Віктор Гнатюк

Вступ до R на прикладах

Харківський Національний Економічний Університет

Харків 2010

Зміст

1

Знайомство з R

R - мова і середовище програмування орієнтовані, в першу чергу, на статистичні обрахунки, написання різного роду програм обробки, аналізу даних та представленні результатів в графічному вигляді. R є безкоштовним програмним середовищем з відкритим кодом, що розповсюджується на основі ліцензії GNU General Public License (заснованою Free Software Foundation)[1] і знаходиться у вільному доступі. Програми написані на R запускаються на більшості платформ і операційних систем - FreeBSD, Linux, MacOS, Windows.

Проект R був ініційований працівниками Оуклендського університету Росом Іхакою та Робертом Джентлеменом (Ross Ihaka, Robert Gentleman University of Auckland, New Zealand) на початку 90-х і є діалектом більш раньої мови програмування S розробленою Bell Laboratories на чолі з Джоном Чемберсом (John Chambers) та колегами. Існує певна вімінність між програмними середовищами, однак програмний код написаний в S, в переважній більшості без змін буде виконуватися в R. Середовище R містить широку гаму статистичних методів та функцій (лінійний і нелінійний регресійний аналіз, статистичні тести, аналіз часових рядів, кластеризації і багато іншого), графічних інструментів і є значно гнучкішим ніж інші статистичні програмні продукти, оскільки користувачі постійно можуть розширювати функціонал зарахунок написання нових функцій. Відповідні пакети, що реалізують нові функції і розширюють можливості R розміщуються в онлайн колекції пакетів R. В мережі Internet на сайті Comprehensive R Archive Network [2] існує величезна колекція пакетів з функціями, що вже використовуються в різноманітних напрямках, від традиційно статистики до геофізики, біоінформатики, економетрії, соціології та інших суспільно важливих дисциплінах. В цьому сенсі R завжди знаходиться попереду в порівняні з пропієтарними програмними середовищами призначеними для статистичних обрахунків і аналізу даних.

[1] http://www.gnu.org/licenses/
[2] http://cran.r-project.org/

Іншою сильною стороною R є можливість приготування *in situ* високоякісних і інформативних графіків для публікацій в наукових виданнях, звітах та web сторінках.

1.1 Початок роботи

R доступний на сайті Comprehensive R Archive Network (CRAN) або одному з його дзеркал за відповідними посиланнями [3] . Після інсталяції та запуску (див. Додаток А) відбувається ініціалізація середовища R [4]

```
R version 2.11.1 (2010-05-31)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

після чого система готова до роботи. Знак > означає готовність до введення і виконання команд. Набір команд також можна запустити окремо з файлу-скрипту.

1.2 Приклади в R

Середовище R активно використовується в задачах, повязаних з обробкою, аналізом і візуалізацією статистичних даних. Один з прикладів демонструє, як згенерувати випадкові числа і представити їхній розподіл у вигляді гістограми

[3] http://cran.r-project.org/mirrors.html

[4] Версія R може відрізнятися. На момент написання офіційний реліз - R version 2.11.1 (2010-05-31)

```
> x <- rnorm(1000) # генерація 1000 випадкових чисел
                   # з розподілу Гауса

# розрахунок гістограми для змінної x, кількість
# інтервалів 50
> histogram <- hist(x, breaks=50, plot=FALSE)

# рисунок гістограми за допомогою функції plot()
> plot(histogram, col="blue",border="red")
```

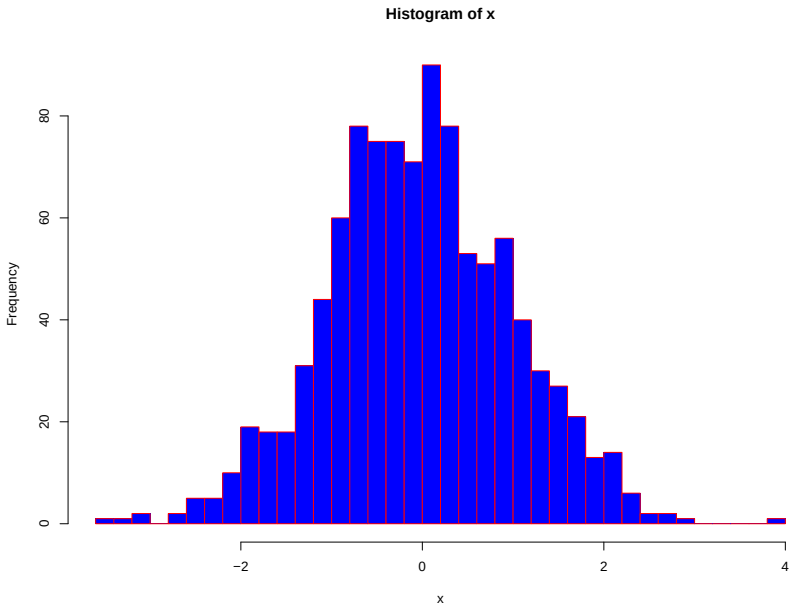

Рис. 1.1. Гістограма побудована на основі вищезгаданого прикладу.

R можна використовувати як калькулятор (в найпростішому випадку)

```
> 1+1
[1] 2

> sqrt(81) # коментарій,
[1] 9    # функція sqrt() виконує рахунок кореня квадратного

> cos(pi/3)
[1] 0.5
```

так і аналізувати дані, що знаходяться в мережі Internet. Наприклад представити історію зміни фондового індексу S&P500 на момент закр-

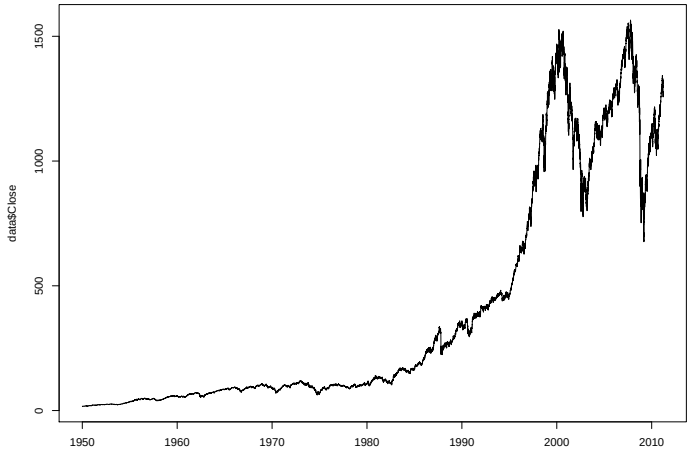

Рис. 1.2. Історія значеннь індексу S&P500 отримані з сайту `finance.yahoo.com` і представлені в графічному вигляді за допомогою R.

иття в графічному вигляді[5]

```
# означення адреси джерела в мережі internet
> address = "http://ichart.finance.yahoo.com/table.csv?
s=%5EGSPC&d=10&e=30&f=2010&g=d&a=0&b=3&c=1950&ignore=.csv"

# зчитування даних і присвоєння їх змінній data
> data <- read.csv(file=url(address))

# конвертація часового формату
> time <- strptime(data$Date,"%Y-%m-%d")

# представлення даних в графічному вигляді
> plot(time,data$Close,type='l')
```

Існують певні обмеження на присвоєння назв об'єктам в R:

- В назвах об'єктів не можуть бути спеціальні символи !, +, -, #.

[5] Адреса джерела даних (параметр address) з часом може стати неактуальною - в такому разі необхідно обновити посилання з сайту http://finance.yahoo.com/indices в частині S&P500 (символ індексу ^GSPC) Historical Prices

- Назви обєктів можуть містити цифри, але не можуть з них починатися.
- Крапка . і символ underscore _ дозволені у використанні назв об'єктів, причому назва об'єкту може починатися з крапки .

Варто зазначити, що середовище R є чутливим до регістру, тобто *variable*, *Variable*, *VARIABLE*,є три різні змінні!

```
> variable <- c(1,2,3,4)
> variable
[1] 1 2 3 4

> Variable <- c('Ponedilok','Vivtorok','Sereda','Chetver')
> Variable
[1] "Ponedilok" "Vivtorok"  "Sereda"    "Chetver"

> VARIABLE <- data.frame(variable,Variable)
> VARIABLE
  variable  Variable
1        1 Ponedilok
2        2  Vivtorok
3        3    Sereda
4        4   Chetver
```

Щоб завершити роботу і вийти з середовища R необхідно ввести команду `q()` або `quit()`.

1.3 Отримання допоміжної інформації.

Ключовим навиком роботи в R є вміння користуватися допоміжною інформацією. R містить вбудовану систему інформації *R-help*. Щоб отримати допоміжну інформацію в командому рядку середовища R вводиться одна з наступних команд (разом з прикладами функцій):

```
help.start()   # загальна система інформації середовища R
help(mean)     # показати інформацію щодо функції mean
?mean          # скорочений формат запису функції help()
apropos("sd")  # показати список всіх функцій, що
               # містить рядок sd
example(mean)  # показати приклад функції mean
help.search("fit")  # пошук за ключовим словом в даному
                    # випадку "fit" в системі допо-
                    # міжної інформації середовища R
??fit   # скорочений формат запису функції help.search()
RSiteSearch("func") # пошук інформації щодо функції func в
                    # системі допоміжної літератури і архіві
```

```
                    # розсилок
```

`help.search()`, `??`*keyword*, `RSiteSearch()` використовуються в ситуаціях, коли не відома точна назва функції.

Деякі пакети містять віньєтки, свого роду анотації, короткий опис того як використовувати пакет, доповнений прикладами

```
vignette()      # показати доступні віньєтки
vignette("frame") # показати віньєтку з пакету grid,
                  # яка стосується побудови
                  # графічного інтерфейсу користувача
```

1.4 Приклади статистичних даних

Разом з програмним середовищем R стають доступними ряд статистичних даних (*datasets*). Щоб переглянути наявні приклади статистичних даних необхідно ввести команду `data()`. Результат залежить від того, які пакети завантажені і підключені в R.

```
> data()

Data sets in package ‘datasets’:

AirPassengers           Monthly Airline Passenger Numbers
                        1949-1960
BJsales                 Sales Data with Leading Indicator
BJsales.lead (BJsales)
                        Sales Data with Leading Indicator
BOD                     Biochemical Oxygen Demand
CO2                     Carbon Dioxide Uptake in Grass Plants
ChickWeight             Weight versus age of chicks on diffe-
                        rent diets
DNase                   Elisa assay of DNase
EuStockMarkets          Daily Closing Prices of Major
                        European Stock Indices,
                        1991-1998
...
```

Щоб переглянути детальну інформацію, щодо статистичних даних необхідно ввести команду `help(`*назва_набору_даних*`)`, наприклад

```
> help(EuStockMarkets)
```

1.5 Демонстрація можливостей R

Середовище R представляє інструмент для ознайомлення і демонстрації функціональних можливостей R. Для цього існує інтерфейс `demo`, що дозволяє запускати демонстраційні скрипти R.

```
> demo()

Demos in package 'base':

is.things               Explore some properties of R objects
                        and is.FOO() functions. Not for new-
                        bies!
recursion               Using recursion for adaptive integra
                        tion
scoping                 An illustration of lexical scoping.

Demos in package 'graphics':

Hershey                 Tables of the characters in
                        the Hershey vector fonts
Japanese                Tables of the Japanese characters in
                        the Hershey vector fonts
graphics                A show of some of R's graphics
                        capabilities
image                   The image-like graphics builtins of R
persp                   Extended persp() examples
plotmath                Examples of the use of mathematics
                        annotation

Demos in package 'stats':

glm.vr                  Some glm() examples from V&R with
                        several predictors
lm.glm                  Some linear and generalized linear
                        modelling examples from 'An In-
                        troduction to Statistical Modelling'
                        by Annette Dobson
nlm                     Nonlinear least-squares using nlm()
smooth                  'Visualize' steps in Tukey's smoo-
                        thers
```

Список всіх доступних демонстраційних прикладів заінстальованих в системі пакетів можна продивитися за допомогою команди

```
> demo(package = .packages(all.available = TRUE))
```

1.6 Пакети в R

1.6.1 Базовий набір пакетів R

Пакети або бібліотеки в R представляють собою колекції функцій і даних призначені для вирішення певного типу задач. Середовище R інсталюється з набором пакетів, повний список, разом з коротким описом, яких можна продивитися за допомогою команди `library()`

```
> library()

Packages in library '/usr/lib/R/library':

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
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                    Main Package of Venables and Ripley's MASS
..
```

Частина з встановлених пакетів завантажуються одночасно разом з початком роботи R. Функція `search()` виводить назви завантажених в середовище R пакетів.

```
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

1.6.2 Інсталяція додаткових пакетів

Можливості R значно розширюються за рахунок використання додаткових пакетів. Додаткові пакети знаходяться у вільному доступі на сайті CRAN [6] або можуть бути написані безпосередньо самим користувачем. Існує кілька варіантів способів інсталяції нових пакетів:

[6] http://cran.r-project.org/web/packages/

1. *Інсталяція пакетів з джерельного коду*
   В першу чергу слід завантажити необхідний пакет *paketname* з сайту CRAN. За допомогою наступної команди з консолі Unix певний пакет *paketname*, наприклад, інсталюється в папку */myfolder/R-packages/*

   ```
   $ R CMD INSTALL paketname -l /myfolder/R-packages/
   ```

2. *Інсталяція пакетів за допомогою R*
   Пакети з сайту CRAN можна заінсталювати безпосередньо за допомогою консолі R (див. Додаток А). Для цього слід скористатится наступною командою

   ```
   > install.packages("paketname")
   ```

   або

   ```
   > install.packages("paketname", lib="/myfolder/R-packages/")
   ```

   і вибрати з якого дзеркала сайту CRAN встановлюватимется пакет.

1.6.3 Підключення додаткових пакетів

Функції і набори даних додаткових пакетів стають доступними до використання після завантаження (підключення) пакетів в середовище R. Щоб підключити вже заінстальований, додатковий пакет використовується функція `library()`

```
> library("mypackage")
#   або вказуючи явно місце знаходження пакету в файловій системі
> library("mypackage", lib.loc=".../librariesFolder/")
```

де *mypackage* - назва пакету/бібліотеки, що підключається, *lib.loc* - місце знаходження пакету. Функція `.libPaths()` виводить адреси місця знаходження пакетів в операційній системі. Наприклад

```
# в Unix подібних системах
> .libPaths()  #
[1] "/usr/lib/R/library"
[2] "/usr/share/R/library"
[3] "/home/X/R/i386-redhat-linux-gnu-library/2.11"

# в Windows
> .libPaths()
[1] "C:/R/R-211~1.1/library"
```

2

Об'єкти і типи даних в R

R представляє дані у вигляді структурованих об'єктів, таких як вектори, матриці, масиви, фактори, списки, датафрейми. Даний розділ демонструє як створюються об'єкти в середовищі R в залежності від типу даних.

2.1 Вектори

Існують три типи векторів числовий, символьний і логічний. Вектори задаються наступним чином

```
> vektor <- c(data1,data2,data3,...)
```

де конструкція `c` - абрівіатура від анг. *collection*. Приклади

```
> a <- c(2,4,6,8,10) # числовий вектор
> a
[1]  2  4  6  8 10

> b1 <- c("Kharkiv","Kyiv","Lviv") # символьний вектор
> b1
[1] "Kharkiv" "Kyiv"  "Lviv"
```

Числа можуть інтерпретуватися як символьні дані (наприклад, поштовий індекс):

```
> b2 <- c("64000","01000","79000") # символьний вектор
> b2
[1] "64000" "01000" "79000"

> c1 <- c(TRUE,FALSE,TRUE,TRUE) # логічний вектор
> c1
[1]  TRUE FALSE  TRUE  TRUE
```

```
> c2 <- (a > 9)
> c2
[1] FALSE FALSE FALSE FALSE  TRUE
```

Вектор може задаватися у вигляді секвенції послідовних чисел за допомогою функції `seq()`

```
> d1 <- (1:15)
> d1
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15

> d2 <- seq(10,0,by=-1)
> d2
[1] 10  9  8  7  6  5  4  3  2  1  0

> seq(0,1,length.out=21)
 [1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50
[12] 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00
```

або реплікацій, повторень чисел чи векторів за допомогою функції `rep()`

```
> f <- rep(1,5)
> f
[1] 1 1 1 1 1

> g <- rep(a,3)
> g
[1]  2  4  6  8 10  2  4  6  8 10  2  4  6  8 10
```

Показати елементи вектора

```
> a[c(1,4)] # показати 1-й і 4-й елементи вектора a
[1] 2 8
```

2.2 Фактори

Фактори являють собою реалізацію символьного вектора і його номінальних значень, як результат класифікації/групування елементів символьного вектора. Фактори задаються за допомогою функції `factor()` наступним чином

```
> f <- factor(x = character(),...)
```

де `x` - вектор символьних значень, ... - додаткові аргументи (див. `?help(factor)`). Нехай існує вектор, елементи якого містять варіанти відповідей на питання "Так", "Ні"або "Незнаю":

```
> answers <- c("Так","Так","Ні","Незнаю","Так","Так","Незнаю",
"Ні","Ні","Ні" )

> answers
 [1] "Так"    "Так"    "Ні"     "Незнаю" "Так"    "Так"
 [7] "Незнаю" "Ні"    "Ні"     "Ні"
```

Фактори показують інформацію про категорії або рівні (Levels) за якими групуються дані.

```
> factor1 <- factor(answers)
> factor1
 [1] Так    Так    Ні     Незнаю Так    Так    Незнаю
 [8] Ні     Ні     Ні
Levels: Незнаю Ні Так
```

Фактор являється більш ефективним об'єктом зберігання символьних даних, що повторюються, особливо коли існує великий об'єм символьних даних. Рівні фактора кодуються середовищем R як цілі числа, відповідно фактор можна представити в закодованому вигляді з використанням функції `as.integer()`

```
> as.integer(factor1)
 [1] 3 3 2 1 3 3 1 2 2 2
```

Загальну кількісну інформацію по категоріям, в даному випадку, варіантам відповідей "Так", "Ні", "Незнаю", можна отримати за допомогою функції `table()`

```
> table(factor1)
factor1
Незнаю     Ні    Так
     2      4      4
```

Фактор також можна згенерувати за допомогою функці `gl()`

```
> gl(1,4)
[1] 1 1 1 1
Levels: 1

> gl(2,4)
[1] 1 1 1 1 2 2 2 2
Levels: 1 2

> gl(3,1)
[1] 1 2 3
Levels: 1 2 3
```

Змінити назви рівнів можна використовуючи аргумент *label*

```
> gl(3,1,20,label = c("Low","Middle","Top"))
[1] Low    Middle Top    Low    Middle Top Low
[8] Middle Top    Low    Middle Top    Low    Middle
[15] Top    Low    Middle Top    Low    Middle
Levels: Low Middle Top
```

2.3 Часові ряди/серії

Часові ряди або ж часові серії, являють собою об'єкт, який дозволяє зберігати змінні в часі дані. Часові серії створюються за допомогою функції `ts()` і складаються з даних (у вигляді векторів, матриці чисел) і дат, що розміщенні між собою з певним заданим часовим інтервалом.

В загальному випадку синтаксис функції `ts()` має вигляд

```
> ts(data, start, frequency,...)
```

де *data* - дані часової серії, *start* - час першої обсервації, *frequency* - число обсервацій за цикл (одиницю) часу. Приклад часової серії випадкових чисел з 2010 по 2012 рік.

```
> timeseries <- ts(data=rnorm(34), frequency = 12,
> start = c(2010))
> plot(timeseries)
```

2.4 Матриці і масиви

Матриця в R представляє собою вектор у вигляді 2-х вимірної таблиці, розміром `n x m`, всі елементи якої належать до чисельного типу даних.

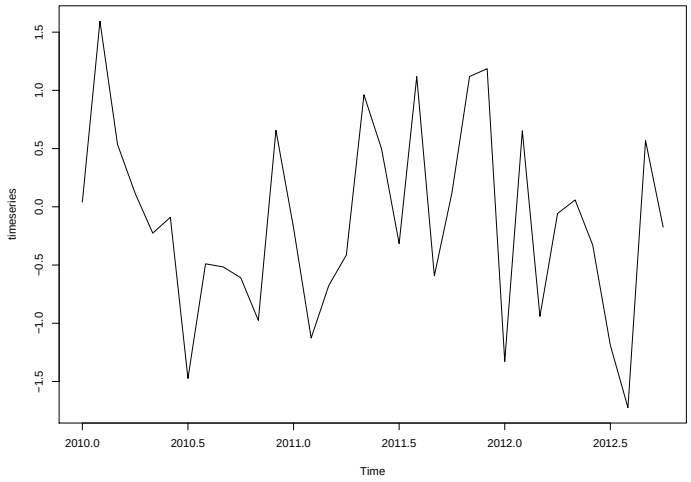

Рис. 2.1. Часовий ряд згенерований з використанням функції `ts()`

Загальний формат запису матриці з використанням функції `matrix()` має вигляд

```
> matrixname <- matrix(vector,nrow,mcol)
```

де *vector* - набір даних у вигляді вектора, *nrow* - кількість рядків, *mcol* - кількість стовбців. Приклад матриці розміром 3 x 5

```
> matrix1 <- matrix(1:15,nrow=3,ncol=5)
> matrix1
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    7   10   13
[2,]    2    5    8   11   14
[3,]    3    6    9   12   15
```

Функція `dim(`*matrix*`)` дозволяє показати розмір матриці

```
> dim(matrix1)
[1] 3 5
```

Матрицю можна згенерувати з векторів розміщуючи їх рядками або стовпчиками за допомогою функцій `rbind()` та `cbind()` відповідно.

```
> rbind(c(1,3,5),c(6,4,2),c(0,0,0),c(7,8,9))
     [,1] [,2] [,3]
```

```
[1,]    1    3    5
[2,]    6    4    2
[3,]    0    0    0
[4,]    7    8    9

> cbind(c(1,3,5),c(6,4,2),c(0,0,0),c(7,8,9))
     [,1] [,2] [,3] [,4]
[1,]    1    6    0    7
[2,]    3    4    0    8
[3,]    5    2    0    9
```

Знаходження елементу, рядка, стовбчика матриці за їхніми індексами виглядає наступним чином.

