Introduction to R

Hans Wackernagel

(Adapted from slides by Peter Reichert)

Purpose of R

Software environment for

- compilation of data;
- statistical data analysis;
- graphical representation of data and analysis results;
- programming language $\longrightarrow$ extensible environment.

Why using R?

1. Very versatile and **extensible** environment.
2. **Large library of packages** that is increased by contributions from statisticians all around the world:
   - availability of a very large set of "classical" techniques
   - many newly developed methods from applied statistical research.
3. **Free software**; participants in a course can walk away with the software and use it wherever they like.
4. R is **the standard statistics software** not only in the academic statistics community, but it is also spreading within industry and among consultants.

Contents

The R Environment

Availability of R

- **Free software**, originates from the statistical computing language **S** developped in 1975-76 at Bell Laboratories.
- Binaries, source code, documentation, and packages for all important computing platforms can be downloaded from one of the **national mirrors** listed at:

`http://www.r-project.org`

Style of using R

1. Command-/script-oriented user interface:
   - provides flexibility for complex data analyses.
2. Easy writing of personal library of scripts or packages.
3. Supports inclusion of data compilation procedures in scripts: working with raw data and updating or correcting data sets.
4. Help utility extremely useful to check commands and arguments.
5. Options for entering/executing commands:
   1. R console and pasting commands from an editor,
   2. integrated graphical environment Rstudio,
   3. R batch mode: execution of commands provided in a file.

Variables, Assignments

Data is stored in variables.

- Variables are created / extended automatically when an assignment is made.
- Assignment operator: "<-" or "="

Examples

```
a <- 1
b <- 2+3
```

or, equivalently,

```
a = 1
b = 2+3
```

Data Types

Elementary Data Types

`numeric` (`integer`,`double`), `complex`, `logical`, `character`, `factor`

Composite Data Types and How to Access Elements

vector: `[ ]`
matrix: `[ , ]`
array: `[ , , , ...]`
list: `[[ ]], $`
data frame: `[[ ]], $, [ , ]`

- Access is possible by index or by name.
- Changing data types: `as.matrix`, `as.vector`, ...

Examples

```
a=c(3,4,5);b=diag(5)
c = list("toto",x=c(1:6),"tata","titi", "tutu")
a; b; c;
a[1]; b[2,3]; c[[4]]; c$x
```

Commands / Functions

Commands and functions are composed of:

- a name
- required arguments
- optional arguments
- a return value.

Arguments can be used by position or name.

Examples

```
matrix(0,3,2); matrix(nrow=3,ncol=2); help(matrix)
```

Getting Help

- Help function `help(topic)` or more simply: `?topic`
- Html help utility:
  access directly within **Rstudio**
  or type: `help.start()`
- **Contributed manuals** in many different languages
- **Task views** to identify packages for specific methodology or application domain.
- R home page: `http://www.r-project.org`
- **Mailing lists** (R-announce, R-packages, R-help), bug tracking, **FAQs**, **Newsletter**

Data Manipulation

Variables, operations, searching, selection, combination, aggregation, dates, directory, data import and export

Variables (1)

- Assignments create / extend variables.
- Commands for listing/handling variables: `ls`, `rm`, `print`, `edit`
- `NA` represents missing data

Example

```
a = 2; a; ls(); rm(list=ls())
```

Scalars

numeric: `a = 2; b = 3.78; c = 1.2 + 3`
logical: `d = TRUE; e = F; f = 5>3`
character: `g = "blue"; h = "C:/Documents and Settings"`
complex: `i = 1 + 1i; j = 1 + 5*1i`
factor: `k = factor(c("a","a","b","b","b","c"))`

Variables (2)

Vectors (1d arrays; elements of same type)

- specify elements: `c`, `numeric`, `character`, `logical`
- sequences: `:`, `seq`, `rep`
- get length: `length`
- access elements: `[ ]`
- naming elements: assignment by *name*, `names`

