Introduction to R

Dr. Willi Mutschler

Dr. Martina Danielova Zaharieva

Summer 2018

General information

Aims and prerequisites

- Objective: Learn how to use R for econometrics and statistics
- Prerequisites:
  1. Basics in probability theory and statistical inference (Mosler and Schmid, Wahrscheinlichkeitsrechnung und schließende Statistik, 2008)
  2. Multiple linear regression (von Auer, Ökonometrie, 2011)
- Laptop (any standard operating system), if possible with Wifi internet access

General information

Literature

Essential: Andreas Behr and Ulrich Pötter
Einführung in die Statistik mit R
2. Aufl., Vahlen: 2011.

Additional: W.N. Venables and B.D. Ripley
Modern Applied Statistics with S, 4th ed., 2002.

Grant V. Farnsworth, *Econometrics in R*, 2008 (pdf).

Phil Spector, *Data Manipulation with R*, 2008.

Paul Murrell, *R Graphics*, 2006.

General information

Schedule (I)

1. Introduction, installation, editor, help, packages
2. Operators and functions
3. Data import and export
4. Indexing
5. User-defined functions
6. Sorting and merging
7. Data description (univariate and multivariate)
8. Random numbers and simulations
9. Linear regressions
10. Numerical optimization
11. Maximum likelihood estimation
12. Advanced graphics

General information

Schedule (II)

Optional additional topics:

- date and time information
- time series
- wishes?
- ...

General information

Final Exam

- The exam consists of assignments you have to solve at home
- We will hand out the exam on Friday April 6 10 am
- Deadline to hand in your solutions: Friday April 13, 10 am
- Important: Do not forget to register with the Prüfungsamt for "Vorgezogene Prüfung"
- We will NOT communicate grades before the beginning of May (after it is not possible anymore to resign your official registration for the course)

Introduction

About R

- S is an object-oriented statistical computing language
- The language S is implemented as S-Plus (commercial) and R (OpenSource)
- The differences between S-Plus and R are minimal
- Similar programming languages: Matlab, GAUSS, Julia
- R is available for Windows, Linux and MacOS
- Internet site: `www.r-project.org`
- Comprehensive R Archive Network : `cran.at.r-project.org`

Introduction

Installation (for Windows)

- Open `www.r-project.org` in your browser
- In the left menu, choose CRAN (or click "download R")
- Choose a mirror (e.g. Germany)
- Choose you operating system (e.g. Windows)
- Choose `base`
- Download the newest version of R
- Execute `R-3.4.x-win.exe` and follow the instructions
- Start R

Command window (R Console) Prompt: > You can input commands and execute them
(by pressing the RETURN key)

Examples

```
> 1+1
> 1+1 # This is a comment:  1+1
> (1+2)*3
> (5/3)^4.5
> 5+2; 7+3; 2*5
```

Introduction

Command window

Concatenation: `c()` Assignment operator: `<-` or `=`

Examples

```
> c(1,4,7)
> a <- c(1,4,7)
> print(a)
> a
> A
> b <- c(1,a,3)
> b
> mean(b)
```

Quitting

- Quit R by the command `q()`
- In general, do *not* save your workspace

- Variable names and commands are case-sensitive
- Names may include letters, numbers, dots
- Use a consistent style (e.g Google's R style guide or at least the recommendations of the next slides, based on "Advanced R" by Hadley Wickham)

- File names should be meaningful and end in .R
- Variable and function names should be lowercase. Use an underscore to separate words within a name. Generally, variable names should be nouns and function names should be verbs. Strive for names that are concise and meaningful (this is not easy!).
- Avoid using names of existing functions and variables.
- Place spaces around all infix operators (=, +, -, <-, etc.).
- Always put a space after a comma, and never before (just like in regular English).

Introduction

Style Guide

- Do not place spaces around code in parentheses or square brackets (unless there's a comma).
- An opening curly brace should never go on its own line and should always be followed by a new line. A closing curly brace should always go on its own line, unless it's followed by else.
- Strive to limit your code to 80 characters per line.
- Use <-, not =, for assignment.
- Comment your code. Each line of a comment should begin with the comment symbol and a single space: #. Comments should explain the why, not the what.

Introduction

Editors

- Long computations should not be done interactively in the command window
- Use an editor to write a program and then execute it in R
- There is a built-in editor in R: Datei – Neues Skript
- External editors:
  - R-Studio, `http://www.rstudio.com/ide/download/`
  - Tinn-R, `http://sciviews.org/Tinn-R/`
  - Notepad++: `http://www.notepad-plus-plus.org/`

- Open a new script
- Type a few lines of commands, e.g.

```
a <- c(1,4,7)
mean(a)
mean(a)^2
```

- Execute a single line by pressing CTRL-R
- Execute multiple lines by marking them and then pressing CTRL-R or CTRL-RETURN
- Save the script, quit R, restart R, open the script and execute it

Introduction

Help etc.

- To obtain details about a command, type
  `?command`
  or
  `help(command)`
- Example: `?mean` or `help(mean)`
- Start the "help center": `help.start()`
- Task Views on CRAN
- R Journal on CRAN

- One of the strengths of R is the large and growing collection of packages that can be downloaded from CRAN (or installed off-line)
- Online installation: Pakete — Installiere Paket(e)...
- Offline installation: `install.packages("packagename")`
- Installed packages are activated by `library(packagename)`
- Help about packages: `library(help=packagename)`
- Install, activate and look into the help of the package MASS

Introduction

R objects

- R is object oriented
- An object can be anything: scalar, vector, matrix, string, table, factor, list, data frame, regression results, ...
- The object type determines how some commands work (e.g. `plot`, `summary`)
- Every object has a unique name
- List of all objects: `ls()`
- Delete (**rem**ove) objects: `rm()`

Examples

```
> x <- c("A","B","C")
> class(x)
> y <- c(1,2,5)
> class(y)
> ls()
> rm(x)
> ls()
```

Examples

```
> x <- c("A","B","C")
> class(x)
> y <- c(1,2,5)
> class(y)
> ls()
> rm(x)
> ls()
```

