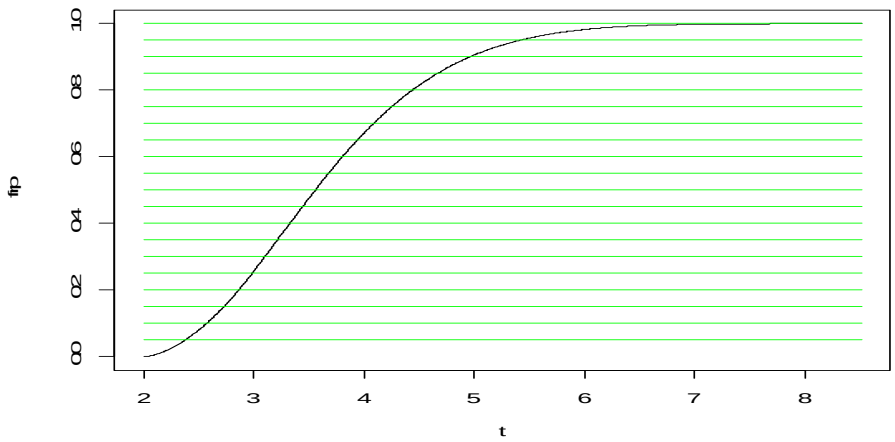


Fig 6.3

şi „densitatea”

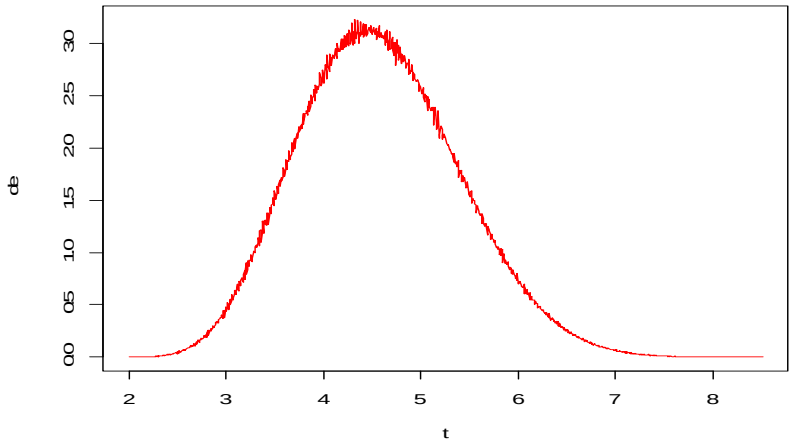


Fig 6.4

Probabilitatea ca 20 de rate a 100 de lei să fie suficiente este

```
prob=tailfrep(1200/250,x1,p1);prob = 0.1249514
```

6.1 9. Calculul mediei şi varianţei variabilelor $X_n = \alpha_1+\alpha_1\alpha_2+...+\alpha_1\alpha_2...\alpha_n$

Dacă *n* este mare – de ordinul zecilor, atunci algoritmul de mai sus nu poate fi aplicat dacă ţinem ca modelul să fie cît de cît realist. Dar dacă presupunem că variabilele $\alpha_i \in (0,1)$ sunt independente şi identic repartizate, atunci putem încerca estimări bazate pe inegaltatea lui cebîşev. Chiar dacă sunt slabe, ne dau o idee despre repartiţie. Să presupunem că

$$E\alpha_i = a,\ E\alpha_i^2 = b,\ EX_n = \mu,\ \mathrm{Var}(X_n) = v \tag{6.1.8}$$

Datorită faptului că variabilele $(\alpha_i)_i$ au fost presupuse independente, avem

$\mu = E\alpha_1 + E\alpha_1E\alpha_2 +....+ E\alpha_1E\alpha_2...E\alpha_n = a +a^2 +...+ a^n$, adică

$$\mu = \frac{a\left(1-a^n\right)}{1-a} \tag{6.1.9}$$

Pe de altă parte $v = \mathrm{Var}\sum_{i=1}^{n}\alpha_1...\alpha_i = \sum_{i,j} c_{i,j}$ unde $c_{i,j} = \mathrm{cov}\left(\alpha_1...\alpha_i, \alpha_1...\alpha_j\right) = E\left(\alpha_1...\alpha_i \cdot \alpha_1...\alpha_j\right) - a^{i+j}$. Dacă $i < j$ avem $c_{i,j} = b^i a^{j-i} - a^{i+j}$, deci

$$c_{i,j} = b^{\min(i,j)} a^{|i-j|} \tag{6.1.10}$$

În concluzie, după efectuarea calculelor ajungem la

$$v = \frac{2a}{1-a}\left(\frac{b\left(1-b^{n-1}\right)}{1-b} - \frac{ab\left(a^{n-1}-b^{n-1}\right)}{a-b}\right) + \frac{b\left(1-b^n\right)}{1-b} - \left(\frac{a\left(1-a^n\right)}{1-a}\right)^2 \tag{6.1.11}$$

Să presupunem, ca la exemplul de la paragraful 7, că rata dobînzii este repartizată Uniform(0.05, 0.15)

Atunci $q \sim$ Uniform(1.05, 1.15). Cum $U \sim U(\alpha,\beta)$, $\Rightarrow E\frac{1}{U} = \frac{\ln\beta - \ln\alpha}{\beta-\alpha}$ şi $E\frac{1}{U^2} = \frac{1}{\alpha\beta}$ avem, în cazul nostru, $a = \frac{\ln\beta - \ln\alpha}{\beta-\alpha}$, $b = \frac{1}{\alpha\beta}$

```
alfa=1.05;beta=1.15;a=(log(beta)-log(alfa))/(beta-alfa);b=1/alfa/beta
```

Rezultă $a = 0.9097178$, $b = 0.8281573$

Scriptul următor, medvar(*n,a,b*) calculează media μ şi varianţa *v*

```
##calculez media si varianta lui X_n
#=alfa_1+alfa_1alfa_2....
##daca Ealfa_n = a, Ealfa_n^2=b
medvar<-function(n,a,b){
mu=a*(1-a^n)/(1-a)
s=b*(1-b^(n-1))/(1-b)-a*b*(a^(n-1)-b^(n-1))/(a-b)
v=2*a*s/(1-a)+b*(1-b^n)/(1-b)-(a*(1-a^n)/(1-a))^2
answ=list(rate=n,media=mu,varianta=v)
answ}
```

De exemplu, dacă $n = 60$, atunci X_{60} are media 10.04189 şi varianţa 0.402313

```
ans=medvar(60,a,b);ans
$rate
[1] 60
```

```
$media
[1] 10.04189

$varianta
[1] 0.402312
```

Atunci abaterea medie pătratică σ este 0. 6342815 . Deci $P(\ 10.02 - k\cdot 0.63 < X_{20} < 10.02 + k\cdot 0.63) > 1 - 1/k^2$. O aproximaţie bună este că $P(\ 10.02 - k\cdot 0.63 < X_{20}) > 1 - \frac{1}{2k^2}$. Pentru $k = 4$, găsim că $P(X_{20} > 7.503) > 96.88\%$. Dacă $S_0 = 1200$, punem condiţia ca $1200/r < 7.5 \Rightarrow r > 160$. O rată de 160 lei ar fi suficientă, cu probabilitate mai mare decît 97% ca să achite creditul de 1200 lei în 20 de rate.

6.2. Probabilitatea de ruină

O companie plăteşte în fiecare zi/săptămînă/lună (depinde cum se face balanţa) X_n lei şi cîştigă Y_n lei. Pierderea sa zilnică este atunci $\xi_n = X_n - Y_n$. Pierderea după n zile este

$$S_0 = 0, \quad n \geq 1 \ \Rightarrow \ S_n = \xi_1 + ... + \xi_n \tag{6.2.1}$$

Dacă admitem (o simplificare matematică!) că variabilele $(X_n)_n$ sunt i.i.d., variabilele $(Y_n)_n$ sunt şi ele i.i.d. şi că X_n este independentă de Y_n pentru orice n, atunci repartiţia variabilei aleatoare S_n este convoluţia repartiţiilor variabilelor aleatoare ξ_n. Iar repartiţia lui ξ_n este convoluţia dintre repartiţia F_X a variabilelor X_n şi repartiţia F_{-Y} a variabilelor $-Y_n$. Convoluţia se notează cu „∗”.

Vom considera repartiţiile F_X şi F_Y ca fiind discrete – altfel nu prea putem folosi calculatorul . Atunci $F_X = \begin{pmatrix} x_1 & x_2 & ... & x_m \\ p_1 & p_2 & ... & p_m \end{pmatrix}$, $F_Y = \begin{pmatrix} y_1 & y_2 & ... & y_m \\ p_1 & p_2 & ... & p_m \end{pmatrix}$ şi rezultă că

$F_X * F_Y = \begin{pmatrix} x_1 + y_1 & x_1 + y_2 & ... & x_m + y_m \\ p_1 q_1 & p_1 q_2 & ... & p_m q_n \end{pmatrix}$. Fiind discrete, cele două repartiţii pot fi codificate prin (x,p) şi (y,q). În cazul acesta putem scrie $(x,p) * (y,q) = (x_1,p_1)$

Scriptul `conva(x,p,y,q)` calculează exact acest lucru: produce vectorii x_1 şi p_1. Este o operaţie mai obişnuită decît convoluţia multiplicativă prezentată în capitolul anterior. Diferenţa este că vaorile x_i şi y_j se **adună**, nu se înmulţesc.

```
##convolutia aditiva obisnuita cu functia tapply
conva<-function(x,p,y,q)
{xay=outer(x,y,"+");pmq=outer(p,q,"*")
a=tapply(pmq,xay,sum)
x1=as.numeric(names(a))
p1=x1;m=length(x1)
for (i in 1:m) {p1[i]=a[[i]]}
a=list(x1=x1,p1=p1)
a}
```

Să presupunem, ca să revenim la scenariul nostru, că $X_n \sim \begin{pmatrix} 0 & 1 & 3 & 10 \\ .3 & .4 & .2 & .1 \end{pmatrix}$. Probabilitatea să avem o zi fără cheltuieli este 30%, să cheltuim 1 UM este 40%, să cheltuim 3 UM este 20% şi, în zile rele, putem cheltui chiar 10 UM, cu probabilitatea

10%. Cîştigurile le presupunem a avea repartiţia $Y_n \sim \begin{pmatrix} 1 & 2 & 3 & 4 \\ .4 & .3 & .2 & .1 \end{pmatrix}$. Am ales exemplul în aşa fel încît $EX_n = EY_n = 2$. Care este repartiţia lui ξ_n?

Instrucţiunile vor fi

```
x=c(0,1,3,10);y=c(1,2,3,4);p=c(3,4,2,1)/10;q=c(4,3,2,1)/10
a=conva(x,p,-y,q);a
$x1  -4 -3 -2 -1  0  1  2  6  7  8  9
$p1 0.03 0.10 0.17 0.26 0.20 0.06 0.08 0.01 0.02 0.03 0.04
```

Probabilitatea să nu avem pierderi este este $P(\xi_n \le 0)$, pentru care avem scriptul

```
frep<-function(t,x,p)
{frep=sum(p[x<=t])frep}
```

Rezultă concret `frep<(0,x1,p1)` = 0.76

Dacă dorim să calculăm repartiţia sumei S_n apelăm funcţia convarep. De exemplu, pentru $n = 3$ avem

```
a=conva(x,p,-y,q);x1=a[[1]];p1=a[[2]];a=convarep(3,x1,p1);a
$xn
[1] -12 -11 -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5
6
[20]   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24
25  26  27
$yn
[1] 0.000027 0.000270 0.001359 0.004762 0.012921 0.028188 0.050933 0.077658
[9] 0.100392 0.111242 0.106719 0.088602 0.065603 0.045132 0.030813 0.027390
[17] 0.031509 0.037536 0.041444 0.038796 0.030717 0.020202 0.012105 0.006282
[25] 0.003435 0.003792 0.004395 0.005130 0.004788 0.003618 0.001993 0.000870
[33] 0.000405 0.000056 0.000111 0.000174 0.000219 0.000204 0.000144 0.000064
```

Pentru probabilitatea de a nu fi în pierdere

```
x1=a[[1]];p1=a[[2]];prob=frep(0,x1,p1);prob = 0.648676
```

Observăm că este mai mică.

Să presupunem acum că avem o rezervă de u UM. Dacă pierderea depăşeşte u, declarăm faliment. **Care este probabilitatea de faliment după *n* zile?**

Putem reformula problema în termeni de pierdere maximă.

Fie $L_n = \max(S_0,S_1,...,S_n)$. Aceasta este **pierderea maximă** pe care o avem după n zile. Ruina apare dacă $L_n > u$. Probabilitatea $P(L_n > u)$ se notează cu $\psi_n(u)$ şi se numeşte **probabilitatea de ruină** după n paşi. Observăm că ea este coada funcţiei de repartiţie a pierderii maxime L_n.

Fie $F = F_X * F_{-Y}$ repartiţia pierderii ξ_n şi F_n repartiţia sumei S_n.

Observăm că $L_1 = (\xi_1)_+$. Dacă o variabilă aleatoare are repartiţia F, notăm cu F_+ repartiţia părţii sale pozitive. Adica $F_+(x) = F(x)$ dacă $x \ge 0$ şi $F_+(x) = 0$ dacă $x < 0$.

Aşadar $F_1 = F_+$

Avem apoi $L_2 = \max(0,\xi_1,\xi_1 + \xi_2) = \max(0,\xi_1 + 0,\xi_1 + \xi_2) = \max(0,\xi_1 + \max(0,\xi_2))$. Dar ξ_1 are aceeaşi repartiţie cu ξ_2 iar $\max(0,\xi_2)$ are aceeaşi repartiţie ca şi $L_1 = (\xi_1)_+$, deci L_2 are aceeaşi repartiţie ca $(L_1 + \xi_2)_+$. În concluzie, repartiţia sa va fi $F_2 = (F_1 * F)_+$.

Analog avem $L_3 \overset{D}{=} (L_2+\xi_3)_+$ şi, în general rezultă recurenţa

$$L_{n+1} \overset{D}{=} (L_n+\xi_{n+1})_+ \Leftrightarrow F_{n+1} = (F_n * F)_+ \tag{6.2.2}$$

Scriptul care calculează partea pozitivă a unei repartiţii este `ppos`

```
## partea pozitiva a unei repartitii
ppos<-function(x,p)
{a=data.frame(v=c(0,x[x>0]),p=c(sum(p[x<=0]),p[x>0]))
a}
```

cel care calculează partea pozitivă a unei convoluţii este `convap`

```
##convolutia pozitiva a doua repartitii
convap<-function(x,p,y,q)
{a=conva(x,p,y,q);x1=a[[1]];p1=a[[2]]
a=ppos(x1,p1)
a}
```

iar cel care calculează convoluţia pozitivă iterată, adică exact repartiţia F_n din (2.2) este `convaprep`

```
##convolutia pozitiva iterata pentru Fn+1=(F*Fn)+
# (x,p) este pierderea, (y,p) e cistigul
# se convoluteaza la inceput (x,p) cu (-y,q) si rezulta F
convaprep<-function(n,x,p,y,q)
{y=-y;z=conva(x,p,y,q)
x=z[[1]];p=z[[2]] #aici s-a facut F
y=0;q=1
for (i in 1:n) {z1=convap(x,p,y,q);y=z1[,1];q=z1[,2]}
z1}
```

Astfel, dacă dorim să calculăm repartiţia pierderii maxime după 10 zile, folosim comanda

```
a=convaprep(10,x,p,y,q);a
   v            p
   0 3.176752e-01
   1 6.331282e-02
   2 6.736288e-02
   3 4.082990e-02
   4 4.043586e-02
   5 4.000655e-02
   6 4.474262e-02
   7 4.794617e-02
   8 4.961629e-02
   9 4.887334e-02
  10 3.192275e-02
  11 2.745033e-02
  12 2.284033e-02
  13 2.112670e-02
  14 1.962573e-02
  15 1.826939e-02
  16 1.636947e-02
  17 1.408383e-02
  18 1.158424e-02
  19 9.172632e-03
  ..............
```

Rezultatul este prezentat sub forma unei matrici $N\times 2$, unde N este numărul de valori pe care le poate lua S_n. În cazul nostru, $N = 91$.

Să presupunem că managerul companiei are la dispoziţie un capital $u = 20$ UM pentru a plăti cheltuielile. Care este probabilitatea ca acest capital să nu fie suficient?

Este $\psi_{10}(20) = P(L_{10} > 20)$. Pentru ea avem la dispoziţie funcţia `tailfrep`

```
prob=tailfrep(20,x1,p1);prob = 0.04675292
```

Dar după 100 de zile?

```
> tailfrep(20,x1,p1); prob =0.4706843
```

Deja probabilitatea este aproape de ½.

După 200 de zile, probabilitatea este $\psi_{200}(20) = 0.6063088$

De curiozitate, $\psi_{200}(100) = 0.0224711$. Probabilitatea de ruina este de doar 2.2% dacă se dispune de o rezervă de 100 lei.

Se poate arăta că dacă $EX \geq EY$ (acesta este cazul nostru), ***ruina este sigură.*** Adică $\psi_n(u) \to 1$ dacă $n \to \infty$ indiferent de capitalul iniţial u. Este adevărat că dacă $EX = EY$ se converge lent.

Ruina nu este sigură doar dacă $EX < EY$.

Să înlocuim în exemplul nostru probabilităţile în aşa fel încît

$X_n \sim \begin{pmatrix} 0 & 1 & 3 & 10 \\ .6 & .2 & .1 & .1 \end{pmatrix}$. Acum $EX = 1.5$, $E\xi_n = -0.5$, deci ruina nu mai este sigură. Repartiţia lui L_{100} e calculată cu instrucţiunile

```
> x=c(0,1,3,10);p=c(6,2,1,1)/10;a=convaprep(100,x,p,y,q)
> x1=a[,1];p1=a[,2]
> summary(x1)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0     225     450     450     675     900
> tailfrep(20,x1,p1); prob =  0.135263
```

Un capital iniţial de doar 20 UM face ca probabilitatea de ruină după 100 de zile să fie în jur de 13.5%.

Putem compara această probabilitate cu marginea dată de Lundberg.

Notăm $\psi(u) = \lim_{n\to\infty} \psi_n(u)$. Marginea lui Lundberg este $\psi(u) \leq e^{-Ru}$ unde R este soluţia ecuaţiei $\boldsymbol{m}_\xi(t) = 1$.

```
##calculul functiei generatoare de momente
mgf<- function(t,x,p)
{y=exp(t*x); mgf=y%*%p
mgf}
```

Soluţia este R = -0.08551

Marginea Lundberg: $\psi(20) \leq e^{-20u} = e^{-1.7102} = 0.1808296$

Marginea Lundberg este una pesimista.

6. 3. Probleme de asigurări. Suma daunelor, recurenţa Panjer

Pentru un asigurator, fiecare client reprezintă o cerere potenţială de daună, deci el trebuie să fie pregătit să îşi onoreze obligaţiile. Ce rezervă trebuie să aibă?

În cel mai simplu model, clienţii sunt modelaţi printr-un şir de variabile aleatoare şi identic repartizate $(X_n)_n$ cu o funcţie de repartiţie F. Acesta se numeşte un ***portofoliu omogen de tip F.*** În planificarea rezervei, mărimea N a portofoliului (adică numărul de clienţi de tip F) este necunoscută. Este o altă variabilă aleatoare N cu valori numere naturale. În modelul cel mai simplu, contorul N va fi independent de variabilele $(X_n)_n$.

Atunci suma pe care trebuie să o aibă la dispoziţie managerul va fi o variabilă aleatoare

$$S_N = X_1 + X_2 + ...+ X_N \text{ (dacă } N \geq 1) \text{ sau } S_N = 0 \text{ (dacă } N = 0) \qquad (6.3.1)$$

Asiguratorul îşi va alege un risc α (de exemplu $\alpha = 5\%$, $\alpha = 1\%$ etc) şi va aloca portofoliului de tip F o suma $\sum$ în aşa fel ca $P(S_N > \sum) \leq 1 - \alpha$

Ca să fie posibil acest lucru, va trebui să ştie repartiţia lui S_N. De regulă, acest lucru este imposibil. O primă abordare ar fi să se folosească inegalitatea lui Cebîşev

$$P(|S_N - ES_N| > k\sigma(S_N)) < 1/k^2 \qquad (6.3.2)$$

Sau, acelaşi lucru, dar pus în formă afirmativă

$$P(ES_N - k\sigma(S_N) \leq S_N \leq ES_N + k\sigma(S_N) > 1 - 1/k^2 \qquad (6.3.3)$$

pentru că există formule pentru ES_N şi $\text{Var}(S_N)$, numite formulele lui Wald [9]

$$ES_N = EX_1 \cdot EN, \ \text{Var}(S_N) = \text{Var}(X_1)EN + \text{Var}(N)E^2X_1 \qquad (6.3.4)$$

Exemplul 1. Să spunem că $X_n \sim \text{Bin}(10, ½)$ şi $N \sim \text{Poisson}(100)$. Ne aşteptăm să avem în jur de 100 de clienţi şi fiecare să ceară, în medie 5 lei. Ce sumă să avem pregătită?

Avem $EX_1 = 10 \cdot ½ = 5$, $\text{Var}(X_1) = 10 \cdot ½ \cdot (1 - ½) = 2.5$, $EN = 100$, $\text{Var}(N) = 100$

Răspuns. $ES_N = 5 \cdot 100 = 500$, $\text{Var}(S_N) = 2.5 \cdot 100 + 100 \cdot 5^2 = 2750 \Rightarrow \sigma(S_N) = \sqrt{2750} \approx 52.44$

Aplicînd (6.3.4) vedem că $P(500 - k \cdot 52.44 \leq S_N \leq 500 + k \cdot 52.44) > 1 - 1/k^2$

Alegînd $k = 10$, avem $P(S_N \leq 1024) > 0.99$. Deci ar trebui planificaţi 1024 de lei.

Inegalitatea lui Cebîşev poate fi îmbunătăţită, dacă înlocuim varianţa cu semivarianţa superioară $\text{Var}_+(X) = E(X - EX)_+^2$ şi abaterea standard cu abaterea superioară $\sigma_+(X)$. S-a constatat empiric faptul că nu se greşeşete prea mult dacă se ia $\text{Var}_+(X) = \text{Var}(X)/2 \Leftrightarrow \sigma_+ = \sigma/\sqrt{2}$

Inegalitatea (3.3) devine

$$P(S_N \leq ES_N + k\sigma(S_N)/\sqrt{2}) > 1 - 1/k^2 \qquad (6.3.5)$$

În cazul exemplului de mai sus, ar rezulta $P(S_N \leq 500 + 370.8) > 0.99 \Leftrightarrow P(S_N \leq 871) > 0.99$

Parcă ar fi suficienţi 871 lei, nu 1024. Poate că e nevoie şi de mai puţini?

Dacă dorim să calculăm o limită şi mai precisă, ar trebui să putem calcula repartiţia lui S_N.

Uneori acest lucru este posibil. Dacă acceptăm că X_n au o repartiţie discretă,

[9] Vezi, de exemplu Gheorghiţa Zbăganu, *Metode matematice în teoria riscului şi actuariat,* Editura Universităţii Bucureşti, 2004

$X_n \sim \begin{pmatrix} 0 & 1 & ... & k \\ p(0) & p(1) & .... & p(k) \end{pmatrix}$ şi că $N \sim \begin{pmatrix} 0 & 1 & 2 & ... \\ q(0) & q(1) & q(2) & ... \end{pmatrix}$, atunci $S_N \sim \begin{pmatrix} 0 & 1 & 2 & ... \\ f(0) & f(1) & f(2) & ... \end{pmatrix}$ şi probabilităţile $f(n)$ se pot calcula.

Într-adevăr,

$$f(j) = P(S_N = j) = \sum_{n=1}^{j} P(S_n = j) q_j \qquad (6.3.6)$$

Pentru probabilităţile $P(S_n = j)$ avem la dispoziţie scriptul convarep

```
Convarep<-function(n,x,p)
{y=0;q=1
for (i in 1:n){a=conva(x,p,y,q);y=a[[1]];q=a[[2]]}
xn=a[[1]];yn=a[[2]]
a=list(xn=xn,yn=yn)}
```

care calculează a n –a convoluţie a repartiţiei (x,p) a lui X_n.

Dificultatea în programare nu ar fi aşa de mare; facem o matrice $pr_{j,i} = P(S_j = i)P(N = i)$ şi calculăm sumele de la (3.6)

Iată scriptul convarepN (convoluţii aditive repetate de N ori) care calculează probabilităţile f_j şi funcţia de repartiţie $F(j) = f_0 + \ldots + f_j$

```
## functia convarepN calculeaza repartitia sumei S_N
## daca nici x nici N nu au prea multe valori
convarepN<-function(x,p,N,q){
nx=length(x);nN=length(N);nmax=nx*nN;f=c(rep(0,nmax));F=f
pr=c(rep(0,nN*nmax));dim(pr) = c(nN,nmax)
pr[1,1]=q[1]
for (j in 2:nN){a=convarep(j-1,x,p);p1=a[[2]];nj=length(p1)
for (i in 1:nj) {
pr[j,i]=p1[i]*q[j]
}}
for (j in 1:nmax) {f[j]=sum(pr[,j])}
for (j in 1:nmax) {F[j]=sum(f[1:j])}
a= list(f=f,F=F)a}
```

Rezultatul este prezentat într-o listă. Apelarea se face prin comanda

```
a= convarepN(x,p,N,q);f=a[[1]];F=a[[2]]
```

Problema este că prin apelarea scriptului convarep se face un consum mare de resurse. De exemplu, dacă $N \sim$ Uniform($\{0,1,...,40\}$ şi $X_n \sim$ Binom(10, ½), rularea scriptului convarepN durează 2 minute. Scriptul se poate optimiza, renunţînd la scriptul convarep, dar programul se mai complică. Putem compara cît de eficientă este metoda inegalităţii lui Cebîşev faţă de cea exactă.

Exemplul 2. $X_n \sim$ Binom(10, ½), $N \sim$ Uniform($\{0,1,...,40\} \Rightarrow EX = 5$, $\mathrm{Var}(X) = 2.5$, $EN = 20$, $\mathrm{Var}(N) = 140$, $ES_N = 5 \cdot 20 = 100$, $\mathrm{Var}(S_N) = \mathrm{Var}(X_1)EN + \mathrm{Var}(N)E^2X_1 = 2.5 \cdot 20 + 140 \cdot 5^2 = 3550$, aşadar $ES_N = 100$, $\sigma(S_N) \approx 59.58$. Formula (3.5) ne dă $P(S_N \le 100 + k \cdot 42.13075) > 1 - 1/k^2$. Dacă dorim un interval de predicţie cu risc 1% suntem forţaţi să luăm $k = 10$ şi obţinem predicţia $S_N \le 521$. Adică managerul de care

era vorba la început să aibă pregătiţi 521 de lei pentru a avea doar riscul 1% ca suma să fie insuficientă.
Calculul exact reclamă calculul cuantilei de 0.99 (sau, cum i se mai spune, valoarea la risc, adică primul k cu proprietatea că $F(k) \geq 0.99$
Instrucţiunea care face acest lucru este `which(F<.99)` şi obţinem 207 lei. Într-adevăr, $F(207) = 0.990797$, $F(206) = 0.9892509$[10]. De altfel, marginea dată de inegalitatea Cebîşev este absurdă, căci S_N nu ia decît 451 de valori.

Există cazuri în care calculul repartiţiei lui S_N se poate face cu mult mai rapid, şi cu resurse mult mai puţine, prin algoritmul lui Panjer.

Definiţie. Variabila aleatoare $N \sim \begin{pmatrix} 0 & 1 & 2 & ... \\ q_0 & q_1 & q_2 & .... \end{pmatrix}$ *se numeşte* ***contor de tip Panjer*** *dacă* $q_0 > 0$ *şi există* $a,b \in \Re$ *în aşa fel încît* $p_n = p_{n-1}\left(a + \frac{b}{n}\right)$ [11]

Există trei tipuri de contoare de tip Panjer

- $N \sim \text{Bin}(n,\alpha) \Rightarrow a = -\frac{\alpha}{1-\alpha}$, $b = \frac{\alpha(n+1)}{1-\alpha}$
- $N \sim \text{Poisson}(\lambda) \Rightarrow a = 0$, $b = \lambda$
- $N \sim \text{Negbin}(n, \alpha) \Rightarrow a = 1 - \alpha$, $b = (1 - \alpha)(n-1)$

Panjer a arătat că dacă $X_n \sim \begin{pmatrix} 0 & 1 & ... & k \\ p_0 & p_1 & ... & p_k \end{pmatrix}$ atunci numerele $f_n = P(S_N = n)$ satisfac recurenţa

$$f_0 = g(p_0),\ n \geq 1 \Rightarrow f_n = \frac{1}{1-ap_0}\sum_{j=1}^{n}\left(a + \frac{bj}{n}\right)p_j f_{n-j} \qquad (6.3.7)$$

Aici $g(x) = \sum_{n=0}^{\infty} q_n x^n$ este funcţia generatoare a contorului N.

Pe baza acestei recurenţe se poate face un algoritm de calcul rapid al convoluţiei F^{*n} pe care îl prezentăm în scriptul `convrap(x,p,N,epsilon)`. Ideea este că dacă $X_n \sim \begin{pmatrix} 0 & 1 & ... & k \\ p_0 & p_1 & ... & p_k \end{pmatrix}$ şi $Y_n = (X_n \mid X_n > 0) \sim \begin{pmatrix} 1 & 2 & ... & k \\ \frac{p_0}{1-p_0} & \frac{p_1}{1-p_0} & ... & \frac{p_k}{1-p_0} \end{pmatrix}$, atunci $X_1 + ...+ X_n \overset{D}{=} Y_1 + ...+ Y_N$ unde $N \sim \text{Binom}(n, 1 - p_0)$ este independent de (X_n). [12]

În cuvinte, la adunare nu contează zerourile, iar numărul variabilelor aleatoare X_k diferite de 0 este o variabilă aleatoare $N \sim \text{Binom}(n, P(X_1 > 0)) = \text{Binom}(n, 1 - p_0)$.

---

[10] Atragem atenţia că F[k] nu calculează $F(k)$, ci $F(k-1)$, deoarece în „R" indicii vectorilor încep cu 1, nu cu 0 ca în alte medii de programare (de exemplu Basic).

[11] Vezi de exemplu G. Zbăganu, *Metode Matematice în teoria riscului...., pg.* 170

[12] Într-adevăr, dacă $Y = (X \mid X > 0)$, $Ee^{tY} = E(e^{tX} \mid X > 0) = E(e^{tX}; X > 0)/(1- p_0) = (m_X(t) - p_0)/(1- p_0)$. Fie $S = X_1 + ...+ X_n$ şi $T = Y_1 + ...+ Y_N$. Atunci, cum N *este* independent de X, $Ee^T = g_N(m_Y) = (p_0 + (1 - p_0)m_Y)^n = \left(p_0 + \frac{m_X - p_0}{1-p_0}(1-p_0)\right)^n = (m_X)^n = Ee^S$. Adică $S \overset{D}{=} T$.

Deoarece $g_N(x) = (p_0 + (1-p_0)x)^n$, $f_0 = p_0^n$, $a = -\frac{1-p_0}{p_0}$, $b = \frac{(1-p_0)(n+1)}{p_0}$ iar recurenţa devine $f_k = \frac{1}{1-ap(Y=0)}\sum_{j=1}^{k}\left(a+\frac{bj}{k}\right)p(Y=j)f_{k-j} = \frac{1}{p_0}\sum_{j=1}^{k}\left(-1+\frac{(n+1)j}{k}\right)p_j f_{k-j}$, deci

$$f_0 = p_0^n,\ k \geq 1 \Rightarrow f_k = \frac{1}{kp_0}\sum_{j=1}^{k}((n+1)j-k)p_j f_{k-j} \qquad (6.3.8)$$

Vom face un script care să calculeze f_S şi F_S, probabilităţile şi funcţia de repartiţie ale lui *S*. Apar două probleme de programare:

- în principiu, cu excepţia primului caz, *S* va avea o infinitate de valori. Nici un calculator nu poate calcula aşa ceva. Deci trebuie să ne hotărîm pînă unde calculăm probabilităţile f_k. Vom alege un ε şi vom calcula f_k pînă la primul *k* la care $F(k) > 1 - \varepsilon$
- în "R", vectorii sunt numerotaţi cu indici începînd de la 1, şi nu de la 0. Trebuie mare atenţie ca să nu ne încurcăm în indici. Dacă *x,p,f* sunt gîndiţi ca vectori, atunci $f_j = f[j+1]$, $p_j := p[j+1]$ şi recurenţa devine

  $f[1] = g(p[1]),\ f[n+1] = \frac{1}{1-ap[1]}\sum_{j=1}^{n}\left(a+\frac{bj}{n}\right)p[j+1]f[n-j+1]$
- Totuşi, *f* şi F trebuie iniţializaţi ca vectori de o anumită lungime. Am ales această lungime să fie egală cu nmax= ES + 15σ(S). Inegalitatea lui Cebişev spune că $P(S > \text{nmax}) < 1/5^2 = 0.44\%$. În realitate ea este mult mai mică.

Scriptul următor face acest lucru. El este gîndit ca o funcţie Panjer(x,p,tip,N,pN,epsilon).

Mai întîi verificăm dacă datele de intrare sunt corecte: *x* trebuie să fie un vector cu numere naturale, p un vector cu numere positive de sumă 1 de aceeaşi lungime ca şi x iar pN trebuie să fie un număr cuprins între 0 şi 1.

Apoi standardizăm pe x şi *p*. Sortăm pe x în ordine crescătoare şi completăm la vectorul p locurile libere 0. De exemplu, dacă x = (5,3,6), p=(0.5, 0.2, 0.3) scriem x = (0,1,2,3,4,5,6) , *p* = (0, 0, 0, .2, 0, .5, .3)

Verificăm daca `tip` este una din literele "b", "p" sau "n". Dacă tip = "b", avem de a face cu un contor de tip Binom(*N,pN*). Nu verificăm dacă N este număr natural, e răspunderea utilizatorului. Dacă tip = "p" atunci contorul este repartizat Poisson(*N*). Nu mai verificăm pN. Dacă, în fine, tip = "n", atunci N ~ Negbin(N,pN). Este o repartiţie negative binomială generalizată, în sensul că N nu trebuie să fie neapărat număr natural.