Examples

```
a = c(1,3,7.5);
names(a) = c("x","y","z"); names(a)
d = 0:10; e = seq(0,10,by=2); f = rep(0,5)
g = c(0:10,20); g[5] = 99; g
length(a); a[2]; a["y"]; a
```

Variables (3)

Matrices (2d arrays; elements of same type)

- specify elements: `matrix, diag`
- get dimension: `dim, nrow, ncol`
- access elements: `[ , ]`
- naming elements: `rownames, colnames, dimnames`

Examples

```
a = matrix(0,nrow=3,ncol=4); b = diag(rep(1,5))
c = matrix(1:16,nrow=4); c
d = matrix(1:16,nrow=4,byrow=TRUE); d
rownames(d) = 1:4; colnames(d) = c("A","B","C","D")
d; rownames(d); d[1,2]; d["1","B"]; d[1,]; d[1,][2]
```

Variables (4)

Lists (list of elements of different type)

- specify elements: `list`
- get length: `length`
- access elements: `[[ ]], $`
- naming elements: assignment by *name*, `names`

Examples

```
a = list(1,1:3,c("a","b","c"))
b = list(a=1,b=1:3,c=c("a","b","c"))
a; b; a[[2]]; b[[3]]; b[["c"]]; b$c
names(b); names(b) = c("A","B","C"); b
b[[3]][2]; b$C[2]
```

Variables (5)

Data Frames (list of vectors of same length)

- specify elements: `data.frame`
- get dimensions: `dim`, `length`, `nrow`, `ncol`
- access elements: `[[ ]]`, `$`, `[ , ]`
- naming elements: assignment by *name*, `names`, `rownames`, `colnames`

Examples

```
a = data.frame(a=rep(1,3),b=1:3,c=c("a","b","c"))
a; a$b; a[[2]]; a[["b"]]
a$b[2]; a[[2]][2]
names(a); names(a) = c("A","B","C"); a
```

Missing Values

- Missing values are represented by `NA` and `is.na`.
- Note that many functions have an argument `na.rm`.

Examples

```
b = c(1,2,NA,4,5);b; is.na(b);
mean(b); mean(b,na.rm=TRUE)
```

Data Selection

- Selection by specifying ranges of indices, vectors of row or column names, or arrays of logical variables.
- "`ifelse`" selects elements conditionally.

Examples

```
a = matrix(1:50,nrow=10,ncol=5)
colnames(a) = c("A","B","C","D","E")
a[,c("B","C")]
a[1:3,c("A","C")]
a[1:5,c(F,T,T,F,F)]
a[a[,"A"]>5,]; a[,-c(1:2)]
b = c(1,2,3,4,5); c = c(5,4,3,2,1);
ifelse(b>c,b,c)
```

Operations

Arithmetic and Numerical Operations

- element-wise operators: `+`, `-`, `*`, `/`, `^` , `%%`, `%/%`, `...`
- vector/matrix operators: `%*%`, `t`, `solve`, `...`
- logical operators: `<`, `>`, `<=`, `>=`, `==`, `!=`, `!`, `&`, `|`, `...`
- character operations: `paste`, `nchar`, `substr`, `strsplit`, `...`

Examples

```
a = 1:10; b = rep(5,10);
a > b; a==b; a!=b; a<=b; a < b | a==b
a = matrix(c(1,2,3,4),nrow=2); a %*% t(a)
a = paste("a","b",sep="."); substr(a,1,1);
strsplit(a,"\\.")
```

Searching

Searching and related Operations

- find elements without duplications: `unique, duplicated`
- `levels` (for factors)
- find indices of first elements in table: `match`
- find indices of elements in table: `grep`
- substitute: `gsub`

Examples

```
a = c("A","B","A","A","B");
match("B",a); grep("B",a)
gsub("B","xxx",a)
unique(a)
b = as.factor(a); levels(b)
```

