A Brief Guide to **R** for Beginners in Econometrics

Mahmood Arai
Department of Economics, Stockholm University
First Version: 2002-11-05, This Version: 2009-09-02

1 Introduction

1.1 About R

R is published under the GPL (GNU Public License) and exists for all major platforms. **R** is described on the **R** Homepage as follows:

> *"R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To download R, please choose your preferred CRAN-mirror."*

See **R** Homepage for manuals and documentations. There are a number of books on **R**. See `http://www.R-project.org/doc/bib/R.bib` for a bibliography of the R-related publications. Dalgaard [2008] and Fox [2002] are nice introductory books. For an advanced book see Venables and Ripley [2002] which is a classic reference. Kleiber and Zeileis [2008] covers econometrics. See also CRAN Task View: Computational Econometrics. For weaving **R** and LaTeXsee Sweave `http://www.stat.uni-muenchen.de/~leisch/Sweave/`. For reproducible research using **R** see Koenker and Zeileis [2007]. To cite **R** in publications you can refer to: R Development Core Team [2008]

1.2 About these pages

This is a brief manual for beginners in Econometrics. For latest version see `http://people.su.se/~ma/R_intro/R_intro.pdf`. For the Sweave file producing these pages see `http://people.su.se/~ma/R_intro/R_intro.Rnw`. The symbol `#` is used for comments. Thus all text after `#` in a line is a comment. Lines following `>` are **R**-commands executed at the **R** prompt which as standard looks like `>`. This is an example:

```
> myexample <- "example"
> myexample

[1] "example"
```

R-codes including comments of codes that are not executed are indented as follows:

```
myexample <- "example" # creates an object named <myexample>
myexample
```

The characters within < > refer to verbatim names of files, functions etc. when it is necessary for clarity. The names <mysomething> such as <mydata>,<myobject> are used to refer to a general dataframe, object etc.

1.3 Objects and files

R regards things as *objects*. A dataset, vector, matrix, results of a regression, a plot etc. are all objects. One or several objects can be saved in a file. A file containing **R**-data is not an object but a set of objects.

Basically all commands you use are *functions*. A command: something(object), does something on an object. This means that you are going to write lots of parentheses. Check that they are there and check that they are in the right place.

2 First things

2.1 Installation

R exists for several platforms and can be downloaded from [CRAN-mirror].

2.2 Working with R

It is a good idea to create a directory for a project and start **R** from there. This makes it easy to save your work and find it in later sessions.

If you want R to start in a certain directory in *MS-Windows*, you have to specify the <start in directory> to be your working directory. This is done by changing the <properties> by clicking on the right button of the mouse while pointing at your R-icon, and then going to <properties>.

*Displaying the working directory within **R**:*

```
> getwd()

[1] "/home/ma/1/R/R_begin/R_Brief_Guide"
```

Changing the working directory to an existing directory </home/ma/project1>

```
setwd("/home/ma/project1")
```

2.3 Naming in R

Do not name an object as <my_object> or <my-object> use instead <my.object>. Notice that in **R** <my.object> and <My.object> are two different names. Names starting with a digit (<1a>) is not accepted. You can instead use <a1>)

You should not use names of variables in a data-frame as names of objects. If you do so, the object will shadow the variable with the same name in another object. The problem is then that when you call this variable you will get the object – the object shadows the variable / the variable will be masked by the object with the same name.

To avoid this problem:

1- Do not give a name to an object that is identical to the name of a variable in your data frames.

2- If you are not able to follow this rule, refer to variables by referring to the variable and the dataset that includes the variable. For example the variable <wage> in the data frame <df1> is called by:

```
df1$wage.
```

The problem of "shadowing" concerns **R** functions as well. Do not use object names that are the same as **R** functions. <conflicts(detail=TRUE)> checks whether an object you have created conflicts with another object in the **R** packages and lists them. You should only care about those that are listed under <.GlobalEnv> – objects in your workspace. All objects listed under <.GlobalEnv> shadows objects in **R** packages and should be removed in order to be able to use the objects in the **R** packages.

The following example creates <T> that should be avoided (since <T> stands for <TRUE>), checks conflicts and resolves the conflict by removing <T>.

```
T <- "time"
conflicts(detail=TRUE)
rm(T)
conflicts(detail=TRUE)
```

You should avoid using the following one-letter words <c,C,D,F,I,q,t,T> as names. They have special meanings in **R**.

Extensions for files

It is a good practice to use the extension <R> for your files including R-codes. A file <lab1.R> is then a text-file including R-codes.

The extension <rda> is appropriate for work images (i.e files created by <save()>). The file <lab1.rda> is then a file including R-objects.

The default name for the saved work image is <.RData>. Be careful not to name a file as <.RData> when you use <RData> as extension, since you will then overwrite the <.Rdata> file.

2.4 Saving and loading objects and images of working spaces

Download the file <DataWageMacro.rda> http://people.su.se/~ma/R_intro/data/.

You can read the file <DataWageMacro.rda> containing the data frames <lnu> and <macro> as follows.

```
load("DataWageMacro.rda")
ls()       # lists the objects
```

The following command saves the object <lnu> in a file <mydata.rda>.

```
save(lnu, file="mydata.rda")
```

To save an image of the your workspace that will be automatically loaded when you next time start **R** in the same directory.

```
save.image()
```

You can also save your working image by answering <yes> when you quit and are asked
<Save workspace image? [y/n/c]:>.

In this way the image of your workspace is saved in the hidden file <.RData>.

You can save an image of the current workspace and give it a name <myimage.rda>.

```
save.image("myimage.rda")
```

2.5 Overall options

<options()> can be used to set a number of options that governs various aspects of computations and displaying results.