Următorul pas este să calculăm E*S* şi Var(*S*)

Aplicînd formulele lui Wald[13] găsim

$$ES = EX \cdot EN,\ VarS = EN \cdot Var(X) + E^2N \cdot Var(N) \qquad (6.3.9)$$

Pe baza lor calculăm dimensiunea lui *f* şi *F*, anume nmax = E*S* + 15σ(*S*), sue = 1 – epsilon.

Acestea au fost pregătirile. Partea principală a scriptului este calculul recurenţei (6.3.7). Cum recurenţa (6.3.7) este liniară în raport cu f_0, preferăm să punem f[1] = 1 şi să înmulţim la sfîrşit cu

$f_0 = g(p_0)$

[13] Vezi de exemplu G. Zbăganu, op. Cit, p. 164

Scriptul este următorul:

```
##functia Panjer (x,p,"type",N,pN,epsilon) calculeaza repartitia lui S(N),
## N  este un contor Panjer(a,b)
## daca type = "p", N ~ Poisson (N);se ia in considerare doar N, pN=0
## daca type = "b" e vorba de un contor binomial de parametru (N,pN)
## daca type = "n" e contor negativ binomial Bin(N,pN)
## 1. verificam daca x,p sunt de aceeasi lungime si daca p e probabilitate
## 2. Daca nu, mesaj de eroare, daca da, calculam a,b, f0
## 3.  Calculam nmax = length(f) in asa fel incit sa evitam dimensiuni prea
mari
## nmax= int(ES + 10 sig(S))+1 cu ES = EXEN, sig^2(S)= ENVarX+VarNE^2X
## 4. Completam x ca sa fie de forma 0,1,...,(nmax-1) si p ca length(p)=nmax
## calculam f si F pina cind F*f0>(1-epsilon);
## Ca sa evitam zerourile de masina incepem cu f[1]=1
## recurenta este f[n+1]=sum(f=p[j+1]f[n+1-j](a+bj/n), 1<=j<=n

Panjer<-function(x,p,type,N,pN,epsilon)
   {x = floor(abs(x)); pN=min(pN,1)  ## x are componente nr naturale şi pN<1
   nx=length(x);np=length(p);
   if (nx!=np) {print ("Eroare: x si p au lungimi diferite");return()}
   for (i in 1:nx)
   {if (p[i]<0) {print("Eroare:p nu e probabilitate, are componente
negative");return()}
   if (abs(sum(p)-1)>10^(-9)) {print("Eroare:p nu e probabilitate, suma nu e
1");return()}
## standardizam pe x si p
o=order(x); x=x[o];p=p[o]  ## am aranjat ca x sa fie crescator
n1=max(x)+1; x1=1:n1;x1=x1-1;p1=c(rep(0,n1))
for (j in 1:nx) {p1[x[j]+1]=p[j]};sue=1-epsilon

#3 intializam a,b,f0, ES, VarS
mux=x%*%p;varx=(x^2)%*%p - mux^2;qN=1-pN
if (type=="b")
{muN = N*pN;varN=N*pN*qN; mus=mux*muN;vars=muN*varx+varN*mux^2;
   f0=(qN+pN*p1[1])^N;a=-pN/qN;b=-a*(N+1)}
   else if (type == "p")
{muN=N;varN=N;mus=mux*muN;vars=muN*varx+varN*mux^2;f0=exp(N*(p1[1]-
1));a=0;b=N}
   else if (type == "n")
   {muN=N*qN/pN;varN=N*qN/pN/pN;mus=mux*muN;vars=muN*varx+varN*mux^2
      f0=(pN/(1-qN*p1[1]))^N;a=qN;b=a*(N-1)}
   else {print ("Nu inteleg tipul")}
   }
   #4 calculam nmax si initializam f,F
nmax=mus+15*sqrt(vars);nmax=floor(nmax)+1;p=c(p1,rep(0,(nmax-n1)))
   f=c(rep(0,n1));F=f;x=x1;su=0
   ct=1/(1-a*p[1]);sue=(1-epsilon)/f0;n=1;f[1]=1;F[1]=1
      for (n in 1:nmax)
      {f[n+1]=0;n2=min(n,n1)
            for (j in 1:n2)
            { f[n+1]=f[n+1]+p[j+1]*f[n+1-j]*(a + b*j/n)}
      f[n+1]=ct*f[n+1]
      F[n+1]=F[n] + f[n+1]
      n=n+1
      if (F[n]>sue) break
      }
f=f0*f;F=f0*F
a=list(ES=mus,sigmaS=sqrt(vars),n_epsilon=n,f=f,F=F)}
```

Apelarea lui se face prin comanda

a = Panjer(x,p,type,N,pN,epsilon) (3.10)

Funcţia furnizează o listă a cu

- a[[1]] = E*S*

- a[[2]] = σ(*S*)
- a[[3]]=n_ε, unde n_ε este primul *n* cu proprietatea că $P(S \le n_\varepsilon) > 1 - \varepsilon$
- a[[4]] = f, cu f[n] = $P(S = n\text{-}1)$
- a[[5]] = *F* cu F[n] = $P(S \le n - 1)$

Revenim acum la *Exemplul 1:* $X_n \sim \text{Bin}(10, ½)$ şi $N \sim \text{Poisson}(100)$, în care ne aşteptăm să avem în jur de 100 de clienţi şi fiecare să ceară, în medie 5 lei. Dacă acceptăm un risc de 1% , ne interesează a[[3]] cu ε = .01 . Probabilităţile p_j se calculează cu ajutorul funcţiei dbinom:

```
x=0:10; p=c(rep(0,11)); for (i in 1:11) {p[i]=dbinom(i-
1,10,.5)}
[1] 0.0009765625 0.0097656250 0.0439453125 0.1171875000 0.2050781250
[6] 0.2460937500 0.2050781250 0.1171875000 0.0439453125 0.0097656250
[11] 0.0009765625
```

Comanda principală este

```
   a=Panjer(x,p,"p",100,.1,.01)
 > ES=a[[1]];ES
 [1,]  500
 > sigmaS = a[[2]]; sigmaS
 [1,] 52.44044
 > VaR_epsilon= a[[3]]; VaR_epsilon
 [1] 627
```

Deci în 99% din cazuri, suma cerută este mai mică decît 627 lei. Cu inegalitatea Cebîşev îmbunătăţită găsisem 871 lei.
Dacă dorim să vizualizăm densitatea f şi funcţia de repartiţie F putem proceda astfel:

```
x=1:nn;plot(x,f/max(f),type="l",col="blue",sub="Albastru:
densitatea lui S, rosu: functia de repartitie");
lines(F,col="red").
```

Obţinem graficul

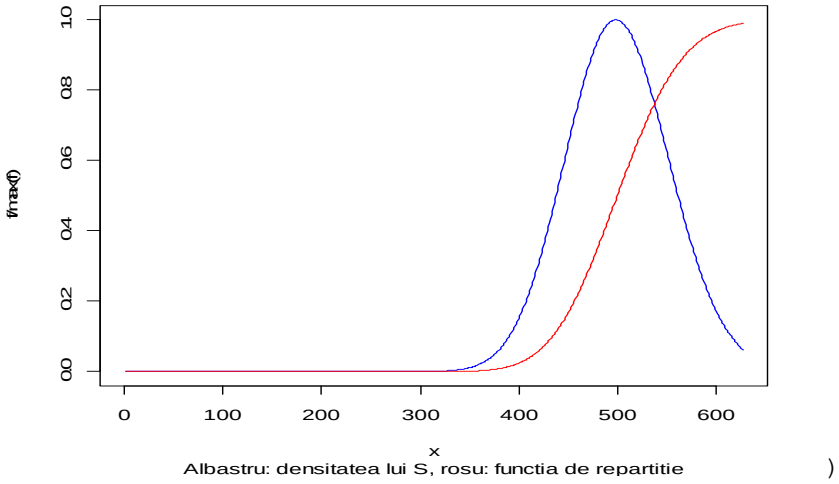


Fig 6.5

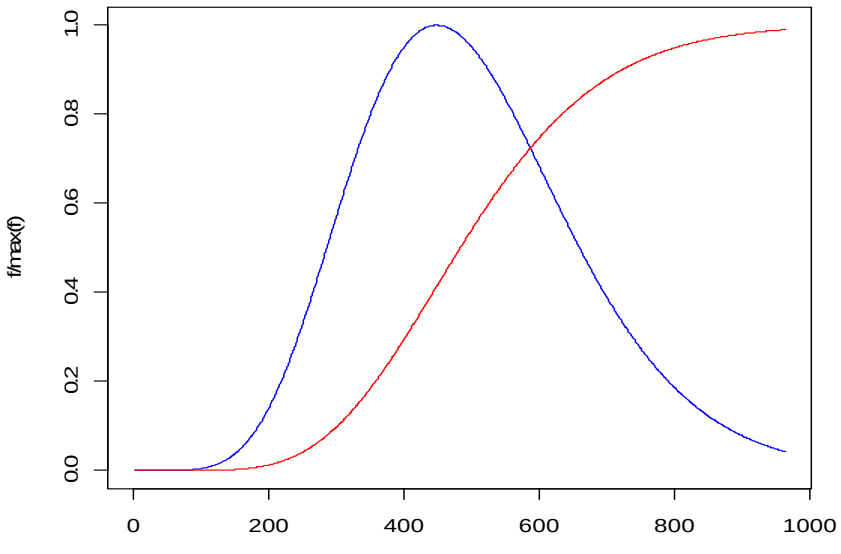
Se vede clar cum cele mai probabile valori sunt în jurul lui x=500. Dacă am fi vrut să ne asumăm riscul de 0.1% în loc de 1%, atunci valoarea la risc ar fi fost 671. La riscul de 0.01% valoarea ar fi fost VaR = 708 iar la $\varepsilon = 10^{-6}$ ar fi fost 771 lei. Valoarea de 871 lei, găsită la riscul 1% cu inegalitatea lui Cebîşev ar fi suficientă la un risc de $\varepsilon = 10^{-10}$!!

```
> a=Panjer(x,p,"p",100,1/11,.0000000001);VaR_S=a[[3]];VaR_S
[1] 872
```

Exemplul 3. Să zicem că X_n ~ Bin(10, ½) şi N ~ Negbin(10, 1/11). Şi acum ne aşteptăm să avem în jur de 100 de clienţi şi fiecare să ceară, în medie 5 lei. Dar diferenţa este destul de mare

```
a=Panjer(x,p,"n",10,1/11,.01)
ES=a[[1]];sigS=a[[2]];VaR_S=a[[3]]; f = a[[4]]; F=a[[5]]
ES = 500
sigS = 166.5833
VaR_S  = 964
```

Deşi media este aceaşi, acum abaterea medie pătratică este de trei ori mai mare ca înainte: 166.58. Valoarea la risc e normal să fie mai mare: acum este 964 lei. Graficul arată diferenţa



Contor negativ binomial.Albastru: densitatea lui S, rosu: functia de repartitie

Fig 6.6

La riscul 0.1%, suma necesară ar fi fost 1168.

Putem acum compara algoritmul de convoluţie convarep(n,x,p) (algoritmul de bun simţ) care calculează convoluţia de n ori a repartiţiei (x,p) = $\begin{pmatrix} x_1 & x_2 & \dots & x_k \\ p_1 & p_2 & \dots & p_k \end{pmatrix}$ cu

algoritmul rapid convarap(x,p,N,ε) care calculează repartiţia sumei $S = Y_1 + ...+ Y_N$ cu $N \sim \text{Bin}(n, 1 - P(X_n = 0))$, unde $Y_i = (X_i \mid X_i > 0)$ pînă valoarea la risc $\text{VaR}_{1-\varepsilon}(S)$ (adică pînă la primul n care are proprietatea că $P(S > n) < \varepsilon$.) Scriptul este următorul

```
Convarap <- function(x,p,n,epsilon)
{nx=length(x)
o=order(x);x=x[o];p=p[o] ##sortăm pe x
xmic=min(x);x1=x-xmic
## xmic devine 0, xmare devine xmare+xmic
pN=1-p[1];N=n
q=p/pN;q=q[2:nx]
y=x1[2:nx]; ## se reţin doar acei y>0
xa=Panjer(y,q,"b",N,pN,epsilon)
f=xa[[4]];F=xa[[5]];nn=length(f);x=0:(nn-1);x=x+N*xmic
b=list(x=x,f=f,F=F)
b
}
```

O primă observaţie este că algoritmul convarep permite efectuarea convoluţiilor fără nici o restricţie, pe cînd la convarap trebuie ca $x_i \geq 0$.

O a doua este că n nu poate fi prea mare şi nici $P(X = x_1)$ nu poate fi prea mic din următorul motiv: la iniţializare, $f_0 = P(X = x_1)^n$.

Să luăm cazul unui zar. $x = (1,2,3,4,5,6)$, $p = (1,1,1,1,1,1)/6$. $x_1 = 1/6$, deci $P(S = n) = 1/6^n$. Numerele în „R" sunt date în dublă precizie, adică sunt cuprinse între 10^{-63} şi 10^{63}. Dacă o să facem $n = 1000$, deja nu mai avem încredere în f_0, care devine 0. Şirul (f_n) capătă un comportament aleator. Totuşi, pentru $n = 400$, ambele scripturi sunt funcţionale, al doilea fiind de 120 de ori mai rapid ca primul.

Pentru valori mici ale lui n şi mai mari ale lui k, în mod paradoxal este mai bun algoritmul „baby". De ex. dacă n = 100, x = (1,3,5,7,11,13,17,19,23,29), p=(1,1,1,1,1,1,1,1,1,1)/10, atunci suntem mai siguri pe algoritmul naiv convarep, deşi durează 30 de secunde, decît pe cel sofisticat, care durează jumătate de secundă

```
b=convarap(x,p,100,.00000001);t1=b[[1]];f1=b[[2]]
a=convarep(100,x,p);t=a[[1]];f=a[[2]]
plot(t1,f1,type="l");lines(t,f,col="red")
```

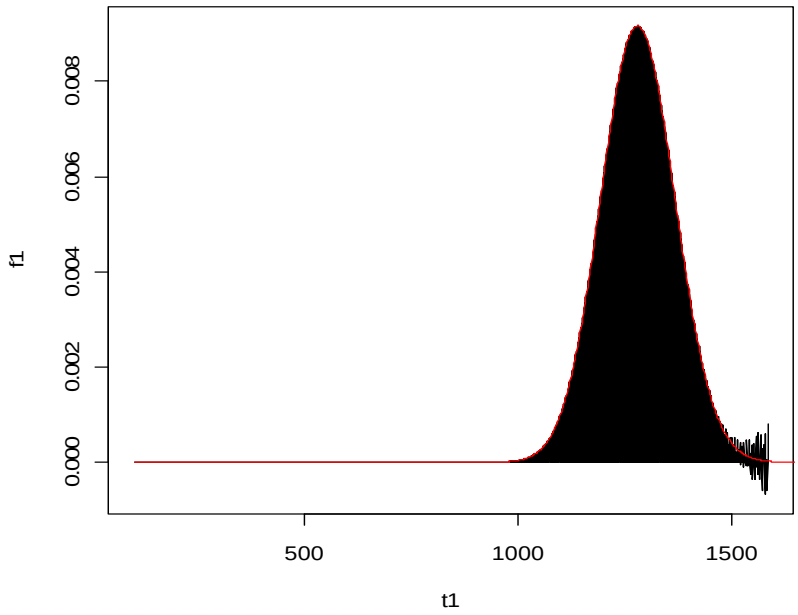


Fig 6.7

Cu negru sunt figurate densităţile f_i calculate după algoritmul rapid. Deja unele devin negative.

Există alte tehnici sofisticate de a rezolva problema convoluţiei dar nu fac obiectul lucrării de faţă.

6.4 . Probleme de demografie, asigurări de persoane, tabele de mortalitate

Un tabel de mortalitate este un tabel cu două coloane, x şi l în care x reprezintă vîrsta iar l_x este probabilitatea ca un individ născut astăzi să depăşească vîrsta de x ani, înmulţită cu 100.000. Deci neapărat l_0 = 100000. Tabelele de mortalitate se construiesc pînă la o vîrstă maximă, notată convenţional cu ω. De obicei ω = 100 sau ω = 110.

Datele brute pe baza cărora se construieşte tabelul sunt doi vectori, **n** şi **d** unde n_i este numărul total al populaţiei cu vîrstă cuprinsă în intervalul $[i,i+1)$ şi d_i numărul de decese ale populaţiei cu vîrste cuprinse între aceste limite.

De exemplu, n_0 este numărul de copii cu vîrste cuprinse între 0 şi 1 an, calculat la 1 ianuarie iar d_0 este numărul deceselor acestei categorii de populaţie, calculat în acelaşi an, dar la 31 decembrie. Sunt şi alte convenţii, dar aceasta este cea mai simplă.

Iată un **exemplu** de date brute, luat de la institutul de statistică în care avem de a face cu patru populaţii: barbaţi 1994, femei 1994, bărbaţi 2008 şi femei 2008. Exemplul este pus într-un fişier .txt cu numele de `barbatfemei19942008` care trebuie să fie în directorul de lucru, pentru a nu fi complicat de deschis cu instrucţiunea `read.table`.

x	1994				2008			
	barbati		femei		barbati		femei	
	total	decese	total	decese	total	decese	total	decese
0	127308	3303	120921	2591	110665	1416	105025	1018
1	125979	282	119583	249	109056	93	103456	77
2	131763	197	125846	132	112199	65	105912	49
3	127804	158	122161	110	110315	52	104550	37
4	175284	192	167954	124	107839	34	101622	24
5	176293	255	170667	159	106203	40	100940	37
6	182212	179	173773	140	107077	42	100304	30
7	188160	118	181410	80	109577	46	104001	28
8	169580	95	163040	54	114464	35	108485	20
9	175287	103	167986	43	113859	30	108823	17
10	157988	91	151287	44	114695	31	108680	13
...								
80	29268	3576	48002	4847	40991	4087	65403	4968
81	28345	3743	46395	4975	35451	3848	58600	4947
82	24626	3490	40820	4789	31021	3585	52227	4992
83	20438	3179	33543	4396	26196	3498	45978	4981
84	17178	2930	28381	4250	21888	3082	39356	4833
85>	57985	13476	98009	21478	66841	13225	133181	25154

De exemplu, dacă ne uităm la x= 80, vedem că la 1 ianuarie 1994 erau 29268 bărbaţi şi 48002 femei cu vîrste între 80-81 de ani, iar la 31 decembrie 1994 muriseră din ei 3576 bărbaţi şi 4847 femei. Tot în acel rînd vedem că la 1 ianuarie 2008 erau 40991 bărbaţi şi 65403 femei din care, la 31 decembrie 2008 decedaseră 4087 bărbaţi şi 4968 femei.

Se poate vedea că ultimul rînd face excepţie: aici sunt cuprinse toate persoanele cu vîrste de peste 85 de ani. La 1 ianuarie 1994 erau 57985 şi 98009 femei (din ei decedaseră la 31 decembrie 2008 un număr de 13476 bărbaţi şi 21478 femei) iar la 1 ianuarie 2008 erau 66841 bărbaţi şi 133181 femei din care la 31 decembrie decedaseră 13225 bărbaţi şi 25154 femei.

Completarea tabelului pînă la vîrsta de 85 de ani se face identificînd probabilitatea unui eveniment cu frecvenţa lui.

Mai precis: să presupunem că timpul de viaţă la naştere al unui individ este o variabilă aleatoare T. Vrem să calculăm coada sa, $\underline{F}(x) = P(T > x)$, care în demografie se notează cu ${}_xp_0$. Normal ar trebui să punem ${}_xp_0 = \dfrac{\text{total supravieţuitori cu vîrste} > x}{\text{total supravietuitori}}$, dar acest lucru este imposibil, deoarece nu ştim numărul supravieţuitorilor: ar trebui să ştim vîrsta decesului fiecărui individ....

Dacă acceptăm identificarea probabilităţii cu frecvenţa, interpretăm raportul $p_0 = d_0/n_0$ cu probabilitatea $P(T \le 1)$: probabilitatea ca un nou născut să moară înainte de a împlini un an. Atunci numărul $q_0 = 1 - p_0$ este $P(T > 1)$, adică probabilitatea ca el să supravieţuiască un an. Apoi, $p_1 = d_1 / n_1$ este probabilitatea ca un supravieţuitor de 1 an să moară în intervalul de timp (1,2] , adică înseamnă $P(T \le 2 \mid T > 1)$. Deci $q_1 = 1 - p_1$ înseamnă probabilitatea ca un individ să supravieţuiască 2 ani ştiind că el deja a supravieţuit 1 an, adică $q_1 = P(T > 2 \mid T > 1)$, adică avem $q_1 = P(T > 2)/P(T > 1) \Rightarrow P(T > 2) = q_0 q_1$. Din aproape în aproape găsim formula

$$P(T > x+1) = q_1 \ldots . q_x \qquad (6.4.1)$$

valabilă pentru x număr natural. Înmulţim acest număr cu 100000 şi obţinem $l_x = 10^5 q_x$, rotunjit la cel mai apropiat întreg. Rezultatul este valabil, în cazul exemplului nostru numai pînă la $x = 84$, deoarece nu avem date mai multe.

Functia tabmort realizează acest lucru.

```
## functia tabmort (numar,death) creeaza un vector
## cu acelasi numar de componente de elemente
tabmort<-function(numar,death)
{nv=length(numar);nm=length(death);tabel=c(rep(0,nv))
if (nv!=nm) {print ("eroare:cei doi vectori au lungimi
diferite");return()}
p=death/numar;q=1-p
for (j in 1:nv) {tabel[j]=prod(q[1:j])}
tabel=c(1,tabel)
tabel = floor(100000*tabel+.5)
tabel
}
```

Pentru a o aplica, am făcut o pregătire: am copiat în clipboard fisierul barbatfemei19942008 (cu exceptia primelor doua rînduri, care nu se potrivesc cu restul) şi am obţinut o lista cu 9 elemente, numita tabgen

```
tabgen=read.table("clipboard")
```

Din lista `tabgen` extragem cei 8 vectori `(numar,death)` cu care creăm apoi 4 tabele de mortalitate, notate cu a,b,c,d pe care le plotăm. Primul se referă la bărbaţi 1994 (linia neagră) al doilea la bărbaţi 2008 (cea roşie), al treilea la femei 1994 (albastră) şi ultimul la femei 2008 (verde)

```
> virsta=tabgen[[1]];nrbb94=tabgen[[2]]
nrbm94=tabgen[[3]];nrfm94=tabgen[[4]];nrmm94=tabgen[[5]]
> nrbb08=tabgen[[6]];nrbm08=tabgen[[7]]
nrfm08=tabgen[[8]];nrmm08=tabgen[[9]]
a=tabmort(nrbb94,nrbm94);plot(a,type="l",sub="tabel
mortalitate pe sexe 1994-2008" )
b=tabmort(nrbb08,nrbm08);lines(b,col="red")
c=tabmort(nrfm94,nrmm94);lines(c,col="blue")
d=tabmort(nrfm08,nrmm08);lines(d,col="green")
```

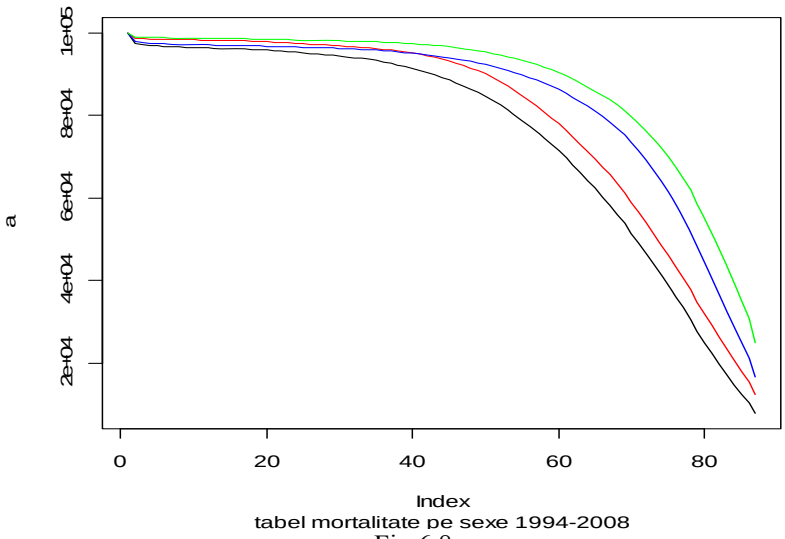


Fig 6.8

Pentru a putea completa tabelul pînă la ω = 100 sau ω = 110 trebuie să acceptăm unele ipoteze neverificabile. Una din ele ar fi că **riscul de moarte** este constant pe intervale de forma $(k,k+1]$ şi pe intervalul $(84,\infty)$ creşte liniar Alta ar fi că riscul de moarte creşte exponenţial (ipoteza lui Gompertz).

În ambele modele se ascunde o ipoteză neverificabilă, anume că repartiţia timpului mediu de viaţă la naştere, T este absolut continuă şi coada sa, $\underline{F}_T(x)$ este de forma

$$\underline{F}_T(x) = e^{-\int_0^x r(y)dy} \Leftrightarrow r(x) = -\frac{F_T'(x)}{F_T(x)} \tag{6.4.2}$$

Funcţia r se numeşte **riscul instantaneu de moarte.** Dacă acceptăm că r este o funcţie în scară, constantă pe intervale de forma $(k,k+1)$, atunci avem că ${}_{x+1}p_0 = {}_xp_0 e^{-\mu(x)}$ de unde deducem expresia

$$r_x = \ln\frac{l_x}{l_{x+1}} \Leftrightarrow e^{r_x} = \frac{l_x}{l_{x+1}} \tag{6.4.3}$$

În figura de mai jos prezentăm riscurile de moarte la cele patru populaţii: bărbaţi 1994/2008 (curba neagră şi cea roşie) şi femei 1994/2008 (curba albastră şi verde)

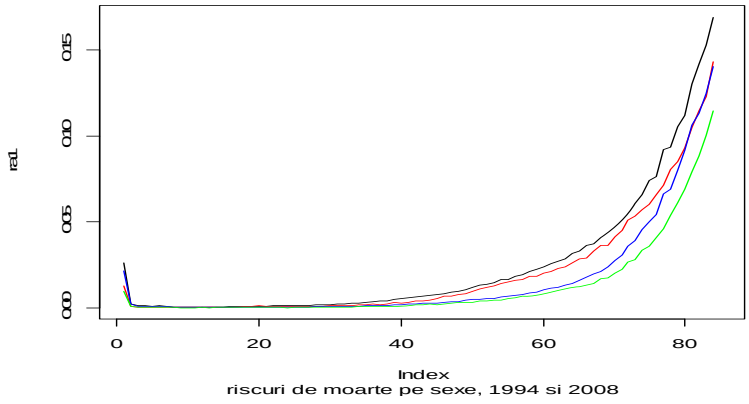


Fig. 6.9

Figura sugerează că ar fi acceptabil un model în care să se presupună că de la 85 de ani încolo, riscul de moarte r creşte liniar. Pe această ipoteză este construit scriptul `tp0(t, tab)` care calculează funcţia de supravieţuire ${}_tp_0$ pentru orice t plecînd de la un table de mortalitate, `tab`. Din tabelul de mortalitate l reţinem doar acele poziţii x în care $l_x > 10$. Atunci , din formula (4.3) deducem că

$$ {}_tp_0 = l_{[t]}\left(\frac{l_{[t]+1}}{l_{[t]}}\right)^{\{t\}} \text{ dacă } t \le \omega \text{ şi } {}_tp_0 = l_\omega q^{t-\omega}\left(\frac{q}{p}\right)^{\frac{(t-\omega)(t-\omega+1)}{2}} \text{ dacă } t > \omega \qquad (6.4.4) $$

Am notat $p = \frac{l_{\omega-1}}{l_{\omega-2}}$ şi $q = \frac{l_\omega}{l_{\omega-1}}$.

```
## Functia tp0 (t, tabel de mortalitate) calculeaza functia
## de supravietuire a unui nou nascut la momentul t in ipoteza ca
## riscul de moarte creste liniar dupa virsta nv. Ca sa evitam
impartiri cu 0, retinem din tabelul de mortalitate "tab" doar
elementele mai mari ca 10
tp0<-function(t,tab)
{nv=length(tab);ntv=n-1;k=floor(t);s=t-k
nvv=length(which(tab>10))
p=tab[nvv-1]/tab[nvv-2];q=tab[nvv]/tab[nvv-1]
r=q/p;ss=t-nvv+1
if (k<(nvv-1)) {tp0=tab[k+1]*(tab[k+2]/tab[k+1])^s}
else tp0=tab[nvv]*q^(ss)*r^((ss^2+ss)/2)
tp0=tp0/100000
#list(tp0=tp0,nv=nv,r=r,k=k)
tp0}
```

Cu ajutorul funcţiei care calculează ${}_tp_0$ pentru orice număr pozitiv, nu neapărat întreg, este uşor de calculat funcţia de supravieţuire la vîrsta x, care se notează cu ${}_tp_x$ şi se defineşte ca

$$ {}_tp_x = P(T - x > t \mid T > x) = \frac{P(T > x + t)}{P(X > x)} = \frac{{}_{x+t}p_0}{{}_xp_0} \qquad (6.4.5) $$

Scriptul este

```
## functia tpx (t,x,tabel de mortalitate) calculeaza probabilitatea
de ## supravietuire a unui individ de x ani in aceeasi ipoteza ca si
tp0
      tpx<-function(x,t,tab)
      {tpx=tp0(t+x,tab)/tp0(x,tab);tpx}
```

Variabila aleatoare cu repartiţia (4.5) se notează cu T_x : timpul rezidual de viaţă la vîrsta de x ani. Deci $T_x = (T - x \mid T > x)$. Media acestei variabile aleatoare este notată cu e_x. Aplicînd formula $ET_x = \int_0^\infty P(T_x > t)dt = \int_0^\infty {}_tp_x dt$, pe care o aproximăm cu $e_x = \sum_{x \ge 0} {}_tp_x$ formăm scriptul

```
## speranta medie de viata a unui individ in virsta de x ani
ex<-function(x,tab){ex=0
for (j in 0:120) {ex=ex+tpx(x,j,tab)}
ex}
```

În figura de mai jos este prezentat graficul funcţiei ex, respectînd culorile din celelalte două figure de mai sus: negru înseamnă “bărbaţi 1994”, roşu înseamnă “bărbaţi 2008”, albastru este “femei 1994” iar verde este “femei 2008”

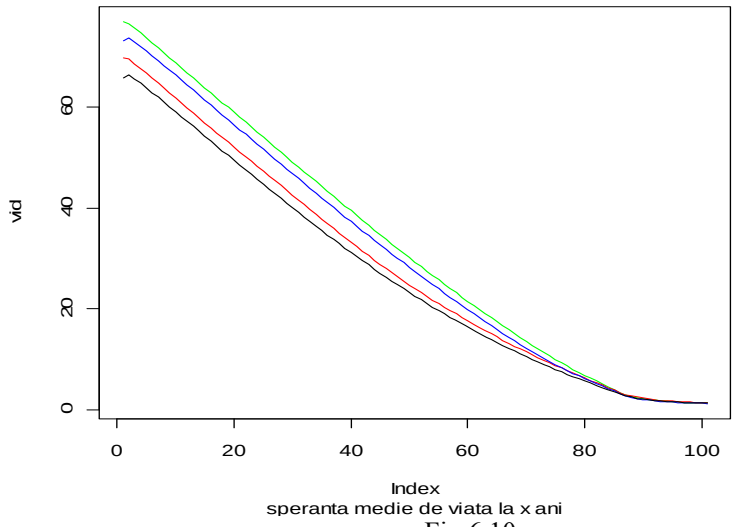


Fig 6.10

Tabelele acestea sunt importante în studiul asigurărilor de viaţă.

Iată un exemplu tipic.

Un individ de vîrstă x *vrea să îşi facă următoarea asigurare de viaţă: să primească o sumă de* S_0 *lei peste t ani dacă va fi în viaţă, iar dacă nu, urmaşii lui să primească aceeaşi sumă la momentul decesului..Pentru aceasta el constituie un depozit bancar cu rata anuală a dobînzii* i la care să aibă dreptul el (numai peste n ani) sau moştenitorii lui (la momentul T_x al decesului)

Ce sumă Π să depună în aşa fel încît probabilitatea ca suma să fie insuficientă să fie ε?

Datele problemei sunt S_0, x, n, i, ε. Formal, abordarea este următoarea: fie T_x timpul rezidual de viaţă al individului şi $\tau = \min(T_x, n)$. Dacă suma depusă este Π, atunci suma pe care o va avea el sau urmaşii lui peste τ ani va fi $S_1 = \Pi(1 + i)^{\tau}$. Condiţia pusă este ca $S_1 > S_0$ şi se cere cel mai mic Π cu proprietatea că $P(S_1 > S_0) \geq 1 - \varepsilon$.