Нехай маємо матрицю

```
> matrix2<- matrix(c(1,3,5,6,4,2,0,0,0,7,8,9),2,6)
> matrix2
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    4    0    0    8
[2,]    3    6    2    0    7    9
```

тоді

```
> matrix2[1,]   # перший рядок матриці matrix2
[1] 1 5 4 0 0 8

> matrix2[,6]   # шостий стовбчик матриці matrix2
[1] 8 9

> matrix2[2,3]  # елемент 2-го ряду і 3 стовпчика матриці matrix2
[1] 2
```

На відміну від матриці, масив являє собою вектор значень *vector* у вигляді таблиці, в якій може бути довільна кількість вимірів k. Синтаксис запису масиву в R задається наступним чином

```
> arrayname <- array(vector,k)
```

Приклад 3-х вимірного масиву який складається з 4-х таблиць (матриць)

```
> array(1:16,c(2,2,4))
, , 1

     [,1] [,2]
```

```
[1,]    1    3
[2,]    2    4

, , 2

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

, , 3

     [,1] [,2]
[1,]    9   11
[2,]   10   12

, , 4

     [,1] [,2]
[1,]   13   15
[2,]   14   16
```

Незалежно від розташування елементів матриці або масиву, всі елементи доступні, як елементи одного вектора.

2.5 Блоки даних або датафрейми

Датафрейми є одними з основних і фундаментальних об'єктів середовища R, який можна охарактеризувати, як матрицю з різними типами даних. В стовпчиках такої матриці можуть зберігатися числові, символьні, логічні вектори, фактори, матриці чисел, інші блоки даних. Символьні вектори автоматично конвертуються у фактори. Датафрейми створюються з використанням функції `data.frame()`. Загальний формат запису датафрейму має вигляд

```
> dataframename<-data.frame(var1,var2,var3...varN)
```

де *var1,var2,var3...varN*- змінні різних типів. Приклад датафрейму

```
> data1 <- c(101,102,103,104)
> data2 <- c("Lviv", "Kyiv", "Kharkiv", NA)
> data3 <- c("79000","01000","64000","00000")
> data4 <- c(TRUE,FALSE,TRUE,FALSE)
> alldata <- data.frame(data1,data2,data3,data4)
> names(alldata) <- c("ID","City","PostID","Check")
```

```
> alldata
   ID    City PostID Check
1 101    Lviv  79000  TRUE
2 102    Kyiv  01000 FALSE
3 103 Kharkiv  64000  TRUE
4 104    <NA>  00000 FALSE
```

Кожний стовпчик або змінна датафрейму має унікальне імя. Вміст змінної датафрейму можна отримати за допомогою команди, яка складається з імені датафрейму, знаку `$` та імені змінної `dataframe$`*variable*

```
> alldata$City
[1] Lviv    Kyiv    Kharkiv <NA>
Levels: Kharkiv Kyiv Lviv

> alldata$PostID
[1] 79000 01000 64000 00000
Levels: 00000 01000 64000 79000
```

Використання функції `attach()` дозволяє звертатися до змінних датафрейму безпосередньо за назвою/іменем змінної.

```
> attach(alldata)
> City
[1] Lviv    Kyiv    Kharkiv <NA>
Levels: Kharkiv Kyiv Lviv

> PostID
[1] 79000 90000 64000 00000
Levels: 00000 01000 64000 79000
```

Щоб повернутися до попереднього формату написання змінних використовується функція `detach()`

```
> detach(alldata)
> City
Error: object 'City' not found

> alldata$City
[1] Lviv    Kyiv    Kharkiv <NA>
Levels: Kharkiv Kyiv Lviv
```

Аналогічним чином функції `attach()` і `detach()` використовуються для роботи зі змінними списку (list).

Доступ до елементів датафрейму здійснюється вказуючи номер/назву рядка і/або стовпчика

```
> alldata[4,"ID"]
[1] 104

> alldata[2:3,]
   ID    City PostID Check
2 102    Kyiv  90000 FALSE
3 103 Kharkiv  64000  TRUE

> alldata[,"City"]
[1] Lviv    Kyiv    Kharkiv  <NA>
Levels: Kharkiv Kyiv Lviv

> alldata[["City"]][1]
[1] Lviv
Levels: Kharkiv Kyiv Lviv
```

2.6 Списки

Список являє собою об'єкт, що за своєю структурою подібний до вектора. Однак на відміну від вектора, де всі елементи належать до одного типу даних, список може містити різні об'єкти з різними типами даних, включно з іншими списками і блоками даних (датафреймами). Відповідно списки використовуються в ситуаціях коли дані близькі за своїм контентом, але неоднорідні за своєю структурою. Списки генеруються за допомогою функції `list()`

```
> x1 <- 1:5
> x2 <- c('Ponedilok','Vivtorok','Chetver','Sereda','Piatnycia')
> x3 <- c(T,T,F,F,T)
> y <- list(daynumber=x1, day=x2, order=x3)
> y
$daynumber
[1] 1 2 3 4 5

$day
[1] "Ponedilok" "Vivtorok"  "Chetver"   "Sereda"    "Piatnycia"

$order
[1]  TRUE  TRUE FALSE FALSE  TRUE
```

Функція `names()` дозволяє отримати імена компонент списку

```
> names(y)
[1] "daynumber" "day"       "order"
```

Доступ до компонентів списку відбувається:

- (- за індексом компоненти)

- (- за іменем компоненти)

```
> slobozan <- list( Kharkivskaobl = list(mista = c('Kharkiv' =
1440676, 'Izum' = 53223, 'Krasnograd' = 27600 , 'Bohoduxiv'
= 17653, 'Zmijiv' = 16976, 'Liubotyn' = 25700,'Balakliia' =
32117,'Lozova'= 71500), naselenia = 2760948, oblcentr =
'Kharkiv'), Sumskaobl = list(mista = c('Sumy' = 273984,
'Okhtyrka' = 49431, 'Trostianets' = 23370, 'Konotop' = 93671,
'Shostka' = 80389  ), naselenia = 1164619, oblcentr = 'Sumy'))

> slobozan[[1]]
$mista
   Kharkiv       Izum Krasnograd  Bohoduxiv     Zmijiv   Liubotyn
   1440676      53223      27600      17653      16976      25700
   Balakliia    Lozova
   32117        71500

$naselenia
[1] 2760948

$oblcentr
[1] "Kharkiv"



> slobozan$Kharkivskaobl
$mista
   Kharkiv       Izum Krasnograd  Bohoduxiv     Zmijiv   Liubotyn
   1440676      53223      27600      17653      16976      25700
   Balakliia  Lozova
   32117      71500

$naselenia
[1] 2760948

$oblcentr
[1] "Kharkiv"

> slobozan$Sumskaobl$oblcentr
[1] "Sumy"
```

Результати більшості статистичних аналізів представляються у вигляді списків.

2.7 Логічні типи даних і оператори

Об'єкти логічних типів даних можуть приймати значення `TRUE` або `FALSE` і використовуються для того, щоб показати чи умова істина чи хибна. Такі об'єкти зазвичай є результатом логічних операцій.

Логічні оператори

```
 <  менший ніж
 <= менший або рівний
 >  більший за
 >= більший або рівний
 == рівний
 &  логічний оператор "І"
 |  логічний оператор "АБО"
 !  логічний оператор "НІ"
 != не рівний
```

3

Експорт/Імпорт даних в R

3.1 Експорт даних

Записати дані з середовища R в файл можна кількома способами, в залежності від необхідної вихідної структури/формату файлу даних. Основною фунцією є `write.table()`, яка дозволяє зберегти матрицю чисел або датафрейм у вигляді таблиці даних. Загальний синтаксис функції

```
> write.table(x, file = "", ...)
```

де *x* - об'єкт, що записується (матриця або датафрейм), *file* - назва файлу, в який записуються дані, ... - додаткові аргументи. Детальніше про додаткові аргументи `?write.table` або `help(write.table)`

В результаті використання функції `write.table()` створюється файл з вказаними іменами рядків та стовбчиків (при умові, що імена існували), в якому дані розділені між собою пробілами.

```
> xval<-mtcars[1:10,]
> write.table(xval,file="data1.txt")
```

Файл *data1.txt* містить десять рядків з набору даних *mtcars* (див. `?mtcars`)

```
"mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" .
"Mazda RX4" 21 6 160 110 3.9 2.62 16.46 0 .
"Mazda RX4 Wag" 21 6 160 110 3.9 2.875 17.02 0 .
"Datsun 710" 22.8 4 108 93 3.85 2.32 18.61 1 .
"Hornet 4 Drive" 21.4 6 258 110 3.08 3.215 19.44 1 .
"Hornet Sportabout" 18.7 8 360 175 3.15 3.44 17.02 0 .
"Valiant" 18.1 6 225 105 2.76 3.46 20.22 1 .
"Duster 360" 14.3 8 360 245 3.21 3.57 15.84 0 .
"Merc 240D" 24.4 4 146.7 62 3.69 3.19 20 1 .
"Merc 230" 22.8 4 140.8 95 3.92 3.15 22.9 1 .
"Merc 280" 19.2 6 167.6 123 3.92 3.44 18.3 1 .
```

Функції `write.csv()` і `write.csv2()` є різновидами функції `write.table()` з означеними аргументами, використання яких дозволяє зберігати дані у форматі CSV розділених , і ; в першому і другому випадку, відповідно.

```
> write.csv(xval,file="data2.csv")
```

Файл *data2.csv* має вигляд

```
"","mpg","cyl","disp","hp","drat","wt","qsec","vs" .
"Mazda RX4",21,6,160,110,3.9,2.62,16.46,0 .
"Mazda RX4 Wag",21,6,160,110,3.9,2.875,17.02,0 .
"Datsun 710",22.8,4,108,93,3.85,2.32,18.61,1 .
"Hornet 4 Drive",21.4,6,258,110,3.08,3.215,19.44,1  .
"Hornet Sportabout",18.7,8,360,175,3.15,3.44,17.02,0 .
"Valiant",18.1,6,225,105,2.76,3.46,20.22,1 .
"Duster 360",14.3,8,360,245,3.21,3.57,15.84,0 .
"Merc 240D",24.4,4,146.7,62,3.69,3.19,20,1 .
"Merc 230",22.8,4,140.8,95,3.92,3.15,22.9,1 .
"Merc 280",19.2,6,167.6,123,3.92,3.44,18.3,1 .
```

Формат CSV є поширеним форматом збереження даних і використовується для експорту/імпорту даних між різними програмними середовищами.

3.2 Запис даних в форматі Excel

За допомогою пакету `xlsReadWrite` дані з середовища R можна зберегти у форматі MS Office Excel (файл з розширенням **.xls*). Перед викликом функції `write.xls()`, призначеної для збереження даних у форматі Excel в робоче середовище R завантажується пакет `xlsReadWrite`.

```
> library(xlsReadWrite)
> write.xls(data1, ".../rtoexceldata.xls")
```

3.3 Перенаправлення даних з екрану в файл

Дані з робочого середовища R можна зберегти за допомогою функці `sink()` і функції `cat()`. Функція `sink(file)` направляє стандартний вихід всіх команд з екрану терміналу середовища R в файл *file*.

```
> x <- c(1,3,5,7,9,15,7)
> sink("testdatafile.txt") # Скерування стандартного виходу в
файл testdatafile.txt
```

```
> print(x)
> cat("The mean of x is",round(mean(x),3))
> sink() # Завершення запису в файл і перенаправлення знову в
стандартний вихід середовища R
```

На відміну від попередньої функції `sink()`, `cat()` направляє в файл дані, що явно задані в межах самої функції.

```
> cat("2 3 5 7", "11 13 17 19", file="exportcat.dat", sep="\n")
```

3.4 Імпорт даних

Необхідним навиком роботи в R є введення та імпорт даних. Великі об'єми даних зазвичай імпортуються з файлів, а окремі, дані вводятся безпосередньо з клавіатури. Щоб полегшити роботу з файлами, R містить кілька функцій, які дозволяють безпосередньо взаємодіяти з операційною системою. Наприклад при написанні команди або скрипту зчитування даних з файлу необхідно знати точне ім'я файлу даних. Аби проглянути список об'єктів в папці/каталозі існує фунція `list.files()`

```
> list.files()
[1] "data.csv"  "myfile.txt" "Rlib" "testscript.R"
```

Кількість об'єктів в даній папці

```
> length(list.files())
[1] 4
```

Функція `getwd()` дозволяє отримати повний адрес робочого каталогу. Натомість фунція `setwd("../Folder")` змінює адрес робочого каталогу. У випадку коли файл знаходиться поза робочим каталогом необхідно вказати повний адрес перед самим файлом.

3.5 Імпорт даних з форматованого текстового файлу

Завантажити дані з текстового файлу (ASCII формат файлів) в R можна за допомогою наступних функцій:

- `read.table()` – імпортує дані у вигляді датафрейму з форматованого текстового файлу;

- `read.csv()` – імпортує датафрейми з текстового файлу, в якому дані відокремлюються символом коми;

- `read.delim()` – зчитує дані з текстового фауйлу, в якому дані відокремлені символами табуляції;
- `read.fwf()` – імпортує дані з файлу формат фіксованої ширини;
- `scan()` – надає можливості низько-рівневого зчитування даних.

Базовою є функція `scan()`, яка часто використовується для зчитування файлів великих розмірів. Функцій `read.table()`, `read.csv()`, `read.fwf()` є похідними від базової.

3.6 Функції `read.table()`,`read.csv()` і `read.delim()`

Найбільш зручний спосіб імпортувати текстові дані є використання функції `read.table()`. Синтаксис функції має вигляд

```
> read.table(file,...)
```

де *file* - назва файлу і повний шлях до нього (якщо файл знаходиться за межами робочої директорії), ... - набір аргументів. Деталі `?read.table` або `help(read.table)`

Нехай маємо текстовий файл *group.txt*, що містить назви змінних і дані

```
"Names" "Age" "Weight,kg" "Height,cm" "Sex" "ID"
"Sofia" 2 12 55 "W" "10001"
"Petro" 40 101 173 "M" "10002"
"Vitalij" 37 90 175 "M" "10003"
"Inna" 18 52 180 "W" "10004"
"Anna" 20 55 170 "W" "10005"
```

Тоді зчитування даних, включаючи іменна змінних, з файлу може виглядати так

```
> groupdata <- read.table("group.txt",header=TRUE)
 groupdata
    Names Age Weight.kg Height.cm Sex    ID
1   Sofia   2        12        55   W 10001
2   Petro  40       101       173   M 10002
3 Vitalij  37        90       175   M 10003
4    Inna  18        52       180   W 10004
5    Anna  20        55       170   W 10005
```

Опції *header* ставиться значення `TRUE` для зчитування першого рядка текстового файлу, що містить імена змінних, у якості назв стовпчиків.

Функції `read.csv()` і `read.delim()` є більш компактним записом функції `read.table()` у випадку завантаження даних відокремлених комами і

символами табуляції, відповідно. Існують також варіації у вигляді функцій `read.csv2()` і `read.delim2()` призначені для зчитування даних з файлів, в яких десяткові числа відокремленні від цілих комами.

Наприклад за допомогою функції `read.csv()` використовуючи файл `data2.csv` (див. стр. 22) дані імпортуються в середовище R, наприклад, як датафрейм з іменем *datacsv*. За допомогою функції `print()` вміст датафрейму виводиться на екран

```
> datacsv<-read.csv('data2.csv')
> print(datacsv)
```

3.7 Функція `read.fwf()`

Формат файлів з фіксованою шириною зустрічається доволі рідко, оскільки більшість даних у текстових файлах відокремленні або комою або символом табуляції. Тим не менш такі файли зустрічаються і під час імпорту даних з використанням функції `read.fwf()` вказується параметер *width* - тобто вектор, який містить числові значеня кількості символів (ширини) для кожної змінної, що імпортується. Функція `read.fwf()` створює і записує дані в тимчасовий файл, в якому дані відокремлюються символами табуляції, а безпосередній імпорт даних в середовище R здійснюється функцією `read.table()`.

```
> dat.ff <- tempfile()
> cat(file=dat.ff,"12345678","abcdefgh",sep="\n")
> read.fwf(dat.ff,width=c(2,4,1,1))

>   V1   V2 V3 V4
> 1 12 3456  7  8
> 2 ab cdef  g  h

> unlink(dat.ff) # видаляє файл
```

Функція `unlink()` видаляє вказаний файл або директорію/папку.

3.8 Функція scan()

Функція `scan()` використовується в ситуаціях в яких попередні функції є менш ефективними або неспроможні коректно імпортувати дані. `scan()` зчитує дані у вигляді вектора або списку. Синтаксис функції має вигляд

```
> scan(file = "",...)
```

де *file* - назва файлу і повний шлях до нього (якщо файл знаходиться за межами робочої директорії), ... - набір аргументів.

```
content <- scan('group.txt',what="character",sep='')
Read 36 items
> content
 [1] "Names"     "Age"    Weight,kg" "Height,cm" "Sex"  "ID"
 [7] "Sofia"     "2"      "12"        "55"        "W"    "10001"
[13] "Petro"     "40"     "101"       "173"       "M"    "10002"
[19] "Vitalij"   "37"     "90"        "175"       "M"    "10003"
[25] "Inna"      "18"     "52"        "180"       "W"    "10004"
[31] "Anna"      "20"     "55"        "170"       "W"    "10005"
```

Аргументи *what* і *sep* задають тип даних та символ сепарації даних відповідно. Деталі див `?scan` або `help(scan)`.

У випадку коли джерело даних, тобто аргумент *file* не вказано, введення даних здійснюється інтерактивно (див. стр. 27)

3.9 Імпорт даних з файлів EXCEL (*.xls файли)

Найкращий спосіб прочитати **.xls* файли - зберегти дані в середовищі у вигляді форматованого **.csv* файлу та імпортувати його як вищезгаданий `csv` файл. В Windows системах для доступу до даних файлів Excel, можна скористатися пакетом `RODBC`. Перший рядок повинен містити змінні/імена стовбчиків.

```
# перший рядок містить імена зміних
# дані зчитуються з workSheet mysheet

> library(RODBC)
> channel <- odbcConnectExcel("c:/exelfile.xls")
> mydata <- sqlFetch(channel, "mysheet")
> odbcClose(channel)
```

3.10 Імпорт даних з файлів SPSS

Щоб імпортувати дані з програми SPSS необхідно перед цим в R завантажити пакет `Hmisc`

```
# Збереження даних SPSS в формат який розпізнається R
> get file='c:\dataSPSS.sav'.
> export outfile='c:\dataSPSS.por'.

# Завантаження даних в R
```

```
> library(Hmisc)
> mydata <- spss.get("c:/dataSPSS.por", use.value.labels=TRUE)
# остання опція конвертує назви в фактори
```

3.11 Введення даних з клавіатури

Блоки даних можна створити інтерактивно вводячи дані безпосередньо з клавіатури. В першому випадку дані вводяться безпосередньо в середовище R за допомогою функції `scan()`. Введення чисел

```
> numbervector<-scan()
1: 0
2: 10
3: 20
4: -30
5: 40
6: 50
7:
Read 6 items
> numbervector
[1]   0  10  20 -30  40  50
```

Введення даних символьного типу

```
 charvector <- scan(what = "", sep = "\n")
1: Перший рядок
2: Другий
3: Третій
4: Напевно вистачить?
5:
Read 4 items

 charvector
[1] "Перший рядок"       "Другий"        "Третій"
[4] "Напевно вистачить?"
```

Пустий рядок (тобто два рази Enter) означає закінчення режиму введення даних.

В другому випадку дані в R заносяться через редактор за допомогою функції `edit()` або `fix()`. Змінні, в які вносяться дані, мусять бути попередньо задекларовані.

```
> country<-c("EU","USA","Japan")
> currencyquant<-c(100,100,1000)
```

```
> currency<-c("EURO","USD","JPY")
> rate<-c(1070.08,793.77,95.12)
> national<-rep("UAH",3)
> mydata <- data.frame(country,currency,currencyquant,
> rate,national)

> temp <- edit(mydata)
> mydata <-temp # перезапис/оновлення даних об'єкту mydata
```

3.12 Отримання інформації про об'єкти

Проглянути об'єкти робочого середовища R можна за допомогою функції `ls()`

```
> ls()
[1] "datafile" "xdata"
```

Проглянути змінні об'єкту можна з використанням функції `names()`

```
> names(xdata)
 [1] "X"    "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec"
[9] "vs"   "am"  "gear" "carb
```

Проглянути вміст об'єкту за допомогою функції `print()`

```
> print (xdata)
                   X  mpg cyl  disp  hp drat    wt  qsec vs  .
1          Mazda RX4 21.0   6 160.0 110 3.90 2.620 16.46  0  .
2      Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  .
3         Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1  .
4     Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  .
5  Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  .
6            Valiant 18.1   6 225.0 105 2.76 3.460 20.22  1  .
7         Duster 360 14.3   8 360.0 245 3.21 3.570 15.84  0  .
8          Merc 240D 24.4   4 146.7  62 3.69 3.190 20.00  1  .
9           Merc 230 22.8   4 140.8  95 3.92 3.150 22.90  1  .
10          Merc 280 19.2   6 167.6 123 3.92 3.440 18.30  1  .
```

Проглянути перші 10 рядків об'єкту `xdata` можна з використанням функції `head()`

```
> head(xdata, n=4)
```

```
              X  mpg cyl disp  hp drat    wt  qsec vs am   .
1      Mazda RX4 21.0   6  160 110 3.90 2.620 16.46  0  1  .
2  Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1  .
3     Datsun 710 22.8   4  108  93 3.85 2.320 18.61  1  1  .
4 Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0  .
```

Проглянути останні 2 рядків об'єкту `xdata` можна з використанням функції `tail()`

```
> tail(xdata, n=2)
          X  mpg cyl  disp  hp drat   wt qsec vs am gear carb
9  Merc 230 22.8   4 140.8  95 3.92 3.15 22.9  1  0    4    2
10 Merc 280 19.2   6 167.6 123 3.92 3.44 18.3  1  0    4    4
```

Проглянути структуру об'єкту за допомогою функції `str()`

```
> str(xdata)
'data.frame': 10 obs. of  12 variables:
 $ X   : Factor w/ 10 levels "Datsun 710","Duster 360",..: 5 ...
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2
 $ cyl : int  6 6 4 6 8 6 8 4 4 6
 $ disp: num  160 160 108 258 360 ...
 $ hp  : int  110 110 93 110 175 105 245 62 95 123
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : int  0 0 1 1 0 1 0 1 1 1
 $ am  : int  1 1 1 0 0 0 0 0 0 0
 $ gear: int  4 4 4 3 3 3 3 4 4 4
 $ carb: int  4 4 1 1 2 1 4 2 2 4
```

Проглянути розмірність об'єкту

```
> dim(xdata)
[1] 10 12
```

Проглянути класс об'єкту з використанням функції `class()`

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

3.13 Спеціальні значення

3.13.1 NA і NaN

В середовищі R відсутні значення змінних представляються символами `NA` (Not Available).