⟶ EXERCISE 1

Operators and functions

Logical operators

`&`	and
`|`	or
`!`	not
`NA`	no answer (or: not available)
`==`	equal (do *not* use =)
`>`, `>=`	greater than, greater than or equal
`<`, `<=`	less than, less than or equal
`!=`	not equal

Operators and functions

Logical operators

Examples

```
5 < 7
1+1 == 3
a <- c(-1,4,9)
a >= 2 & a < 8
b <- c(NA,1,2,3)
b > 0
is.na(b)
a == 4
a = 4
```

Operators and functions

Logical operators

Examples

```
5 < 7
1+1 == 3
a <- c(-1,4,9)
a >= 2 & a < 8
b <- c(NA,1,2,3)
b > 0
is.na(b)
a == 4
a = 4
```

$\longrightarrow$ Exercise 2

Operators and functions

Arithmetic operators and mathematical functions

`+`, `-`	plus, minus
`*`, `/`	multiplication and division
`^`	power (exponentiation)
`Inf`, `-Inf`	infinity (plus or minus)
`NaN`	not a number
`abs`	absolute value
`sqrt`	square root
`exp`,`log`	exponential function and natural logarithm (*not* `ln`)
`sin`	sinus (other trigonometric functions as well)
`sum`	sum

Operators and functions

Arithmetic operators and mathematical functions

Examples

```
x <- c(-1,0,2,9,3)
abs(x)
sqrt(x)
1/x
-1/x
0/x
log(x)
x^c(2,3,2,3,2)
x^c(2,3)
log(x)<0
```

Operators and functions

Arithmetic operators and mathematical functions

Examples

```
x <- c(-1,0,2,9,3)
abs(x)
sqrt(x)
1/x
-1/x
0/x
log(x)
x^c(2,3,2,3,2)
x^c(2,3)
log(x)<0
```

$\longrightarrow$ Exercise 3

Operators and functions

Matrix functions

`matrix`	creates a matrix from a vector
`dim`	dimensions of a matrix
`t`	transpose
`%*%`	matrix multiplication
`det`	determinant
`solve`	inverse
`eigen`	eigenvalues and eigenvectors
`diag`	diagonal
`cbind`	merge matrices column-wise
`rbind`	merge matrices row-wise

Examples

```
x <- matrix(c(2,3,4,5,1,1,9,3,2),3,3)
dim(x)
det(x)
solve(t(x)%*%x)
x%*%c(8,5,1)
diag(x)
diag(x) <- 0
solve(x)%*%x
matrix(NA,4,4)
rbind(cbind(x,x),c(0,1))
```

Examples

```
x <- matrix(c(2,3,4,5,1,1,9,3,2),3,3)
dim(x)
det(x)
solve(t(x)%*%x)
x%*%c(8,5,1)
diag(x)
diag(x) <- 0
solve(x)%*%x
matrix(NA,4,4)
rbind(cbind(x,x),c(0,1))
```

$\longrightarrow$ EXERCISE 4

Operators and functions

Set operations and special functions

`unique`	the set of all unique elements of a vector
`union`	$x \cup y$
`intersect`	$x \cap y$
`setdiff`	$x \backslash y$
`%in%`	$x \in y$
`sort`	sort the elements of a vector
`cumsum`	cumulated sum of a vector (also `cumprod`, `cummin`, `cummax`)
`which.min`	find the index of the smallest vector element (also `which.max`)

Operators and functions

Set operations and special functions

`unique`	the set of all unique elements of a vector
`union`	$x \cup y$
`intersect`	$x \cap y$
`setdiff`	$x \backslash y$
`%in%`	$x \in y$
`sort`	sort the elements of a vector
`cumsum`	cumulated sum of a vector (also `cumprod`, `cummin`, `cummax`)
`which.min`	find the index of the smallest vector element (also `which.max`)

$\longrightarrow$ EXERCISE 5

Operators and functions

Sequences and replications

`seq` sequence from a to b of length n,
`seq(from=a,to=b,length=n)`,
or by increments of size d,
`seq(from=a,to=b,by=d)`

`a:b` integer sequence from a to b

`rep` replicate a vector n times
`rep(what,times=n)`,
or each element n times,
`rep(what,each=n)`

Operators and functions

Sequences and replications

`seq` sequence from a to b of length n,
`seq(from=a,to=b,length=n)`,
or by increments of size d,
`seq(from=a,to=b,by=d)`

`a:b` integer sequence from a to b

`rep` replicate a vector n times
`rep(what,times=n)`,
or each element n times,
`rep(what,each=n)`

$\longrightarrow$ EXERCISE 6

Operators and functions

The apply command

- The `apply` command is very R specific and powerful, but it takes some time to get used to it
- The general syntax is

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

- The function `FUN` is applied to all rows (`MARGIN=1`) or all columns (`MARGIN=2`) of `X`
- All function returns are collected in a vector if they are univariat, or in a matrix if they are multivariat
- There are some variants of `apply` (`sapply`, `lapply`)

Operators and functions

The apply command

Examples

```
x <- matrix(c(2,3,4,5,1,1,9,3,2),3,3)
apply(x,1,sum)
apply(x,2,sum)
apply(x,2,quantile,prob=c(0.1,0.9))
apply(x,2,function(z) z^2)
apply(x,2,rep,each=2)
apply(x,1,rep,each=2)
```

Operators and functions

The apply command

Examples

```
x <- matrix(c(2,3,4,5,1,1,9,3,2),3,3)
apply(x,1,sum)
apply(x,2,sum)
apply(x,2,quantile,prob=c(0.1,0.9))
apply(x,2,function(z) z^2)
apply(x,2,rep,each=2)
apply(x,1,rep,each=2)
```

⟶ Exercise 7

Data import and export

General remarks

- R is all about working with data
- There are various ways to read data from different sources in many formats
- In R, datasets are usually represented as `data.frame` objects
- The structure of dataframes is similar to matrices
- Each column is a variable, each row is an observation
- R has a large collection of "standard datasets", see `data()`

Data import and export

Manual data input

- Very small datasets can be typed in directly, e.g.
  `x <- data.frame(v1=c(2,6,1,1),v2=c(9,9,8,8))`
- To edit existing objects, use `data.entry`, e.g.
  `y <- data.entry(x)`
- However, editing data within R is *not* recommended
- Datasets should be stored outside R, preferably in separate directories
- The datasets should be easily accessible by data-managing programs (e.g. Excel, Stata, ASCII editors, ...)