Cum $S_1 > S_0 \Leftrightarrow (1+i)^{\tau} > \frac{S_0}{\Pi} \Leftrightarrow \tau \ln(1+i) > \ln \frac{S_0}{\Pi} \Leftrightarrow \tau > \frac{\ln \frac{S_0}{\Pi}}{\ln(1+i)}$ rezultă că suma minimă care trebuie depusă, Π va avea valoarea

$$\Pi = S_0(1 + i)^{-y} \text{ unde } y \text{ este soluţia ecuaţiei } P(\tau > y) = 1 - \varepsilon \qquad (6.4.6)$$

În concluzie, problema se reduce la calculul cuantilelor repartiţiei lui τ. Cuantila este inversa funcţiei de repartiţie. Altfel spus, este soluţia y a ecuaţiei se numeşte cuantila la nivel ε a lui τ. În cazul nostru avem

$P(\tau > y) = 1 - \varepsilon \Leftrightarrow P(\min(T_x , n) > y) = 1 - \varepsilon \Leftrightarrow P(T_x > y) = 1 - \varepsilon, y < n \Leftrightarrow {}_yp_x = 1 - \varepsilon$

Deci y este cuantila $F_{\tau}^{-1}(\varepsilon)$

```
##cuantilele lui tpx
cuant<-function(x,tab,p){p=1-
p;pas=.05;sa=seq(0,120,by=pas);pa=sa;na=length(sa)
for (j in 1:na) {pa[j]=tpx(x,sa[j],tab)}
v=which(pa>p);ja=length(v);ja1=ja+1;alfa=pa[ja];beta=pa[ja
1];tot=alfa+beta
y=beta*ja/tot+alfa*ja1/tot;y=y*pas
y}
```

Atunci suma Π care trebuie depusă în depozit pentru a garanta cu riscul ε existenţa unei sume S_0 la momentul $\tau = \min(T_x, n)$ va fi

$\Pi = S_0/(1 + i)$`cuant(x,tab,p)`

De exemplu, pentru a afla suma necesară unui bărbat de 60 de ani care doreşte ca la momentul τ să aibă în depozit suma de 10.000 RON dacă dobînda este de 5% şi folosim tabelul de mortalitate pentru 2008, scriem comenzile

```
b=tabmort(nrbb08,nrbm08);
y=cuant(60,b,.05) ;PI =1000*1.05^(-y);PI
[1] 8927.597
```

Sau,mai profesionist

```
s0=10000;i=.05;PI=s0/(1+i)^y;PI
[1] 8927.597
```

Sau, ca să arătăm cum se poate face totul cu o singură comandă

```
PI=s0/(1+i)^cuant(60,tabmort(nrbb08,nrbm08),.05);PI
[1] 8927.597
```

Dacă dobînda era 7% şi riscul de 1%, am fi avut

```
PI=s0/1.1^cuant(60,tabmort(nrbb08,nrbm08),.01);PI
[1] 9511.922
```

Dacă o femeie de 60 de ani vrea să facă acelaşi lucru, atunci am fi pus

```
PI=s0/1.1^cuant(60,tabmort(nrfm08,nrmm08),.01);PI
 [1] 8983.24
```

Se poate vedea cum contează diferenţa de vîrstă şi sex. Dacă modificăm vîrsta la 50 de ani, avem

`PI=s0/1.1^cuant(50,tabmort(nrfm08,nrmm08),.01);PI = 7936.383`
(la femei)

`PI=s0/1.1^cuant(50,tabmort(nrbb08,nrbm08),.01);PI = 9112.59`
(la bărbaţi)

Dacă vîrsta este de 5 ani, iar S_0 = 40.000 (asigurare de tip ***endowing*** – adică înzestrare) avem

```
PI=s0/1.1^cuant(5,tabmort(nrfm08,nrmm08),.01);PI = 2600.906
```
(la femei)

```
PI=s0/1.1^cuant(5,tabmort(nrbb08,nrbm08),.01);PI = 7456.137
```
(la bărbaţi)

Dacă doritorul de asigurare vrea să micşoreze costul asigurării şi să păstreze acelaşi risc ca suma să fie suficientă, atunci ar trebui să se asocieze cu alţi asiguraţi de acelaşi tip, sau să apeleze la un asigurator.

Ideea (pe care o vom explica pe primul exemplu, al bărbatului de 60 de ani) este următoare : dacă actualizăm suma pe care o depune el ca la momentul $\min(n,T_X)$ să se obţină suma S_0 = 10.000 RON este o variabilă aleatoare, anume $X = S_0/(1+i)^{\min(T_x,n)}$. Ar vrea să depună o sumă, Π cît mai mică cu putinţă ca $P(\Pi(1+i)^{\min(T_x,n)} < S_0) \le \varepsilon$, adică $P(X > \Pi) \le \varepsilon$. Dacă este singur, atunci va trebui să aleagă pe Π ca fiind cuantila de nivel $1 - \varepsilon$ a variabilei aleatoare X.

Dar dacă se asociază cu alţii, în aceeaşi situaţie şi depun cu toţii aceeaşi sumă, Π? Să presupunem că s-a format o asociaţie din N asemenea indivizi, dornici să coopereze. Să acceptăm că timpii lor reziduali, notaţi cu T_j au toţi aceeaşi repartiţie ca T_x. Să mai presupunem că T_j sunt independenţi şi că toţi indivizii noştri fac un depozit comun în care depun ***aceeaşi sumă,*** Π. Fiecare îşi retrage bani de acolo atunci cînd este nevoie. Acesta este numit în asigurări ***principiul solidarităţii.***

Fie $X_j = S_0/(1+i)^{\min(T_j,n)}$. Variabilele aleatoare X_j vor fi de asemenea independente. Probabilitatea ca depozitul să fie insuficient este $P(X_1 + \ldots + X_N > N\Pi)$. Dacă dorim ca această probabilitate să fie mai mică sau egală cu ε, va trebui să alegem Π în aşa fel încît $n\Pi$ să fie cuantila la nivel $1 - \varepsilon$ a repartiţiei sumei $S_N = X_1 + \ldots + X_N$, adică a convoluţiei F^{*N}, unde F este repartiţia lui X_j.

Este complicat de lucrat cu convoluţiile, dacă repartiţia este continuă. Dar se poate aplica inegalitatea lui Cebîşev. Să spunem că $EX_j = \mu$, $\sigma(X_j) = \sigma$. Aproximăm $E(X-\mu)_+^2$ cu $\sigma^2/2$. Atunci putem aplica inegalitatea lui Cebîşev de la (3.5) sub forma $P(S_N > ES_N + k\sigma(S_N)/\sqrt{2}) < 1/k^2$, care în cazul nostru devine $P(S_N > N\mu + k\sigma\sqrt{\frac{N}{2}}) < 1/k^2$. Pentru un risc de ε = 4% , de exemplu, alegem k = 5 şi avem $P(S_N > N\mu + 5\sigma\sqrt{\frac{N}{2}}) < .04$. Luăm Π în aşa fel încît $N\Pi = \mu + 5\sigma\sqrt{\frac{N}{2}}$, adică $\Pi = \mu + \frac{5\sigma}{\sqrt{2N}}$. Pentru aceasta avem de calculat EX şi $\sigma(X)$.

Dacă scriem $1+i = e^{\delta}$, atunci $X_j = S_0 e^{-\delta \min(T_j,n)}$, de unde $EX_j = S_0 L_n(\delta)$, $EX_j^2 = S_0^2 L_n(2\delta)$, unde

$$L_n(\delta) = E e^{-\delta \min(T_j,n)} \tag{6.4.7}$$

este transformata Laplace a variabilei aleatoare $\min(T_j,n)$. Aceasta se poate calcula foarte comod folosind formula de integrare prin părţi

$$Eg(X) = g(0) + \int_0^{\infty} g'(t)\underline{F}_X(t)dt \tag{6.4.8}$$

pentru funcţia $g(x) = e^{-\delta \min(x,n)}$. Cum $g(0) = 1$ şi $g'(x) = -\delta e^{-\delta x} 1_{(0,n)}(x)$, formula de mai sus devine

$L_n(\delta) = 1 - \delta\int_0^n e^{-\delta t}\underline{F}(t)dt$, unde $\underline{F}$ este funcţia de supravieţuire a variabilei aleatoare T_x (coada sa), care se notează cu ${}_tp_x$ şi se calculează cu ajutorul funcţiei

`tpx(x,t,tab)` cu care am făcut deja cunoştinţă. În definitiv din (4.8) deducem formulele

$$L_n(\delta) = 1 - \delta\int_0^n e^{-\delta t} \cdot {}_t p_x dt \,,\ \mathrm{E}X_j = S_0 L_n(\delta),\ \sigma(X_j) = S_0\sqrt{L_n(2\delta) - L_n{}^2\delta} \qquad (6.4.9)$$

Scriptul `Laplace(x,n,δ,tab,nrint)` calculează pe $L_n(\delta)$ cu $\delta = \ln(1+i)$ prin metoda trapezelor, cu pasul diviziunii intervalului [0,n] egal cu n/`nrint`.
Amintim semnificaţia parametrilor:

- `x` este vîrsta asiguratului;
- $\delta = \ln(1+i)$ unde i este rata anuală a dobînzii;
- `n` este durata depozitului, dacă asiguratul nu moare în această perioadă;
- `tab` este tabelul de mortalitate pe baza căruia se face calculul
- `nrint` este numărul intervalelor echidistante în care se împarte intervalul [0,n] pentru a putea calcula integrala

```
 ## Laplace(x,n,delta,tab,nrint) calculeaza integrala 1 -
#delta int(exp(-tdelta)tpx
Laplace<-function(x,n,delta,tab,nrint)
{  pas=n/nrint;s=1;
for (j in 1:nrint) {s=s+exp(-delta*j*pas)*tpx(x,j*pas,tab)}
s=s-(1+exp(-delta*n)*tpx(x,n,tab))/2; s=s*pas*delta
rez=1-s;rez}
```

iar scriptul calculează media şi varianţa variabilei aleatoare $X = X_j = S_0 e^{-\delta \min(T_j, n)}$

```
##media si varianta daca x= 60,n=20, i=.05,
medvarxni<-function(s0,x,n,i,tab)
{
delta=log(1+i);nrint=12*n;la1=Laplace(x,n,delta,tab,nrint)
   la2=Laplace(x,n,2*delta,tab,nrint)
ex=s0*la1;sigx=sqrt(la2-la1^2)
rez=list(EX=ex,SigmaX=s0*sigx)
rez   }
```

Revenim la individul de 60 de ani care doreşte 10000 RON peste 20 de ani. Calculul pe baza tabelei din 1994 ne dă

```
> medvarxni(s0,x,n,i,a)
$EX 5374.11
$SigmaX 1796.007
```

Pe baza tabelei din 2008 avem

```
medvarxni(s0,x,n,i,b)
$EX] 5188.782
$SigmaX 1738.958
```

Dacă e vorba de o femeie, atunci tabela din 1994 dă rezultatul

```
> medvarxni(s0,x,n,i,c)
$EX 4754.943
$SigmaX 1481.948
```

iar dacă folosim tabela din 2008,

```
> medvarxni(s0,x,n,i,d)
$EX 4549.405
$SigmaX 1377.932
```

Marginea $\Pi = \mu + \frac{5\sigma}{\sqrt{2N}}$, care garantează un risc de 4% de insuficienţă a depozitului devine $\Pi_N = 5374 + \frac{6350}{\sqrt{N}}$.

Comparată cu valoarea exactă, în cazul unei autoasigurări cu risc 4% care este $\Pi_0 = 9238$, ea devine deja mai mică dacă numărul participanţilor la asigurarea în grup este mai mare ca 3. Într-adevăr, $\Pi_2 = 9864$, dar $\Pi_3 = 9040 < 9238$, $\Pi_4 = 8549$,.... ,$\Pi_{100} = 6009$.

Aceleaşi rezultate se obţin şi dacă folosim tabele din 2008. Autoasigurarea realizată de

```
PI=s0/(1+i)^cuant(60,tabmort(nrbb08,nrbm08),.04);PI
```

ne dă $\Pi_0 = 9104$, iar fomula $\Pi_N = \mu + \frac{5\sigma}{\sqrt{2N}}$ ne dă $\Pi_3 = 8738 < 9104$, $\Pi_4 = 8263$,..., $\Pi_{100} = 5804$

Spre deosebire de cazul autoasigurării, fie în individual, fie în grup, în cazul în care se apelează la o companie de asigurări (care din asemenea lucruri trăieşte), la primele cu risc de forma $EX + \alpha\sigma(X)$ se mai adaugă şi un comision. Chiar şi aşa, este o soluţie preferabilă faţă de autoasigurarea individuală.

6.5. Simulări de variabile aleatoare, probleme de portofolii

După cum am văzut la capitolul anterior, limbajul „R" oferă multe repartiţii cărora li s ecalculează funcţia de repartiţie (p) densitatea (d), cuantila (q) şi se pot simula (r). Ele sunt

Repartiţia	Numele ei în R	Parametri
beta	beta	α,β
binomială	binom	*k,p*
Cauchy	cauchy	*m,a*
χ^2	chisQ	*n*
exponentială	exp	λ
F	f	*m,n*
gamma	gamma	ν,λ
geometrică	geom	p
hipergeometrică	hyper	*a*, n, *k*[14]
log-normală	lnorm	μ,σ
logistică	logis	μ,σ [15]

[14] Extragem *k* bile dintr-o urnă cu *a* bile albe şi *n* bile negre; *X* este numărul de bile albe.

[15] Repartiţia logistică are funcţia de repartiţie $\frac{1}{1+e^{-\frac{x-\mu}{\sigma}}}$. Este mai rar folosită..

negativ binomială	nbinom	*k,p*
normală	norm	μ,σ[16]
Poisson	pois	λ
Student	t	*n*
uniformă	unif	*a,b*
Weibull	weibull	*k,* λ
Wilcoxon	wilcox	m, n

Pentru a genera un vector cu *n* componente i.i.d. cu aceste repartiţii se pune în faţa numelui lor din "R" litera *r*. Apoi se pune numărul de variabile aleatoare dorite şi parametrii repartiţiei.

De exemplu:

```
x=rnorm(100,10,4);x
# x este un vector cu 100 de componente repartizate N(10,4^2)
x=rbinom(10,10,.4);x
 [1] 3 5 1 5 4 6 4 5 6 3
#  x este un vector cu 10 #componente repartizate Binomial(10,.4)
x=rnbinom(6,10,.4);x
[1] 19  8  9 25  8 19
# x  este un vector cu 10 componente repartizate Negbin(10,.4)
x=rhyper(10,4,6,6);x
 [1] 3 2 3 3 2 2 2 2 1 3
# x  este un vector cu 10 componente ~Hypergeometric(4,6,6)
```

Problemă. ***Cum se simulează o repartiţie care nu face parte dintre acestea? De exemplu, cum simulăm timpii de viaţă reziduali de viaţă*** $(T_j)_{1\le j\le n}$ ***ai unor indivizi care au aceeşi vîrstă,*** **x***?*

Dacă este vorba de repartiţii discrete, limbajul "R" punde la dispoziţie două instrucţiuni, care fac, printre altele acest lucru .

```
sample(x, size, replace = FALSE/TRUE, prob = NULL/p)
sample.int(n, size = n, replace = FALSE, prob = NULL)
```

Aici x este un vector în care se pun valorile variabilei aleatoare *X* (nu neapărat numere, pot fi şi simboluri); p este un vector de aceeaşi lungime ca şi x format cu numere positive, din care se formează automat o probabilitate, de către "R", înlocuind p[j] cu p[j]/sum(p) . Dacă nu se pune nici o probabilitate, p, se consideră că p este repartiţia uniformă, p = c(rep(1,length(x)). Dacă parametrul "replace" (sau chiar "rep", se accepta prescurtaqrea!) este TRUE, atunci instrucţiunea `sample(x,n,rep=TRUE,prob=p)` simulează un şir de n variabile aleatoare X_j $\sim \begin{pmatrix} x_1 & x_2 & .... & x_k \\ p_1 & p_2 & ... & p_k \end{pmatrix}$, cu $k =$ `length(x)`.

Exemplu. Să se simuleze un şir de 100 variabile aleatoare $X_j \sim \begin{pmatrix} a & b & c & d \\ .1 & .2 & .3 & .4 \end{pmatrix}$.

Soluţie.

```
x= c("a", "b", "c", "d"); nr=100;xsim=sample(x,nr,rep=TRUE,prob=1:4)
> xsim
```

[16] În codificarea "R", x=rnorm(1,μ,σ) produce o variabila aleatoare $X \sim N(\mu,\sigma^2)$. Parametrul al doilea reprezintă abaterea medie pătratică şi nu varianţa. Uneori se face confuzie.

```
  [1] "a" "b" "c" "d" "d" "a" "d" "d" "b" "a" "c" "a" "d" "b" "c" "d" "c"
"d"
 [19] "c" "d" "c" "d" "c" "c" "c" "b" "b" "d" "c" "d" "b" "c" "d" "c" "b"
"a"
 [37] "c" "d" "c" "a" "b" "b" "c" "a" "c" "b" "b" "a" "a" "b" "a" "c" "c"
"c"
 [55] "d" "d" "b" "d" "b" "c" "d" "d" "c" "d" "a" "d" "a" "d" "a" "c" "d"
"a"
 [73] "c" "b" "d" "d" "c" "c" "c" "c" "d" "c" "d" "a" "d" "a" "d" "b" "c"
"d"
 [91] "c" "d" "c" "a" "c" "d" "c" "b" "d" "b"
```

Ne putem convinge că este aşa folosind instrucţiunea `table`, care ne arată frecvenţele empirice ale celor patru simboluri

```
> table(xsim)
xsim
 a  b  c  d
17 18 33 32
```

Dacă simulam mai multe, se vedea mai clar cum probabilităţile sunt proportionale cu 1,2,3,4. de exemplu, dacă scriem

```
nr=10000;xsim=sample(x,nr,rep=TRUE,prob=1:4);table(xsim)
xsim
   a    b    c    d
1011 1983 3000 4006
```

acest lucru se vede mai clar.

Instrucţiunea
`sample.int(n, size = n, rep = FALSE/TRUE, prob = NULL/p)`
face cam acelaşi lucru, în cazul particular în care x = 1:k. De exemplu, dacă în loc de “a”, “b”, “c”, “d” am fi pus 1,2,3,4 , am fi putut folosi, cu acelaşi rezultat instrucţiunea

```
xsim=sample.int(4,nr,rep=TRUE,prob=1:4);table(xsim)
xsim
   1    2    3    4
1020 1997 2870 4113
```

Dar instrucţiunea `sample` face mai mult de atît: dacă `rep = FALSE`, atunci generează un aranjament aleator al componentelor lui x. Dacă `prob=NULL`, adică nu punem nici o probabilitate, toate aranjamentele au aceeaşi şansă de apariţie. Este ca şi cum am avea o urnă cu bilele x = (x_1,…,x_k) din care scoatem, succesiv şi fără înlocuire, un număr de `size` bile. Dacă parametrul `size` lipseşte, se generează o permutare aleatoare a lui x.
Exemple:

```
> x= c("a", "b", "c", "d");
nr=3;xsim=sample(x,nr,rep=FALSE); xsim
[1] "b" "c" "a"
xsim=sample(x,rep=FALSE); xsim
[1] "c" "b" "d" "a"
> xsim=sample.int(4,rep=FALSE); xsim
[1] 1 3 4 2
```

```
xsim=sample.int(4,2,rep=FALSE); xsim
[1] 3 1
```

Dacă există şi parametrul p – adică probabilitatea, atunci

```
xsim=sample(x,size,rep=FALSE,prob=p)
```

generează un aranjament $x_{\sigma(1),\ldots,}x_{\sigma(\text{size})}$ în care probabilităţile p se aplică succesiv. Mai precis, dacă $X \sim \begin{pmatrix} x_1 & x_2 & .... & x_k \\ p_1 & p_2 & \ldots & p_k \end{pmatrix}$ şi size = d ≤ k, se generează un vector aleator $V = (V_1,\ldots,V_d)$ cu repartiţia

$$P(V_1 = x_{\sigma(1)}, V_2 = x_{\sigma(2)}, \ldots, V_d = x_{\sigma(d)}) =$$

$$p_{\sigma(1)} \frac{p_{\sigma(2)}}{1 - p_{\sigma(1)}} \frac{p_{\sigma(3)}}{1 - p_{\sigma(1)} - p_{\sigma(2)}} ..... \frac{p_{\sigma(d)}}{1 - p_{\sigma(1)} - \ldots - p_{\sigma(d-1)}}$$

Exemplu. $X \sim \begin{pmatrix} 1 & 2 & 3 & 4 \\ p_1 & p_2 & p_3 & p_4 \end{pmatrix}$

Dacă d= 2, atunci $V = (V_1,V_2)$ cu $P(V_1 = i, V_2 = j) = \frac{p_i p_j}{1 - p_i}$

Dacă d= 3, atunci $V = (V_1,V_2,V_3)$ cu $P(V_1 = i_1, V_2 = i_2, V_3 = i_3) = \frac{p_{i_1} p_{i_2} p_{i_3}}{(1 - p_{i_1})(1 - p_{i_1} - p_{i_2})}$ etc.

Un caz particular: $X \sim \begin{pmatrix} 1 & 2 & 3 & 4 \\ .1 & .2 & .3 & .4 \end{pmatrix}$. Dacă d= 2 avem 12 de aranjamente, cu următoarele probabilităţi

12	13	14	21	23	24	31	32	34	41	42	43
2/90	3/90	4/90	2/80	6/80	8/80	3/70	6/70	12/70	4/60	8/60	12/60
0.0222	0.0333	0.044	0.025	0.75	0.1	0.042	0.0857	0.1714	0.0667	0.1333	0.2

Ca să ne convingem că aşa stau lucrurile, facem 10000 de simulări

```
> for (j in 1:nsim)
+ {xsim=sample.int(4,2,prob=p);vax[j]=10*xsim[1]+xsim[2]}
> table(vax)
vax
  12   13   14   21   23   24   31   32   34   41   42   43
 207  307  439  272  729  948  414  847 1753  685 1319 2080
```

În concluzie, în cazul variabilelor aleatoare unidimensionale discrete (cu o mulţime finită de valori) problema simulării este rezolvată complet prin instrucţiunea `sample`.

Vom prezenta algoritmul general de simulare al unei repartiţii care nu face parte din cele furnizate de "R": acesta este **algoritmul cuantilei**. Ideea este că dacă $U \sim U(0,1)$, atunci

$$X = F^{-1}(U) \text{ este o variabilă aleatoare cu funcţia de repartiţie } F \qquad (6.5.1)$$

Apare o problemă tehnică, aceea că inversa $F^{-1}(p)$ nu există întotdeauna. De aceea o înlocuim cu cuantila de la nivel p, definită prin

$$F^{-1}(p) = x \Leftrightarrow F(x-0) \le p,\ F(x) \ge 0 \qquad (6.5.2)$$

Metoda: găsim două valori y_1 şi y_2 cu proprietatea că $F(x_1) < p,\ F(x_2) > p$, destul de apropiate şi aproximăm funcţia F cu o funcţie de gradul I care trece prin $(x_1,F(x_1))$ şi $(x_2,F(x_2))$. Un algoritm simplu şi eficient dacă diferenţa dintre x_1 şi x_2 nu este prea mare . În loc să lucrăm cu funcţia de repartiţie, putem lucra cu coada ei, $\underline{F} = 1 - F$, dar atunci p trebuie înlocuit cu $1 - p$.

Prezentăm două scripturi: cuantinc care calculează cuantila plecînd de la un tabel $(x, F(x))$ şi scriptul cuantdec, care o calculează folosind coada ei $(x, \underline{F}(x))$. Aici x este un vector cu valori ale lui x iar y este vectorul cu valorile lui F sau $\underline{F}$.

```
cuantinc<-function(x,y,p)
{k=length(which(y<p));x1=x[k];x2=x[k+1]
rez=x[k]+(x[k+1]-x[k])*abs((y[k]-p)/(y[k]-y[k+1]))
rez}

cuantdec<-function(x,y,p)
{p= 1-p;k=length(which(y>p));x1=x[k];x2=x[k+1]
rez=x[k]+(x[k+1]-x[k])*(y[k]-p)/(y[k]-y[k+1])
rez}
```

Exemplu de aplicare: cuantila unei repartiţii Pareto(a,b,c). Coada sa este dată de $\underline{F}(x) = \left(\frac{1}{1+ax^b}\right)^c$, $x \ge 0$. Cuantila se poate calcula şi analitic, de data aceasta:

$\underline{F}(x) = p \Leftrightarrow \underline{F}(x) = 1 - p \Leftrightarrow x = \left(\frac{(1-p)^{-\frac{1}{c}} - 1}{a}\right)^{\frac{1}{b}}$. După cum se vede în scriptul de mai jos, dacă se ia un pas de .01 pentru x, atunci valoarea exactă x2 dată de formula de mai sus coincide cu valoarea aproximativă, x1, dată de funcţia cuantdec:

```
fz<-function(x,a,b,c) {y=1/(1+a*x^b)^c;y}
a=1;b=2;c=3
x=seq(0,10,by=.01);y=fz(x,a,b,c);p=.9
x1=cuantdec(x,y,p)
x2=(((1-p)^(-1/c)-1)/a)^(1/b)
x1 1.074484
x2 1.074446
```

Exemplu în care nu avem nici o formulă analitică, ci doar un tabel de mortalitate. Să se simuleze timpii reziduali de viaţă pentru un lot de 100 de bărbaţi de 60 de ani.
Soluţie. Folosim tabelul de mortalitate din 2008.

```
## O simulare: N timpi de viata reziduali, barbati de 60
#de ani
b=tabmort(nrbb08,nrbm08);x=0:600/10;y=x
for (j in 0:600) {y[j+1]=tpx(60,x[j+1],b)}
N=100;bb=1:N;for (k in 1:N) {bb[k]=cuantdec(x,y,runif(1))}
> summary(bb)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
0.2468 11.3200 18.8100 17.6100 25.4900 32.4600
```

Simularea ajută la verificare unor ipoteze. De exemplu putem să ne convingem că dacă 1000 de bărbaţi de 60 de ani se autoasigură cu 6000 de lei pentru ca la momentul decesului (sau după 20 de ani) urmasii lor (sau ei, dacă sunt în viaţă) să primească 10.000 lei, se creează un fond suficient:

```
> ## 100 de simulari de sume in depozit
> medie=1:100;for (j in 1:100){
+ N;for (k in 1:N) {bb[k]=cuantdec(x,y,runif(1))}
+ depozit=1.05^bb;depozit=6000*depozit;medie[j]=mean(depozit)
+ }
> summary(medie)
   Min.     1st Qu.   Median     Mean      3rd Qu.
Max.
   14171    14423    14552      14556     14671          15013
```

Ar putea chiar să parieze că pot primi 14000 lei!

Vectori aleatori şi portofolii

Pentru simularea vectorilor aleatori nu există algoritmi generali eficienţi.

Totuşi, există un algoritm rapid de generare a unui vector aleator repartizat uniform în interiorul mulţimii $\Delta_{k,S} = \{\mathbf{x} \in \Re_+^k \mid x_1 + ... + x_k = S\}$. Este mulţimea vectorilor de probabilitate n – dimensionali, care se mai numesc şi ***portofolii.***

Motivul este următorul: la bursă există mai multe active financiare ($A_1,...,A_k$) împărţite în acţiuni. După cum se ştie, preţul acţiunilor este aleator.

Să presupunem că dacă se investeşte 1 leu în activul A_j rentabilitatea este o sumă de bani R_j. Mai precis: dacă un jucător la bursă cumpără acţiuni de 1 leu din activul A_j la momentul t_0 , şi vrea să le vîndă la momentul t_1, el nu le va vinde tot cu 1 leu, ci cu o sumă X_j, depinzînd de cotaţia de pe piaţa financiară a activului A_j . Diferenţa $R_j = X_j - 1$ se numeşte ***rentabilitatea*** operaţiunii financiare. Sigur că ea depinde de momentele t_0 şi t_1. Din definiţie, variabilele aleatoare $R_j$sunt întotdeauna mai mari sau egale cu 1.

Dacă în loc să cumpere acţiuni în valoare de 1 leu dintr-un singur activ financiar A_j jucătorul de bursă cumpără de x_1 lei acţiuni din activul A_1, de x_2 lei acţiuni din activul A_2,.... atunci rentabilitatea operaţiunii va fi $R(\mathbf{x}) = x_1R_1 + x_2R_2 +.....+ x_kR_k$. Putem întotdeauna să considerăm suma $S = x_1 + ...+ x_k$ ca fiind egală cu 1, deoarece este clar că $R(S\mathbf{x}) = SR(\mathbf{x})$.

Un portofoliu în sens strict este atunci format din ***vectorul alocărilor*** $\mathbf{x} = (x_1,...,x_k) \in \Delta_{k,S}$ al banilor ***alocaţi*** fiecărui activ şi din ***vectorul rentabilităţilor*** $\mathbf{R} = (R_1,...,R_k)$. ***Valoarea portofoliului*** este $R(\mathbf{x}) = x_1R_1 + x_2R_2 +.....+ x_kR_k$, media $ER(x)$ se numeşte ***randamentul portofoliului*** iar varianţa $Var(R(x))$ este ***riscul portofoliului.*** Fie $\mu_j = ER_j$ şi $c_{i,j} = Cov(R_i,R_j) = EX_iX_j - \mu_i\mu_j$. Atunci

$$ER(\mathbf{x}) = x_1\mu_1 + x_2\mu_2 +.....+ x_k\mu_k := \mathbf{x}'\mu,\ Var(R(x)) = \sum_{i,j} c_{i,j}x_ix_j = \mathbf{x'Cx} \qquad (6.5.3)$$

Am notat cu $\mathbf{x}$' transpusul lui $\mathbf{x}$, adică vectorul linie cu aceleaşi componente.

Problema portofoliului optim este atunci următoarea: ***să se găsească cel mai bun portofoliu***. Doar că aşa fiind pusă, nu are sens.

Ca două portofolii să fie comparabile, ar trebui să se construiască din aceeaşi sumă *S*. Ne-ar trebui un criteriu de optim. Unul din ele este, care are sens, este: ***să se găsească portofoliul de randament maxim şi risc minim.*** Numai că aceasta de obicei nu are soluţii.

Alte idei sunt: dintre toate portofoliile cu un randament acceptabil, să se găsească cel de risc minim; sau, dintre cele de un risc acceptabil, să se găsească cele de randament maxim.

O metodă care permite o abordare generală este ***principiul utilităţii medii*** [17]**.** O utilitate este o funcţie u: $I \to \Re$ continuă strict crescătoare, unde I este un interval de numere reale. Utilitatea medie a unui portofoliu (**x** , **R**) este Eu(**x**'**R**) . Sigur că pentru ca formula să aibă sens trebuie ca **x'R** $\in I \; \forall$ **x** $\in \Delta_{k,S}$. Atunci problema capătă un sens precis

$$\textbf{\textit{Să se găsească}}\ \mathbf{x} \in \Delta_{k,S}\ \textbf{\textit{astfel ca}}\ \mathrm{E}u(\mathbf{x}'\mathbf{R}) = \text{maxim} \qquad (6.5.4)$$

O metodă rapidă de rezolvare, cu o precizie acceptabilă a problemei (5.4) este metoda Monte Carlo: se simulează N portofolii uniform repartizate în $\Delta_{k,S}$ şi se alege acela pentru care utilitatea medie este maximă.

Algoritm de generare al unui vector repartizat uniform în $\Delta_{k,S}$.

Se simulează k variabile aleatoare independente repartizate Exp(1) şi se împarte la suma lor, după care se înmulţeşte cu S.

Exemplu : S = 100, k = 5.

```
> s=100;k=5; x=rexp(k,1); x=x/sum(x)*s;x
[1]  3.154435 20.061981 38.808900 12.395507 25.579176
```

*Exemplu de calcul****. Rentabilităţi normal repartizate, utilitate CARA.***

Presupunem că vectorul R este repartizat normal N(μ,C). În practică se acceptă, deşi este clar că nu există variabile aleatoare normale repartizate cu valori în $[-1,\infty)^k$.

Luăm utilitatea $u(t) = -e^{-rt}$. O utilitate ee această formă se numeşte CARA cu coeficient de aversiune la risc egal cu r.

Se ştie că **x**'**R** ~ N(x'μ, $\sqrt{\mathbf{x}'\mathbf{C}\mathbf{x}}^{\,2}$), Cum funcţia generatoare de momente a unei repartiţii N(μ,σ^2) este $\boldsymbol{m}(t) = e^{t\mu+\frac{t^2\sigma^2}{2}}$ avem că Eu(x'R) = $-e^{-rx'\mu+\frac{r^2x'Cx}{2}}$. Aşadar problema (6.5.4) devine

$$\textit{Să se găsească}\ \mathrm{x} \in \Delta_{k,S}\ \textit{astfel ca}\ -e^{-rx'\mu+\frac{r^2x'Cx}{2}} = \max \Leftrightarrow \mathbf{x}'\boldsymbol{\mu} - \frac{r}{2}\mathbf{x}'\mathbf{C}\mathbf{x} = \max \qquad (6.5.5)$$

Continuăm exemplul în care S= 100. Fie μ = (0, .1, .2, .2,.3) şi C = AA', unde alegem A o matrice aleatoare 5×5.

```
> a=floor(10*(runif(25)-.5));dim(a)=c(5,5);C=a%*%t(a);
C=C/10;C
     [,1] [,2] [,3] [,4] [,5]
[1,]  3.9  0.2 -2.4  0.6 -1.8
```

[17] Vezi, de exemplu Gheorghiţa Zbăganu, *Metode matematice în teoria riscului şi actuariat,* Editura Universităţii Bucureşti, 2004

```
[2,]  0.2  4.7  1.6 -0.4  2.3                                    (6.5.6)
[3,] -2.4  1.6  2.3 -0.6  2.3
[4,]  0.6 -0.4 -0.6  0.8 -0.6
[5,] -1.8  2.3  2.3 -0.6  3.1
> eigen(C)
$values
[1] 8.79649765 4.63101195 0.66399316 0.64015618 0.06834105
```

Instrucţiunea eigen(C) am pus-o ca să ne convingem ca C este o matrice pozitiv definită. Coeficienţii de corelaţie între variabile sunt

```
> for (i in 1:5) {for (j in 1:5)
{ro[i,j]=C[i,j]/sqrt(C[i,i]*C[j,j])}}
> ro
            [,1]        [,2]       [,3]       [,4]       [,5]
[1,]  1.00000000  0.04671418 -0.8013367  0.3396831 -0.5176776
[2,]  0.04671418  1.00000000  0.4866393 -0.2062842  0.6025569
[3,] -0.80133668  0.48663925  1.0000000 -0.4423259  0.8613568
[4,]  0.33968311 -0.20628425 -0.4423259  1.0000000 -0.3810004
[5,] -0.51767758  0.60255689  0.8613568 -0.3810004  1.0000000
```

Funcţia portopt calculează portofoliul optim prin metoda Monte Carlo. Parametrii sunt

- S = suma investită (S = 100)
- r = coeficientul de aversiune la risc. Dacă r = 0, brokerul este neutru la risc şi portofoliul va consta în a pune toţi banii pe activul de randament maxim. Dacă r este mare, banii se vor aloca în aşa fel încît riscul să fie minim.
- mu este media lui R: un vector cu componente mai mari ca -1
- C este matricea de covarianţă
- N este numărul de simulări

```
## Maximul functiei m'x - rx'Cx/2 cu r coeficientul de
aversiune la risc
portopt<-function(S,r,mu,C,N)
   { k=length(mu);xx=matrix(nrow=N,ncol=k);val=c(rep(0,N))
   for (j in 1:N)
         {x=rexp(k,1);x=S*x/sum(x);xx[j,]=x
         val[j]=mu%*%x-r*x%*%C%*%x/2
         }
   valmax=max(val);j=which(val==valmax)
      x = xx[j,];x}
```

Luăm

```
mu =(0.35  0.35  0.20 0.20 0.30)
```

şi **C** matricea de la 6.5.6. Dispersiile Var(R_i) sunt elementele de pe diagonală ale lui C:

```
sig = (3.9  4.7  2.3   0.8  3.1)
```

Am ales numerele în aşa fel încît activele cele mai rentabile să fie şi cele mai riscante iar cel mai sigur să fie R_4 . Activul R_5 are şi randament acceptabil şi risc acceptabil: ar trebui luat în calcul dacă există aversiune la risc.

```
r=0;xoptim=portopt(S,r,mu,C,N);xoptim
     x = 70.61 26.60  0.66  1.17  0.96
```

Dacă făceam calculul exact, ne aşteptam ca toţi cei 100 de lei să fie investiţi în A_1 şi A_2. Algoritmul Monte Carlo ne–a spus să punem `70.61 + 26.60 = 97.21` lei în A_1 şi A_2.
Să luăm acum un coeficient moderat de risc,

```
r=.1; xoptim =portopt(S,r,mu,C,N); xoptim
      xoptim = 31.07  0.38 44.04 24.46  0.06
```

Deja se vede cum activul A_2 este evitat: are acelaşi randament ca A_1 dar este mai riscant.
Mai experimentăm

```
r=1    ⇒ xoptim = 22.08  0.58 42.83 33.69  0.82
r=100  ⇒ xoptim = 27.31  0.10 42.38 28.99  1.21
```

Dacă r este mare, atunci maximul funcţiei $\mathbf{x'\mu} - \frac{r}{2}\mathbf{x'Cx}$ se atinge cam în acelaşi punct în care **x'Cx** este minim. Minimul lui **x'Cx** (1652.5)se atinge în

```
x = 24.8   0   41.82 33.39    0
```

punct care nu diferă mult de cel de la r = 100.