Here are some useful options. We start by setting the line with to 60 characters.

```
> options(width = 60)
```

```
options(prompt="   R> ") # changes the prompt to <   R> >.
options(scipen=3)        # From R version 1.8. This option
# tells R to display numbers  in fixed format instead of
# in exponential form, for example <1446257064291> instead of
# <1.446257e+12> as the result of <exp(28)>.

options()                # displays the options.
```

2.6 Getting Help

```
help.start()  # invokes the help pages.
help(lm)      # help on <lm>, linear model.
?lm           # same as above.
```

3 Elementary commands

```
ls()              # Lists all objects.
ls.str()          # Lists details of all objects
str(myobject)     # Lists details of <myobject>.
list.files()      # Lists all files in the current directory.
dir()             # Lists all files in the current directory.
myobject          # Prints simply the object.
rm(myobject)      # removes the object <myobject>.
rm(list=ls())     # removes all the objects in the working space.

save(myobject, file="myobject.rda")
# saves the object <myobject> in a file <myobject.rda>.

load("mywork.rda")# loads "mywork.rda" into memory.

summary(mydata)   # Prints the simple statistics for <mydata>.
hist(x,freq=TRUE) # Prints a histogram of the object <x>.
# <freq=TRUE> yields frequency and
# <freq=FALSE> yields probabilities.

q()               # Quits R.
```

The output of a command can be directed in an object by using `< <- >` , an object is then assigned a value. The first line in the following code chunk creates vector named `<VV>` with a values 1,2 and 3. The second line creates an object named `<VV>` and prints the contents of the object `<VV>`.

```
> VV <- c(1, 2, 3)
> (VV <- 1:2)

[1] 1 2
```

4 Data management

4.1 Reading data in plain text format:

Data in columns

The data in this example are from a text file: `<tmp.txt>`, containing the variable names in the first line (separated with a space) and the values of these variables (separated with a space) in the following lines.

The following reads the contents of the file <tmp.txt> and assigns it to an object named <dat>.

```
> FILE <- "http://people.su.se/~ma/R_intro/data/tmp.txt"
> dat <- read.table(file = FILE, header = TRUE)
> dat

  wage school public female
1   94      8      1      0
2   75      7      0      0
3   80     11      1      0
4   70     16      0      0
5   75      8      1      0
6   78     11      1      0
7  103     11      0      0
8   53      8      0      0
9   99      8      1      0
```

The argument <header = TRUE> indicates that the first line includes the names of the variables. The object <dat> is a data-frame as it is called in **R**.

If the columns of the data in the file <tmp.txt> were separated by <,>, the syntax would be:

```
read.table("tmp.txt", header = TRUE, sep=",")
```

Note that if your decimal character is not <.> you should specify it. If the decimal character is <,>, you can use <read.csv> and specify the following argument in the function <dec=",">.

4.2 Non-available and delimiters in tabular data

We have a file <data1.txt> with the following contents:

```
1 . 9
6 3 2
```

where the first observation on the second column (variable) is a missing value coded as <.>. To tell **R** that <.> is a missing value, you use the argument: <na.strings=".">

```
> FILE <- "http://people.su.se/~ma/R_intro/data/data1.txt"
> read.table(file = FILE, na.strings = ".")

  V1 V2 V3
1  1 NA  9
2  6  3  2
```

Sometimes columns are separated by other separators than spaces. The separator might for example be `<,>` in which case we have to use the argument `<sep=",">`.

Be aware that if the columns are separated by `<,>` and there are spaces in some columns like the case below the `<na.strings=".">` does not work. The NA is actually coded as two spaces, a point and two spaces, and should be indicated as: `<na.strings="  .  ">`.

```
1,  .  ,9
6,  3  ,2
```

Sometimes missing value is simply <blank> as follows.

```
1  9
6 3 2
```

Notice that there are two spaces between 1 and 9 in the first line implying that the value in the second column is blank. This is a missing value. Here it is important to specify `<sep=" ">` along with `<na.strings="">` .

4.3 Reading and writing data in other formats

Attach the library `<foreign>` in order to read data in various standard packages data formats. Examples are SAS, SPSS, STATA, etc.

```
library(foreign)
# reads the data <wage.dta> and put it in the object <lnu>
lnu <- read.dta(file="wage.dta")
```

`<read.ssd()>` , `<read.spss()>` etc. are other commands in the foreign package for reading data in SAS and SPSS format.

It is also easy to write data in a foreign format. The following codes writes the object <lnu> to stata-file <lnunew.dta>.

```
library(foreign)
write.dta(lnu,"lnunew.dta")
```

4.4 Examining the contents of a data-frame object

Here we use data from Swedish Level of Living Surveys LNU 1991.

```
> FILE <- "http://people.su.se/~ma/R_intro/data/lnu91.txt"
> lnu <- read.table(file = FILE, header = TRUE)
```

Attaching the <lnu> data by <attach(lnu)> allows you to access the contents of the dataset <lnu> by referring to the variable names in the <lnu>. If you have not attached the <lnu> you can use <lnu$female> to refer to the variable <female> in the data frame <lnu>. When you do not need to have the data attached anymore, you can undo the <attach()> by <detach()>

A description of the contents of the data frame lnu.

```
> str(lnu)
```

```
'data.frame':        2249 obs. of  6 variables:
 $ wage    : int  81 77 63 84 110 151 59 109 159 71 ...
 $ school  : int  15 12 10 15 16 18 11 12 10 11 ...
 $ expr    : int  17 10 18 16 13 15 19 20 21 20 ...
 $ public  : int  0 1 0 1 0 0 1 0 0 0 ...
 $ female  : int  1 1 1 1 0 0 1 0 1 0 ...
 $ industry: int  63 93 71 34 83 38 82 50 71 37 ...
```

```
> summary(lnu)
```

```
      wage            school           expr      
 Min.   : 17.00   Min.   : 4.00   Min.   : 0.00  
 1st Qu.: 64.00   1st Qu.: 9.00   1st Qu.: 8.00  
 Median : 73.00   Median :11.00   Median :18.00  
 Mean   : 80.25   Mean   :11.57   Mean   :18.59  
 3rd Qu.: 88.00   3rd Qu.:13.00   3rd Qu.:27.00  
 Max.   :289.00   Max.   :24.00   Max.   :50.00  
     public           female          industry    
 Min.   :0.0000   Min.   :0.0000   Min.   :11.00  
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:50.00  
 Median :0.0000   Median :0.0000   Median :81.00  
 Mean   :0.4535   Mean   :0.4851   Mean   :69.74  
 3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:93.00  
 Max.   :1.0000   Max.   :1.0000   Max.   :95.00  
```

4.5 Creating and removing variables in a data frame

Here we create a variable <logwage> as the logarithm of <wage>. Then we remove the variable.

```
> lnu$logwage <- log(lnu$wage)
> lnu$logwage <- NULL
```

Notice that you do not need to create variables that are simple transformations of the original variables. You can do the transformation directly in your computations and estimations.