Data import and export

Saving and loading R objects

- All R objects can be saved by the command `save(obj1,obj2,...,file="c:/path/name.Rdata")`
- In principle, other file name extensions are possible, but not recommended
- All objects saved in a file can be loaded by the command `load("c:/path/name.Rdata")`
- The data format is R specific and cannot even be read by S-Plus

- A convenient command to read simple text files is `read.csv("c:/path/filename.txt")`
- The command assumes the following data format:
  1. The first row contains the variables names, delimited by commas
  2. The following rows are the observations, the variables are again delimited by commas
  3. The decimal sign is a dot (not a comma)
- Use `read.csv2` if the variables are delimited by semi-colons and the decimal sign is a comma (i.e. German style)

Data import and export

Reading and writing text files

- More options are available for the command `read.table`
- If the dataset is very large, reading a dataframe may be rather time consuming
- You can use the faster (but less convenient) command `scan`
- Exporting text files from R is usually not necessary. If it is, use `write.csv`, `write.csv2` or `write.table`

Data import and export

Reading and writing text files

- More options are available for the command `read.table`
- If the dataset is very large, reading a dataframe may be rather time consuming
- You can use the faster (but less convenient) command `scan`
- Exporting text files from R is usually not necessary. If it is, use `write.csv`, `write.csv2` or `write.table`

$\longrightarrow$ EXERCISE 8

Data import and export

Other data formats

- There are many packages that provide easy access to datasets in other data formats
- The `foreign` package includes commands to read the following formats: `dbf`, `Stata`, `SPSS`, `SAS`, and a few more (but not Excel)
- Excel sheets can be read using the command `read_excel` provided by the package `readxl`
- It is possible to access SQL data using the ODBC interface package `RODBC` or `dplyr`(we skip that)

Data import and export

Reading data online

- If the data file is located on a server one can simply specify the `url` instead of the file name
- Zipped files can be unzipped by `unzip`
- Some packages (e.g. `TTR` and `fImport`) provide downloading options for financial data

Examples

```
x1 <- read.csv("http://www..../bsp1.txt")
library(TTR) y <-getYahooData("IBM")
```

Data import and export

Reading data online

- If the data file is located on a server one can simply specify the `url` instead of the file name
- Zipped files can be unzipped by `unzip`
- Some packages (e.g. `TTR` and `fImport`) provide downloading options for financial data

Examples

```
x1 <- read.csv("http://www..../bsp1.txt")
library(TTR) y <-getYahooData("IBM")
```

$\longrightarrow$ EXERCISE 9

Indexing

Indexing vectors

- R has a rich indexing syntax
- The basic ideas are the same for vectors, matrices and other objects
- Indexing is used to read or manipulate specified elements of the objects
- Indexes are always given in square brackets: `[]` (or sometimes `[[]]`)
- Indexes can be either numerical or logical
- We will start with vectors and then look at matrices and dataframes
- The symbols `i` and `j` denote integer variables (not vectors)

Numerical indexing

`x[1]`	first element
`x[2]`	second element
`x[i]`	*i*-th element
`x[-i]`	all elements, without position *i*
`x[a:b]`	all elements from position *a* to position *b*
`x[k]`	`k` numerical vector: all elements at positions given in *k*

Logical indexing

`x[a]`	`a` logical vector: all elements where *a* is true (`a` must have the same length as `x`)

Indexing

Indexing vectors

Examples

```
x <- c(2,3,4,5,1,1,9,3,2)
x[2]
x[4:7]
x[20]
x[-9]
x[-3]
x[c(1,5,1,9,9)]
a <- (x<4)
x[a]
x[x<4]
```

Indexing

Indexing vectors

Examples

```
x <- c(2,3,4,5,1,1,9,3,2)
x[2]
x[4:7]
x[20]
x[-9]
x[-3]
x[c(1,5,1,9,9)]
a <- (x<4)
x[a]
x[x<4]
```

$\longrightarrow$ EXERCISE 10

Numerical indexing

`x[i,j]`	element in row i, column j
`x[,j]`	column j (as a vector)
`x[i,]`	row i (as a vector)
`x[,-j]`	without column j
`x[-i,]`	without row i
`x[a:b,j]`	elements a to b in column j
`x[k,m]`	`k`,`m` numerical vectors: all elements at positions given in k and m

Logical indexing Let `a` denote a logical matrix of the same dimension as `x`;
let `k` and `m` denote logical vectors of suitable length

`x[a]`	All elements of `x` at positions where `a` is true, as a *vector!*
`x[,m]`	All columns of `x` where `m` is true
`x[k,]`	All rows of `x` where `k` is true

Of course, one may use numerical indexing for one dimension and logical indexing for the other dimension

`x[k,1:2]`	All elements of columns 1 and 2 where `k` is true
`x[3,m]`	All elements of row 3 where `m` is true

Examples

```
x <- matrix(1:16,4,4)
x[3,3]
x[,4]
x[2,]
x[,-1]
x[-3,]
x[2:4,4]
x[c(1,4,2,2,2),1:2]
```

Examples

```
x <- matrix((-7:8)^2,4,4)
a <- (x<10)
x[a]
x[,c(TRUE,FALSE,TRUE,FALSE)]
x[x[,1]<30,3:4]
x[x[,2]==1 | x[,3]==1,]
x[2:4,4]
x[c(1,4,2,2,2),1:2]
```

Indexing

Indexing matrices

Examples

```
x <- matrix((-7:8)^2,4,4)
a <- (x<10)
x[a]
x[,c(TRUE,FALSE,TRUE,FALSE)]
x[x[,1]<30,3:4]
x[x[,2]==1 | x[,3]==1,]
x[2:4,4]
x[c(1,4,2,2,2),1:2]
```

$\longrightarrow$ Exercise 11

Indexing

Indexing dataframes

- Dataframes have the same index methods as matrices
- Logical conditions can include strings (character variables)
- There are three additional ways to extract dataframe columns:

  1. `x$varname`
  2. `x[[i]]`
     where `i` can also be a numerical vector
  3. `x["varname"]`
     or `x[c("varname1","varname2",...)]`

- Dataframe variables can be addressed directly by their name when you `attach` the dataframe, e.g. `attach(x)`

- Dataframes have the same index methods as matrices
- Logical conditions can include strings (character variables)
- There are three additional ways to extract dataframe columns:
  1. `x$varname`
  2. `x[[i]]`
     where `i` can also be a numerical vector
  3. `x["varname"]`
     or `x[c("varname1","varname2",...)]`
- Dataframe variables can be addressed directly by their name when you `attach` the dataframe, e.g. `attach(x)`

⟶ EXERCISE 12

Functions

User-defined functions

- One can define new functions in R
- Functions are objects of class `function`
- Each function has a name, one or more inputs (arguments) and one output (return)
- Inputs can be any objects (usually vectors)
- The function can return only one object (which can be a list)
- Variables defined within a function are only local

Functions

User-defined functions

Syntax

```
fn <- function(x,y){
  block of commands to compute output out
  return(out)
  }
```

Example

```
utility <- function(cons,gam){
  U <- (cons^(1-gam)-1)/(1-gam)
  return(U)
  }
```

Functions

User-defined functions

Syntax

```
fn <- function(x,y){
  block of commands to compute output out
  return(out)
  }
```

Example

```
utility <- function(cons,gam){
  U <- (cons^(1-gam)-1)/(1-gam)
  return(U)
  }
```

$\longrightarrow$ Exercise 13

Functions

More on function arguments

- Each arguments of a function must have a unique name
- When calling a function, the order of the arguments is arbitrary, if the argument names are explicitly used, e.g. `fn(x=5,y=9)` or `fn(y=9,x=5)`
- Without argument names, R assigns the values in the order of the function definition
- A function definition may include default values for arguments, e.g. `fn <- function(x,y=1){...}`
- If an argument with a default value is missing in a function call, R uses the default value

Sorting and merging

Sorting

- The `sort` command sorts (numeric or character) vectors
- By default, the elements are sorted ascendingly, but one can also sort descendingly.
- Matrices are sorted as vectors
- Dataframes cannot be sorted by `sort`
- The function `order(x)` returns a vector of the position of the smallest, the second smallest, . . . , the largest elements of `x`
- Hence, `x[order(x)]` returns the sorted vector
- The `order` command is useful for sorting matrices and dataframes!

Sorting and merging
Merging

- Two dataframes can be merged by common column names
- The command `merge(x,y,by=...)` merges two dataframes `x` and `y` by a common variable given in the `by`-option
- What happens if there are observations in `x` that are missing in `y` (or vice versa)?
- There are options to choose the way R deals with missings

Sorting and merging

Merging

- Two dataframes can be merged by common column names
- The command `merge(x,y,by=...)` merges two dataframes `x` and `y` by a common variable given in the `by`-option
- What happens if there are observations in `x` that are missing in `y` (or vice versa)?
- There are options to choose the way R deals with missings

$\longrightarrow$ EXERCISE 14

Data description

Frequency tables

- Let $x = (x_1, \ldots, x_n)'$ be a vector of (numerical or character) observations
- The command `table(x)` returns an object of the class "table", representing the frequency distribution of x
- The top row shows the values that occur (as a character vector)
- The bottom row shows the absolute frequencies
- The distribution can be plotted by `plot(table(x))` or `barplot(table(x))`

Data description

Frequency tables

- Let $x = (x_1, \ldots, x_n)'$ be a vector of (numerical or character) observations
- The command `table(x)` returns an object of the class "table", representing the frequency distribution of x
- The top row shows the values that occur (as a character vector)
- The bottom row shows the absolute frequencies
- The distribution can be plotted by `plot(table(x))` or `barplot(table(x))`

$\longrightarrow$ EXERCISE 15

Data description

Cumulative distribution function

- Let x be a vector of n observed values
- The cdf of x is

$$F(z) = \frac{1}{n} \sum_{i=1}^{n} 1\left(x_i \leq z\right)$$

- To calculate F at a single point z we simply count the number of observations $\leq z$
- The build-in function `ecdf` (**e**mpirical **cdf**) can be used to plot cdfs in a nice way, `plot(ecdf(x))`

Data description

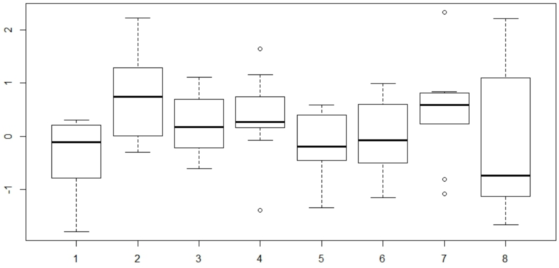
Quantiles

- Quantiles can be computed by `quantile(x,prob=...)`
- The argument `prob` can be a scalar or a vector of probabilities
- If `prob` is a vector the `quantile` function returns a vector
- Note that there are many definitions of quantiles
  (see the option `type` of `quantile`)
- For large datasets, the differences are negligible

Data description

Boxplots

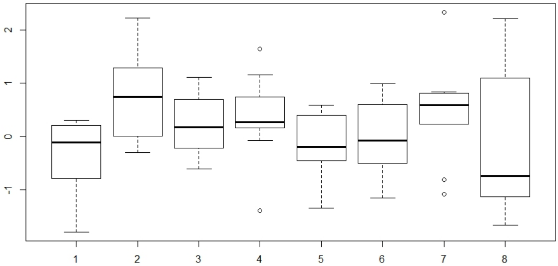
- If the argument of `boxplot` is a vector, one boxplot is generated
- If the argument is a matrix (or dataframe), one boxplot for each column is generated



Data description

Boxplots

- If the argument of `boxplot` is a vector, one boxplot is generated
- If the argument is a matrix (or dataframe), one boxplot for each column is generated



$\longrightarrow$ EXERCISE 16

Data description

Mean, variance, standard deviation

`mean` Calculates the mean of a vector, the mean of all elements of a matrix, each column mean of a dataframe

`sd` Calculates the standard deviation of a mean, or the standard deviation of each column of a matrix or dataframe