Putem calcula şi cuantilele valorii $\xi = x'R$ a portofoliului optim. În cazul studiat, cînd repartiţia este normală, ştim că $\xi \sim N(\mathbf{x}'\mu, \sqrt{\mathbf{x'Cx}})$. De exemplu, dacă r = .1, probabilitatea de a fi în pierdere cu portofoliul optim alcătuit se poate calcula prin comenzile

```
> r=1;
x=portopt(S,r,mu,C,N);mux=x%*%mu;sig=sqrt(x%*%C%*%x);
pnorm(0,mux,sig)
[1] 0.2820430
> mux =  24.07753,  sig =  42.05184
```

Este o probabilitate mare, fiindcă şi varianţa portofoliului este mare. Dacă înlocuiam C cu C/10 am fi obţinut

```
> r=1;
x=portopt(S,r,mu,C,N);mux=x%*%mu;sig=sqrt(x%*%(C/10)%*%x);
pnorm(0,mux,sig)
p= 0.0390437;  mux = 24.62120;  sig = 13.72716
```

Dacă schimbăm funcţia de utilitate cu alta – de exemplu cu $u(x) = \sqrt{(x+100)_+}$ atunci nu am mai fi putut aplica metoda de mai sus fiindcă nu există formule analitice cu care să putem calcula $E\sqrt{(X+100)_+}$ dacă $X \sim N(\mu,\sigma^2)$. Atunci ar fi trebuit să înlocuim media cu media aritmetică a unui eşantion de un anumit volum dintr-o populaţie repartizată $X \sim N(\mu,\sigma^2)$. Se poate simula uşor în „R”. Dar dacă vrem să simulăm un vector n_d dimensional, $X \sim N(\mu,C)$, pachetul de bază nu ne ajută. Sigur că există în R şi scripturi gata făcute în acest scop, dar nu în pachetul de bază. Programarea este simplă: trebuie scris $X = AY + \mu$, unde $AA' = C$ şi
`Y =rnorm(`n_d`,0,1)` este un vector cu n_d componente independente repartizate N(0,1).

```
## simularea unei repartitii normale N(mu,C)
## N este numarul de simulari,mu este media, un vector cu nd
## componente, C este o matrice de corelatie ndxnd

simnorm<-function(N,mu,C)
{nd=length(mu);nd1=length(C[1,]);nd2=length(C[,1])
if (nd != nd1) {print ("eroare1, dimensiuni gresite");return()}
else if (nd != nd2) {print ("eroare2, dimensiuni
gresite");return()}
Ov=eigen(C);v=Ov[[1]];O = Ov[[2]]
D=O-O; for (j in 1:nd) {D[j,j]=v[j]}
DD=sqrt(D);A=O%*%DD%*%t(O)
xx=matrix(nrow=N,ncol=nd)
for (j in 1:N) {xx[j,]=A%*%rnorm(nd,0,1)+mu}
## fac si proba empirica
mup=mu;CE=C
for (j in 1:nd) {mup[j]=mean(xx[,j])}
for (i in 1:nd) {for (j in 1:nd) {CE[i,j]=cov(xx[,i],xx[,j])}}
rez=list(xx=xx,muemp=mup, covemp=CE)
rez}
```

Scriptul nu numai că simulează N vectori repartizaţi N(μ,C), dar le calculează apoi media şi covarianţa empirică. Din legea numerelor mari, ştim că ele nu trebuie să difere prea mult de EX şi Cov(X). Iată rezultatul pentru N = 10000, μ = (0.35 0.35 0.20 0.20 0.30) şi C matricea de la (5.6).

```
N=10000;xx=simnorm(N,mu,C);muemp=xx[[2]];covemp=xx[[3]]
> muemp = 0.3705928 0.3555581 0.1921526 0.2123407 0.2905003
> covemp
           [,1]       [,2]       [,3]       [,4]       [,5]
[1,]  3.9503142  0.1655118 -2.4350401  0.6147029 -1.8183625
[2,]  0.1655118  4.6166988  1.5967822 -0.3969151  2.2854434
[3,] -2.4350401  1.5967822  2.3158990 -0.6054019  2.3049773
[4,]  0.6147029 -0.3969151 -0.6054019  0.7873231 -0.6095916
[5,] -1.8183625  2.2854434  2.3049773 -0.6095916  3.0935673
```

Dacă dorim să vedem cum se comportă componentele lui *X*, putem folosi funcţia summary:

```
> xsim=xx[[1]]
> for (j in 1:nd)  {print(summary(xsim[,j]))}
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-7.5530 -0.9729  0.3854  0.3706  1.6930  7.8400
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-7.8290 -1.0810  0.3550  0.3556  1.7980  9.7950
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-5.9630 -0.8188  0.1742  0.1922  1.1900  5.4900
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.8190 -0.3803  0.2119  0.2123  0.8116  3.5680
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-5.9740 -0.8904  0.2826  0.2905  1.4550  6.9090
```

Simularea poate fi folosită atunci cînd avem de calculat cantităţi pentru care nu avem o formulă analitică. De exemplu, dacă vrem să calculăm portofoliul optim $x_o \in \Delta_{nd,S}$ în aşa fel încît $E\sqrt{(\mathbf{x_o}'\mathbf{R}+10n_d)_+}$ să fie maximă, va trebui să simulăm un număr de *N* valori ale vectorului rentabilităţilor, *R* şi un număr de N_X valori de portofolii uniform repartizate din care să îl alegem cel pentru care media empirică este maximă.

Capitolul 7

Aplicaţii rezolvate prin funcţii predefinite în R

7.1 Teste statistice

Reamintim că orice presupunere privind repartiţia necunoscută a unei variabile aleatoare poartă numele de *ipoteză statistică*, iar metodele de verificare a ipotezelor statistice poartă numele de *teste statistice*.

Ipoteza care se verifică se numeşte *ipoteza nulă* (notată de obicei cu H_0) şi orice altă ipoteză admisibilă se numeşte *ipoteză alternativă* (notată de obicei cu H_a).

Verificarea ipotezei nule se face pe baza unei selecţii (eşantion) de volum n, $x_1,..., x_n$, dintr-o populaţie statistică.

Testele statistice, după scopul lor, se pot clasifica în:

- teste de *comparare* a unor parametri ai populaţiei, ce verifică ipoteze precum compararea mediilor a două populaţii, compararea mediilor mai multor populaţii, compararea dispersiilor, etc;
- teste de *omogenitate* sau *de independenţă* care verifică ipoteze de tipul dependenţei sau independenţei unor factori de clasificare;
- testele de *concordanţă*, care verifică dacă distribuţia selecţiei este conforma cu o anumită distribuţie teoretică aşa cum ar fi distribuţia normală.

Testele de comparare se împart la rândul lor în două categorii fundamentale:

- teste *parametrice*, pentru care se presupune că populaţiile din care provin eşantioanele au distribuţii cunoscute cu cel puţin un parametru necunoscut;
- teste *neparametrice*, pentru care nu se face nicio presupunere despre distribuţiile populaţiilor din care provin eşantioanele.

Testele parametrice (fie δ un parametru al distribuţiei populaţiei) pot fi [7]:

- teste *unilaterale* (direcţionale) în care H_0: $\theta = \theta_0$ şi H_1: $\theta < \theta_0$ sau $\theta > \theta_0$.
- teste *bilaterale* (nedirecţionale) în care H_0: $\theta = \theta_0$ şi H_1: $\delta \neq \theta_0$;

La testele parametrice, dacă valoarea parametrului care apare în ipoteza alternativă este unică, atunci ea se numeşte ipoteză simplă, altfel se numeşte ipoteză compusă.

Zona de acceptare a unei ipoteze, numită şi *interval de încredere*, este un interval în care se acceptă, printr-un test statistic, ipoteza nulă, căreia i se asociază probabilitatea 1-α. *Zona de respingere* este intervalul dintr-o distribuţie de selecţie a unei statistici considerate, în care se respinge ipoteza nulă, căreia i se asociază o probabilitate α. Probabilitatea α este numită *prag de semnificaţie a testului*.

Se pot produce erori de acceptare sau de respingere pe nedrept a unei ipoteze, numite erori de primă spetă sau erori de tipul I şi de-a doua speţă sau erori de tipul II. *Eroarea de tipul I* este eroarea în care se respinge pe ipoteza nulă când în realitate ea este adevărată, probabilitatea asociată fiind α. Valoarea α se alege de obicei ca având valoarea 0,05. *Eroarea de tipul II* este cea în care se acceptă ipoteza nul atunci când ea este de fapt falsă, probabilitatea asociată fiind β. *Regiunea critică* (de respingere) a testului reprezintă valorile numerice ale testului statistic pentru care ipoteza nulă va fi respinsă. Nivelul de încredere al unui test este 1–α, iar puterea testului este 1–β.

Un test statistic, în general, parcurge următorii paşi [7]:

1. Se formulază ipotezele statistice: o ipoteză nulă H_0 şi o ipoteză alternativă H_1.
2. Pe baza selecţiei se calculează o statistică numită statistica testului.
3. Se alege un prag de semnificaţie pentru testul statistic.
4. Se compară valoarea actuală a statisticii testului cu valoarea teoretică.
5. Se stabilesc regulile de decizie de acceptare sau respingere a ipotezei nule.

Există funcţii R predefinite, precum **z.test** care se găseşte în pachetul de programe R **TechingDemos**. Acest pachet de funcţii R se poate găsi ca arhivă zip la adresa web

http://cran.r-project.org/web/packages/TeachingDemos/index.html

După descărcarea pe calculatorul personal a pachetului, pentru instalare se poate folosi opţiunea „*Install package(s) from local zip files...*" din meniul *Packages*. După instalarea cu succes, se poate alege opţiunea „Load package...", din acelaşi meniu pentru încarcarea pachetului TechingDemos.

Există funcţii R predefinite, precum **sign.test** care se găseşte în pachetul de programe **BSDA** (Basic Statistics and Data Analysis). Acest pachet de funcţii R se poate găsi ca arhivă zip la adresa web

http://cran.r-project.org/web/packages/BSDA/index.html

Se descarcă, se instalează şi se încarcă pachetul BSDA ca în cazul pachetului TechingDemos.

Alte funcţii referitoare la teste statistice se mai găsesc în pachetul standard „stats" care se instalează automat la instalarea software-ului R.

7.1.1 Teste statistice pentru un singur eşantion

7.1.1.1 Testul *Z* pentru un singur eşantion

Testul de concordanţă Z (sau testul normal) verifică o **ipoteză referitoare la media unei populaţii repartizate normal, cu dispersia cunoscută**.

Se consideră selecţia (eşantionul) X_1, X_2, ..., X_n de variabile aleatoare independente şi identic repartizate, de repartiţie $N(m, \sigma^2)$, cu σ cunoscut. Testul Z se aplică pentru eşantione este de dimensiune $n \geq 30$.

Ipoteza nulă este

$$H_0: m = m_0,$$

iar ipoteza alternativă

$$H_a: m \neq m_0.$$

Statistica Z se calculează după formula

$$Z = \frac{\bar{x} - m_0}{\sigma / \sqrt{n}},$$

unde $\bar{x}$ reprezintă media de selecţie a eşantionului:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

Ca urmare a teoremei limită centrală, statistica Z are repartiţia normală standard $N(0,1)$ pentru n mare. Avem în acest caz, dacă vom alege un risc α, ipotezele şi criteriile de acceptare sau respingere conform cu tabelului următor:

H_0	H_a	Regiunea de respingere
$m = m_0$	$m \neq m_0$	$\lvert z \rvert \rangle z_{1-\frac{\alpha}{2}}$
$m = m_0$	$m > m_0$	$z \rangle z_{1-\alpha}$
$m = m_0$	$m < m_0$	$z \langle -z_{1-\alpha}$

Aplicaţie: Să presupunem că avem 10 voluntari care au făcut un test de inteligenţă. Media obţinută la acelaşi test, referitor la întreaga populaţie este 75. Vrem să verificăm dacă există o diferenţă statistică semnificativă (cu nivel de semnificaţie de 5%) între media eşantionului cu valorile 65, 78, 88, 55, 48, 95, 66, 57, 79, 81 la testul de inteligenţă şi media populaţiei, presupunând că dispersia este cunoscută şi egală cu 18.
Soluţie: Pentru a rezolva această aplicaţie, se poate folosi testul *Z* cu un eşantion.

În pachetul de funcţii R TechingDemos există funcţia predefinită `z.test`, cu prototipul:

```
z.test(x, mu = 0, stdev, alternative = c("two.sided", "less",
          "greater"), sd = stdev, conf.level = 0.95)
```

unde
- `x` = vectorul valorilor din eşantionul considerat
- `mu` = media teoretică populaţiei
- `stdev` = dispersia cunoscută a polulaţiei
- `alternative` = direcţia ipotezei alternative, valoarea implicită fiind “two.sided” (bilateral)
- `sd` = alternativa la stdev
- `conf.level` = intervalul de încredere

În cazul nostru, o secvenţă de cod R necesară pentru rezolvarea problemei este următoarea:

```
> x = c(65, 78, 88, 55, 48, 95, 66, 57, 79, 81)
> z.test (x,75,18)    #media=75, dispersia=18

        One Sample z-test

data:  x
z = -0.6676, n = 10.000, Std. Dev. = 18.000, Std.
Dev. of the sample
mean = 5.692, p-value = 0.5044
alternative hypothesis: true mean is not equal to 75
95 percent confidence interval:
 60.04369 82.35631
sample estimates:
mean of x
     71.2
```

Astfel sunt calculate: valoarea statisticii care este –0,6676, intervalul de încredere pentru media vectorului *x*, etc. Acum putem compara valoarea statisticii testului cu valoarea distribuţiei student cu 10-1=9 grade de libertate determinată din tabele sau se poate folosi funcţia R pentru determinarea acestei valori:

```
qnorm(p, ...)
```

care calculează valoarea teoretică a statisticii *t* cu pragul de semnificaţie al testului `p`. În cazul nostru:

```
> qnorm(0.975)   #Z_(1-alpha/2)
[1] 1.959964
```

Valoarea calculată a statisticii *Z* este egală cu –0.6676 şi în valoare absolută este mai mică decât valoarea teoretică 1.959964 pentru α = 0.05. Aşadar, se acceptă ipoteza nulă a testului şi astfel se decide că media eşantionului este similară cu media întregii populaţii.

Alternativ, se poate folosi *p*-valoarea calculată. ***p*-valoarea unui test statistic** este cea mai mică valoare a nivelului de semnificaţie pentru care informaţia extrasă din eşantion este semnificativă (H_0 adevărată se respinge). Cu un prag de semnificaţie al testului de 5%, se poate aplica regula: dacă valorea *p* este mai mare decât 0.05 atunci se acceptă ipoteza nulă H_0, iar dacă este mai mică decât 0.05 atunci se respinge ipoteza nulă H_0 în favoarea ipotezei alternative H_1. În cazul nostru, *p*-valoarea este 0.5044 > 0.05 şi deci se acceptă ipoteza nulă H_0.

7.1.1.2 Testul *t* pentru un singur eşantion

Testul *t* (sau testul Student) pentru un singur eşantion se foloseşte pentru a verifica o **ipoteză referitoare la media unei populaţii repartizate normal, cu dispersia de data aceasta necunoscută**.

Se consideră tot o selecţie $X_1, X_2, ..., X_n$ de variabile aleatoare independente şi identic repartizate, de repartiţie $N(m, \sigma^2)$, cu σ însă necunoscut.

Ipoteza nulă este

$H_0: m = m_0$,

iar ipoteza alternativă

$H_a: m \neq m_0$.

Dispersia populaţiei din care provine selecţia poate fi estimată prin estimatorul deplasat $S^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$ sau prin estimatorul nedeplasat $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$, unde $\bar{x}$ reprezintă media de selecţie a eşantionului, $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

Dacă volumul selecţiei este mare ($n \geq 30$) atunci $s^2 \approx S^2$ şi se poate folosi statistica Z. Dacă însă volumul selecţiei este mic ($n < 30$), se poate folosi testul Student în care se calculează statistica testului următoare:

$$t = \frac{\bar{x} - m_0}{s/\sqrt{n}}$$

Această statistica a testului *t* este repartizată Student cu *n*-1 grade de libertate. În rest se procedează ca la testul *Z* prezentat anterior.

În pachetul de funcţii R „stats” există funcţia predefinită `t.test`, cu prototipul:

```
t.test(x, y = NULL, alternative = c("two.sided", "less",
 "greater"), mu = 0, paired = FALSE, var.equal = FALSE,
                  conf.level = 0.95)
```

unde

- `x` = vectorul valorilor din primul eşantion
- `y` = al doilea vector (opţional) de valori folosit pentru teste cu două eşantioane
- `alternative` = direcţia ipotezei alternative, valoarea implicită fiind "two.sided" (bilateral)
- `mu` = media teoretică populaţiei
- `paired` = indicator logic pentru un t-test pereche
- var.equal = variabilă logică ce indică dacă cele două varianţe sunt egale sau nu
- `conf.level` = intervalul de încredere

Considerând din nou aplicaţia de la testul anterior, de data aceasta cu dispersie necunoscută, întâi să calculăm statisticele clasice:

```
> mean(x)  #media datelor din vectorul x
[1] 71.2
> sd(x)    #deviatia stantard a datelor din vect. x
[1] 15.34637
```

Acum, o secvenţă de cod R care rezolvă aplicaţia considerată este următoarea:

```
> x = c(65, 78, 88, 55, 48, 95, 66, 57, 79, 81)
> t.test (x, mu=75)

        One Sample t-test

data:  x
t = -0.783, df = 9, p-value = 0.4537
alternative hypothesis: true mean is not equal to 75
95 percent confidence interval:
 60.22187 82.17813
sample estimates:
mean of x
     71.2
```

Astfel sunt calculate: valoarea statisticii care este –0.783, numărul gradelor de libertate care este 10-1=9, intervalul de încredere pentru media vectorului *x*, etc. Acum putem compara valoarea statisticii testului cu valoarea distribuţiei student cu 9 grade de libertate determinată fie din tabele (Tabelul 2 de la Anexa 1), fie se poate folosi funcţia R pentru determinarea acestei valori:

```
qt(p, df, ...)
```

care calculează valoarea teoretică a statisticii *t* cu pragul de semnificaţie al testului `p` şi numărul gradelor de libertate `df`. În cazul nostru:

```
> qt(0.975, 9)
[1] 2.262157
```

Aşadar, valoarea statisticii *t* este egală cu -0.783 şi în valoare absolută este mai mică decât valoarea critică 2.262157 cu $\alpha = 0.05$ şi deci se acceptă ipoteza testului şi se decide că media eşantionului este semnificativ similară cu media întregii populaţii.

Alternativ, se poate folosi *p*-valoarea calculată. Cu un prag de semnificaţie al testului de 5%, se poate aplica regula: dacă valorea *p* este mai mare decât 0.05 atunci putem accepta ipoteza nulă H_0, iar dacă este mai mică decât 0.05 atunci respingem ipoteza nulă H_0 în favoarea ipotezei alternative H_1. În cazul nostru, *p*-valoarea este $0.4537 > 0.05$ şi deci se acceptă ipoteza H_0.

7.1.1.3 Testul binomial

Testul neparametric binomial poate fi folosit când avem o variabilă cu două valori. Acesta se bazează pe următoarea formulă:

$$P(k \text{ succese în } n \text{ încercări}) = C_n^k p^k q^{n-k}$$

Adică, probabilitatea de a obţine k evenimente din n încercări se calculează ca produs al combinaţiilor de n luate câte k multiplicat cu probabilitatea teoretică a succesului ridicată la numărul de succese înmulţit cu probabilitatea teoretică a eşecului ridicată la puterea numărului de eşecuri.

Testul se realizează cu ajutorul statisticii Z:

$$Z = \frac{\bar{x} - p}{\sqrt{pq/n}} \sim N(0,1)$$

În R, în pachetul de programe standard "stats" există funcţia predefinită

```
binom.test(x, n, p = 0.5, alternative = c("two.sided",
        "less", "greater"), conf.level = 0.95)
```

unde

- `x` = numărul de succese sau un vector cu 2 componente: (număr de succese, număr de eşecuri)
- `n` = numărul de încercări
- `p` = probabilitatea de succes
- `alternative` = direcţia ipotezei alternative, valoarea implicită fiind "two.sided" (bilateral)
- `conf.level` = nivelul de încredere al testului.

Aplicaţie: Să se testeze ipoteza [8] că senatul de la o universitate americană spune adevărul sau nu şi anume că nu se face discriminare sexuală la admitere. Se aleg 500 de studenţi şi se găsesc 267 băieţi. Se consideră cu nivelul de semnificaţie 5%.

Soluţie: O secvenţa de cod R ce poate soluţiona aplicaţia este următoarea:

```
> binom.test(267,500)

   Exact binomial test

data:  267 and 500
number of successes = 267, number of trials = 500,
 p-value = 0.1399
alternative hypothesis: true probability of success
 is not equal to 0.5
95 percent confidence interval:
 0.4891844 0.5784114
sample estimates:
probability of success
  0.534
```

S-a obţinut o p-valoare apropiată de 0.1399, fiind astfel mai mare decât pragul de semnificaţie 0.05 şi astfel se acceptă ipoteza nulă. Prin urmare, este foarte probabil ca senatul să spună adevărul.

7.1.1.4 Testul χ^2

Unul dintre cele mai importante teste statistice pentru cea mai bună potrivire a modelului în cazul unei variabile nominale cu două sau mai multe valori. Ipoteza nulă în acest test este aceea că numărul observaţiilor în fiecare categorie este egal cu cel prezis de teorie, iar ipoteza alternativă este aceea că numărul observat este diferit de cel aşteptat. Statistica testului se calculează în felul următor: se ia numărul observat (n_i) , se scade numărul aşteptat (n'_i) şi se ridică diferenţa la pătrat. Cu cât deviatia faţă de ipoteza nulă este mai mare, cu atât mai mare ne aşteptăm ca diferenţa dintre numărul observat şi cel aşteptat. Prin ridicarea la pătrat, toate aceste diferenţe devin pozitive. Apoi, fiecare dintre aceste diferenţe se împarte la numărul aşteptat şi aceste diferente standardizate:

$$\chi^2 = \sum_{i=1}^{k} \frac{(n_i - n'_i)^2}{n'_i}$$

Ca în majoritatea testelor statistice, cu cât mai mare este diferenţa dintre valoarea observată şi cea aşteptată, cu atât mai mare devine valoarea testului statistic. Distribuţia testului statistic sub ipoteza nulă se potriveşte cu distribuţia χ^2. K. Pearson a arătat că, în cazul în care probabilităţi $p_i = n'_i / n_i$ nu sunt apropiate de 0 sau 1, iar produsele $n'_i = n_i \cdot p_i$, unde $p_i = f(x_i)$, după estimarea parametrilor, nu sunt prea mici (practic nu sunt mai mici decât 5), funcţia considerată are repartiţia χ^2 cu (s–1)–k grade de libertate, s fiind numărul de valori observate, iar k numărul parametrilor estimaţi. Dacă între repartiţia de selecţie şi repartiţia teoretică există concordanţă, atunci statistica χ^2 definită în relaţia $\chi^2 = \sum_{i=1}^{k} \frac{(n_i - n'_i)^2}{n'_i}$ trebuie să fie mai mică şi nu va depăşi o valoare determinată $\chi^2_{(s-1)-k;\alpha}$ corespunzătoare numărului gradelor de libertate (s–1)–k şi pragului de semnificaţie α dat. Regiunea critică a testului va fi dată de inegalitatea $\chi^2 > \chi^2_{(s-1)-k;\alpha}$ şi deci, dacă $\chi^2 \le \chi^2_{(s-1)-k;\alpha}$ acceptăm ipoteza H_0, în caz contrar o respingem.

În pachetul standard „stats” există funcţia R predefinită

```
chisq.test(x, y = NULL, correct = TRUE,
    p = rep(1/length(x), length(x)), rescale.p = FALSE,
    simulate.p.value = FALSE, B = 2000)
```

unde

- `x` = vectorul valorilor din primul eşantion
- y = vector (opţional) de valori din al doilea eşantion
- `correct` = indicator logic referitor la faptul că se aplică sau nu corecţie de continuitate când se aplică testul pentru tabele 2 pe 2 – o jumătate se scade din celelalte
- `p` = vectorul probabilităţilor de aceeaşi lungime ca vectorul *x* cu valoari implicite egale
- `rescale.p` = indicator logic cu valoarea TRUE în cazul în care este necesară rescalarea la suma 1, FALSE în cazul în care suma valorilor lui *p* nu este 1
- `simulate.p.value` = indicator logic ce specifică dacă se calculează valorile *p* cu simulare Monte-Carlo
- `B` = întreg ce specifică numărul de duplicate folosite în testul Monte-Carlo.

Aplicaţie: Considerăm că la o facultate din capitală 70% dintre studenţi sunt din Bucureşti, 10% din Piteşti, 10% din Ploieşti şi 10% din alte judeţe. Vrem să testăm dacă proporţiile observate în eşantionul considerat diferă semnificativ de cele din ipoteza statistică.
Soluţie: Presupunem că frecvenţele observaţiilor sunt următoarele: 145 studenţi din Bucureşti, 24 din Piteşti, 20 din Ploieşti şi 11 din alte judeţe.

Următoarea secvenţă de cod R aplică testul χ^2 pentru aplicaţia considerată

```
> frecvente = c(145,24,20,11)
> proportii = c(0.7,0.1,0.1,0.1)
> chisq.test(frecvente,p=proportii)

        Chi-squared test for given probabilities

data:  frecvente
X-squared = 5.0286, df = 3, p-value = 0.1697
```

Cuantila χ^2 cu 4-1=3 grade de libertate se pot determina fi din tabele (Tabelul 3 de la anexa 1), fie în R astfel:

```
> qchisq(0.95,3)
[1] 7.814728
```

Aceste rezultate (*p*-valoarea este 0.1697 > 0.05 sau valoarea calculată a statisticii testului este 5.0286 < valorea cuantilei cu 3 grade de libertate = 7.814728) arată că proporţiile studenţilor din eşantionul considerat nu diferă semnificativ de valorile furnizate în ipoteza statistică.

7.1.2 Teste statistice pentru două eşantioane

7.1.2.1 Testul *t* pentru două eşantioane nepereche

Multe cazuri de analiză statistică implică o comparaţie între mediile a două colectivităţi generale. Spre exemplu, un patron al unui restaurant doreşte să vadă dacă există diferenţe între vânzările realizate înainte şi după o campanie de publicitate, un grup de consumatori doreşte să vadă dacă există o diferenţă semnificativă între consumul electric pentru două tipuri de cuptoare cu microunde etc.

Testul *t* pentru două eşantioane independente (nepereche) este probabil cel mai folosit test statistic şi cu siguranţă cel mai cunoscut. Utilitatea testului constă în faptul că statisticienii examinează cel foarte des natura a două variabile pentru a afla dacă variabilele sunt asociate sau nu. Testul este folosit ca o metodă de **evaluare a diferenţelor mediilor a două grupuri**, care pot fi independente (ca de exemplu, [7] tensiunea arterială a pacienţilor sub tratament cu un anumit medicament şi tensiunea arterială a unui grup de pacienţi care primesc placebo) sau dependente (ca de exemplu, tensiunea arterilă a unui grup de pacienţi înainte şi după ce primesc un anumit medicament). Testul poate fi folosit şi pentru eşantioane de dimensiune mică, dar care sunt distribuite aproximativ normal.

În general, testul *t* este utilizat în următoarele trei situaţii, diferenţiate de situaţia existentă între dispersiile populaţiilor şi independenţa eşantioanelor:

- eşantioane independente, dispersii egale;
- eşantioane independente, dispersii diferite;
- eşantioane dependente (perechi, corelate).

Cel mai des întâlnit caz este acel al unui test *t* pentru două eşantioane independente, unde componentele din cele două eşantioane nu sunt asociate. Presupunând că se cunosc două eşantioane (x_1, ..., x_n) şi (y_1, ..., y_m), n>30, m>30, cu mediile m_1, respectiv m_2 şi că diferenţa dintre cele două este distribuită normal, ipotezele statistice sunt următoarele:

H_0: $m_1=m_2$
H_a: $m_1\neq m_2$

Statistica testului care se calculează este următoarea:

$$t = \frac{\bar{x} - \bar{y}}{\left(\frac{1}{n} + \frac{1}{m}\right)\sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2 + \sum_{j=1}^{m}(y_i - \bar{y})^2}{n+m-2}}}$$

Aceasta este repartizată Student cu $n+m$-2 grade de libertate.

Regiunea critică este dată de:

$$|t| > t_{\alpha/2, n+m-2}$$

Reamintim că în pachetul standard „stats” se găseşte funcţia R `t.test` pe care am folosit-o pentru a aplica testul *t* pentru un singur eşantion. Prototipul acesteia era

```
t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
    mu = 0, paired = FALSE, var.equal = FALSE, conf.level = 0.95)
```

unde
- `x` = vectorul valorilor din primul eşantion
- `y` = al doilea vector (opţional) de valori din al doilea eşantion
- `alternative` = direcţia ipotezei alternative, valoarea implicită fiind "two.sided" (bilateral)
- `mu` = media teoretică populaţiei
- `paired` = indicator logic pentru un t-test pereche
- `var.equal` = variabilă logică ce indică dacă cele două varianţe sunt egale sau nu
- `conf.level` = intervalul de încredere

De data aceasta o să aplicăm funcţia `t.test` pentru două grupuri.

Exemplu: Considerăm doi vectori independenţi (nepereche) cu câte 13, respectiv 8 componente. Aplicăm testul *t* pentru a vedea dacă mediile lor sunt egale statistic. Dacă nu specificăm în prealabil, se consideră că dispersiile celor două eşantioane nu sunt egale.
Soluţie: Următoarea secvenţă de cod R aplică testul *t* pentru aplicaţia considerată

```
> x <- c(79.98, 80.04, 80.02, 80.04, 80.03, 80.03,
+ 80.04, 79.97, 80.05, 80.03, 80.02, 80.00, 80.02)
> y <- c(80.02, 79.94, 79.98, 79.97, 79.97, 80.03,
+ 79.95, 79.97)
> t.test(x, y)

Welch Two Sample t-test
data:  x and y
t = 3.2499, df = 12.027, p-value = 0.006939
alternative hypothesis: true difference in means is
not equal to 0
95 percent confidence interval:
 0.01385526 0.07018320
sample estimates:
mean of x mean of y
 80.02077  79.97875
```

Se observă că *p*-valoarea este 0.006939 şi nu este mai mare decât 0.05, deci există o diferenţă statistică între mediile celor două grupuri.

Dacă vrem să considerăm cele două dispersii egale, procedăm astfel:

```
> x <- c(79.98, 80.04, 80.02, 80.04, 80.03, 80.03,
+ 80.04, 79.97, 80.05, 80.03, 80.02, 80.00, 80.02)
> y <- c(80.02, 79.94, 79.98, 79.97, 79.97, 80.03,
+ 79.95, 79.97)
> t.test(x, y, var.equal=TRUE)

Two Sample t-test
data:  x and y
t = 3.4722, df = 19, p-value = 0.002551
alternative hypothesis: true difference in means is
not equal to 0
95 percent confidence interval:
 0.01669058 0.06734788
sample estimates:
mean of x mean of y
```

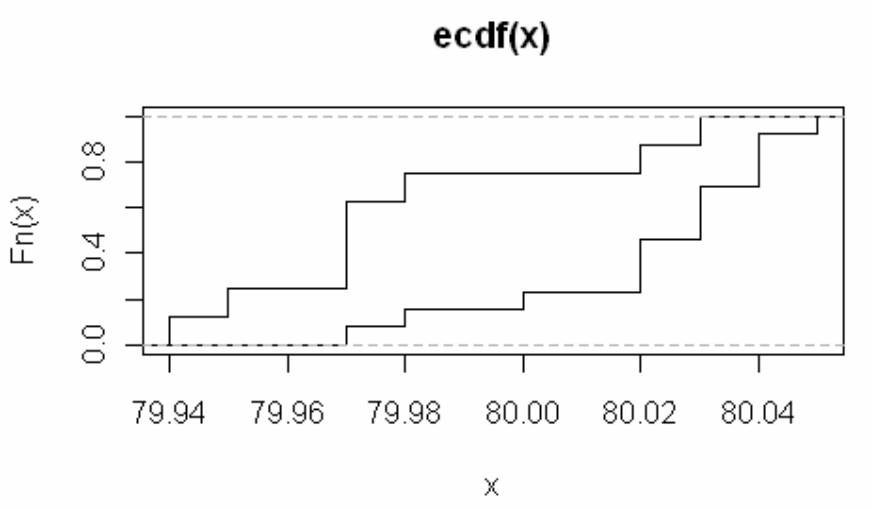
```
80.02077  79.97875
```

Rezultatele indică faptul că şi de data aceasta (valoarea probabilităţii *p* este 0.002551 < 0.05) există o diferenţă semnificativă statistic între mediile celor două grupuri. Cu alte cuvinte, cum media primului grup este 80.02077 şi media celui de-al doilea grup este 79.97875, primul grup este mai semnificativ statistic decât al doilea grup.