4.6 Choosing a subset of variables in a data frame

```
# Read a <subset>  of variables (wage,female) in lnu.
lnu.female <- subset(lnu, select=c(wage,female))

# Putting together two objects (or variables) in a data frame.
attach(lnu)
lnu.female <- data.frame(wage,female)

# Read all variables in lnu but female.
lnux <- subset(lnu, select=-female)

# The following keeps all variables from wage to public as listed above
lnuxx <- subset(lnu, select=wage:public)
```

4.7 Choosing a subset of observations in a dataset

```
attach(lnu)

# Deleting observations that include missing value in a variable
lnu <- na.omit(lnu)

# Keeping observations for female only.
fem.data <- subset(lnu, female==1)

# Keeping observations for female and public employees only.
fem.public.data <- subset(lnu, female==1 & public==1)

# Choosing all observations where wage > 90
highwage <- subset(lnu, wage > 90)
```

4.8 Replacing values of variables

We create a variable indicating whether the individual has university education or not by replacing the values in the schooling variable.

Copy the schooling variable.

```
> lnu$university <- lnu$school
```

Replace university value with 0 if years of schooling is less than 13 years.

```
> lnu$university <- replace(lnu$university, lnu$university <
+     13, 0)
```

Replace university value with 1 if years of schooling is greater than 12 years

```
> lnu$university <- replace(lnu$university, lnu$university >
+     12, 1)
```

The variable <lnu$university> is now a dummy for university education. Remember to re-attach the data set after recoding. For creating category variables you can use <cut>. See further the section on <factors> below.

```
> attach(lnu, warn.conflicts = FALSE)
> table(university)

university
FALSE  TRUE
 1516   733
```

To create a dummy we could simply proceed as follows:

```
> university <- school > 12
> table(university)

university
FALSE  TRUE
 1516   733
```

However, we usually do not need to create dummies. We can compute on <school > 12> directly,

```
> table(school > 12)

FALSE  TRUE
 1516   733
```

4.9 Replacing missing values

We create a vector. Recode one value as missing value. And Then replace the missing with the original value.

```
a <- c(1,2,3,4)    # creates a vector
is.na(a) <- a ==2  # recode a==2 as NA
a <- replace(a, is.na(a), 2)# replaces the NA with 2
# or
a[is.na(a)] <- 2
```

4.10 Factors

Sometimes our variable has to be redefined to be used as a category variable with appropriate levels that corresponds to various intervals. We might wish to have schooling categories that corresponds to schooling up to 9 years, 10 to 12 years and above 12 years. This could be coded by using <cut()>. To include the lowest category we use the argument <include.lowest=TRUE>.

```
> SchoolLevel <- cut(school, c(9, 12, max(school),
+     include.lowest = TRUE))
> table(SchoolLevel)

SchoolLevel
  (1,9]  (9,12] (12,24]
    608     908     733
```

Labels can be set for each level. Consider the university variable created in the previous section.

```
> SchoolLevel <- factor(SchoolLevel, labels = c("basic",
+     "gymnasium", "university"))
> table(SchoolLevel)

SchoolLevel
     basic  gymnasium university
       608        908        733
```

The factor defined as above can for example be used in a regression model. The reference category is the level with the lowest value. The lowest value is 1 that corresponds to verb+<Basic>+ and the column for <Basic> is not included in the contrast matrix. Changing the base category will remove another column instead of this column. This is demonstrated in the following example:

```
> contrasts(SchoolLevel)

           gymnasium university
basic              0          0
gymnasium          1          0
university         0          1

> contrasts(SchoolLevel) <- contr.treatment(levels(SchoolLevel),
+     base = 3)
> contrasts(SchoolLevel)

           basic gymnasium
basic          1         0
gymnasium      0         1
university     0         0
```

The following redefines `<school>` as a numeric variable.

```
> lnu$school <- as.numeric(lnu$school)
```

4.11 Aggregating data by group

Let us create a simple dataset consisting of 3 variables V1, V2 and V3. V1 is the group identity and V2 and V3 are two numeric variables.

```
> (df1 <- data.frame(V1 = 1:3, V2 = 1:9, V3 = 11:19))
```

```
  V1 V2 V3
1  1  1 11
2  2  2 12
3  3  3 13
4  1  4 14
5  2  5 15
6  3  6 16
7  1  7 17
8  2  8 18
9  3  9 19
```

By using the command <aggregate> we can create a new data.frame consisting of group characteristics such as <sum> , <mean> etc. Here the function sum is applied to <df1[,2:3]> that is the second and third columns of <df1> by the group identity <V1>.

```
> (aggregate.sum.df1 <- aggregate(df1[, 2:3], list(df1$V1),
+     sum))
```

```
  Group.1 V2 V3
1       1 12 42
2       2 15 45
3       3 18 48
```

```
> (aggregate.mean.df1 <- aggregate(df1[, 2:3], list(df1$V1),
+     mean))
```

```
  Group.1 V2 V3
1       1  4 14
2       2  5 15
3       3  6 16
```

The variable <Group.1> is a factor that identifies groups.

The following is an example of using the function aggregate. Assume that you have a data set <dat> including a unit-identifier <dat$id>. The units are observed repeatedly over time indicated by a variable dat$Time.

```
> (dat <- data.frame(id = rep(11:12, each = 2),
+     Time = 1:2, x = 2:3, y = 5:6))
```

```
  id Time x y
1 11    1 2 5
2 11    2 3 6
3 12    1 2 5
4 12    2 3 6
```

This computes group means for all variables in the data frame and drops the variable <Time> and the automatically created group-indicator variable <Group.1>.

```
> (Bdat <- subset(aggregate(dat, list(dat$id), FUN = mean),
+     select = -c(Time, Group.1)))
```

```
  id   x   y
1 11 2.5 5.5
2 12 2.5 5.5
```

Merge <Bdat> and <dat$id> to create a data set with repeated group averages for each observation on <id> and of the length as <id>.

```
> (dat2 <- subset(merge(data.frame(id = dat$id),
+     Bdat), select = -id))
```

```
    x   y
1 2.5 5.5
2 2.5 5.5
3 2.5 5.5
4 2.5 5.5
```

Now you can create a data set including the <id> and <Time> indicators and the deviation from mean values of all the other variables.

```
> (within.data <- cbind(id = dat$id, Time = dat$Time,
+     subset(dat, select = -c(Time, id)) - dat2))
```

```
  id Time    x    y
1 11    1 -0.5 -0.5
2 11    2  0.5  0.5
3 12    1 -0.5 -0.5
4 12    2  0.5  0.5
```

4.12 Using several data sets

We often need to use data from several datasets. In **R** it is not necessary to put these data together into a dataset as is the case in many statistical packages where only one data set is available at a time and all stored data are in the form of a table.