Наприклад, якщо "розтягнути"вектор або масив на величину, що перевищує розмір вектора із заданими значеннями, нові елементи, що з'явилися в результаті "розтягнення"приймають значення `NA`.

```
> a <- c(10,0,5)
> a
[1] 10  0  5

> length(a) <- 5

> a
[1] 10  5  0 NA NA
```

Тест на відсутність значень

```
> is.na(x) # результат буде TRUE якщо в x значення вісутнє
> y <- c(1,2,3,NA)
> is.na(y) # результатом буде вектор логічних значеннь (F F F T)

[1] FALSE FALSE FALSE  TRUE
```

Помилки або результати недопустимих операцій представляються символом `NaN` (не числове значення).

```
> z <- sqrt(c(1,-1))

Warning message:
In sqrt(c(1, -1)) : NaNs produced

> print(z)
[1]   1 NaN

> is.nan(z)
[1] FALSE  TRUE
```

3.13.2 Нескінченість Inf

Якщо результат обрахунків є занадто великим числом, R повертає `Inf` у випадку позитивного числа і `-Inf` у випадку негативного числа.

```
> 2 ^ 1024
[1] Inf
> - 2 ^ 1024
[1] -Inf
```

3.13.3 Значення NULL

В середовищі R існує нульовий об'єкт, який задається символом NULL. Значення NULL може повертатися як результат обрахунку R виразів або функцій чиї значення не визначені або використовуватися як аргумент функції, щоб вказати явно відсутність значення аргумента.

```
> x1 <- as.null(list(var1=1,var2='c'))
> print(x1)
NULL
```

3.14 Кодування значень змінних

Значення змінної в R можна перекодувати. Наприклад значення змінної яка дорівнює 99 можна перекодувати в NA

```
# select rows where v1 is 99 and recode column v1
> mydata[mydata$v1==99,"v1"] <- NA
```

3.15 Виключення відсутніх значень з аналізу

Застосування арифметичних функцій до змінних із значенями NA дають результат теж NA

```
> x <- c(1,2,NA,3)
> mean(x)  # повертає результат NA
[1] NA
```

Натомість вказуючи явно

```
> mean(x, na.rm=TRUE) # повертає результат 2
[1] 2
```

na.rm=TRUE - аргумент, який в даному випадку окреслює, що NA значення потрібно усувати (не враховувати).

4

Функції і конструкції в R

Переважна більшість операцій і задач в R здійснюється за допомогою функцій. Функції являють собою програмний код з певного набору змінних, констант, операторів, інших функцій, що призначені для виконання певної конкретної задачі або операцій і поверненням результату виконання функції. За своїм призначенням функції можна поділити на кілька окремих груп арифметичні, символьні, статистичні та інші, а також вбудовані і власні, тобто ті, що написані безпосередньо самими користувачами.

4.1 Вбудовані функції

До вбудованих належать функції, які є складовою частиною програмного середовища R.

4.1.1 Арифметичні функції

```
abs(x) # модуль величини x

ceiling(x) #  округлення до цілого в більшу сторону,
           #  ceiling(9.435)
           #   10
floor(x)   #  округлення до цілого в меншу сторону,
           #   floor(2.975) буде 2
round(x, digits=n)   # округлення до вказаного числа digits
                     # після коми(крапки),
                     # round(5.475, digits=2) буде 5.48
signif(x, digits=n)  # округлення до вказаного числа digits
                     # з урахуванням чисел перед комою
                     #  signif(3.475, digits=2) буде 3.5
trunc(x)   # відкиненя значень після коми (крапки)
           # trunc(4.99) буде  4
cos(x), sin(x), tan(x), acos(x), cosh(x), acosh(x)
```

```
exp(x)  #  e^x
log(x)  #  логарифм натуральний x
log10(x) # логарифм десятковий x
sqrt(x)  # корінь квадратний x
```

4.1.2 Функції для роботи з символьними типами даних

```
grep(pattern, x , ignore.case=FALSE, fixed=FALSE)
  #  Пошук значень pattern серед даних в x і повернення
  #  значення індексу елементу, що співпав
grep("A", c("x","y","A","z"), fixed=TRUE)
[1] 3

substr(x, start=n1, stop=n2)
  # Вибір або заміна символів вектора символьного типу
substr(x, 2, 4)
[1] "yxx"

substr(x, 2, 4) <- "01100"
print(x)
[1] "z011wt"

paste(..., sep="")
  # Об'єднання символів або рядків
  # після використання символа відокремлення, що задається
  # аргументом sep=""

paste("x",1:3,sep="")
[1] "x1" "x2" "x3"
paste("x",1:3,sep="val")
[1] "xval1" "xval2" "xval3"

strsplit(x, split) # Розділяє елементи вектора
                   # x за вказаним split критерієм
strsplit("abc", "")
[[1]]
[1] "a" "b" "c"
strsplit("abcabcab", c("c","a"))
[[1]]
[1] "ab" "ab" "ab"

split(1:10, 1:2)
$`1`
[1] 1 3 5 7 9

$`2`
[1]  2  4  6  8 10
```

```
sub(pattern, replacement, x, ignore.case =FALSE, fixed=FALSE)
  # Заходить pattern в об'єкті x
  # і замінює на значення задене в replacement

sub("\\s","!","Привіт Всім")
[1] "Привіт!Всім"

toupper(x)  # заміна на ВЕЛИКІ ЛІТЕРИ
toupper("великі")
[1] "ВЕЛИКІ"

tolower("МАЛІ ЛІТЕРИ")  # заміна на малі
[1] "малі літери"
```

Приклади статистичних функції див. стр. 47 в розділі "Статистика".

4.2 Написання власних функцій

Для розширення функціоналу програмного середовища в R існує можливість написання власних нових функцій . Загальний синтаксис власної функції має вигляд

```
functionname <- function(arg1, arg2,...) {

body # певний функціональний код у вигляді набору команд,
 # зміних, констант, операторів, інших вже існуючих функцій.

return(object)

}
```

де `functionname` - ім'я функці, *arg1*, *arg2*,... - вхідні аргументи функції , `body` - тіло функції, код який виконує функція, `object` - результат, що повертається в результаті виконання функції. Функції також можуть бути і без вхідних аргументів. Приклад функції, результатом виконання якої є одночасне представлення середнього і значення середьоквадратичного відхилення

```
> functionAverageDev <- function(x){
    aver <- mean(x)
    stdev <- sd(x)
    c(MEAN=aver, SD=stdev)
    }
```

```
> functionAverageDev(1:100)
    MEAN       SD
50.50000 29.01149
```

Функція `functionAverageDev()` викликає дві стандартні функції середовища R : `mean()` і `sd()`. Функцію можна записати у скриптовий файл і викликати її для виконання з файлу. Наприклад файл *avsqroot.R* містить код функції середнього і квадрату середнього значення

```
> functionAvSqroot <- function(x){
    aver <- mean(x)
    sq <- sqrt(aver)
    c(MEAN=aver, SquareRoot=sq)
    }
```

Перед викликом функції з скриптового файлу необхідно вказати ім'я файла, де знаходиться функція, за допомогою команди `source()`.

```
> functionAvSqroot(1:100)
Error: could not find function "functionAvSqroot"

> source("avsqroot.R")

> functionAvSqroot(1:100)
      MEAN SquareRoot
 50.500000   7.106335
```

Використання скриптового файлу для запису функції є зручним в ситуаціях коли тіло функції містить код значного об'єму.

4.2.1 Аргументи і змінні функції

Синтакс заголовку функції вказує чи аргумент функції є обов'язковим чи може вказуватися опціонально. У випадку якщо обов'язковий аргумент не вказаний, то при виконані функції виникне помилка.

```
> power <- function(x,n=3){
  x^n
}

> power()
Error in x^n : 'x' is missing


> power(2)
```

```
[1] 8
```

аргумент `n` - не є обов'язковим, однак може приймати інше значення.

```
> power(2,6)
[1] 64
```

Аргументами функцій можуть бути також об'єкти будь-якого типу, навіть функції.

```
> exampl <- function(x, function2)
{
  x <- runif(x)
  function2(x)
}

> exampl(5,log)
[1] -0.5747896 -0.6354184 -0.8888178 -0.2795405 -0.7150940
```

Останній вираз в тілі функції є результатом виконання функції. В даному випадку ним може бути тільки одне значення об'єкту.

```
> myf <- function(x,y){
  z1 <- sin(x)
  z2 <- cos(y)
  z1+z2
}
```

Значення кількох об'єктів можна отримати використовуючи список.

```
> myf <- function(x,y){
  z1 <- sin(x)
  z2 <- cos(y)
  list(z1,z2)
}
```

Щоб вийти з функції раніше останьої лінії коду використовується функція `return()`. Будь-який код після `return()` ігнорується.

```
> myf <- function(x,y){
  z1 <- sin(x)
  z2 <- cos(y)
  if(z1 < 0){
     return( list(z1,z2))
  else{
```

```
    return( z1+z2)
  }
}
```

Змінні в середнині функції є локальними зміними. Локальні змінні не взаємодіють/перекриваються з об'єктами поза межами функції (глобальними змінними) і існують тільки в межах самої функції. Якщо змінна знаходиться в тілі функції то ця змінна повертається як результат функції

```
> y <- 0
> functionY <- function(){
   y <- 10
}

> print(functionY())
[1] 10
> print(y)
[1] 0
```

Аргумент ... використовується для передачі аргументів з однієї функції в іншу. В якості ... можуть використовуватися аргументи у вигляді змінних або інших функцій. Наприклад

```
> product <- function(x,...) {
 arguments <- list(...)
      for (y in arguments)
      x <- x*y
      x
}
> product(10,10,10)
1000

> plotfunc <- function(upper = pi, ...)
{
  x <- seq(0,upper,l = 100)
  plot(x,..., type = "l", col="blue")
}
> plotfunc(tan(x))
```

4.3 Управління потоками - тести і цикли

4.3.1 Функції if і switch

Загальний синтаксис конструкції `if()` (`if-else()`) має вигляд

```
if(test)
{
   ...істинне твердження...
}
else
{
   ...хибне твердження...
}
```

де `test` - логічний вираз. Якщо результатом виконання логічного виразу буде `TRUE`, виконуватиметься код заключений в частині, що відповідає істинному твердженню. У випадку `FALSE` виконуватиметься код, що знаходиться в частині хибне твердження. Блок else може бути відсутнім.

```
> compare <- function(x, y){
  n1 <- length(x)
  n2 <- length(y)
  if(n1 != n2){

   if(n1 > n2){
    z=(n1 - n2)
    cat("Ліва змінна має на ",z," елемент(ів) більше \n")
   }
   else{
    z=(n2 - n1)
    cat("Права змінна має на ",z," елемент(ів) більше \n")
   }
}
else{
  cat("Одинакова кількість елементів ",n1,"\n")
}
}

> x <- c(1:4)
> y <- c(1:9)
> compare(x, y)
 Права змінна має на  5  елемент(ів) більше
```

Використання функції `switch()` має наступний синтаксис

```
> switch(object,
    "value1" = {expr1},
    "value2" = {expr2},
    "value3" = {expr3},
               {other expressions}
```

```
)
```

Якщо об'єкт object має значення `value2` то виконується код `expr2`, якщо `value1` то `expr1` відповідно і тд. У випадку коли не має співпадінь, тоді або виконується код other expressions, або результатом буде `NULL` у випадку відсутності other expressions.

```
> x <- letters[floor(1+runif(1,0,5))]

> y <- switch(x,
              a="Bonjour",
              b="Gutten Tag",
              c="Hello",
              d="Привіт",
              e="Czesc"
             )

> print(y)
```

4.3.2 Цикли з використанням for, while і repeat

Конструкції `for()`, `while()` і `repeat()` використовуються при організації циклів в R і мають наступний синтаксис:

```
1. конструкція for

for (i in for_object)
{
  some_expressions
}
```

де `some–expressions` - вираз, що виконується кожен раз для кожного i-го елементу об'єкта `for–object`

Приклад функції x^n

```
> ninpower <- function(x, power){
  for(i in 1:length(x))
  {
    x[i] <- x[i]^power
  }
 x
}

> ninpower(1:5,2)
[1]  1  4  9 16 25
```

Об'єкт `for{object` може бути вектором, масивом, дата фреймом, або списком.

2. конструкція while

```
while (logical condition)
{
    some expressions
}
```

some expressions - певний вираз виконується доти доки логічний вираз logical condition не стане FALSE.

Приклад

```
> itershow <- function(){
  tmp <- 0
  z <- 0
while(tmp < 150){
    tmp <- tmp + rbinom(1,5,0.5)
    print(tmp)
     z <- z +1
}
cat(" Для виходу з циклу виконано ",z," ітерацій \n")
}
```

3. конструкція repeat

```
repeat
{
    some expressions
}
```

Виконання виразу some expressions повторюється безкінечну кількість разів доти, поки не відбудеться виконання команди break призначеної для виходу з даної конструкції циклу.

```
> itershow2 <- function(){
  tmp <- 0
  z <- 0
repeat{
    tmp <- tmp + rbinom(1,5,0.5)
     z <- z +1
   if (tmp > 100) break
}
cat(" Для виходу з циклу виконано ",z," ітерації \n")
}
```

```
> itershow2()
  Для виходу з циклу виконано  43  ітерації
```

```
> repeat {
 g <- rnorm(1)
 if (g > 2.0) break
 cat(g);cat("\n")
}
```

Варто зазначити, що більш ефективним з точки зору розподілу ресурсу і часу необхідних на виконання коду в середовищі R є використання вбудованих функцій (наприклад `apply` функції), на відміну від вище описаних функціональних конструкцій.

4.4 Сімейство apply функцій

В R існує кілька функцій, які дозволяють уникати використання циклів. До них належать функції з сімейства функцій `apply`, а також функції `by()` і `outer()`. Використання функцій дозволяє ефективніше проводити обрахунки елементів векторів, масивів, списків, датафреймів уникаючи використання циклів. Особливо відчутно в ситуаціях де мають місце операції з великими об'ємами даних.

4.4.1 Функція apply()

Функція `apply()` дозволяє проводити операції/обрахунки на елементах (рядках або стовбчиках) масиву. Синтаксис функції має вигляд

```
apply(x, margin, fun, ...)
```

де x - масив(матриця) даних, *margin* - 1 (рядки), 2 (стовбчики) або 1:2 (по елементно), `fun` - функція, яка застосовується до елементів масиву.

Приклади

```
# середнє значення по рядках
> m <- matrix(10:29, nrow = 10, ncol = 2)
> apply(m,1,mean)
 [1] 15 16 17 18 19 20 21 22 23 24

# середнє значення по стовбчиках
> m <- matrix(10:29, nrow = 10, ncol = 2)
> apply(m,2,mean)
[1] 14.5 24.5
```

Функція apply() може використовуватися не тільки з вбудованими функціями R але й написаними безпосередньо користувачами.

```
# всі елементи масиву діляться на 10
> apply(m, 1:2, function(x) x/10)
      [,1] [,2]
 [1,]  1.0  2.0
 [2,]  1.1  2.1
 [3,]  1.2  2.2
 [4,]  1.3  2.3
 [5,]  1.4  2.4
 [6,]  1.5  2.5
 [7,]  1.6  2.6
 [8,]  1.7  2.7
 [9,]  1.8  2.8
[10,]  1.9  2.9


# обрахунок загальної кількості елементів
# які більші за 15

> tresh <- function(x,y){
>  sum(x>y)
> }

> apply(m,2,tresh,15)
[1]  4 10
```

4.4.2 Функції lapply(),sapply() і replicate()

Функція lapply() дозволяє проводити операції/обрахунки на компонентах списків.

```
> l1 <- list(a = 1:10, b = 11:20, c = 21:30)
> lapply(l1, mean)
$a
[1] 5.5

$b
[1] 15.5

$c
[1] 25.5


# застосування функції is.numeric до компонентів списку
```

```
# функція is.numeric повертає логічне значення TRUE
# якщо змінна/компонент містить числові значення

> l2 <- list(a = 1:10, b = 'Tekst', c = TRUE)
> lapply(l2, is.numeric)
$a
[1] TRUE

$b
[1] FALSE

$c
[1] FALSE
```

Функція `sapply()` може розглядуватися як спрощений варіант функції `lapply()`. На відміну від функції `lapply()` результатом виконання якої є список, функція `sapply()` повертає результат у вигляді вектора або матриці (якщо таке можливо, якщо ні то результатом буде об'єкт зі структорою списку).

```
> l1 <- list(a = 1:10, b = 11:20, c = 21:30)
> sapply(l1,mean)
   a    b    c
 5.5 15.5 25.5
```

Функція `replicate()` є свого роду обгорткою до функції `sapply()`, яка дозволяє провести серію обрахунків, здебільшого генерування серії випадкових чисел. Синтаксис функції має вигляд

```
replicate(n,expr,simplify=TRUE)
```

де n - число реплікацій (повторів), `expr` - функція або вираз що повторюється, необов'язковий параметр simplify=TRUE пробує спростити результат і представити у вигляді вектора або матриці значень.

```
> replicate(5, runif(10))

> hist(replicate(50, mean(rexp(10))))
```

4.4.3 Функція rapply()

Функція `rapply()` дозволяє застосовувати операції/обрахунки на компонентах списків і представляти результат у вигляді вектору або списку. Синтаксис функції має вигляд

```
rapply(X, FUN, how = c("unlist", "replace", "list"))
```

де x - список даних, `fun` - функція, яка застосовується до елементів масиву, *how* - аргумент, який окреслює в якому вигляді буде представлено результат (у вигляді списку або подефолту у вигляді вектора)

```
> l1 <- list(a = 1:10, b = 11:20, c = 21:30)
> rapply(l1,function(x) x^2)
a1  a2  a3  a4  a5  a6  a7  a8  a9 a10  b1  b2  b3  b4  b5  b6
 1   4   9  16  25  36  49  64  81 100 121 144 169 196 225 256
b7  b8  b9 b10 c1  c2  c3  c4  c5  c6  c7  c8  c9 c10
289 324 361 400 441 484 529 576 625 676 729 784 841 900

> rapply(l1,function(x) x^2, how="list")
$a
 [1]   1   4   9  16  25  36  49  64  81 100

$b
 [1] 121 144 169 196 225 256 289 324 361 400

$c
 [1] 441 484 529 576 625 676 729 784 841 900
```

4.4.4 Функція tapply()

Функція `tapply()` дозволяє розділити вектор даних на групи, що класифікуються певним фактором і застосувати задану функцію до цих груп. Синтаксис функції має вигляд

```
tapply(x, index, fun, ...)"
```

де x - вектор значень, *index* - фактор, тієї ж розмірності що і x, `fun` - функція що застосовується до груп, ... - додаткові аргументи.

```
> x <- sample(1:4, size=50, rep=T)
> y <- as.factor(sample(c("A","B","C","D"), size=50, replace=T))
> tapply(x,y,sum)
 A  B  C  D
29 25 30 37
```

4.4.5 Функція by()

Функція `by()` свого роду є аналогом вище згаданої функції `tapply()`, з тією різницею, що функція `by()` застосовується до датафреймів. Датафрейм розділяється/класифікується факторами на підмножину датафреймів і до кожної підмножини застосовується функція. Синтаксис `by()` має вигляд

```
by(data, indeces, fun, ...)
```

де `data` - набір даних у вигляді датафрейму, *indeces* - фактор або список факторів, `fun` - функція, що застосовується до підмножини датафреймів класифікованих фактором, ... - додаткові аргументи. Приклад використання функції `by()` для окреслення середньомісячних погодніх показників (температури ^{o}C, швидкості вітра м/с, кількості опадів мм)

```
> weather <- data.frame(temperature=round(runif(30)*30),
  windspeed=round(runif(30)*10,1),opady=runif(30)*50,
  period=sample(c('Day','Night'),30,replace=TRUE))
> res <- by(weather[,1:3],weather$period,mean)
> res
weather$period: Day
temperature   windspeed       opady
   12.94444     4.90000    26.69330
------------------------------------------------------------
weather$period: Night
temperature   windspeed       opady
   9.916667    4.041667   25.585381
```

4.4.6 Функція outer()

Функція `outer()` дозволяє виконувати операції над елементами 2-х масивів або векторів, уникаючи явного використання циклів. Синтаксис функції має вигляд

```
outer(x, y, fun="*", ...)
```

де x,y - масив або вектор даних, `fun` - задає функцію або операцію, яка виконується між елементами x на y. По дефолту виконується операція множення x та y.

```
> x <- 1:5
> y <- 1:5
> z <- outer(x,y)
> z
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    2    4    6    8   10
[3,]    3    6    9   12   15
[4,]    4    8   12   16   20
[5,]    5   10   15   20   25
```

Скороченим записом функції `outer()` є оператор

`%o%`

.

```
> x %o% y
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    2    4    6    8   10
[3,]    3    6    9   12   15
[4,]    4    8   12   16   20
[5,]    5   10   15   20   25
```

Додавання двох векторів

```
> x <- 1:5
> y <- 1:5
> z <- outer(x,y,"+")
> z
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    3    4    5    6
[2,]    3    4    5    6    7
[3,]    4    5    6    7    8
[4,]    5    6    7    8    9
[5,]    6    7    8    9   10
```

Використовуючи функцію `outer()` разом з функцією `paste()` можна генерувати усі можливі комбінацій на основі елементів векторів

```
> x <- c("A", "B", "C", "D")
> y <- 1:10
> z <- outer(x, y, paste, sep="")
> z
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] "A1" "A2" "A3" "A4" "A5" "A6" "A7" "A8" "A9" "A10"
[2,] "B1" "B2" "B3" "B4" "B5" "B6" "B7" "B8" "B9" "B10"
[3,] "C1" "C2" "C3" "C4" "C5" "C6" "C7" "C8" "C9" "C10"
[4,] "D1" "D2" "D3" "D4" "D5" "D6" "D7" "D8" "D9" "D10"
```

5

Статистика

Даний розділ демонструє, як використовуючи простий синтаксис R можна представити узагальнену статистичну інформацію, генерувати випадкові числа відповідного статистичного розподілу, спрогнозувати, допасувати і змоделювати дані, провести статистичний аналіз зв'язку/залежностей між змінними різного ступеня складності. Все це реалізується стандартними функціями пакету R. На сайті CRAN [1] існують додаткові пакети, що розширюють статистичні можливості R, які однак не є предметом розгляду даного розділу.

