Introduction and usefull hints for the R software

Maarten Jansen

What is R

- Statistical software and programming language
- Freely available (inluding source code)
- Started as a free re-implementation of the S-plus programming language
- Websites
  - **R-CRAN = Comprehensive R Archive Network**

    `http://cran.r-project.org/`
  - **The R homepage**

    `http://www.r-project.org/`
  - **Belgian mirror for installation** (KU Leuven)

    `http://www.freestatistics.org/cran/`

GUI and Rcmdr

- In **Windows**, R opens by default as a **Graphical User Interface** (the basic R-Gui)
- Further refined interaction with the user is possible through the **R-commander**, which is an R-package that needs to be installed
- Other GUI's for R include
  - RStudio
  - Rattle
  - Red-R
  - JGR
  - ...
- We first discuss the installation of R-packages

R-packages

- R-packages add advanced routines/software to basic R
- See

  `http://cran.r-project.org/web/packages/`

  for a list of packages
- Some of these packages come with documentation (i.e., an online book), which is listed on this page

  `http://cran.r-project.org/other-docs.html`
- Packages have to be **downloaded and installed** (once)
  - Using the **R-GUI**: Click the button packages, this starts a dialogue and lists all packages availaible
  - Alternatively, type

    ```
    install.packages("mypackage",dependencies = TRUE)
    ```

    Mind the quotes
  - Whatever method, you choose, while installing a package, R needs access to the internet to download it, so you must work on-line

- Packages have to be **loaded** in every R session again, using the command

  ```
  library(mypackage)
  ```

  No quotes this time
- Example: download, install and load the R-commander
  - Download and install (once):

    ```
    install.packages("Rcmdr",dependencies = TRUE)
    ```
  - Load (every session again):

    ```
    library(Rcmdr)
    ```

Browsing through folders and loading data (I)

- Loading statistical routines: packages
- Loading data:

  ```
  datatable = read.table(file.choose())
  ```

  opens a window to browse for a file; then creates a matrix from that file
- Alternatively:

  ```
  getwd()
  ```

  tells you the working directory.

  ```
  setwd("F:/Rfiles")
  ```

  sets the working directory to `e/Rfiles` (mind the quotes)
  Then load the file

  ```
  datatable = read.table("mesquite.txt")
  ```

Browsing through folders and loading data (II)

- Changing the working directory is also possible through the menu of the basic RGui:

  ```
  File > Change dir... >
  ```
- All at once:

  ```
  datatable = read.table("F:/Rfiles/mesquite.txt")
  ```
- In the Rcmdr:
  (1) Click Data, Import data (not: Load data set), from text file...;
  (2) Enter name for data set, tick off “Variable names in file”
  (3) Browse through your folders.

A text file as input data file

- An input file typically contains a table of values (see example of `mesquite.txt` on slide 8). Once successfully loaded, this table is accessible throughout the session.
- The columns of the table are given default names `"V1"`, `"V2"`, etc., unless the data file provides a header with the names as in the example of `catdataexemple.txt` (see slide 9) The names in the header are stored into the current session if the option header is activated:

  ```
  datatable = read.table("catdataexemple.txt",header=TRUE)
  ```
- In Rcmdr: proceed as on slide 6, but tick the “Variable names in file”
- Extract variables from tables:

  ```
  datatable$V1
  x5 = datatable$V1[5]
  ```

  Or use `attach` (see slide 13)

The file `mesquite.txt` looks like this:

```
2.50  2.3  1.70  1.40  723.0
5.20  4.0  3.00  2.50  4052.0
2.00  1.6  1.70  1.40  345.0
1.60  1.6  1.60  1.30  330.9
1.40  1.0  1.40  1.10  163.5
3.20  1.9  1.90  1.50  1160.0
1.90  1.8  1.10  .80  386.6
2.40  2.4  1.60  1.10  693.5
2.50  1.8  2.00  1.30  674.4
2.10  1.5  1.25  .85  217.5
2.40  2.2  2.00  1.50  771.3
2.40  1.7  1.30  1.20  341.7
1.90  1.2  1.45  1.15  125.7
2.70  2.5  2.20  1.50  462.5
1.30  1.1  .70 .70  64.5
2.90  2.7  1.90  1.90  850.6
2.10  1.0  1.80  1.50  226.0
4.10  3.8  2.00  1.50  1745.1
2.80  2.5  2.20  1.50  908.0
1.27  1.0  .92 .62  213.5
```