Există şi modalitatea grafică pentru compararea celor două grupuri de date:

```
>plot(ecdf(x),do.points=FALSE,verticals=TRUE,xlim=range(x, y))
>plot(ecdf(y),do.points=FALSE,verticals=TRUE,add=TRUE)
```

care conduce la următorul grafic al funcţiilor de repartiţie empirice



7.1.2.2 Testul Wilcoxon pentru două eşantioane nepereche

Testul Wilcoxon (al rangurilor cu semn) pentru două eşantioane nepereche este o variantă neparametrică a testului *t* prezentat anterior aplicat pentru două eşantioane independente (nepereche). Doar că testul Wilcoxon nu face presupunerea că diferenţa dintre cele două variabile este distribuită normal.

Presupunerea care se face este aceea că mediana diferenţelor între perechi de observaţii este zero. La testul *t* se făcea presupunerea că media diferenţelor perechilor de observaţii este zero.

Ideea acestui test este următoarea. În primul rând, valorile perechi pentru care diferenţa este zero se ignoră. Apoi, fiecare valoare absolută a diferenţelor dintre observaţii primeşte un rang: cea mai mică diferenţă primeşte valoarea 1, următoarea rangul 2, etc. Dacă avem diferenţe egale, ambele vor primi ca rang media rangurilor. Rangurile tuturor diferenţelor dintr-o direcţie (pozitiv/negativ) se însumează şi rangurile din direcţia opusă de asemenea şi se obţin valorile W_- şi W_+. Cea mai mică dintre aceste valori este valoarea testului, *W*. Spre deosebire de alte teste, valori mici pentru *W* sunt puţin probabile în cazul ipotezei nule.

În pachetul de funcţii R „stats” există funcţia predefinită `wilcox.test`, cu prototipul:

```
wilcox.test(x, y = NULL, alternative = c("two.sided", "less",
"greater"), mu = 0, paired = FALSE, exact = NULL, correct = TRUE,
          conf.int = FALSE, conf.level = 0.95)
```

sau

```
wilcox.test(formula, data, subset, na.action)
```

unde

- `x` = vectorul valorilor din primul eşantion
- `y` = al doilea vector (opţional) de valori folosit pentru teste cu două eşantioane
- `alternative` = direcţia ipotezei alternative, valoarea implicită fiind "two.sided" (bilateral)
- `mu` = media teoretică populaţiei cu valoarea implicită 0
- `paired` = indicator logic pentru un test pereche, implicit cu valoarea FALSE
- `exact` = indicator logic ce specifică dacă ar trebui calculată o p-valoare exactă
- `correct` = indicator logic ce specifică dacă ar trebui aplicată corecţie de continuitate la aproximarea normală a valorii lui *p*
- `conf.int` = indicator logic ce specifică dacă ar trebui clculat un interval de încredere
- `conf.level` = intervalul de încredere
- `formula` = este o formulă de forma membrul_stâng ~ membrul_drept, unde membrul_stâng este o variabilă numerică ce dă valorile, iar membrul_drept este un factor cu două nivele dându-se corespunztoare celor două grupuri
- `data` = o matrice opţională sau data frame ce conţine variabilele din formula `formula`, implicit acestea fiind luate din `environment`;
- `na.action` = o funcţie care indică ce ar trebui să se întâmple în cazul în care datele conţin valori NA.

Aplicaţie: Să considerăm o companie care produce baterii, printre care şi baterii scumpe. Însă se doreşte modificarea necostisitoare a celor scumpe astfel încât să se ajungă la creşterea duratei de viaţă. Ipotezele statistice ar fi următoarele:

H_0: durata de viaţă a bateriilor modificate este aceeaşi cu cea a celor scumpe
H_a: durata de viaţă a bateriilor modificate este mai mare cu cea a celor scumpe

Presupunem că avem la dispoziţie un eşantion de 6 baterii scumpe şi un eşantion de 5 baterii modificate. Cele 11 sunt construite în condiţii identice şi independent. Datele disponibile se referă la durata de viaţă a bateriilor considerate şi sunt următoarele: $X = (48, 53, 74, 111, 113, 335)$ şi $Y = (62, 101, 167, 174, 190)$.

Următoarea secvenţă de cod R aplică testul Wilcoxon pentru exemplul considerat:

```
> x=c(48,53,74,111,113,335)
> y=c(62,101,167,174,190)
> wilcox.test(x,y)

        Wilcoxon rank sum test
data:  x and y
W = 10, p-value = 0.4286
alternative hypothesis: true location shift is not
equal to 0
```

Cum *p*-valoarea este 0.4286 > 0.05 rezultă că diferenţa este nesemnificativă statistic între cele două eşantioane.

7.1.2.3 Testul *t* pentru două eşantioane pereche

Testul *t* pentru două eşantioane dependente (pereche) este folosit, în general, când măsurătorile provin de la acelaşi subiect înainte şi după o anumită acţiune. Ipoteza nulă pe care o testăm este aceea că diferenţa medie între perechile observate este zero. Pentru a aplica acest test mai trebuie ca cele două eşantioane să aibă aceeaşi lungime.

Presupunem că se cunosc două eşantioane dependente de lungime egală: $(x_1,..., x_n)$ şi $(y_1,..., y_n)$. În acest caz, statistica testului este următoarea:

$$t = \frac{\bar{x} - \bar{y}}{d},$$

unde $\bar{x}$, respectiv $\bar{y}$ reprezintă media de selecţie celor două eşantioane, iar d reprezintă eroarea standard a diferenţei dintre componentele celor două eşantioane care se poate calcula astfel

$$d = \sqrt{\frac{\sum_{i=1}^{n}(x_i - y_i)^2 - \frac{1}{n}\left(\sum_{i=1}^{n}(x_i - y_i)\right)^2}{n-1}}$$

După calcularea lui *t* dependent va trebui sa comparăm valoarea obţinută cu valoarea *t* teoretică.

Aplicaţie: Un cercetător doreşte să studieze influenţa metodelor de relaxare asupra reducerii conduitelor agresive la adolescenţi. O grupa de 12 subiecţi cu tulburări comportamentale este testată iniţial în ce priveşte nivelul agresivităţii. După acest moment, cei 12 adolescenti sunt învăţati să practice relaxarea şi urmează sedinţe de relaxare de câte 1 oră de 5 ori pe săptămână. Dupa o perioada de 2 luni, subiecţii sunt retestaţi utilizând aceeaşi probă pentru a observa dacă nivelul lor de agresivitate a scăzut în urma practicării metodelor de relaxare. Datele obţinute de cei 12 adolescenţi la chestionarul de agresivitate pe parcursul celor două testări sunt următoarele:

înainte = (21, 18, 15, 11, 24, 23, 17, 19, 19, 17, 20, 15)
după = (15, 13, 16, 13, 13, 12, 11, 10, 15, 17, 14, 16)

Secvenţa de instrucţiuni R ce rezolvă exemplul considerat este următoarea

```
> inainte = c(21,18,15,11,24,23,17,19,19,17,20,15)
> dupa = c(15,13,16,13,13,12,11,10,15,17,14,16)
> t.test(inainte, dupa, paired=TRUE)

        Paired t-test
data:  pre_test and post_test
t = 3.3726, df = 11, p-value = 0.006223
alternative hypothesis: true difference in means is
not equal to 0
95 percent confidence interval:
 1.56327 7.43673
sample estimates:
mean of the differences
     4.5
```

p-valoarea este 0.006223 şi nefiind chiar nulă rezultă că se acceptă ipoteza nulă. Acelaşi lucru se poate determina şi comparând valoarea testului cu valoarea quantilei distribuţiei *t* care se poate afla în R astfel

```
> qt(0.975,11)
[1] 2.200985
```

Se observă că valoarea obţinută 3.3726 este mai mare decât valoarea teoretică 2.200985 şi deci mediile celor două eşantioane pereche sunt semnificativ diferite statistic.

7.1.2.4 Testul *F* pentru două eşantioane independente

Testul *F* (testul Snedecor) pentru două eşantioane independente este un test de comparare folosit pentru a **determina dacă dispersiile acestora sunt egale sau nu** statistic. Testarea ipotezei privind dispersia poate fi utilizată pentru a trage concluzii privitoare la consistenţa unor procese economice ori privitoare la riscurile asociate.

Situaţia când se poate aplica testul *F* poate fi recunoscută prin:

- două populaţii caracterizate de variabilele X_1 respectiv X_2;
- variabilele sunt repartizate normal, $X_1 \sim N(m_1, \sigma_1^2)$ şi $X_2 \sim N(m_2, \sigma_2^2)$;
- două eşantioane, unul din fiecare populaţie: $(x_1,...., x_{n_1})$, respectiv $(y_1,...., y_{n_2})$.

Ipotezele testului pot fi de tip atât de tip lateral cât şi de tip bilateral.

Testul bilateral:

H_0: $\sigma_1^2 / \sigma_2^2 = 1$ (egalitatea dispersiilor celor două populaţii)

H_a: $\sigma_1^2 / \sigma_2^2 \neq 1$

Testul unilateral:

H_0: $\sigma_1^2 / \sigma_2^2 = 1$

H_a: $\sigma_1^2 / \sigma_2^2 < 1$ sau $\sigma_1^2 / \sigma_2^2 > 1$

Când ipoteza nulă este adevărată, statistica

$$F^* = \frac{S_1^2}{S_2^2},$$

este repartizată Snedecor F_{α, n_1-1, n_2-1} pentru un prag de semnificaţie α, unde

$$S_1^2 = \frac{1}{n_1 - 1}\sum_{i=1}^{n_1}(x_i - \bar{x})^2 \text{ estimaţie nedeplasată pentru } \sigma_1^2$$

$$S_2^2 = \frac{1}{n_2 - 1}\sum_{i=1}^{n_1}(y_i - \bar{y})^2 \text{ estimaţie nedeplasată pentru } \sigma_2^2$$

$$\bar{x} = \frac{1}{n_1}\sum_{i=1}^{n_1} x_i \text{ media de selecţie a primului eşantion}$$

$$\bar{y} = \frac{1}{n_2}\sum_{i=1}^{n_1} y_i \text{ media de selecţie a celui de-al doilea eşantion}$$

Repartiţia *F* (Snedecor) cu ν_1, respectiv ν_2 grade de libertate are funcţia de densitate de forma:

$$f(x) = \left(\frac{\nu_1}{\nu_2}\right)^{\nu_1/2} \frac{\Gamma\left(\frac{\nu_1+\nu_2}{2}\right)}{\Gamma\left(\frac{\nu_1}{2}\right)\cdot\Gamma\left(\frac{\nu_2}{2}\right)} x^{\frac{\nu_1}{2}-1}\left(1+\frac{x}{\nu_2}\right)^{-\frac{\nu_1+\nu_2}{2}}, x\geq 0$$

Decizia, la pragul de semnificaţie α, pentru testul *bilateral* este următoarea: se respinge ipoteza nulă H_0 în favoarea ipotezei alternative H_a dacă:

$$F^* > F_{1-\alpha/2, n_1-1, n_2-1} \text{ sau } F^* < F_{\alpha/2, n_1-1, n_2-1}$$

Decizia, la pragul de semnificaţie α, pentru testul *unilateral* este următoarea: se respinge ipoteza nulă H_0 în favoarea ipotezei alternative H_a dacă:

$$F^* > F_{1-\alpha, n_1-1, n_2-1}$$

În pachetul de funcţii R „stats" există funcţia predefinită `wilcox.test`, cu prototipul:

```
var.test(x, y, ratio = 1, alternative = c("two.sided", "less",
              "greater"), conf.level = 0.95, ...)
```

sau

```
var.test(formula, data, subset, na.action, ...)
```

unde

- `x` = vectorul valorilor din primul eşantion
- `y` = al doilea vector (opţional) de valori folosit pentru teste cu două eşantioane
- `ratio` = raportul presupus pentru dispersiile populaţiilor din care provin cele două eşantioane
- `alternative` = direcţia ipotezei alternative, valoarea implicită fiind "two.sided" (bilateral)
- `conf.level` = intervalul de încredere
- `formula` = este o formulă de forma membrul_stâng ~ membrul_drept, unde membrul_stâng este o variabilă numerică ce dă valorile, iar membrul_drept este un factor cu două nivele dându-se corespunztoare celor două grupuri
- `data` = o matrice opţională sau data frame ce conţine variabilele din formula `formula`, implicit acestea fiind luate din `environment`;
- `subset` = vector (opţional) ce specifică submulţimea de observaţii ce se va folosi
- `na.action` = o funcţie care indică ce ar trebui să se întâmple în cazul în care datele conţin valori NA.

Exemplu: Generăm 50 de variabile distribuite $N(0,2)$ şi 30 de variabile distribuite $N(1,1)$. Pentru a vedea dacă cele două eşantioane au dispersiile egale sau diferite se poate folosi următorul cod de instrucţiuni R:

```
> x <- rnorm(50, mean = 0, sd = 2)
> y <- rnorm(30, mean = 1, sd = 1)
> var.test(x, y)    # x şi y au aceeaşi dispersie?

        F test to compare two variances
data:  x and y
F = 5.4473, num df = 49, denom df = 29, p-value =
6.182e-06
```

```
alternative hypothesis: true ratio of variances is not
equal to 1
95 percent confidence interval:
  2.736839 10.248600
sample estimates:
ratio of variances
        5.447279

> qf(0.975,49,29) #quantila F(1-alpha/2,n1-1,n2-1)
[1] 1.990354
```

Cum valoarea calculată (5.4473) este mai mare decât valoarea teoretică (1.990354), atunci se respinge ipoteza nulă.

7.1.2.5 Testul Kolmogorov-Smirnov

Se foloseşte pentru cazul când dorim să determinăm dacă două eşantioane sunt semnificativ diferite. Spre deosebire de alte teste, testul Kolmogorov-Smirnov nu presupune nimic despre distribuţia datelor şi se poate spune că acest test este neparametric şi liber de distribuţii.

În general, testul Kolmogorov-Smirnov se poate aplica pentru un eşantion sau pentru două eşantioane. Când se aplic testul pentru un eşantion, se compară distribuţia testată cu alte distribuţii cunoscute, cum ar fi distribuţia uniformă, normală sau log-normală. Testul Kolmogorov-Smirnov pentru două eşantioane compară dacă cele două eşantioane de date provin din aceeaşi distribuţie, deşi nu se specifică exact tipul distribuţiei.

Din studierea convergenţei funcţiei empirice de repartiţie F_n către funcţia teoretică de repartiţie F, Kolmogorov a demonstrat următoarea teoremă:

$$\lim_{n\to\infty} P\left(d_n < \frac{\lambda}{\sqrt{n}}\right) = K(\lambda) = \sum_{k=-\infty}^{\infty} (-1)^k e^{-2k^2\lambda^2},$$

unde $\lambda > 0$ şi $d_n = \max\left|F_n(x) - F(x)\right|$.

Funcţia K este calculată în tabele pentru diverse valori ale lui λ (tabelul distribuţiei Kolmogorov).

Cu ajutorul acestei teoreme se poate da un criteriu de verificare a ipotezei H_0 că repartiţia empirică urmează o anumită lege de repartiţie.

Dacă ipoteza H_0 este adevărată, atunci diferenţele $\left|F_n(x) - F(x)\right|$ nu vor depăşi o anumită valoare $d_{\alpha;n}$ pe care o fixăm astfel încât: $P(d_n > d_{\alpha;n} / H_0) = \alpha$, unde α este riscul de gradul întâi. Dar $P(d_n > d_{\alpha;n}) = 1 - P(d_n \le d_{\alpha;n})$.

Luând $d_{\alpha;n} = \frac{\lambda_\alpha}{\sqrt{n}}$, înseamnă că atunci când H_0 este adevărată şi n suficient de mare avem: $P\left(d_n > \frac{\lambda_\alpha}{\sqrt{n}}\right) = 1 - P\left(d_n \le \frac{\lambda_\alpha}{\sqrt{n}}\right) = 1 - K(\lambda_\alpha) = \alpha$.

Unui prag de semnificaţie α dat îi corespunde prin relaţia $K(\lambda_\alpha) = 1 - \alpha$ o valoare λ_α astfel încât, pentru un volum n dat al selecţiei găsim valoarea $d_{\alpha;n} = \frac{\lambda_\alpha}{\sqrt{n}}$.

Regiunea critică pentru ipoteza H_0 este dată de relaţia $d_n > \frac{\lambda_\alpha}{\sqrt{n}}$. Deci:

- dacă $d_n < \frac{\lambda_\alpha}{\sqrt{n}}$, există concordanţă între F_n şi F şi se acceptă ipoteza H_0
- dacă $d_n \geq \frac{\lambda_\alpha}{\sqrt{n}}$, nu există concordanţă şi respingem ipoteza H_0.

În pachetul standard „stats" din R există funcţia ks.test

```
ks.test(x, y, alternative = c("two.sided",
    "less", "greater"), exact = NULL)
```

unde
- `x` = este un vector de valori din primul eşantion
- `y` = este (opţional) un vector de valori din al doilea eşantion sau un şir de caractere ce reprezintă funcţia de distribuţie ca de exemplu pnorm.
- `alternative` = direcţia ipotezei alternative, valoarea implicită fiind "two.sided" (bilateral).
- `exact` = NULL sau un indicator logic ce stabileşte dacă *p*-valoarea trebuie calculat; nu se foloseşte pentru unul sau două eşantioane.

Exemplu: Să considerăm două eşantioane: primul cu 50 de componente repartizate $N(0,1)$, iar al doilea cu 30 de componente repartizate $U(0,1)$.

```
> x <- rnorm(50)
> y <- runif(30)
> ks.test(x, y) # test Kolmogorov-Smirnov (au
+aceeaşi distribuţie?)

Two-sample Kolmogorov-Smirnov test
data:  x and y
D = 0.54, p-value = 1.598e-05
alternative hypothesis: two-sided
```

Cum *p*-valoarea este $1.598 \cdot 10^{-5} < 0.5$, rezultă că cele două grupuri sunt semnificativ diferite statistic.

7.1.2.6 Testul Sign

Testul Sign (al semnelor) este unul dintre cele mai simple teste statistice. Acesta se aplică când nu putem măsura diferenţa dintre cele două eşantioane, dar putem observa că **există o diferenţă între eşantioanele** în discuţie. Se utilizează semnul diferenţei şi nu valoarea acesteia, atunci când ambele valori sunt măsurate pentru aceiaşi subiecţi. Dacă nu ar exista nicio diferenţă între valorile perechi, atunci numărul diferenţelor pozitive ar trebui sa fie egal cu cel al diferenţelor negative. Cu cât numărul diferenţelor de un anumit semn este mai mare comparativ cu cel al diferenţelor de semn opus, cu atât creşte posibilitatea ca diferenţa dintre variabile să fie statistic semnificativă. Putem formula ipoteza testului astfel: $p = \boldsymbol{P(X>Y)}$ şi ipoteza nulă este în acest caz

H_0: p=0.5

Acestă ipoteză implică faptul că dându-se o pereche aleatoare de măsurători $(x_i, y_i)_i$ atunci x_i şi y_i au aceeaşi şansă de a fi mai mari unul decât celălalt.

Variabilele celor două eşantioane trebuie să fie de tip numeric, iar valorile să fie exprimate în aceeaşi unitate de măsură, pentru a se putea face diferenţa lor.

În pachetul de programe BSDA există funcţia R predefinită

```
SIGN.test(x, y = NULL, md = 0, alternative = "two.sided",
                  conf.level = 0.95)
```

unde

- `x` = este un vector de valori din primul eşantion (valori NA sau Inf se acceptă, însă sunt ignorate)
- `y` = este (opţional) un vector de valori din al doilea eşantion (valori NA sau Inf se acceptă, însă sunt ignorate)
- `md` = valoarea medianei populaţiei specificată în ipoteza nulă

`alternative` = direcţia ipotezei alternative, valoarea implicită fiind "two.sided" (bilateral).

- `conf.level` = intervalul de încredere

Exemplu: Să considerăm doi vector ce conţin valori înainte de un eveniment şi după acesta. Vom testa ipoteza nulă că „Nu există diferenţe între cei doi vectori de măsurători", cu ipoteza alternativă „Există diferenţe între cei doi vectori de măsurători"

```
> x=c(15,14,14,13,15,13,12,13)
> y=c(16,14,15,15,17,15,14,15)
> SIGN.test(x,y)

        Dependent-samples Sign-Test
data:  x and y
S = 0, p-value = 0.01563
alternative hypothesis: true median difference is
not equal to 0
95 percent confidence interval:
 -2.000 -0.675
sample estimates:
median of x-y
           -2

                  Conf.Level L.E.pt U.E.pt
Lower Achieved CI     0.9297     -2 -1.000
Interpolated CI       0.9500     -2 -0.675
Upper Achieved CI     0.9922     -2  0.000
```

Cum *p*-valoarea este `0.01563` este foarte mică, atunci se respinge ipoteza nulă in favoarea ipotezei alternative.

7.1.2.7 Testul de independenţă χ^2

Testul de independenţă χ^2 se foloseşte pentru a **verifica dacă două variabile aleatore sunt independente sau nu**. Se consideră un tabel $s \times k$ în care cele s linii corespund la s selecţii independente, iar cele k coloane se referă la k evenimente E_1, … E_k. Se notează cu n_{ij} frecvenţa observată a evenimentului E_j în selecţia de ordin i (numerele n_{ij} se numesc celule de frecvenţă). Teoria statistică spune că dacă $(n_{i.})_{i=1,2,\ldots,s}$ sunt cunoscute apriori, dacă sunt obţinute s selecţii independente, dacă parametrii θ_1, θ_2,…, θ_l sunt estimaţi cu ajutorul celulelor de frecvenţă observate şi metodei minimului χ^2, substituind parametrii θ_1, θ_2,…, θ_l în $p_j = P(E_j)$ şi aceştia în formula

$$X^2 = \sum_{i=1}^{s} \sum_{j=1}^{k} \frac{(n_{ij} - n_i p_j)^2}{n_i p_j},$$

cu

$$n_i = \sum_{j=1}^{k} n_{ij} \ , \ i = 1, 2, \ldots, s,$$

atunci varibila aleatoare X^2, pentru n mare, are aproximativ o repartiţie χ^2 cu numărul gradelor de libertate egal cu numărul celulelor de frecvenţă micşorat cu numărul de parametri estimaţi. În cazul unei selecţii de volum n, avem $s = 1$ şi cantitatea X^2 rezultată are pentru un număr n mare aproximativ o repartiţie χ^2 cu numărul gradelor de libertate egal cu numărul de celule micşorat cu numărul parametrilor estimaţi şi cu o unitate.

Fie p_{ij} probabilitatea ca un individ luat la întâmplare din populaţia considerată să aparţină liniei j şi coloanei j. Dacă $p_{i\circ}$ este probabilitatea ca un individ să aparţină liniei i şi $p_{\circ j}$ este probabilitatea ca un individ să aparţină coloanei j, atunci ipoteza că cele două variabile sunt independente poate fi scrisă sub forma

$$H_0: \ p_{ij} = p_{i\circ} p_{\circ j} \ , \ i = 1, 2, \ldots, s \text{ şi } j = 1, 2, \ldots, k.$$

Dacă avem o selecţie de volum n şi n_{ij} sunt frecvenţele observate, atunci

$$X^2 = \sum_{i=1}^{s} \sum_{j=1}^{k} \frac{(n_{ij} - n p_{i\circ} p_{\circ j})^2}{n_i p_{i\circ} p_{\circ j}}$$

Deoarece $p_{i\circ}$ şi $p_{\circ j}$ sunt necunoscute, conform metodei verosimilităţii maxime, acestea se pot estima cu $n_{i\circ} / n$, respectiv $n_{\circ j} / n$, unde

$$n_{i\circ} = \sum_{j=1}^{k} n_{ij} \text{, respectiv } n_{\circ j} = \sum_{i=1}^{s} n_{ij} \ .$$

Atunci varibila

$$X^2 = \sum_{i=1}^{s} \sum_{j=1}^{k} \frac{(n_{ij} - n_{i\circ} n_{\circ j} / n)^2}{n_{i\circ} n_{\circ j} / n}$$

urmează repartiţia χ^2 [sk–(s+k–2) = (s–1)(k–1)].

Testul este valid dacă cel puţin 80% dintre frecvenţele probabile depăşesc valoarea 5 şi toate frecventele probabile depăşesc valoarea 1. Această condiţie este necesară pentru ca repartiţia multinomială să poată fi aproximată cu o repartiţie normală, dar limitează semnificativ utilizarea testului χ^2. În cazul în care o frecvenţă probabilă este sub valoarea 2 sau dacă mai mult de 20% din frecvenţele probabile sunt sub valoarea 5, se recomandă utilizarea testului exact al lui Fisher. Atenţie că frecvenţele probabile calculate în cadrul testului nu sunt frecvente observate.

Aplicaţie: Să considerăm baza de date „*survey*”, în care coloana „*smoke*” conţine răspunsurile referitoare la fumat a 237 de studenţi, cu variantele „heavy”, „regul” (regulary), „ocass” (ocassionally) sau „never”, iar coloana „*exer*” se referă la cât de multă mişcare fac studenţii, cu variantele acceptate „freq” (frecvent), „some” sau „none”. Aceste date se pot pune într-un tabel de contingenţă astfel:

```
> library(MASS)
> t = table(survey$Smoke, survey$Exer)
> t
        Freq None Some
  Heavy    7    1    3
  Never   87   18   84
  Occas   12    3    4
  Regul    9    1    7
```

Să testăm acum ipoteza că obiceiul fumatului studenţilor chestionaţi este independent de mişcarea fizică făcută de aceştia la nivelul de semnificaţie de 0,05.

Soluţie: Putem aplica testul χ^2 de independenţă prin utilizarea funcţiei predefinite `chisq.test` astfel

```
> chisq.test(t)

        Pearson's Chi-squared test

data:  t
X-squared = 5.4885, df = 6, p-value = 0.4828

Warning message:
In chisq.test(t) :
  Chi-squared approximation may be incorrect
```

Cum p-valoarea găsită are valoarea 0,4828 care este mai mare decât pragul de semnificaţie de 0,05, atunci acceptăm ipoteza nulă conform căreia obiceiul fumatului este independentă de nivelul de mişcare fizică a studentului. Mesajul de eroare se datorează numărului de celule din tabelul de contingenţă. Pentru a evita acest avertisment, putem combina coloanele a doua cu a treia şi noul tabel sa-l numim `t2` şi apoi reaplicăm testul pentru acest nou tabel.

```
> t2=cbind(t[,"Freq"],t[,"None"]+t[,"Some"])
> t2
      [,1] [,2]
Heavy    7    4
Never   87  102
Occas   12    7
Regul    9    8
```

```
> chisq.test(t2)
        Pearson's Chi-squared test
data:  t2
X-squared = 3.2328, df = 3, p-value = 0.3571
```

Din nou se observă din nou independenţa celor două variabile considerate.

7.1.2.8 Testul de independenţă al lui Fisher

Testul exact al lui Fisher reprezintă o alternativă a testului χ^2 în examinarea asociaţiilor în cadrul unui tabel de contingenţă 2×2, atunci când frecvenţele probabile sunt mici. Condiţia de aplicare a acestui test este ca totalurile pe rânduri şi pe coloane să fie fixe, cunoscute dinainte.

Pentru a şti dacă între cele două variabile aleatoare componente ale vectorului aleator (X, Y) există o legătură semnificativă, se poate emite ipoteza H_0 că cele două variabile aleatoare sunt independente şi că deci repartiţiile unităţilor în cele patru rubrici ale tabelei vor fi următoarele

	x_1	x_2	Total
y_1	$\frac{n_{\circ 1} n_{1\circ}}{n}$	$\frac{n_{\circ 2} n_{1\circ}}{n}$	$n_{1\circ}$
y_2	$\frac{n_{\circ 1} n_{2\circ}}{n}$	$\frac{n_{\circ 2} n_{2\circ}}{n}$	$n_{1\circ}$
	$n_{\circ 1}$	$n_{\circ 2}$	

Pentru a vedea dac între cele două variabiel aleatoare există sau nu o conexiune, trebuie să calculăm nu numai probabilitatea $P(n_{11}, n_{12}, n_{21}, n_{11} / n_{\circ 1} n_{\circ 2})$, dar şi probabilităţile de a se obţine repartiţii care faţă de cea observată se îndepărtează mai mult de repartiţia din cazul independenţei.

Dacă $n_{11} < n_{\circ 1} n_{1\circ} / n$, tabelele ce se îndepărtează mai mult sunt acelea care au prima celulă egală respectiv cu (n_{11}-1), (n_{11}-2),…, 1, 0 unităţi.

Dacă $n_{11} > n_{\circ 1} n_{1\circ} / n$, tabelele extreme sunt acelea care au în prima celulă respectiv (n_{11}+1), (n_{11}+2),…, $\min\{n_{\circ 1}, n_{1\circ}\}$.

Compararea sumelor

$$P(n_{11}) = \sum_{i=0}^{n_{11}} p(i), \text{ dacă } n_{11} < n_{\circ 1} n_{1\circ} / n$$

$$P(n_{11}) = \sum_{i=n_{11}}^{\min\{n_{\circ 1}, n_{1\circ}\}} p(i), \text{ dacă } n_{11} > n_{\circ 1} n_{1\circ} / n$$

În pachetul de programe "stats" există funcţia R predefinită

```
fisher.test (x, y = NULL, workspace = 200000, hybrid = FALSE,
    control = list(), or = 1, alternative = "two.sided",
            conf.int = TRUE, conf.level = 0.95,
            simulate.p.value = FALSE, B = 2000)
```

unde
- `x` = este un tabel de contingenţă 2×2 sau un prim obiect factor;
- `y` = este (opţional) un al doilea obiect factor sau ignorat dacă `x` este matrice;
- `workspace` = un întreg ce specifică dimensiunea spaţiului de lucru folosit în algoritm, cu unitatea de măsură de câte 4 bytes;
- `hybrid` = o valoare logică implicită FALSE, folosită pentru tabele mai mari de 2×2, caz în care indică probabilităţile exacte (implicite) sau o aproximare hibridă ce ar trebui calculată;
- `control` = o listă conţinând componente predefinite pentru un control jos al algoritmului; în prezent, singurul folosit este "mult", un întreg pozitiv ≥ 2 cu valoarea implicită 30, folosit doar pentru tabele mai mari decât 2×2; acesta specifică de câte ori ar trebui alocat spaţiu pentru căi, cât şi pentru valori;
- `or` = raportul de diferenţe din ipoteză (folosit doar pentru tabele 2×2);
- `alternative` = direcţia ipotezei alternative, valoarea implicită fiind "two.sided" (bilateral), dar mai poate fi şi "greater" sau "less" – se poate specific doar prima literă din şir; valabil doar pentru tabele 2×2;
- `conf.int` = parametru logic ce indică dacă se doreşte determinarea unui interval de încredere intervalul de încredere
- `conf.level` = nivelul de încredere pentru intervalul de încredere returnat – folosit doar petru tabele 2×2 dacă `conf.int=TRUE`;
- `simulate.p.value` = o variabilă logică ce specifică dacă p-valoarea se va calcula prin simulare Monte Carlo – folosit pentru tabele mai mari decât 2×2;
- `B` – un întreg ce specifică numărul de duplicate folosit în testul Monte Carlo.

Pentru a şti dacă între cele două variabile aleatoare componente ale vectorului aleator (X, Y) există o legătură semnificativă, se poae emite ipoteza H_0 că cele două variabile aleatoare sunt independente şi deci că repartiţiile unităţilor în cele patru rubrici ale tabelei vor fi următoarele:

	x_1	x_2	total
y_1	$n_{\circ 1} n_{1\circ} / n$	$n_{\circ 2} n_{1\circ} / n$	$n_{1\circ}$
y_2	$n_{\circ 1} n_{2\circ} / n$	$n_{\circ 2} n_{2\circ} / n$	$n_{2\circ}$
	$n_{\circ 1}$	$n_{\circ 2}$	

Pentru a vedea dacă între cele două variabile aleatoare există sau nu o conexiune, trebuie să calculăm nu numai probabilitatea $P(n_{11}, n_{12}, n_{21}, n_{11} / n_{\circ 1}\, n_{\circ 2})$, dar şi probabilităţile de a se obţine repartiţii care faţă de cea observată se îndepărtează mai mult de repartiţia din cazul independenţei.

Dacă $n_{11} < n_{\circ 1} n_{1\circ} / n$, tabelele ce se îndepărtează mai mult sunt acelea care au prima celulă egală respectiv cu (n_{11}-1), (n_{11}-2),…, 1, 0 unităţi.

Dacă $n_{11} > n_{\circ 1} n_{1\circ} / n$, tabelele extreme sunt acelea care au în prima celulă respectiv (n_{11}+1), (n_{11}+2),…, $\min\{ n_{\circ 1}, n_{1\circ} \}$.

Compararea sumelor

$$P(n_{11}) = \sum_{i=0}^{n_{11}} p(i) \text{, dacă } n_{11} < n_{\circ 1} n_{1\circ} / n$$

$$P(n_{11}) = \sum_{i=n_{11}}^{\min\{n_{\circ 1}, n_{1\circ}\}} p(i) \text{, dacă } n_{11} > n_{\circ 1} n_{1\circ} / n$$

probabilităţilor relative tabelei observate şi cu probabilităţile care se îndepărtează tot mai mult de repartiţia în cazul independenţei comparată cu nivelul de semnificaţie α, permite să decidem asupra concordanţei sau neconcordanţei rezultatelor experimentale cu ipoteza de H_0 de independenţă.