Sorting

Sorting of Vectors and Getting Sort Permutations

- reverse order: `rev`
- sorting of vectors: `sort`
- get sort indices : `order`

Examples

```
a = c(1,5,3,7,9,8,2,6,4,10)
sort(a)
ind = order(a); a[ind]
```

Data Combination

Combining vectors, matrices, and data frames

- extending vectors: `c`
- combining columns: `cbind`
- combining rows: `rbind`

Examples

```
a = 1:10; b = rep(5,10); c = matrix(1:20,ncol=2)
d = c(a,b); e = rbind(c,c); f = cbind(c,a,b)
```

Data Aggregation

Aggregating data by applying a function

- apply function over rows or columns: `apply`
- aggregate data according to factors: `tapply`,`aggregate`

Examples

```
a = data.frame(category=c("A","A","B","B","A","B"),x=1:6,y=2:7,z=rep(2,6))
apply(a[,-1],2,sum)
tapply(a$x,a$category,mean)
aggregate(a[,-1],list(a[,1]),mean)
```

Flow Control

Flow control is similar to other programming languages

- conditional execution: `if, else`
- loops: `for, while, repeat`
- exit loop: `break`
- switch to next loop execution: `next`

Statements must be enclosed by braces, "{", "}", if there is more than one statement in a loop or if-clause.

Examples

```
for(i in 1:10) { print(i); print(sqrt(i)) }
```

Dates

Formatting and Using Dates

- transform string to date: `as.Date`
- transform date to string: `format`
- transform date to numeric: `as.numeric`

Examples

```
a = as.Date("31.01.2006",format="%d.%m.%Y") + 1
a; format(a,"%d.%m.%Y"); as.numeric(a)
```

Working Directory

Accessing and Changing the Working Directory

- Access by the menu bar: File -> Change dir...
  or by commands: getwd(), setwd(*dir*)
- Use "unix directory notation" with
  - "/": directory separator
  - ".": current directory
  - "..": parent directory
- `list.files(`*path*`,`*pattern*`,...)`
  list files in directory specified by "*path*"

Examples

```
setwd("C:/user/Rcourse/caseStudy")
setwd("../datasets")
```

Data Import

Read Data from Text Files

- read data frame: `read.table`
- read data in "comma separated value" format (export option in Excel): `read.csv, read.csv2`
- more general read command: `scan`
- read from R format: `load`

Examples

a = read.table("*Dataset.dat*",header=TRUE)
b = scan("*Dataset.dat*",what="list")
c = read.csv2("frenchExcelFile.csv",header=TRUE)
c = read.csv("angloExcelFile.csv",header=TRUE)
load(file="*Dataset.RData*")

Data Export

Write Data to Text Files

- write data frame: `write.table, write.csv, write.csv2`
- more general write command: `write`
- save in R format: `save`

Examples

```
write.table(a,"test.dat",header=T,row.names=F)
save(a,file="test.Rdata")
```

Graphics

Simple plots, combined plots, exploratory data analysis plots

Generic Plot Functions

Generic Plot Functions

- x-y Plots: `plot`
- histograms: `hist`
- box plots: `boxplot`
- scatterplot matrices: `pairs`

Examples

```
x = c(0,1,2,3,4,5,6,7,8,9,10)
y = c(3,5,2,6,4,2,7,4,3,3,4)
plot(x,y)
hist(y)
boxplot(y)
pairs(cbind(x,y))
```

Generic Plot Functions

Add Elements to Plots

- Add lines: `lines`
- Add points: `points`
- Add legend: `legend`
- Add axes: `axis`

Examples

```
x = c(0,1,2,3,4,5,6,7,8,9,10); y = c(3,5,2,6,4,2,7,4,3,3,4)
plot(x,y,pch=19)
lines(c(0,10),c(mean(y),mean(y)),lty="solid")
legend("topright",legend=c("data","mean"),
       lty=c(NA,"solid"),pch=c(19,NA) )
```

Plot Parameters

The parameter function `par` allows the user to access a large number of plot parameters

- colour: `col`
- line type: `lty`
- line width: `lwd`
- symbol: `pch`
- margins: `mar`
- multiple figures: `mfrow`
- etc.