`var` Calculates the variance of a vector, or the covariance matrix of a matrix or dataframe

`na.rm` All three functions have the option `na.rm` (remove missings) which can be `TRUE` or `FALSE` (default)

Data description

Mean, variance, standard deviation

`mean` Calculates the mean of a vector, the mean of all elements of a matrix, each column mean of a dataframe

`sd` Calculates the standard deviation of a mean, or the standard deviation of each column of a matrix or dataframe

`var` Calculates the variance of a vector, or the covariance matrix of a matrix or dataframe

`na.rm` All three functions have the option `na.rm` (remove missings) which can be `TRUE` or `FALSE` (default)

$\longrightarrow$ Exercise 17

Data description

Histograms

- The built-in command `hist` generates a plot of the histogram
- An improved command in the `library(MASS)` is `truehist`
- See the help file of `truehist` for the options
- Important options are: `xlab,ylab,xlim,ylim,main`
- One can easily add lines and curves to the plot (with `abline` or `lines`)

Data description

Histograms

Examples

```
library(MASS)
x <- rnorm(2000) # we come back to this later
truehist(x)
abline(v=0)
g <- seq(-3,3,length=500)
lines(g,dnorm(g))
```

Examples

```
library(MASS)
x <- rnorm(2000) # we come back to this later
truehist(x)
abline(v=0)
g <- seq(-3,3,length=500)
lines(g,dnorm(g))
```

$\longrightarrow$ EXERCISE 18

Data description

Contingency table

- Contingency tables are multivariate frequency tables
- The command `table` can tabulate multivariate data
- If there are more than two dimensions, one can display the tables either as arrays or as flat tables
- One can use the `apply` command to compute marginal distributions

Data description

Contingency table

Examples

```
data(UCBAdmissions)
UCBAdmissions
ftable(UCBAdmissions)
ftable(UCBAdmissions,row.vars=c("Dept","Gender"))
plot(UCBAdmissions)
apply(UCBAdmissions,c(1,2),sum)
```

Data description

Covariance

- If there are two vectors x and y of the same length n, then `cov(x,y)` or `var(x,y)` compute the covariance

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

- If x is a (n, m)-matrix or dataframe, then `cov(x)` or `var(x)` compute the covariance matrix of its columns
- If missing values exist, one can specify which observations should be included (option `use`)

Data description

Correlation

- If there are two vectors x and y of the same length n, then `cor(x,y)` computes the correlation coefficient (Bravais-Pearson)
- If x is a (n, m)-matrix or dataframe, then `cor(x)` computes the correlation matrix of its columns
- If missing values exist, one can specify which observations should be included (option `use`)
- Use the option `method` to compute Spearman's or Kendall's correlation coefficients

Data description

Correlation

- If there are two vectors x and y of the same length n, then `cor(x,y)` computes the correlation coefficient (Bravais-Pearson)
- If x is a (n, m)-matrix or dataframe, then `cor(x)` computes the correlation matrix of its columns
- If missing values exist, one can specify which observations should be included (option `use`)
- Use the option `method` to compute Spearman's or Kendall's correlation coefficients

$\longrightarrow$ Exercise 19

Programming

Loops

- If the same commands should be executed for different values of some variable, loops are useful
- There are three kinds of loops: `for`, `while`, `repeat`
- By far the most important loop is the `for`-loop
- General syntax:

```
for([var] in vector) {
    [commands]
}
```

- The commands are executed for each value of `vector`

Example

```
z <- rep(NA,10)
for(i in 1:10) {
    z[i] <- i^2
}
print(z)
```

- Syntax of the `while`-loop:

```
while([condition]) {[commands]}
```

- Syntax of the `repeat`-loop:

```
repeat {[commands]}
```

- The `repeat`-loop does never stop but can be left using the command `break`

Syntax of the `if`-command

```
if([condition]) {
    [commands]
}
```

- The condition must not be a vector (else only its first element is used)
- If there is just a single command, the brackets can be omitted
- The opening curly bracket must appear in the same line as the `if`-command
- It is possible to add `else {[commands]}`

- The function `ifelse` can be used for multiple conditions
- Syntax of the `ifelse`-function

```
x <- ifelse([logical vector],a,b)
```

- Then `x` is a vector of the same length as `[logical vector]`
- The elements of `x` are taken either from `a` or from `b`
- The value vectors `a` and `b` can be scalars

- The function `ifelse` can be used for multiple conditions
- Syntax of the `ifelse`-function

```
x <- ifelse([logical vector],a,b)
```

- Then `x` is a vector of the same length as `[logical vector]`
- The elements of `x` are taken either from `a` or from `b`
- The value vectors `a` and `b` can be scalars

⟶ EXERCISE 20

Random numbers

Standard distributions

- A large number of standard distributions is implemented in R
- There is a common syntax for cdfs, density functions, quantile functions, and random number generation:

`pNAME(x,pars)`	cumulative distribution function at x
`dNAME(x,pars)`	density (or probability) function at x
`qNAME(p,pars)`	quantile function at p
`rNAME(n,pars)`	generate n random draws

- Here `NAME` is the abbreviated name of the distribution and `pars` are its parameters

Random numbers

Standard distributions

Some continuous distribution names:

`norm`	normal
`unif`	uniform
`lnorm`	log-normal
`exp`	exponential
`t`	t-distribution
`chisq`	χ^2-distribution
`F`	F-distribution

Random numbers

Standard distributions

Some discrete distribution names:

`binom`	binomial
`pois`	Poisson
`geom`	geometric
`hyper`	hyper-geometric
`nbinom`	negative binomial
`multinom`	multinomial

Random numbers

Standard distributions

- Define a vector x on an appropriate grid $[a, b]$
- Plots of cdf and density functions:

```
plot(x,pNAME(x,pars))
plot(x,dNAME(x,pars))
```

- Define a grid vector p on $[0, 1]$; plot of quantile function:

```
plot(x,qNAME(p,pars))
```

- Define a vector x on an appropriate grid $[a, b]$
- Plots of cdf and density functions:

```
plot(x,pNAME(x,pars))
plot(x,dNAME(x,pars))
```

- Define a grid vector p on $[0, 1]$; plot of quantile function:

```
plot(x,qNAME(p,pars))
```

