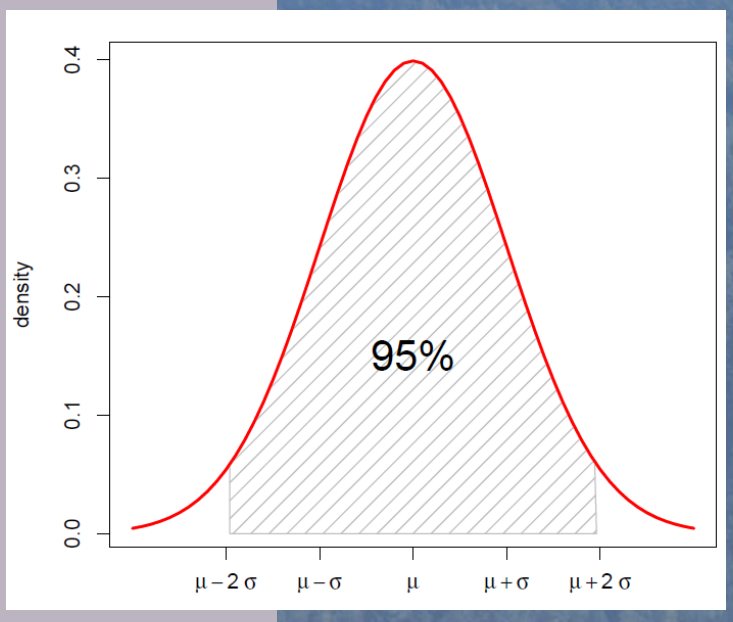
TECHNISCHE UNIVERSITÄT DRESDEN

Data Analysis with R
Selected Topics and Examples




```
x <- 1:10
y <- 2 + 0.5 * x + rnorm(x, sd = 0.5)
m <- lm(y ~ x)
plot(x, y, pch = 16)
abline(m, lwd = 2, col = "red")
```

Thomas Petzoldt

Data Analysis with R
Selected Topics and Examples

Thomas Petzoldt

October 6, 2014

—

This manual will be regularly updated,
more complete and corrected versions may be found on
`http://tu-dresden.de/Members/thomas.petzoldt/downloads`.
Suggestions and corrections are welcome.

—

Contents

Preliminary Notes

This tutorial is, similar to R, in permanent development. Suggestions and improvements are welcome, please use always the most recent version.

Due to this, it is explicitly forbidden to upload this tutorial to other than the original internet locations. Printing, copying and distribution on CDROM or flash drives is allowed, as long as these preliminary notes are not removed.

Parts of this tutorial are based on a German course script from the same author. I am grateful to Christof Bigler for comments on an earlier German version of this tutorial and to Jürgen Niedballa for help in translation of selected chapters.

However, both are not responsible for **my** mistakes, and suggestions and improvements are always welcome. Note also that all information provided herein comes without any warranty.

The translation is still preliminary and more chapters, sections, figures and examples may follow. The most recent version will always be available from `http://tu-dresden.de/Members/thomas.petzoldt/downloads`.

1 Introduction

This tutorial assumes preliminary knowledge of basic applied statistics from a bachelor degree in natural sciences or engineering and aims to extend this on a problem-oriented level.

This means that while some essential concepts are repeated, more serious gaps have to be filled up by self study. The main focus is to establish a certain degree of "statistical feeling", therefore statistical theory is mostly skipped and should be taken from textbooks for full understanding.

1.1 Further reading

A huge amount of statistical literature can be found on the market, and it is difficult to give recommendations, depending on existing knowledge and technical skills. Nevertheless, I want to give a few suggestions:

- **As a well-readable introduction:** Dalgaard, P., 2008: Introductory Statistics with R. Springer, New York, 2nd edition.
- **A very well understandable introduction into many fields of statistics, especially regression and time series analysis:** Kleiber, C. and Zeileis, A., 2008: Applied Econometrics with R, Springer, New York.
- **As an excellent introduction to R with strong emphasize to ANOVA methods:** Crawley, M. J., 2012: The R Book. Wiley.
- **A compendium about the R language and many fields of application:** Adler, J., 2010: R in a Nutshell. O'Reiley.
- **As comprehensive reference to many standard and also more specific methods with S-PLUS and R:**VENABLES, W. N. and B. D. RIPLEY, 2002: Modern Applied Statistics with S. Springer, New-York.
- **Advanced methods for ecological statistics** can be found in Zuur, A. F. et al., 2008: Mixed Models and Extensions in Ecology with R. Springer, New York.
- **and in general** many online documents about statistical data analysis with with R, see `www.r-project.org`.

1.2 Tasks of Statistics

It is sometimes common practice to apply statistical methods at the end of a study "to defend the reviewers", but it is definitely much better to employ statistics from the beginning for planning observations and experiments and for finding an optimal balance between measurement effort and intended results. In this context, statistics can help to:

1. Formulate hypotheses (descriptive and explorative statistics),
2. Plan observations and experiments (optimal experimental design, estimation of the sample size),
3. And finally to test hypotheses (test statistics and model selection methods).

Similarly, it is often distinguished between:

Descriptive research: Monitoring with the goal to "identify effects, relationships or correlations". The observational subject is not manipulated.

Experimental research: Test whether an anticipated effect can be reproduced in a controlled experiment:

- Manipulation of single conditions,
- Elimination of disturbances (constant experimental conditions),
- Experimental design as simple as possible.

Only experimental research is able to demonstrate causal relationships in a conclusive way. This may lead sometimes to disrespect against observational research, but this is not appropriate because experimental research always needs good observations for formulating its hypotheses.

2 An Introductory R Session

2.1 Program Start and Help System

The easiest way to learn "R" is the creative understanding and modification of given examples, the usage of R for solving practical problems and the diagnosis of the frequently occurring problems and error messages. Don't worry: error messages are a normal phenomenon in scientific computing and not an indication of a dysfunction of the computer or the human brain. The opposite is true, a certain amount of stress hormones helps to acquire permanent learning effects. Then, after a certain level of experience reading the official R-Documentation (An Introduction to R, VENABLES *et al.*, 2012) or any good R-book is strongly recommended.

The first sections of this "crash course" are intended to give an overview over some of the most important elements of R and an insight into a typical work flow, that may be useful for the first statistical analyses and as a starting point for self-education.

We begin our first session by starting **RStudio**, a platform independent interface that makes working with R easier. RStudio divides the screen into 3 (resp. 4) windows (called panes), where some of them have additional tabs to switch between different views.

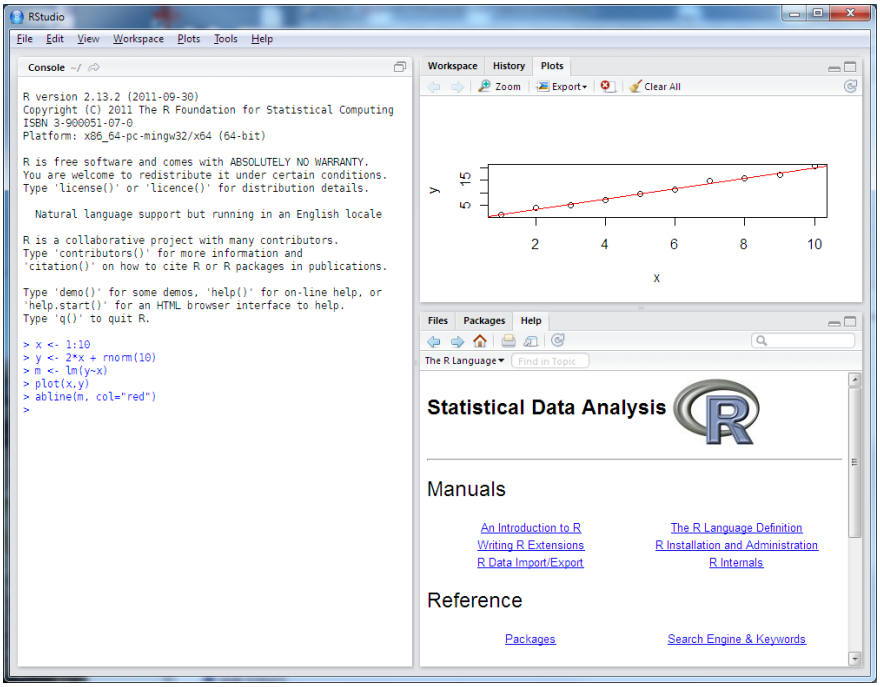


Figure 2.1: R Studio with 3 panes. Browse in the help files or use **File – New R Script** to open the the source code pane.

In a fresh RStudio session, one "Pane" should be the main help page of R. It is a good idea to browse a little bit around to get an impression about the amount and the typical style of the available help topics. The most important sections are "An Introduction to R", "Search Engine & Keywords", "Packages", the "Frequently Asked Questions" and possibly "R Data Import/Export".

We start now to explore the R-System itself.

2.2 First Steps

Entering an arithmetic expression like this:

```
2 + 4
```

shows that R can be used as a pocket calculator, that immediately outputs the result:

```
[1] 6
```

Instead of printing the result to the screen, it is also possible to save the result into a named **variable** using the assignment operator "<−".

```
a <- 2 + 4
```

It seems that nothing happens, but the result is now saved in the variable `a` that can be recalled at any time by entering the variable name alone:

```
a
```

```
[1] 6
```

Variable names in R start always with a character (or for special purposes a dot), followed by further characters, numerals, dots or underscores, where a distinction is made between small and capital letters, i.e. the variables `value` and `VALUE` can contain different data. A few character combinations are **reserved words** and cannot be used as variables: `break, for, function, if, in, next, repeat, while` and "`...`" (three dots). Other identifiers like `plot` can be re-defined, but this should be done with care to avoid unwanted confusion and side effects.

You may also notice, that the output of the example above had a leading `[1]`, which means that the line begins with the first element of `a`. This brings us to a very important feature of R that variables usually contain more than single values, but vectors, matrices, lists, etc.

The most basic data type is the vector, that can be filled with data by using the `c` (combine) function:

```
values <- c(2, 3, 5, 7, 8.3, 10)
values
```

```
[1]  2.0  3.0  5.0  7.0  8.3 10.0
```

To create a sequence of values, one can use the `:` (colon):

```
x <- 1:10
x
```

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

or, even more flexibly the `seq` function:

```
x <- seq(2, 4, 0.25)
x
```

```
[1] 2.00 2.25 2.50 2.75 3.00 3.25 3.50 3.75 4.00
```

Sequences of repeated equal values can be obtained with `rep`:

```
x <- rep(2, 4)
x
```

```
[1] 2 2 2 2
```

There are many ways to use these functions, try for example:

```
seq(0, 10)
seq(0, 10, by = 2)
seq(0, pi, length = 12)
rep(c(0, 1, 2, 4, 9), times = 5)
rep(c(0, 1, 2, 4, 9), each = 2)
rep(c(0, 1, 2, 4, 9), each = 2, times = 5)
```

Instead of accessing vectors as a whole, it is also possible to extract single elements, where the index of the requested data is itself a vector:

```
values[5]
```

```
[1] 8.3
```

```
values[2:4]
```

```
[1] 3 5 7
```

```
values[c(1, 3, 5)]
```

```
[1] 2.0 5.0 8.3
```

Sometimes, elements of a vector may have individual names, which makes it easy to access them:

```
named <- c(a = 1, b = 2.3, c = 4.5)
named
```

```
  a   b   c
1.0 2.3 4.5
```

```
named["a"]
```

```
a
1
```

In R (and in contrast to other languages like C) vector indices start with 1. Negative indices are also possible, but they have the special purpose to delete one or several elements:

```
values[-3]
```

```
[1]  2.0  3.0  7.0  8.3 10.0
```

It is also possible to extend a given vector by preceding or appending values with the combine function (`c`):

```
c(1, 1, values, 0, 0)
```

```
 [1]  1.0  1.0  2.0  3.0  5.0  7.0  8.3 10.0  0.0  0.0
```

The length of a vector can be determined with:

```
length(values)
```

```
[1] 6
```

and it is also possible to have empty vectors, i.e. vectors that exist, but do not contain any values. Here the keyword `NULL` means "nothing" in contrast to "0" (zero) that has length 1:

```
values <- NULL
values
```

```
NULL
```

```
length(values)
```

```
[1] 0
```

Such empty vectors are sometimes used as "Containers" for appending data step by step:

```
values <- NULL
values
```

```
NULL
```

```
length(values)
```

```
[1] 0
```

```
values <- c(values, 1)
values
```

```
[1] 1
```

```
values <- c(values, 1.34)
values
```

```
[1] 1.00 1.34
```

If a data element should be removed completely, this can be done using the `remove` function:

```
rm(values)
values
```

```
Error: Object "values" not found
```

The complete workspace can be deleted with `rm` (remove):

```
rm(list = ls(all = TRUE))
```

The R session can be closed by using the menu as usual or by entering:

```
> q()
```

Sometimes and depending of the configuration, R asks whether the "R workspace" should be saved to the hard disk. This may be useful for continuing work at a later time, but has the risk to clutter the workspace and to get irreproducible results at a later session, so it is recommended to say "No" for now, except if you exactly know why.

Later we will learn how to save only the data (and commands) that are needed.

2.3 Graphics

Now, we will see how to use R as a function plotter by drawing sine- or cosine functions within an interval between 0 to 10. First, we create a table of values for `x` and `y` and in order to get a smooth curve, it is reasonable to choose a small step size. As a rule of thumb I always recommend to use about 100...400 small steps as a good compromise between smoothness and memory requirements, so let's set the step size to 0.1:

```
x <- seq(0, 10, 0.1)
y <- sin(x)
plot(x, y)
```

Instead of plotting points, it is of course also possible to draw cotinuous lines. This is indicated by supplying an optional argument `type="l"`. **Important:** the symbol used here for `type` is the **small** letter "L" for "line" and not the – in printing very similar – numeral "1" (one)!

Note also that in R optional arguments can be given by using a "keyword = value" pair. This has the advantage that the order of arguments does not matter, because arguments are referenced by name:

```
plot(x, y, type = "l")
```

Now we want to add a cosine function with another color. This can be done with one of the function `lines` or `points`, for adding lines or points to an existing figure:

```
y1 <- cos(x)
lines(x, y1, col = "red")
```

With the help of `text` it is also possible to add arbitrary text, by specifying first the x- and y- coordinates and then the text:

```
x1 <- 1:10
text(x1, sin(x1), x1, col = "green")
```

Many options exist to modify the behavior of most graphics functions so the following specifies user-defined coordinate limits (`xlim, ylim`), axis labels and a heading (`xlab, ylab, main`).

```
plot(x, y, xlim = c(-10, 10), ylim = c(-2, 2),
     xlab = "x-Values", ylab = "y-Values", main = "Example Graphics")
```

The above example is a rather long command and may not fit on a single line. In such cases, R display a + (plus sign) to indicate that a command must be continued, e.g. because a closing parenthesis or a closing quote are still missing. Such a + at the beginning of a line is an automatic "prompt" similar to the ordinary ">" prompt and must never be typed in manually.

In contrast to the long line continuation prompt, it is also possible to write several commands on one line, separated by a semi-colon ";".

Finally, a number (or hash) symbol # means that the following part of the line is a comment and should therefore be ignored by R.

In order to explore the wealth of graphical functions, you may now have a more extensive look into the online help, especially regarding `?plot` or `?plot.default`, and you should experiment a little bit with different plotting parameters, like `lty`, `pch`, `lwd`, `type`, `log` etc. R contains uncountable possibilities to get full control over the style and content of your graphics, e.g. with user-specified axes (`axis`), legends (`legend`) or user-defined lines and areas (`abline`, `rect`, `polygon`). The general style of figures like (font size, margins, line width) can be influenced with the `par()` function.

In addition, R and its packages contain numerous "high level"-graphics functions for specific purposes. To demonstrate a few, we first generate a data set with normally distributed random numbers (mean 0, standard deviation 1), then we plot them and create a histogram. Here, the function `par(mfrow=c(2,2))` divides the plotting area into 2 rows and 2 columns for showing 4 separate figures:

```
par(mfrow = c(2, 2))
x <- rnorm(100)
plot(x)
hist(x)
```

Now, we add a so-called *normal probability plot* and a second histogram with relative frequencies together with the bell-shaped density curve of the standard normal distribution. The optional argument `probability = TRUE` makes sure that the histogram has the same scaling as the density function, so that both can be overlayed:

```
qqnorm(x)
qqline(x, col="red")
hist(x, probability = TRUE)
xx <- seq(-3, 3, 0.1)
lines(xx, dnorm(xx, 0, 1),col = "red")
```

Here it may also be a good chance to do a little bit summary statistics like: z.B. `mean(x)`, `var(x)`, `sd(x)`, `range(x)`, `summary(x)`, `min(x)`, `max(x)`, ...

Or we may consider to test if the generated random numbers `x` are really normal distributed by using the Shapiro-Wilks-W-Test:

```
x <- rnorm(100)
shapiro.test(x)

        Shapiro-Wilk normality test
```

```
data:  x
W = 0.9936, p-value = 0.9218
```

A p-value bigger than 0.05 tells us that the test has no objections against normal distribution of the data. The concrete results may differ, because `x` contains random numbers, so it makes sense to repeat this several times. It can be also useful compare these normally distributed random numbers generated with `rnorm` with uniformly distributed random numbers generated with `runif`:

```
par(mfrow=c(2,2))
y <- runif(100)
plot(y)
hist(y)
qqnorm(y)
qqline(y, col="red")
mean(y)
var(y)
min(y)
max(y)
hist(y, probability=TRUE)
yy <- seq(min(y), max(y), length = 50)
lines(yy, dnorm(yy, mean(y), sd(y)), col = "red")
shapiro.test(y)
```

At the end, we compare the pattern of both data sets with box-and-whisker plots:

```
par(mfrow=c(1, 1))
boxplot(x, y)
```

Exercise: Repeat this example with new random numbers and vary sample size (`n`), mean value (`mean`) and standard deviation (`sd`) for `rnorm`, and use different `min` and `max` for `runif`. Consult the help pages for an explanation of these arguments.

Table 2.1: An overview over some of the most important object types (classes) in R

Object	Possible modes	Mixing of modes possible	Columns have same length	Remark
vector	numeric, character, complex, logical	no	–	
factor	numeric, character	no	–	categories like, "control" or "treatment", very important for ANOVA and graphics but sometimes confusing because of "coding"
array	numeric, character, complex, logical	no	yes	multi-dimensional data structure (rows, columns, layers, ...)
matrix	numeric, character, complex, logical	no	yes	same like array with only two dimensions, can always be converted into data.frame
data.frame	numeric, character, complex, logical	yes	yes	a typical table, can be converted into a list or a matrix
list	numeric, character, complex, logical, function, expression, formula	yes	no	most flexible type, can be similar to data.frame (if all columns have same length) or nested tree structure

2.4 Data Structures: Basic R-Objects

In addition to vectors, R contains several other types of objects for saving data, Table 2.1 lists some of the most important. This is only a selection and base R as well as contributed packages contain many more classes. It is also possible to define user-defined classes.

All objects have the two built-in attributes `mode` (data type) and `length` (number of data in the object). Under specific circumstances some of these data types can be converted into each other, e.g. by using functions like `as.matrix`, `as.data.frame` etc.

Conversion of **factors** into other data types should be done with care, because contents are encoded as `levels`. The following example shows how to convert factors to numeric values properly:

```
x <- c(2, 4, 6, 5, 8)
f <- as.factor(x)
as.numeric(f)                # wrong !!!
```

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

```
as.numeric(as.character(f)) # correct
```

```
[1] 2 4 6 5 8
```

```
as.numeric(levels(f))[f]    # even better
```

```
[1] 2 4 6 5 8
```

This type of factor coding is not specific to R and appears also in other statistics packages.

2.5 Entering Data

Several different methods exist to input data into R. The most important are extensively explained in a special manual "R Data Import/Export" and we want to show only a selection here:

1. direct input in the R code,
2. input via the clipboard,
3. input from a text file.

Other methods are direct data base access, import of data from other statistics packages like SPSS, SAS, Stata or Minitab (`library(foreign)`), reading of GIS-Shapefiles (`library(shapefiles)`), and even sound files or pictures.

2.5.1 Direct Input

We used this method already when creating vectors with the `c` (combine)-Function:

```
x <- c(1, 2, 5, 7, 3, 4, 5, 8)
```

In the same way it is possible to create other data types like data frames:

```
dat <- data.frame(f = c("a", "a", "a", "b", "b", "b"),
                  x = c(1,   4,   3,   3,   5,   7)
       )
```

It is also possible to input a data frame conveniently with R Studio. For this purpose:

1. Select the "Files" Tab and browse through the folder structure of your computer until you find the text file with the data, e.g. a file downloaded from the internet:
2. From the menu select Workspace – Import Data Set – From Text File.
3. Select the requested file and a window will show up, where you can enter options like the name of the variable the data are to be assigned to, or options whether a header with the variable names is present.

2.5.2 Copy and Paste from the Clipboard

R is able to read data directly from the clipboard that were pasted from an editor or a spreadsheet program like Excel or LibreOffice. Let's for example create a spreadsheet table from the following example that contains some data from a lake area in north-eastern Germany (data source: KOSCHEL *et al.*, 1987):

We now select the data and copy them to the clipboard (right mouse, copy), then we change to R and retrieve the content of the clipboard with the following command line:

```
dat <- read.table("clipboard", header=TRUE)
```

The character argument `"clipboard"` is the file name for the data source, `header=TRUE` tells us that the first line contains the variable names. In some countries that have the comma and not the dot as a decimal separator, an additional argument `dec = ","` may be required.