Plot Parameters (continued)

Examples

```
x = c(0,1,2,3,4,5,6,7,8,9,10); y = c(3,5,2,6,4,2,7,4,3,3,4)
par.default = par(no.readonly=TRUE)   # store values
par(mfrow=c(2,2))   # 2 x 2 graphics page
plot(x,y)
plot(x,y,type="l")
points(x,y,
       col=ifelse(y>=5, "red", "black"),
       pch= ifelse(y<=2 | y>=5,19,1) )
hist(y)
boxplot(y)
par(par.default)   # set back to default values
```

Plot Devices

Redirecting Output to Files

- write **pdf** file: `pdf`
- write **PostScript** file: `postscript`
- write **jpeg** or **png** file: `jpeg, png`
- terminate redirection: `dev.off`

Examples

```
x = c(0,1,2,3,4,5,6,7,8,9,10); y = c(3,5,2,6,4,2,7,4,3,3,4)
pdf("test.pdf")
plot(x,y)
dev.off()
```

Statistical Techniques

Descriptive statistics, probability distributions, linear and nonlinear regression, etc.

Statistical Techniques

- There is a large number of available statistical techniques in the base package of R and even many more in the contributed packages.
- In general, the quality of the contributed packages is may be very high; but there is no quality control by the R development team.
- This section only gives a few examples of statistical techniques provided in R.

Statistical Techniques

Descriptive Statistics

- summary statistics: `summary`
- sample range: `range`
- sample mean: `mean`
- sample standard deviation: `sd`
- sample variance: `var`
- sample correlation matrix: `cor`
- sample quantiles: `quantile`

Examples

```
y = c(3,5,2,6,4,2,7,4,3,3,4)
summary(y); range(y); mean(y); sd(y); var(y)
quantile(y,seq(0,1,by=0.05))
```

Statistical Techniques

Univariate Probability Distributions

- normal: `norm`
- log-normal: `lnorm`
- beta: `beta`
- gamma: `gamma`
- Student's t: `t`
- uniform: `unif`
- etc.

These generic function names are combined with the prefix "d" for probability density, "p" for cumulative distribution function, "q" for quantile function, and "r" for random numbers.

Examples

```
x = rnorm(1000,0,1); hist(x,freq=FALSE)
lines(seq(-3,3,by=0.1),dnorm(seq(-3,3,by=0.1),0,1))
```

Statistical Techniques

Regression

- linear regression: `lm`
- nonlinear regression: `nls`
- generic functions on results: `summary`, `residuals`, `predict`,`coefficients`

Examples

```
x = c(0,1,2,3,4,5,6,7,8,9,10); y = c(3,5,2,6,4,2,7,4,3,3,4)
res.lm = lm(y ~ x); summary(res.lm)
residuals(res.lm)
predict(res.lm,interval="confidence")
coefficients(res.lm)
summary(res.lm)$coefficients
```

Packages
Finding, Installing and Using Packages

Packages

A large number of contributed packages are available for R.

- These are available at the R homepage: `http://www.r-project.org`.

Some examples:

- packages for multivariate data analysis and classification: `ade4`
- geostatistics: `geoR, gstat`
- exploring spatial data: GeoXp

Packages are installed with the command "**install.packages**" and loaded with the command "**library**".

Examples

```
install.packages() # then choose interactively
install.packages("geoR"); library(geoR)
install.packages(c("GeoXp","ade4"))
```

Programming in R

Writing functions, object-oriented programming

Functions

Most of the R commands are functions. the "function" command of R allows its users to define their own functions. These work analogously to the R commands.

Examples

```
square = function(x) { return(x*x) }
a = 1:10; square(a)
topower = function(x,y=2){
z = x^ y
return(z)}
topower(2,3); topower(y=3,x=2); topower(2)
```

Editing functions

You can create an empty function:

Example

```
myfun = function(){}
```

and then edit it:

Example

```
options(editor = "gedit")
myfun = edit(myfun)
```

Function .First

Standard options can be defined in the file **.First** which is read each time R is started.

Example

```
.First= function () { options(editor="emacs") }
```