It is for example possible to run a regression using one variable from one data set and another variable from another dataset as long as these variables have the same length (same number of observations) and they are in the same order (the i:th observation in both variables correspond to the same unit). Consider the following two datasets:

```
> data1 <- data.frame(wage = c(81, 77, 63, 84, 110,
+     151, 59, 109, 159, 71), female = c(1, 1, 1,
+     1, 0, 0, 1, 0, 1, 0), id = c(1, 3, 5, 6, 7,
+     8, 9, 10, 11, 12))
> data2 <- data.frame(experience = c(17, 10, 18,
+     16, 13, 15, 19, 20, 21, 20), id = c(1, 3,
+     5, 6, 7, 8, 9, 10, 11, 12))
```

We can use variables from both datasets without merging the datasets. Let us regress `<data1$wage>` on `<data1$female>` and `<data2$experience>`.

```
> lm(log(data1$wage) ~ data1$female + data2$experience)

Call:
lm(formula = log(data1$wage) ~ data1$female + data2$experience)

Coefficients:
     (Intercept)      data1$female  data2$experience
        4.641120         -0.257909          0.001578
```

We can also put together variables from different data frames into a data frame and do our analysis on these data.

```
> (data3 <- data.frame(data1$wage, data1$female,
+     data2$experience))

   data1.wage data1.female data2.experience
1          81            1               17
2          77            1               10
3          63            1               18
4          84            1               16
5         110            0               13
6         151            0               15
7          59            1               19
8         109            0               20
9         159            1               21
10         71            0               20
```

We can merge the datasets. If we have one common variable in both data sets, the data is merged according to that variable.

```
> (data4 <- merge(data1, data2))
```

```
   id wage female experience
1   1   81      1         17
2   3   77      1         10
3   5   63      1         18
4   6   84      1         16
5   7  110      0         13
6   8  151      0         15
7   9   59      1         19
8  10  109      0         20
9  11  159      1         21
10 12   71      0         20
```

Notice that unlike some other softwares, we do not need the observations to appear in the same order as defined by the <id>.

If we need to match two data sets using a common variable (column) and the common variable have different names in the datasets, we either can change the names to the same name or use the data as they are and specify the variables that are to be used for matching in the data sets. If the matching variable in <data2> and <data1> are called <id2> and <id> you can use the following syntax:

```
merge(data1,data2, by.x="id", by.y="id2")
```

<by.x="id", by.y="id2"> arguments says that id is the matching variable in data1 and id2 is the matching variable in data2.

You can also put together the datasets in the existing order with help of <data.frame> or <cbind>. The data are then matched, observation by observation, in the existing order in the data sets. This is illustrated by the following example.

```
> data1.noid <- data.frame(wage = c(81, 77, 63),
+     female = c(1, 0, 1))
> data2.noid <- data.frame(experience = c(17, 10,
+     18))
> cbind(data1.noid, data2.noid)

  wage female experience
1   81      1         17
2   77      0         10
3   63      1         18
```

If you want to add a number of observations at the end of a data set, you use <rbind>. The following example splits the clumns 2,3 and 4 in <data4> in two parts and then puts themtogether by <rbind>.

```
> data.one.to.five <- data4[1:5, 2:4]
> data.six.to.ten <- data4[6:10, 2:4]
> rbind(data.one.to.five, data.six.to.ten)
```

```
   wage female experience
1    81      1         17
2    77      1         10
3    63      1         18
4    84      1         16
5   110      0         13
6   151      0         15
7    59      1         19
8   109      0         20
9   159      1         21
10   71      0         20
```

5 Basic statistics

Summary statistics for all variables in a data frame:

```
summary(mydata)
```

Mean, Median, Standard deviation, Maximum, and Minimum of a variable:

```
mean (myvariable)
median (myvariable)
sd (myvariable)
max (myvariable)
min (myvariable)
# compute 10, 20, ..., 90 percentiles
quantile(myvariable, 1:9/10)
```

When **R** computes <sum> , <mean> etc on an object containing <NA>, it returns <NA>. To be able to apply these functions on observations where data exists, you should add the argument <na.rm=TRUE>. Another alternative is to remove all lines of data containing <NA> by <na.omit>.

```
> a <- c(1, NA, 3, 4)
> sum(a)

[1] NA

> sum(a, na.rm = TRUE)

[1] 8

> table(a, exclude = c())

a
   1    3    4 <NA>
   1    1    1    1
```

You can also use <sum(na.omit(a))> that removes the NA and computes the sum or <sum(a[!is.na(a)])> that sums the elements that are not NA (!is.na) in <a>.

5.1 Tabulation

Read a dataset first.

Cross Tabulation

```
> attach(lnu, warn.conflicts = FALSE)
> table(female, public)

      public
female   0   1
     0 815 343
     1 414 677

> (ftable.row <- cbind(table(female, public), total = table(female)))

    0   1 total
0 815 343  1158
1 414 677  1091

> (ftable.col <- rbind(table(female, public), total = table(public)))

         0    1
0      815  343
1      414  677
total 1229 1020

     # Try this:
     # relative freq. by rows: female
     ftable.row/c(table(female))
     # relative freq. by columns: public
     ftable.col/rep(table(public),each=3)
     # rep(table(public),each=3) repeats
     #each value in table(public) 3 times
```

Creating various statistics by category. The following yields average wage for males and females.

```
> tapply(wage, female, mean)

       0        1
88.75302 71.23190
```

Using <length>, <min>, <max>, etc yields number of observations, minimum, maximum etc for males and females.

```
> tapply(wage, female, length)

   0    1
1158 1091
```

The following example yields average wage for males and females in the private and public sector.

```
> tapply(wage, list(female, public), mean)

         0        1
0 89.52883 86.90962
1 71.54589 71.03988
```

The following computes the average by group creating a vector of the same length. Same length implies that for the group statistics is retained for all members of each group. Average wage for males and females:

```
> attach(lnu, warn.conflicts = FALSE)
> lnu$wage.by.sex <- ave(wage, female, FUN = mean)
```

The function <mean> can be substituted with <min>, <max>, <length> etc. yielding group-wise minimum, maximum, number of observations, etc.