Now, the data are saved in the data frame `dat` and it is possible to access them as usual:

Table 2.2: Morphometrical and chemical properties of selected lakes (S=Stechlinsee, NN=Nehmitzsee Nord, NS=Nehmitzsee Süd, BL=Breiter Luzin, SL = Schmaler Luzin, DA = Dagowsee, HS = Feldberger Haussee; z=mean depth (m), t=theoretical retention time (a), P=phosphorus concentration ($\mu g l^{-1}$), N=nitrogen concentration ($mg l^{-1}$), Chl=chlorophyll concentration ($\mu g l^{-1}$), PP=annual primary production ($g C m^{-2} a^{-1}$), SD = secchi depth (m)), data: KOSCHEL *et al.* (1987)

No	Lake	z	t	P	N	Chl	PP	SD
1	S	23.70	40.00	2.50	0.20	0.70	95.00	8.40
2	NN	5.90	10.00	2.00	0.20	1.10	140.00	7.40
3	NS	7.10	10.00	2.50	0.10	0.90	145.00	6.50
4	BL	25.20	17.00	50.00	0.10	6.10	210.00	3.80
5	SL	7.80	2.00	30.00	0.10	4.70	200.00	3.70
6	DA	5.00	4.00	100.00	0.50	14.90	250.00	1.90
7	HS	6.30	4.00	1150.00	0.75	17.50	420.00	1.60

```
dat
colMeans(dat[2:8])
boxplot(dat[2:8])
```

2.5.3 Reading from a Textfile

Reading from the clipboard sounds attractive, but it has a big disadvantage because it needs several manual steps and cannot be automated. Therefore, it is much better to first save the data to a text file on the hard disk before using read table. In the same way it is also possible to read text files directly from the internet.

Sometimes, it is necessary to know the full path to the data set, but it is also possible to set the working directory of R to the data directory. This can be done with the function `setwd` or (even better!) by using the respective menu functions from the R Gui or from Tinn-R resp. RStudio.

```
setwd("x:/gaeste/praktik/stat/")
mydata <- read.table("hall.txt", header=TRUE)
```

Note that we always use the ordinary slash and not the backslash, even on Windows.

It is also possible to locate and then select the file with a dialog window:

```
mydata <- read.table(file.choose(), header=TRUE)
```

but this has, again, the disadvantage that it cannot be automated.

If the data are available on an internet server, it can be read directly from there:

```
mydata <- read.table("http://www.simecol.de/data/hall.txt", header=TRUE)
```

Now, we are ready to inspect the content of this new variable `mydata`:

```
View(mydata)
```

Function `View` opens a table view, because `mydata` is a data frame. This is similar to `fix`, but in contrast to this it is not possible to change the values and the View window can be left open (it is a so-called non-modal window). The data set `hall.txt` contains growth curves from an experiment with Daphnia (water flea) that were taken from a figure of a publication (HALL, 1964), where growth is dependent on time (`day`), temperature (`temp`) and food concentration that was measured in the nowadays completely outdated turbidity unit "Klett", `klett`), but it is not of importance for our example.

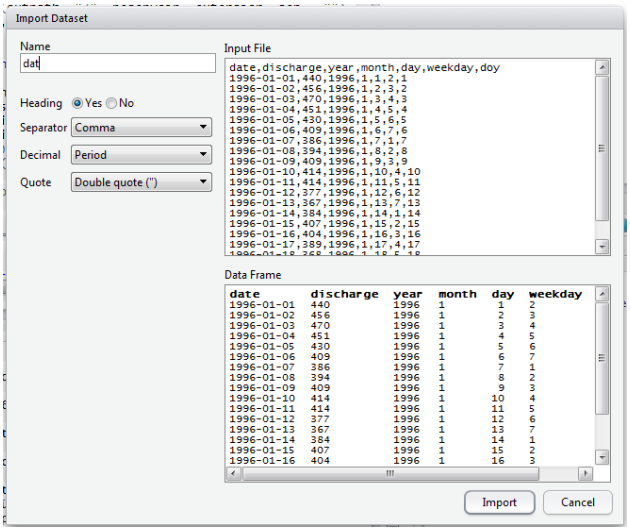


Figure 2.2: Import Dataset Wizard of RStudio.

2.6 The "Import Dataset" Wizard of RStudio

RStudio contains a nice feature that makes importing of data more convenient. Essentially, this "Import Dataset" wizard helps us to construct the correct `read.table` od `read.csv` function interactively. The upper left window in Figure 2.2 shows the original inpuit file and the lower window indicates whether the data frame was correctly recognized. It is possible to try different options until a satisfying result is obtained.

Hint: Do not forget to set a **Name** for the resulting data frame (e.g. `dat`), otherwise R uses the file name.

2.7 Working with Dataframes

For large tables it is often not very useful to display the full content, so it is much better to use the function `str` (structure) that gives a compact overview over type, size and content of a variable:

```
str(mydata)
```

This function is very universal and also suitable for complicated object types like lists. Of course, there are many more possibilities for inspecting the content of a variable:

```
names(mydata)
mode(mydata)
length(mydata)
```

and sometimes even:

```
plot(mydata)
```

Single columns of a data frame can be accessed by using indices (with `[]`) similar to a vector or a matrix or by using the column name and the `$`) operator:

```
mean(mydata[4])
mean(mydata$leng)
mean(mydata["leng"])
mean(mydata[["leng"]])
plot(mydata$day, mydata$leng)
```

Note the difference of the output of the `[]` and the `[[]]` version. The difference is as follows: single brackets return a data frame with one column, but double square brackets return the content of the column, i.e. a vector.

The `$`-style can be abbreviated by using the `attach` function, but `attach` is a relict from rather old times and many people recommend not to use this anymore. The reason is, that attached data must always be detached (`detach`) after use, otherwise this may lead to very strange errors.

```
attach(mydata)
plot(day, leng)
detach(mydata)
```

To be on the safe side, it may be a good idea to use `detach` repeatedly until an error message confirms us that there is nothing else that can be detached.

It is even better to use another function `width`, that opens the data frame only temporarily:

```
with(mydata, plot(day, leng))
```

or if you want to combine several commands

```
with(mydata, {
  print(mean(leng))
  print(sd(leng))
})
```

A very powerful feature of R is the use of logical vectors as "indices", with similar results like data base queries. A prerequisite for this is that all vectors have the same length.

```
par(mfrow=c(1, 2))
with(mydata, {
  plot(day[temp == 20], leng[temp == 20])
  plot(day[temp == 20 & klett == 16], leng[temp == 20 & klett == 16])
})
```

A logical comparison requires always a double "==". Logical operations like `&` (and) and `|` (or) are also possible. Note that **and** has always precedence before **or**, except this is changed with parenthesis.

A subset of a data frame can also be extracted with the `subset` function:

```
twentydegrees <- subset(mydata, mydata$temp == 20)
View(twentydegrees)
```

Like in the example before, the condition argument allows also logical expressions with `&` (and) and `|` (or).

At the end of this section we show how to convert a data frame into a matrix and how to access single elements in matrix-like manner:

```
mydata   <- read.table("hall.txt", head = TRUE)
mymatrix <- as.matrix(mydata)
```

The element from the 2nd row and the 4th column can be selected with:

```
mymatrix[2, 4]
```

```
[1] 0.5227271
```

the complete 5th row with:

```
mymatrix[5, ]
```

```
    klett       temp        day       leng
0.2500000 20.0000000  9.2000000  0.9431816
```

and rows 5:10 of the 4th column (`leng`) with:

```
mymatrix[5:10, 4]
```

```
        5         6         7         8         9        10
0.9431816 0.9602271 1.1250000 1.2215910 1.3068180 1.3920450
```

Additional methods for working with matrices, data frames and lists can be found in R textbooks or in the official R documentation.

Mean values for factor combinations

The last examples are intended to demonstrate how powerful a single line can be in R. Function `aggregate` can be used to compute statistics (e.g. mean values) depending on given criteria. The first argument of the function is a data frame containing numeric data, the second argument a list (or data frame) of criteria (as factors) and the third argument a function that will be applied to all possible factor combinations, e.g., `mean`, `median`, `sd`, `max` etc.

```
aggregate(mydata, list(klett = mydata$klett, temp = mydata$temp), mean)
```

or, because it is not meaningful here to calculate mean values for temperature, Klett-units and time:

```
aggregate(list(leng = mydata$leng),
          list(klett = mydata$klett, temp = mydata$temp), mean)
```

Categorial Box Plots

Categorical boxplots can be obtained by using the so-called "formula-interface". The variable at the left hand side is the dependent variable, while independent factors used for classification are written at the right hand side. The formula is then read "leng versus klett and temp":

```
boxplot(leng ~ klett + temp, data = mydata)
```

Many, but not all R-Functions support this formula interface that can be read as: `leng` *versus* `klett` *and* `temp`.

An even more powerful and compact visualization can be obtained with the lattice `lattice`-package:

```
library(lattice)
xyplot(leng ~ day|temp * klett, data = mydata)
```

Though the lattice package has a rather complex syntax, it is also very flexible and powerful, so the time to learn it is worth the effort.

However, like in all similar cases, one may reach its goal also step by step with basic R functions only:

```
attach(mydata)
group1 <- leng[klett == 0.25 & temp == 11]
group2 <- leng[klett == 1    & temp == 11]
group3 <- leng[klett == 16   & temp == 11]
# ....
boxplot(group1, group2, group3, names=c("0.25/11", "1/11", "16/11"))
detach(mydata)
```

2.8 Output of Results

The most simple method to save outputs from R is to copy it directly from the R console to any other program (e.g. LibreOffice, Microsoft Word or Powerpoint) via the Clipboard.

In Windows is is also possible to print the console output (File – Print) or to save it to a text file (File – Save to File . . .).

A third possibility available on all systems is redirection of the complete screen output to a logfile by using the `sink` function:

```
sink("logfile.txt")
```

the following output is now directed to the text file and does not appear on the screen until the redirection is closed with a call of `sink` without arguments:

```
sink()
```

Because one cannot see what happens during `sink` is active it is recommended to use the optional `split` argument, so that the output appears in **both** places, the screen and the logfile:

```
sink("logfile.txt", split = TRUE)
x <- 1:10
y <- 2*x + rnorm(x)
summary(lm(y ~ x))
sink()
```

In addition to these basic functions R has a wealth of possibilities to save output and data for later use in reports and presentations. All of them are of course documented in the online help, e.g. `print`, `print.table`, `cat`, `pdf`, `png`, etc. or within a specific "R-Data Import/Export Manual". The add-on package `xtable` contains functions for creating LaTeX or HTML-tables while full HTML output is supported by the `R2HTML` package.

2.9 Quitting an R-Session

The R-window can be closed as usual with the menu or by entering `q()` (quit):

```
q()
```

Depending on the configuration, we may now be asked whether we want to "Save workspace image" and answering "Yes" would force to save all data from the R-Workspace into a file `.Rdata`, so that all data will be automatically and immediately available in the next session, given that R is started in the same working directory. Alternatively it is also possible to save or restore an R-Workspace manually into a file (Save Workspace, Load Workspace).

Exercises

1. Explore different possibilities to plot the Hall-Data set. Draw one figure for each temperature level and distinguish food concentration by using different colors, plot symbols or line types. Make use of the annotation options for adding axis labels and main titles.
2. Read the data set `lakeprofile.txt` (Data from a students field course at IGB Berlin from 13.10.1992). Plot vertical profiles for all variables.

   Note: Measurement units may also use Greek letters or super- and subscripts by using the `expression`-function. This looks professional, but works rather technical so that we should postpone it to a later time.
3. R contains lots of data sets for exploring its graphical and statistical functions and that can be activated by using the `data` function, e.g. `data(iris)`. Use this data set and find appropriate ways for visualization. A description of the data set can be found as usual in the help file `?iris`.

2.10 Loops and Conditional Execution

2.10.1 Compound Statements

In R it is possible to group several statements together by enclosing them into curly braces. As an example, function `system.time`, can be used to calculate the CPU time that is used by a block of successive function calls, e.g. generating and plotting a series of random data:

```
system.time({
  x <- rnorm(10000)
  plot(x)
})
```

```
   user  system elapsed
   0.02    0.10    0.12
```

This means, that on the test system (Intel i7 PC with 3.0 GHz), total CPU time was 0.12 seconds, where computation time was 0.02s and the rest was used by system functions, most likely the graphics.

2.10.2 Loops

In contrast to other programming languages, loops are rarely needed in R. Being a vectorized language, most R functions work on bigger data structures like vectors and matrices by default. However, loops are sometimes unavoidable.

Often, a `for`-loop can be employed, that has the following syntax:

```
for (name in vector) statement
```

where `name` is called a "loop variable", `vector` can be either a vector or a list and `statement` can be a single R expression or a block of statements in a compound statement.

A loop is repeated as many times as there are elements in the vector. These elements are assigned one after each other to the loop variable. So the following loop:

```
for (i in 1:10) {
  print(i)
}
```

prints the numbers from 1 to 10 to the screen.

Loops can be nested, and it is very common to use indentation to improve readability. Proper indentation[1] should always be used for your own work and even in the class room, because this would help your colleagues and supervisors to see what your code does and also to eliminate errors.

The following example of a nested `for`-loop plots 9 figures for all temperature and food combinations of the HALL (1964) data set:

[1] I recommend 2 spaces per indentation level.

```
halldata<-read.table("http://www.simecol.de/data/hall.txt", header = TRUE)
attach(halldata)
par(mfrow=c(3, 3))
for (klett.i in c(0.25, 1, 16)) {
  for (temp.i in c(11, 20, 25)) {
    dat <- subset(halldata,
      halldata$klett == klett.i & halldata$temp == temp.i)
    plot(dat$day, dat$leng)
  }
}
detach(halldata)
```

In addition to `for`, R contains a full set of loops (cf. `repeat` and `while`), and additional control statements (`next`, `break`) to modify their execution.

2.10.3 Conditional execution

Conditional execution (also called branching) can be used to handle distinct cases or exceptions. The basic syntax is:

`if (`expr1`)` expr2 `else` expr3

where is a logical expression that can be TRUE or FALSE, `expr2` is an expression that is called in the TRUE case and `expr3` otherwise.

In the example before, no data have been available for the combination of `klett==0.25` and `temp==11`, so the figure contains a broken plot for this case. By means of an `if` it would be possible to handle this case as an exception:

```
halldata<-read.table("hall.txt", sep = " ", header = TRUE)
par(mfrow=c(3, 3))
with(halldata, {
  for (klett.i in c(0.25, 1, 16)){
    for (temp.i in c(11, 20, 25)) {
      dat <- subset(halldata,
                    klett == klett.i & temp == temp.i)
      if (nrow(dat) == 0) {                                    # no data
        plot(1, 1, axes = FALSE, type="n", xlab = "", ylab = "") # empty plot
        box()                                                  # rectangle
      } else {
        plot(dat$day, dat$leng)
      }
    }
  }
})
```

2.10.4 Vectorised ifelse

In many cases it is possible (and easier) to use the vectorized `ifelse`-function instead of branching with `if`. The following example shows how to replace all zero values by a small value (10^{-6}). In case of `x==0`,

a value of `1e-6` is returned, otherwise the original value. The final result is then saved back to `x`:

```
x <- c(0, 0, 1, 3, 4, 2, 1, 7, 0, 2)
x <- ifelse(x == 0, 1e-6, x)
x
```

```
[1] 1e-06 1e-06 1e+00 3e+00 4e+00 2e+00 1e+00 7e+00 1e-06 2e+00
```

2.11 User-defined functions

It is possible to extend the functionality of R with user-defined functions. Such functions are very handy, if one wants to encapsulate things that are repeatedly needed or to simplify complicated algorithms. The syntax of a user-defined function is:

```
name <- function(arg1, arg2, arg3, ... ) expression
```

where `name` is the name of the function (an arbitrarily selected valid variable name), `arg1, arg2, ...` are the arguments of the function and `expression` is a block of statements that is to be executed within the function. Most functions have a so-called *return value* that is returned at their end, which can of course be a list or other data structure that contains multiple values. The following example from "An Introduction to R" implements the classical two-sample t-test:

```
twosam <- function(y1, y2) {
  n1 <- length(y1)
  n2 <- length(y2)
  yb1 <- mean(y1)
  yb2 <- mean(y2)
  s1 <- var(y1)
  s2 <- var(y2)
  s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
  tst <- (yb1 - yb2)/sqrt(s * (1/n1 + 1/n2))
  tst
}
```

We can now compare this with R's built-in function for the t-test:

```
 x <- c(2, 3, 4, 5, 8)
 y <- c(1, 3, 5, 9, 9)
 twosam(x, y)
```

```
[1] -0.5255883
```

```
 t.test(x, y, var.equal = TRUE) # built-in t-test function
```

```
        Two Sample t-test

data:  x and y
t = -0.5256, df = 8, p-value = 0.6134
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
```

```
 -5.387472  3.387472
sample estimates:
mean of x mean of y
      4.4       5.4
```

We see that the built-in t-test contains additional functionality, e.g. the computation of p-values.

We see also, that the names of the variables that are passed to the function call (here `x` and `y`) do not necessarily need to be the same as in the function definition (here `y1` and `y2`), because the assignment of the function arguments is just defined by their order. This is a very important feature that allows to write universal functions (sub-routines) that can be used in different contexts without caring too much about their internal details.

Another very important property of functions is, that their internal variables are only valid within the local scope of that function, so that "outer" variables with overlapping names are not overwritten. As an example, calling `yb1` outside if `twosam` would just give an error:

```
yb1
```

```
Error: Object "yb1" not found
```

because `yb1` is not known outside of that function.

2.11.1 Debugging

For debugging purposes, i.e. if we suspect that something is wrong, it can be necessary to inspect values of internal variables. For such cases, it would be possible to output internal variables with `print` or, as an alternative, to switch the debug mode for this function on with `debug(twosam)`. This mode can be switched of with `undebug(twosam)`.

R contains many additional possibilities, e.g. usage of optional named arguments with default values, the ellipsis (`...`-argument, three dots) closures, object orientation or linking to C- or Fortran routines. Details about this can be found in "An Introduction to R" and "Writing R Extensions", that are both part of the official R documentation.

2.11.2 Exercises

1. Implement and test a user-defined function for the exponential and the logistic population growth:

$$N_t = N_0 \cdot e^{r \cdot t}$$
$$N_t = \frac{K}{1 + (\frac{K}{N_0} - 1) \cdot e^{-r \cdot t}}$$

2. Develop (or find) a function `circle`, that draws circles to the screen. Hint: This function is based on the sine and cosine functions.

3 Basic Principles and Terminology

Before we can start with some practical exercises, we should first clarify some of the most important concepts and a little bit statistical terminology.

3.1 The principle of parsimony

The principle of parsimony, sometimes also called "Occams razor" is attributed to an English philosopher from the 14th century who stated that "simpler explanations are, other things being equal, generally better than more complex ones".

In relation to statistical analysis and modeling this implies that (CRAWLEY, 2002):

- "models should have as few parameters as possible
- linear models should be preferred to non-linear models
- experiments relying on few assumptions should be preferred to those relying on many
- models should be pared down until they are minimal adequate
- simple explanations should be preferred to complex explanation"

This principle is one of the most important fundamentals, not only in statistics but also in science in general. However, over-simplification has to be avoided as well, especially in complex fields like ecology.

3.2 Types of variables

Variables are all these things that are more or less directly measured or experimentally manipulated, e.g phosphorus concentration in a lake, air temperature, or abundance of animals. In contrast to this, **parameters** are quantities that are estimated by using a particular (statistical) model, for example mean values, standard deviation or the slope of a linear model.

Independent variables (*explanation or explanatory variables*) are variables that are intentionally manipulated or that are assumed to result from external processes. **Dependend variables** (*response or explained variables*) are the variables that we are interested in and/or that form the resulting outcome of an observation or an experiment.

It is also necessary to distinguish different types of scales, because the scale determines which kind of analysis or test can be done with a particular variable:

Binary or boolean variables can have exactly one of two alternative states: true or false, one or zero, present or absent.

Nominal variables (or factors) can be characterized by names, e.g. "control", "treatment 1", "treatment 2" or "red", "green", "blue". By definition, nominal variables do not have any natural order.

Ordinal variables (ranks, ordered factors) do have an order, but not a defined distance in between. An example are the trophic states of a lake, e.g. (oligotrophic, mesotrophic, eutrophic, polytrophic, hypertrophic). However, if we add a state that does not fit into this order like an acidified (i.e. dystrophic) lake, then the whole variable becomes nominal.

Metric variables can be measured continuously and two sub-types can be distinguished:

Interval scale Here, differences make sense, but ratios are undefined. As an example one can say that a temperature of 20°C is 10 degrees warmer than 10°C, but it does not make sense to say that it is double. If you don't believe this, then please tell what is the ratio between 10°C and -5°C?

Ratio scale: Variables with a ratio scale have an absolute zero, so that ratios make sense. A tree with 2 m has double the hight of a tree with 1 m.

The above order tells us also something about the value or quality of the scales, ascending from lower to higher, because "higher scales" contain more information than "lower scales". Furthermore, it is always possible to transform a variable from a higher scale to a lower, so that more possibilities are available for their analysis.