5.1 Основні статистичні функції

Узагальнююча (описова) статистика даних в R здійснюється за допомогою базових статистичних функцій , представлених в таблиці.

```
Базові статистичні функції

mean(x) # середнє арифметичне значень об'єкту x
sd(x)  # середньо-квадратичне відхилення x.
var(x) # дисперсія x
median(x) # медіанна x
quantile(x, probs) # квантиль змінної x, де x - числовий вектор
                   # probs - числовий вектор  ймовірностей
sum(x), # сума по x
min(x) # мінімальне значення x
max(x) # максимальне значення x
range(x)  # мінімальне і максимальне з
          # діапазону значень x
```

Приклади використання базових статистичних функцій

[1] http://cran.r-project.org/web/packages/

```
> x <- c(-5:10)

> mean(x)
[1] 2.5

> sd(x)
[1] 4.760952

> var(x)
[1] 22.66667

> median(x)
[1] 2.5

> quantile(x, c(.3,.75)) # 30-й and 75-й процентилі x
  30%   75%
-0.50  6.25

> sum(x)
[1] 40

> min(x)
[1] -5

> max(x)
[1] 10

> range(x)
[1] -5 10
```

Базові статистичні функції використовуються як окремо так і в комбінації з `apply` функціями, що дозволяє застосовувати їх до списків або датафреймів. Нехай маємо курси валют: американського долара - USD,євро - EURO, швейцарського франка - CHF, за період з 06.09.2010 по 10.09.2010 у вигляді датафрейму. Представимо середнє значення курсів валют протягом вибраного тижня з використанням функції `sapply()`

```
> EURO <- c(1014.9384, 1018.1017, 1007.821, 1004.1042,1005.5276)
> USD <- c(790.82,790.82,790.82,790.82,790.82)
> CHF <- c(778.148,778.9607,781.075,782.1953,781.9641)
> kursy <- c(EURO,USD,CHF)

> sapply(kursy,mean)
     EURO       USD       CHF
1010.0986  790.8200  780.4686
```

Крім того в R існують вбудовані узагальнюючі, так звані, generic функції `summary()`, `fivenum()` призначені для розрахунку і одночасного предеставлення значень основних статистичних показників. Результатом використання функції `summary()` є значення максимального і мінімального, середнього статистичного, медіани, 1-го та 3-го квартилів

```
> summary(EURO)
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
1004    1006    1008    1010    1015    1018
```

Функція `fivenum()` є альтернативною generic функцією, повертає значення мінімального, 3-х найважливіших квантилів (1-го, медіани та 3-го квартилів) і максимальне значення.

```
> fivenum(EURO)
[1] 1004.104 1005.528 1007.821 1014.938 1018.102
```

5.2 Функції розподілу ймовірностей

В середовищі R реалізовано набір функцій густини, функції розподілу, квантилів більшості розподілів ймовірностей. Назви майже всіх функцій розподілу ймовірностей наслідують наступну конвенцію запису: складаються з першої літери відповідної функції

Функція густини ймовірностей `d`
Функції розподілу ймовірностей `p`
Квантильної функції `q`
Генератор випадкових чисел `r`

і скороченої назви розподілу ймовірностей.
Кілька прикладів поширених на практиці розподілів
Функції нормального розподілу

```
dnorm(x) #  Нормальний розподіл або розподіл Гауса
pnorm(q) # (комулятивна) функція розподілу ймовірностей
qnorm(p) # квантиль нормального розподілу
rnorm(n, m=0,sd=1)  # генератор n випадкових величин з
                    # нормального розподілу
                    # де аргументи m - середнє арифметичне,
                    # sd - середньо-квадратичне відхилення.
```

Приклади

```
> x <- seq(-10,10,by=.1)
> y1 <- dnorm(x)
```

```
> plot(x,y1)

> y2 <- pnorm(x)
> plot(x,y2)

> y3 <- qnorm(x)
> plot(x, y3)

> z <- rnorm(1000) # генерація 1000 випадкових чисел
            # з нормального розподілу (m = 0, sd= 1)
> hist(z,breaks=50)
```

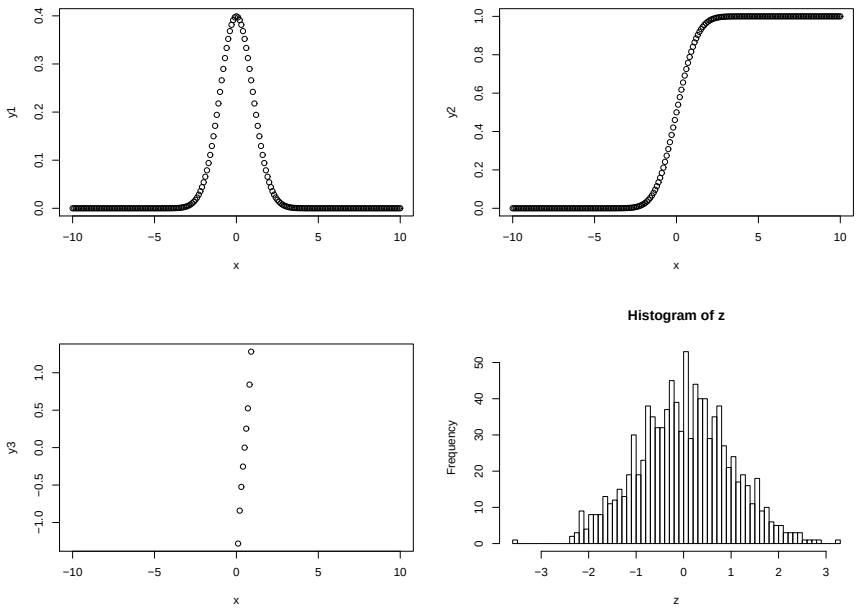

Рис. 5.1. Рисунок графіків нормального розподілу

Функції рівномірного розподілу

```
dunif(x, min=0, max=1) # Рівномірний розподіл
punif(q, min=0, max=1) # Функція розподілу
qunif(p, min=0, max=1) # квантиль ріномірного розподілу
runif(n, min=0, max=1) # Генератор випадкових чисел
```

Приклади

```
> x <- seq(-10,10,by=0.01)
> y1 <- dunif(x)
> plot(x, y1)
```

```
> y2 <- punif(x)
> plot(x, y2)

> x <- seq(-1,1,by=0.01)
> y3 <- qnorm(x)
> plot(x, y3)

> z <- runif(100)
> hist(z)
```

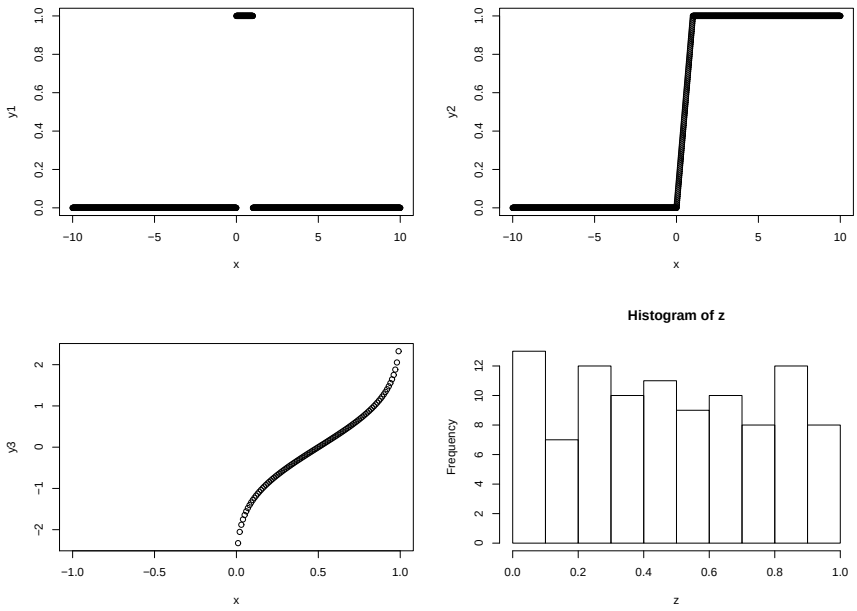

Рис. 5.2. Рисунок графіків рівномірного розподілу

Функції Log-нормального розподілу

```
dlnorm(x, meanlog = 0, sdlog = 1) # Log-нормальний розподіл
plnorm(q, meanlog = 0, sdlog = 1) # Функція розподілу
qlnorm(p, meanlog = 0, sdlog = 1) # квантиль ріномірного розподілу
rlnorm(n, meanlog = 0, sdlog = 1) # Генератор випадкових чисел
```

Приклади

```
> x <- seq(0,10, by=0.01)
> y1<-dlnorm(x)
```

```
> plot(x,y1)

> y2<-plnorm(x)
> plot(x,y2)

> y3<-qlnorm(x)
> plot(x,y3)

> z <- rlnorm(100)
> hist(z)
```

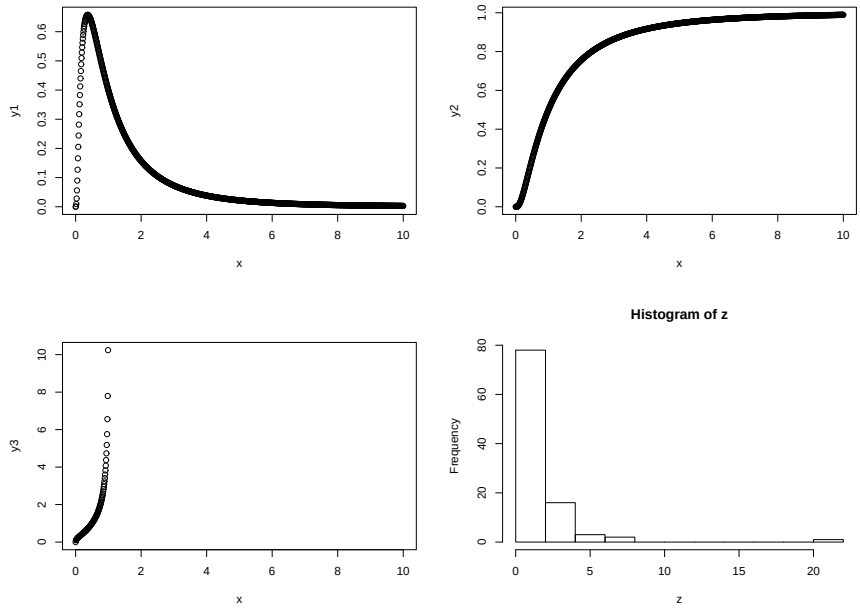

Рис. 5.3. Рисунок графіків Log-нормального розподілу

Функції експоненційного (експоненціального) розподілу

```
dexp(x, rate = 1)  # експоненційний (експоненціальний) розподіл
pexp(q, rate = 1)  # Функція розподілу
qexp(p, rate = 1)  # квантиль експоненційного розподілу
rexp(n, rate = 1)  # генератор випадковий чисел
```

Приклади

```
> x <- seq(0,10, by=0.01)
```

```
> y1 <- dexp(x)
> plot(x, y1)

> y2 <- pexp(x)
> plot(x, y2)

> z <- rexp(100)
> hist(z)
```

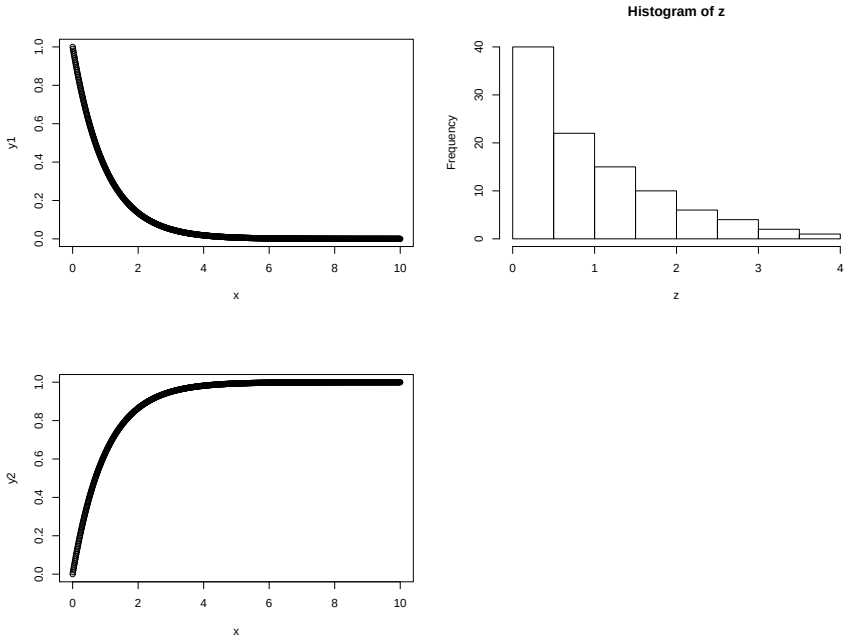

Рис. 5.4. Рисунок графіків експоненційного розподілу

Інформація про доступні для використання в середовищі R розподіли ймовірностей (що входять до базового пакету `stats`), реалізованих у відповідності до вищезгаданої конвенції запису функцій розподілу ймовірностей, представленна в таблиці

Тип розподілу (варіант англ.)	Назва в R
Бета (beta)	beta
Біноміальний (binomial)	binom
Коші-Лоренца (Cauchy)	cauchy
χ^{2} (chi-squared)	chisq
Експоненційний (exponential)	exp
Фішера (F)	f
Гамма (gamma)	gamma

```
Геометричний (geometric)           geom
Гіпергеометричний (hypergeometric) hyper
Log-нормальний (log-normal)        lnorm
Логістичний (logistic)             logis
Нормальний (normal)                norm
Пуассона (Poisson)                 pois
T-розподіл Стьюдента (T)           t
Рівномірний (uniform)              unif
Вейбула (Weibull)                  weibull
```

В R існує функція `sample()` за допомогою якої, можна генерувати випадкові дані, у вигляді вектора даних, на основі значень іншого вектора. Приклад

```
> sample(1:6,15,replace=T)
 [1] 3 6 4 1 5 5 6 3 2 1 1 1 3 6 1
```

симулює і відповідно генерує значення 6-граного кубика, в даному випадку після 15-ти "віртуальних кидків". По дефолту функція `sample()` генерує значення, що неповторюються двічі. Аргумент replace=T вказує, що значення можуть повторюватися.

В R імплементовано значну кількість алгоритмів, які дозволяють генерувати випадкові числа. Більше інформації

```
> ?set.seed
```

5.3 Регресійний аналіз

Регресія або регресійний аналіз сукупність статистичних методів, що застосовуються для знаходження, аналізу, моделювання і прогнозування залежностей між змінними. В середовищі R існує широкий набір функцій (`lm()`,`aov()`,`glm()`) для проведення різних видів регресійного аналізу. Кожна з функцій містить ключовий параметр `formula` - модель, згідно якої аналізуються дані.

5.3.1 Лінійна регресія

Базовою є лінійна регресія, яка моделює лінійну залежність між змінною y і незалежними змінними $x_1, ..., x_n$. В загальному випадку (лінійної множинної регресії), залежність між змінними має вигляд

$$y_i = b_0 + b_{i,1}x_{i,1} + b_{i,2}x_{i,2} + ... + b_{i,j}x_{i,j} + \epsilon_i$$

де y_i - змінна, значення якої спостерігається для i-вої обсервації, $b_{i,j}$ - коєфіцієнти (параметри) регресії, $x_{i,j}$ - незалежні змінні, предиктори,

$\epsilon_1, ..., \epsilon_n$ - залишкові компоненти, незалежні однаково розподілені випадкові величини, інтерпретуються як шум або помилки.

Лінійний регресійний аналіз в R реалізується з використанням функції `lm()` . Синтаксис функції `lm()` має вигляд

```
lm(formula, data, ...)
```

де `formula` - символьний опис моделі, `data` - об'єкт даних (датафрейм), `...` - додаткові аргументи.

Параметр `formula` відіграє важливу роль в R окреслюючи модель, тобто зв'язок між змінними, згідно якої аналізуються данi`data`. Достовірність результатів регресійного аналізу залежить від вибраної моделі.

В найпростішому випадку лінійної залежності

$$y_i = b_0 + b_1 x_i + \epsilon_i$$

формула моделі має вигляд

```
y ~ x
```

Наступний приклад демонструє використання базових функцій для проведення лінійного регресійного аналізу статистичних даних `longley`.

```
> dani.lm <- lm(GNP ~ Population,data = longley)
> dani.lm

Call:
lm(formula = GNP ~ Population, data = longley)

Coefficients:
(Intercept)   Population
   -1275.21        14.16
```

Результат використання функції `lm` зберігається в даному випадку в об'єкті `dani.lm`, який являє собою об'єкт класу 'lm'. Щоб вивести повну інформацію про об'єкт `dani.lm` можна скористатися функцією `print.default()` .

```
> print.default(dani.lm)
$coefficients
(Intercept)  Population
-1275.21004    14.16157

$residuals
      1947       1948       1949       1950       1951
-14.399443  -3.763893 -21.294247 -11.120025  17.026813
```

```
   1952       1953
18.127734  10.683026
...................................................

1962 554.894    130.081

attr(,"class")
[1] "lm"
```

Тобто об'єкт класу `lm`, в даному випадку dani.lm, насправді містить значно більше інформації.

Використовуючи спеціальні, вже згадуванні generic функції, результат виконання яких залежить від типу об'єкту даних або класу об'єкту, можна отримати узагальнюючу статистичну інформацію результатів аналізу, спрогнозувати значення, побудувати діагностичні графіки і тд. Приклади деяких generic функцій, основною з яких є функція `summary()` представлено в Таблиці

generic функція	результат
summary(object)	узагальнена статистична інформація обєкту
coef(object)	коефіцієнти регресії
resid(object)	залишки
fitted(object)	допасовані/встановленні значення
anova(object)	таблиця дисперсійного аналізу
predict(object)	прогнозовані значення
plot(object)	створення діагностичних графіків
aov(object)	ANOVA аналіз
glm(object)	узагальнений лінійний аналіз

Функція `summary()` в випадку застосування її до обєктів класу `lm` повертає формулу моделі, залишки (residuals), коєфіцієнти регресії, середньоквадратичне відхилення оцінки регресії, коєфіцієнт детермінації R^2, статистику дисперсійного аналізу (F-statistic)

```
> summary(dani.lm)
```

```
Call:
lm(formula = GNP ~ Population, data = longley)

Residuals:
     Min       1Q   Median       3Q      Max
-21.2942 -11.3519  -0.6804  11.2152  18.1277

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1275.2100    59.8256  -21.32 4.53e-12 ***
Population     14.1616     0.5086   27.84 1.17e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 13.7 on 14 degrees of freedom
Multiple R-squared: 0.9823,Adjusted R-squared: 0.981
F-statistic: 775.2 on 1 and 14 DF,  p-value: 1.168e-13
```

Один із способів перевірки достовірності вибраного моделю, заданого параметром formula, є представлення результатів аналізу в графічному вигляді. Застосування generic функції `plot()` до обєктів класу 'lm'

```
> par(mfrow=c(2,2))
> plot(dani.lm)
```

дозволяє отримати наступні діагностичні графіки рис.5.5

В R існують функції, що дозволяють оновити або змінити параметри лінійного аналізу. Функція `update()` використовується для оновлення або зміни моделі лінійного аналізу. Результатом застосування функції `update()` є об'єкт класу `'lm'`. Конструкція `.+Armed.Forces` у вигляді параметра

```
> dani.lm2year <- update(dani.lm, ~.+Year)
```

дозволяє оновити модель лінійного аналізу, з урахуванням доданої змінної Year. Результатом виконання в даному випадку буде об'єкт `dani.lm2year`

```
> dani.lm2year

Call:
lm(formula = GNP ~ Population + Year, data = longley)

Coefficients:
(Intercept)   Population         Year
 -34306.363        2.175       17.620
```

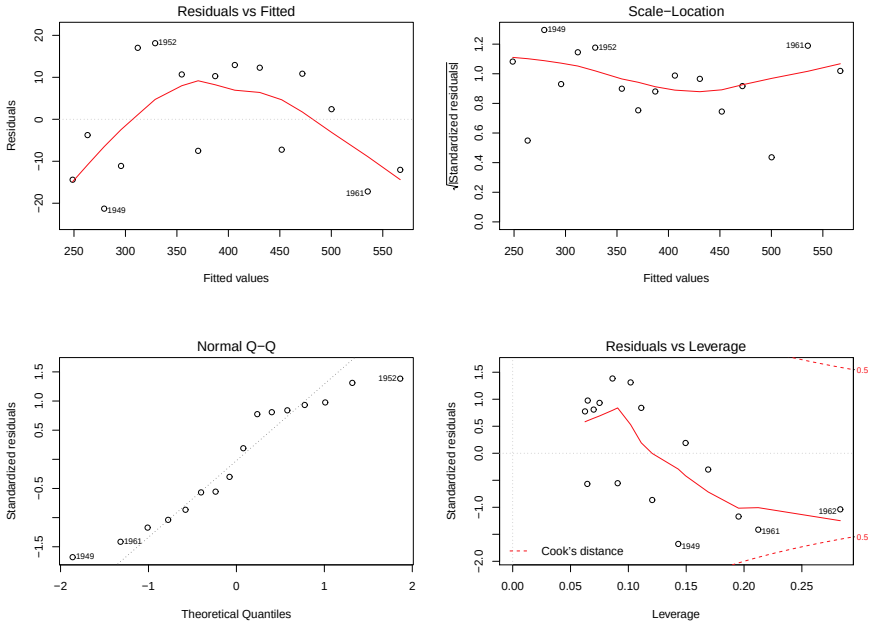

Рис. 5.5. Графіки результатів регресійного аналізу

Функція `add1()` включає додатковий аргумент (змінну) до аналізованої моделі і дозволяє спостерігати за змінами значень суми квадратів і залишкових сум.

```
> dani.lm1 <- add1(dani.lm, ~.+Year)
> dani.lm1
Single term additions

Model:
GNP ~ Population
       Df Sum of Sq    RSS    AIC
<none>              2629.0 85.628
Year    1    1272.8 1356.2 77.037
```

Функція `drop1()` дозволяє спостерігати за змінами значень суми квадратів і залишкових сум усуваючи змінні, окрім необхідних, з моделі.

```
> drop1(dani.lm2year,~ Population)
Single term deletions

Model:
GNP ~ Population + Year
```

```
           Df Sum of Sq    RSS    AIC
<none>                  1356.2 77.037
Population  1     41.39 1397.5 75.518
```

5.3.2 Нелінійна регресія

Функція `nls()` дозволяє провести нелінійний регресійний аналіз і знайти параметри моделі нелінійної регресії виду

$$y_i = f(x_i, \beta) + \epsilon_i$$

де y_i - змінна, значення якої спостерігається для i - ї обсервації, `f` - певна нелінійна функція, x_i - незалежні змінні, предиктори, $\beta = (\beta_1, ..., \beta_p)$ - коєфіцієнти (параметри) регресії, що розраховуються/допасовуються на основі даних (y_i,x_i), $\epsilon_1, ..., \epsilon_n$ - залишкові компоненти, незалежні однаково розподілені випадкові величини, інтерпретуються як шум або помилки. Синтаксис функції `nls()` має вигляд

```
nls(formula, data, start,...)
```

де `formula` - формула нелінійної моделі , `data` - дані у вигляді датафрейму, списку, `start` - вектор початкових (приблизних) значеннь коєфіцієнтів регресії, `...` - додаткові аргументи.