6 Matrixes

In R we define a matrix as follows (see ?matrix in R):

A matrix with 3 rows and 4 columns with elements 1 to 12 filled by columns.

```
> matrix(1:12, 3, 4)

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

A matrix with 3 rows and 4 columns with elements 1,2,3, ..., 12 filled by rows:

```
> (A <- matrix(1:12, 3, 4, byrow = TRUE))

     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

```
> dim(A)

[1] 3 4

> nrow(A)

[1] 3

> ncol(A)

[1] 4
```

6.1 Indexation

The elements of a matrix can be extracted by using brackets after the matrix name and referring to rows and columns separated by a comma. You can use the indexation in a similar way to extract elements of other types of objects.

```
A[3,]        # Extracting the third row
A[,3]        # Extracting the third column
A[3,3]       # the third row and the third column
A[-1,]       # the matrix except the first row
A[,-2]       # the matrix except the second column
```

Evaluating some condition on all elements of a matrix

```
> A > 3

      [,1]  [,2]  [,3] [,4]
[1,] FALSE FALSE FALSE TRUE
[2,]  TRUE  TRUE  TRUE TRUE
[3,]  TRUE  TRUE  TRUE TRUE

> A == 3

      [,1]  [,2]  [,3]  [,4]
[1,] FALSE FALSE  TRUE FALSE
[2,] FALSE FALSE FALSE FALSE
[3,] FALSE FALSE FALSE FALSE
```

Listing the elements fulfilling some condition

```
> A[A > 6]

[1]  9 10  7 11  8 12
```

6.2 Scalar Matrix

A special type of matrix is a scalar matrix which is a square matrix with the same number of rows and columns, all off-diagonal elements equal to zero and the same element in all diagonal positions. The following exercises demonstrates some matrix facilities regarding the diagonals of matrixes. See also `?upper.tri` and `?lower.tri`.

```
> diag(2, 3, 3)

     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    2    0
[3,]    0    0    2

> diag(diag(2, 3, 3))

[1] 2 2 2
```

6.3 Matrix operators

Transpose of a matrix

Interchanging the rows and columns of a matrix yields the transpose of a matrix.

```
> t(matrix(1:6, 2, 3))

     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

Try matrix(1:6,2,3) and matrix(1:6,3,2, byrow=T).

Addition and subtraction

Addition and subtraction can be applied on matrixes of the same dimensions or a scalar and a matrix.

```
#  Try this
A <- matrix(1:12,3,4)
B <- matrix(-1:-12,3,4)
C1 <- A+B
D1 <- A-B
```

Scalar multiplication

```
# Try this
A <- matrix(1:12,3,4); TwoTimesA = 2*A
c(2,2,2)*A
c(1,2,3)*A
c(1,10)*A
```

Matrix multiplication

For multiplying matrixes R uses <%∗%> and this works only when the matrixes are conform.

```
E <- matrix(1:9,3,3)
crossproduct.of.E <- t(E)%*%E
# Or another and more efficient way of obtaining crossproducts is:
crossproduct.of.E <- crossprod(E)
```

Matrix inversion

The inverse of a square matrix $\mathbf{A}$ denoted as $\mathbf{A}^{-1}$ is defined as a matrix that when multiplied with $\mathbf{A}$ results in an Identity matrix (1's in the diagonal and 0's in all off-diagonal elements.)

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

```
FF <- matrix((1:9),3,3)
detFF<- det(FF)          # we check the determinant

B <- matrix((1:9)^2,3,3)  # create an invertible matrix
Binverse <- solve(B)
Identity.matrix <- B%*%Binverse
```

7 Ordinary Least Squares

The function for running a linear regression model using OLS is `<lm()>`. In the following example the dependent variable is `<log(wage)>` and the explanatory variables are `<school>` and `<female>`. An intercept is included by default. Notice that we do not have to specify the data since the data frame `<lnu>` containing these variables is attached. The result of the regression is assigned to the object named `<reg.model>`. This object includes a number of interesting regression results that can be extracted as illustrated further below after some examples for using `<lm>`.

Read a dataset first.

```
> reg.model <- lm(log(wage) ~ school + female)
> summary(reg.model)

Call:
lm(formula = log(wage) ~ school + female)

Residuals:
     Min       1Q   Median       3Q      Max
-1.46436 -0.15308 -0.01852  0.13542  1.10402
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.087730   0.022203  184.10   <2e-16 ***
school       0.029667   0.001783   16.64   <2e-16 ***
female      -0.191109   0.011066  -17.27   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2621 on 2246 degrees of freedom
Multiple R-squared: 0.2101,        Adjusted R-squared: 0.2094
F-statistic: 298.8 on 2 and 2246 DF,  p-value: < 2.2e-16
```

Sometimes we wish to run the regression on a subset of our data.

```
lm (log(wage)  ~   school + female, subset=wage>100)
```

Sometimes we need to use transformed values of the variables in the model. The transformation should be given as the in the function <I()>. I() means Identity function. `<expr^2>` is `<expr>` squared.

```
lm (log(wage) ~  school + female + expr + I(expr^2))
```

Interacting variables: `<female>`, `<school>`

```
lm (log(wage)  ~   female*school, data=lnu)
```

Same as:

```
lm (log(wage) ~ female + school + female:school, data= lnu)
```

A model with no intercept.

```
reg.no.intercept <- lm (log(wage)  ~   female - 1)
```

A model with only an intercept.

```
reg.only.intercept <- lm (log(wage)  ~   1 )
```

The following example runs `<lm>` for females and males in the private and public sector separately as defined by the variables `<female>` and `<public>`. The data `<lnu>` are split into four cells: males in private (0,0), females in private (1,0), males in public (0,1) and females in public (1,1). The obtained object `<by.reg>` is a list and to display the `<summary()>` of each element we use `<lapply>` (list apply).