3.3 Probability

According to the classical definition, probability p is the chance of a specific event, i.e. the number of events we are interested in, divided by the total number of events. In a die roll, for example, the probability of any certain number is 1/6 because the die has 6 sides and each number occurs only one time.

Unfortunately, this classical definition is not applicable to non-countable populations, because either the denominator or the numerator (or both) may be undefined or infinite. What is, for example, the probability that the height of a tree is 12.45 m?

To solve this, mathematicians use an **axiomatic** definition of probability:

Axiom I: $0 \leq p \leq 1$

Axiom II: impossible events have $p = 0$, certain events have $p = 1$

Axiom III: for events A and B, that exclude each other, i.e. in set theory $A \bigcap B = \emptyset$ holds: $p(A \bigcup B) = p(A) + p(B)$

3.4 Sample and Population

The objects, from which we have measurements or observations form a **sample**. In contrast to this, a **population** is the set of all objects that had the same chance to become part of the sample. This means that the population is defined by the way how samples are taken, i.e. how **representative** our sample is for our (intended) observation object.

In principle, there are two different sampling strategies:

Random sampling means that the individual objects for measurement are selected at random from the population (examples: random selection of sampling sites by means of numbered grid cells; random placement of experimental units in a climate chamber, random order of treatments in time).

Stratified sampling requires that the population is divided into sub-populations (strata). These are separately sampled (at random!) and the population characteristics are estimated by using weighted mean values. Therefore, it is essential, to have valid information about the size of the strata to derive the weighting coefficients. Examples are election forecasts, volumetric mean for a water body from measurements for different layers, gut content of fish estimated from size classes. The advantage of stratified sampling is to get better estimates from smaller samples, but this holds only if the weight coefficients are correct!

By convention, the "true" but unknown parameters of a population are symbolized with Greek letters (μ, σ, γ, α, β); the calculated parameters ($\bar{x}$, s, $r^2 \ldots$) are called "estimates". A single value x_i of a random variable X can also be assumed to consist of an expectation value $\mathbf{E}(X)$ of the random variable and an individual error ε_i. The expectation value (e.g. a mean) of this error term is zero:

$$x_i = \mathbf{E}(X) + \varepsilon_i \tag{3.1}$$

$$\mathbf{E}(\varepsilon) = 0 \tag{3.2}$$

3.5 Statistical Tests

Statistical tests are employed for testing hypotheses about data, e.g. specific properties of distributions or their parameters. The basic idea is to estimate probabilities for a hypothesis about the population from a sample.

Effect size and significance

In statistical testing **significance** has to be clearly distinguished from **effect size**. The effect size Δ measures the size of an observed effect like the difference between two mean values ($\bar{x}_1 - \bar{x}_2$) or the size of a correlation coefficient (r^2). Even more important is the **relative effect size**, the ratio between an effect and random error or the signal-noise ratio that is often represented as the ratio between the effect (e.g. difference between mean values) and the standard deviation:

$$\delta = \frac{\bar{\mu}_1 - \bar{\mu}_2}{\sigma} = \frac{\Delta}{\sigma}$$

In contrast to this, **significance** means that an effect really exists with a certain probability and that it is unlikely a result of random fluctuations alone.

For testing whether an effect is significant or not, it is important to formulate clear hypotheses in terms of statistical language, a so-called Null hypothesis (H_0) and one or more alternative Hypotheses (H_a):

H_0: Null hypothesis that two populations are not different with respect to a certain parameter or property. It is assumed, that an observed effect is purely random and that the true effect is zero.

H_a: The alternative hypothesis (experimental hypothesis) claims that a specific effect exists. An alternative hypothesis is never completely true or "proven". The acceptance of H_A means only than H_0 is unlikely.

Table 3.1: α and β errors

	Reality	
Decision of the test	H_0 true	H_0 false
H_0 retained	true, $1-\alpha$	β error
H_0 rejected	α error	correct decision with power $1-\beta$

Statistical tests can only test for differences between samples, not for equality. This means that H_0 is formulated to be rejected and that it is impossible to test if two populations are equal – for principal reasons. If a test cannot reject "the null", it means only that an observed effect can also be explained as a result of random variability or error and that a potential effect was too small to be "significantly" detected, given the available sample size.

Note: Not significant does not necessarily mean "no effect", it means "no effect or sample size too small".

Whether an effect is significant or not is determined by comparing the **p-value** of a test with a pre-defined critical value, the significance level α (or probability of error). Here, the p-value is an estimate for the probability that the null hypothesis is wrong and on the other hand, α is the amount of **false positives**, i.e. wrong rejections of H_0 that we tolerate.

To sum up, there are two possibilities for wrong decisions (cf. Table 3.1):

1. H_0 falsely rejected (error of the first kind or α error),
2. H_0 falsely retained (error of the second kind or β error).

It is common convention to use $\alpha = 0.05$ as the critical value in many sciences, but any other small value (e.g. 0.01 or 0.1) would be possible as well. The essential thing, however, is that this value has to be defined **before** doing the test and should not be adjusted afterwards.

In contrast to this, β is often unknown. It depends on the the relative effect size, the sample size and the power of a certain test. It can be estimated before an experiment by using power analysis methods, or, even better β is set to a certain value, e.g. 0.2 and then power analysis is used to determine the required sample size.

3.6 The model selection approach

Significance testing is one of the most important concepts in statistics, but it is not the only one. In contrast of testing whether a certain effect is significant or not, one can also test which of several candidate models is more appropriate to explain a given data set. This is a direct application of the principle of parsimony, but there are essentially two questions:

1. What is the best model?
2. What is the most parsimonious model?

We may be tempted to answer the first question by just using a model with the best fit, for example with the lowest mean square error or the highest R^2, but such an approach would lead to very complex, overcomplicated models because a complex model with many parameters has much more flexibility to fit a complicated data set. Therefore, what we need is not the model that maximally fits the data. We need the model with the best compromise between goodness of fit and model complexity, i.e. the smallest model that fits the data reasonably well.

This approach is called the model selection technique, an essential concept of modern statistics that sometimes supersedes p-value based testing.

3.7 Log likelihood and AIC/BIC

Two measures are needed to measure goodness of fit on one side and model size on the other, which are then combined to so-called information theory-based measures like AIC (Akaike Information Criterion) and BIC (Bayes Information Criterion). Here, goodness of fit is measured by means of the **log likelihood**, where likelihood is a measure that tells us how good a certain model explains an observed set of data. Likelihood is related to probability, but in contrast to probability where we know that the maximum value is one, likelihood is unscaled and we don't know the maximum value within a certain setting. Log likelihood is just the logarithm to make the numbers more handy and to transform the multiplicative character of likelihood into an additive relationship.

The second part of an information theory measure is a penalty term, that penalizes the number of parameters. Depending how this is defined, we get:

$$AIC = -2\ln(L) + 2k$$
$$BIC = -2\ln(L)) + k\ln(n)$$

with log likelihood $\ln(L)$, number of parameters k and sample size n.

In model selection, we have usually have several candidate models, that include or exclude certain explanation variables. Here the

full model is the model that includes all potential explanation variables,

null model is the model with no explanation variables. Often, it is just the mean of the series and in R it is symbolized with `~1`.

minimal adequate model (or most parsimonious model) is the model with the best compromise between goodness of fit (log likelihood) and number of parameters (k), i.e. the model with the lowest AIC resp. BIC.

Note that AIC and BIC are logarithmic values, so it does not matter wether they are positive or negative. It just matters which of the models has the lowest value. More about this can be found in JOHNSON and OMLAND (2004) and many modern statistics textbooks.

4 Statistical Parameters

Statistical tests work in one of two different ways:

1. by comparing data directly (non-parametric or distribution-free methods) or
2. by estimating specific measures (parameters) that subsume certain properties of interest from the distribution of the samples.

In statistics, calculating parameters from sample data is called "estimation" and the obtained parameter values, the "estimates" are symbolized with Latin letters while the true values of the population are symbolized by Greek letters. Sometimes, there are also different estimators for one (theoretical) property of a population, which are characterized by:

- Unbiasedness (the estimation converges towards the true value with increasing n),
- Efficiency (a relatively small n is sufficient for a good estimation),
- Robustness (the estimation is not much influenced by outliers or certain violations of statistical assumptions).

Depending on a particular question, different classes of parameters exist, especially measures of location (e.g. mean, median), variation (e.g. variance, standard deviation) or dependence (e.g. correlation).

4.1 Measures of location

The arithmetic mean of a sample is the sum of all values, divided by the sample size:

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

the geometric mean is defined as the nth root of the product of all data:

$$G = \sqrt[n]{\prod_{i=1}^{n} x_i}$$

but in most cases it is more practical to use the logarithmic form of the formula to avoid huge numbers that would make problems for the computer:

$$G = \exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \ln x_i\right)$$

The harmonic mean is the reciprocal of the mean of the reciprocals of a sample:

$$\frac{1}{H} = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{x_i} \quad ; x_i > 0$$

These measures can be easily calculated in R with the `mean`-function:

```
x <- c(42, 43, 45, 51, 52, 55, 58, 61, 65, 67)
mean(x)            # arithmetic mean
```

```
[1] 53.9
```

```
exp(mean(log(x))) # geometric mean
```

```
[1] 53.23059
```

```
1/mean(1/x)        # harmonic mean
```

```
[1] 52.56164
```

All these measures of location have in common, that they can be rather strongly influenced by outliers. A measure that is more robust is the **median**, that is the "middle value" that separates an ordered sample into two parts with half the sample size:

- n uneven:
$$m = x_{(n+1)/2}$$
- n even:
$$m = \frac{x_{n/2} + x_{n/2+1}}{2}$$

The **trimmed mean** forms a compromize between the arithmetic mean and the median. It is calculated like the mean after discarding a proportion of the smallest and biggest values, usually 5 to 25 percent:

```
median(x)            # median
```

```
[1] 53.5
```

```
mean(x, trim=0.1)    # trimmed mean
```

```
[1] 53.75
```

```
mean(x, trim=0.5)    # same as the median
```

```
[1] 53.5
```

Often the median or the trimmed mean are preferred over the mean, especially if the samples are likely to contain outliers or stem from a skewed distribution.

The **mode** is the value that occurs most frequently in a sample. Strictly speaking, this measure is defined only for discrete (binary, nominal, ordinal) scales, but it is also possible to obtain an estimate for continuous scales, e.g. from binned data that are frequencies of data according to size classes. As a first guess, one can simply use the middle value of the class with the highest frequency, but a better estimate uses a weighted mean respecting the frequencies of neighboring classes:

$$D = x_{lo} + \frac{f_k - f_{k-1}}{2f_k - f_{k-1} - f_{k+1}} \cdot w$$

Here, f is the class frequency, w the class width k the index of the most abundant class and x_{lo} its lower limit.

Another, more computer intensive but also more modern method is based on so-called *kernel density estimates*, Fig. 4.1), where the mode is estimated by its maximum:

```
hist(x, probability = TRUE)
dens <- density(x)
lines(dens)
dens$x[dens$y == max(dens$y)]
```

```
[1] 54.22913
```

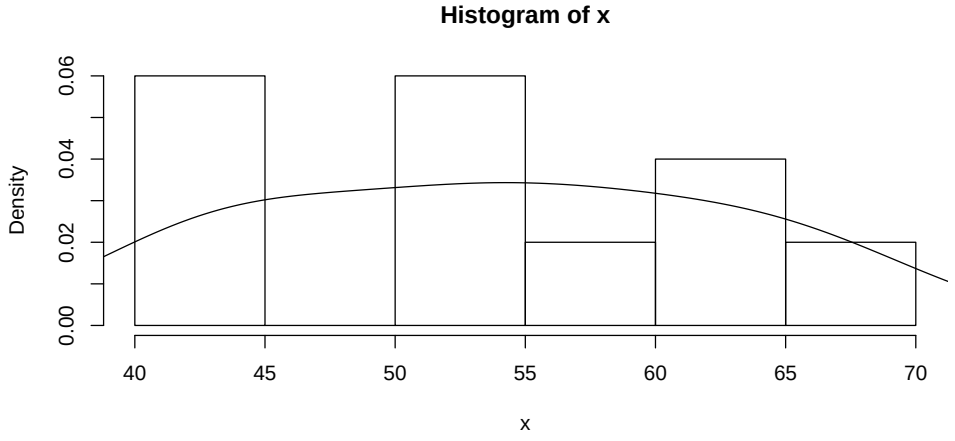


Figure 4.1: Histogram and density estimate.

A sample can also have more than one mode (e.g. for the abundance of a fish population with several age classes, cohorts), it is then called a multi-modal distribution.

The following example demonstrates a bi-modal distribution: (Fig. 4.2):

```
library(simecol)  # contains a function to detect peaks
# a bi-modal distribution from two normally distributed samples
x <- c(rnorm(50, mean=10), rnorm(20, mean=14))
hist(x, prob=T)
dens <- density(x)
lines(dens)
peaks(dens, mode="max")$x # outputs the modal values
```

```
[1] 10.32967 14.16258
```

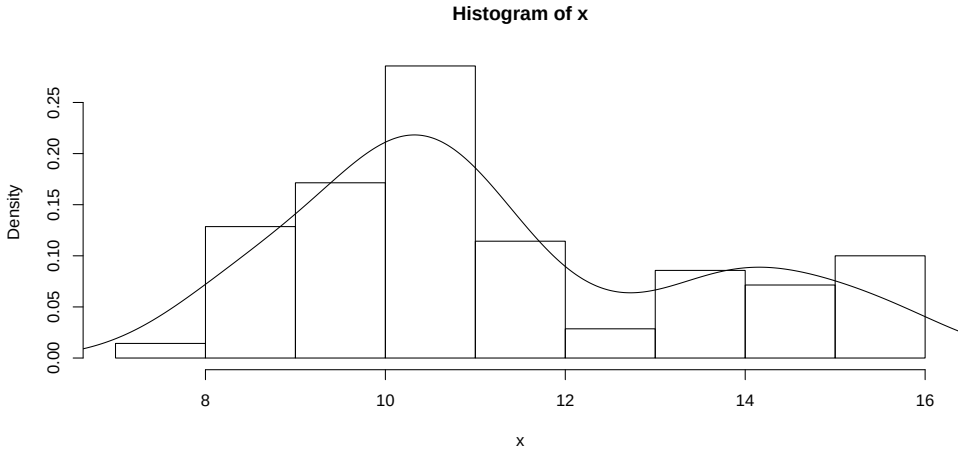


Figure 4.2: Histogram and kernel density estimate for a bi-modal distribution.

4.2 Measures of variation

Measures of variation are used to measure variability in the data or to qualify the accuracy of statistical parameters. The most common measure of variation is the variance s^2 and its square root, the standard deviation s:

$$s_x^2 = \frac{SQ}{df} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

with SQ sum of squared deviations from the mean $\bar{x}$ and df degrees of freedom, that is equal to the sample size n for the population and $n-1$ for a sample. If you don't know whether n or $n-1$ is correct, you should always use $n-1$.

In practice, s^2 is computed with the following formula:

$$s_x^2 = \frac{\sum(x_i)^2 - (\sum x_i)^2/n}{n-1}$$

using the sum and the sum of squares of the data.

The square root of the variance $s = \sqrt{s^2}$ is called the standard deviation. Its main advantage is, that it has the same measurement unit like the mean $\bar{x}$, so both can be directly compared.

The **coefficient of variation** cv or the relative standard deviation:

$$cv = \frac{s}{\bar{x}}$$

is very useful for comparing variations between different measurement units, locations or variables. It can only be calculated for data with a ratio scale, i.e. for measurement units that have an absolute zero (like meters) but not for variables like Celsius temperature or pH.

The **range** measures the difference between maximum and minimum of a sample:

$$r_x = x_{max} - x_{min}$$

but as it is likely to be influenced by outliers, it is much better to use the **interquartile range, IQR or** I_{50} that omits 25% of the smallest and 25% of the biggest values (one needs at least about 12 values):

$$I_{50} = Q_3 - Q_1 = P_{75} - P_{25}$$

Here Q_3 and Q_1 are called the first and the 3rd **quartiles** of an ascending ordered sample (also called the 25th or 75th percentiles, P_{25}, P_{75}).

For normally distributed samples we find a fixed relationship between I_{50} and the standard deviation:

$$\sigma = E(I_{50}/(2\Phi^{-1}(3/4))) \approx E(I_{50}/1.394)$$

where Φ^{-1} is the quantile function of the normal distribution.

Another, compared to the IQR even more robust measure of variation is the *median absolute deviation*:

$$MAD = median(abs(median - x_i))$$

This value is often rescaled by default with 1.4826 to approximate the standard deviation.

Application in R

All measures of variation can be easily calculated in R:

```
  x <- rnorm(100, mean=50, sd=10)  # 100 random numbers
  var(x)                           # variance
```

```
[1] 116.591
```

```
  sd(x)                            # standard deviation
```

```
[1] 10.79773
```

```
  range(x)                         # range
```

```
[1] 27.1372 80.5034
```

```
  quantile(x, c(0.25, 0.75))       # quartiles
```

```
     25%      75%
41.36227 57.98437
```

```
  IQR(x)                           # interquartile range
```

```
[1] 16.62209
```

```
  diff(quantile(x, c(0.25, 0.75))) # same, calculated from quartiles
```

```
     75%
16.62209
```

```
  mad(x)                           # median absolute deviation
```

```
[1] 12.3913
```

4.3 Standard error of the mean

In contrast to the deviation measures presented above, the **standard error** does not describe variability of the sample, but instead the variability of a statistical measure in general. Therefore, the "standard error of the mean" estimates the variability of the mean as a function of the standard deviation of the original sample and the sample size:

$$s_{\bar{x}} = \frac{s}{\sqrt{n}}$$

It measures accuracy of the mean and plays a central role for the calculation of confidence intervals and in many statistical tests, e.g. the t-test.

As a rule of thumb and for normally distributed samples with a sample size of about $n > 30$, the true mean μ can be expected with probability of 68 % within a range of $\bar{x} \pm s_{\bar{x}}$, but for small samples, quantiles of the t-distribution must be used instead.

The standard error is often used for error bars in graphics. This is correct, if the accuracy of the mean should be indicated. Sometimes, however, the error bars are intended to show variability of the samples. In such cases, it is more appropriate to use the standard deviation or, even better, the quartiles or minimum and maximum. In any case it is mandatory to indicate the type of error bars in the figure legend or figure caption.

Error bars in R

Error bars are most easily plotted with function `barplot2` from the add-on package `gplots` (Fig. 4.3):

```
library(gplots) # contains the barplot2 function
nx <- 2
ny <- 4
nz <- 10
x  <- rnorm(nx, mean=5, sd=1)
y  <- rnorm(ny, mean=4, sd=1)
z  <- rnorm(nz, mean=8, sd=2)
m  <- c(mean(x), mean(y), mean(z))
sx <- c(sd(x), sd(y), sd(z))/sqrt(c(nx, ny, nz))
barplot2(m, ci.l=m-sx, ci.u=m+sx, plot.ci=TRUE,
  names=c("x", "y", "z"), xlab="mean +/- se")
```

4.4 Excercises

1. Generate samples of random numbers with $n = 30$ and determine the different location and variation measures with Excel, R or your pocket calculator.
2. Load the data set `nit90` into R and plot and print a histogram. Then mark the different location and variation measures with pencil on the plot.
3. Determine appropriate location and variation measures of the variable "Sepal.Length" for the three species of the iris data set. How accurate are these measures? Is it possible to distinguish the species by "Sepal.Length"?

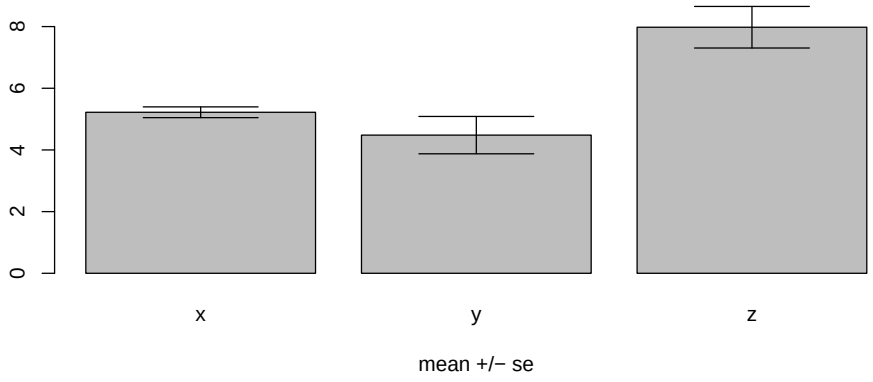


Figure 4.3: The standard error is the appropriate measure to compare means.

5 Distributions

Probability distributions are one of the core concepts in statistics and many statistics courses start with coin flipping or dice rolls. We begin with random number generators on the computer to get:

- a feeling about randomness and how a "true normal distribution" can look like,
- a tool for experimental design, for exploring statistical methods using data with known properties and for testing the power of our intended analyses beforehand.

So measured data are to be tested for normal distribution frequently. In practice, however, it turns out most of the times that there are more or less strong violations from it. This raises the question whether these violations are relevant. As for the random numbers used hereafter: they are created using a specific type of random number generator and therefore the type of distribution of the population is known.

For visualization we will first use the function `hist()`. Later we will get to know more types of graphics and tests.