На відміну від лінійної регресії, formula моделі нелінійної регресії має звичайну математичну нотацію. Наприклад, для нелінійної моделі

$$y_i = \beta_1(1 - exp^{-b2*x_i}) + \epsilon_i$$

формула моделі матиме вигляд

```
y ~ b1*(1-exp(-b2*x))
```

Нехай маємо певну зашумлену нелінійну залежість

```
> x <- runif(100,0,15)
> y <- 2*x/(5+x)
> y <- y +rnorm(100,0,0.15)
> xy.dani <- data.frame (x=x,y=y)
> plot(xy.dani)
```

Допасування даних, представлених в графічному вигляді і знаходження коєфіцієнтів нелінійної регресії, можна реалізувати з використанням наступної моделі

$$y_i = \frac{\beta_1 x_1}{\beta_2 x_2} + \beta_3 + \epsilon_i$$

і, наприклад, наступних початкових значень коєфіцієнтів регресії $\beta_1 = 1.5$, $\beta_2 = 3$

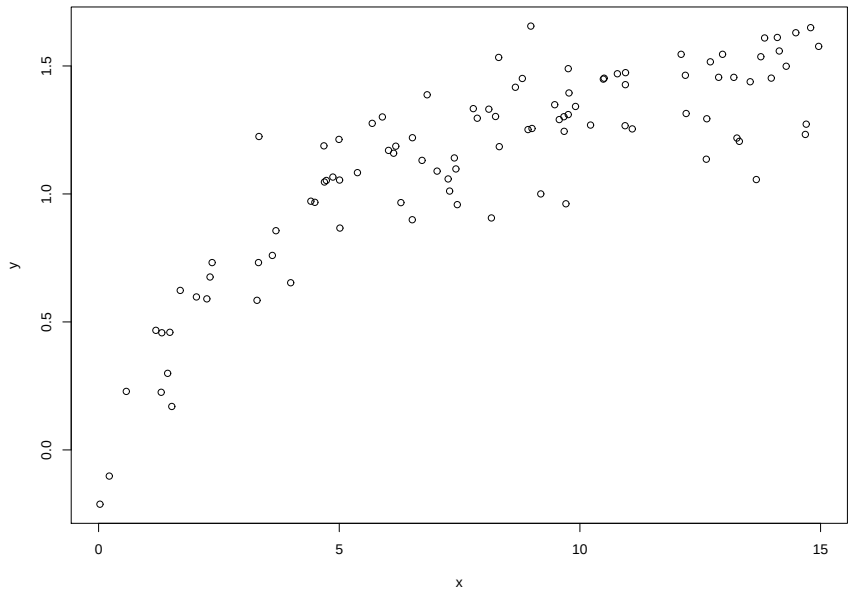

Рис. 5.6. Графік даних нелінійної залежності

```
> fitnl <- nls(
y ~ beta1*x /(beta2 + x),
start = list( beta1 = 1.5, beta2 = 3), # декларуються  початкові
data=xy.da)                            # (приблизні) значення
```

Результатом виконання функції `nls()` є об'єкт класу '*nls*'

```
> fitnl
Nonlinear regression model
  model:  y ~ beta1 * x/(beta2 + x)
   data:  xy.dani
beta1 beta2
2.014 4.995
 residual sum-of-squares: 2.092

Number of iterations to convergence: 3
Achieved convergence tolerance: 8.634e-07
```

Проглянути результат нелінійного регресійного аналізу в узагальненому структурованому вигляді можна використовуючи generic функцію `summary()`

```
> Formula: y ~ beta1 * x/(beta2 + x)
```

```
Parameters:
      Estimate Std. Error t value Pr(>|t|)
beta1  2.01433    0.08791  22.914  < 2e-16 ***
beta2  4.99545    0.60934   8.198 9.57e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1461 on 98 degrees of freedom

Number of iterations to convergence: 3
Achieved convergence tolerance: 8.634e-07
```

Використовуючи generic функцію `predict()` на основі результатів проведеного аналізу можна спрогнозувати/обрахувати інші значення і властиві їм похибки.

```
> x <- seq(0,15,by=0.1)
> prog.dani <- data.frame(x=x)
> y.zprog <- predict(fitnl, newdata = prog.dani)
> prog.dani$yzprog <- y.zprog
> lines(prog.dani$x,prog.dani$yzprog,cor="blue")
```

Для порівняння спрогнозовані значення можна представити в графічному вигляді

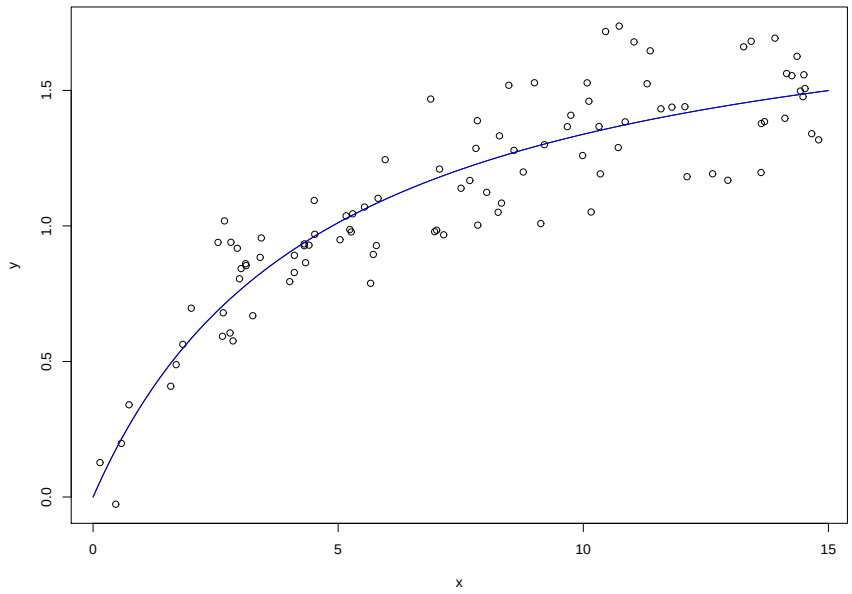

Рис. 5.7. Графік залежності y = 2.01433 * x/(4.99545 + x) допасованої в результаті нелінійного регресійного аналізу до даних представлених на Рис.5.6

6

Графіки і графічні параметри

Однією з сильних сторін R є широкі можливості представлення даних в графічному вигляді. Середовище R містить два базові пакети **graphics** та **lattice** для представлення даних в графічному вигляді. Пакет **graphics** містить інтуїтивно зрозумілий набір команд, який дозволяє будувати графіки різних типів, а також редагувати атрибути графіків, в той час як пакет **lattice** містить альтернативний набір функцій для побудови складних графіків. Даний підрозділ представляє огляд графічних функцій, а також приклади їх використання, які демонструють основні можливості пакету **graphics**.

6.1 Типи графіків

6.1.1 Функція `plot()`

Графічні об'єкти в R створюється за допомогою функції `plot()`. Функція `plot()` є основною графічною функцією з широким набором аргументів, що дозволяє використовувати її для побудови різних типів графіків. Функція `plot()` також є *generic* функцією, тобто результат її використання (у вигляді відповідного графіку або набору графіків) залежить від об'єкту даних до яких застосовується функція.

Загальний вигляд команди

```
plot(x,y,...)
```

де *x*, *y* - змінні (координати змінних) на графіку, ... - додаткові параметри. Кілька основних параметрів

- *type* - задає тип графіка: точковий (*type='p'* по дефолту), лінійний (*type='l'*), точки з'єднані лініями (*type='b'*), ступінчатий (*type='s'*) та інші. Використання функції `plot()` з параметром *(type='n')* створює графік зі всіма його атрибутами (осями, назвами і тд.) не відображаючи при цьому дані (у вигляді точок, символів, ліній).
- *xlim, ylim* - встановлює діапазон значень x і y осей, відповідно.

- *col* - встановлює колір графіку.
- *log* - дозволяє встановити логаріфмічну шкалу осей (log='y', log='x', log='xy')

Більше інформації

```
> ?plot
```

Приклад побудови графіка за допомогою функції `plot()`

```
> plot(10^c(1:9),ylim=c(10^3,10^9),type="b",col='red',log="y")
```

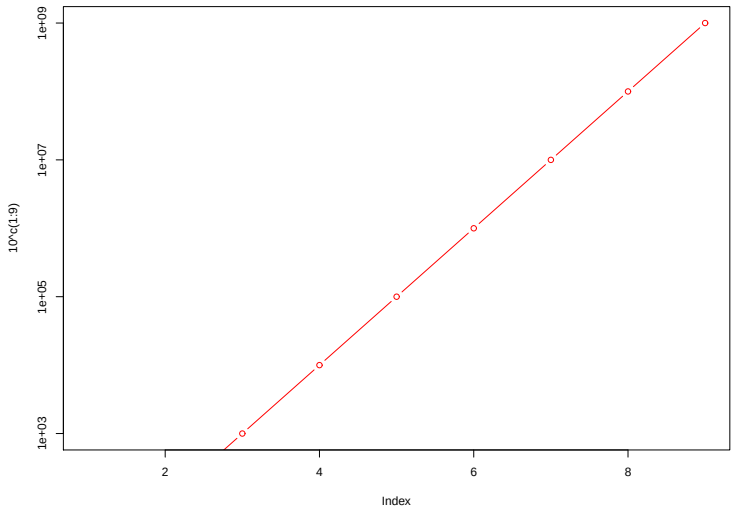

Рис. 6.1. Графік побудований з використанням функції `plot()`

Функція `plot()` використовується для побудови графіків на основі даних. Для побудови графіку певної математичної функції або виразу використовується графічна функція `curve()`.

```
> f <- function(x) exp(x) * cos(2*pi*x)
> curve(f, 0, 10, main='Grafik funkcii exp(x) * cos(2*pi*x)')
```

6.1.2 Лінійні графіки

Лінійні графіки створюються використовуючи базову функцію `plot()` з означеним параметром *type='l'* або з використанням функції `lines()`, яка

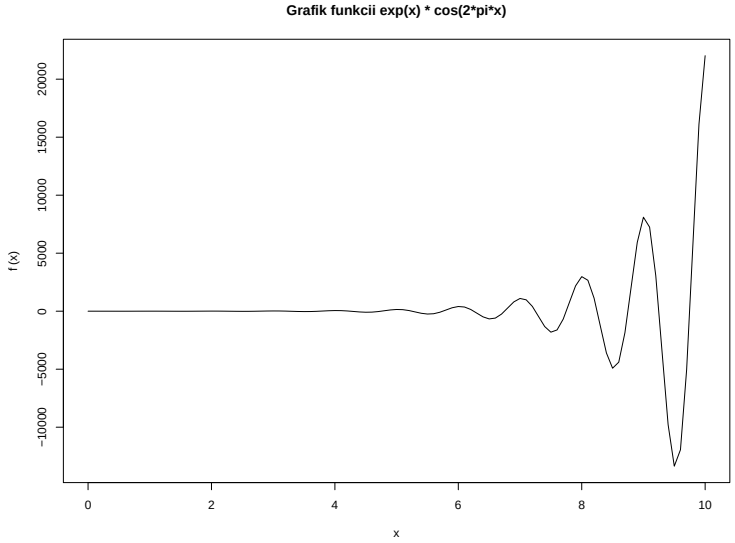

Рис. 6.2. Графік $e^x * cos(2\pi x)$ побудований з використанням функції `curve()`.

застосовується до накладання ліній до вже існуючого графіку. Загальний синтаксис функції `lines()` має вигляд

```
lines(x,y,...)
```

де - x,y - змінні (координати змінних) на графіку, ... - графічні параметри.

```
> plot(c(2,0),c(-1,2), type="l")
```

Побудова лінійного графіку з використанням функції `lines()`

```
> plot(c(2,0),c(-1,2))
> lines(c(0,0,2,0), c(2,1,-1,1), col="red")
```

6.1.3 Гістограми і графіки густини розподілу

Гістограма створюється за допомогою функції `hist()`,

```
hist(x,...)
```

де `x` - вектор числових значень, ... - додаткові параметри.

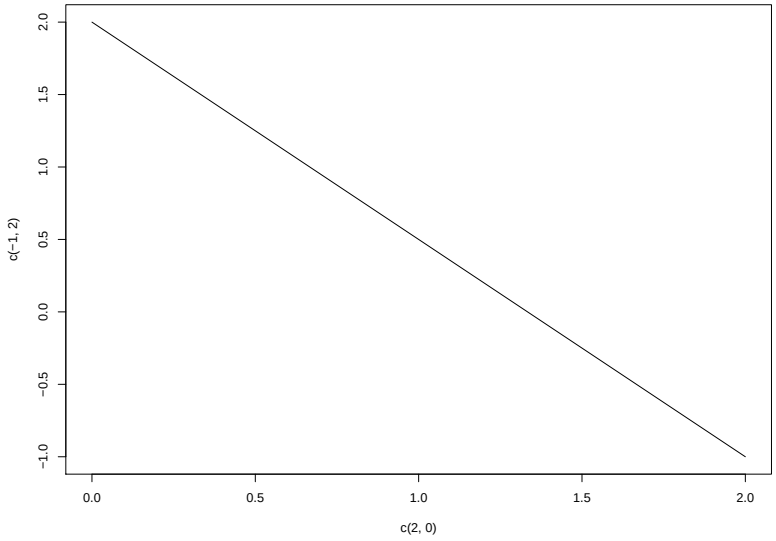

Рис. 6.3. Лінійний графік побудований з використанням функції `plot()`

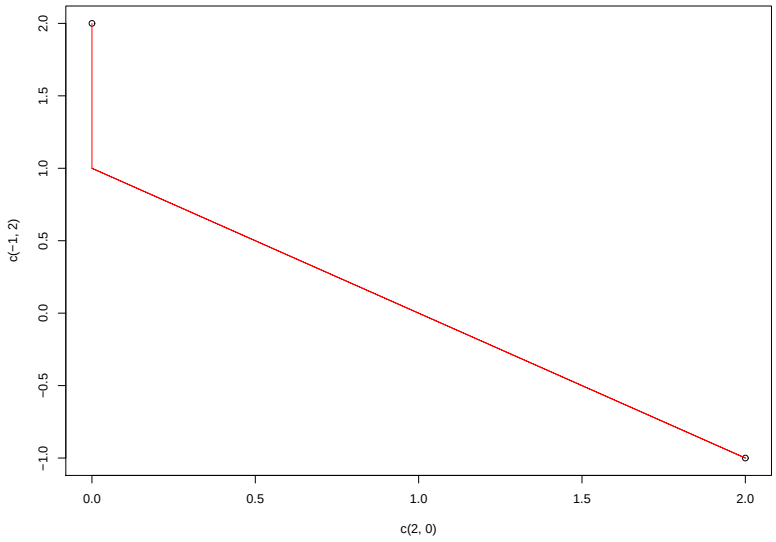

Рис. 6.4. Лінійний графік побудований з використанням функції `plot()`

```
> x <- rnorm(100)
> hist(x)
```

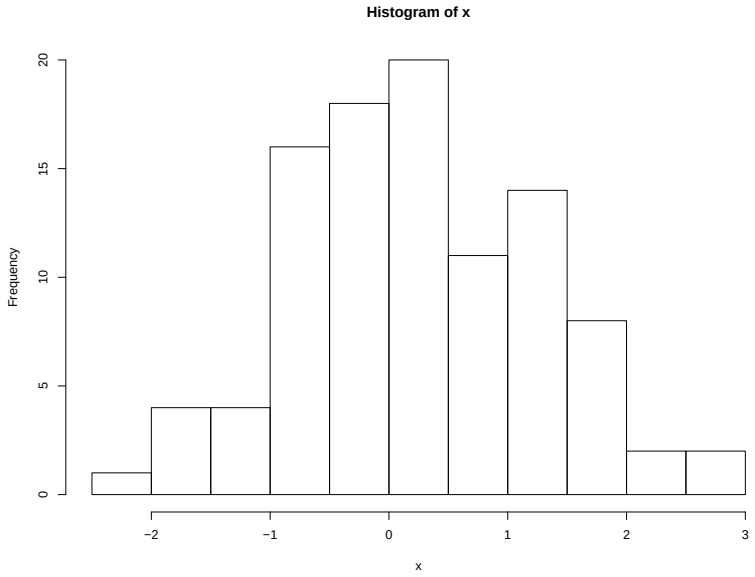

Рис. 6.5. Гістограма

Кольорова гістограма з заданою кількістю інтервалів отримується окреслюючи додаткові параметри

```
hist(x, breaks=20, col="cyan")
```

Для визначення форми розподілу змінної часто використовуються графіки густини розподілу. В R графік густини розподілу створюється за допомогою команди

```
plot(density(x))
```

де x - вектор числових значень. Приклад графіка густини розподілу випадкових величин

```
> d1 <- density(rnorm(100))
> plot(d1)
```

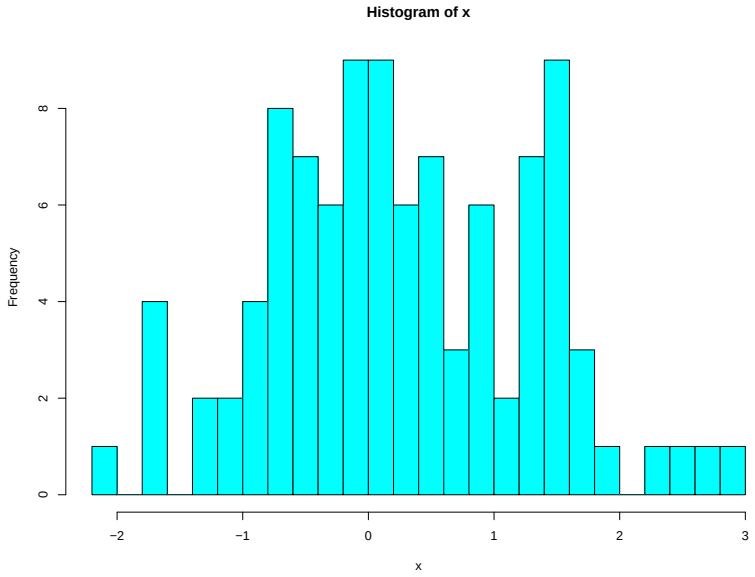

Рис. 6.6. Кольорова гістограма із заданою кількістю інтервалів

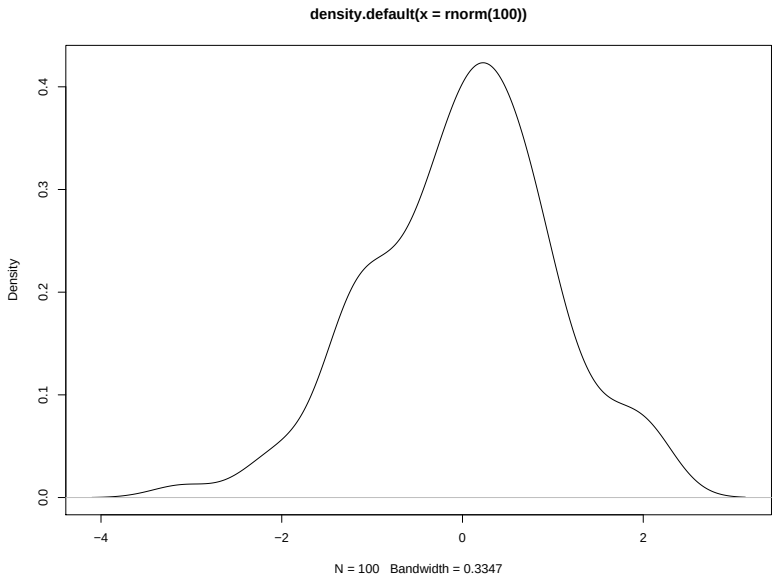

Рис. 6.7. Графік густини нормального розподілу

Приклад графік густини розподілу із кольоровим заповненням

```
> d2 <- density(runif(100))
> plot(d2)
> polygon(d2, col="black", border="red")
```

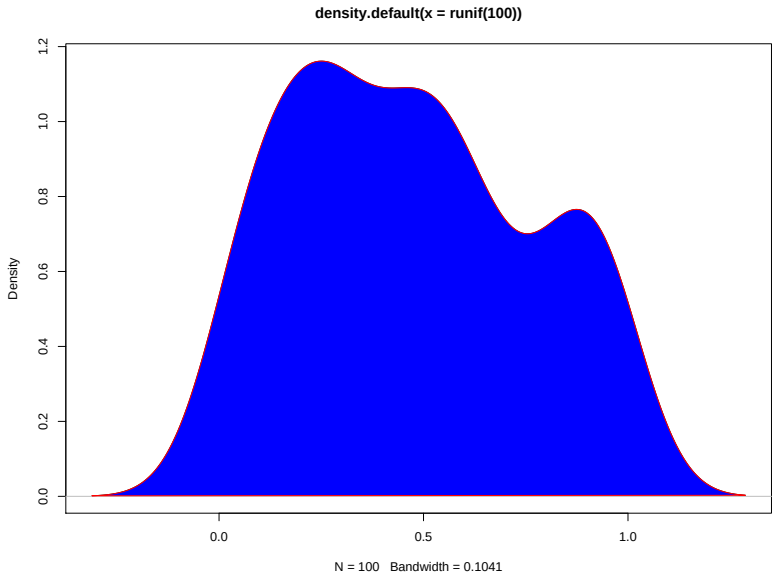

Рис. 6.8. Графік густини нормального розподілу з кольоровим заповненням

6.1.4 Q-Q(Квантиль-Квантильний) графік

Q-Q графіки дозволяють порівняти розподіл значень певного набору (емпіричних) даних з вибраним теоретичним розподілом. Якщо обидва розподіли подібні між собою, тоді точки на Q-Q графіку вистроюються у вигляді прямої лінії, якщо розподіли не схожі то точки на графіку нагадуватимуть криву.