```
by.reg <- by(lnu, list(female,public),
          function(x) lm(log(wage) ~ school, data=x))
# summary of the separate regressions
lapply(by.reg, summary)
# summary for the second element in the
#list i.e females in private sector.
summary(by.reg[[2]])
```

The following lists mean of variables for male and female workers (the first line), creates a list named `by.female.lnu` of two data sets (the second line) and runs regressions for male and female workers (the third and fourth lines).

```
by(lnu, list(female), mean)
by.female.lnu <- by(lnu, list(female),
      function(x) x); str(by.female.lnu)
summary(lm(log(wage) ~ school, data=by.female.lnu[[1]]))
summary(lm(log(wage) ~ school, data=by.female.lnu[[2]]))
```

7.1 Extracting the model formula and results

The model formula

```
(equation1 <- formula(reg.model))
log(wage) ~ school + female
```

The estimated coefficients

```
> coefficients(reg.model)

(Intercept)      school      female
 4.08772967  0.02966711 -0.19110920
```

The standard errors

```
> coef(summary(reg.model))[, 2]

(Intercept)      school      female
0.022203334 0.001782725 0.011066176
```

`<coef(summary(reg.model))[,1:2]>` yields both `<Estimate>` and `<Std.Error>`

The t-values

```
> coef(summary(reg.model))[, 3]

(Intercept)      school      female
  184.10432    16.64144   -17.26967
```

Try also <coef(summary(reg.model))>. Analogously you can extract other elements of the lm-object by:

The variance-covariance matrix: <vcov(reg.model)> :

Residual degrees of freedom:
<df.residual(reg.model)>

The residual sum of squares:
<deviance(reg.model)>

And other components:
<residuals(reg.model)>
<fitted.values(reg.model)>
<summary(reg.model)$r.squared>
<summary(reg.model)$adj.r.squared>
<summary(reg.model)$sigma>
<summary(reg.model)$fstatistic>

7.2 White's heteroskedasticity corrected standard errors

The package <car> and <sandwich> and <Design> has predefined functions for computing robust standard errors. There are different weighting options.

The White's correction

```
> library(car)
> f1 <- formula(log(wage) ~ female + school)
> sqrt(diag(hccm(lm(f1), type = "hc0")))

(Intercept)      female      school
0.022356311 0.010920889 0.001929391
```

Using the library <sandwich>.

```
> library(sandwich)
> library(lmtest)
> coeftest(lm(f1), vcov = (vcovHC(lm(f1), "HC0")))

t test of coefficients:

             Estimate Std. Error t value  Pr(>|t|)
(Intercept)  4.0877297  0.0223563 182.845 < 2.2e-16 ***
female      -0.1911092  0.0109209 -17.499 < 2.2e-16 ***
school       0.0296671  0.0019294  15.376 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

<hc0> in library <car> and <HC0> in library sandwich use the original White formula. The <hc1> <HC1> multiply the variances with $\frac{N}{N-k}$. Using library <Design>.

```
> library(Design, war, warn.conflicts = FALSE)
> f1 <- formula(log(wage) ~ female + school)
> fm1 <- robcov(ols(f1, x = TRUE, y = TRUE))
```

7.3 Group-wise non-constant error variance

This uses the library <Design> and account for non-constant error variance when data is clustered across sectors indicated by the variable <industry>

```
> robcov(ols(f1, x = TRUE, y = TRUE), cluster = industry)

Linear Regression Model

ols(formula = f1, x = TRUE, y = TRUE)

         n Model L.R.       d.f.         R2      Sigma
      2249      530.5          2     0.2101     0.2621

Residuals:
     Min       1Q   Median       3Q      Max
-1.46436 -0.15308 -0.01852  0.13542  1.10402

Coefficients:
             Value Std. Error      t Pr(>|t|)
Intercept  4.08773   0.036285 112.66        0
female    -0.19111   0.016626 -11.49        0
school     0.02967   0.002806  10.57        0

Residual standard error: 0.2621 on 2246 degrees of freedom
Adjusted R-Squared: 0.2094
```

When the number of groups M are small, you can correct the standard errors as follows:

```
> library(Design)
> f1 <- formula(log(wage) ~ female + school)
> M <- length(unique(industry))
> N <- length(industry)
> K <- lm(f1)$rank
> cl <- (M/(M - 1)) * ((N - 1)/(N - K))
> fm1 <- robcov(ols(f1, x = TRUE, y = TRUE), cluster = industry)
> fm1$var <- fm1$var * cl
> coeftest(fm1)
```

```
t test of coefficients:

            Estimate Std. Error t value  Pr(>|t|)
Intercept  4.0877297  0.0222033 184.104 < 2.2e-16 ***
female    -0.1911092  0.0110662 -17.270 < 2.2e-16 ***
school     0.0296671  0.0017827  16.641 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

7.4 F-test

Estimate the restricted (restricting some (or all) of slope coefficients to be zero) and the unrestricted model (allowing non-zero as well as zero coefficients). You can then use `anova()` to test the joint hypotheses defined as in the restricted model.

```
> mod.restricted <- lm(log(wage) ~ 1)
> mod.unrestricted <- lm(log(wage) ~ female + school)
> anova(mod.restricted, mod.unrestricted)

Analysis of Variance Table

Model 1: log(wage) ~ 1
Model 2: log(wage) ~ female + school
  Res.Df     RSS   Df Sum of Sq      F    Pr(>F)
1   2248 195.338
2   2246 154.291    2    41.047 298.76 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Under non-constant error variance, we use the White variance-Covariance matrix and the model F-value is as follows. The `<-1>` in the codes below remove related row/column for the intercept

```
> library(car)
> COV <- hccm(mod.unrestricted, "hc1")[-1, -1]
> beta <- matrix(coef(mod.unrestricted, , 1))[-1,
+     ]
> t(beta) %*% solve(COV) %*% beta/(lm(f1)$rank -
+     1)

         [,1]
[1,] 253.1234
```

8 Time-series

Data in rows

Time-series data often appear in a form where series are in rows. As an example I use data from the Swedish Consumer Surveys by the National Institute of Economic Research containing three series: consumer confidence index, a macro index and a micro index.

First I saved the original data in a file in text-format. Before using the data as input in **R** I used a text editor and kept only the values for the first three series separated with spaces. The series are in rows. The values of the three series are listed in separate rows without variable names.