5.1 Uniform distribution

Uniformly distributed random numbers have the same probability of occurence in a given interval (e.g. $(0, 1)$). In R random numbers are created with the function `runif`, with `r` standing for *random* and `unif` for *uniform*. The argument put in parentheses says how many random numbers are to be generated. We will start by generating 10 random numbers and display them:

```
runif(10)
```

```
[1] 0.3897159 0.6406357 0.5274679 0.9420222 0.7753624 0.3865327 0.9052764
[8] 0.4073827 0.5199052 0.5725259
```

before we create 400 new random numbers and save them in the variable `x`:

```
x <- runif(400)
```

Now we can plot the random numbers (Fig. 5.1):

```
par(mfrow=c(1,2))
plot(x)
hist(x)
```

To get an idea of how different uniformly distributed random numbers look like the generation and plotting can be combined in one command line, that can then repeated several times:

```
hist(runif(x))
```

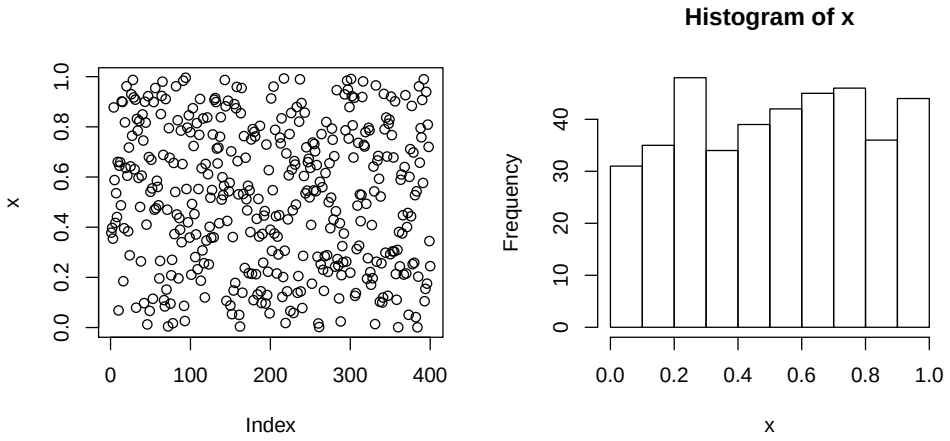


Figure 5.1: Uniformly distributed random numbers, on the left as a sequence of numbers, on the right as histogram.

If the random numbers are not to lie in the interval $(0,1)$ but in another interval $(5,10)$ we can either use normalization:

```
xmin <-  5
xmax <- 10
runif(100)  * (xmax - xmin) + xmin
```

or we supply the maximum and minimum to the function `runif` as optional arguments:

```
runif(100, xmin, xmax)
```

The histogram (Fig. 5.1, right) shows how an *empirical* uniform distribution (created with random numbers) looks like. But what does the ideal case, a theoretical uniform distribution, look like?

To find out we will use the function `dunif`, in which `d` stands for *density*. We can illustrate it by plotting the density function over the interval $(-0.2,1.2)$ with 100 (or, even better, 500) x values:

```
x <- seq(-0.2, 1.2, length=500)
plot(x, dunif(x), type="l",
          xlab="random variable X",
          ylab="probability density")
```

The density function $f(X)$ is often termed "pdf" (*probability density function*). The area under curve is 1, i.e. 100% of the results are located between $-\infty$ and $+\infty$ (or, as in our example, even in the interval $(0,1)$).

$$F(X) = \int_{-\infty}^{+\infty} f(X)dX = 1 \tag{5.1}$$

If we want to know, however, what percentage of events (here: how many of the random numbers created before) are smaller than a specified value x, we use the distribution function F as a definite integral:

$$F(x) = \int_{-\infty}^{x} f(X)dX \tag{5.2}$$

In R, this integral is available as function `punif` (*probability function*).

```
x <- seq(-0.2, 1.2, length=500)
plot(x, punif(x), type="l",
                  xlab="random variable X",
                  ylab="distribution function")
```

Cumulative frequencies

A uniform distribution appears as a straight ascending line in the interval $(0,1)$. For empiric distributions this corresponds to the cumulative frequency, which can be displayed as a cumulative histogram too. As the R- function `hist` knows no cumulative plots, we have to apply some handwork here. We will use `hist` only for binning (replacing data values falling into a small interval (the bin) with a value representative for that interval), but turn off the plotting function. Afterwards we have a look at the object `h` using `str`, calculate the cumulative sum of `h$counts` and plot the result with the more general function `barplot`:

```
x <- runif(400)
hist(x)

h <- hist(x, plot=FALSE)  # binning only, no plot
hcum <- cumsum(h$counts)
barplot(hcum, names.arg=round(h$mids,2),col="white")
```

Up to now, all histograms showed the absolute frequencies, while often relative frequencies are needed. For the relative class frequencies there is an option in `hist`, and the cumulated frequencies can be obtained with a simple division by the number of observations:

```
hcum <- cumsum(h$counts)/sum(h$counts)
barplot(hcum, names.arg=round(h$mids,2),
              col="white", ylab="Probability")
```

As a last important function in this respect the quantile function `qunif` is to be mentioned. It is the inverse of `punif`, with the help of which we are able to calculate up to what value a certain percentage of events can be found.

Example: In which symmetrical range can we find 95% of all values given an uniform distribution $U(40,60)$:

```
qunif(c(0.025, 0.975), min=40, max=60)
```

```
[1] 40.5 59.5
```

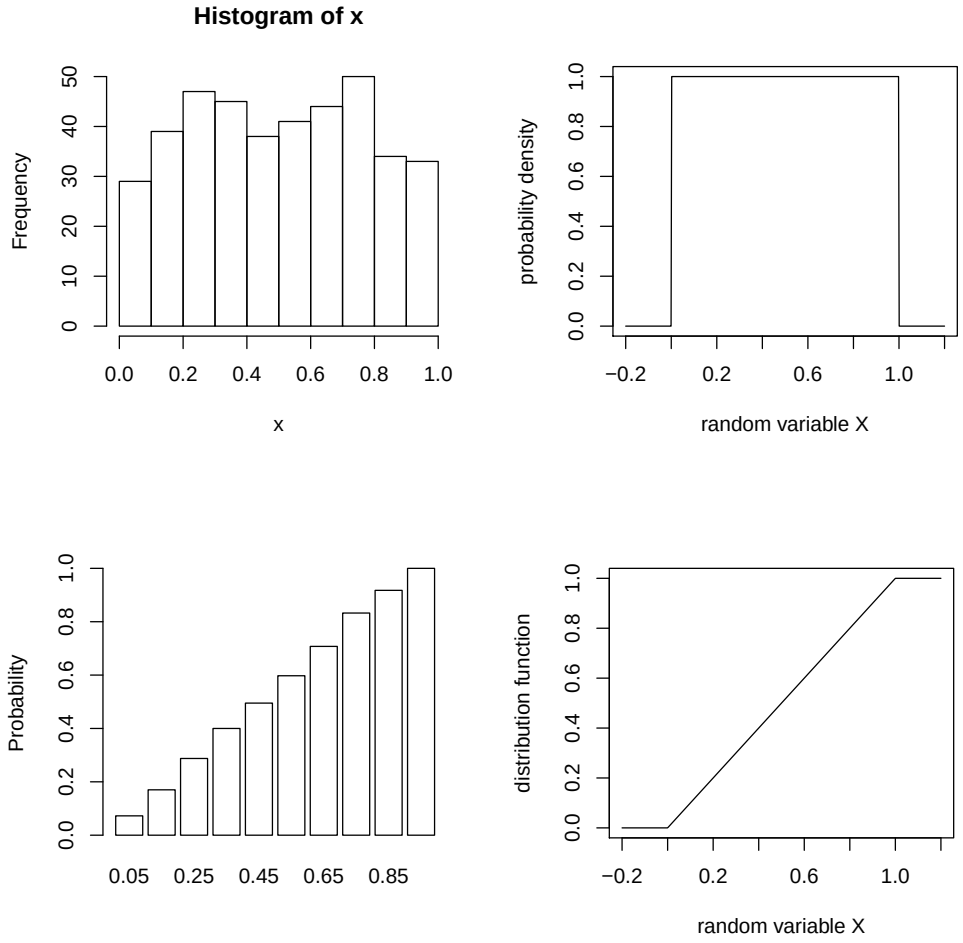


Figure 5.2: Uniform distribution, top: absolute frequency and density function, bottom: relative cumulated frequency and distribution function.

5.2 Normal distribution

The book by CRAWLEY (2002) gives a very intuitive introduction to the understanding of the normal distribution. He uses a simple exponential function as a starting point:

$$y = \exp(-|x|^m) \tag{5.3}$$

To illustrate this we plot this function in the interval $(-3,3)$ with an exponent of $m = 1,2,3,8$ (Fig. 5.3). On this occasion we can also practice defining a user-defined function, which we will call `f` here:

```
f <- function(x, m) {exp(-abs(x)^m)}  # the exponential function
par(mfrow=c(2,2))                     # space for 4 graphics
x <- seq(-3, 3, length=100)           # domain of definition
plot(x, f(x,1), type="l")
plot(x, f(x,2), type="l")
plot(x, f(x,3), type="l")
plot(x, f(x,8), type="l")
```

The function with $m = 2$ has an extremely high relevance for various reasons. It simply has to be scaled in a way for the area under curve to become 1 and we receive the standard normal distribution with mean value $\mu = 0$ and variance $\sigma^2 = 1$:

$$f(z) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-z^2/2} \tag{5.4}$$

Based on the standard normal distribution we can obtain other normal distributions with any desired mean value μ and any variance σ^2 by further scaling:

$$z = \frac{x - \mu}{\sigma} \tag{5.5}$$

Here μ moves the whole bell shaped curve along the x axis while σ leads to a stretching or compression in the direction of y. This scaling is termed "standardization" or z-transformation.

The normal distribution in R

For the normal distribution, R contains functions for random numbers (`rnorm`), the density function (`dnorm`), the distribution function (`pnorm`) and the quantiles (`qnorm`).

Now we will generate 100 random numbers from a normal distribution with $\mu = 50$ and $\sigma = 10$, plot them and determine (estimate) the mean value $\bar{x}$ and the standard deviation s of the sample:

```
x <- rnorm(100, 50, 10)
hist(x, probability = TRUE)
mean(x)
```

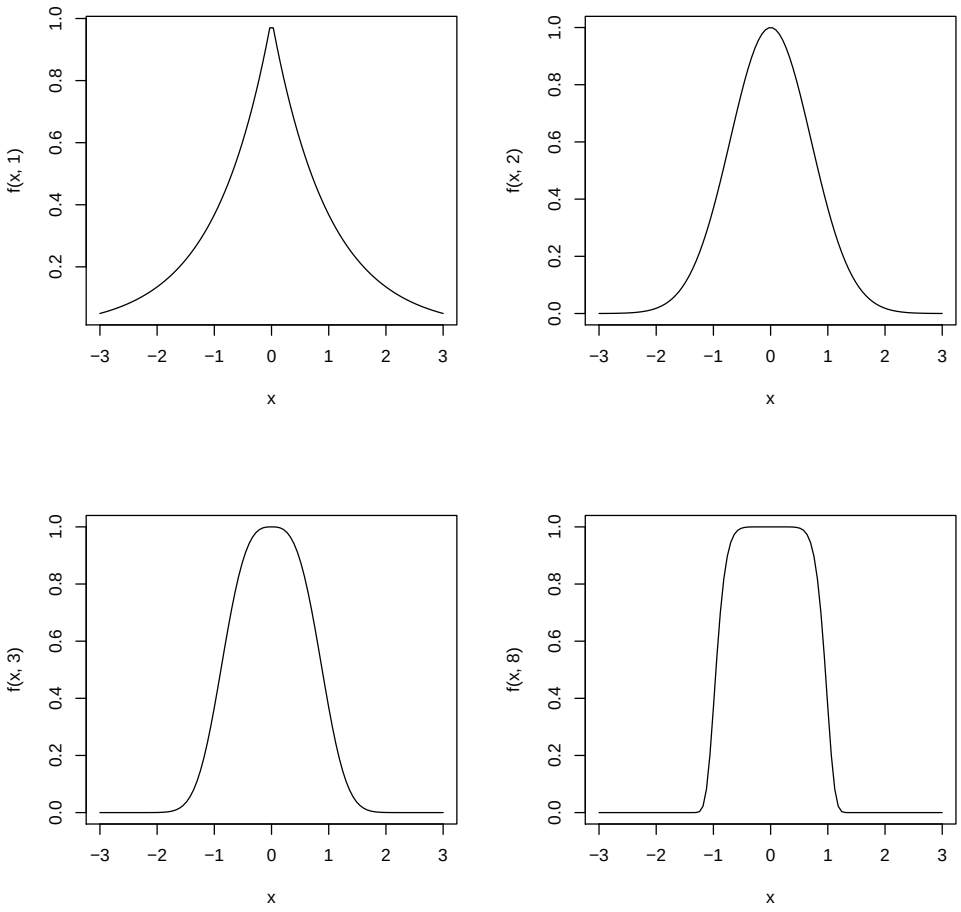


Figure 5.3: Different exponential functions. The function with the exponent $m = 2$ is quite close to a normal distribution and needs only scaling so that the integral becomes 1.0

```
[1] 49.60832
```

```
sd(x)
```

```
[1] 9.811248
```

Now, we will draw a bell-shaped curve into the diagram (Fig. 5.4). Therefore, we will, on the one hand, use a general density estimation with a so-called kernel density estimation and, on the other hand, a theoretical normal distribution with the sample parameters $\bar{x}$ und s:

```
lines(density(x), col="blue")
xnew <- seq(min(x), max(x), length=100)
lines(xnew, dnorm(xnew, mean(x), sd(x)), col="red")
```

```
[1] 49.60832
```

There is an important difference between the kernel density curve (see details in VENABLES and RIPLEY, 2002, or the R online help) and `dnorm`: in the first case we simply perform a general density estimation (a smoothing) without any concrete assumptions about the underlying distribution, in the second case we are already applying a statistical model, i.e. we assume a normal distribution.

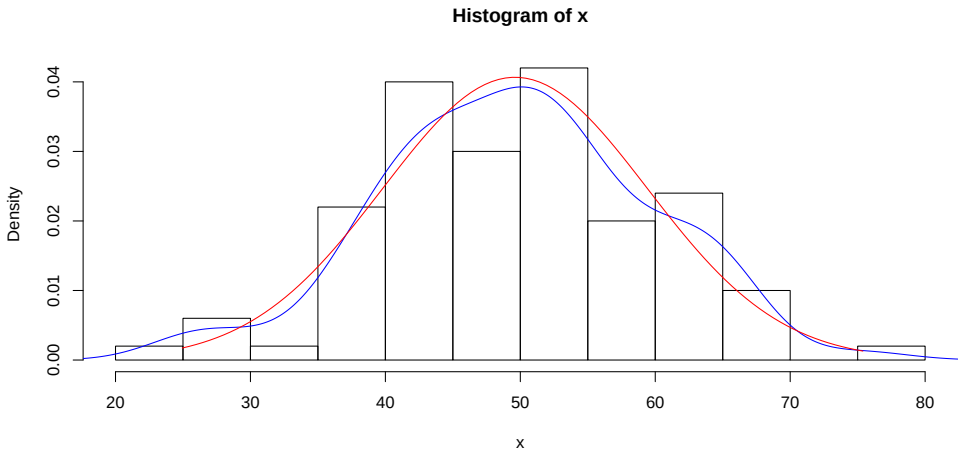


Figure 5.4: Normal distribution: classes (bars), density estimation (blue line) and theoretical density function (red line).

5.3 t-distribution

The t-distribution is a testing distribution, i.e. it describes how different statistical parameters are distributed. Other testing distributions are the χ^2-distribution (chi-squared distribution) and the F-distribution.

In principle the t-distribution is very similar to the normal distribution. In addition to the location parameter μ and the scaling parameter σ it contains another parameter for the degrees of freedom df. If the number of the degrees of freedom is low the t-distribution has very broad "tails", i.e. there is an increased probability of extreme values. For $df \to \infty$ or practically already for $df \approx 30$ the t-distribution converges towards a normal distribution.

Example

As an example we will plot a standard normal distribution with several t-distributions (Fig. 5.5) and a different number of degrees of freedom each.

```
x <- seq(-3,3, length=100)
plot(x, dnorm(x), type="l", col="red")
lines(x,dt(x, df=1), col="cyan")
lines(x,dt(x, df=2), col="blue")
lines(x,dt(x, df=4), col="cyan")
lines(x,dt(x, df=8), col="blue")
lines(x,dt(x, df=16), col="blue")
lines(x,dt(x, df=32), col="green")
```

In a second example we will examine the dependence of the frequently needed t-value $t_{1-\alpha/2}$ with $\alpha = 0.05$ (95% quantile in a two-sided test) from the number of degrees of freedom (Fig. 5.6). The two-sided 5% standard normal quantile (dashed line) serves for comparison:

```
plot(1:30, qt(0.975, 1:30), type="l",
   ylab="Student´s t", xlab="d.f.", ylim=c(0,15))
abline(h=qnorm(0.975), lty="dotted")
```

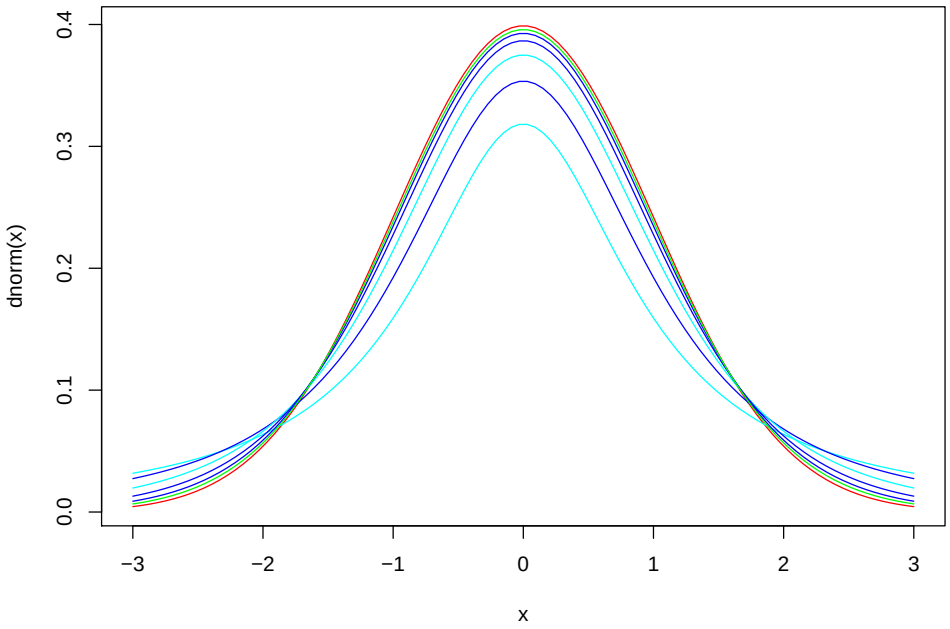


Figure 5.5: Density distribution of the normal distribution (red) and t-distributions with a varying number of degrees of freedom.

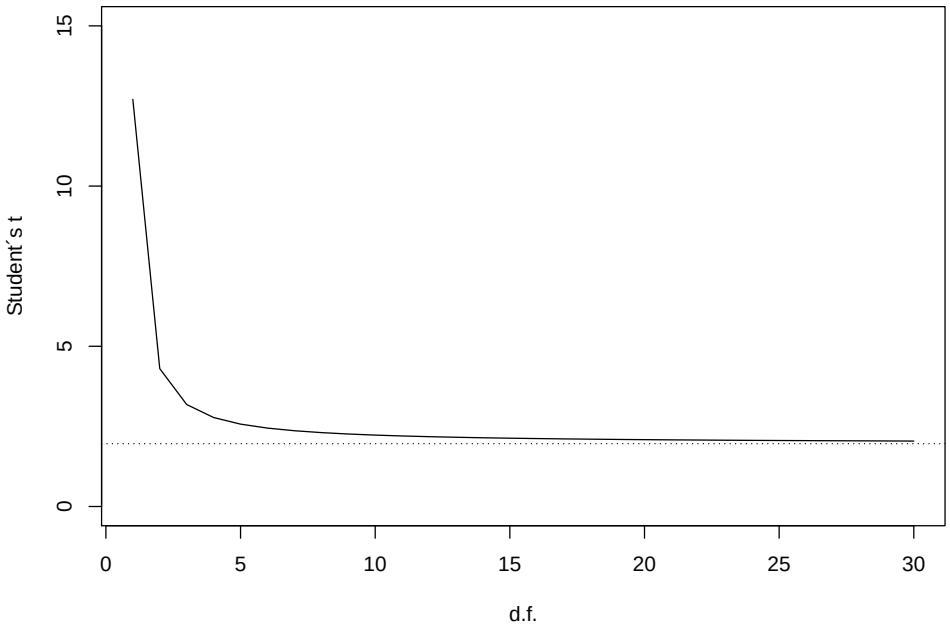


Figure 5.6: Dependence of the t-quantile from the number of degrees of freedom. Easily observable is the strong rise of the t-value especially at $df < 5$. At $df > 30$ the t-value (t =2.04) reaches approximately the quantile of the normal distribution (1.96).

The plot shows that we have to expect an additional "penalty" at sample sizes smaller than 5. This means that in addition to the effects of the standard error (see below) confidence intervals are growing very rapidly because of the t-distribution, and statistical tests are dramatically deteriorating in discriminatory power.

