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).

R-environments

A "professional" way of working with *R* is to edit *R*-script files in an editor and to transfer the written code to a running *R* process. This can be set up on any platform. Below, we will describe one possible setting under Windows which was chosen for this exercise class. However, this is not the only possibility. Depending on which platform you are working on, we recommend the following:

linux Emacs with the add-on package Emacs Speaks Statistics (`http://stat.ethz.ch/ESS/`)

mac The built-in editor is already quite powerful (`http://cran.r-project.org/bin/macosx/`)

windows TinnR (`http://www.sciviews.org/Tinn-R/`) and WinEdt (`http://www.winedt.com/` and `http://cran.r-project.org/web/packages/RWinEdt/index.html`) both support R syntax highlighting and direct submission of *R* code

platform independent Java GUI for *R* (`http://jgr.markushelbig.org/JGR.html`) or Eclipse (`http://www.eclipse.org/`) with StatEt (`http://www.walware.de/goto/statet/`)

Getting started with *R*

To start *R*: *Start / All Programs / R /R 2.10.1* (or maybe a different version).

Creating and deleting objects

Type in the *R*Console:
> 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 functions 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*-functions 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)**. (Keep pressing *<RETURN>* to browse through the pictures.)

Working with an .R (script-)file
It is useful to type the functions 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.

Now create a new folder *RFiles* in your home directory.

Open a new script file in the editor, use *File / New Script* in the R Console. 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 functions to *R* by marking the functions with the left mouse button and clicking on *Edit / Send line or selection*. 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*-functions 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 to restore 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]**. You also can define matrices with the function **matrix()** (**mat3 <- matrix(x,nrow=2,ncol=2)**).

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:

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

The dataset stream contains the concentration of zinc scaled in four levels (variable ZINC) from different rivers (variable STREAM). In addition the variable DIVERSITY, which depicts the species diversity of the river. The variable ZNGROUP encodes the zinc levels numerically. You may examine the object directly by **stream**. Single variables are accessible by **stream[,"ZINC"]**. 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 of the data frame. **stream** is a small dataset so it can be displayed by calling it directly, but in general it is useful to use the **str** function, which displays the structure and type of an object, but not all elements: **str(stream)**. We see that the variable ZINC is already identified as a factor. However the variable ZNGROUP is identified as a continuous variable.

summary(stream) displays information about the columns of **stream**. **summary** extracts the most important information from lots of *R*-objects, e.g., the results of statistical tests or model fits.

Graphics

Draw a histogram of the DIVERSITY-values of the stream-data.

par(mfrow = c(1,2)) # Number of pictures one below the other [1] or side by side [2]
important to save paper!

hist(stream[,"DIVERSITY"]) # draw histogram.

Now draw a scatter-plot of the diversity against ZNGROUP:

plot(stream[,"ZNGROUP"],stream[,"DIVERSITY"]) # produces the scatter-plot.

Compute an analysis of variance in order to test if there is a difference in diversity for different amounts of zinc: Therefore you first have to change ZNGROUP into a factor.

stream[,"ZNGROUP"] <- as.factor(stream[,"ZNGROUP"])

fit.av <- aov(DIVERSITY ~ ZNGROUP, data = stream) # ANOVA.

summary(fit.av) # returns the model summary.

In order to check the normality assumptions draw a normal Q-Q plot:
You can extract the residuals of fit.av with **resid(fit.av)** and the fitted values with **fitted(fit.av)**.

par(mfrow = c(1,1)) # setting device back to one picture.

qqnorm(resid(fit.av)) # draws the normal Q-Q plot.

qqline(resid(fit.av)) # adds the corresponding diagonal.

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

Note that there is a distinction between "high-level"- (such as **plot**, **hist**) and "low-level"-graphic functions (such as **qqline**). 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 a function, you can use the *R*-online help. For example, **help(plot)** explains the **plot**-function. 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 the previous advice.

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

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

Ending *R*

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

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