To read this file `<macro.txt>`, the following code puts the data in a matrix of 3 columns and 129 rows with help of `<scan>` and `<matrix>` before defining a time-series object starting in 1993 with frequency 12 (monthly). The series are named as `<cci>,<macro.index>` and `<micro.index>`. Notice that `<matrix>` by default fills in data by columns.

```
> FILE <- "http://people.su.se/~ma/R_intro/macro.txt"
> macro <- ts(matrix(scan(FILE), 129, 3), start = 1993,
+     frequency = 12, names = c("cci", "macro.index",
+         "micro.index"))
```

Here I give an example for creating lag values of a variable and adding it to a time-series data set. See also `<diff>` for computing differences.

Let us create a new time series data set, with the series in the data frame `<macro>` adding lagged `<cci>` (lagged by 1 month). The function `<ts.union>` puts together the series keeping all observations while `<ts.intersect>` would keep only the overlapping part of the series.

```
> macro2 <- ts.union(macro, l.cci = lag(macro[,
+     1], -1))
```

You can use the function aggregate to change the frequency of you time-series data. The following example converts the frequency data. `nfrequency=1` yields annual data. `FUN=mean` computes the average of the variables over time. The default is `<sum>`.

```
> aggregate(macro, nfrequency = 1, FUN = mean)

Time Series:
Start = 1993
End = 2002
Frequency = 1
             cci macro.index micro.index
1993 -19.7583333 -33.3833333  -13.241667
1994  -0.2916667  14.1666667   -5.158333
```

```
1995 -10.8583333  -3.7250000  -10.250000
1996  -7.9166667 -20.5500000   -3.866667
1997   3.5333333   0.7833333    2.141667
1998  11.9083333  16.3500000    7.250000
1999  17.6083333  20.6083333   11.891667
2000  26.2666667  40.4083333   14.925000
2001   3.4583333 -13.7083333    9.608333
2002   7.2166667  -7.3250000   10.125000
```

8.1 Durbin Watson

<dwtest> in the package <lmtest> and <durbin.watson> in the package <car> can be used. See also <bgtest> in the package <lmtest> for Breusch-Godfrey test for higher order serial correlation.

```
> mod1 <- lm(cci ~ macro.index, data = macro)
> library(lmtest)
> dwtest(mod1)

        Durbin-Watson test

data:  mod1
DW = 0.063, p-value < 2.2e-16
alternative hypothesis: true autocorrelation is greater than 0
```

9 Graphics

9.1 Save graphs in postscript

```
postscript("myfile.ps")
hist(1:10)
dev.off()
```

9.2 Save graphs in pdf

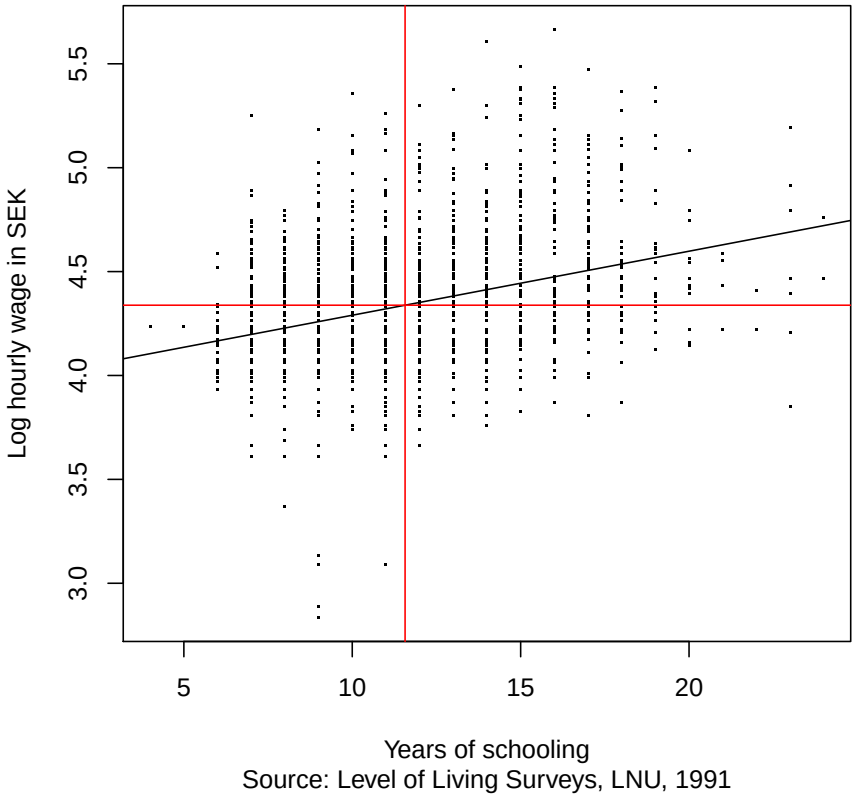
```
pdf("myfile.pdf")
hist(1:10)
dev.off()
```

9.3 Plotting the observations and the Regression line

Plot the data and the regression line. Plot `<school>` against `<log(wage)>`

```
> X.LABEL = "Years of schooling"
> Y.LABEL = "Log hourly wage in SEK"
> TITLE <- "Figure 1: Scatterplot and the Regression line"
> SubTitle <- "Source: Level of Living Surveys, LNU, 1991"
> plot(school, log(wage), pch = ".", main = TITLE,
+     sub = SubTitle, xlab = X.LABEL, ylab = Y.LABEL)
> abline(lm(log(wage) ~ school))
> abline(v = mean(school), col = "red")
> abline(h = mean(log(wage)), col = "red")
```