5.4 Logarithmic normal distribution (lognormal)

Many processes that can be observed in nature do not follow a normal distribution, but are limited by zero on the left side while having large extreme values on the right side. The flow rate or rivers, nutrient concentrations in waters or phytoplankton biomass in a lake may suffice as examples. On such processes the logarithmic normal distribution can be applied quite successfully.

```
x <- rlnorm(1000, meanlog=0, sdlog=0.5)
hist(x, probability=TRUE)
xnew <- seq(min(x), max(x), length=100)
lines(xnew, dlnorm(xnew, meanlog=mean(log(x)),
  sdlog=sd(log(x))), col="red")
```

The typical characteristic of the logarithmic normal distribution is that it is defined by the mean values and standard deviations of the logarithms. The according sample parameters are called $\bar{x}_L$ and s_L, R knows them as `meanlog` and `sdlog`.

```
x <- rlnorm(1000, meanlog=0, sdlog=0.5)
mean(x); sd(x)
```

```
[1] 1.09834
```

```
[1] 0.5693962
```

```
mean(log(x)); sd(log(x))
```

```
[1] -0.03037786
```

```
[1] 0.5036964
```

Therefore, the parameters $\bar{x}_L$ and s_L do not represent mean and standard deviation of the sample itself, but of the so-called parent distribution. Thus, taking the log from values of a lognormal distribution results in a normal distribution:

```
hist(log(x), probability=TRUE)
xnew <- seq(log(min(x)), log(max(x)), length=100)
lines(xnew, dnorm(xnew, mean=mean(log(x)), sd=sd(log(x))), col="red")
```

5.5 Gamma distribution

The gamma distribution is a right skewed distribution too, which is very useful for lots of practical problems, especially *generalized linear models* (GLM), which have been increasingly applied and which allow for analysis of variance of not normally distributed data, among other things. The gamma distribution is being

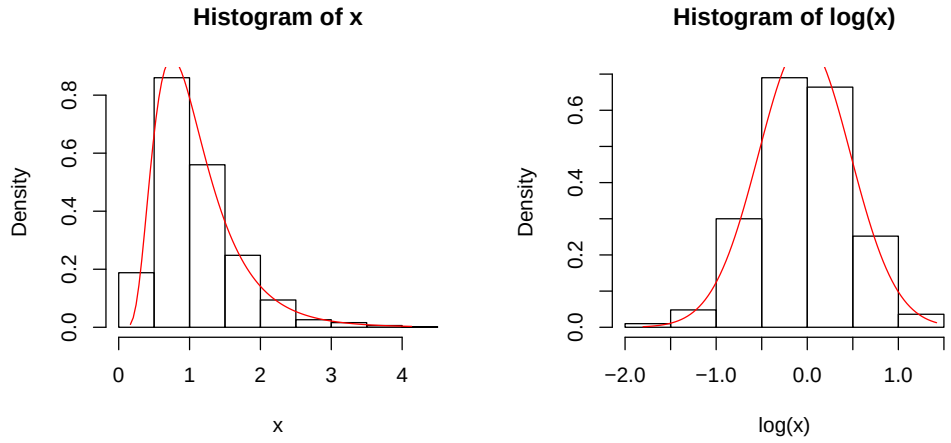


Figure 5.7: Logarithmic normal distribution (left) and according normal distribution (*parent distribution*, right).

described by the two parameters *shape* and *rate* or, alternatively, by *shape* and $scale = 1/rate$. The density function is:

$$f(x) = \frac{1}{\beta^{\alpha}\Gamma(\alpha)} x^{\alpha-1}\mathbf{e}^{-x/\beta} \tag{5.6}$$

Here α represents the *shape* parameter and β the *scale* parameter. Interestingly, $\alpha \cdot \beta$ is the mean and $\alpha \cdot \beta^2$ the variance. The χ^2-distribution is a special case of the gamma distribution with $\alpha = df/2$, $\mu = df$ and $\sigma^2 = 2df$. The exponental distribution is a special case with $\mu = \beta$, $\sigma = \beta^2$ und $\alpha = 1$.

As we see the gamma distribution is very flexible. For visualization we will draw a few examples (Fig. 5.8):

```
x <- seq(0.01, 4, length=100)
par(mfrow=c(2, 2))
plot(x, dgamma(x, .5, .5), type="l")
plot(x, dgamma(x, .8, .8), type="l")
plot(x, dgamma(x, 2, 2), type="l")
plot(x, dgamma(x, 10, 10), type="l")
```

Subsequently we will generate 1000 random numbers for these examples with `rgamma` and calculate mean value and variance.

Example

The data set `prk_nit.txt` contains individual biomasses of *Nitzschia acicularis* cells, which were determined in two practical courses. We will try to fit a gamma distribution (Fig. 5.9):

```
dat <- read.table("prk_nit.txt", header=TRUE, sep="\t")
#str(dat)
attach(dat)
rate <- mean(Nit90) / var(Nit90)
```

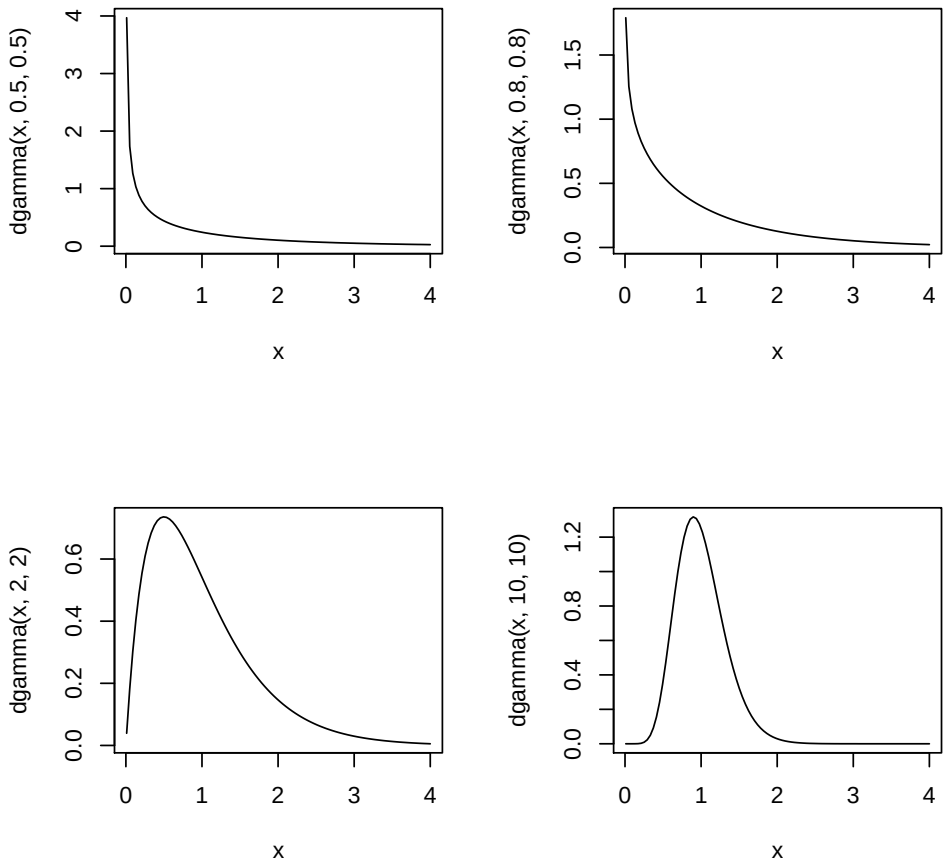


Figure 5.8: Examples showing the flexibility of the gamma distribution.

```
shape <- rate * mean(Nit90)
xnew <- seq(0.01, max(Nit90), length=100)
hist(Nit90, probability=TRUE)
lines(xnew, dgamma(xnew, rate=rate, shape=shape), col="red")
detach(dat)
```

5.6 Poisson distribution

The Poisson distribution is a discrete distribution. It is applied e.g. in bacteria and plankton counting or waiting queues and failure models. In the Poisson distribution $\mu = \sigma^2$ applies and this mean value parameter is called λ. The confidence interval depends solely on the number of counted units (k). The size of the confidence interval of a plankton counting can now easily be determined:

```
k <- 200 # counted units
qpois(c(0.025, 0.975), k)
```

```
[1] 173 228
```

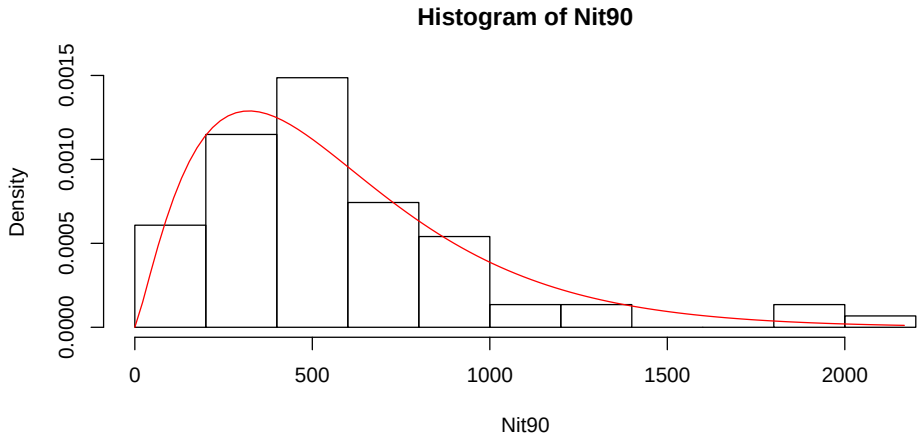


Figure 5.9: Histogram of the Nitzschia data set and its estimated gamma distribution.

The counting error is around 15% for $k = 200$, given the organism distribution matches a Poisson distribution, i.e. there is no clustering or dispersion.

Exercise: Plot the confidence interval of the Poisson distribution *relative* to k for an interval of $k = 5 \ldots 1000$.

5.7 Testing for distribution

Very often we want to know whether a data set belongs to a specific type of distribution. Though this sounds quite easy, it is in fact rather difficult for theoretical reasons. As we may remember, statistical tests prove for deviations from the null hypothesis, but here we want to test if H_0 is true!

This is not really possible, because "not significant" means only that a possible effect is either not existent or just too small to be detected with a given sample size. On the opposite, "significantly different" just means that there is a certain probability of deviation and that we may have false positives.

Another complication results from the fact that the tested data are not required to belong to a given distribution perfectly, e.g. to be "ideally normal distributed", which would be indeed impossible for real-world data. In effect, we are using sometimes a bigger α, e.g. 0.1 to allow some deviations and to avoid false positives and it is strongly recommended to use graphical methods.

5.7.1 Shapiro-Wilks-W-Test

The Shapiro-Wilks-W has become the standard test for testing for normality, while the the χ^2 (Chi-squared) test is nowadays rarely used for this purpose. It is of course very important for other types of problems.

For demonstration purposes let's generate 100 uniform random numbers and test whether they stem from a normally distributed population:

```
x <- rnorm(100)
shapiro.test(x)
```

```
        Shapiro-Wilk normality test

data:  x
W = 0.9906, p-value = 0.7165
```

In this example p is larger than 0.1, so we keep H_0 and conclude that nothing speaks against acceptance of the normal. This is of course obvious, because we used the generator function for normally distributed random numbers.

If we repeat this experiment, we may get false positives, i.e. data sets that are considered "not normal" according to the test, but that are still from a computer generated normal distribution in reality.

5.7.2 Graphical examination of normality

Already simple box plots (Fig. 5.10) allow a first assessment, whether a sample is normally distributed or has atypical variance or is obviously skewed:

```
x1 <- rnorm(100, mean = 50, sd = 10)       # normal distribution
x2 <- runif(100, min = 30, max = 70)       # uniform distribution
x3 <- rlnorm(100, meanlog = 2, sdlog = 1) # lognormal distribution
boxplot(x1, x2, x3,
  names=c("Normal", "Uniform", "Lognormal"))
```

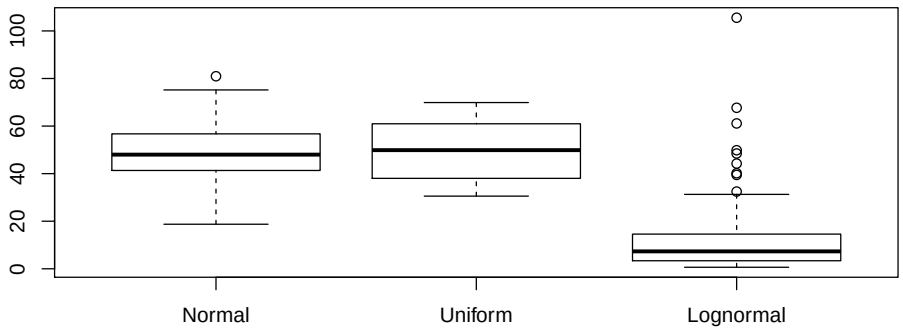


Figure 5.10: Comparison of differently distributed samples with boxplots.

Instead of one of the formerly used histograms for plotting:

- absolute frequency (frequency f_i per class i)
- relative frequency ($f_{i,rel} = \frac{f_i}{\sum f_i}$)
- cumulative frequency ($f_{i,cum} = \sum_{j=1}^{i} f_j$)
- or relative cumulative frequency (Scumulative percentage, $f_{i,cum,rel} = \frac{f_{i,cum}}{\sum f_i}$

it is also possible to plot the cumulative frequency against a theoretical distribution, and it is even simpler to plot just the ordered values against theoretical quantiles. This is called quantile-quantile plot (or Q-Q-plot) and can be done with function `qqplot`. For comparison with the normal, we just use `qqnorm`, and `qqline`, (Fig. 5.11)

```
qqnorm(x)
qqline(x, col="red")
```

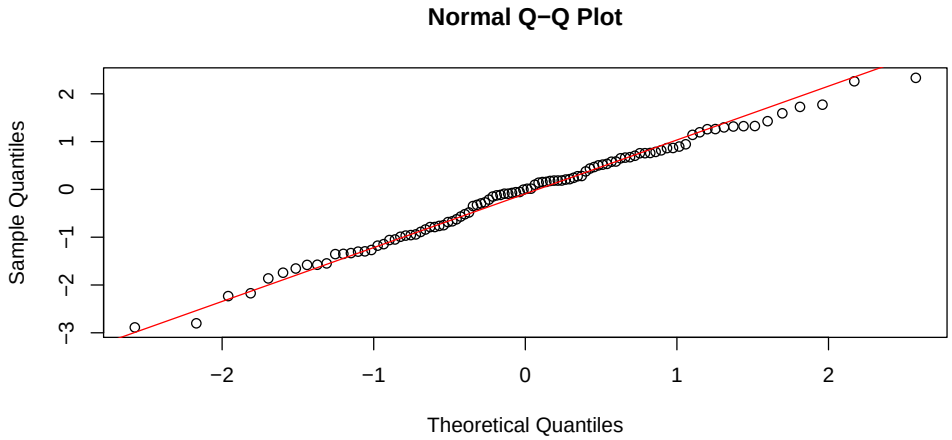


Figure 5.11: Q-Q-plot for graphical test of normal distribution.

A normal distribution can be assumed if the points follow a straight line.

Exercise 1: Generate series of normally distributed random numbers with $\mu = 50, \sigma = 10$ and check for normal distribution by means of histograms, Q-Q-plots and the Shapiro Wilks test..

Exercise 2: Test whether the data set `nit90` is normally distributed. The data set is available from the course web page or from `"http://www.simecol.de/data/prk_nit.txt"`.

5.7.3 Transformations

Transformations can be used to convert non-normal distributed data to an approximate normal distribution to fulfill the assumptions of common statistical methods. The transformation is a completely legal method and not an illegal manipulation of data. Its simple reason is the fact that, after transformation, we are able to "recycle" our knowledge about analysis of normally distributed data, so that we are able to go back to common methods. A number of useful transformations can be found in statistics texts like ZAR (1996) or, specifically for water quality and catchment variables in HÅKANSON and PETERS (1995). Here some examples from SACHS (1992):

- $x' = \ln(x), x' = \ln(x+a)$
- $x' = 1/x$ ("very powerful", i.e. to extreme in most cases)
- $x' = 1/\sqrt{x}$ (compromise between ln and $1/x$)

- $x' = (x+a)^c$ (a between 0.5 and 1)
- $x' = a + bx^c$ (very general, includes powers and roots)
- $x' = \sqrt{3/8 + x}$ (counts: 1, 2, 3 $\rightarrow$ 0.61, 1.17, 1.54, 1.84, ...)
- $x' = \lg(x + 3/8)$
- $x' = \ln(\ln(x))$ for giant numbers
- Percentages:
  - $x' = \arcsin\sqrt{x/n}$
  - $x' = \arcsin\sqrt{\frac{x+3/8}{n+3/4}}$

It is very important that transformations have to be monotonous, i.e. that the order of values is unchanged.

5.7.4 Box-Cox transformation

An outstandingly important class of transformations are powers and logarithms, that sometimes are intuitively used by people without testing the pre-requisites. One way, to overcome such an intuitive use and to determine the optimal transformation from this class is the so-called Box-Cox transformation:

$$y' = y^\lambda \tag{5.7}$$

where $\lambda = 0$ means that a logarithmic transformation would be appropriate. Function `boxcox` requires a so-called "model formula" or the outcome of a linear model (`lm`) as the argument, in our example we use just the model formula for a "null model" to test the full data set without explanation variables (`~1`) More about model formulas can be found elsewhere, e.g. in the R documentation.

```
library(MASS) # package that belongs to the book of venables and Ripley
dat <- read.table("prk_nit.txt", header=TRUE)
attach(dat)
boxcox(Nit90 ~ 1)
detach(dat)
```

The dotted vertical lines and the horizontal 95 %-line show the confidence limits for possible transformations. Here we see that either a logarithmic transformation ($\lambda = 0$) or a power of approximately 0.5 are suitable. It is also possible to obtain the numerical value directly:

```
attach(dat)
bc <- boxcox(Nit90 ~ 1)
str(bc)
```

```
List of 2
 $ x: num [1:100] -2 -1.96 -1.92 -1.88 -1.84 ...
 $ y: num [1:100] -237 -233 -230 -226 -223 ...
```

```
bc$x[bc$y == max(bc$y)]
```

```
[1] 0.1818182
```

```
detach(dat)
```

We should keep in mind that these are approximate numbers so that it makes no sense to use more than one decimal.

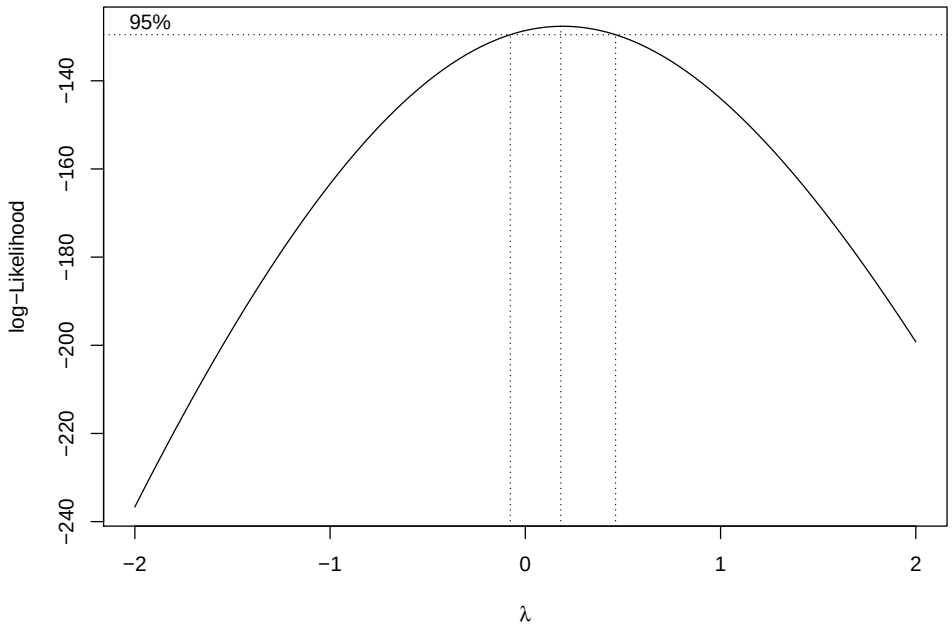


Figure 5.12: Checking for Box-Cox-transformation.

It is also possible to test for several related series at once by using explanation factors (here `f`) on the right hand side of the model formula:

```
nit90 <- na.omit(dat$Nit90)
nit85 <- na.omit(dat$Nit85)
x <- c(nit85, nit90)
f <- factor(rep(c("nit85", "nit90"), times=c(length(nit85), length(nit90))))
boxcox(x ~ f)
```

Exercise

Determine optimal transformations for both *Nitzschia* data sets and for the pooled data and plot the transformed data by means of Q-Q plots and histograms. Add the density function of the normal distribution as bell-shaped curve to the histogram and finally, apply Shapiro Wilks W test.

5.8 The central limit theorem

The central limit theorem of statistics (CLT) tells us that sums of a large number n of independent and identically distributed random values are normally distributed, independently on the type of the original

distribution. The required number n depends, however, on the skewness of the original distribution.

This means for statistical tests, that we can sometimes even use methods for normally distributed data, when:

- we have a large data set,
- the applied method deals with mean values of the data set and not with the original numbers,
- the original distribution is not "too skewed".