Într-o asemenea comparaţie trebuie să ţinem seama dacă alternativa ipotezei H_0 este uni- sau bilaterală. În primul caz vom compara pe

$$P(n_{11}, n_{12}, n_{21}, n_{22} \mid n_{\circ 1}, n_{\circ 2}) = \frac{n_{\circ 1}! n_{\circ 2}! n_{1\circ}! n_{2\circ}!}{n! n_{11}! n_{12}! n_{21}! n_{22}!}$$

cu α, într-un al doilea cu α/2.

Trebuind să recurgem la factoriale, vom lua în întotdeauna în consideraţie rubrica în care apare cea mai mică frecvenţă observată. Aceasta este posibil, deoarece toate tabelele se alctuiesc cu aceeaşi repartiţie marginală şi pe de altă parte fiind vorba de tabele dictonice, au un singur grad de libertate, care odată fixată valoarea pentru o celulă, automat rămân fixate şi valorile pentru celelalte.

Aplicaţie: Cluster şi Galii în 2002 cu zburat cu un avion pentru a urmări speciile stârcul mare albastru şi marele egrete de la locul de odihnă la primul loc de hrănire de pe lacul Peltier, Minnesota şi au notat tipul pământ pe care au aterizat, obţinându-se următorul tabel

	Stârc	Egretă
Vegetaţie	15	8
Ţărm	20	5
Apă	14	7
Structuri	6	1

Să testăm ipoteza conform căreia cele două specii de păsări folosesc locurile de aterizare în proporţii egale.

Soluţie:

```
t = matrix (c(15,8,20,5,14,7,6,1), nrow=4, ncol=2,
dimnames = list( Loc = c("vegetatie", "tarm", "apa",
"structuri"), Pasari = c("starc","egreta")) )
> t
             Pasari
  Loc        starc  egreta
  vegetatie     15      14
  tarm           8       7
  apa           20       6
  structuri      5       1
> fisher.test(t)

        Fisher's Exact Test for Count Data

data:  t
p-value = 0.1587
alternative hypothesis: two.sided
```

Aplicarea testului lui Fisher conduce la o *p*-valoare de 0.1587 > 0.05, deci nu este nicio dovadă că cele două specii de păsări folosesc locuri de aterizare în proporţii diferite.

7.1.3 Teste statistice pentru 3 sau mai multe eşantioane

7.1.3.1 Testul de analiză dispersională ANOVA

Analiza dispersională (ANOVA, "Analysis of variance") este cea mai folosită metodă pentru compararea mediilor grupurilor pentru variabilele cantitative. ANOVA combină şi extinde testele t şi χ^2, prin testarea egalităţii dintre trei sau mai multe medii (pentru trei sau mai multe grupuri). De subliniat este faptul că ANOVA se poate folosi cu rezultate foarte bune şi la comparaţia dintre două medii, însă testul îşi arată adevărata valoare la trei sau mai multe medii.

Se testează aşadar legătura dintre o variabilă metrică (pentru care se calculează media) şi o variabilă calitativă (a cărei valori sau categorii sunt considerate grupuri independente). De asemenea, ANOVA face o introducere clară în analiza cauzală: variabila cauză (independentă) este cea calitativă, iar variabila efect (dependentă) este cea metrică.

Sunt foarte multe modele experimentale care pot fi analizate cu diferite tipuri de analize dispersionale, însă în această secţiune vom vorbi despre analiza dispersională simplă (într-un singur sens).

Aplicaţie: Se considerăm următoarele două variabile: "vârsta" = vârsta la angajare a unei persoane (variablă cantitativă) şi "stratin" = strategia de atragere a tinerilor (variabilă calitativă). În cazul nostru, ne raportăm doar la strategia de atragere a tinerilor; este important însă de ştiut că vârsta la angajare a tinerilor poate fi influenţat de diverşi factori şi că există variante ale analizei de varianţă care iau în calcul mai mulţi asemenea factori. De pildă, varianta care ia în calcul doi factori se numeşte ANOVA bi-factorial (în engl. two-way ANOVA), iar varianta care ia în calcul mai mulţi factori se numeşte ANOVA multi-factorial (în engl. multi-way ANOVA).

Să presupunem că studiem oportunităţile de acces a tinerilor pe piaţa forţei de muncă şi analizăm diferite strategii folosite pentru a atrage tinerii să se angajeze. Ipoteza pe care dorim să o testăm este următoarea: "media de vârstă a persoanelor nou angajate este influenţată de strategia de atragere utilizată". Setul de ipoteze sttistice pentru ANOVA este:

$H_0 : m_1 = m_2 = \ldots = m_k$
H_a : cel puţin două medii sunt diferite

În cuvinte, ipoteza nulă susţine că nu este nici o diferenţă între rezultatele diferitelor strategii (fie sunt toate strategiile foarte bune şi atrag mulţi tineri, fie sunt toate foarte slabe şi nu atrag tineri), iar ipoteza alternativ susţine cel puţin o strategie dă rezultate mai bune decât cel puţin una dintre celelalte (este posibil ca o strategie să aibă un rezultat de mijloc, care nu este semnificativ diferit nici faţă de strategia de succes maxim, nici faţă de strategia care atrage cei mai puţini tineri; diferenţa semnificativ în acest caz există doar între prima şi ultima strategie).

După cum se poate vedea în tabelul următor, primul grup (cel de control, din localitatea unde nu s-a aplicat nici o strategie) are o medie a vârstei la angajare de 27,8 ani cu o abatere standard de 3,65 ani; al doilea grup (din localitatea unde s-a aplicat prima strategie) are o medie de 23,6 ani cu o abatere standard de 3,34 ani, iar al treilea grup o medie de 26,3 ani cu o abatere standard de 3,59 ani. La o primă

vedere, toate cele trei grupuri conţin tineri: exist vreo diferenţă semnificativă între cele trei medii? Cum testăm, mai exact, acest lucru?

Nr.crt	Localitate 1	Localitate 2	Localitate 3
1	22	28	20
2	27	22	28
3	32	24	31
4	30	18	26
5	29	21	26
6	27	26	30
7	33	25	21
8	24	20	25
9	24	24	29
10	30	28	27
$\bar{x}$	27,8	23,6	26,3
s	3,65	3,34	3,59

Analizând tabelul, putem extrage câteva informaµii interesante, care ne vor ajuta în cele ce vor urma. Avem un eşantion total format din 30 de persoane, deci n=30. Acest eşantion este format din trei grupuri independente de câte 10 persoane fiecare (subeşantioane din trei localităţi diferite); avem deci: n_1=10, n_2=10 şi n_3=10. Pentru fiecare dintre cele trei localităţi / grupuri putem calcula câte o medie şi câte o abatere standard; mai avem aşadar: $\bar{x}_1$ = 27,8 cu s_1 = 3,65, $\bar{x}_2$ = 23,6 cu s_2 = 3,34 şi $\bar{x}_3$ = 26,3 cu s3 = 3,59. În acelaşi timp, putem calcula o medie generală pentru eşantionul total (pentru toate cele 30 de observaţii) $\bar{\bar{x}}$ = 25,9 precum şi o abatere standard total s = 14,714.

Poate fi determinată variaţia mediilor de grupuri în jurul mediei generale:

$$s_{\bar{x}}^2 = \frac{1}{k-1}\sum_{j=1}^{k}(\bar{x}_j - \bar{\bar{x}})$$

Aceasta indică o estimare a erorii standard din populaţie, de unde putem extrage foarte simplu varianţă din populaţie ES = $\sigma/\sqrt{n}$.

Ne confruntăm, deci, cu două tipuri de variaţii: o variaţie între grupuri (variaţia mediilor de grup în jurul mediei generale) şi una în interiorul grupurilor (variaţiile observaţiilor în jurul fiecărei medii de grup). Ambele tipuri de variaţii sunt folosite ca estimări ale variaţiei generale în populaţie.

Analiza de varianţă se bazează pe compraţia dintre două estimări ale varianţei σ^2 pentru întreaga populaţie. Logica analizei este următoarea: dacă cele două estimări ale varianţei din populaţie σ^2 sunt aproximativ egale, atunci ipoteza nulă este adevărată (în populaţie, toate mediile sunt egale). Dacă ipoteza de nul nu este adevărat, atunci cele două estimări ale varianţei vor fi semnificativ diferite.

```
> localit1 = c(22,27,32,30,29,27,33,24,24,30)
> localit2 = c(28,22,24,18,21,26,25,20,24,28)
> localit3 = c(20,28,31,26,26,30,21,25,29,27)
> localit = c(localit1,localit2,localit3)
> n=rep(10,3)
> v=rep(1:3,n)
> table(localit)
localit
```

```
18 20 21 22 24 25 26 27 28 29 30 31 32 33
 1  2  2  2  4  2  3  3  3  2  3  1  1  1
>f=function(x) c(mean=mean(x),var=var(x),n=length(x))
> tapply(localitati,v,f)
$`1`
    mean      var        n
27.80000 13.28889 10.00000

$`2`
    mean      var        n
23.60000 11.15556 10.00000

$`3`
mean  var    n
26.3 12.9 10.0

> date = data.frame(localit=localit,v=factor(v))
> model = lm(localit~v,date)
> anova(model)
Analysis of Variance Table

Response: localit
          Df Sum Sq Mean Sq F value  Pr(>F)
v          2   90.6  45.300  3.6391 0.03987 *
Residuals 27  336.1  12.448
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’
0.1 ‘ ’ 1
```

Valoarea de 3,639 (mult mai mare decât 1) ne sugerează că ipoteza nulă este pe cale de a fi respinsă, pentru că variaţia explicată de diferenţele dintre grupuri este mult mai mare decât variaţa datorată erorilor aleatoare; existând diferenţe majore între grupuri, vor exista cu siguranţă diferenţe între mediile acestora. Modalitatea alternativă este de a compara *p*-valoarea cu pragul de semnificaţie ales; cum *p*-valoarea este 0.03987 şi este mai mică decât $\alpha = 5\%$ se respinge ipoteza nulă: cel puţin una dintre strategii a dat rezultate.

Anova nu ne spune însă decât dacă acceptăm sau respingem ipoteza nulă, adică dacă există sau nu o medie diferită statistic de celelalte. În cazul acceptării ipotezei nule cu Anova, pentru a vedea unde este diferenţa, putem folosi un **test al diferenţelor dintre medii** şi anume **testul Tukey**.

În pachetul standard de funcţii R „stats” există funcţia predefinită `Tukey.HSD`, cu prototipul:

```
TukeyHSD (x, which, ordered = FALSE, conf.level = 0.95, ...)
```

unde

- `x` = un obiect de potrivire a modelului, de obicei `aov`;
- `which` = vector de caractere ce specifică lista de termeni din modelul de potrivire pentru care ar trebui calculate intervalele;
- `ordered` = valoare logică ce specifică dacă nivelele factorului ar trebui ordonate crescător după medie înainte de a calcula diferenţele;
- `conf.level` = nivelul de încredere

Aceasta determină o mulţime de intervale de încredere pentru diferenţele dintre mediile nivelelor unui factor. John Tukey a calculat intervale bazate pe valorile mediilor grupurilor şi nu pe baza diferenţelor individuale.

În cazul nostru, o secvenţă de cod R necesară pentru rezolvarea problemei este următoarea:

```
> l1 = c(22,27,32,30,29,27,33,24,24,30)
> l2 = c(28,22,24,18,21,26,25,20,24,28)
> l3 = c(20,28,31,26,26,30,21,25,29,27)
> localitati = c(l1,l2,l3)
> n = rep(10,3)
> v = rep(1:3,n)

> grup = factor(v)
#inainte de functia "aov" trebuie convertit vectorul
#calitativ intr-un factor

> analiza = aov(localitati~grup) #analiza dispersiei
> summary(analiza)
            Df Sum Sq Mean Sq F value  Pr(>F)
grup         2   90.6  45.300  3.6391 0.03987 *
Residuals   27  336.1  12.448
---
Signif. codes: 0‘***’0.001‘**’0.01‘*’0.05‘.’0.1‘ ’1

> TestulTukey = TukeyHSD(analiza,"grup")

> TestulTukey

  Tukey multiple comparisons of means
    95% family-wise confidence level
Fit: aov(formula = localitati ~ grup)
$grup
    diff       lwr        upr     p adj
2-1 -4.2 -8.112164 -0.2878359 0.0335040
3-1 -1.5 -5.412164  2.4121641 0.6136386
3-2  2.7 -1.212164  6.6121641 0.2194422
```

Se observă că cele mai apropiate medii sunt cele ale grupurilor 1 şi 3, apoi ale grupurile 3 şi 2. Se observă că *p*-valoarea pentru grupurile 2 şi 1 este 0.0335040, valoare mai mică decât 0.05, aşadar media grupului 2 este diferită statistic de media grupului 1 la un nivel de semnificaţie de 5%.

7.1.3.2 Testul Kruskal-Wallis

Acesta este o **variantă neparametrică pentru a testa egalitatea mediilor** între grupuri, identică cu Anova unidirecţională, datele fiind înlocuite de ranguri. În plus, testul Kruskal-Wallis **nu presupune populaţii distribuite normal**.

Pe baza celor k eşantioane independente din k populaţii, paşii de calcul ai testului sunt următorii:

- se ordonează valorile din cele k eşantioane ca şi cum ele ar fi fost obţinute din aceeaşi populaţie;
- se atribuie câte un rang observaţiilor aşezate în ordine, de la 1 la n (pentru observaţii cu acelaşi rang, fiecăruia i se va atribui media rangurilor pe care le-ar fi primit dacă nu ar fi avut acelaşi rang).
- se calculează statistica $H_{KW} = \frac{12}{n(n+1)} \sum_{i=1}^{k} \frac{R_i^2}{n_i} - 3(n+1)$, unde R_i reprezintă suma rangurilor din eşantionul i, n_i reprezintă numărul observaţiilor eşantionului i, i=1,2,…, k, iar n este volumul selecţiei totale;
- se compară H_{KW} cu valoarea teoretică $\chi^2_{(1-\alpha)(k-1)}$ şi dacă $H_{KW} \geq \chi^2_{(1-\alpha)(k-1)}$ atunci ipoteza nulă se respinge, respectiv dacă $H_{KW} < \chi^2_{(1-\alpha)(k-1)}$ atunci ipoteza nulă se acceptă.

În pachetul standard de funcţii R „stats” există funcţia predefinită `kruskal.test`, cu următoarele variante de prototip:

```
kruskal.test(x,...)
```

sau

```
kruskal.test(x,g,...)
```

sau

```
kruskal.test(formula,data,subset,na.action,…)
```

unde

- `x` = un vector numeric (eşantionul) sau o listă de vectori numerici (eşantioanele independente);
- `g` = vector de factori ce ne dau grupurile elementelor corespunzătoare din `x`; acest argument este ignorat dacă `x` este listă;
- `formula` = formulă de forma valori_date ~ grupurile corespunzătoare;
- `data` = matrice opţională sau data frame ce conţine variabile de forma formula.
- `subset` = vector opţional ce specifică o submulţime de observaţiile ce pot fi folosite;
- `na.function` = funcţie ce specifică ce-ar trebui să se întâmple cu datele ce conţin NA.

Aceasta aplică testul de însumare a rangurilor Kruskal-Wallis pentru ipoteza nulă că parametrii distribuţiei lui `x` sunt aceeaşi în fiecare eşantion, au ipoteza alternativă că ei diferă în cel puţin un parametru.

Aplicăm acum testul Kruskal-Wallis în R pentru aplicaţia de la Anova unidirecţională:

```
> by(localitati,grup,summary)
grup: 1
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  22.00   24.75   28.00   27.80   30.00   33.00
------------------------------------------------
grup: 2
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.00   21.25   24.00   23.60   25.75   28.00
------------------------------------------------
grup: 3
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  20.00   25.25   26.50   26.30   28.75   31.00
```

```
> kruskal.test(localitati,grup)

        Kruskal-Wallis rank sum test

data:  localitati and grup
Kruskal-Wallis chi-squared = 5.8546, df = 2, p-value
= 0.05354
```

Se observă că *p*-valoarea este 0.05354, puţin mai mare decât pragul de semnificaţie 0.05, aşadar se acceptă ipoteza nulă conform căreia mediile grupurilor sunt egale statistic.

7.2 Serii de timp

O serie de timp reprezintă o mulţime de date măsurate la intervale egale de timp. Datele sunt numerice şi reprezintă valoarea unei mărimi fizice (serie unidimensională) sau a mai multor mărimi măsurate simultan (serie multidimensională sau vectorială). In cele ce urmează o să exemplificăm o parte din teoria seriilor de timp pe date reale. Utilizăm ca date concentraţia de CO în centrul oraşului Bucureşti. Datele măsurate la interval de o oră provin de la staţia Cercul militar, Calea Victoriei, între 1/1/2007 01:00 şi 12/30/2008 24:00. Ca date de lucru am utilizat esantionul dintre 1/3/2008 01:00 - 31/5/2008 24:00 şi am încercat să prezicem poluarea pentru următoarele câteva zile.

Datele sunt citite iniţial dintr-un fişier text şi reprezentate grafic:

```
co=read.table(file.choose())
vco=as.matrix(co)
tsco<-ts(vco,frequency=168)
plot(tsco,main="Concentratia de CO intre 1/3/2008 01:00 -
31/5/200824:00")
```

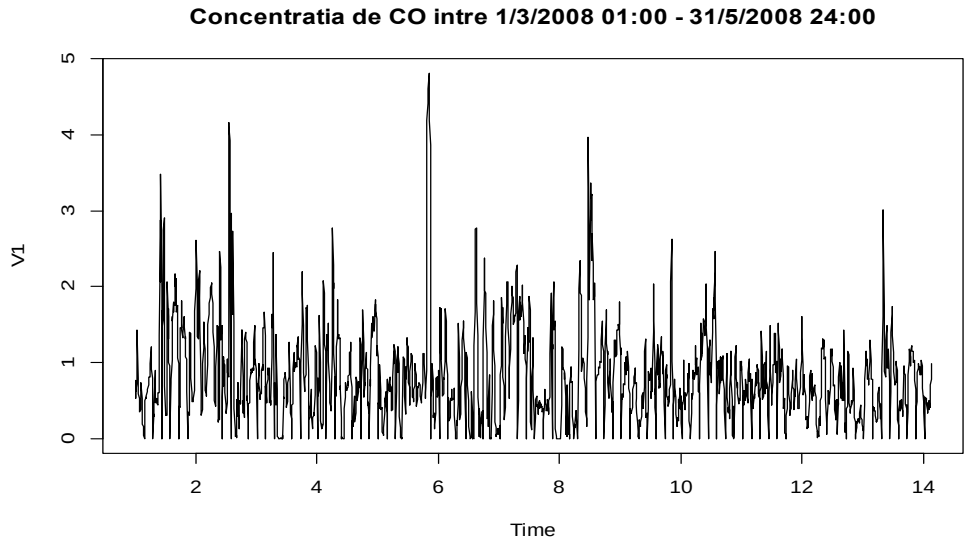


Fig 7.1

Funcţiile din R pentru analiza seriilor de timp prevăd că datele măsurate să fie convertite în obiecte de tip *ts*. Conversia se face prin comenzile *ts(...)* sau *as.ts(...)*, mai exact

```
ts(data = NA, start = 1, end = numeric(0), frequency = 1,
  deltat = 1, ts.eps = getOption("ts.eps"), class = ,
name= )
as.ts(x, ...)
```

Dintre parametrii din funcţiile de mai sus *data* reprezintă vectorul sau matricea cu datele măsurate, *start* este momentul de începere al observaţiilor, *frequency* este

numărul de observaţii pe unitatea de timp, x este un obiect R. Noi am luat frecvenţa 168=numărul de ore dintr-o săptămână (bănuind că există o periodicitate săptămânală). Ca urmare unitatea de timp este săptămâna.

7.2.1 Analiza statistică a seriilor de timp

Analiza statistică a seriilor de timp conform cu modelul Box-Jenkins[18] cuprinde următoarele stagii:

a. Identificarea modelului
b. Estimarea parametrilor
c. Validarea modelului (diagnosticul veridicităţii modelului ales)
d. Utilizarea modelelor pentru predicţii

Ca şi resursă de bază pentru analiza seriilor de timp utilizăm cartea [4] unde se găsesc elementele de teorie ca şi modul de utilizare a sistemului R pentru analiza seriilor de timp. Exemplele sunt în general luate din baze de date din realitate care se instalează odată cu R. Cartea lui Brockwell şi Davis[19] conţine pas cu pas etapele analizei seriilor de timp cu concentrare pe modelele ARIMA cu explicaţiile teoretice necesare şi demonstraţii pentru teoremele mai uşoare. Exemplele sunt realizate într-un program de calcul elaborat de autori, ITSM2000, care în varianta gratuită este disponibil pe internet (dar este limitat la serii ce nu depăşesc 200 de înregistrări). Pentru cei ce urmăresc teoria matematică a seriilor de timp o altă carte a aceloraşi autori[20], apărută în 1991, pune la dispoziţie demonstraţii riguroase pentru teoreme dificile din domeniu.

a. Identificarea modelului

Pentru identificarea modelului de serie temporală este nevoie de:

a1. Identificarea tendinţei, adică a unei componente ca variază în timp după o formulă cunoscută şi care poate fi extrapolată pentru a prezice valori pentru momente viitoare. Uneori unele transformări ale seriei iniţiale permit identificarea mai uşoară a tendinţei. Identificarea unor componente cu variaţie periodică exprimabile de asemenea într-o formă care să permită determinarea valorilor lor pentru momente viitoare face ca evoluţia deterministă a seriei să fie exprimată prin $m_t + s_t$ unde m_t este tendinţa neperiodică iar s_t este partea periodică. In anumite determinări ale tendinţei partea periodică este inclusă în formula pentru m_t astfel că tendinţa şi partea periodică nu se disting.

a2. După extragerea din serie a tendinţei şi a componentelor sezoniere urmează analiza resturilor. Dacă aceste resturi se comportă ca nişte valori ale unor v. a. independente atunci ele nu mai pot fi prognozate. Dacă însă există o anume corelaţie între ele atunci există posibilitatea de a stabili modele stochastice pentru ele

[18] G. E. P. Box, G. M. Jenkins. *Time Series Analysis, Forecasting and Control*. (1976). San Francisco: Holden-Day.
[19] P. J. Brockwell, R. A. Davis. *Introduction to Time Series and Forecasting*. Springer-Verlag New York Berlin Heidelberg, 2002
[20] P.J. Brockwell, R.A. Davis, (1991), *Time Series: Theory and Methods*, 2nd Edition, Springer-Verlag, New York.

şi a stabili evolţiile lor cele mai probabile. Pentru aceasta este însă nevoie de considera un model stochastic potrivit.

a1. Identificarea tendinţei şi a componentei periodice

Valorile măsurate ale seriei temporale pot fi privite ca valori ale unui şir de variabile aleatoare $(X_t)_{t\in Z}$ sau $(X_t)_{t\in N}$. Descompunerea în tendinţă plus componenta periodică plus partea reziduală se poate scrie

$$X_t = m_t + s_t + Y_t \qquad (7.2.1)$$

In formula de mai sus m_t este tendinţă iar s_t este componenta periodică. Pentru tendinţă se utilizează în general metoda celor mai mici pătrate. Se specifică modelul de exemplu $m_t = a + bt$ şi se estimează a şi b prin metoda celor mai mici pătrate. Pentru specificarea modelului în R avem informaţii în manualul de introducere în R care vine cu kitul de instalare. Reproducem de acolo câteva specificaţii (seria de date y este modelata ca expresie de alte serii de date x, x1, x2, etc., linear, prin formule):

Notaţia în R	**Explicaţii**
y ~ x sau y~1+x	y~a+bx
y ~ 0 + x sau y ~ -1 + x sau y ~ x - 1	y~bx
log(y) ~ x1 + x2	log(y) ~a+b*x1+c*x2
y ~ 1+x+I(x^2)	y ~ a+b*x+c*x^2
y~1+x+I(sin(2*x))	y ~ a+b*x+c*sin(2*x)

In cadrul seriilor temporale în loc de y ~ x avem tendinţa ca funcţie de timp, x~t, unde t este timpul.
Determinarea parametrilor a, b, c, etc., se face prin comanda *lm* (linear model) ca mai jos:

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr =
TRUE,singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Dintre parametrii funcţiei *lm* doar “formula” (sub forma unui tabel ca mai sus) este obligatoriu deoarece datele “data” sunt menţionate în formulă, iar ceilalţi parametri au valori implicite).

Rezultatul funcţiei *lm* este o listă cu elemente numite din care *coefficients* conţine coeficienţii obţinuţi în urma regeresiei liniare, *residuals* conţine diferenţele între valorile actuale şi cele obţinute prin foemula de regresie. Celelalte componente se pot afla prin help(lm) în consola R.

Secvenţa de mai jos produce un şir de valori care se modelează în două feluri şi apoi se reprezintă grafic datele iniţiale şi modelele:

```
x=1:20/5
y=rnorm(20)/5+2*x+0.2*sin(2*x)
m1=lm(y~1+x)
m2=lm(y~1+x+I(sin(2*x)))
y1=m1$coeff[1]+m1$coeff[2]*x
plot(x,y, main="Modelele y~1+x si y~1+x+I(sin(2*x))")
lines(x,y1, lwd=2, lty=3)
y2=m2$coeff[1]+m2$coeff[2]*x+m2$coeff[3]*sin(2*x)
```

```
lines(x,y2,lwd=3,lty=4)
grid(5,5)
```

Rezultatul este în figura următoare:

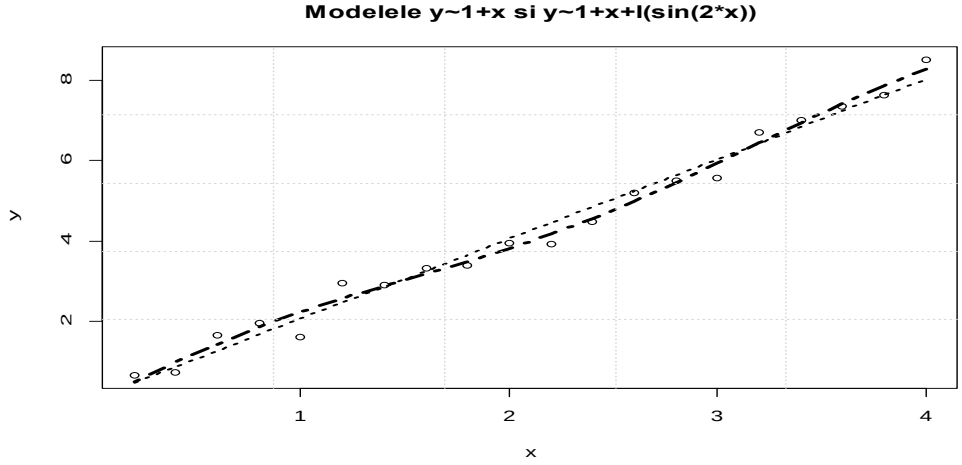


Fig 7.2

Coeficienţii celor dauă modele sunt:

```
> m1

Call:
lm(formula = y ~ 1 + x)

Coefficients:
(Intercept)            x
    0.02623      2.01722
> m2

Call:
lm(formula = y ~ 1 + x + I(sin(2 * x)))

Coefficients:
  (Intercept)              x  I(sin(2 * x))
     -0.06758        2.04058         0.26963
```

Pe figură vedem că modelul y~1+x+I(sin(2*x)) aproximează mai bine graficul (x,y) faţă de modelul `y~1+x`.

Pentru modele neliniare putem utiliza funcţia R numită *nlm*. Forma funcţiei este:

```
nlm(f, p, ..., hessian = FALSE, typsize = rep(1,
length(p)),
    fscale = 1, print.level = 0, ndigit = 12, gradtol =
1e-6,
    stepmax = max(1000 * sqrt(sum((p/typsize)^2)), 1000),
    steptol = 1e-6, iterlim = 100, check.analyticals =
TRUE)
```

Parametrii obligatorii sunt:
i. f=funcţia de minimizat sub forma unei expresii de parametrii căutaţi şi
ii. p=valori iniţiale pentru aceşti parametri.

De exemplu pentru a modela $y \sim p_1 + p_2 x + p_3 e^{p_4 x}$ unde parametrii p_1, p_2, p_3, p_4 se vor determina prin metoda celor mai mici pătrate trebuie să utilizăm funcţia $f(p) = \sum_i \left(y_i - \left(p_1 + p_2 x + p_3 e^{p_4 x}\right)\right)^2$, adică

```
f(p)= sum((y-p[1]-p[2]*x-p[3]*exp(p[4]*x))^2)
```

Datele x şi y trebuie să fie disponibile la momentul apelării funcţiei f. Deşi nu sunt parametri de intrare pentru f, x şi y se văd în contextul în care este funcţia f apelată. Scriptul următor face o estimare a parametrilor p.

```
x=1:20/5
y=rnorm(20)/5+2*x+0.2*sin(2*x)
f<-function(p){sum((y-p[1]-p[2]*x-p[3]*exp(p[4]*x))^2)}
m3<-nlm(f,c(1,2,3,0))
p=m3$estimate
y1=p[1]+p[2]*x+p[3]*exp(p[4]*x)
plot(x,y,main="Estimare neliniara")
lines(x,y1)
grid(5,5)
```

Graficul punctelor date şi al estimării (regresiei) neliniare este:

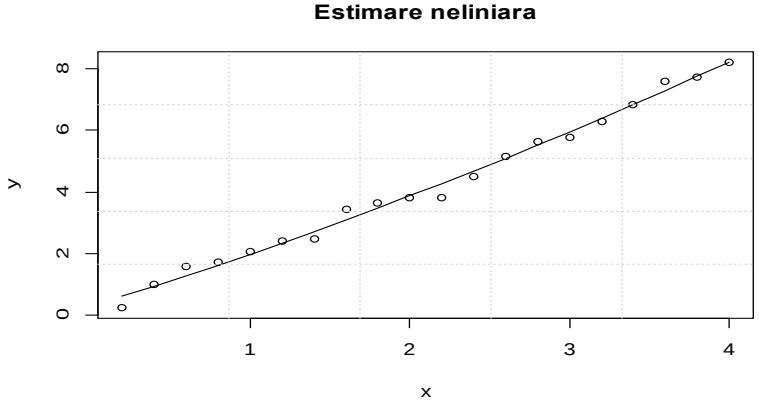


Fig. 7.3

Rezultatul m3 al funcţiei *nlm* este o listă unde *m3$estimate* este un vector cu valorile parametrilor determinaţi de rutina *nlm*. Dacă parametrul *$code* este 1 sau 2 înseamnă că probabil parametrii p sunt corect estimaţi (s-a ajuns la concluzia aceasta din motive diferite pentru $code=1 sau $code=2). Vedem mai jos acest rezultat:

```
> m3
$minimum
[1] 2.24936e-14
```

```
$estimate
[1] 4.7903681 6.2172098 0.7584119

$gradient
[1] -2.586661e-06 -2.085010e-06 -4.790367e-05

$code
[1] 2

$iterations
[1] 7
```

Dacă în modelul liniar sau în cel neliniar sunt cuprinse funcţii periodice atunci se determină simultan şi tendinţă şi partea periodică.

O altă modalitate de identificare a tendinţei este sub forma unei *medieri exponenţiale*:

Valorile tendinţei seriei (x_t) sunt calculate succesiv prin

$$m_t = \alpha x_t + (1-\alpha) m_{t-1} \tag{7.2.2}$$

cu $\alpha \in (0,1)$. Alegerea lui α este subiectivă. Dacă datele încep cu x_1 atunci avem :

$$m_t = (1-\alpha)^{t-1} x_1 + \alpha(1-\alpha)^{t-2} x_2 + ... + \alpha(1-\alpha) x_{t-1} + \alpha x_t .$$

In acest fel tendinţa este păstrată sub forma unui vector de valori $(m_t)_{1\le t\le T}$. Estimarea după timpul maxim de măsurare, T, se face prin $m_{T+k} = m_T$ ceea ce nu este foarte convenabil. O estimare a tendinţei în aşa fel ca ea să poată fi extinsă într-un mod mai rezonabil după t=T se poate face prin procedura Holt care constă în mediarea exponenţială a tendinţei m_t cât şi a pantei P_t sub forma:

$$\begin{aligned} m_1 &= x_1 \\ P_1 &= \frac{(x_n - x_1)}{n-1} \text{ pentru un n, de exemplu } n = 2 \\ m_t &= \alpha x_t + (1-\alpha)(m_{t-1} + P_{t-1}) \\ P_t &= \beta(m_t - m_{t-1}) + (1-\beta) P_{t-1} \end{aligned} \tag{7.2.3}$$

Parametrii α, β sunt subiectiv aleşi în intervalul (0,1). Extrapolarea tendinţei pentru t>T se face linear prin

$$m_{T+k} = m_T + P_T \cdot k \tag{7.2.4}$$

O procedură mai generală este Holt-Winters aditiv care mediază exponenţial tendinţa m_t, componenta sezonieră s_t cât şi panta tendinţei P_t. Pentru aceasta este nevoie de trei parametri α, β, γ aleşi subiectiv în intervalul (0,1). De asemenea este nevoie apriori de lungimea L a părţii periodice şi de valorile iniţiale pentru partea periodică : $(s_t)_{t=1..L}$.

$$\begin{aligned} m_t &= \alpha(x_t - s_{t-L}) + (1-\alpha)(m_{t-1} + P_{t-1}) \\ P_t &= \beta(m_t - m_{t-1}) + (1-\beta) P_{t-1} \\ s_t &= \gamma(x_t - m_t) + (1-\gamma) s_{t-L} \end{aligned} \tag{7.2.5}$$

Predicţia valorilor dincolo de pragul t=T se face după formula:

$$x_{T+k} = m_T + k * P_T + s_{T+l+(h-l) \bmod L} \qquad (7.2.6)$$

Există şi un preocedeu Holt-Winters multiplicativ.