Figure 1: Scatterplot and the Regression line



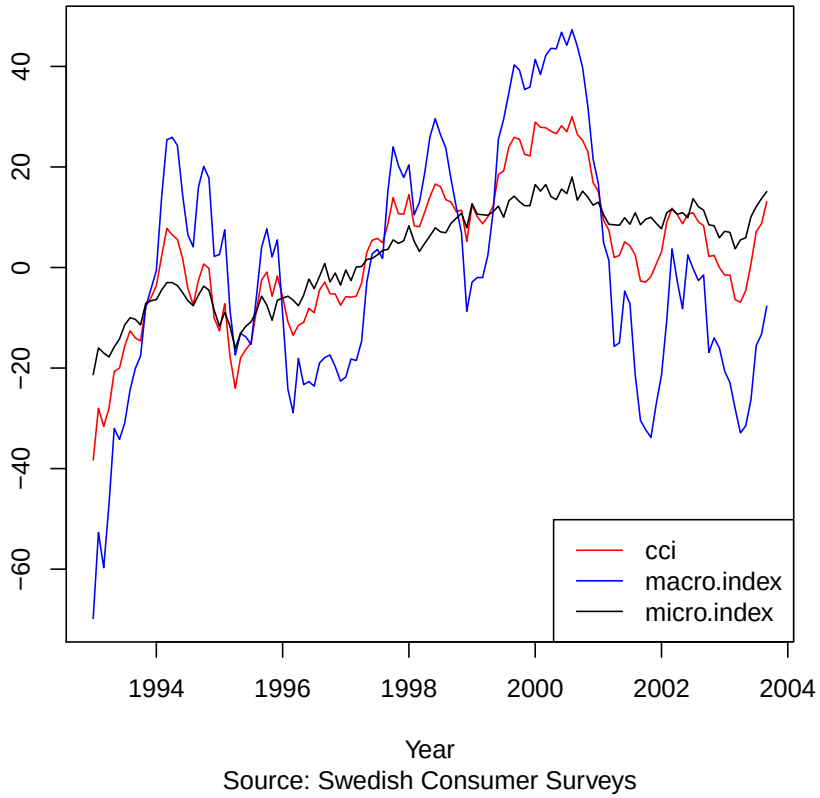
9.4 Plotting time series

Start by reading a time-series data set. For details see section 8.

Plot series in one diagram

```
> TITLE <- "Figure 2: Consumer confidence index in Sweden"
> SubTitle <- "Source: Swedish Consumer Surveys"
> X.LABEL <- "Year"
> COLORS = c("red", "blue", "black")
> ts.plot(macro, col = COLORS, main = TITLE, sub = SubTitle,
+     xlab = X.LABEL)
> legend("bottomright", legend = colnames(macro),
+     lty = 1, col = COLORS)
```

Figure 2: Consumer confidence index in Sweden



`<plot.ts(macro)>` plots series separately.

10 Writing functions

The syntax is: `myfunction <- function(x, a, ...) \{...\}` The arguments for a function are the variables used in the operations as specified in the body of the function i.e. the codes within { }. Once you have written a function and saved it, you can use this function to perform the operation as specified in { ...} by referring to your function and using the arguments relevant for the actual computation.

The following function computes the squared of mean of a variable. By defining the function `<ms>` we can write `<ms(x)>` instead of `<(mean(x))^2)>` every time we want to compute the square of mean for a variable <x>.

```
> ms <- function(x) {
+     (mean(x))^2
+ }
> a <- 1:100
> ms(a)

[1] 2550.25
```

The arguments of a function:

The following function has no arguments and prints the string of text, `<Welcome>`

```
> welc <- function() {
+     print("Welcome")
+ }
> welc()

[1] "Welcome"
```

This function takes an argument `x`. The arguments of the function must be supplied.

```
> myprog.no.default <- function(x) print(paste("I use",
+     x, "for statistical computation."))
```

If a default value is specified, the default value is assumed when no arguments are supplied.

```
> myprog <- function(x = "R") {
+     print(paste("I use", x, "for statistical computation."))
+ }
> myprog()

[1] "I use R for statistical computation."

> myprog("R and sometimes something else")

[1] "I use R and sometimes something else for statistical computation."
```

10.1 A function for computing Clustered Standard Errors

Here follows a function for computing clustered-Standard Errors. (See also the function `robcov` in the library `Design` discussed above.) The arguments are a data frame `<dat>`, a model formula`<f1>`, and the cluster variable `<cluster>`.

```
clustered.standard.errors <- function(dat,f1, cluster){
attach(dat, warn.conflicts = FALSE)
M <- length(unique(cluster))
N <- length(cluster)
K <- lm(f1)$rank
cl <- (M/(M-1))*((N-1)/(N-K))
X <- model.matrix(f1)
invXpX <- solve(t(X) %*% X)
ei <- resid(lm(f1))
uj <- as.matrix(aggregate(ei*X,list(cluster),FUN=sum)[-1])
sqrt(cl*diag(invXpX%*%t(uj)%*%uj%*%invXpX))  }
```

Notice that substituting the last line with
`sqrt( diag(invXpX %*%t(ei*X)%*%(X*ei)%*%invXpX)  )`
would yield White's standard errors.

11 Miscellaneous hints

Income Distribution	see ineq.
Logit	`<glm(formula, family=binomial(link=logit))>`. See `<?glm> & <?family>`.
Negative binomial	`<?negative.binomial or ?glm.nb>` in MASS, VR.
Poisson regression	`<glm(formula, family=poisson(link=log))>`. See `<?glm> & <?family>`.
Probit	`<glm(formula,family=binomial(link=probit))>`. See `<?glm> & <?family>`.
Simultaneous Equations	see sem, systemfit.
Time Series	see `<?ts>` tseries, urca and strucchange.
Tobit	`see <?tobin>` in survival.

12 Acknowledgements

I am grateful to Michael Lundholm , Lena Nekby and Achim Zeileis for helpful comments.

References

Peter Dalgaard. *Introductory Statistics with R*. Springer, 2nd edition, 2008. URL http://www.biostat.ku.dk/~pd/ISwR.html. ISBN 978-0-387-79053-4.

John Fox. *An R and S-Plus Companion to Applied Regression*. Sage Publications, Thousand Oaks, CA, USA, 2002. URL http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/index.html. ISBN 0-761-92279-2.

Christian Kleiber and Achim Zeileis. *Applied Econometrics with R*. Springer, New York, 2008. URL http://www.springer.com/978-0-387-77316-2. ISBN 978-0-387-77316-2.

Roger Koenker and Achim Zeileis. Reproducible econometric research (a critical review of the state of the art). Report 60, Department of Statistics and Mathematics, Wirtschaftsuniversität Wien, Research Report Series, 2007. URL http://www.econ.uiuc.edu/~roger/research/repro/.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. URL http://www.R-project.org. ISBN 3-900051-07-0.

William N. Venables and Brian D. Ripley. *Modern Applied Statistics with S. Fourth Edition*. Springer, New York, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4/. ISBN 0-387-95457-0.