The file `catdataexemple.txt` looks like this:

```
category X
1 23
1 24
1 32
1 25
1 21
1 19
2 32
2 31
2 35
2 28
2 31
3 31
3 24
3 42
```

Creating data

make a **vector** by

`x = c(2,3,4)`

make the **matrix** $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ by

```
A = matrix(c(1,2,3,4,5,6,7,8,9),ncol=3,byrow=TRUE)
```

or

```
A = matrix(c(1,2,3,4,5,6,7,8,9),ncol=3,byrow=1)
```

Matrix-vector product:

```
A%*%x
```

Remarks

- Tables are not matrices
- Simple $\star$ sign is used for pointwise product. The philosophy behind this is that R is a statistical software, not a linear algebra package (like Matlab). Central concept in statistics are random variables. Multiplication of random variables (used in correlation, describing interactions) proceeds point by point on the samples.

Displaying data

Let `x` be a matrix of data in the work space, then

`x`

displays the whole matrix

`x[1,]`

displays the first row

`x[2,]`

displays the second row

`x[,3]`

displays the third column.

Displaying large vectors

The display could look like

```
  [1]  72  66  64  66  40  74  50   0  70  96  92  74  80  60  7
 [19]  30  70  88  84  90  80  94  70  76  66  82  92  75  76  5
 [37]  76  76  68  72  64  84  92 110  64  66  56  70  66   0  8
```

meaning: the first element is `72`, then the second is `66`, etc.
The 19th element is `30`, then the 20th is `70`, etc.
The 37th element is `76`, then the 38th is again `76`, etc.

Naming and renaming columns in tables

Suppose we have uploaded the data table into

```
mesquite = read.table("F:/Rfiles/mesquite.txt")
```

then

```
names(mesquite)
```

displays the names of the columns of table `mesquite`

```
names(mesquite)=c("x1","x2","x3","x4","y")
```

renames the columns

to give direct access to these variables `x1,x2,x3,x4,y` use

```
attach(mesquite)
```

allows to compute, for instance

```
mean(x1)
```

Regression and ANOVA

First the estimation
Perform linear fit `lm`:

```
mesquite = read.table("mesquite.txt")
names(mesquite)=c("x1","x2","x3","x4","y")
attach(mesquite)
fit1=lm(log(y)~log(x1)+log(x2)+log(x3)+log(x4))
fit1
```

In case there are several active data matrices in the session, with possible shared names for the variables, one might prefer not to use `attach`. In that case, we can do the analysis with

```
fit1 = lm(log(y)~log(x1)+log(x2)+log(x3)+log(x4),data=mesquite)
```

Then the inference
Anova must be performed on a fit, it cannot be performed on raw data.

```
anova(fit1)
```

Don't forget to include interactions

```
attach(mesquite)
fit1 = lm(log(y)~log(x1)+log(x2)+log(x3)+log(x4)+log(x1)*log(x2)
```

Remark

```
log(x1*x2)
```

would lead to `NA` (not-a-number, undefined), because $\log(x_1 x_2) = \log(x_1) + \log(x_2)$ (unidentifiable model)

Omitting the intercept

```
fit1 = lm(log(y)~log(x1)+log(x2)+log(x3)-1)
```

Categorical data analysis

```
datatable = read.table("catdataexemple.txt",header=TRUE)
```

the option `header=TRUE` preserves the names in the txt-file (see before)

```
fitC = lm(X~factor(category),data=datatable)
anova(fitC)
```

Setting up simulations

Generating random values...

```
mu = 10
stdev = 3
n_simul = 100
samplesize = 40
set.seed(0)
X = matrix(rnorm(n_simul*samplesize,mean=mu,sd=stdev),nrow = n_s
Xbar = rep(NA,n_simul)
```

The variable `Xbar` is initialised as a vector of values `NA` (Not-a-number). If further calculations fail in replacing this values, this is easily detected (taking zeros as initial value may hide failed calculations if the user thinks 0 is the outcome)

Next, we compute sample means for all rows of `X`, using a user-defined function

User defined functions

Now define a function for confidence intervals

```
confint = function(X,alfa=0.05,side="2"){
  Xbar = mean(X)
  S2 = mean(X^2)-Xbar^2
  n = length(X)
  CIwidth = ifelse(side=="L"|side=="R",
    sqrt(S2)*qt(1-alfa,n-1)/sqrt(n),sqrt(S2)*qt(1-alfa/2,n-1)/sq
  lowerbound = ifelse(side=="L"|side=="2",Xbar-CIwidth,-Inf)
  upperbound = ifelse(side=="R"|side=="2",Xbar+CIwidth,Inf)
  return(list(lowerbound,upperbound,Xbar,S2))
}
```