$\longrightarrow$ EXERCISE 21

Random numbers

Simulations

- Simulation: Evaluate many randomly generated "scenarios" (replications)
- Standard steps:
  1. Choose the number of replications R
  2. Initialize an empty vector Z of length R
  3. Write a `for`-loop, e.g. over $r = 1, \dots, R$
  4. For each r, generate a random scenario, evaluate it, and save the result in $Z[r]$
  5. After the loop, report (or plot) the result vector Z

Example

Simulate the distribution of the moment estimator of the exponential distribution

```
R <- 10000
Z <- rep(NA,R)
for(r in 1:R) {
  x <- rexp(n=10,rate=0.5)
  Z[r] <- 1/mean(x)
}
truehist(Z)
abline(v=2,col="red")
```

Example

Simulate the distribution of the maximum of a Wiener process on the interval $[0, 1]$

```
R <- 10000
N <- 500
Z <- rep(NA,R)
for(r in 1:R) {
  x <- cumsum(rnorm(N,mean=0,sd=sqrt(1/N)))
  Z[r] <- max(x)
}
truehist(Z)
```

Example

Simulate the distribution of the maximum of a Wiener process on the interval $[0, 1]$

```
R <- 10000
N <- 500
Z <- rep(NA,R)
for(r in 1:R) {
  x <- cumsum(rnorm(N,mean=0,sd=sqrt(1/N)))
  Z[r] <- max(x)
}
truehist(Z)
```

$\longrightarrow$ EXERCISE 22

Linear regression

Multiple linear regression

- Consider the linear regression model

$$y_t = \alpha + \beta_1 x_{1t} + \ldots + \beta_K x_{Kt} + u_t$$

for $t = 1, \ldots, T$ with independent $u_t \sim N(0, \sigma^2)$

- Matrix notation

$$y = X\beta + u, \qquad u \sim N(0, \sigma^2 I)$$

- OLS estimator

$$\hat{\beta} = (X'X)^{-1} X'y$$

Linear regression

Multiple linear regression

- The general syntax of regression models is rather idiosyncratic:

```
a <- lm(formula)
```

- Basic "formula" syntax

```
y ~ x1 + x2 + ... + xK
```

- Endogenous variable is on the left of `~`; exogenous variables are on the right of `~`, separated by `+`

Example

```
library(foreign)
x <- read.dta("wave2009.dta")
attach(x)
regr1 <- lm(satisfaction~age+netincome+children)
regr1
summary(regr1)
names(regr1)
```

- The `lm`-object is a list containing:
  1. The estimated coefficients $\hat{\beta}$
  2. The residuals $\hat{u}_t$
  3. The fitted values $\hat{y}_t$
  4. Some other things
- If `a` is an `lm`-object one can access its elements using `coefficients(a)`, `residuals(a)`, `fitted.values(a)`
- Alternatively, one can use the \$-operator: `a$coefficients`, `a$residuals`, `a$fitted.values`

Extensions (I):

- An intercept is added automatically but can be removed: `lm(y~x1+x2-1)`
- If the variables are organized in an unattached dataframe `x`, one can use the syntax: `lm(formula,data=x)`
- The formula may contain mathematical functions, e.g. `lm(log(y)~log(x1))`
- **Attention**: Squares, sums and differences are not allowed!
- Use the function `I()` for squares, sums and differences

Extensions (II):

- Syntax for interaction terms

```
a <- lm(y ~ x1 + x2 + x1:x2)
```

- Abbreviation: `a <- lm(y ~ x1*x2)`
- Weights can be added using the option `weights`
- One can select a subset of observations using the option `subset`

Extensions (III):

- The `lm`-object can be used to add a regression line to a plot:

```
regr <- lm(y~x)
plot(x,y)
abline(regr)
```

- The `lm`-object can be used for forecasting:

```
regr <- lm(y~x1+x2)
xn <- data.frame(x1=c(...),x2=c(...))
predict(regr,newdata=xn,se.fit=TRUE)
```

Extensions (IV):

- Heteroskedasticity consistent standard errors are not reported by default
- The package `sandwich` supplies functions for robust standard errors (the package `sandwich` is included in the package `AER`)
- The syntax for robust standard errors is

```
coeftest(regr,vcov=vcovHC)
```

Example

```
regr2 <- lm(satisfaction~age+netincome,data=x)
regr3 <- lm(satisfaction~age+I(age^2))
regr4 <- lm(satisfaction~log(netincome))
regr5 <- lm(satisfaction~gender*marital)
z <- gender=="Female"
regr6 <- lm(satisfaction~log(netincome),subset=z)
coeftest(regr6,vcovHC)
```

Linear regression

Multiple linear regression

Example

```
regr2 <- lm(satisfaction~age+netincome,data=x)
regr3 <- lm(satisfaction~age+I(age^2))
regr4 <- lm(satisfaction~log(netincome))
regr5 <- lm(satisfaction~gender*marital)
z <- gender=="Female"
regr6 <- lm(satisfaction~log(netincome),subset=z)
coeftest(regr6,vcovHC)
```

$\longrightarrow$ Exercise 23

Numerical optimisation

Univariate optimisation

- In R, optimisation is minimisation
- To maximise a function $f(x)$, minimise $-f(x)$
- Suppose $f(x)$ has a minimum in the interval $[a, b]$
- The minimum is at x^* with $f(x^*) \leq f(x)$ for all $x \in [a, b]$
- The univariate optimisation command is

```
a <- optimize(f,interval=c(a,b))
```

- The command returns a list with two components: `a$minimum` (x^*), `a$objective` ($f(x^*)$)

Example

```
f1 <- function(x) {
  a <- 1/exp(-(x-3)^2)
  return(a)
}
m1 <- optimize(f1,interval=c(-5,5))
print(m1)
m2 <- optimize(f1,interval=c(-2,2))
print(m2)
```

- Consider a scalar-valued function $f(x_1, \ldots, x_n)$
- The multivariate optimisation command is

```
a <- optim(x0,f)
```

- The algorithm starts at $x_0 = (x_{01}, \ldots, x_{0n})$ and searches the minimum iteratively
- The more dimensions n, the harder it is to find the optimum
- Attention: The optimum might be a local optimum!

