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).

Getting started with R

The computers in the student computer rooms have a dualboot installation (Linux and Windows). If you are in a Linux System, restart your computer and choose the Windows system. Log on with your student account and start R with
Start / All Programs / R / R 2.7.2.

Creating and deleting objects

Type in the *R*-window (***R*** is written on the bar below):
> 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)**. In order to remove all objects in the working environment, use **rm(list=ls())**

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.
The following instructions are with respect to Tinn-R. Tinn-R is free GNU software. If you like to install Tinn-R on your own computer, you have to download the software from `http://sourceforge.net/projects/tinn-r/`. In the computer rooms, Tinn-R is already available.
Before using Tinn-R you have to check (or change) the following settings:

Menu *Bearbeiten / GUI Einstellungen*. There you select *SDI* (instead of *MDI*). Save the new settings (click on *Save*). Leave the new window unchanged, and save the settings as suggested (*My Documents\Rconsole*) (click on *Save*). Then close the Rgui configuration editor with a click on *OK* or *Cancel* (ignoring the warning) und close *R* (do not save the workspace).

In order to work with *R* and Tinn-R, you have to start *R* (*Start / All Programs / R / R 2.7.2*) . Then start Tinn-R (*Start / All Programs / Tinn-R / Tinn-R*). The starting of Tinn-R requires time. If you get a warning, then just click *OK*.

Generate a folder *RFiles* in your home directory *(My Computer\T:)*, most suitable in the Windows Explorer with *File / New / Folder*.

Open a new file in (*File / New*). Type on the first row **z <- c(8, 13, 21)** and on the next row **2 * z**. Save the file under the name *tutorial.R* in the folder *RFiles*. With the icon top left (i. e. *Send all*, *Send selection* und *Send current line*) the editor sends the commands directly to *R*. Mark the two rows you have just written with the left mouse button. Click on the icon *Send selection*. The commands are evaluated in *R*.

As a matter of course, you can type your commands directly into *R*. Nevertheless, you should write all your commands into an *.R-File of the editor due to the fact you can save the file at the end of your session. When you continue your task the next time, open the file, mark all the command and send them to *R*. Now all R-objects are available.

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 the 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 evaluate the line. This evaluates all marked lines. Check the results. Do you understand them?

Now create the sequence 2, 4, 6 as object s: **s <- 2*(1:3)**, alternatively **s <- seq(2,6,by=2)**. 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 **dev.print()** 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** or by creating a new output file and use of the copy, cut and paste facilities of Emacs (see above, or via **Edit** in the menu bar).

The command **q()** terminates the *R*-session. Answer **n** to the question
Save workspace image? [y/n/c] or use **q("no")**.

More to come

R can be used to create complex programs and functions. You may take a look at **help(for)** for control commands or at **help(function)** for creating functions.