To demonstrate, how the CLT works, we perform the following simulation experiment. In this we generate a matrix with 100 rows and 12 columns of uniformly distributed random numbers and compute the row sums. Then we plot histograms for the original uniformly distributed data and for the row sums (5.13):

```
 par(mfrow=c(1,2))
 x  <- matrix(runif(12*100), nrow=12)
 xs <- colSums(x)
 hist(x)
 hist(xs)
 shapiro.test(x)

        Shapiro-Wilk normality test

data:  x
W = 0.9537, p-value < 2.2e-16

 shapiro.test(xs)

        Shapiro-Wilk normality test

data:  xs
W = 0.9748, p-value = 0.05161
```

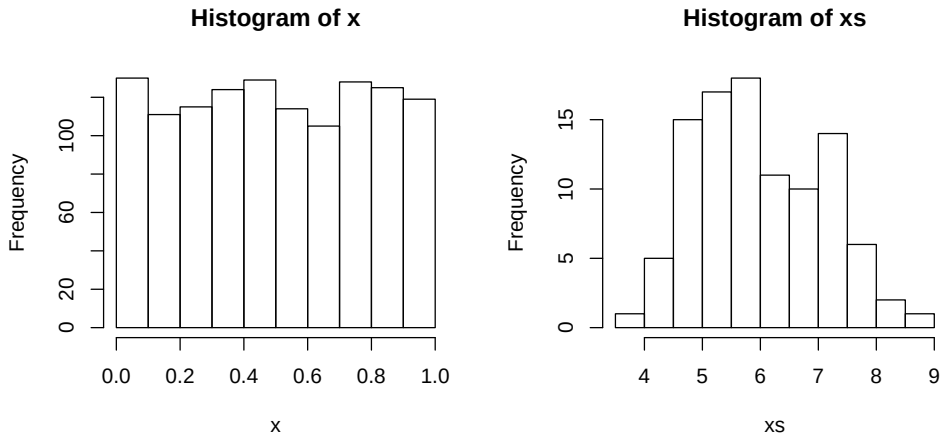


Figure 5.13: Histogram of a uniformly distributed sample withs $N = 1200$ (left) and of 100 means values taken from 12 original values (right). We see that the mean values are already quite close to a normal, despite the fact that the original distribution was uniform.

5.9 Confidence intervals for the mean

In order to assess the accuracy of a mean value, we can use the standard error of the mean:

$$s_{\bar{x}} = \frac{s}{\sqrt{n}} \tag{5.8}$$

This means that the variability of the mean is half, if we increase the sample size four times (2^2). Furthermore, it is possible to estimate the interval in which the true mean is found with 95 % probability, that is the 95 % confidence interval:

$$CI_{95\%} = (\bar{x} - z_{0.975} \cdot \frac{s}{\sqrt{n}}, \bar{x} + z_{0.975} \cdot \frac{s}{\sqrt{n}}) \tag{5.9}$$

with $z_{1-\alpha/2} = z_{0.975} = 1.96$. For small samples ($n \lessapprox 30$) but also in general we can use the t distribution instead of the normal, i.e. the t quantile with $n-1$ degrees of freedom instead of z. The following example shows the estimation of the confidence interval for a normally distributed random variable $\mu = 50$ and $\sigma = 10$:

```
n <- 10
x <- rnorm(n, 50, 10)
m <- mean(x)
s <- sd(x)
se <- s/sqrt(n)
# lower and upper confidence limits
m + qt(c(0.025, 0.975), n-1) * se
```

```
[1] 44.53931 59.23066
```

For real data we should of course respect their original distribution, especially if the sample size is small (see CLT). Then we may consider to estimate the confidence interval for a transformed (e.g. after taking the log) parent distribution and then back-transform (i.e. exp) the results:

```
x <- log(dat$Nit90)
m <- mean(x)
s <- sd(x)
n <- length(x)
se <- s/sqrt(n)
ci <- m + qt(c(0.025, 0.975), n-1) * se
exp(m)   # is the geometric mean
```

```
[1] 475.8295
```

```
exp(ci)  # an asymmetric confidence interval
```

```
[1] 407.8510 555.1383
```

5.10 Outliers

Extremely large or extremely small values are sometimes called "outliers" what means that they are "not really" from the population that we want to analyze, but instead influenced by another process, e.g. by mixing up of two samples in the lab. However, this "other process" can also be something interesting, or point to a new phenomenon, so it is always difficult to exclude values only because they are "too big" or "too small" and it is better to find the reason, why they are so extreme.

Nevertheless, we can find several outlier tests in the literature, e.g. the 4σ-rule, where $\bar{x}$ and s have to be calculated without the outlier(s) and should be $n \geq 10$. Another outlier test according to SACHS (1992) for $n \geq 25$ can be performed as follows. First, we calculate a value T_1 with:

$$T_1 = \left| \frac{x_1 - \mu}{\sigma} \right|$$

and then we lookup a respective table in the book. For linear models and GLMs we can find an outlier test (the Bonferroni outlier test) in package `car`. In the following, the 21st value (i.e. the 12) is identified as an outlier:

```
library(car)
x <- c(rnorm(20), 12) # the 12 is an outlier
outlierTest(lm(x~1))  # x~1 is the null model
```

```
   rstudent unadjusted p-value Bonferonni p
21 10.19573          3.848e-09   8.0808e-08
```

Instead of eliminating outliers it is also possible to use methods that are robust against this from the beginning, e.g. the median or the trimmed mean instead of the ordinary arithmetic mean, to use rank-based methods (like Spearman correlation), robust methods (like robust regression) or bootstrapping.

Note: Outliers may be omitted in an analysis, but the the number and extend of outliers must be mentioned!

Extreme values in boxplots

The boxplot function `boxplot` marks extreme values with stars if they are more than 1.5 times distant from the box limits, compared to the width of the interquartile box (`range=1.5`). However, we should not confuse the term "extreme value" (due to a skewed or heavy tailed distribution) with the term "outlier".

If you intend to present boxplots to people with less experience in statistics it maybe sometimes advisable to omit this option and just to draw the whiskers to the extreme values using option `range=0`:

```
par(mfrow=c(1,2))
x <- c(1,2,3,4,5,6,12)
boxplot(x)
boxplot(x, range=0)
```

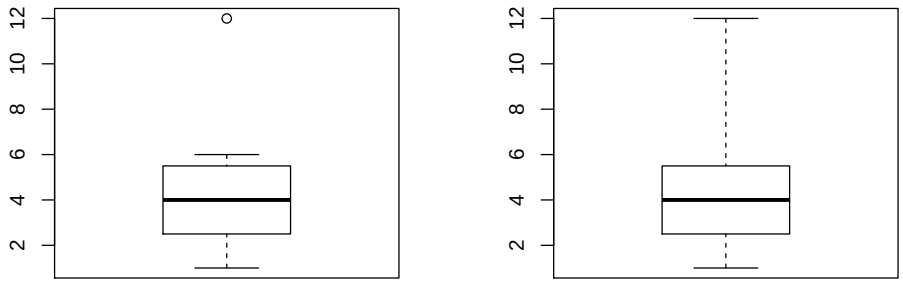


Figure 5.14: Boxplots with separately annotated extreme values (left) and with whiskers that include the extreme values (right).

6 Classical tests

Sophisticated methods such as generalized mixed models or labor-intensive multivariate analysis are not always necessary. In many cases so-called classical tests are sufficient. Here, the principle of parsimony is applies too: the simplest method should be preferred.

In the following chapter we will assume that most tests are more or less known already (otherwise consult your favorite statistics book, e.g. DALGAARD, 2002). Therefore we will limit our efforts to a few short examples and instructions of the application in R.

6.1 Testing for homogeneity of variances

Checking homogeneity (approximate equality) of variances is, on the one hand, a necessary precondition for a number of methods (for example comparison of mean values) and on the other hand the heart of a number of more sophisticated methods (such as analysis of variance). The classical F-test, which is based upon the quotient of two variances, is available in R as `var.test`. More than two variances can be compared using either the parametric Bartlett's test or the non-parametric Fligner-Killeen test. The latter one is considered to be very robust against deviations from the normal distribution. For demonstration purposes we will generate three normally distributed data sets with identical (`x1, x3`) and different (`x2`) variance respectively, and will apply the different variance tests to them. We will visualize them in a boxplot (fig. 6.1):

```
x1 <- rnorm(10, 5,  1)
x2 <- rnorm(10, 5,  2)
x3 <- rnorm(10, 10, 1)
boxplot(x1, x2, x3)
```

For comparing variances we have several tests available, the classical F-test:

```
var.test(x1, x2)
```

```
        F test to compare two variances

data:  x1 and x2
F = 0.211, num df = 9, denom df = 9, p-value = 0.02989
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.05242167 0.84968342
sample estimates:
ratio of variances
         0.2110493
```

Bartlett's test, which is also suited for comparison of more than 2 variances:

```
bartlett.test(list(x1, x2, x3))
```

```
        Bartlett test of homogeneity of variances

data:  list(x1, x2, x3)
Bartlett's K-squared = 7.7136, df = 2, p-value = 0.02114
```

or as a non-parametric alternative the Fligner-Killeen test:

```
fligner.test(list(x1, x2, x3))
```

```
        Fligner-Killeen test of homogeneity of variances

data:  list(x1, x2, x3)
Fligner-Killeen:med chi-squared = 2.2486, df = 2, p-value = 0.3249
```

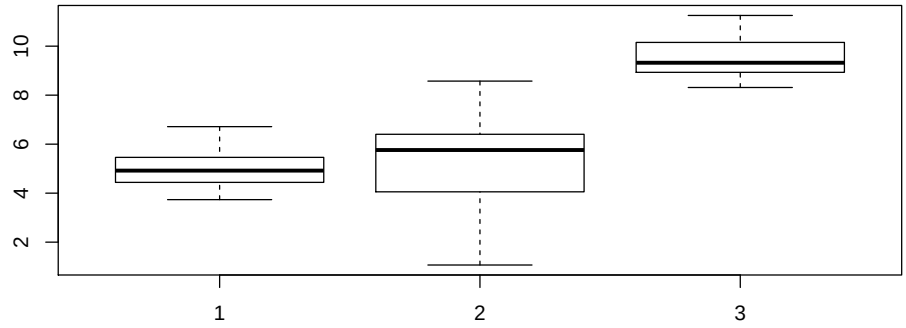


Figure 6.1: Boxplots of 3 samples from normally distributed populations.

6.2 Testing for differences between mean values

6.2.1 One sample t-Test

We can test for differences between mean values with the help of the classical t-test. The one-sample t-test tests if a sample is from a population with a given mean value μ:

```
x <- rnorm(10, 5, 1)# normally distributed sample with mu=5, s=1
t.test(x, mu=5)     # does x come from a sample with mu=5?
```

```
        One Sample t-test

data:  x
t = -1.116, df = 9, p-value = 0.2933
alternative hypothesis: true mean is not equal to 5
95 percent confidence interval:
```

```
 4.404882 5.201915
sample estimates:
mean of x
 4.803398
```

6.2.2 Two sample t-Test

In the two-sample t-test two independent samples are compared:

```
x1 <- rnorm(12, 5, 1)
x2 <- rnorm(12, 5.7, 1)
t.test(x1, x2)
```

```
        Welch Two Sample t-test

data:  x1 and x2
t = -2.5904, df = 17.393, p-value = 0.01882
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.8064299 -0.1862296
sample estimates:
mean of x mean of y
 4.683433  5.679763
```

That means that both samples differ significantly ($p < 0.05$). It has to be mentioned that R did not perform the "normal" t-test but the Welch test (also termed heteroscedastic t-test), for which the variances of both samples are not required to be identical.

The classical procedure would be as follows:

1. Perform a check for the identity of both variances with `var.test` beforehand:

   ```
   var.test(x1, x2)  # F-Test
   ```

   ```
          F test to compare two variances

   data:  x1 and x2
   F = 3.1211, num df = 11, denom df = 11, p-value = 0.0719
   alternative hypothesis: true ratio of variances is not equal to 1
   95 percent confidence interval:
     0.8984895 10.8417001
   sample estimates:
   ratio of variances
             3.121082
   ```

2. if $p < 0.05$ then the variances are significantly different, so the Welch test (see above) needs to be used.

3. if $p > 0.05$, then variances are likely to be equal and the "normal" t-test can be applied:

   ```
   t.test(x1, x2, var.equal=TRUE) # classical t-test
   ```

```
        Two Sample t-test

data:  x1 and x2
t = -2.5904, df = 22, p-value = 0.0167
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.793999 -0.198661
sample estimates:
mean of x mean of y
 4.683433  5.679763
```

6.2.3 Paired t-Test

If the present data are paired (e.g. an allergy test on the left and right arm) the paired t-test is used. Its advantage is that the influence of individual differences (i.e. a covariate) is eliminated. The downside is that the number of degrees of freedom is reduced. But if data are really paired it is definitely advantageous to take that information into account:

```
x1 <- c(2, 3, 4, 5, 6)
x2 <- c(3, 4, 7, 6, 8)
t.test(x1, x2, var.equal=TRUE) # p=0.20    not significant
```

```
        Two Sample t-test

data:  x1 and x2
t = -1.372, df = 8, p-value = 0.2073
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.28924  1.08924
sample estimates:
mean of x mean of y
      4.0       5.6
```

```
t.test(x1, x2, paired=TRUE)    # p=0.016   significant
```

```
        Paired t-test

data:  x1 and x2
t = -4, df = 4, p-value = 0.01613
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.710578 -0.489422
sample estimates:
mean of the differences
                  -1.6
```

It can be seen that the paired t-test has a greater discriminatory power.

6.2.4 Rank-based tests (Wilcoxon and Mann-Whitney U-test)

If data are not approximately normal distributed a transformation might be helpful. Alternatively, a non-parametric, rank-based test (Wilcoxon-test or Mann-Whitney-U-test) could be used. Both are available in R under `wilcox.test`.

```
dat <- read.table("prk_nit.txt", header=TRUE)
attach(dat)
boxplot(Nit90, Nit85)
wilcox.test(Nit90, Nit85)
```

```
        Wilcoxon rank sum test with continuity correction

data:  Nit90 and Nit85
W = 3007, p-value = 3.194e-07
alternative hypothesis: true location shift is not equal to 0
```

An analogous test for more than two samples is the Kruskal-Wallis rank-sum test. (`kruskal.test`).

The package `exactRankTests`[1] contains improved alternatives to the Wilcoxon test. One is `wilcox.exact`, which also accepts ties (i.e. double values), and a permutation test (`perm.test`), which calculates the exact probability based on a complete permutation of all possibilities. The permutation test is relatively demanding in terms of computing power, but does not seriously raise any problems to modern computers. Ultimately, the Wilcoxon test is simply an approximation of the permutation test.

When we perform the permutation test in R we have to take care that missing values (`NA` values) have been eliminated:

```
library(exactRankTests)
wilcox.exact(Nit90, Nit85)
```

```
        Asymptotic Wilcoxon rank sum test

data:  Nit90 and Nit85
W = 3007, p-value = 3.153e-07
alternative hypothesis: true mu is not equal to 0
```

```
perm.test(na.omit(Nit85), Nit90)
```

```
        Asymptotic 2-sample Permutation Test

data:  na.omit(Nit85) and Nit90
T = 16827.5, p-value = 7.744e-06
alternative hypothesis: true mu is not equal to 0
```

```
detach(dat)
```

[1] This package still works, but is not being developed anymore. Instead, the new package `coin` contains a generalized approach to these and other non-parametric tests.

Table 6.1: Locations of *Daphnia* clone groups in the stone pit lake Gräfenhain (clone A, clone B, clone C, epilimnic=top, hypolimnic=bottom, data: Matthes, Marco, unpublished)

	Epilimnion	Hypolimnion
Klon A	50	87
Klon B	37	78
Klon C	72	45

6.3 Testing for correlation

6.3.1 Contingency tables for nominal variables

Contingency tables can be used to uncover relationships between nominal (i.e. categorical or qualitative) data, e.g. between eye and hair color, the kind of treatment and the number of those healed (healed/ not healed), or between a *Daphnia*-clone and its preferred depth in a lake (Table 6.1). Here it is important to use the absolute number of examined individuals (e.g. experimental animals) and not percentages or "individuals per liter".

For testing purposes we start by creating a matrix with 3 rows and 2 columns:

```
x <- matrix(c(50, 37, 72, 87, 78, 45), ncol=2)
x
```

```
     [,1] [,2]
[1,]   50   87
[2,]   37   78
[3,]   72   45
```

and then perform a χ^2-test or Fisher's exact test:

```
chisq.test(x)
```

```
        Pearson's Chi-squared test

data:  x
X-squared = 24.2554, df = 2, p-value = 5.408e-06
```

or:

```
fisher.test(x)
```

```
        Fisher's Exact Test for Count Data

data:  x
p-value = 5.807e-06
alternative hypothesis: two.sided
```

As a result we receive significant correlation between the clones and their location i.e. the locations of the clones differ.

6.3.2 Correlation coefficients for metric variables

For metric data the most frequently used correlation coefficients are Pearson's correlation coefficient, Spearman's rank correlation coefficient or or Kendall's tau. All these correlation tests can be run in R with the help of `cor.test`:

```
x <- c(1,2,3,5,7,9)
y <- c(3,2,5,6,8,11)
cor.test(x, y, method="pearson")
```

```
        Pearson's product-moment correlation

data:  x and y
t = 7.969, df = 4, p-value = 0.001344
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.7439930 0.9968284
sample estimates:
      cor
0.9699203
```

If the linearity of a relationship or the normality of the residuals is doubtful, a rank correlation test can be carried out. Mostly, Spearman's rank correlation coefficient is used:

```
cor.test(x, y, method="spearman")
```

```
        Spearman's rank correlation rho

data:  x and y
S = 2, p-value = 0.01667
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.9428571
```

. . . and sometimes "Kendall's Tau":

```
cor.test(x, y, method="kendall")
```

```
        Kendall's rank correlation tau

data:  x and y
T = 14, p-value = 0.01667
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.8666667
```

If only the correlation coefficients are to be calculated (e.g. for a complete matrix or all the variables in a data frame), then function `cor` can be used.

6.4 Determining the power of statistical tests

Besides the structure of the experimental design and specific properties of the used test, the discriminatory power depends mainly on:

- the relative effect size (effect/standard deviation, $\delta = \frac{(\bar{x}_1 - \bar{x}_2)}{s}$,
- the sample size n,
- and the pre-defined significance level α.

Here, the following rule applies: the smaller α, n and Δ, the bigger the error of the second kind β (type II error), i.e. the probability to overlook effects despite of their existence. Therefore, it is advisable to set the sample size before realizing an experiment and to test the statistical procedure to be applied. Determining the sample size depending on α, β and Δ or, vice versa, estimating β for a given n is called **power analysis**.

In the one-sample case the smallest necessary sample size can be found using a simple formula:

$$n = \left(\frac{z_\alpha + z_{1-\beta}}{\Delta}\right)^2$$

with z being the quantiles (`qnorm`) of the standard normal distribution for α (probability of error) and for $1-\beta$ being the power, whereas $\Delta = \delta/s$ is the relative effect size.

In the two-tailed case (called two-sided as well) with $\alpha = 0.025$ and $\beta = 0.2$ the two z-quantiles are 1.96 and 0.84 respectively. What follows from this is the rule of thumb:

$$n = (1.96 \pm 0.84)^2 \cdot 1/\Delta^2 \approx 8 \cdot 1/\Delta^2$$

This rule allows for a certain estimation, but is valid only for the one sample problem. For other experimental designs and tests specific nomograms or equations exist (e.g. in ZAR, 1996).

6.4.1 Power of the t-test

The power of a t-test, the necessary sample size or the minimum effect size, respectively, can be derived with the function `power.t.test()`. Thus:

```
power.t.test(n=5,delta=0.5,sig.level=0.05)

     Two-sample t test power calculation

              n = 5
          delta = 0.5
             sd = 1
      sig.level = 0.05
          power = 0.1038399
    alternative = two.sided

NOTE: n is number in *each* group
```

results in a alarmingly low power of 0.10, i.e. for $n = 5$ an effect of half a standard deviation that exists in reality will only be detected in 1 out of 10 cases.

For reaching a power of 80 % at $n = 5$ a relative effect size of at least 2 is necessary:

```
power.t.test(n=5,power=0.8,sig.level=0.05)

     Two-sample t test power calculation

              n = 5
          delta = 2.024438
             sd = 1
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

In order to find a comparatively weak effect of $0.5s$ a sample size of at least $n = 64$ would be needed:

```
power.t.test(delta=0.5,power=0.8,sig.level=0.05)

     Two-sample t test power calculation

              n = 63.76576
          delta = 0.5
             sd = 1
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

Here it can be clearly seen that either a large sample size or a large effect strength are needed to find an existant effect. Otherwise, the power will become so low that the experiment wouldn't be worth the effort.

Presently, in R there are further functions for power analysis of balanced one-way ANOVA designs a for a proportion test (`prop.test`). For other problems the simulation method can be used.