Numerical optimisation

Multivariate optimisation

Remarks:

- The function `f` can have more arguments than `x`, e.g. `f(x,a)`
- Additional arguments can be supplied via the `optim` command, e.g. `optim(x0,f,a=...)`
- Additional options can also be set, see `?optim`
- The numerical optimisation method can be changed by the `method`-option
- The `L-BFGS-B` method allows to add limits (to prevent the algorithm from wandering into “forbidden areas”)

Example

```
fn <- function(x) {
  a <- x[1]^2+(x[2]-2)^2
  return(a)
}
m1 <- optim(c(0,0),fn)
print(m1)
m2 <- optim(c(0,0),fn,
  method="L-BFGS-B",lower=c(0,0),upper=c(0.5,1))
print(m2)
```

Numerical optimisation

Multivariate optimisation

Example

```
fn <- function(x) {
  a <- x[1]^2+(x[2]-2)^2
  return(a)
}
m1 <- optim(c(0,0),fn)
print(m1)
m2 <- optim(c(0,0),fn,
  method="L-BFGS-B",lower=c(0,0),upper=c(0.5,1))
print(m2)
```

$\longrightarrow$ Exercise 24

Maximum likelihood

Basic idea

- The basic idea is very natural:
- Choose the parameters such that the probability (likelihood) of the observations $x_1, \ldots, x_n$ as a function of the unknown parameters $\theta_1, \ldots, \theta_r$ is maximized
- Likelihood function

$$L(\theta; x_1, \ldots, x_n) = \begin{cases} P(X_1 = x_1, \ldots, X_n = x_n; \theta) \\ f_{X_1,\ldots,X_n}(x_1, \ldots, x_n; \theta) \end{cases}$$

Maximum likelihood

Basic idea

- For simple random samples

$$L(\theta; x_1, \ldots, x_n) = \prod_{i=1}^{n} f_X(x_i; \theta)$$

- Log-likelihood function

$$\ln L(\theta; x_1, \ldots, x_n) = \sum_{i=1}^{n} \ln f_X(X_i; \theta)$$

- Maximize the log-likelihood function $\longrightarrow \hat{\theta}$

Maximum likelihood

Basic idea

- Sometimes, one can find $\hat{\theta}$ analytically by solving

$$\begin{aligned} \partial \ln L/\partial\theta_1 &= 0 \\ &\vdots \\ \partial \ln L/\partial\theta_r &= 0 \end{aligned}$$

- If the log-likelihood is not differentiable other maximization methods must be used, e.g. numerical maximization of the log-likelihood

Maximum likelihood

Properties of ML estimators

1. Equivariance: If $\hat{\theta}$ is the ML estimator for θ, then $g(\hat{\theta})$ is the ML estimator for $g(\theta)$
2. Consistency: plim $\hat{\theta}_n = \theta$
3. Asymptotic normality
4. Asymptotic efficiency
5. Computability (analytical or numerical); the covariance matrix of the estimator is a by-product of the numerical method

Example

Numerical estimation of the parameters of $N(\mu, \sigma^2)$

```
negloglik <- function(theta,x) {
  mu <- theta[1]
  sigma <- theta[2]
  return(-sum(log(dnorm(x,mu,sigma))))
}
dat <- rnorm(n=50,mean=5,sd=3) # generate a sample
ML <- optim(c(0,1),negloglik,x=dat)
print(ML)
```

Example

Numerical estimation of the parameters of $N(\mu, \sigma^2)$

```
negloglik <- function(theta,x) {
  mu <- theta[1]
  sigma <- theta[2]
  return(-sum(log(dnorm(x,mu,sigma))))
}
dat <- rnorm(n=50,mean=5,sd=3) # generate a sample
ML <- optim(c(0,1),negloglik,x=dat)
print(ML)
```

$\longrightarrow$ Exercise 25

Time series

Packages

Useful packages for time series analysis include:

- `tseries`
- `fGarch` (also installs `timeDate`, `timeSeries`, `fBasics`)
- `vars`
- `zoo`

See Task View “TimeSeries” on the CRAN site for many more time series packages.

Time series

Date and time classes

- The class `Date` represents calendar dates (days)
- Generate a single date,

```
as.Date("YYYY-MM-DD")
```

- Generate a vector of dates,

```
seq(date1,date2,by="unit")
```

where `unit` can be `days`, `weeks`, `months`, `years` (or even `4 days`, `2 months` etc.)

- The class `POSIXt` (`POSIXct`, `POSIXlt`) represents calendar dates and times
- Generate a single date,

  ```
  strptime("datestring","format")
  ```

  where `datestring` describes the date and time, and `format` explains the format
- Example:
  `strptime("2011/12/31-23:59:59","%Y/%m/%d-%H:%M:%S")`

Time series

Date and time classes

- Generate a vector of dates,

  ```
  seq(date1,date2,by="units")
  ```

  where `units` can be `years`, `months`, `weeks`, `days`, `hours`, `mins`, `secs` (also `20 mins`, etc.)
- Date classes can be converted by `as.Date`, `as.POSIXlt`, etc.
- Useful functions for dates include `diff`, `difftime`, `"-"`, `weekdays`, `months`, `quarters`
- Logical operators, e.g. `date1>date2`, `date3==date4`

Examples

```
d1 <- as.Date("2012-01-01")
d2 <- as.Date("2012-01-31")
d2-d1
difftime(d2,d1,units="hours")
dd <- seq(d1,d2,by="days")
print(dd)
plot(dd,rnorm(31),type="o")
weekdays(dd)
dd > as.Date("2012-01-15")
```

Examples

```
f <- "%Y-%m-%d %H:%M:%S"
d1 <- strptime("2012-01-01 15:00:00",f)
d2 <- strptime("2012-01-02 16:00:00",f)
difftime(d2,d1,units="hours")
dd <- seq(d1,d2,by="30 mins")
print(dd)
plot(dd,rnorm(51),type="o")
weekdays(dd)
dd > strptime("2012-01-01 19:00:00",f)
```