Note the use of default values (`alfa=0.05,side="2"`) in the definition of the function

Loading user defined functions from a file

- Make a file `confint.r` with the definition of the function as on slide 19
- The filename is free (does not have to be the same as the function name)
- One file may contain several definitions, or other routines
- Type `source("confint.r")` or `source("F:/Rfiles/confint.r")`

Using user defined functions

A user defined function can be use througout the current session, for instance in

```
X = rnorm(100,mean=mu,sd=stdev)
ci = confint(X)
ci = confint(X,alfa=0.01)
```

Non-default values can be passed in the right order or explicitly by using the variable name followed by an equal sign and a value.

For use on all rows of a matrix, use `apply`:

```
X = matrix(rnorm(200,mean=mu,sd=stdev),nrow=2)
ci = apply(X,1,confint)
```

Some graphical commands

- `plot(x1)`
  `plot(x1,type="l")` (for plots with polylines)

  `plot(y~x1,col="blue")`

  (plotting y as function of x_1)
- Two plots on the same window: suppose we want to plot two rows of the random matrix on page 18.

  ```
  plot(X[,1],type="l",col="red")
  lines(X[,2],col="blue")
  ```

  Define your own colors:

  ```
  lines(X[,3],col=rgb(0.23,0.656,0.28))
  ```
- For adding points:

  ```
  points(X[,3],col=rgb(0.23,0.656,0.28))
  ```

Exporting to eps

Export your graph to encapsulated postscript (eps). In R, you have to redo all the plot commands after initializing

```
postscript("example1.eps",horizontal = FALSE, onefile = FALSE)
```

and conclude by

```
dev.off()
```

The command `postscript` makes the current plot on screen `inactive`.
The command `dev.off()` closes the current active device (postscript, pdf or window on screen)

Exporting to eps or pdf

So, in order to make an eps of the plot on slide 22:

```
postscript("example1.eps",horizontal = FALSE, onefile = FALSE)
plot(X[,1],type="l",col="red")
lines(X[,2],col="blue")
lines(X[,3],col=rgb(0.23,0.656,0.28))
dev.off()
```

Export to pdf:

```
pdf("example1.pdf",horizontal = FALSE, onefile = FALSE)
plot(X[,1],type="l",col="red")
lines(X[,2],col="blue")
lines(X[,3],col=rgb(0.23,0.656,0.28))
dev.off()
```

Graphics for exploratory data analysis

- `boxplot(x1)`
  for a single boxplot; recall that single boxplots are of little practical use; try to compare several variables:
  `boxplot(x1,x2)`
  On a matrix: `boxplot(mesquite)` creates boxplots for all columns, and it labels each plot with the columns' names
  It is possible to apply a transformation to all elements of a matrix:
  `boxplot(log(mesquite))`
- Normal probability plot (= qq plot against theoretical normal distribution):
  `qqnorm(x1)`
- QQ-plot (two empirical distributions): `qqplot(x1,x2)`
- Histograms: `hist(x1)`
  (use `?hist` for info on parameters, such as number of bins etc.)

Getting help

Besides the numerous documentation on the internet, getting help in R proceeds by

```
help(<your command>)
```

or, equivalently,

```
?<your command>
```

Tab completion helps in finding options: type part of a command + Tab

Quitting R

```
quit()
```

or, equivalently,

```
q()
```

(Don't forget the brackets)

When using R, think about

- (brackets)
- ”quotes”
- Question marks ? for help
- Tab completion

Programming in R, compared to Matlab

- **A different basic philosophy**
  - Matlab : **procedural** programming language
  - R: **object** oriented programming language
- Central concept in Matlab programming is a **procedure** or **routine**: A Matlab routine calls other subroutines and functions from files
- Central concept in R is an **object**: Data matrices, data fits (estimations), inferences, packages, R functions are all objects
  - Objects must first be created or defined. Two ways:
    - From the command line
    - From a file: `read.table`, `library`, `source`
  - <u>Once</u> an object is created/defined/loaded/read, it can be used throughout the R session
    - Change properties (e.g.: names of variables in a data matrix)
    - Use a function for the creation of new variables