6.4.2 Simulation method *

Alternatively, the power can be estimated with simulations. This is a little bit more demanding in terms of computing power and delivers only approximate values, but works in principle with any test design. In the following we will execute 1000 t-tests two samples from a well-defined statistical population with known differences between their mean values and will test, how many percent of the tests will deliver a significant result. This will be counted in the variables `a` and `b`:

```
### simulated power of a t-test ###

# population parameters
n <- 10
```

```
xmean1 <- 50
xmean2 <- 55
xsd1   <- xsd2 <- 10
alpha  <- 0.05
# number of test runs in the simulation
nn <- 1000
a <- b <- 0
for (i in 1:nn) {
  # creating random numbers
  x1 <- rnorm(n, xmean1, xsd1)
  x2 <- rnorm(n, xmean2, xsd2)
  # results of the t-test
  p <- t.test(x1,x2,var.equal=TRUE)$p.value
  if (p < alpha) {
     a <- a+1
   } else {
     b <- b+1
  }
}
print(paste("a=", a, ", b=", b, ", a/n=", a/nn, ", b/n=", b/nn))
```

We receive (approximately, as it is a random experiment):

```
[1] "a= 194 , b= 806 , a/n= 0.194 , b/n= 0.806"
```

Here a/n is the number of significant results, i.e. despite of a demonstrably existing difference between both mean values, we will receive a significant result in merely 20% of cases. So the power $(1-\beta)$ is just 20%.

6.5 Exercises

1. Compare the mean values of the flower characteristics "Sepal Length" a) of *Iris setosa* and *Iris versicolor* and b) of *Iris versicolor* and *Iris virginica* respectively. They are part of the Iris dataset (`data(iris)`). Choose a suitable test and interpret the results.
2. Repeat the test with subsets of the samples of the particular species (try $n = 5$ or $n = 10$ for example).
3. Which effect size would be needed to detect a significant difference in a two-sample t-test with $n = 3, \alpha = 0.05$ and $1-\beta = 0.8$?
4. Evaluate the power of a Wilcoxon test in comparison to a t-test using the simulation method.

7 Correlation and regression

7.1 Overview

Correlation and regression analysis serve for answering questions concerning the dependency of two or more random variables. Despite the fact that both methods are usually discussed together and are often executed in one step with a certain software, they have some important differences:

- *Correlation analysis* tests, if there is a dependency *at all* and how strong it ist (*significance test*).

  Given two samples, in correlation analysis both will be regarded as random variables (model II), i.e. there is no distinction between an independent and a dependent one. As a measure of the dependency the correlation coefficient ρ can be used. It is estimated by r.

- *Regression analysis* tries to describe the dependency by means of functional relationships.

  Normally, model I is assumed, that distinguishes between dependent and independent variables, i.e. it is assumed that the "independent" variables are fixed and without error. Creating a calibration curve for a photometer (as used in chemical analytics), as an example, is based on the assumption that the weight of the chemical has distinct levels and all errors in the analysis (such as reaction time, impurities, measurement errors of the photometer, and even the weighing errors) appear as errors of the dependent variable (namely the extinction).

7.2 Correlation analysis

7.2.1 Pearson's product-moment correlation coefficient

PEARSON's product-moment correlation is the "ordinary" correlation coefficient, as we all hopefully know it. With its help it can be tested whether two variables show a *linear* dependency.

Calculation:

$$r_P = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}}$$

or, more favourable:

$$r_P = \frac{\sum X_i Y_i - \sum X_i \sum Y_i / n}{\sqrt{(\sum X_i^2 - (\sum X_i)^2/n)(\sum Y_i^2 - (\sum Y_i)^2/n)}}$$

The correlation coefficient always ranges from $-1 \leq r_P \leq +1$ and it is interpreted as follows:

0	no correlation
$+1$ or -1	strictly functional positive or negative dependency
$0 < \|r_P\| < 1$	positive or negative correlation

In R the correlation coefficient for two random variables x und y can be calculated like follows:

```
x <- c(1, 2, 3, 4, 5, 4, 3)
y <- c(1, 3, 4, 4, 6, 7, 8)
cor(x, y)
```

```
[1] 0.677408
```

and a significance test (null hypothesis $\rho = 0$) can be performed easily, too:

```
cor.test(x, y)
```

```
        Pearson's product-moment correlation

data:  x and y
t = 2.0592, df = 5, p-value = 0.09453
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.1544282  0.9472485
sample estimates:
     cor
0.677408
```

In order to determine it "by hand" there are various possibilities, e.g.:

1. consulting tables of so-called critical values, see table 7.1,
2. approximation with the help of the t-quantile:

$$\hat{t}_{\alpha/2;n-2} = \frac{|r_P|\sqrt{n-2}}{\sqrt{1-r_P^2}}$$

3. or an F-test, see for example SACHS (1992).

The purpose of the significance test is to distinguish random from non-random (significant) correlations. The more measurements were taken, the smaller correlations can be detected to be significant. If only two pairs of values exist, we cannot test for significance, as there is always a straight line between two points and it is said that there are *no degrees of freedom* (d.f.).

Problematic cases

While calculating r_P is always allowed, the test requires bivariate normal distribution, continuous random variables, independent pairs of observations and a dependency that is linear. Other monotonous, nonlinear dependencies may bias r_P as do non-normality and outliers. Thus, a graphical control it is always advisable. If necessary, a transformation (e.g. a logarithm) should be applied.

Table 7.1: Some critical values (r_{crit}) for the Pearson correlation coefficient (null hypothesis $H_0 : \rho = 0$)

n	d.f.	t	r_{krit}
3	1	12.706	0.997
5	3	3.182	0.878
10	8	2.306	0.633
20	18	2.101	0.445
50	48	2.011	0.280
100	98	1.984	0.197
1000	998	1.962	0.062

7.2.2 Spearman's rank correlation coefficient

In contrast to PEARSON's product-moment correlation SPEARMAN's rank correlation tests for any monotonic increasing or decreasing dependency, regardless if it is linear or not. Instead of the actual values ranks are used, so that r_S can be calculated with:

$$r_S = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

with d_i being the difference between ranks of a sample pair $d_i = x_i - y_i$. Here, $-1 \leq r_S \leq +1$ applies too. For testing purposes, a table of critical values can be used again (Table 7.2).

Table 7.2: Critical values for Spearman's rank correlation coefficient, taken from GRIMM and RECKNAGEL (1985)

α	4	5	6	7	8	9	10	11	12
0.05	1.000	0.900	0.829	0.745	0.690	0.683	0.636	0.609	0.580
0.01			0.943	0.893	0.857	0.817	0.782	0.754	0.727

Another test (for $N > 10$) uses the t distribution:

$$\hat{t}_{1-\frac{\alpha}{2};n-2} = \frac{|r_S|}{\sqrt{1-r_S^2}}\sqrt{n-2}$$

Example: Calculating r_S by hand:

x	y	R_x	R_y	d	d^2
1	2.7	1	1	0	0
2	7.4	2	2	0	0
3	20.1	3	3	0	0
4	500.0	4	5	-1	1
5	148.4	5	4	+1	1
					2

$$r_S = \frac{6 \cdot 2}{5 \cdot (25-1)} = \frac{12}{120} = 0.9$$

For comparison: $r_P = 0.58$

Hints:

If there are many ties, mid-level ranks have to be formed for identical values. Afterwards, r_S is estimated by calculating the r_P of the ranks.

Advantages of r_S:

- independent from the type of distribution,
- tests for any *monotonic* correlation,
- is not much influenced by outliers.

Disadvantages:

- a certain loss of information due to ranking,
- no information about the type of dependency,
- no direct link to the coefficient of determination.

In Conclusion, r_S is highly recommended, especially at the beginning of an analysis.

7.2.3 Estimation and testing of r_S with R

```
x <- c(1,2,3,4,5,4,3)
y <- c(1,3,4,4,6,7,8)
cor.test(x,y, method="spearman")
```

```
        Spearman's rank correlation rho

data:  x and y
S = 21.0627, p-value = 0.1343
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.6238795
```

In case of ties (doubled values), R will output a warning, that it cannot compute exact p-values with ties . In such cases the rank correlation coefficient can be estimated using PEARSON's correlation coefficient of ranks [1].

```
cor.test(rank(x), rank(y))
```

[1] One would simply write: "Because of the presence of ties the rank correlation coefficient war estimated with the help of PEARSON's correlation coefficient of ranks."

```
        Pearson's product-moment correlation

data:  rank(x) and rank(y)
t = 1.785, df = 5, p-value = 0.1343
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.2436494  0.9368085
sample estimates:
      cor
0.6238795
```

In the example above both methods give identical results and no significant correlation.

7.2.4 Multiple correlation

Sometimes, one wants to account for dependencies of one variable on several others simultaneously, e.g. the dependence of the chlorophyll concentration (Chl_a) in a lake according to $Chl_a = f(X_1, X_2, X_3, X_4, \dots)$, with X_i being the biomass of the ith phytoplankton species.

In general, a distinction can be made between the multiple correlation coefficient and the partial correlation coefficient. Multiple correlation analysis appears to be very powerful at first sight, but in practice there are often serious difficulties, for example:

1. if the independent variables are correlated with each other (multicollinearity) the multiple r is biased.
2. Nonlinearities are even harder to handle in multiple regression than they are in the two-sample case.

As a way out it is recommended to first provide oneself with an overview using multivariate methods (e.g. NMDS), and then to approach the problem with more insight into the processes involved and a good portion of instinct. HARRELL (2001) or QIAN (2009) provide more information on this topic.

7.3 Linear regression

Regression analysis aims to describe the dependency between two (*simple* regression) or several variables (*multiple* regression) by means of a linear or nonlinear function. In case of not only multiple independent variables (x-variables), but also multiple dependent variables (y-variables) this is referred to as *multivariate* regression. In contrast to correlation analysis, which tests for the existence of a relationship, regression analysis focuses on the quantitative description of this relationship by means of a statistical model. Besides that, there are also differences in the underlying statistical assumptions. While correlation analysis assumes that all variables are random containing an error (model II), regression analysis generally makes a distinction between independent variables (without error) and dependent variables (with errors), which is called model I (for more details see e.g. SACHS, 1992; ZAR, 1996).

7.3.1 Basic principles

In general, linear models form the basis for a variety of statistical methods, e.g. linear regression or analysis of variance. The equation of a multiple linear model is:

$$y_i = \alpha + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p} + \varepsilon_i \tag{7.1}$$

with every value of the dependent variable y_i being calculated from the corresponding value of the independent variable $x_{i,j}$, the model parameters α (y-intercept) and β_j (regression coefficient or slope) and a normally distributed error term ε_i with mean value 0.

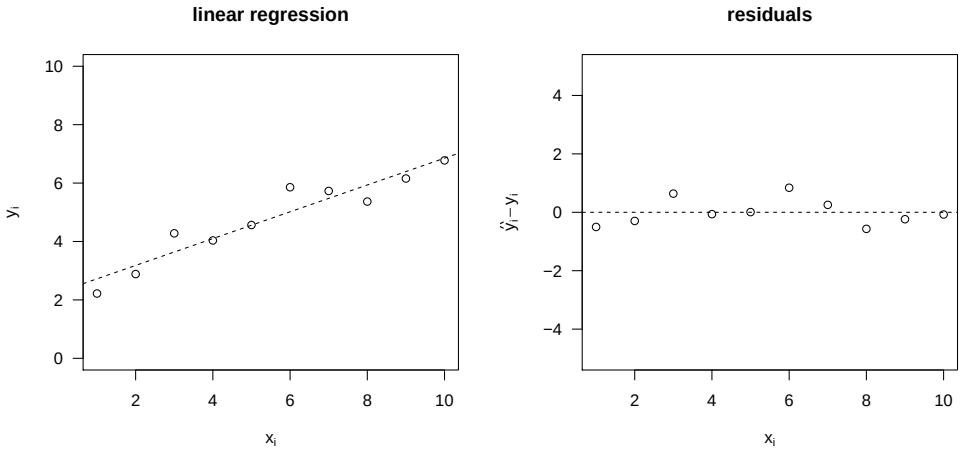


Figure 7.1: Linear regression (left) and residuals (right).

A simple regression model for example (best fit straight line, Fig. 7.1) with one dependent variable y and only one independent variable x can be written as:

$$\hat{y}_i = a + b \cdot x_i \tag{7.2}$$

In this context $\hat{y}$ denotes the estimated values of the dependent variable, i.e. the values on the regression line, while a and b are the estimates of the true model parameters α and β.

In order to estimate a and b an optimization criterion is needed. In most cases the sum of squared deviations $SQ = \sum(\hat{y}_i - y_i)^2$ will be used. The minimum $SQ \rightarrow min$ can be obtained by setting the first partial derivate of SQ with regard to the parameters a and b to zero:

$$\frac{\partial \sum(\hat{y}_i - y_i)^2}{\partial a} = \frac{\partial \sum(a + b \cdot x_i - y_i)^2}{\partial a} = 0 \tag{7.3}$$

$$\frac{\partial \sum(\hat{y}_i - y_i)^2}{\partial b} = \frac{\partial \sum(a + b \cdot x_i - y_i)^2}{\partial b} = 0 \tag{7.4}$$

and after solving the resulting system of equations one receives the determination equations for a and b:

$$b = \frac{\sum x_i y_i - \frac{1}{n}(\sum x_i \sum y_i)}{\sum x_i^2 - \frac{1}{n}(\sum x_i)^2} \tag{7.5}$$

$$a = \frac{\sum y_i - b \sum x_i}{n} \tag{7.6}$$

Therefore, the best fit line can be directly calculated from the given values of x and y via a linear system of equations. For that reason it is termed an **analytical** solution.

Residuals and coefficient of determination

The difference $\varepsilon_i = y_i - \hat{y}_i$ between the corresponding measured and the predicted values of the dependent variable are called residuals (fig. 7.1 right) and their variance is called "residual variance". It is obvious that the residuals are less scattered than the original measurements. This means that a certain fraction of the original variation is now contained within the regression line, or, in other words, explained by the model. That fraction of the original variance now explained by the model is the coefficient of determination r^2, which in the case of the *linear* regression equals the square of Pearson's correlation coefficient.

From this follows in general:

$$r^2 = \frac{\text{explained variance}}{\text{total variance}} = \frac{\text{total variance} - \text{residual variance}}{\text{total variance}} \tag{7.7}$$

or

$$r^2 = \frac{s_y^2 - s_{y_i - \hat{y}_i}^2}{s_y^2} \tag{7.8}$$

and in the case of a linear regression

$$r^2 = \frac{\sum(\hat{y} - \bar{y})^2}{\sum(y - \bar{y})^2} \tag{7.9}$$

The coefficient of determination always ranges from 0 to 1 and a value of $r^2 = 0.85$, for example, means that 85% of total variance is explained by the model.

Significance test

Besides the numerical value of the coefficient of determination and graphical control of the results a significance test is always needed. The larger the sample size n, the smaller values of r^2 can be found the be significant. Significance of the slope can be examined with the help of an F-test:

$$\hat{F}_{1;n-2;\alpha} = \frac{s_{explained}^2}{s_{residual}^2} = \frac{r^2(n-2)}{1-r^2} \tag{7.10}$$

Confidence intervals

Confidence intervals can be estimated for the parameters (e.g. a and b), for the regression line itself and for future observations Y_i at X_i (prediction interval). The confidence intervals of the parameters a and b can be calculated easily with the help of their standard errors s_a and s_b

$$a \pm t_{1-\alpha/2,n-2} \cdot s_a \tag{7.11}$$

$$b \pm t_{1-\alpha/2,n-2} \cdot s_b \tag{7.12}$$

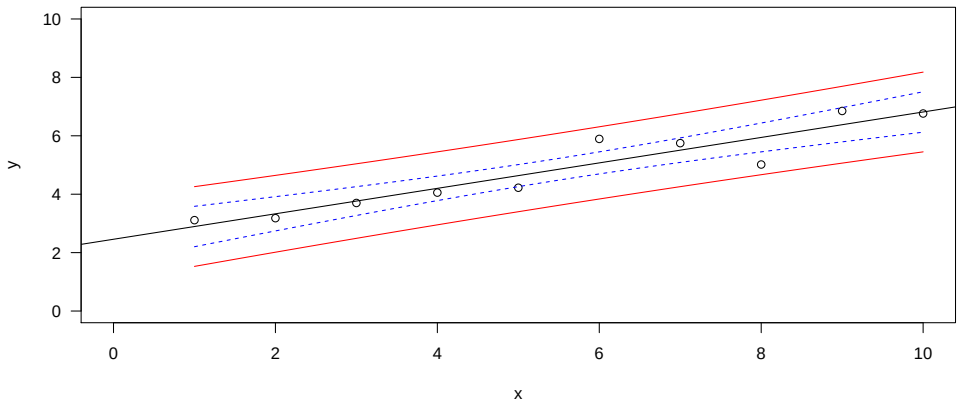


Figure 7.2: Linear regression with confidence interval of the regression line (dashed) and prediction interval for future observations (solid).

The **confidence interval** (as in Fig. 7.2) represents the limits in which the regression is found with a certain probability (e.g. 95%). The hyperbolic shape is comprised of a shift (confidence interval of parameter a) and a rotation (confidence interval of b).

In contrast, the **prediction interval** has a completely different meaning. It tells us within which limits a predicted value y is to be expected for a given x. Thus, whereas the confidence interval characterizes the mean course of the regression line, the prediction interval makes a statement about expected single values.

According to (SACHS, 1992) or (ZAR, 1996) these intervals can be obtained from the total sum of squares Q_x for the x values like follows:

$$Q_x = \sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - \frac{1}{n}\sum_{i=1}^{n} x_i \tag{7.13}$$

and the standard error of the prediction (the standard deviation of the $\hat{y}$ values) for a given x ($s_{y|x}$, to be read as: “s y for x”):

$$s_{y|x} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n-2}} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - a - b \cdot x_i)}{n-2}} \tag{7.14}$$

We first calculate the standard deviation for an estimated mean value $\underline{\hat{y}}$ and the standard deviation for predicted single value $\hat{y}_{\cdot}$ at x

$$s_{\underline{\hat{y}}} = s_{y|x} \cdot \sqrt{\frac{1}{n} + \frac{(x-\bar{x})^2}{Q_x}} \tag{7.15}$$

$$s_{\hat{y}_{\cdot}} = s_{y|x} \cdot \sqrt{1 + \frac{1}{n} + \frac{(x-\bar{x})^2}{Q_x}} \tag{7.16}$$

and thus receive the confidence interval of the regression line as:

$$\hat{y} \pm \sqrt{2 \cdot F_{(2,n-2)} \cdot s_{\underline{\hat{y}}}} \tag{7.17}$$

and the prediction interval as:

$$\hat{y} \pm t_{(n-2)} s_{\hat{y}.} \tag{7.18}$$

with F and t being the appropriate quantiles of the F- and t-distribution.

Assumptions of the linear regression

Only if the following prerequisites are met, the parameters (a, b) can be estimated without bias and the significance test will be reliable (SACHS, 1992):

1. Model I is assumed (x is defined, y is a random variable).
2. For every value of x, y is a random variable with mean $\mu_{y|x}$ and variance $\sigma^2_{y|x}$.
3. y values are independent and identically distributed (no autocorrelation and homogeneous variance σ^2)
4. For multiple regression all x_j must not be correlated with each other (multicollinearity condition).
5. The residuals e and the y values need to be normally distributed.

In this context it is especially important that the variance of the residuals is homogenous along the whole range of x, i.e. the variation of the residuals must not increase or decrease or show any systematic pattern.

Further information on the basics and the execution of regression analysis can be found in general statistics textbooks like KÖHLER *et al.* (2002), SACHS (1992) or ZAR (1996), e.g. about the calculation in confidence intervals, significance tests for a and b or about alternatives to the method of least squares.

7.3.2 Implementation in R

Model formula in R

R has a special formula syntax for describing statistical models. A simple linear model (y versus x) can be described as:

```
y ~ x + 1
```

or shorter, because "+ 1" can be omitted:

```
y ~ x
```

In contrast to the former, "-1" means a regression model with zero intercept:

```
y ~ x - 1
```

The complete formula syntax and further examples are contained within the R documentation or can be found in textbooks, for example in VENABLES *et al.* (2001).

A simple example

First, we create a vector containing 10 `x` values:

```
x <- 1:10
```

and a vector of `y` values that depend on `x`:

```
y <- 2 + 0.5 * x + rnorm(x)
```

where `rnorm(x)` returns a vector with random numbers with the same length as there are values in `x`. The random numbers come from a standard normal distribution with mean zero and standard deviation one.

First, we plot the data with:

```
plot(x, y)
```

Then, a linear model can be fitted with the *linear model* function `lm()`:

```
reg <- lm(y ~ x)
```

The R object delivered by `lm` (called `reg` in our case) now contains the complete results of the regression, from which significance tests, residuals and further information can be extracted by using specific R functions. Thus,

```
summary(reg)
```

```
Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-1.33483 -0.21043 -0.03764  0.59020  1.04427

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.77581    0.59426   1.306 0.228008
x            0.64539    0.09577   6.739 0.000147 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8699 on 8 degrees of freedom
Multiple R-squared:  0.8502,        Adjusted R-squared:  0.8315
F-statistic: 45.41 on 1 and 8 DF,  p-value: 0.0001467
```

gives us the most important results of the regression, e.g. the coefficients, with *intercept* being equivalent to parameter *a* and the dependence on *x* being equivalent to the slope parameter *b*. The coefficient of determination is found as *Multiple R-squared*. Besides that also standard errors of the parameters and significance levels are printed.