Procedura Holt-Winters implementată în R este apelabilă prin:

```
HoltWinters(x, alpha = NULL, beta = NULL, gamma = NULL,
seasonal = c("additive", "multiplicative"), start.periods
= 2, l.start = NULL, b.start = NULL, s.start = NULL,
optim.start = c(alpha = 0.3, beta = 0.1, gamma =
0.1),optim.control = list())
```

Dintre parametrii funcţiei doar x de tip *ts* este obligatoriu. Rezultatul funcţiei este un obiect HoltWinters adică o listă cu componentele `fitted`, `x`, `alpha`, `beta`, `gamma`, `coefficients`, `seasonal`, `SSE`, `call`. Lungimea L a perioadei este luată egală cu frecvenţa (frequency) utilizată la definirea seriei temporale. Dintre elementele listei rezultat, `fitted` este o listă ce conţine timpul t, valoarea estimată $\hat{x}_t$, m_t numit level, P_t numit tendinţă şi s_t. Prin comanda help(HoltWinters) se poate afla şi semnificaţia celorlalţi parametri.

Predicţia valorilor ce urmează în serie după timpul maxim T de măsurare se poate face prin funcţia *predict*. In scriptul ce urmează după determinarea tendinţei şi a părţii sezoniere (perioada=168) pentru datele CO dinainte, se face o predicţie pentru o săptămână (=168 ore) pentru nivelul de CO. Mai jos avem graficul concentraţiei de CO cu negru (seria de timp tsco a fost creată mai înainte) împreună cu valorile netezite prin metoda Holt-Winters cu verde şi predicţia Holt-Winters pentru 168 de ore cu albastru.

```
plot(tsco,main="Co masurat + CO prezis Holt-
Winters",cex=2)
m4<- HoltWinters(tsco)
lines(fitted(m4)[,1], col = 3)
pr <- predict(m4, 168, prediction.interval = FALSE)
lines(pr, main="Predictie Holt-Winters",col=4)
```

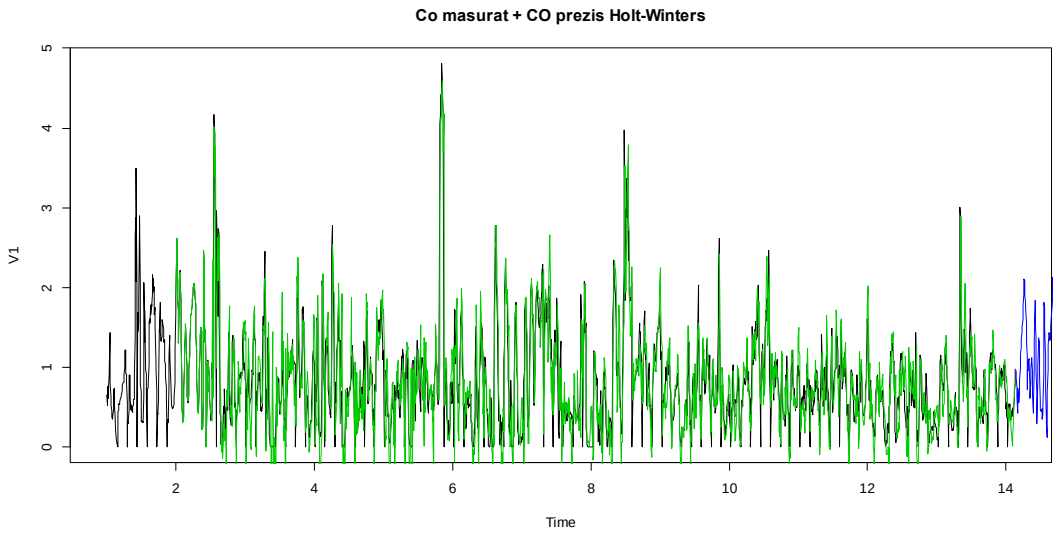


Fig. 7.4

Funcţia *predict* are forma:

```
predict(object, n.ahead=1, prediction.interval = FALSE,
        level = 0.95, ...)
```

unde *object* este un model (de exemplu Holt-Winters) ce conţine elementele de predicţie după ultima valoare măsurată (conform formulelor Holt-Winters de mai sus). Rezultatul funcţiei predict este o serie de timp cu valorile prezise în continuarea celor ale seriei iniţiale.

O altă modalitate de eliminare a tendinţei este procedura de diferenţiere. Astfel de la o serie (x_t) se obţine o nouă serie (y_t) cu $y_t = x_{t+1} - x_t$. O nouă diferenţiere conduce la seria (z_t) cu $z_t = y_{t+1} - y_t = x_{t+2} - 2x_{t+1} + x_t$ care se numeşte diferenţa de ordin doi s.a.m.d. Funcţia *diff* din R realizează aceste diferenţe finite la dreapta:

```
diff(x, lag = 1, differences = 1, ...)
```

Parametrii sunt *x*=vector sau matrice de numere, *differences* este ordinul diferenţei finite, *lag* este pasul de timp (de exemplu la lag=2 prima diferenţă se calculează după formula $y_t = x_{t+2} - x_t$). Dacă notăm B operatorul de întârziere $(BX)_t = X_{t-1}$ atunci seria obţinută prin prima diferenţă la lag=1 este $Y = (1 - B)X$, iar la lag=2 este $Y = (1 - B^2)X$. Dacă *y=diff(x)* atunci x poate fi restaurat prin *c(x[1],x[1]+cumsum(y))*.

Transformarea seriei într-o serie staţionară (vezi mai jos) se poate încerca şi printr-o transformare de tip Box-Cox:

$$y_t = \begin{cases} \dfrac{x_t^{\lambda} - 1}{\lambda} & \text{pour } \lambda \neq 0 \\ \ln(x_t) & \text{pour } \lambda = 0 \end{cases} \qquad (7.2.7)$$

In pachetul *TeachingDemos* există funcţia *vis.boxcox* care permite vizualizarea interactivă a graficului seriei temporale după o transformare Box-Cox. De asemenea funcţia *bct(y, lambda)* din acelaşi pachet calculează transformarea Box-Cox de mai sus.

a2. Determinarea modelului pentru reziduali

Analiza rezidualilor (diferenţele dintre valorile măsurate şi cele date de tendinţă şi componenta sezonieră) are scopul de a detremina dacă partea care rămâne după extragerea tendinţei şi a periodicităţii este formată din valori fără legătură între ele, caz în care aceste valori reziduale nu mai pot fi prezise, sau au o anumită corelaţie şi atunci se poate încerca o previziune a lor bazată pe un model statistic. Se pot încerca testele *Box.test* (din pachetul stats) sau *runs.test* (din pachetul tseries).
In cele ce urmează ne vom ocupa de serii de timp staţionare. Un şir de variabile aleatoare $(X_t)_{t\in Z}$ se numeşte temporală staţionară slab (staţionar de ordinul 2) dacă media $\mu_X(t) = E(X_t)$ este constantă (nu depinde de t) şi funcţia de *autocovarianţă*

$$\gamma_X(r,s) = E[(X_r - \mu_X(r))(X_s - \mu_X(s))] \tag{7.2.8}$$

depinde doar de diferenţa h=r-s. In cele ce urmează notăm E(X) media v.a. X. Pentro o serie temporală staţionară notăm $\gamma_X(h) = E((X_{t+h} - \mu_X)(X_t - \mu_X))$, independent de t. Avem:

$$\gamma_X(0) = \text{var}(X_t) > 0$$
$$\gamma_X(h) = \gamma_X(-h)$$
$$|\gamma_X(h)| \le \gamma_X(0)$$
$$\forall k \ge 1, \forall a_1, a_2, ..a_k \Rightarrow \sum_{i,j=1}^{k} a_i \gamma(i-j) a_j \ge 0$$

In general vom scrie $\gamma(h)$ sau γ_h în loc de $\gamma_X(h)$ dacă nu este pericol de confuzie asupra seriei temporale X. Numim funcţia de *autocorelaţie* funcţia:

$$\rho(h) = \frac{\gamma(h)}{\gamma(0)} \tag{7.2.9}$$

In general vom scrie ρ_h în loc de $\rho(h)$ Estimatori numerici pentru medie, funcţia de autocovarianţă şi autocorelaţie sunt:

$$\overline{X}_n = \frac{X_1 + X_2 + ...X_n}{n} \tag{7.2.10}$$

$$\hat{\gamma}_h = \frac{1}{n}\sum_{t=1}^{n-|h|}(X_{t+|h|} - \overline{X}_n)(X_t - \overline{X}_n) \tag{7.2.11}$$

$$\hat{\rho}_h = \frac{\hat{\gamma}_h}{\hat{\gamma}_0} \tag{7.2.12}$$

Box şi Jenkins recomandă $n \ge 50$ et $h \le n/4$ pentru estimările de mai sus.

Estimatorii pentru medie şi corelaţie pot fi studiaţi doar în unele ipoteze asupra seriei temporale X. Astfel introducem unele tipuri de serii temporale staţionare :

Zgomotul alb este un tip de serie temporală $(Z_t)_{t\in Z}$ în care fiecare v.a. este de medie zero şi varianţă σ^2, iar covarianţa este

$$\gamma_Z(h)=\begin{cases}\sigma^2 & ,h=0\\ 0 & ,h\neq 0\end{cases} \tag{7.2.13}$$

Variabilele aleatoare Z_t nu sunt neapărat independente.

Seria temporală i.i.t. este la fel ca zgomotul alb dar toate variabilele Z_t sunt independente identic distribuite.

Modelul general linear este o serie temporală $(X_t)_{t\in Z}$ în care $X_t=\sum_{j=-\infty}^{\infty}\psi_j Z_{t-j}$, $(Z_t)_{t\in Z}$ este un zgomot alb şi $\sum_{j=-\infty}^{\infty}|\psi_j|<\infty$. Se verifică fapţietul că în acest caz avem $\gamma_X(h)=\sigma^2\sum_{j=-\infty}^{\infty}\psi_{j+h}\psi_j$.

Serii staţionare MA(q) (medii mobile) sunt serii de forma $X_t=Z_t+\sum_{i=1}^{q}\theta_i Z_{t-i}$ unde $(Z_t)_{t\in Z}$ este un zgomot alb . Notăm $\theta_0=1$. Avem $E(X_t)=0$ şi

$$\gamma_X(h)=\mathrm{Cov}(X_{t+h},X_t)=\begin{cases}\sigma^2\sum_{i=0}^{q-|h|}\theta_i\theta_{i+|h|}, & |h|\le q\\ 0, & |h|>q\end{cases} \tag{7.2.14}$$

Seriile staţionare de tip MA(q) sunt singurele serii staţionare care au $\gamma_X(h)=0$ pentru h>q.

Seria staţionară autoregersivă AR(p) este o serie de tipul (în general cerem şi ca media să fie zero) $X_t=\phi_1X_{t-1}+\phi_2X_{t-2}+...\phi_pX_{t-p}+Z_t$ unde $(Z_t)_{t\in Z}$ este un zgomot alb şi Z_t este independent de X_{t-k}, k>0. Dacă polinomul $\phi(z)=1-\phi_1z-\phi_2z^2-...-\phi_pz^p$ are toate rădăcinile mai mari ca 1 în modul (serie cauzală) atunci $X_t=Z_t+\sum_{j=1}^{\infty}\psi_jZ_{t-j}$ cu $\sum_{j=1}^{\infty}|\psi_j|<\infty$. Inmulţind relaţia de definiţie a lui X_t cu X_{t-k} găsim sistemul şi luând mediile obţinem pentru k=1,2,..p:

$$\begin{cases}\rho_1=\phi_1+\phi_2\rho_1+\phi_3\rho_2+...+\phi_p\rho_{p-1}\\ \rho_2=\phi_1\rho_1+\phi_2+\phi_3\rho_1+...+\phi_p\rho_{p-2}\\ .....\\ \rho_p=\phi_1\rho_{p-1}+\phi_2\rho_{p-2}+\phi_3\rho_{p-3}+...+\phi_p\end{cases} \tag{7.2.15}$$

care prin rezolvare ne dă $\rho_1, \rho_2, ... \rho_p$. Avem de asemenea $\rho_0 = 1$. Pentru k>p avem analog $\rho_k = \phi_1\rho_{k-1} + \phi_2\rho_{k-2} + \phi_3\rho_{k-3} + ... + \phi_p\rho_{k-p}$. Funcţia de autocorelaţie se poate astfel determina complet şi nu se anulează în principiu nicăieri.

Se defineşte funcţia de autocorelaţie parţială $\phi_{k,k}$ în felul următor: Pentru X_t şi X_{t+k} se consideră componentele perpendiculare pe spaţiul vectorial generat de $X_{t+1}, X_{t+2}, ... X_{t+k-1}$, fie ele $X_{t,0}$ respectiv $X_{t,k}$. Funcţia de autocorelaţie parţială se defineşte acum ca şi $\phi_{k,k} = \text{Corr}(X_{t,0}, X_{t,k})$. Deoarece proiecţiile depind doar de produsele scalare reciproce (corelaţii între X_i, X_j) şi seria este staţionară rezultă că $\phi_{k,k}$ nu depinde de t ci doar de ρ_i pentru i=0,1,2,..k. Pentru seriile autoregresive $\phi_{k,k}$ este zero dacă k>p şi aceasta este o modalitate de a detecta dacă o serie este de tip AR(p) estimând pe $\phi_{k,k}$ prin $\hat{\phi}_{k,k}$ ce se calculează cu ajutorul estimatorilor $\hat{\rho}_i$ pentru corelaţiile ρ_i.

Serii staţionare autoregresive medie mobilă ARMA(p,q) sunt serii care verifică o relaţie de forma:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + ... \phi_p X_{t-p} + \theta_1 Z_{t-1} + \theta_2 Z_{t-2} + ... + \theta_q Z_{t-q} + Z_t$$

unde Z_t este un zgomot alb independent de X_{t-k} pentru k>0. Dacă seria este cauzală adică polinomul $\phi(z) = 1 - \phi_1 z - \phi_2 z^2 - ... - \phi_p z^p$ are rădăcinile mai mari ca 1 în modul atunci $X_t = Z_t + \sum_{j=1}^{\infty} \psi_j Z_{t-j}$ cu $\sum_{j=1}^{\infty} |\psi_j| < \infty$. Dacă şi polinomul $\theta(z) = 1 + \theta_1 z + \theta_2 z^2 + ... + \theta_q z^q$ are toate rădăcinile în modul mai mari ca 1 seria se numeşte invertibilă. Funcţia de autocovarianţă se poate determina exact pentru seriile cauzale dacă se dau coeficienţii ϕ, θ, σ^2, unde σ^2 este varianţa lui Z_t. Funcţia ARMAacf din pachetul stats face acest lucru în R. Apelul se face prin:

```
ARMAacf(ar = numeric(0), ma = numeric(0), lag.max = r, pacf =
FALSE)
```

Se vede din apelul funcţiei că prin opţiunea *pacf* putem obţine funcţia de autocorelaţie parţială.

Serii nestaţionare ARIMA(p,d,q) sunt serile care prin diferenţa de ordin d la dreapta devine staţionară de tip ARMA(p,q).

Am văzut în modelele de serii temporale staţionare că tipurile MA(q), AR(p) sunt determinate de corelaţiile ρ_k. Estimatorii pentru Proprietăţile estimatorilor pentru medie şi corelaţie pentru seriile staţionare pot fi enumţate astfel:

i. Dacă $\sum_{h=-\infty}^{\infty} |\gamma_X(h)| < \infty$ şi $v = \sum_{h=-\infty}^{\infty} \gamma_X(h)$ atunci $\overline{X}_n - \mu$ este $AN(0, \frac{v}{n})$ (AN=asimptotic normală). In particular acest lucru se întâmplă dacă $X_t = \mu + \sum_{j=-\infty}^{\infty} \psi_j Z_{t-j}$ cu $\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$ şi $(Z_t)_{t \in Z}$ este un zgomot alb.

ii. Dacă $X_t = \mu + \sum_{j=-\infty}^{\infty} \psi_j Z_{t-j}$ cu $\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$, $(Z_t)_{t\in Z}$ este un zgomot alb şi în plus $\sum_{j=-\infty}^{\infty} |j|\psi_j^2 < \infty$, atunci $\sum_{h=-\infty}^{\infty} |\gamma_X(h)| = \sigma^2\left(\sum_{j=-\infty}^{\infty} |\psi_j|\right)^2 < \infty$ şi

$$\begin{pmatrix} \hat{\rho}(1) \\ \hat{\rho}(2) \\ . \\ . \\ \hat{\rho}(m) \end{pmatrix} \text{ est AN}\left(\begin{pmatrix} \rho(1) \\ \rho(2) \\ . \\ . \\ \rho(m) \end{pmatrix}, \frac{1}{n} W\right)$$

unde matricea $W \in R^{m\times m}$ are componentele

$$\begin{aligned} w_{i,j} &= \sum_{k=-\infty}^{\infty} \left[\begin{matrix} \rho(k+i)\rho(k+j) + \rho(k-i)\rho(k+j) + 2\rho(i)\rho(j)\rho^2(k) - \\ -2\rho(i)\rho(k)\rho(k+j) - 2\rho(j)\rho(k)\rho(k+i) \end{matrix}\right] = \\ &= \sum_{k=1}^{\infty} (\rho(k+i) + \rho(k-i) - 2\rho(i)\rho(k)) \cdot (\rho(k+j) + \rho(k-j) - 2\rho(j)\rho(k)) \end{aligned} \tag{7.2.16}$$

(formula lui Bartlett).

In particular $n^{1/2}(\hat{\rho}(h) - \rho(h))$ are pentru n mare o distribuţie normală de medie 0 şi varianţă $w_{h,h}$.

In R funcţia de autocorelaţie empirică $\hat{\rho}$ este calculată cu funcţia *acf* care are forma:

```
acf(x, lag.max = NULL,
    type = c("correlation", "covariance", "partial"),
    plot = TRUE, na.action = na.fail, demean = TRUE, ...)
```

Rezultatul aplicării funcţiei *acf* este o listă ce conţine *lag*=intervale unde este estimat $\hat{\rho}$, *acf*=estimările pentru $\hat{\rho}$, *type*=la fel ca în apelul funcţiei acf, *n.used*=numărul de observaţii din serie, *series*=numele seriei de timp, snames=numele componentelor în o serie multivariată. Se vede din definiţie că funcţia *acf* determină şi corelaţia parţială dacă alegem corespunzător parametrul *type*. Dacă alegem *plot=TRUE* atunci este tipărită funcţia de autocorelaţie împreună cu intervalul de încredere de 95%.

Pentru determinarea tipului de serie temporală între tipurile MA(q), AR(p), ARMA(p,q) procedăm astfel:

MA. Dacă funcţia de autocorelaţie dată de *acf* este nulă pentru intervale temporale de lungime k>q(sau cade în intervalul de încredere cu încrederea 95% în jurul lui zero, tipărit pe grafic de funcţia *acf*), atunci alegem modelul MA(q).

AR. Dacă funcţia de autocorelaţie parţială dată de *acf* este nulă pentru intervale temporale k>p(sau cade în intervalul de încredere cu încrederea 95% în jurul lui zero, tipărit pe grafic de funcţia *acf*), atunci alegem modelul AR(p).

Putem simula în R serii temporale de tip ARIMA(p,d,q) prin funcţia *arima.sim* care are forma:

```
arima.sim(model, n, rand.gen = rnorm, innov = rand.gen(n, ...),
     n.start = NA, start.innov = rand.gen(n.start, ...),...)
```

Doar parametrii model şi n sunt obligatorii, ceilalţi având valori implicite. Mai jos simulăm o serie AR(2) şi îi tipărimACF şi PACF. Scriptul este:

```
s1<-arima.sim(model=list(ar=c(-.3,-.4)), n=10000)
par(mfrow=c(1,2))
acf(s1)
acf(s1, type="partial")
```

Rezultatul este în figura următoare:

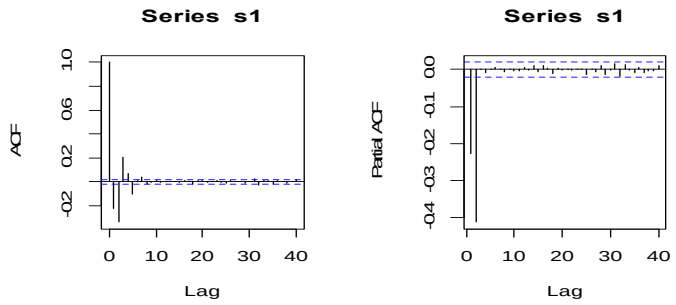


Fig. 7.5

Forma PACF sugerează un model AR(2). Acum simulăm un model MA(2) şi tipărim ACF+PACF:

```
s1<-arima.sim(model=list(ma=c(-.3,-.4)), n=10000)
par(mfrow=c(1,2))
acf(s1)
acf(s1, type="partial")
```

Rezultatul este:

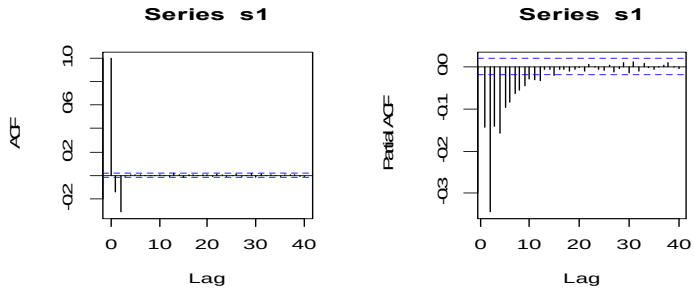


Fig. 7.6

Graficele sugerează un model MA(2).

ARMA. Pentru modelele ARMA(p,q) un indiciu asupra valorilor potrivite pentru p şi q este funcţia extinsă de autocorelaţie[21] (vezi [4], pag. 116). Funcţia extinsă de autocorelaţie EACF se calculează pentru parametrii *x*=serie temporală, p,q=numere naturale în intervalele [0, pmax] respectiv [0, qmax]. Rezultatul funcţiei este o mulţime de coeficienţi calculaţi după un algoritm complicat dar care ar trebui să formeze în octantul unde p şi q sunt coordonate un triunghi de zerouri cu vârful în valorile actuale pentru p şi q ai seriei ARMA(p,q). Funcţia *eacf* se găseşte în pachetul TSA şi are forma:

```
eacf(z, ar.max = 7, ma.max = 13)
```

Apelată pe o serie ARMA(2,2) simulată ne dă următorul rezultat:

```
s1<-arima.sim(model=list(ma=c(-.3,.5), ar=c(0.4,-0.2)),
n=10000)
par(mfrow=c(1,2))
acf(s1)
acf(s1, type="partial")
eacf(s1)
```

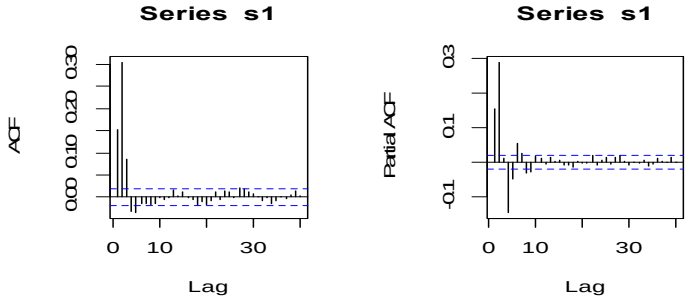


Fig. 7.7

> eacf(s1)

AR/MA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	x	x	x	x	x	o	o	o	o	o	o	o	o	o
1	x	x	x	x	o	o	o	o	o	o	o	o	o	o
2	x	x	o	x	x	o	o	o	o	o	o	o	o	o
3	x	x	x	o	o	o	o	o	o	o	o	o	o	o
4	x	x	x	x	o	o	x	o	o	o	o	o	o	o
5	x	x	x	o	x	o	x	o	o	o	o	o	o	o
6	x	x	x	o	x	x	o	x	o	o	o	o	o	o
7	x	x	o	x	x	x	x	x	o	x	o	o	o	o

Pe graficele ACF şi PACF seria nu pare AR sau MA. EACF sugerează un model ARMA(2,2), ARMA(1.4) sau MA(6). Dacă se rulează exemplul din nou se obţine altă figură dată de *eacf* din cauză că simularea produce alte valori pentru serie.

O altă metodă de determinare a parametrilor p şi q este de a propune toate valorile posibile pentru p şi q în intervalele [0, pmax] respectiv [0, qmax] ca în cazul

[21] Tsay, R. S. and Tiao, G. (1984). "*Consistent estimates of autoregressive parameters and extended sample autocorrelation function for stationary and nonstationary ARMA Models.*" Journal of the American Statistical Association, 79, 385, 84–96.

eacf şi de a determina coeficienţii modelului (vezi mai departe). Valorile p şi q optime se determină prin criteriile AIC, BIC etc. (vezi mai departe).

b. Estimarea parametrilor

După determinarea tipului ARMA(p,q) urmează estimarea parametrilor. Pentru modelele AR(p) *metoda momentelor* (Yule-Walker) este eficace. Această metodă constă în rezolvarea sistemului (7.2.15) în necunoscutele $\phi_1,\phi_2,...\phi_p$ dacă $\rho_1,\rho_2,...\rho_p$ se înlocuiesc cu estimările lor $\hat{\rho}_1,\hat{\rho}_2,...\hat{\rho}_p$. Pentru modele ce conţin şi parte medie mobilă metoda este mult mai complicată.

Metoda celor mai mici pătrate constă în a scrie:

$$Z_t = X_t - \phi_1 X_{t-1} - \phi_2 X_{t-2} - ..\phi_p X_{t-p} - \theta_1 Z_{t-1} - \theta_2 Z_{t-2} - ... - \theta_q Z_{t-q} \quad (7.2.17)$$

şi a minimiza:

$$S(\phi,\theta) = \sum_{t=p+1}^{T} (Z_t)^2 \quad (7.2.18)$$

considerând $Z_p, Z_{p-1},...Z_{p-q+1}$ egale cu 0. Metoda este aplicabilă şi pentru serii multidimensionale.

Metoda cea mai utilizată este *metoda verosimilităţii maxime* deşi metoda celor mai mici pătrate a lui Hannan şi Rissanen poate fi îmbunătăţită încât să aibă aceeaşi eficienţă asimptotică. Funcţia de verosimilitate are o expresie complicată şi poate fi calculată cu ajutorul algoritmului inovaţiilor. Detalii se pot găsi în cărtile menţionate ale lui Brockwell şi Davis de exemplu[22], demonstraţiile găsindu-se în cartea pentru matematicieni[23]. Tot prin metoda verosimilităţii maxime se poate stabili care sunt valorile optime pentru parametrii p şi q în modelul ARMA. Pentru aceasta se estimează parametrii $\phi_1,\phi_2,..\phi_p$, $\theta_1,..\theta_q$ şi σ^2 prin metoda verosimilităţii maxime şi apoi se calculează criteriul AICC:

$$AICC = -2\log\left(L\left(\phi_1,\phi_2,..\phi_p,\theta_1,..\theta_q,\sigma^2\right)\right) + 2(p+q+1)/(n-p-q-2) \quad (7.2.19)$$

In acest criteriu $L\left(\phi_1,\phi_2,..\phi_p,\theta_1,..\theta_q,\sigma^2\right)$ este funcţia de verosimilitate a cărei expresie

$$L\left(\phi,\theta,\sigma^2\right) = \frac{1}{\left(2\pi\sigma^2\right)^{n/2}} \left(r_0 r_1 ... r_{T-1}\right) e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{T}\frac{(X_j-\hat{X}_j)^2}{r_{j-1}}} \quad (7.2.20)$$

depinde $\hat{X}_1,\hat{X}_2,...\hat{X}_T$ şi r_0, r_1, r_{T-1} calculaţi prin algoritmul inovaţiilor (vezi citările de la subsol). Criteriul constă în a alege parametrii p şi q care minimizează AICC. Penalizarea $2(p+q+1)/(n-p-q-2)$ determină ca valorile optime pentru p şi q să nu fie foarte mari.

In sistemul R există funcţii care determină parametrii pentru modelele AR sau ARIMA (deci şi ARMA). Pentru modelul AR avem funcţia:

---

[22] P. J. Brockwell, R. A. Davis. *Introduction to Time Series and Forecasting*. Springer-Verlag New York Berlin Heidelberg, 2002, pag. 156-162

[23] P.J. Brockwell, R.A. Davis, (1991), *Time Series: Theory and Methods*, 2nd Edition, Springer-Verlag, New York, cap. 8

```
ar(x, aic = TRUE, order.max = NULL,
   method=c("yule-walker", "burg", "ols", "mle", "yw"),
   na.action, series, ...)
```

Metoda "yule-walker" este metoda momentelor, "ols" este metoda celor mai mici pătrate, iar "mle" este metoda verosimilităţii maxime. Sunt calculate modelele AR până la ordinul order.max şi se alege modelul cu aic minim. Seria de timp x poate fi şi multidimensională.

Rezultatul funcţiei *ar* este o listă ce conţine: `order, ar, var.pred, x.mean, x.intercept, aic, n.used, partialacf, order.max, resid, method,series, call, asy.var.coef`. Cu comanda *help(ar)* se pot obţine informaţii asupra smnificaţiei acestor elemente. Avem de exemplu *order*=ordinul p al modelului AR determinat, *resid*=diferenţele $X_t - \phi_1 X_{t-1} - ... - \phi_p X_{t-p}$, *ar*=coeficienţii $\phi_1,...\phi_p$ estimaţi.

Iată un model de aplicare a funcţiei *ar*. Se simulează un model AR şi apoi se estimează parametrii săi fară a cunoaşte lungimea p a lor. Scriptul este mai jos:

```
s2<-arima.sim(model=list(ar=c(0.4,-0.2)), n=1000)
m1<-ar(s2,order.max=5,AIC=T,method='yw')
m2<-ar(s2,order.max=5,AIC=T,method='ols')
m3<-ar(s2,order.max=5,AIC=T,method='mle')
```

Rezultatele se văd mai jos în consola R:

```
> m1

Call:
ar(x = s2, order.max = 5, method = "yw", AIC = T)

Coefficients:
     1       2
 0.390  -0.209

Order selected 2  sigma^2 estimated as  1.029
> m2

Call:
ar(x = s2, order.max = 5, method = "ols", AIC = T)

Coefficients:
      1        2
 0.3905  -0.2092

Intercept: -0.0003215 (0.03208)

Order selected 2  sigma^2 estimated as  1.027
> m3

Call:
ar(x = s2, order.max = 5, method = "mle", AIC = T)

Coefficients:
      1        2
```

```
 0.3901  -0.2089

Order selected 2  sigma^2 estimated as  1.026
```

Vedem că a fost determinat corect ordinul p=2 şi s-au determinat aproximativ corect coeficienţii.

Pentru estimarea modelelor ARMA(p,q) sau ARIMA(p,d,q) avem funcţia *arima* în pachetul stats, care are forma:

```
arima(x, order = c(0, 0, 0),
      seasonal = list(order = c(0, 0, 0), period = NA),
      xreg = NULL, include.mean = TRUE,
      transform.pars = TRUE,
      fixed = NULL, init = NULL,
      method = c("CSS-ML", "ML", "CSS"),
      n.cond, optim.method = "BFGS",
      optim.control = list(), kappa = 1e6)
```

Parametrii sunt *x*: o serie temporală univariată sau un vector numeric, *order*: specificare a tipului de modelare, *seasonal*: specificare a părţii sezoniere, *xreg*: variabile externe faţă de care se face o regresie liniară şi se modelează apoi ARIMA rezidualii, *method*: metoda de determinare a parametrilor (implicită "CSS"=metoda celor mai mici pătrate), *optim:* este metoda utilizată pentru minimizare (implicită="BFGS", adică Broyden, Fletcher, Goldfarb and Shanno). Mai multe detalii cu help(arima). Rezultatul funcţiei ARIMA este o listă arima ce conţine:

`coef`: o listă cu coeficienţii
`sigma2`: varianţa σ^2 a variabilelor Z_t din modelul ARMA(p,q)
`residuals`: valorile Z_t aşa cum rezultă din estimare
`arma`: specificarea modelului
`aic: coeficientul Akaike`, valabil doar pentru metoda ML (ver. maximă)
`code`: codul de convergenţă întors de funcţia *optim*; dacă code=0 atunci optimizarea a decurs normal, iar pentru alte coduri se poate utiliza comanda help(optim)
`series`: numele seriei x

Ceilalţi parametri de ieşire pot fi consultaţi cu *help(arima)*.

Următorul script generează o seria ARMA(2,2) şi apoi se încearcă cu funcţia *arima* să se determine modelul.

```
s1<-arima.sim(model=list(ma=c(-.3,.5), ar=c(0.4,-0.2)),
n=10000)
model1<-arima(s1,order=c(2,0,2),method="ML")
```

Rezultatul este:

```
> model1

Call:
```

```
arima(x = s1, order = c(2, 0, 2), method = "ML")

Coefficients:
         ar1      ar2      ma1     ma2  intercept
      0.4012  -0.2091  -0.3021  0.4996    -0.0040
s.e.  0.0310   0.0295   0.0279  0.0243     0.0147

sigma^2 estimated as 0.9877:  log likelihood = -14127.79,
aic = 28265.57
```

Se vede că practic au fost determinaţi corect coeficienţii ar şi ma. Coeficientul *intercept* din lista de coeficienţii este media variabilelor X_t ale seriei de timp (probabil ar trebui să se schimbe denumirea în *mean*).

O extensie a funcţiei *arima* din pachetul stats se găseşte în pachetul TSA cu numele tot *arima*. Cititorul interesat poate găsi informaţiile cu ajutorul comenzii help din R.

Până la versiunea 2.14.0 nu există variante de modelare pentru serii de timp ARIMA(p,d,q) multidimensionale. De asemenea trebuie menţionat că modelul ARMA trebuie să fie cauzal (modelele necauzale sunt echivalente cu modele cauzale dar cu alt zgomot alb). De asemenea se poate considera modelul invertibil pentru că pentru orice model neinvertibil există unul invertibil cu aceeaşi funcţie de sutocorelaţie (care determină în final coeficienţii modelului). Detalii se găsesc în cărţiele menţionate ale lui Brockwell şi Davis.

c. Validarea modelului (diagnosticul veridicităţii modelului ales)

Pentru validarea modelului trebuie verificat că rezidualii produşi de modelele statistice sunt variabile aleatoare independente de medie zero şi aceeaşi varianţă σ^2. De regulă se fac teste pentru a verifica dacă valorile reziduale Z_t provin din v.a. indepnedente identic distribuite. Pentru aceasta există un număr de teste.