Для побудови Q-Q графіків в R існує функція `qqplot()`, `qqnorm()` і `qqline()`. Функція `qqplot()` дозволяє отримати Q-Q графік з двох наборів даних. Синтаксис функції

```
qqplot(x,y,...)
```

Функція `qqnorm()` створює Q-Q графік, де в якості теоретичного розподілу заданий нормальний розподіл.

```
qqnorm(y, ...)
```

Функція `qqline()` додає лінію яка проходить через перший (25) і четвертий (75) квартилі (перцентилі) на Q-Q графік

```
qqline(y, datax = FALSE, ...)
```

Приклад Q-Q графіку

```
> x<- rnorm(100)
> qqnorm(x)
> qqline(x)
```

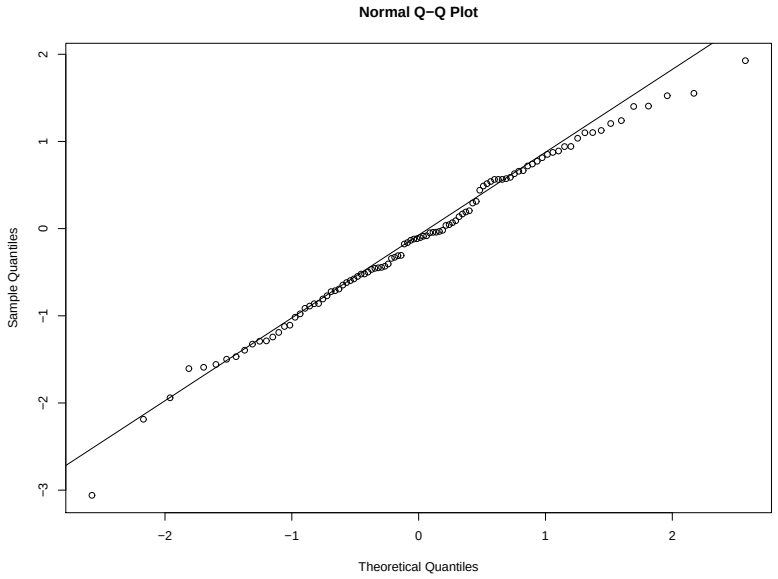

Рис. 6.9. Q-Q графік побудований за допомогою функцій `qqnorm()` та `qqline()`.

6.1.5 Точкові графіки

Точкові графіки створюються за допомогою графічної функції `dotchart()` синтаксис якої має вигляд

```
dotchart(x, labels, ...)
```

де x - вектор числових значень, *labels* - вектор міток/назв кожної точки, . . . - додаткові параметри. Наприклад, параметр *groups* дозволяє встановити критерій за яким можна згрупувати елементи вектора x. Більше інформації див. `help(dotchart)`. Приклад побудови точкового графіку на основі даних **longley**, що входить до середовища R

```
> dotchart(Armed.Forces,label=Year,cex=.9,
  main="Кількість солдатів збройних сил по рокам, тис",
  xlab="Кількість солдатів збройних сил,тис")
```

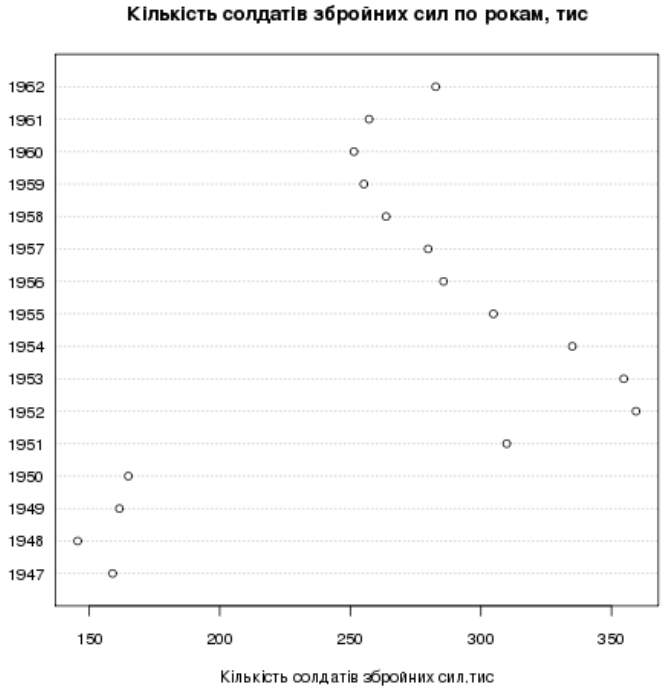

Рис. 6.10. Точковий графік побудований за допомогою функції `dotchart()`.

6.1.6 Стовпчикові діаграми

Стовпчикові діаграми будуються за допомогою функції `barplot()`

```
barplot(height,...),
```

де *height* - вектор або матриця чисел, . . . - додаткові параметри. У випадку вектора, діаграма складається з послідовності прямокутних стовбчиків, причому значення елементів вектора визначають висоту стовпчиків.

У випадку матриці разом з параметром *beside*=TRUE суміжні рівні/колонки матриці відображаються у вигляді окремо згрупованих стовпчиків діаграми. Використання *beside*=FALSE дозволяє представлення різних рівнів у вигляді підстовпичків складених в один окремий стовпчик.

```
> a <- c(10, 13, 7)
> b <- c(4, 9, 10)
> barplot(rbind(a, b), beside=TRUE, names=c("I","II","III"),
> + col=c("blue","magenta"))
```

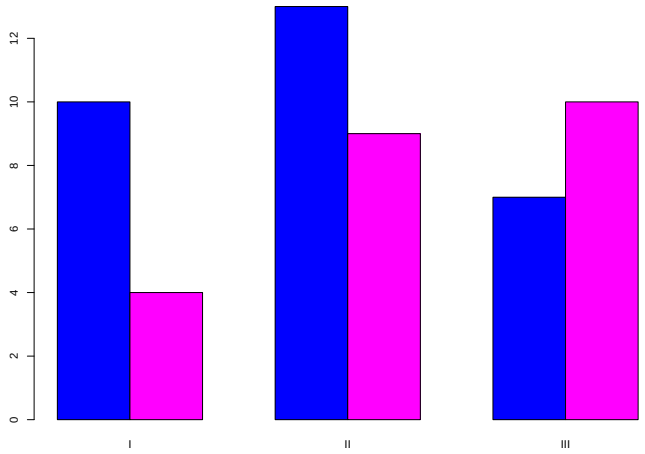

Рис. 6.11. Стовпчикова діаграма побудована за допомогою функції `barplot()`

6.1.7 Кругові діаграми

Кругові діаграми в R створюються за допомогою функції `pie(x, labels)` , де *x* - вектор числових значень (причому *x(i)*¿=0), *labels* - вектор символьного типу, елементами якого є імена секторів діаграми.

Кілька прикладів побудови кругових діаграм за допомогою функції `pie()`

Проста кругова діаграма

```
> slices <- c(10,4,4,6,14,16)
> lbls <- c("Belarus","Estonia","Lithuania","Latvia","Poland",
>  "Ukraine")
> pie(slices, lbls, main="Pie Chart of Countries")
```

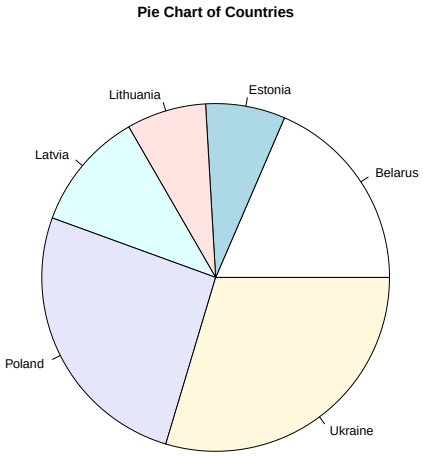

Рис. 6.12. Кругова діаграма побудована за допомогою функції `pie()`.

Кругові діаграми з вказаними процентами

```
> slices <- c(10, 4,4,6, 14, 16)
> lbls <- c("Belarus","Estonia","Lithuania","Latvia","Poland","Ukraine")
> pct <- round(slices/sum(slices)*100)
> lbls <- paste(lbls, pct)
> lbls <- paste(lbls,"%",sep="") # додає % до значень
> pie(slices,labels = lbls, col=rainbow(length(lbls)),
    main="Pie Chart of Countries")
```

Існує можливість побудови 3D кругові діаграми підключивши до середовища R пакет **plotrix** і викликавши функцію `pie3D()`.

6.1.8 Боксплоти або скринькові діаграми

Боксплоти використовуються для візуального представленя узагальненої статистичної інформації у вигляді комбінації статистичних характеристик: найменшого спостережуваного значення (min), 0.25 квантиля (Q1), медіани (Q2), 0,75 квантиля (Q3), найбільшого спостережуваного значення (max). Синтаксис функції виглядає

```
boxplot(x,...,horizontal,...)
```

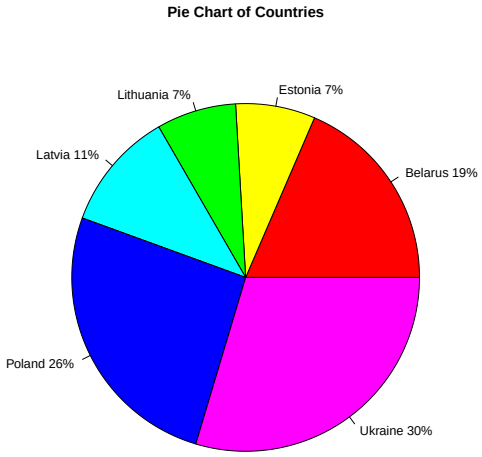

Рис. 6.13. Кругова діаграма з вказаними значеннями побудована за допомогою функції `pie()`.

де x - вектор значень (даних), *horizontal* - параметр, який задає розташування боксплотів на графіку (дефолтне значення 'FLASE' - вертикальне розташування),... - додаткові аргументи (див. `?boxplot`).

Нехай маємо певний набір статистичних даних.

```
> (dani <- c(1,3,5,-10,9,15,7,4,25))
[1]   1   3   5 -10   9  15   7   4  25
```

і відповідно, побудований з використанням функції `boxplot()` боксплот Рис. 6.14

```
> boxplot(dani)
```

- Товста лінія на рівні y=5 є медіаною.
- Нижня і верхня сторона рамки, що оточує медіану відповідає за перший Q1 і третій Q3 квартилі,відповідно.
- "Вуса"з обох сторін рамки показують діапазон значень даних, невключаючи девіантні значення (outliers).
- Девіантні значення (кільця на графіку) - значення, що істотньо віддалені від решти значень даних.

6.1.9 Порівняльні діаграми

Порівняльні діаграми використовуються для демонстрації і дослідження залежності змінних від певного фактора (факторів). Кожна окрема діагр-

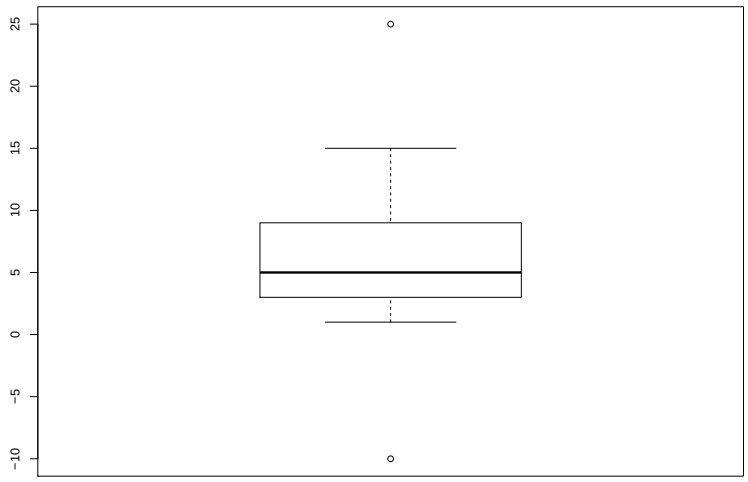

Рис. 6.14. Боксплот побудований за допомогою функції `boxplot()`

ама показує залежність між змінними відповідного рівня фактора. Синтаксис функції для побудови порівняльних діаграм має вигляд `coplot()`

```
coplot(formula, data, ...)
```

де *formula* - задає тип порівняльних діаграм (однофакторні $y \quad x$ — *fact1* або двофакторні $y \quad x$ — *fact1*fact2*), *fact1*,*fact2* - певні фактори (категорії), *data* - дані, ... - додаткові аргументи. Більше про аргументи `?coplot`.

Рис 6.15 представляє річні порівняльні діаграми залежності ВНП (Валового Національного Продукту) від кількості населення отримані за допомогою

```
> coplot(GNP~Population | Year,data=longley)
```

6.1.10 Матриці діаграм розсіювання

Функція `pairs()` дозволяє створити матрицю точкових графіків або діаграм розсіювання. Синтаксис даної графічної функції має вигляд

```
pairs(x, ...)
```

де x - кординати точок, представлені як елементи матриць або датафреймів. Діаграму розсіювання на основі даних з набору longley демонструє наступний приклад

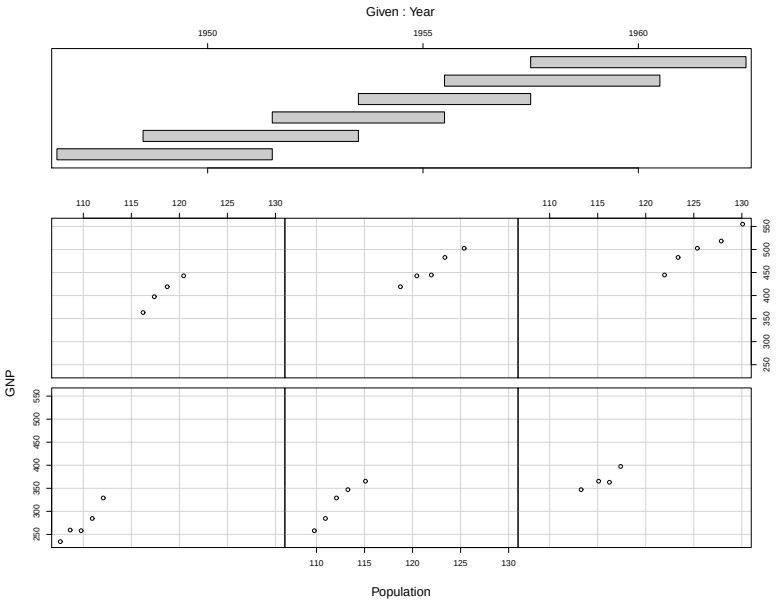

Рис. 6.15. Графік порівняльних діаграм побудований за допомогою функції `coplot()` і набору даних `longley`

```
> pairs(~Population+Employed+Unemployed+Armed.Forces,data=longley)
```

6.1.11 Точкові графіки високої щільності

В ситуації коли існує велика кількість точок і спостерігається значне перекриття між точками застовується прозорість кольорів. Даний підхід дозволяє розрізнити точки, які взаємно накладаються.

```
> x <- rnorm(1000)
> y <- rnorm(1000)
> plot(x,y,col=rgb(0,100,0,50,maxColorValue=255), pch=16)
```

Щоб отримати значення кольорів у RGB форматі в R використовується функція `col2rgb()`. Наприклад `col2rgb("orange")` дає r=255, g=165, b=0. Нульове значення означає повну прозорість. Додаткова інформація `help(rgb)`.

6.1.12 Графіки умовних розподілів

Функція `cdplot()` дозволяє представити розподіл категорій (факторів), залежних від певних числових значень, у вигляді графіку умовного розподілу.

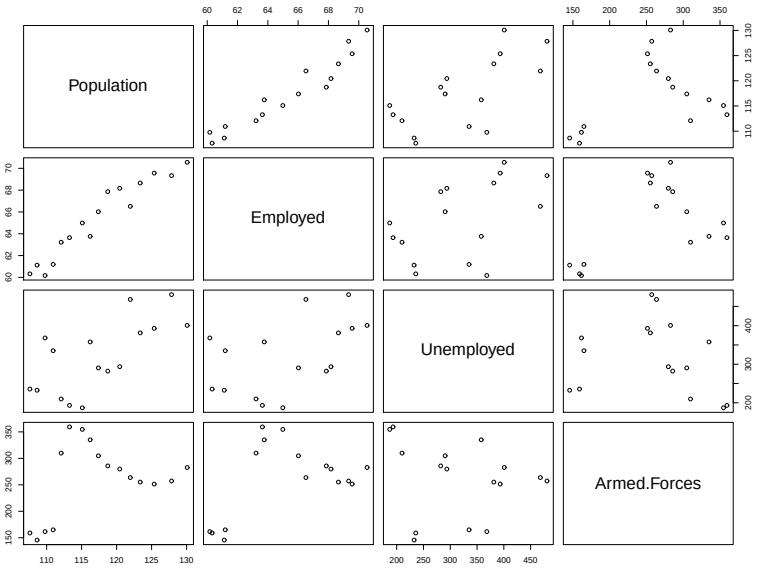

Рис. 6.16. Матриця діаграм розсіювання побудована на основі набору даних longley

```
cdplot(formula, ...)
```

Рис.6.18 демонструє графік умовного розподілу чоловіків і жінок за їхнем ростом

```
> sex <- factor(c(2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1,
  1, 1, 2, 1, 1, 1, 1, 1), levels = 1:2, labels = c("M", "W"))
> height <- c(153, 157, 158, 163, 186, 187, 170, 186, 190, 189,
  170, 180, 180, 180, 182, 183, 185, 175, 185, 186, 178, 179, 181)

> cdplot(sex ~ height,xlab=' height,cm')
```

6.1.13 3D графіки

Для побудови трьохвимірних графіків неперервних даних в базовому пакеті `graphics` передбачені функції `image()`, `contour()` і `persp()`:

- Функція `image()` генерує графік у вигляді прямокутників, кольори яких представляють значення змінної z.
- Функція `contour()` дозволяє отримати контурний графік, значення змінної z відображають контурні лінії.
- Функція `persp()` будує трьохвимірні графіки.

