Tutorial (Windows Version)

This tutorial will give you some brief basic knowledge about *R*.

R is free software (copyright: GNU public license) and is available from `http://stat.ethz.ch/CRAN/`. At this URL you find a comprehensive **Documentation**, `Manual`, "An Introduction to R" (about 100 pages `pdf`) and a shorter introduction `Contributed`, "R for Beginners / R pour les dbutants" (31 pages, english/french).

For convenience we will use *R*'s internal editor during this short introduction. Please note that different editors allow syntax highlighting for *R*. One of those is Tinn-R for Microsoft Windows. Tinn-R is a small, free and simple replacement for the basic code editor provided by Rgui. Latest Tinn-R version and source is available on `http://sourceforge.net/projects/tinn-r/`.

Getting started with *R*

To start *R*: *Start / Programs / Statistics / R /R 2.12.1* (or maybe a different version).

Creating and deleting objects

Type in the *R*-window:
> x <- 2 <*RETURN*>
> x <*RETURN*>
Result: **[1] 2**

The assignment operator **<-** has created an object **x**. *R* is vector-oriented, so **x** is a vector with one element of value 2.

Next try (all commands have to be confirmed by <*RETURN*>; this is omitted from now on):
> y <- c(3,5) (c for combine)
> y
Result: **[1] 3 5**, a vector with two elements.

Warning: Do not use names of *R*-commands as object names, for example: **c**, **t**, **T**, **F**, **max**, ...

ls() shows all objects you have already generated. To remove **x**, use **rm(x)**.

R-demonstrations

Get a list of all demonstrations with **demo()**. For example, take a look at the graphics demo of *R*: **demo(graphics)**.

Working with an .R (script-)file

It is useful to type the commands into a text file of an editor instead of directly typing them into *R*. They can then be transferred to *R*. This procedure enables easy corrections of typing errors and a reasonable saving and reproduction of the work. We use the built-in editor.

Create a new folder *RFiles* in your home directory using the Windows Explorer,
File / New / Folder.

Open a new script file in the editor, *File / New Script*. Type **z <- c(8,13,21)** as first line and **2*z** as second line. Save the file as *tutorial.R* in the directory *RFiles*.

You can send an entire sequence of commands to *R* by marking the commands with the left mouse button and clicking on the third item at the top. Alternatively, you can also mark a region and type C-r (i.e., press and hold the control key and then press r).

Computing with vectors

Type **fib <- c(1,1,2,3,5,z)** as third line of *tutorial.R* (gives the first eight Fibonacci-numbers). Evaluate this line, and take a look at **fib**. Type **2*fib+1**, **fib*fib** and **log(fib)** as next three lines of *tutorial.R*. Mark all three lines with the left mouse button and type C-r. This evaluates all marked lines. Check the results. Do you understand them?

From now on you should write (almost) all *R*-commands into the *.R-file to evaluate them from there. At the end, you may save it.

If you open the file next time, mark all the code and type C-r torestore your whole work.

Now create the sequence 2, 4, 6 as object s: **s <- 2*(1:3)**. Take a look at **fib[3]**, **fib[4:7]**, **fib[s]**, **fib[c(3,5)]** and **fib[-c(3,5)]**.

Create a vector **x** with 8 elements, some of which are positive, some negative. Check **x > 0** and **fib[x > 0]**.

Matrices: creation and computation

Create two vectors **x <- 1:4** and **y <- 5:8** and the matrices **mat1 <- cbind(x,y)** and **mat2 <- rbind(x,y,x+y)** (cbind means column-bind, rbind means row-bind). Take a look at the whole matrices **mat1** and **mat2** and try **mat2[3,2]**, **mat2[2,]** und **mat2[,1]**.

Computation with matrices using +, * etc. follows the same rules as computation with vectors, namely elementwise.

For the matrix product, use %*%, e.g. **mat2 %*% mat1**.

Data Frames

A data frame is a generalized matrix. The main difference between data frames and matrices is that matrices need all elements to be of the same type (e.g. numeric, character), while data frames allow every column to have another type.

Reading and looking at datasets

ASCII-data is most easily read by **read.table**, which generates a data frame. **read.table** works also for datasets from the web. Try:

no2 <- read.table("http://stat.ethz.ch/Teaching/Datasets/no2Basel.dat", header=TRUE)

You may examine the created object directly by **no2**. Single variables are accessible by **no2[,"NO2"]**. You may take a look at the original file, in particular its first line, to understand why *R* knows the name of the variable. This can be done by calling the above URL from a web browser, e.g., Firefox or Mozilla. The parameter **header=TRUE** of **read.table** tells *R* that the variable names are in the first line. **no2** is still small enough, but in general it is useful to use **str** first, which displays the structure and type of an object, but not every single element: **str(no2)**. **summary(no2)** displays information

about the columns of **no2**. **summary** extracts the most important information from lots of *R*-objects, e.g., the results of statistical tests or regression fits.

An alternative to **read.table** is the command **scan**, which reads vectors and lists. A list is a more general structure which may contain elements of different types and sizes, e.g. vectors of varying lengths, data frames, sublists, etc.

Graphics
Draw a histogram of the NO2-values of the no2-data.
par(mfrow = c(1,2)) # Number of pictures one below the other [1] or side by side [2]
important to save paper!
hist(no2[,"NO2"]) # draw histogram.
Now compute the regression line of the NO2-content against temperature and show it graphically next to the histogram:
lm.T <- lm(NO2 ~ Temp, data = no2) # fits regression.
plot(NO2 ~ Temp, data = no2)
abline(lm.T, col = 4, lty = 2) # col: colour; lty=2: dashed line
summary(lm.T) # regression summary (details later)

title("Titel xy") adds a title to your graphic and clicking the print button in the *R*-window prints the graphic.

Note that there is a distinction between "high-level"- (such as **plot**, **hist**) and "low-level"-graphics commands (such as **abline**). The former make up a new graphic, while the latter add something to existing graphics.

Getting R-help
If you want to know the details about commands, you can use the *R*-online help. For example, **help(plot)** explains the **plot**-command. You can execute the example at the end of the help page by **example(plot)**. Note that it is a good idea to execute **par(ask=TRUE)** first, to give you time to observe the graphics. You may check **help(par)** to understand this.

An alternative to the **help**-command: **help.start()** starts the html-help of *R* in a web browser.

If you look for help about some topic without knowing the command names, e.g., about histograms, **help.search("histogram")** delivers a list of commands which correspond to the keyword. In parantheses you find the name of the package to which the command belongs. Most commands used by us in the beginning are contained in the package "base", which is automatically loaded. Other packages must be loaded by **library(Libname)**, before their commands and help pages are accessible.

Ending *R*
You can save your work by saving the file of commands *tutorial.R* (see above; of course it is useful to use new files for new projects, e.g., *exercise1.R*, *exercise2.R*, ...). The commands have to be evaluated again to restore your work. *R*-objects may be saved also by **save** and **write**.

The command **q()** terminates the *R*-session. Answer **NO** to the question
Save workspace image?