1. *Funcţia de autocorelaţie* $\hat{\rho}_n$ ar trebui să se anuleze pantru $n \geq 1$. In R acest lucru este cercetat prin funcţia *acf*.

2. *Testul rangului.* Dacă $y_1, y_2, ... y_n$ sunt realizări ale unei serii i.i.d. atunci fie P numărul de perechi (i,j) astfel ca i>j şi $y_i > y_j$. Pentru n mare P are o repartiţie aproximativ $N(m, \sigma^2)$ cu $m = \frac{n(n-1)}{4}$, $\sigma^2 = n(n-1)(2n+5)/72$

3. *Testul Ljung–Box* constă în a calcula: $Q(h) = n(n+2)\sum_{t=1}^{h} \frac{\hat{\rho}(t)}{n-t}$. Pentru n mare şi y_i v.a. iid, Q(h) are o repartiţie aproximativ χ^2 cu h grade de libertate.

4. Testul s*emnului diferenţelor*. In acest test S este numărul de valori i pentru care $y_i > y_{i-1}$. Pentru n mare şi valorile y_i provenind de la v.a. iid, S are o repartiţie aproximativ $N(m, \sigma^2)$ cu $m = \frac{n-1}{2}$, $\sigma^2 = \frac{n+1}{12}$.

5. *Testul punctelor de întoarcere*. La momentul i avem un punct de întoarcere dacă $y_{i-1} < y_i > y_{i+1}$ sau $y_{i-1} > y_i < y_{i+1}$. Fie T numărul punctelor de întoarcere. Dacă n este mare şi valorile y_i provin de la v.a. iid atunci T are o repartiţie aproximativ $N\left(\frac{2(n-2)}{3}, \frac{16n-29}{90}\right)$

In R există funcţia *tsdiag* care tipăreşte rezidualii standardizaţi (la varianţă 1), calculează funcţia de autocorelaţie a rezidualilor şi calculeză testul Box (sau Ljung–Box) până la un h dat. Forma funcţiei este:

```
tsdiag(object, gof.lag, ...)
```

unde *object* este o serie temporală modelată în R (deci are rezidualii calculaţi), *gof.lag* este numărul maxim de paşi h (vezi testul 3 (Ljung–Box) până la care se calculează Q(h).

In scriptul următor se simulează o serie ARMA(2,2) căreia i se determină parametrii cu funcţia *arima* şi apoi cu funcţia *tsdiag* se studiază dacă modelul este valid.

```
s1<-arima.sim(model=list(ma=c(-.3,.5),      ar=c(0.4,-0.2)),
n=10000)
model1<-arima(s1,order=c(2,0,2),method="ML")
tsdiag(model1)
```

Obţinem următoarele reprezentări grafice din *tsdiag*:

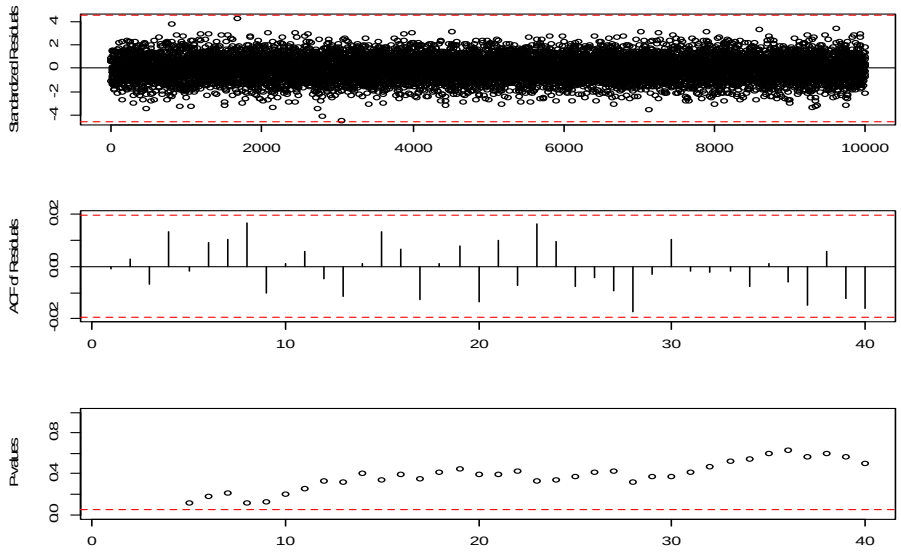


Fig. 7.8

Se vede că *acf* este suficient de mică pentru putea fi considerată zero, iar testul Ljung–Box indică faptul că în general pentru aproape toţi h p-valoarea testului este

suficient de mare pentru a admite că rezidualii sunt iid. Acum dacă cercetăm modelul determinat găsim:

```
> model1

Call:
arima(x = s1, order = c(2, 0, 2), method = "ML")

Coefficients:
         ar1      ar2      ma1     ma2  intercept
      0.3831  -0.1976  -0.2773  0.5047     0.0016
s.e.  0.0289   0.0288   0.0258  0.0239     0.0152

sigma^2 estimated as 1.018:   log likelihood = -14276.47,   aic =
28562.93
```

Vedem că au fost estimaţi destul de bine coeficienţii cu care s-a simulat seria ARMA.

d. Predicţii

După validarea modelului de serie de timp acesta poate fi utilizat pentru predicţia valorilor viitoare. In general prntru prezicerea valorilor unei serii temporale unde cunoaştem $(X_t)_{t=0,1,...T}$ pentru valori t>T definim predicţia $\widehat{X}_{T+h}$ ca proiecţia lui X_{T+h} pe spaţiul generat de $(X_t)_{t=0,1,...T}$. Această proiecţie poate fi calculată ţinând seama de produsele scalare conţinute în funcţia de autocorelaţie. Pentru seriile staţionare calculul conduce la rezolvarea unui sistem asemănător cu (7.2.15) unde ϕ sunt necunoscute (sistemul Yule-Walker). Acest sistem poate fi rezolvat recursiv prin algoritmul Durbin-Levinson. O altă metodă de determinare a proiecţiei valabilă şi pentru serii nestaţionare este algoritmul inovaţiilor. Detalii se pot găsi în cărţile citate ale lui Brockwell şi Davis. In sistemul R există funcţia *predict* care pe baza modelului (în general ARIMA) face predicţia valorilor pentru un număr de paşi temporali daţi. Forma funcţiei este:

```
predict(object, n.ahead = 1, newxreg = NULL,
        se.fit = TRUE, ...)
```

In funcţie de tipul de obiect R (*lm, ts, glm, , loess, nls, poly, princomp, spline*) căruia i se aplică funcţia predict poate avea argumente diferite în partea ..... Pentru predicţia seriilor ARIMA *object* din funcţia *predict* trebuie să fie un obiect rezultat al funcţiei *arima.* Următorul script produce o predicţie pentru 20 paşi în avans pentru o seria ARMA(2,2) simulată ca în exemplul de precedent.

```
s1<-arima.sim(model=list(ma=c(-.3,.5), ar=c(0.4,-0.2)), n=1000)
model1<-arima(s1,order=c(2,0,2),method="ML")
#tsdiag(model1)
p1<-predict(model1, n.ahead=50)
p1
plot(s1, main="Seria s1 de 1000 masuratori si predictia pentru 50
pasi", xlab="Timpul")
lines(p1$pred, col=5,lwd=3)
```

Obţinem următorul rezulatat la consolă

```
$pred
Time Series:
Start = 1001
End = 1050
Frequency = 1
 [1]  0.30741965 -0.01205071 -0.15271697 -0.10329277 -0.04444154 -0.03973509
 [7] -0.05590263 -0.06270560 -0.06009920 -0.05718103 -0.05699457 -0.05781187
[13] -0.05814027 -0.05800335 -0.05785878 -0.05785189 -0.05789317 -0.05790899
[19] -0.05790182 -0.05789466 -0.05789444 -0.05789652 -0.05789728 -0.05789691
[25] -0.05789656 -0.05789655 -0.05789666 -0.05789669 -0.05789667 -0.05789666
[31] -0.05789666 -0.05789666 -0.05789666 -0.05789666 -0.05789666 -0.05789666
[37] -0.05789666 -0.05789666 -0.05789666 -0.05789666 -0.05789666 -0.05789666
[43] -0.05789666 -0.05789666 -0.05789666 -0.05789666 -0.05789666 -0.05789666
[49] -0.05789666 -0.05789666

$se
Time Series:
Start = 1001
End = 1050
Frequency = 1
 [1] 0.9632667 0.9663791 1.0064598 1.0089120 1.0107742 1.0116303 1.0116409
 [8] 1.0117388 1.0117442 1.0117491 1.0117511 1.0117512 1.0117514 1.0117514
[15] 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514
[22] 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514
[29] 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514
[36] 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514
[43] 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514 1.0117514
[50] 1.0117514
```

Obţinem următorul grafic cu valorile seriei şi valorile prezise:

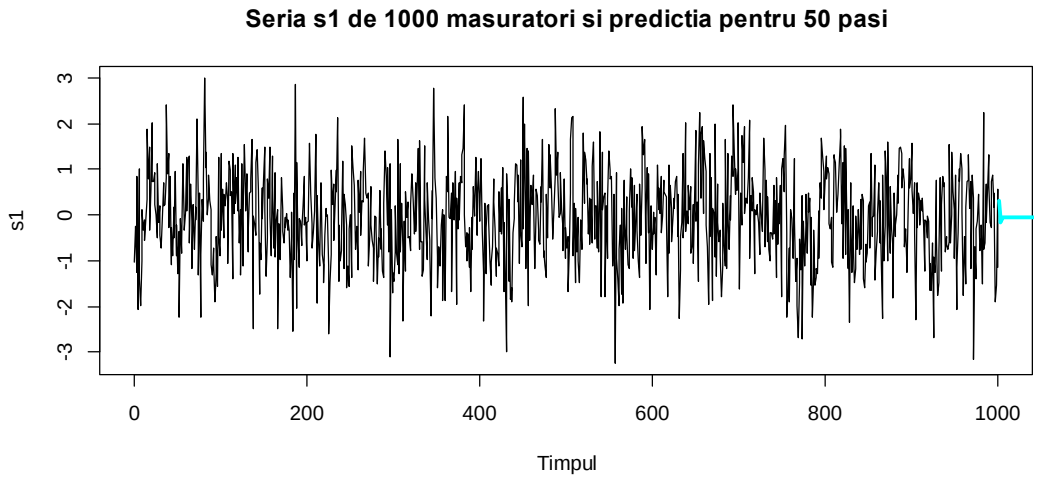


Fig. 7.9

Constatăm că predicţiile produse de model nu urmează variaţiile seriei.

Aplicăm aceeaşi tehnică de tratare pentru seria reală de CO menţionată la începutul secţiunii de serii de timp obţinem:

```
>co=read.table(file.choose())
>tsco<-ts(co, frequency=168)
> plot(tsco, main="CO intre 1 ian.2008 si 30. iun. 2008",
+xlab="Timpul in saptamani")
```

Graficul seriei este în figura următoare:

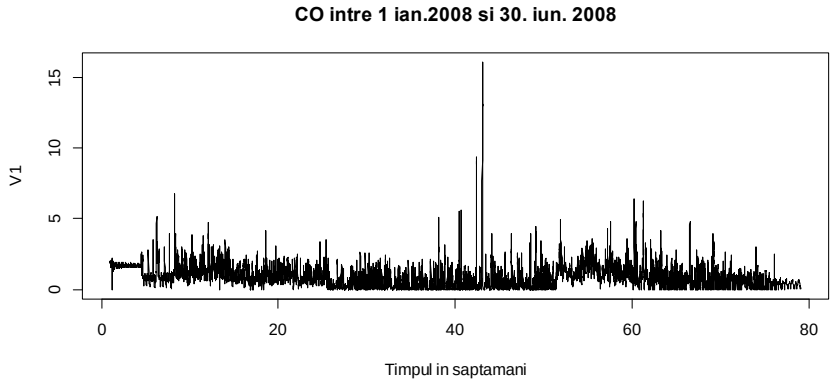


Fig. 7.10

Facem graficul pentru ACF şi PACF pentru a vedea ce model ar fi potrivit.

```
> acf(tsco)
```

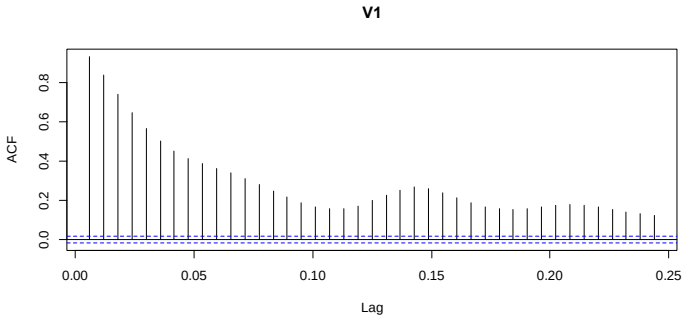


Fig. 7.11
Constatăm că ACF nu se anulează deci nu poate fi un model MA

Calculăm PACF:

```
> acf(tsco, type="partial")
```

Obţinem:

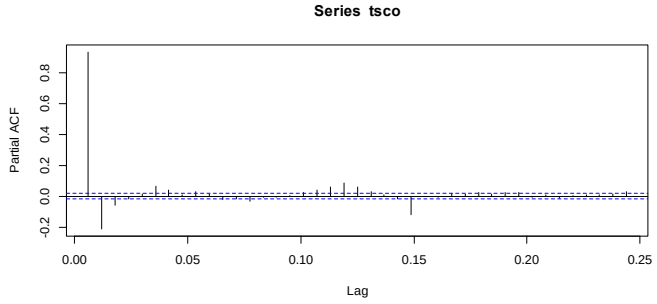


Fig. 7.12
Figura sugerează un model AR

Incercăm şi cu EACF:

```
> eacf(tsco, ar.max = 20, ma.max = 20)
```

Obţinem:

```
AR/MA
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
0  x x x x x x x x x x x  x  x  x  x  x  x  x  x  x  x
1  x x x o x x x x x x x  x  x  x  o  x  x  x  x  x  o
2  x o x o x x x x x x o  x  o  x  o  o  o  o  x  x  o
3  x o o x x x o o x x o  x  o  o  o  o  o  o  o  o  o
4  x x x x x x o x x o o  o  o  o  o  o  o  o  o  o  o
5  x x x x o o o o x x o  o  x  o  o  o  o  o  o  o  o
6  x x x x x o o o x x o  o  o  o  o  o  o  o  o  o  o
7  x x x x x o o x x o x  o  x  o  o  o  o  o  o  o  o
8  x x x x x o o x x o o  o  o  o  o  o  o  o  o  o  o
9  x x x x x x x x x o o  o  x  o  o  o  o  o  o  o  o
10 x x x x x x x x x o o  o  x  o  o  o  o  o  o  o  o
11 x x x x x x x x x o o  o  x  x  o  o  o  o  o  o  o
12 x x x x o x x x x x o  o  o  o  o  o  o  o  o  o  o
13 x x x x o x x x x x o  x  o  o  o  o  o  o  o  o  o
14 x x x x x x x x x x x  x  o  o  o  o  o  o  o  o  o
15 x x o x x o x x x x x  x  x  x  o  o  o  o  o  o  o
16 x x x x x x x x x x o  o  x  x  x  o  o  o  o  o  o
17 x x o x x o x x o x x  o  o  x  x  x  o  o  o  o  o
18 x x o x o o x x x x o  o  o  x  x  o  o  o  o  o  x
19 x o x x o x o o o x x  o  o  x  x  x  x  x  o  o  o
20 x x x x x x o o o o x  x  o  x  x  x  x  x  x  o  o
>
```

Tabloul obţinut ar sugera un model ARMA(p,q) cu p în jur de 7-11 şi q 9 sau 10. Dacă nu suntem prea pretenţioşi ar merge parcă p şi q în jur de 5. Incercând am obţinut mesajul că sunt probleme de convergenţă. Incercăm cu valori mai mici.

```
> m1<-arima(tsco,order=c(3,0,2),method="ML")
> tsdiag(m1)
```

Obţinem:

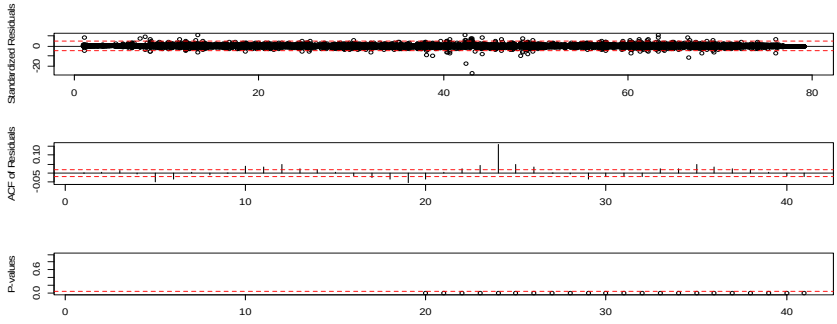


Fig. 7.13

ACF arată că există corelaţii semnificative între reziduali pentru diferenţe mari de timp. Mergem totuşi mai departe. Predicţia dată de model arată astfel:

```
p1=predict(m1,n.ahead=168)
plot(p1$pred, col=5,lwd=3, main="Predictii CO")
```

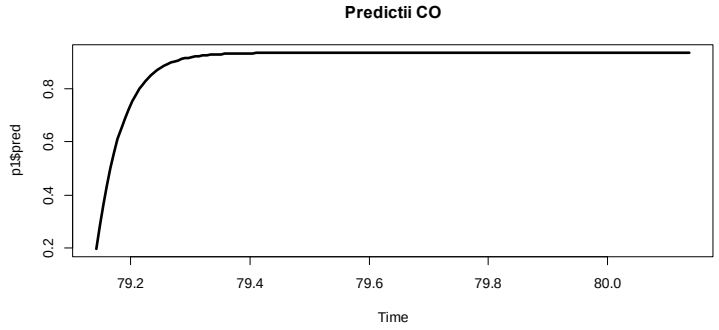


Fig. 7.14

Predicţiile pentru CO nu mai urmăresc variaţiile constatate în modelul real. Se pare că nu am găsit modelul potrivit de serie temporală.

In general aplicate la modele reale teoria nu se potriveşte întotdeauna cu realitatea. In secţiunea următoare prezentăm o altă tehnică de analiză a seriilor de timp care s-a dovedit mai bună pentru aceste date (CO).

7.2.2 Serii temporale deterministe şi haos

Metoda sistemelor haotice, aşa cum este descrisă în lucrarea,[24] permite într-o anumită măsură descrierea şi predicţia evoluţiei unor serii de timp. O implementare practică a acestor metode se găseşte în pachetul gratuit de programe TISEAN disponibil publicului pe http://www.mpipks-dresden.mpg.de/ ~ tisean (Institutul Max Planck pentru Fizică, Dresda). Autorii pachetului îl descriu într-o lucrare[25] disponibilă liber pe internet.

Ideea de bază a metodei este că datele urmează o evoluţie deterministă

$$\bar{x}' = \bar{X}(x) \tag{7.2.21}$$

cu $\bar{x}$ într-un anumit spaţiu normat finit sau infinit dimensional E, dupa o anumită perioada de timp traiectoria sistemului dinamic (7.2.21) evoluează în jurul unui atractor A. Dinamica în A poate fi surprinsă efectiv de aproape orice funcţie $s : E \to R$.

Fie $\Phi(t,x)$ fluxul lui (7.2.21) pentru o valoare fixată a lui $t=\tau$. Fie $F(\bar{x}) = \Phi(\tau,\bar{x})$, $F^{-1}(\bar{x}) = \Phi(-\tau,\bar{x})$, şi fie $s : U \to R$ definită într-o vecinatate a lui A. Atunci, conform unei teoreme a lui F. Takens[26] în condiţii generale pentru s şi A şi pentru un $m \in N$ suficient de mare, aplicaţia

$$\begin{aligned} &S : A \to R^m \\ &S(\bar{x}) = \left(s\left(F^{-(m-1)}(\bar{x})\right),\ s\left(F^{-(m-2)}(\bar{x})\right),\ldots s\left(F^{-1}(\bar{x})\right),\bar{x}\right) \end{aligned} \tag{7.2.22}$$

este o scufundare a lui A în R^m.

Pe imaginea atractorului $S(A) \subset R^m$ avem

$$S\left(\Phi(n\tau,\bar{x}_0)\right) = \bar{s}_n = \left(s_{n-(m-1)}, s_{n-(m-2)} \ldots s_{n-1}, s_n\right) \tag{7.2.23}$$

Dinamica lui F de pe atractorul A este exprimată pe imaginea sa S(A) prin

$$\bar{s}_n \to G(\bar{s}_n) = \bar{s}_{n+1}$$

ca în figura următoare:

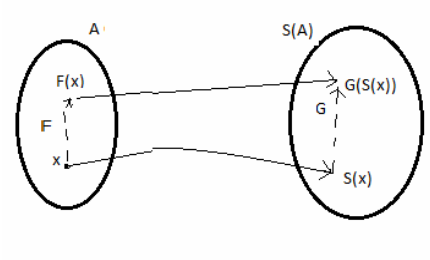


Fig. 7.15

O uşoară modificare a lui (7.2.233) introducând o anumită întârziere este $d \in N^*$ conduce la

$$\bar{s}_n = \left(s_{n-(m-1)d}, s_{n-(m-2)d} \ldots s_{n-d}, s_n\right) \tag{7.2.24}$$

---

[24] Eckmann, J.P., D. Ruelle, 1985. *Ergodic theory of chaos and strange attractors,* Rev. Mod. Phys. 57, pp. 617-656

[25] Rainer Hegger, Holger Kantz, and Thomas Schreiber, 1999. *Practical implementation of nonlinear time series methods: The TISEAN package*, Chaos 9,.pp. 413-435.

[26] H. D. I. Abarbanel, Reggie Brown, John J. Sidorowich, and Lev Sh. Tsimring, 1993. *The analysis of observed chaotic data in physical systems*, Rev M Phys, 65, pp. 1331-1392.

Intârzierea optima sugerată de Fraser şi Swinney pentru (7.2.24) este recomandată ca fiind primul minim local al informaţiei mutuale empirice:

$$H = \sum_{i,j} p_{i,j}(d) \ln\left(\frac{p_{i,j}(d)}{p_i\, p_j}\right) \tag{7.2.25}$$

Probabilitatile p_i şi $p_{i,j}(d)$ în (7.2.25) sunt obţinute prin partiţionarea datelor $(s_n)_{1\le n\le T}$ într-un numar de subintervale al valorilor datelor măsurate: p_i este probabilitatea ca funcţia s să ia valori în subintervalul i iar $p_{i,j}(d)$ este probabilitatea de tranziţie de la subintervalul i la subintervalul j după d unităţi.

Pentru a determina m, dimensiunea de scufundare, folosim metoda celui mai apropiat vecin fals (vezi referinţele în subsolul paginii precedente). Ideea este că dacă $\bar{s}_j$ este cel mai apropiat vecin al lui $\bar{s}_i$ în R^m atunci $\bar{s}_{i+1} = G(\bar{s}_i)$ şi $\bar{s}_{j+1} = G(\bar{s}_j)$ sunt de asemenea apropiaţi şi raportul $R_i = \| \bar{s}_{i+1} - \bar{s}_{j+1} \| / \| \bar{s}_i - \bar{s}_j \|$ este mai mic decât un prag euristic Rt. In caz contrar punctul $\bar{s}_j$ este marcat ca vecin fals. Dacă procentul de puncte cu un cel mai apropiat vecin fals este prea mare, consideram ca m este prea mică.

Metode de predicţie

Predicţia de ordin zero este data de

$$s_{n+k} = \frac{1}{card(U_n)} \sum_{\bar{s}_j \in U_n} s_{j+k} \tag{7.2.26}$$

unde U_n este o mică vecinătate a lui $\bar{s}_n$ în R^m. Predicţia este făcută pentru $n+k>T$, ultima măsuratoare.

Predicţia local liniară este dată de formula

$$s_{n+1} = \bar{a}_n \bar{s}_n + b_n \tag{7.2.27}$$

unde coeficienţii $\bar{a}_n \in R^m$, $b_n \in R$ sunt determinaţi astfel încât

$$\sigma^2 = \sum_{\bar{s}_j \in U_n} \left(s_{j+1} - \bar{a}_n \bar{s}_j - b_n\right)^2 \tag{7.2.28}$$

să fie minimă. Suma este efectuată după toţi $\bar{s}_j$ dintr- o mică vecinătate U_n a lui $\bar{s}_n$ dar destul de mare încât să conţină suficiente puncte pentru a asigura ca problema celor mai mici patrate (7.2.28) să fie nesingulară. In lucrările citate în pagina anterioară se găsesc şi alte metode de predicţie.

Aşa cum am mai menţionat aceste metode au fost implementate de Rainer Hegger, Holger Kantz şi Thomas Schreiber în pachetul TISEAN care este disponibil şi în R. Pentru a putea fi folosit în R trebuie ca pachetul TISEAN cu executabilele să fie dezarhivat într-un director. Pe urmă trebuie instalat în R pachetul RTisean care conţine apeluri în stilul R pentru executabilele din TISEAN. La primul apel al unei funcţii din RTisean se cere directorul cu executabilele TISEAN. Informaţii despre funcţiile din pachet pot fi obţinute prin ??RTisean. Acest pachet de analiză a seriilor temporale a fost utilizat pe date de poluare obţinute în municipiul Bucureşti, în

particular pentru concentraţia de CO al cărei grafic este în Fig. 7.1. Reproducem[27] câteva rezultate obţinute în predicţia poluării utilizând tehnici din această secţiune. Calculele au fost făcute cu rutinele TISEAN apelate direct (nu din R, dar o examinare a fişierului de help din RTisean arată că apelul din R se face cu aceeaşi parametri ca şi apelul direct din consola windows). Rezultatele au fost salvate în fişiere text de unde au fost trecute în Excel pentru examinare şi pentru reprezentare grafică. Utilizând RTisean nu mai este nevoie de acest şir de operaţii care încetineşte mult analiza de variante. Mai jos reproducem câteva grafice unde sunt reprezentate valorile măsurate şi cele prezise pe baza măsurătorilor anterioare. Predicţiile sunt pentru o săptămână (168 ore).

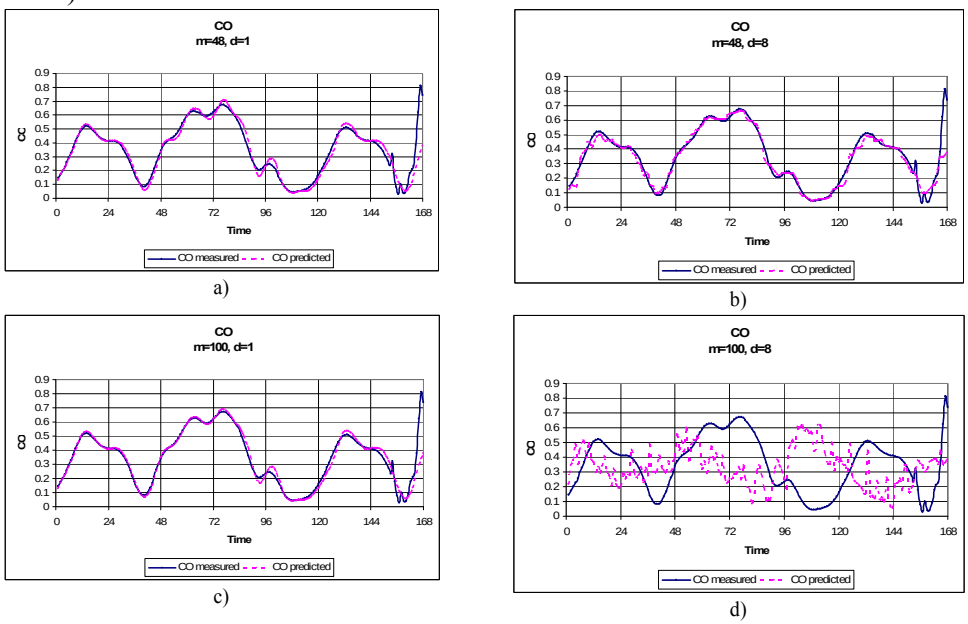


Fig. 7.16

Concentrţia de CO este nefiresc de bine estimată prin predicţia 7.2.26 pentru o dimensiune de scufundare m>=48. Contrar cu teoria, la o întârziere d=1 predicţia e mai bună decât pentru d=8, unde este primul minim al informaţiei mutuale empirice 7.2.25. Rutina TISEAN utilizată a fost lzo (în R se numeşte lzo.test).

Nu au fost la fel de bune predicţiile pentru alţi poluanţi. Pentru particulele PM 2.5 se vede din graficele de mai jos.

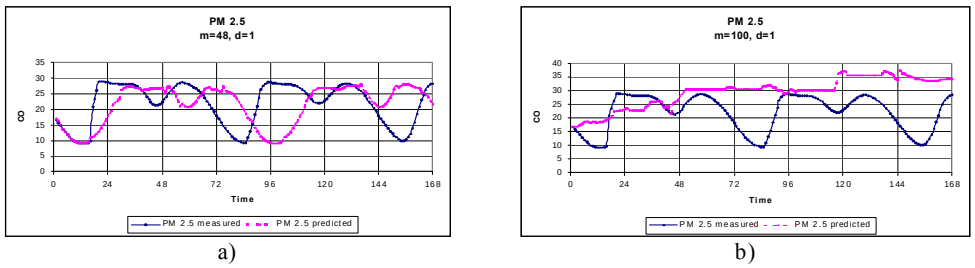
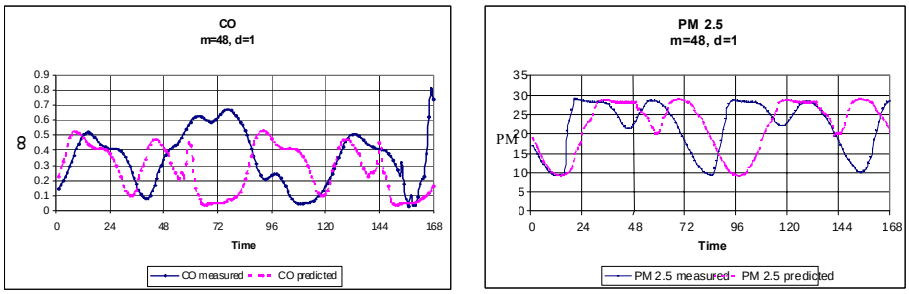


Fig. 7.17

Predicţiile pentru 168 de ore a poluării cu PM 2.5

[27]Viorel Petrehus, Ileana Armeanu, Camelia Slave, *Analysis of the Urban Air Pollution in Bucharest*, Proceedings BALCOR 2011, vol 2. pp 72-78

In momentul când s-au făcut predicţii simultane (funcţia *s* din (7.2.22) are valori în R^p, deci S are valori în R^{mp}, pentru p poluanţi consideraţi simultan) am constatat prin multe încercări că predicţiile nu devin mai bune. In figura următoare vedem ce se întâmplă dacă simultan luăm CO şi PM 2.5 în considerare. Datele măsurate pentru un an şi jumătate sunt utilizate pentru a prezice poluarea pe o săptămână.



a) **b)**

Fig. 7.18

Cele mai bune predicţii simultane sunt obţinute pentru d = 1 şi sunt semnificative numai pentru un interval de timp mai mic de 20 de ore

Din experimentele numerice am ajuns la următoarele concluzii:

1. Întârzierea d = 1 este o alegere bună, uneori mai bună ca întârzierea la care informaţia mutuală empirică ajunge la un minim local.

2. O alegere bună pentru vecinătatea U_n, care determină câte puncte intră în calcul în formula (7.2.26) se află experimental. Pentru predicţia local liniară o alegere bună a numărului de puncte în formula (7.2.28) este de cel puţin patru ori numărul de parametri de estimat.

3. Alegerea dimensiunii de scufundare astfel încât fracţiunea de puncte având cel mai apropiat vecin fals este mai mica de 5% , dă rezultate bune. De asemenea, este indicat să alegem m, astfel ca vectorul $\bar{s}_n$ să cuprindă datele unei perioade în cazul în care se suspectează că datele au o anumită periodicitate.

4. Este posibil ca datele să nu respecte o lege deterministă. În acest caz, orice alegere a lui m şi d nu este o alegere bună.

5. Previziunile pe termen scurt sunt în general mai bune ca cele pe termen lung.

6. Prin experimentare numerică ajungem la un moment dat să găsim o dimensiune m de scufundare şi o întârziere d care utilizate pentru anumite date dintr-un anumit loc duc la previziuni acceptabile. Este de aşteptat ca acestea să reflecte particularităţi ale sistemului dinamic (7.2.21) din acel loc şi să fie valabile şi pentru alt set de măsurători. Acest fenomen, determinarea parametrilor de scufundare m şi d, ar fi analogul determinării parametrilor într-un model statistic (de exemplu ARMA(p,q)).

Bibliografie

**. The R Development Core Team, *R: A Language and Environment for Statistical Computing*, 2011-12-13.

1. M. Allehard. *A Tyny Handbook of R, Springer*, 2011-12-13.

2. Y. Aragon. *Séries Temporelles avec R. Méthodes et cas*. Springer, 2011.

3. M. Dumitrescu, A. Bătătorescu. *Applied Statistic using the R System*, Ed. Universităţii Bucureşti, 2006.

4. J. D. Cryer, Kung-Sik Chan. *Time Series Analysis, with Applications in R*, Springer, 2008.

5. J. Maindonald, W. J. Braun. *Data Analysis and Graphics using R – an Example-Based Approach*, Cambridge University Press, 2003

6. Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

7. A. Păun, M. Păun. *Analiză Statistică folosind limbajul R*, Editura Matrix Rom, Bucureşti, 2009.

8. V. Petrehuş, S.A. Popescu, „Probabilităţi şi Statistică", Universitatea Tehnică de Construcţii, Bucureşti, 1997, civile.utcb.ro/cmat/cursrt/psvp.pdf.

9. D. Ruppert. *Statistics and Data Analysis for Financial Engineering*. *Use R!* Springer, 2010.

10. G. Zbăganu. *Metode Matematice în Teoria Riscului şi Actuariat,* Editura Universităţii Bucureşti, 2004.

.