Функція `persp()` дозволяє представляти 3D графіки з різної перспективи. Синтаксис функції `persp()` має вигляд

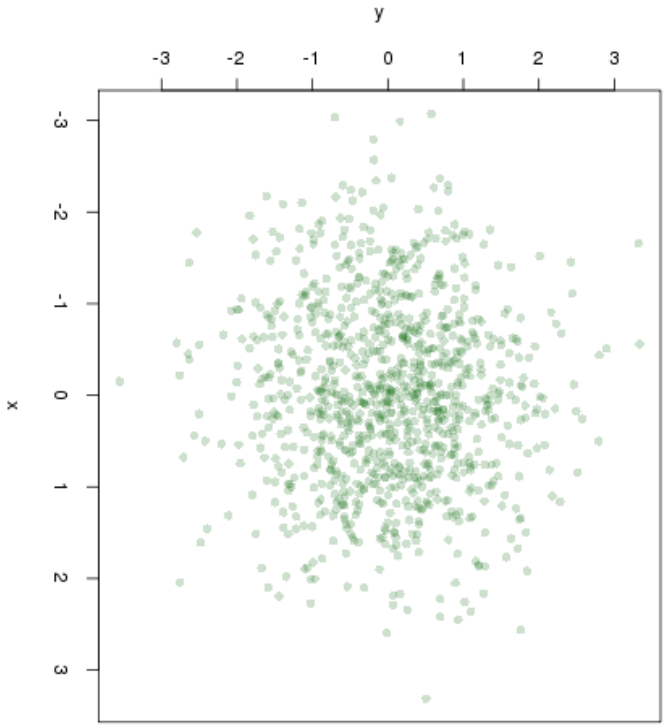

Рис. 6.17. Точковий графік високої щільності

```
persp(x,y,z,...,theta, phi,...)
```

де *x,y,z* - дані, *theta*, *phi* - кути, які задають перспективу (азимутальний і вертикальний напрямки,відповідно). Приклад використання функції `persp()`

```
x <- seq(-10, 10, length.out = 50)
y <- x
rotsinc <- function(x,y) {
    sinc <- function(x) { y <- sin(x)/x ; y }
    10 * sinc( sqrt(x^2+y^2) )
   }
z <- outer(x, y, rotsinc)
persp(x, y, z,theta=20, phi=50, shade = 0.3,col = "lightblue")
```

6.2 Збереження графіків у файл

Для запису графіків у файл використовуються наступні функції в залежності від необхідного формату графічного файлу:

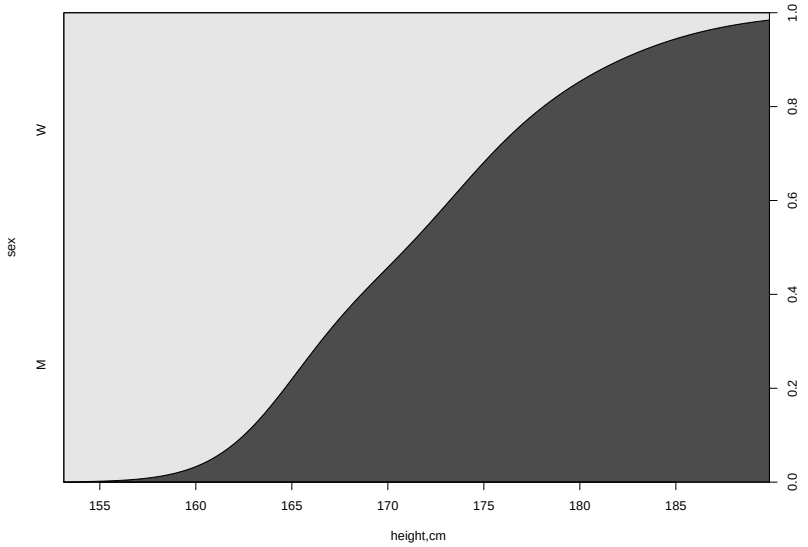

Рис. 6.18. Точковий графік умовного розподілу побудований за допомогою функції `cdplot()`

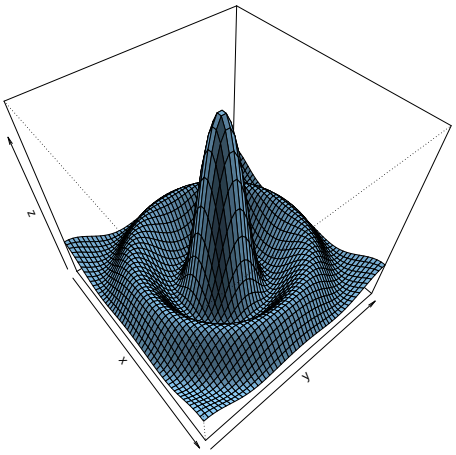

Рис. 6.19. 3D графік поверхні побудований за допомогою функції `persp()`

1. растровий графічний формат

```
bmp("myplot.bmp") - файл формату BMP
jpeg("myplot.jpg") - файл формату JPG
png("myplot.png")  - файл формату PNG
tiff("myplot.png") - файл формату TIFF
```

2. векторний графічний формат

```
postscript("myplot.ps") - файл postscript-ового формату
pdf("myplot.pdf") - файл формату PDF
svg("myplot.svg") - для Unix/Linux
CairoSVG("myplot.svg") - для Windows
win.metafile("myplot.wmf")
```

Приклад запису графіка в файл формату PostScript

```
> postscript("Testplot.ps")
> x<-seq(0,10*pi,by=pi/100)
> plot(x,sin(x))
> dev.off()
```

Функція `dev.off()` закриває режим запису графіки в файл і зберігає файл.

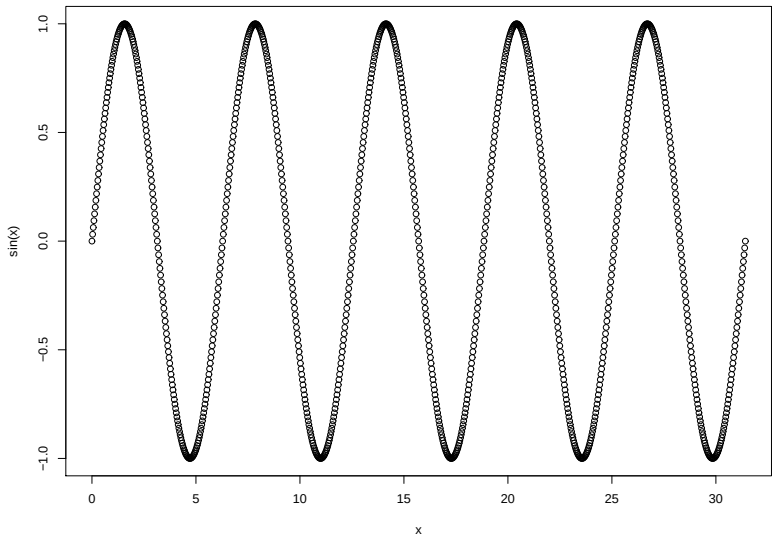

Рис. 6.20. Рисунок графіку збережений у вигляді PostScipt файлу за допомогою функції `postscript()`

За допомогою параметрів *width* і *height* можна встановити розмір рисунку, що зберігається в файл. Додаткова інформація про параметри див. `?postsript`, `?bmp`, `?svg`.

6.3 Графічні параметри

6.3.1 Глобальні і локальні установки

Щоб змінити атрибути графіка (колір, шрифт, символи, діапазон значень на осях, і тд) необхідно окреслити або змінити відповідні графічні параметри. В середовищі R графічні параметри являють собою абрівіатуру від їхніх повних назв англійською мовою. Повний список графічних параметрів і їхні поточні значення можна проглянути використовуючи функцію `par()`. Так само використовуючи функцію але вже з заданими значеннями `par(graphparameter1=value1, graphparameter2=value2,..)` можна змінити значення графічних параметрів. Встановлення параметрів за допомогою функції `par()` має глобальний характер, відповідно проявлятиметься на всіх графічних об'єктах R до наступного використання даної функції або виходу з середовища R.

```
> par()
> backup_par <- par()    # бекап поточних параметрів
> par(col="red") # встановлення червоного кольору
> x <- seq(0,6*pi,by=pi/10)
> plot(sin(x)) # побудова графіку з новими параметрами
> par(backup_par) # відновлення початкових параметрів
```

Однак на практиці частіше окреслюються графічні параметри конкретного графічного об'єкту, в якості додаткових аргументів відповідної графічної функції.

```
> x <- seq(0,6*pi,by=pi/10)
> plot(x,sin(x),type="b",col="darkgreen")
```

Додаткова інформація про графічні параметри та їхні можливі значення відповідних графічних функцій (`plot()`,`lines()`,`hist()`,..) доступна за допомогою функції `help()`

```
> help(plot)
> help(lines)
> help(hist)
> ....
```

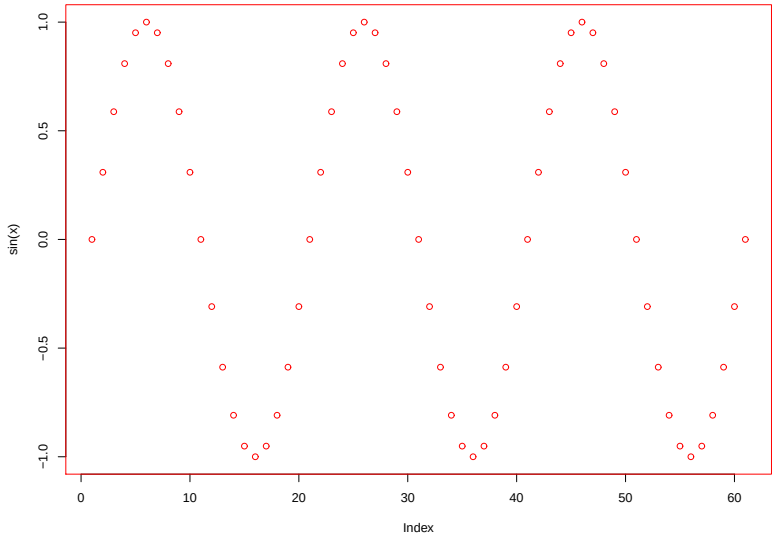

Рис. 6.21. Графік побудований за допомогою функції `plot()` з окресленим глобально параметром `par(col='red')`

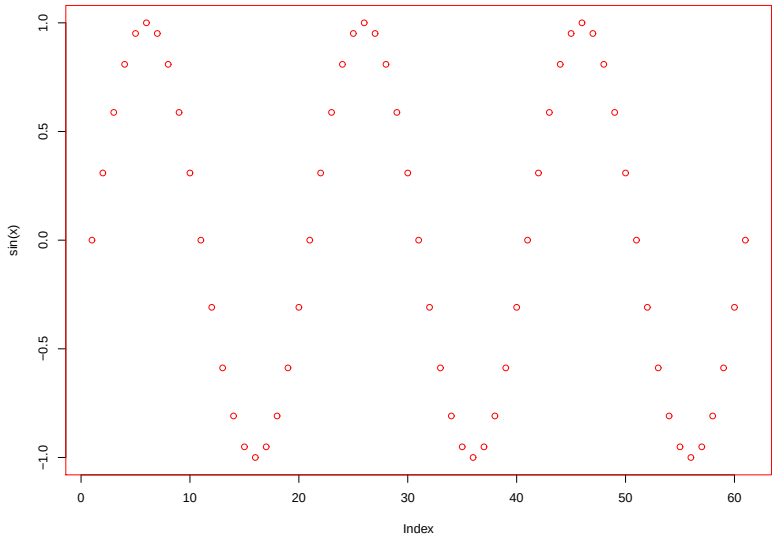

Рис. 6.22. Графік побудований за допомогою функції `plot()` з окресленим локально параметром `plot(...,col="darkgreen")`

6.3.2 Мультиграфіки

Використовуючи параметри *mfrow*, *mfcol* на одному рисунку можна одночасно представити кілька різних графіків у відповідному порядку. Наприклад

mfrow - встановлює порядок розміщення графіків рядками

```
> par(mfrow=c(3,2))
> plot(rnorm(100),type='l',col='red')
> plot(rnorm(100),type='l',col='green')
> plot(rnorm(100),type='l',col='blue')
> plot(rnorm(100),type='l',col='black')
> plot(rnorm(100),type='l',col='magenta')
```

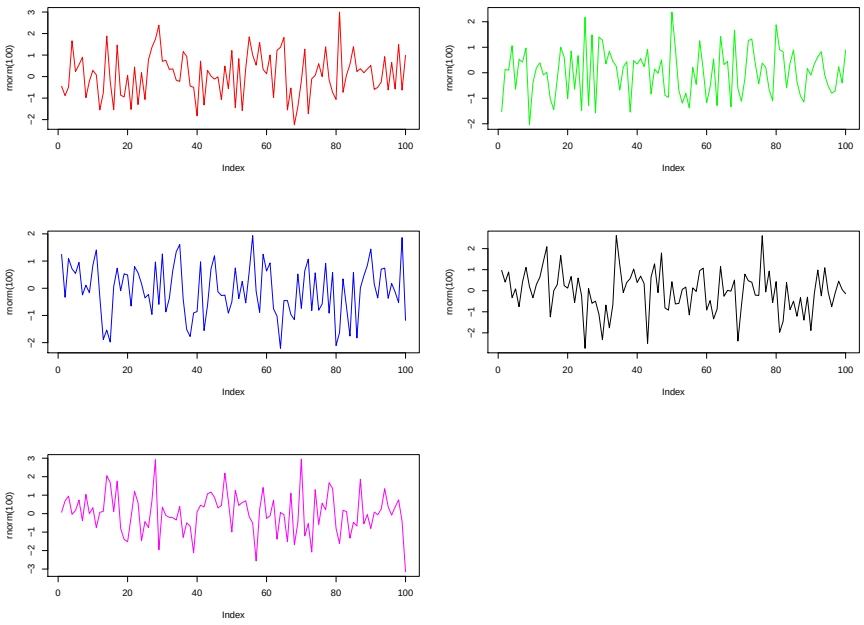

Рис. 6.23. Рисунок кількох графіків (мультиграфік) розташування яких окреслено за допомогою параметру `par(mfrow=c(3,2)`

mfcol - встановлює порядок розміщення графіків стовбчиками

```
> par(mfcol=c(3,2))
> plot(rnorm(100),col='red')
> plot(rnorm(100),col='green')
> plot(rnorm(100),col='blue')
> plot(rnorm(100),col='black')
```

```
> plot(rnorm(100),col='magenta')
```

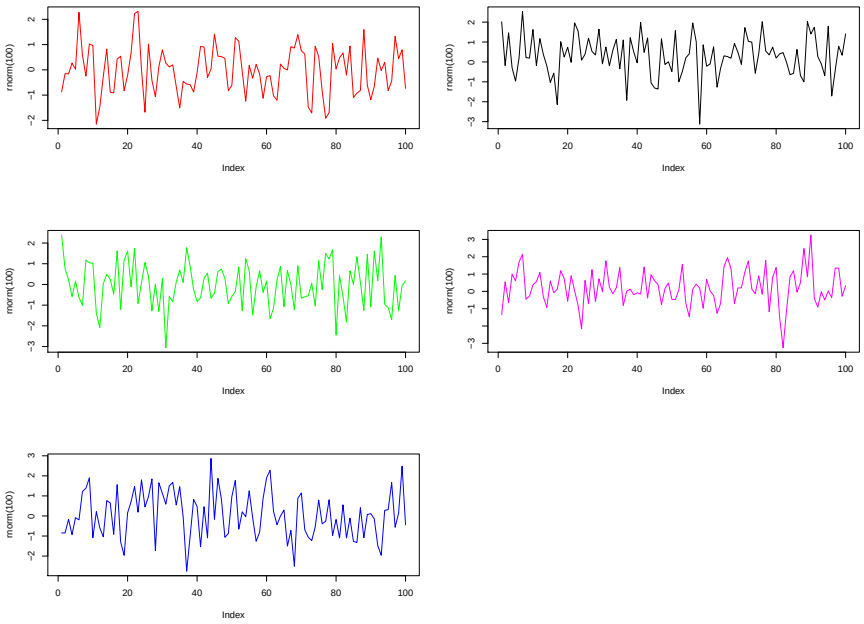

Рис. 6.24. Мультиграфік, розташування графіків на якому окреслено за допомогою параметру `par(mfcol=c(3,2))`

Більш гнучкішою в розташуванні графіків є функція `layout()`. Навідміну від параметрів *mfrow* і *mfcol* функція `layout()` дозволяє задати порядок і положення кожного окремого графіка в загальному ансамблі усіх графічних об'єктів рисунка.

```
> l = layout(
>  rbind(
>    c(1,1),
>    c(2,3)
>        )  )
> plot(rnorm(100),type='l',col='red')
> plot(rnorm(100),col='black')
> hist(rnorm(100))
```

6.3.3 Колір

Колір графіка встановлюється параметром *col* і може приймати значення у вигляді назви кольору, індексу, шістнадцяткового коду або значення за

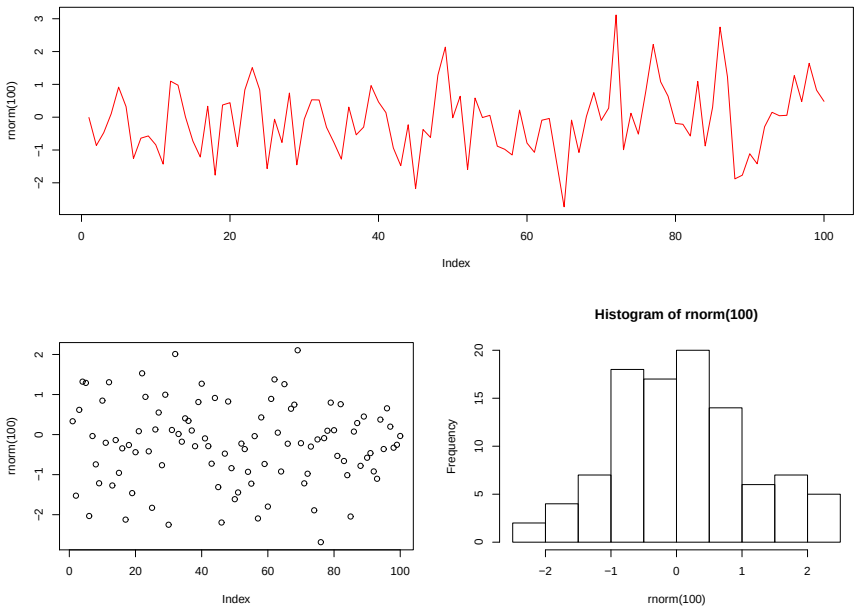

Рис. 6.25. Мультиграфік, розташування графіків на якому окреслено за допомогою функції `layout()`

схемою RGB. Наприклад *col='red'*, *col=552*, *col='#FF0000'*, *col=rgb(255,0,0,maxColorValue=255)* встановлює червоний колір графіка.

```
> hist(runif(100),col='#FF0000')
```

Існує ряд додаткових параметрів, які дозволяють встановити або змінити колір вибраного компоненту графіка:

```
col.axis  встановлює колір значень шкали
col.lab   встановлює колір назв осей графіка
col.main  встановлює колір назви графіка
col.sub   встановлює колір підпису до графіка, який задається
          параметром sub
bg   встановлює колір фону графіка plot background color
```

Функція `colors()` виводить список назв доступних кольорів.

6.3.4 Символи

Точки на графіку можуть бути представлені у вигляді певних символів. Параметр *pch* дозволяє встановити тип символу і приймає як числові так і символьні значення, рис. 6.28

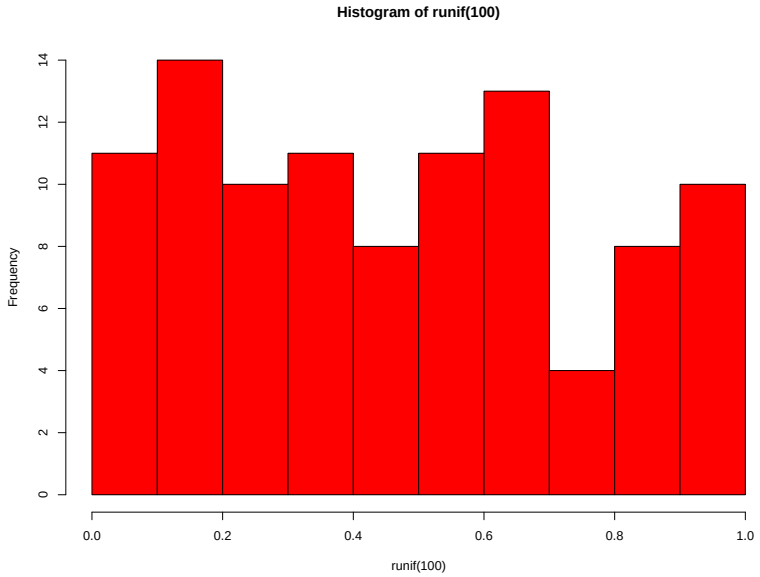

Рис. 6.26. Мультиграфік, розташування графіків на якому окреслено за допомогою функції `layout()`

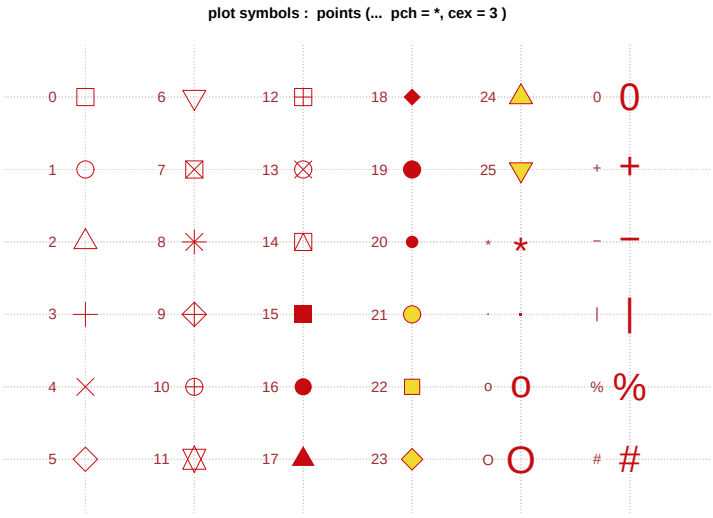

Рис. 6.27. Графічні символи, які задаються за допомогою параметра `pch`.