Examples

```
f <- "%Y-%m-%d %H:%M:%S"
d1 <- strptime("2012-01-01 15:00:00",f)
d2 <- strptime("2012-01-02 16:00:00",f)
difftime(d2,d1,units="hours")
dd <- seq(d1,d2,by="30 mins")
print(dd)
plot(dd,rnorm(51),type="o")
weekdays(dd)
dd > strptime("2012-01-01 19:00:00",f)
```

$\longrightarrow$ Exercise 26

Time series

Time series classes

- The elementary time series class is `ts`
- A `ts`-object is either a vector (univariate time series) or a matrix (multivariate time series) plus attributes
  1. start
  2. end
  3. frequency
- The attributes of a time series object can be read by the function `tsp` (**t**ime **s**eries **p**roperties)
- Most time series functions also accept vectors (or matrices) as inputs

Examples

```
x <- cumsum(rnorm(30))
y <- ts(x,start=c(2002,2),frequency=4)
print(y)
plot(y)
lag(y)
lag(y,-1)
diff(y)
window(y,start=c(2003,1),end=c(2008,4))
```

- The class `ts` is not very flexible and mainly meant for quarterly or monthly data
- The class `zoo` is more flexible and allows irregular time series
- A zoo object is a vector or matrix plus index vector as an attribute giving time points (or periods)
- The index vector can be of (almost) any class
- Suitable classes are: `vector`, `Date`, `POSIXlt`

Examples

```
x <- cumsum(rnorm(30))
y <- zoo(x,order.by=1981:2010)
print(y)
print(index(y))
plot(y)
lines(lag(y,-1),col="red")
lines(rollmean(y,k=4),col="blue")
diff(y)
window(y,start=1985,end=2000)
```

Time series

Time series: ACF

- One of the most important statistics about a time series is the autocorrelation function

$$\hat{\rho}(s) = \frac{\sum_{t=s+1}^{T}\left(X_t - \bar{X}\right)\left(X_{t-s} - \bar{X}\right)}{\sum_{t=1}^{T}\left(X_t - \bar{X}\right)^2}$$

- The function `acf` computes (and plots) the autocorrelation function for $s = 0, 1, 2, \ldots,$lag.max
- It can also be used to compute autocovariance and partial autocorrelations

- One of the most important statistics about a time series is the autocorrelation function

$$\hat{\rho}(s) = \frac{\sum_{t=s+1}^{T} \left(X_t - \bar{X}\right)\left(X_{t-s} - \bar{X}\right)}{\sum_{t=1}^{T} \left(X_t - \bar{X}\right)^2}$$

- The function `acf` computes (and plots) the autocorrelation function for $s = 0, 1, 2, \ldots,$lag.max
- It can also be used to compute autocovariance and partial autocorrelations

$\longrightarrow$ EXERCISE 27

- $AR(p)$, autoregressive process of order p,

$$(X_t - \mu) = \alpha_1 (X_{t-1} - \mu) + \ldots + \alpha_p (X_{t-p} - \mu) + \varepsilon_t$$

  where ε_t is a white noise process with variance σ_ε^2
- $AR(p)$ models can be estimated by

```
ar(x,order.max=p,aic=F)
```

- The lag order p can be given or automatically determined by the Akaike information criterion (set `aic=TRUE`)

Time series

Model estimation: ARIMA(p,d,q)

- $ARIMA(p, d, q)$, autoregressive integrated moving average process of order p, d, q

$$\begin{aligned}\left(\Delta^d X_t - \mu\right) &= \alpha_1 \left(\Delta^d X_{t-1} - \mu\right) + \ldots + \alpha_p \left(\Delta^d X_{t-p} - \mu\right) \\ &\quad + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q}\end{aligned}$$

  where ε_t is a white noise process with variance σ_ε^2 and Δ^d is the difference operator taken d times.
- *ARIMA* models can be estimated by

```
arima(x,order=c(p,d,q))
```

Time series

Model estimation: GARCH(p,q)

- $GARCH(p,q)$, generalized autoregressive conditional heteroskedastic process of order p, q

$$
\begin{aligned}
X_t &= \sigma_t \cdot \varepsilon_t \\
\varepsilon_t &\sim N(0,1) \\
\sigma_t^2 &= \alpha_0 + \alpha_1 X_{t-1}^2 + \ldots + \alpha_p X_{t-p}^2 \\
&\quad + \beta_1 \sigma_{t-1}^2 + \ldots + \beta_q \sigma_{t-q}^2
\end{aligned}
$$

- Often, there is also a mean equation for X_t, e.g.

$$X_t = \mu + \gamma_1 Z_{1t} + \ldots + \gamma_K Z_{Kt} + \sigma_t \varepsilon_t$$

Time series

Model estimation: GARCH(p,q)

- *GARCH* models without mean equation can be estimated by

  `garch(x,order=c(p,q))`

  of the `tseries` package
- To estimate more complex *GARCH* models, use the command

  `garchFit`

  of the `fGarch` package, see `?garchFit`

- $VAR(p)$, multivariate vector autoregressive models of order p can be estimated by

  `VAR(x,p)`

  where `x` is a multivariate time series object (or a matrix) and `p` is a scalar
- Package `vars`
- The return object is of class `varest` and can be used for forecasting and impuls response functions

Time series

Unit root tests

- The standard unit root test is the augmented Dickey-Fuller test (ADF test)
- Test of $H_0 : \rho = 0$ in the regression

$$\Delta X_t = \alpha + \delta t + \rho X_{t-1} + \sum_{j=1}^{k} \phi_j \Delta X_{t-j} + \varepsilon_t$$

- Function `adf.test` in the `tseries` package
- Other tests are also implemented, e.g. Phillips-Perron (`pp.test`) and KPSS (`kpss.test`)

- The standard unit root test is the augmented Dickey-Fuller test (ADF test)
- Test of $H_0 : \rho = 0$ in the regression

$$\Delta X_t = \alpha + \delta t + \rho X_{t-1} + \sum_{j=1}^{k} \phi_j \Delta X_{t-j} + \varepsilon_t$$

- Function `adf.test` in the `tseries` package
- Other tests are also implemented, e.g. Phillips-Perron (`pp.test`) and KPSS (`kpss.test`)

$\longrightarrow$ EXERCISE 28