Розмір символів задається параметром *cex* (див. Розмір символів та атрибутів графіка)

```
> plot(rnorm(100),pch=8)
```

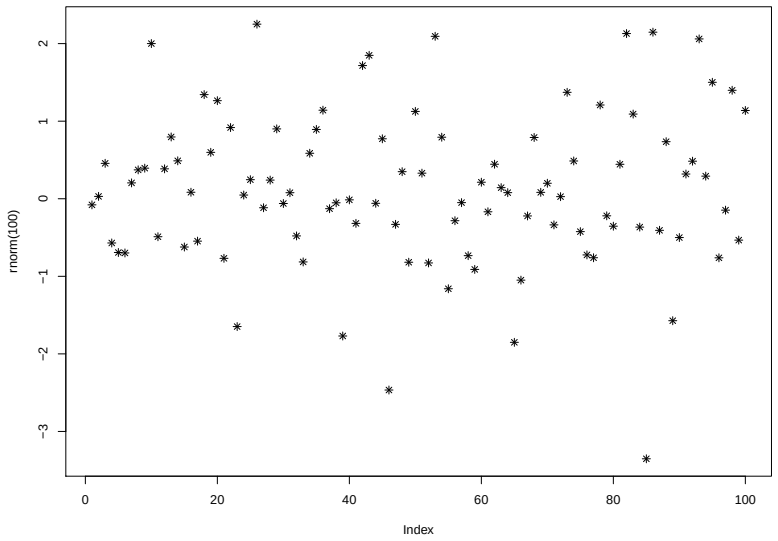

Рис. 6.28. Графік з символами окресленими за допомогою параметру `pch`

6.3.5 Лінії

Подібно до символів, середовище R дозволяє окреслити тип і ширину лінії за допомогою параметрів *lty* та *lwd*, відповідно. Параметр *lty* приймає значення від 0 до 6. Дефолтне значення *lwd*=1.

Ширина лінії масштабується відносно дефолтного значення параметра *lwd* (по дефолту *lwd*=1). Значення *lwd*=1.5 означає в півтора рази ширшу лінію.

6.3.6 Розмір символів, ліній та атрибутів графіка

Розмір символів, лінії на графіку окреслюється параметром *cex* і маштабується відносно дефолтного значення (по дефолту *cex*=1). Наприклад значення *cex*=2 означає в два рази більший розмір, *cex*=0.5 в два рази менший, відповідно.

Наявність похідних від *cex* додаткових параметрів дозволяє встановити розміри окремих компонент графіка

```
cex.axis  зміна розмірів значень на осях відносно параметру cex
cex.lab   зміна розмірів підписів x, y осей відносно параметру
          cex
cex.main  зміна розміру назви графіку відносно параметру cex
cex.sub   зміна розміру додаткового підпису до графіка відносно
          параметру cex
```

6.3.7 Назви і підписи

Назви осей і підписи до графіку можуть задаватися безпосередньо параметрами графічної функції

```
main - назва графіку
sub - підпис під графіком
xlab - назва oci x
ylab - назва oci y
```

Приклад

```
> x <- seq(0,6*pi,by=pi/10)
> plot(x,sin(x),type="b",col="darkgreen", main="Графік функції
> sin(x)",xlab="Значення на oci X", ylab="SIN(X)")
```

або з використанням функції `title()`

```
title(main="назва графіку", sub="підпис під графіком",
      xlab="назва oci x", ylab="назва oci y")
```

В першому випадку, щоб не показувати на графіку назви осей, які проставляються графічними функціями автоматично, використовуються аргументи *xlab=*, *ylab=* без заданих значень.

```
> x <- seq(0,6*pi,by=pi/10)
> plot(x,sin(x),type="b",col="darkgreen", main="",
> xlab="", ylab="")
> title(main="Функція SIN(X)", sub="додатковий підпіс",
> xlab="X", ylab="Y = SIN(X)")
```

В ситуації коли не потрібно виводити назви і підписи на графіках використовується глобальний параметр `ann='FALSE'`.

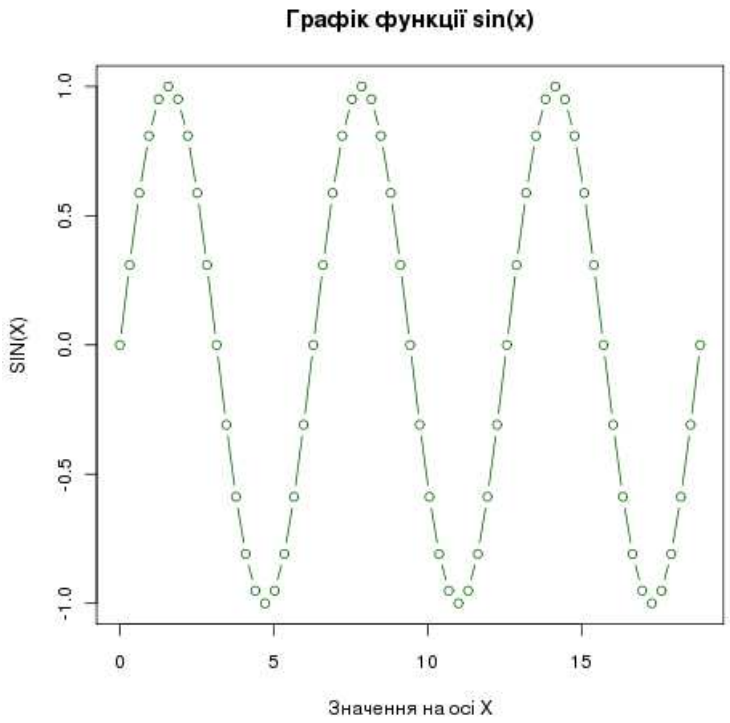

Рис. 6.29. Графік з окресленими атрибутами

6.3.8 Текст на графіку

Анотації на графіку розміщуються за допомогою функцій `text()` і `mtext()`, причому `text()` дозволяє розмістити текст безпосередньо на графіку, а функція `mtext()` на одному з його полів. Функції мають наступний синтаксис:

```
text(x,y,tekst, ...)
```

де *x*,*y* - координати на графіку за якими розміщується текст, *tekst* - сам текст, ... - додаткові аргументи.

```
mtext(tekst, side, line, ...)
```

де *tekst* - текст, що виводится, *side* - сторона графіку, на полях якої виводиться текст (1 = знизу, 2 = зліва, 3 = зверху, 4 = зправа), *line* - відстань від найближчого краю графіка до тексту (задається числом текстових рядків).

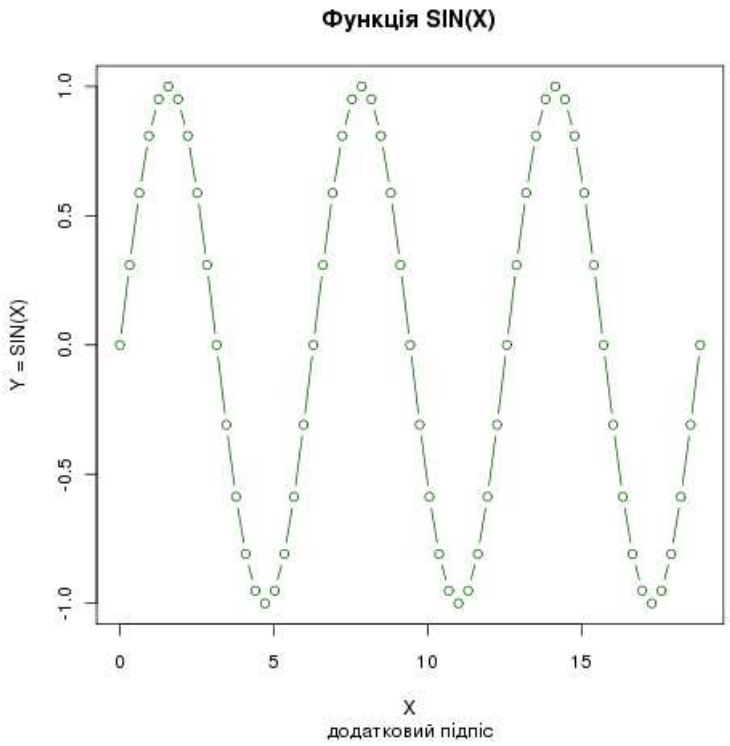

Рис. 6.30. Графік з атрибутами окресленими за допомогою функції `title()`

Варто зазначити, що серед додаткових аргументів є параметри, які дозволяють змінювати вигляд текстових анотацій. За допомогою параметрів *font*(1=plain, 2=bold, 3=italic, 4=bold italic, 5=symbol), *ps* , *family*("serif", "sans", "mono", "symbol") можна встановити гарнітуру, розмір і шрифт текстових аннотацій на графіку.

```
x<-seq(0,6*pi,by=pi/10)
plot(sin(x)) # побудова графіку з новими параметрами
text(2,5," ")
mtext("текст на полях", side = 4, line = 1)
```

Функція `text()` може використовуватися для встановленя міток на графіку.

```
x<-seq(0,6*pi,by=pi/10)
plot(sin(x)) # побудова графіку з новими параметрами
text(2,5,"max. значення")
text(2,5,"min. значення")
```

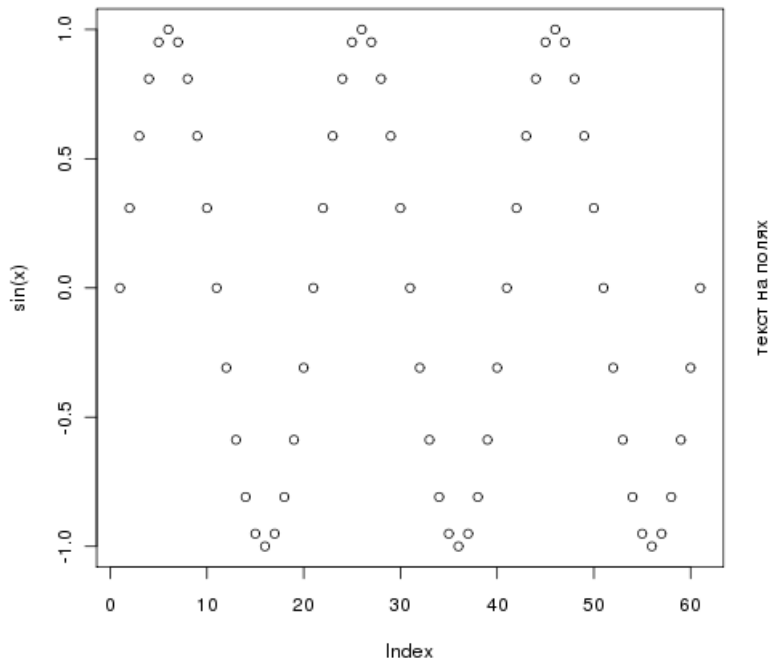

Рис. 6.31. Графік з заданим текстом на полях

Координати графіка можна отримати за допомогою функції `locator(1)`, відповіно дозволяє точніше встановити локалізацію міток на графіку.

6.3.9 Легенда

Додати легенду на графік можна використовуючи функцію `legend()`. Синтаксис функції має вигляд:

```
legend(position, description, title, bty, ...)
```

де *position* - область на графіку де розміщується легенда (приймає значення "topleft", "top", "left", "center", "right", "bottomleft", "bottom", "bottomright"), *description* - опис або вміст легенди, *title* - назва легенди, *bty* - встановлює/скасовує рамку навколо легенди, ... - додаткові параметри.

```
> x<-seq(0,6*pi,by=pi/10)
> plot(sin(x),type='l',col='green',ylim=c(-2,2)) # побудова
                                # графіку з новими параметрами
```

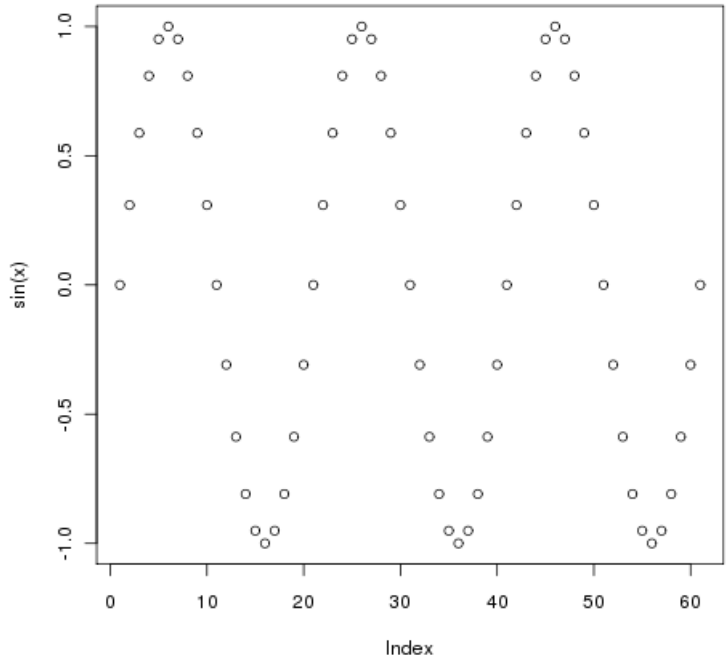

Рис. 6.32. Мітки на графіку задані з використанням функції `text()`

```
> lines(cos(x),type='l',col='red')
> lines(sin(x)+cos(x),type='l',col='blue')
> legend("topright",
  legend=c("sin(x)","cos(x)","sin(x)+cos(x)"),
  col=c("green","red","blue"),
  bty="n",
  lty=1)
```

Додаткова інформація `?legend` або `help(legend)`.

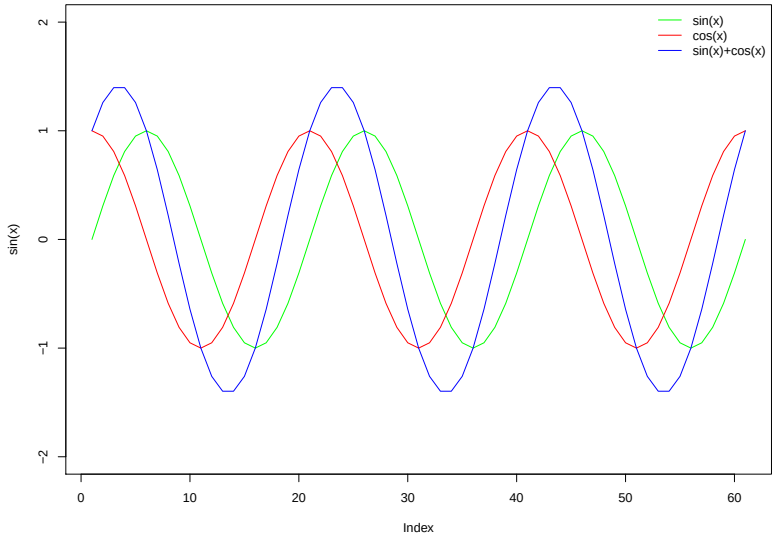

Рис. 6.33. Графік з вказаною легендою

7

Додаток А

7.1 Інсталяція R

Середовище R доступне для різних дистрибутивів Linux і Unix подібних систем, а також операційних систем типу Windows і MacOS.

У випадку Linux найпростішим способом інсталяції R є використання менеджера пакетів типу `yum`, `apt`. Для Red Hat або Fedora інсталяція з терміналу виглядає

```
$ sudo yum install R
```

Іншим варіантом інсталяції R є завантаження з дзеркала сайту CRAN відкомпільованого бінарного дистрибутиву R та встановлення за допомогою менеджера пакетів RPM

```
$ rpm -ivh  R-2.11.1.fc11.i586.rpm
```

Для решти Linux платформ дистрибутиви доступні на сайті `http://cran.r-project.org/bin/linux/`

У випадку операційної системи Windows вибирається одне з дзеркал сайту CRAN, завантажується інсталяційний **.exe* файл і встановлюється у вибрану директорію/папку опереційної системи Windows.

7.2 Запуск R

В Linux/Unix R запускається в вікні термінала, Рис.7.1

```
$ R
```

Використовуючи додаткові параметри R можна запустити з Tcl/Tk графічним інтерфейсом

```
$ R -g Tk
```

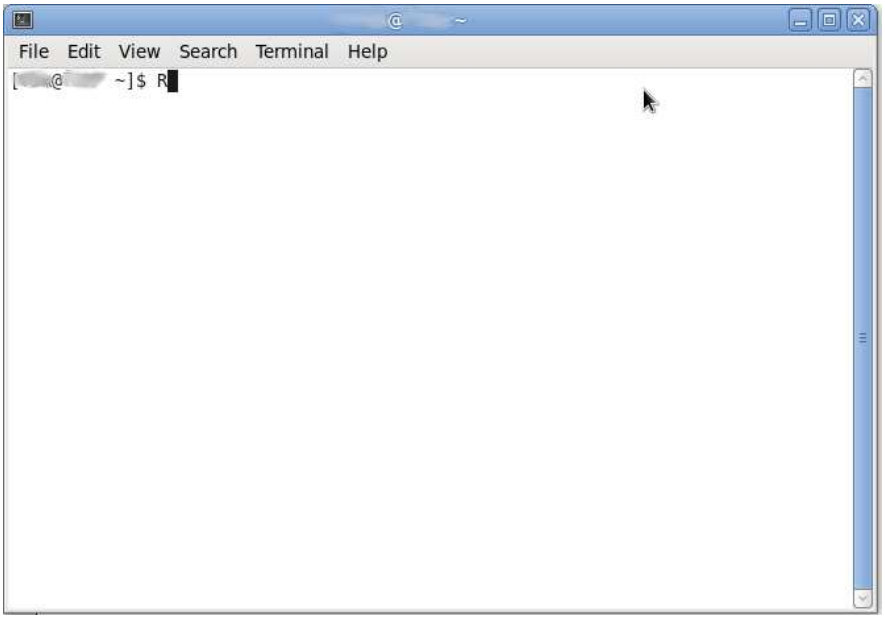

Рис. 7.1. Вікно термінала в системі Linux/Unix

В Windows запускаються на виконання файли R.exe (робота в терміналі Rterm) або Rgui.exe (робота з графічним інтерфейсом RGui). Існує ряд графічних редакторів та інтегрованих середовищ IDE, які дозволяють оптимізувати написання програм на R в залежності від індивідуальних потреб і уподобань[1].

7.3 Запуск скриптового файлу *.R

Команди або функції R записані в текстовому файлі у вигляді скрипту (програми на R) можна завантажити і запустити на виконання використовуючи функцію `source()`. Наприклад маємо текстовий файл *Scrypt.R* який містить функцію `print()`. В результаті запуску скрипту *Scrypt.R* отримаємо

```
> source("Scrypt.R")
[1] "Просто чудовий день!"
```

Встановлючи параметр *echo*=`TRUE` в консолі виводимуться команди/функції R перед результатом їхнього виконання

```
> source("Scrypt.R",echo=TRUE)

> print("Просто чудовий день!")
[1] "Просто чудовий день!"
```

[1] Див. перелік на сайті http://uk.wikipedia.org/wiki/R_(мова_програмування)

7.4 Запуск R у фоновому режимі

Програми написані в R і збережені у вигляді скриптів (файли розширенням **.R*) можна також запускати у фоновому режимі. Наприклад результат виконання програми або скрипту *my_script.R* скеровується в окремий файл *resultaty.out*

```
# в UNIX системах
R CMD BATCH [options] my_script.R resultaty.out

# в Windows
"C:\R-2.11.0\bin\R.exe" CMD BATCH
   --vanilla --slave "c:\my projects\my_script.R"
```

8

Додаток Б

8.1 Об'єкт formula

Формула `formula` відіграє важливу роль в статистичних розрахунках з використанням середовища R , окреслюючи модель згідно якої аналізуватимуться дані. Наведені нижче приклади відображають запис формули відповідо до аналітичного вигляду статистичної моделі

Для лінійної залежності з двома незалежними змінними x_1 і x_2 виду

$$y = b_0 + b_1 x_1 + b_2 x_2 + \epsilon$$

формула моделі відповідно має вигляд

```
y ~ x1+x2
```

Лінійна залежність включає в себе точку перетину з віссю y. Наступна формула дозволя виключити точку перетину з віссю ординат

```
y ~ 0+x
y ~ -1 + x
y ~ x - 1
```

Матиматичні оператори $\times$,`-`,`^`, і `:` в випадку використання їх у формулах лінійної регресії приймають зовсім інший зміст.

- Оператор `:`
  Формула

  ```
  y ~ x1:x2
  ```

  відповідає лінійній залежності виду

  $$y = b_0 + b_{12} x_1 x_2 + \epsilon$$

- Оператор $\times$
  Формула

```
y ~ x1*x2
```

еквівалентна запису

```
y ~ x1+x2+x1:x2
```

і відповідає лінійній залежності виду

$$y = b_0 + b_1 x_1 + b_2 x_2 + b_{12} x_1 x_2 + \epsilon$$

- Оператор `^`
  Формула

  ```
  y ~ (x1+x2)^2
  ```

  еквівалентна запису

  ```
  y ~ x1+x2+x1:x2
  ```

  і відповідає залежності виду

  $$y = b_0 + b_1 x_1 + b_2 x_2 + b_{12} x_1 x_2 + \epsilon$$

- Оператор `-`
  Використовується для усунення компонент з формули. Наприклад формула

  ```
  y ~ (x1+x2+x3)^2 - x1:x3
  ```

  еквівалентна запису

  ```
  y ~ x1 + x2 + x3 + x1:x2 + x2:x3
  ```

  і в аналітичній формі відповідає рівнянню

  $$y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_{12} x_1 x_2 + b_{23} x_2 x_3 + \epsilon$$

- Оператор I(...)
  дозволяє записувати компоненти формули з операторами `:`,×,`^`,`-` як звичайними арифметичними операторами. Наприклад формула

  ```
  y ~ x1+I(x2+x3)
  ```

  в аналітичній формі відповідає залежності

  $$y = b_0 + b_1 x_1 + b_2 (x_2 + x_3) + \epsilon$$

Література

1. K. Hornik, *The R FAQ*, `http://cran.r-project.org/doc/FAQ/R-FAQ.html`, 2009

2. W. N. Venables, D. M. Smith and the R Development Core Team, *An Introduction to R*, `http://cran.r-project.org/doc/manuals/R-intro.pdf`, 2009

3. J. Verzani, *Using R for Introductory Statistics*,CHAPMAN & HALLCRC,2005

4. W. N. Venables and B. D. Ripley,*Modern Applied Statistics with S* , Springer, 2002.

5. J. J. Faraway, *Practical Regression and Anova using R*, 2002

6. *R Journal*, `http://journal.r-project.org/`, 2001-2010

7. M.J. Crawley, `The R Book`, Wiley, 2007

Покажчик