The regression line can be added to the plot with the universal line drawing function of R, `abline` that directly accepts a linear model object (`reg`) as its argument:

```
abline(reg)
```

With the help of `predict` the y values belonging to given x values can be calculated:

```
x1 <- c(1.5, 2.5, 7.5)
y1 <- predict(reg, data.frame(x=x1))
points(x1, y1, col="green")
```

In doing so, one has to keep in mind that `predict` expects the new data in a data frame with the **same variable names** as in the initial call of `lm`. Using `predict` we can also create confidence and prediction intervals (see Fig. 7.2):

```
x <-1:10
y <- 2 + 0.5 * x + 0.5 * rnorm(x)
reg <- lm(y ~ x)
plot(x,y, ylim=c(0,10), xlim=c(0,10))
abline(reg)
newdata <- data.frame(x=seq(min(x), max(x), length=100))
conflim <- predict(reg, newdata=newdata, interval="confidence")
predlim <- predict(reg, newdata=newdata, interval="prediction")
lines(newdata$x, conflim[,2], col="blue", lty="dashed")
lines(newdata$x, conflim[,3], col="blue", lty="dashed")
lines(newdata$x, predlim[,2], col="red")
lines(newdata$x, predlim[,3], col="red")
```

There are numerous additional possibilities, for example `coef(reg)`, which delivers the coefficients, and `residuals(reg)`, which delivers the residuals for futher use. The function `plot(reg)` can be used for getting diagnostic plots. Finally, `str(reg)` shows the elements contained within `reg`, that can be used for further calculations.

7.3.3 Exercise: Relationship between chlorophyll and phosphate

Problem

The file `oecd.txt` contains the mean annual values of total phosphate (TP $\mu g l^{-1}$) and chlorophyll a (CHLa in $\mu g l^{-1}$) from 92 lakes digitized from a figure in VOLLENWEIDER and KEREKES (1980)[2]. The data are to be visualised and a suitable regression line is to be fit.

Solution

First, the data are read from a text file, in which the first row holds the variable names (`header=TRUE`). Using `attach(mydata)` the variables within the data frame `mydata` can be accessed directly.

```
dat <- read.table("oecd.txt", header=TRUE)
plot(dat$TP, dat$CHLa)
reg <- lm(CHLa ~ TP, data = dat)
abline(reg)
summary(reg)
```

[2]because some data points coincide, two of the original 94 lakes are missing.

The plot indicates that the assumptions of the linear regression were heavily violated. Thus, both the the x axis and the y axis are to be transformed. As in the original publication we will us a logarithmic transformation. In accordance with most other programming languages, in R `log()` stands for the natural logarithm. In order to fit a line to the transformed data, the logarithm can be applied directly within `lm`, but it is also possible to transform the data beforehand:

```
logTP   <- lof(TP)
logCHLa <- log(CHLa)
plot(logTP, logCHLa)
reg <- lm(logCHLa ~ logTP)
abline(reg)
summary(reg)
```

7.3.4 Additional exercises

1. Convert the equation $\ln(CHLa) = a + b \cdot \ln(TP)$ into the form $CHLa = a' \cdot TP^b$.
2. Remove the data in which not phosphorous (P), but nitrogen (N) or light (I) are the limiting factors and recalculate the regression. Plot all regression lines together into one plot.
3. Calculate the coefficient of determination using equation 7.8 and compare the result to the output of R.

7.4 Nonlinear regression

7.4.1 Basic concepts

Many seemingly nonlinear functions can be fitted using linear techniques, e.g. polynomials or periodic (sine and cosine) functions, and some others can be fitted by using transformations of the variables, e.g. logarithms or reciprocals. So, for instance:

$$y = a \cdot x^b \tag{7.19}$$

is equivalent to the function

$$\ln(y) = \ln(a) + b \cdot \ln(x) \tag{7.20}$$

It has to be noted, however, that carrying out such a linearisation transforms the residuals as well. Sometimes, this is correct and necessary to meet the assumptions of linear regression, as in the above example of the dependency of chlorophyll from total phosphate. However, care has to be taken in such cases and at least graphical diagnostics of the transformed data is required. Unfortunately, common spreadsheet programs hide this from the user, and this can result in misleading and essentially wrong results.

Typical problematic examples are fits of Monod functions (see below) to reciprocally transformed data, which often leads to biased estimates. Therefore, it is highly recommended to fit such functions directly using nonlinear methods.

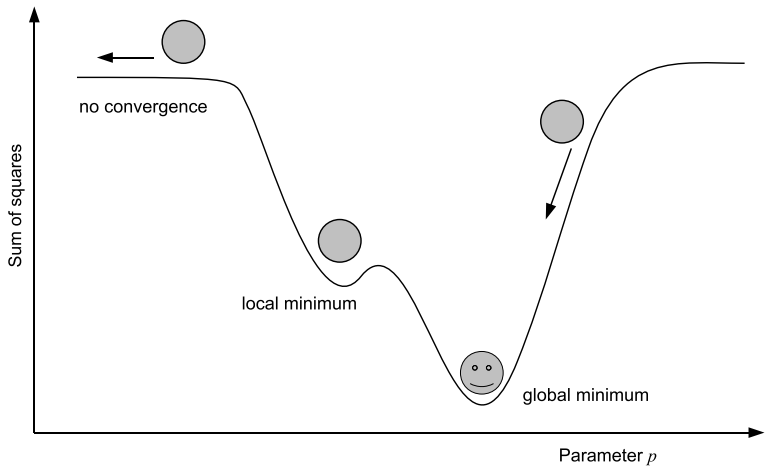


Figure 7.3: Numerical optimization problem: The goal is to find the global minimum of a function by means of the least squares between measurements y_i and predictions $\hat{y}_i$. In the case of two or more parameters p_i one gets a multidimensional "optimization landscape".

Nonlinear regression with numerical optimization methods

In all cases where an analytical solution of the partial derivatives is unknown or not existent and if linearization would lead to a violation of the regression assumptions, a numerical optimization method has to be used.

Similar to linear models, the quality of the fit is usually measured by the sum of least squares SQ:

$$SQ = \sum (y_i - f(\mathbf{x}_i, \mathbf{p}))^2 = \min! \tag{7.21}$$

with y being the dependent variable, $\mathbf{x}$ the matrix of one or more independent variables and $\mathbf{p}$ the parameter vector.

The theory and practice of optimization methods is a very broad field and many different optimization methods are now available in R, but a detailed description of this is beyond of the scope of this tutorial. More information can be found in the CRAN optimization task view at `http://cran.r-project.org/web/views/Optimization.html`.

The individual methods differ with respect to several aspects, for instance with respect to the necessity of using derivatives or whether the search can be performed without the need for derivatives, or regarding their efficiency and behavior in solving numerically difficult problems.

In Newton-type methods (e.g. Gauss-Newton algorithm, Newton-Raphson method, Quasi-Newton method), second partial derivatives (Hessian matrix) are required or estimated internally using numerical methods. Due to this such methods are very efficient an converge quickly.

In gradient or simplex methods the direction of the steepest descent is followed. The methods are less efficient, but nevertheless recommendable, e.g. for finding starting values in case of the existence of local minima.

In very difficult cases (e.g. for the calibration of complex models) stochastic methods (Monte Carlo methods) or so-called "evolutionary strategies" or "genetic algorithms" can be helpful.

The general availability of fast computers and powerful algorithms in statistics packages and spreadsheet software has led to the situation that nowadays optimization methods can easily be applied in many cases. Nevertheless, a certain amount of caution and sensitivity of the user are always indicated.

Choosing a suitable model

Choosing a suitable regression model (regression function) for a given problem or data set cannot be expected from an optimization algorithm, but is up to the user. In the ideal case, physical, chemical, biological or other theoretical considerations would lead to a mechanistic model (BATES and WATTS, 1988). Naturally, the choice of the model is the task of the user that is most familiar with the subject. Therefore, for natural scientists as we are, model creation is an essential part of our business.

In making the correct choice of a regression model, experience, literature studies and a suitable function library will help. The chosen function should be mechanistically justified, but nevertheless as simple as possible. Very often, a good regression function (e.g. the Monod or the logistic function) is just an analytical solution of a differential equation model. Beyond that, it is possible to fit differential equation models directly.

Furthermore, one always begins with a simple model and builds it up stepwise by adding further terms and parameters where appropriate.

A serious problem of nonlinear models comes up in situations where some of the individual parameters of a model depend on each other too strongly, thus compensating each other. Consider for example the following trivial case:

$$y = a + \frac{b}{c} \cdot x \tag{7.22}$$

where it is obvious that b and c cannot be determined at the same time, as simultaneously increasing values of a and b will keep the quotient constant. In such a case the parameters are said to be **unidentifiable**.

Often, the relationship is not that obvious or less strict. Broadly speaking, the identifiability depends upon the number of parameters, the number of data points, the variance of the residuals and the correlation between the parameters, i.e., how strictly they depend upon each other. If some or all the parameters are hard or impossible to determine, the model has to be simplified and parameters need to be aggregated. Besides that, the data set may be enlarged, measurement errors reduced or the parameter in question may be determined separately in an additional experiment. Moreover, the R package FME (SOETAERT and PETZOLDT, 2010) contains tools for identifiability analysis, e.g. calculating identifiability indices according to BRUN *et al.* (2001).

Determination of starting values

All numerical methods have in common, that they need either starting values or parameter ranges to be searched through. Then, the optimization algorithm tries to find a global minimum of a given quality criterion (fig. 7.3) and stops, when a minimum was found or prints a warning message if no convergence has been achieved after a given number of iterations. The convergence and tolerance parameters can be set by the user.

Often, there is not only one global minimum, but also additional local minima, and there is basically no warranty that the global minimum can be found within finite computation time. For this reason it is sometimes common to perform multiple optimization runs with different starting values. Suitable starting values are

often found by thought, manual trial and error, or approximations, for example via a linearizing transformation. For some functions specific methods are available which determine starting values automatically.

Evaluation of the model fits

In nonlinear regression, graphical validation of model results is of particular importance. Besides plotting the measured data and best-fit curves the residuals should be examined. Here, no pattern or systematic deviation should be visible and the variance has to be homogeneous.

An important criterion is the coefficient of determination, which in the nonlinear case cannot simply be derived from the square of the correlation coefficient. Instead, the general approach according to equation 7.8 is applicable, thus:

$$r^2 = 1 - \frac{s_\varepsilon^2}{s_y^2} \tag{7.23}$$

in which s_ε^2 is the variance of the residuals ($\varepsilon = \hat{y} - y$) and s_y^2 the variance of the dependent variable. In certain cases the coefficient of determination may become negative. That means that the residuals possess a greater variance than the original measured values. The reason for this is a failed optimization. In diagnostic plots this can instantly be recognized, as the fitted curve lies besides the points. In order to solve the problem one can try to repeat model fitting with new starting values or one may consider to choose another function type for the regression equation.

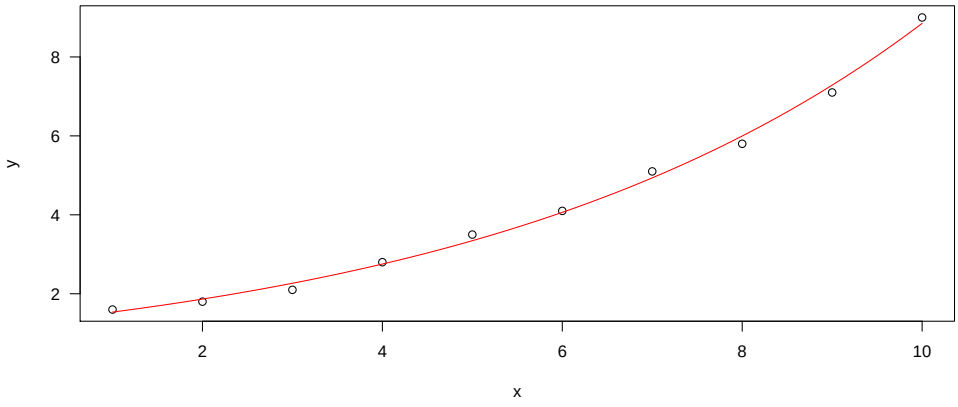


Figure 7.4: Exponential relationship.

7.4.2 Implementation in R

In R there are several packages for optimization of nonlinear problems. For the nonlinear regression with the method of least squares the function `nls` can used, which by default employs a Gauss-Newton algorithm. The function `nls` expects a regression equation as well as the data and starting values given as lists.

The regression equation which is to be fitted can be given as so called “model formula”, e.g. an exponential growth function as `y ~ a * exp(b * x)`. It has to be noted that in nonlinear regression (in contrast to

linear regression) all symbols (e.g. ^) are being interpreted as "arithmetic". Even more general is the use of an R-function for the model equation:

```
f <- function(x, a, b) {
  a * exp(b * x)
}
```

A third possibility is using predefined "autostart functions", in which the starting values are determined automatically, e.g. `SSlogis` for the logistic model or `SSmicmen` for the Michaelis-Menten model.

The procedure is to be explained using the example of an exponential model $y = a \cdot \exp(bx)$. This model is very universal and can be used to describe exponential growth, but is also found in the laws of 1st order decay of substances or light absorption. We begin by defining an R-function `f`, provide the data (`x`, `y`) and assign the start values for the parameters to `pstart`. With the optional argument `trace=TRUE` we allow interim results of the optimization algorithm to be displayed. Finally, the results are shown on the screen and plotted. The evaluation of the equation (for the plot, for instance) can be performed using `predict`:

```
f <- function(x, a, b) {a * exp(b * x)}
x <- 1:10
y <- c(1.6, 1.8, 2.1, 2.8, 3.5, 4.1, 5.1, 5.8, 7.1, 9.0)
pstart <- list(a = 1, b = 1)
plot(x, y)
aFit <- nls(y ~ f(x, a, b), start = pstart, trace = FALSE)
x1 <- seq(1, 10, 0.1)
y1 <- predict(aFit, list(x = x1))
lines(x1, y1, col = "red")
```

If `trace=TRUE` is set, one could watch how the algorithm approximates the parameters iteratively. As a result we receive the parameter values with details on their significance:

```
summary(aFit, correlation=TRUE)

Formula: y ~ f(x, a, b)

Parameters:
  Estimate Std. Error t value Pr(>|t|)
a 1.263586   0.049902   25.32 6.34e-09 ***
b 0.194659   0.004716   41.27 1.31e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1525 on 8 degrees of freedom

Correlation of Parameter Estimates:
  a
b -0.97

Number of iterations to convergence: 13
Achieved convergence tolerance: 2.504e-08
```

The option `correlation=TRUE` enables the output of correlation coefficients between the model parameters (a and b in our case).

In our example the correlation between parameters is about -0.97, a relatively high absolute value. This means that a certain interaction between the parameters is present, but as the residuals are small (which means our data are "good") this was no big problem. By no means this correlation coefficient must be confused with the correlation coefficient (or rather, the coefficient of determination) of the regression model itself, which can be calculated with the equation 7.23:

```
1 - var(residuals(aFit))/var(y)
```

```
[1] 0.9965644
```

Thus, the nonlinear coefficient of determination amounts to 0.9966, i.e. the exponential model explains 99.66% of the variance of y.

7.4.3 Exercise

Problem

On the occasion of a practical course for students of the TU Dresden, conducted at the Institute of Freshwater Ecology and Inland Fisheries, Department Limnology of Stratified Lakes, in September 2001, the heterotropic potential (glucose intake rate IR in μg C $L^{-1}h^{-1}$) of bacteria was determined in dependence of substrate availability (glucose concentration S, in μgL^{-1}). In a sample from lake Fuchskuhle, taken in a water depth of 2.5 m, the following values were measured:

```
# substrate ug C/L
S  <- c(25, 25, 10, 10, 5, 5, 2.5, 2.5, 1.25, 1.25)
# intake rate ug C /(L*h)
IR <- c(0.0998, 0.0948, 0.076, 0.0724, 0.0557,
        0.0575, 0.0399, 0.0381, 0.017, 0.0253)
```

What we are interested in are the parameters K and V_m of a Michaelis-Menten kinetics:

$$IR = \frac{V_m \cdot S}{K + S} \tag{7.24}$$

Solution 1

Frequently, the Michaelis-Menten equation is fitted using linearisations, but in most cases a nonlinear fit is preferable. We modify the `nls` example from above and receive:

```
f <- function(S, Vm, K) {
  Vm * S/(K + S)
}
pstart <- list(Vm = max(IR), K = 5)
aFit   <- nls(IR ~ f(S, Vm, K), start = pstart, trace = TRUE)
plot(S, IR, xlim = c(0, max(S)), ylim = c(0, max(IR)))
x1 <- seq(0, 25, length=100)
```

```
lines(x1, predict(aFit, list(S = x1)), col = "red")
summary(aFit)
Rsquared <- 1 - var(residuals(aFit))/var(IR)
paste("r^2=", round(Rsquared, 4))
```

In order to obtain a smooth curve we once again use a vector `x1` containing 100 values ranging from 0 to 25.

Solution 2

There is an even simpler solution, the application of the in R predefined autostart model `SSmicmen`, so that we can avoid the need for defining the model ourselves and, even more useful, the need to specify starting values:

```
aFit <- nls(IR ~ SSmicmen(S, Vm, K), trace=TRUE)
plot(S, IR, xlim=c(0, max(S)), ylim=c(0, max(IR)))
x1 <- seq(0, 25, length=100)
lines(x1, predict(aFit,list(S=x1)), col="red")
summary(aFit)
paste("r^2=", round(1 - var(residuals(aFit))/var(IR), 4))
```

7.4.4 Additional exercises

1. Linearise the implementation example (exponential model), fit a linear model and compare the results.
2. Fit a suitable model to the data of a batch experiment with the *Microcystis aeruginosa* stem PCC 7806 (JÄHNICHEN *et al.*, 2001):

```
# time (days)
x <- c(0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20)
# cells (per mL)
y <- c(0.88, 1.02, 1.43, 2.79, 4.61, 7.12,
     6.47, 8.16, 7.28, 5.67, 6.91) * 1e6
```

8 Time Series Analysis

Time series analysis poses a number of typical problems which can be handled using specific methods. The following chapter demonstrates a selection of basic procedures. For more detailed yet comprehensible information chapter 6 from KLEIBER and ZEILEIS (2008) is to be recommended, or alternatively a specific book on time series analysis, e.g. SHUMWAY and STOFFER (2006).

In accordance with the convention found in most textbooks on time series analysis the variables will not be called x and y hereafter. The independent variable will be referred to as t and the dependent variable as x.

8.1 Stationarity

Stationarity of time series is one of the central concepts in time series analysis. A stochastic process (x_t) is called a strictly (or strong) stationary process, when the distribution of (x_{s+t}) is independent from the index s. Weakly or wide-sense stationary random processes only require that 1st and 2nd moments (i.e. mean value, variance and covariance) do not vary with respect to time.

Two Example Data Sets

To clarify the concept of stationarity we will compare the following two time series:

$$x_t = \beta_0 + \beta_1 t + u_t \tag{8.1}$$

$$x_t = x_{t-1} + c + u_t \tag{8.2}$$

Here t stands for time, β_0, β_1 and c are constants and u_t is a random process (so-called *white noise*). One can discover that the time series following the equation 8.1 resembles a linear regression model, whereas the time series following the equation 8.2 corresponds to a *random walk* with a drift constant c.

For illustration of different types of stationarity we will generate two example time series:

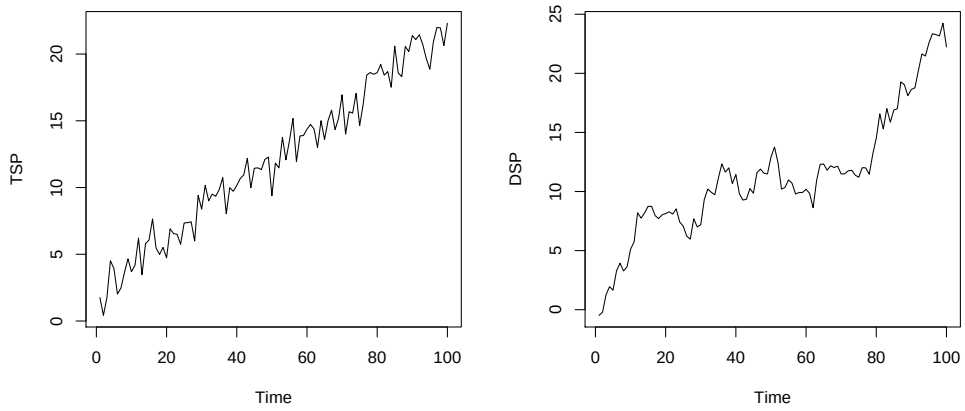
```
set.seed(1237) # makes random number generator reproducible
time <- 1:100
## linear regression model
TSP <- 2 + 0.2 * time + rnorm(time)
## random walk
DSP <- numeric(length(time))
DSP[1] <- rnorm(1)
for (tt in time[-1]) DSP[tt] <- DSP[tt-1] + 0.2 + rnorm(1)
```

In R there are specific classes for handling time series data. The most important ones are `ts` for equidistant data and `zoo` for non-equidistant data. A vector can be converted into an equidistant time series using the function `ts`:

```
TSP <- ts(TSP)
DSP <- ts(DSP)
```

Doing so only the x-values of the time series are saved. Time itself is contained only in the form of beginning, end and frequency. It can be extracted with the utility function `time()`. Another very useful function is `tsp()` (*time series properties*).

```
par(mfrow=c(1,2))
plot(TSP)
plot(DSP)
```



It can be easily seen that both time series seem to have some kind of trend. Just, how can we test if that trend is significant? The most obvious method may be a linear regression of x_t against time, but is that correct?

A Simulation Experiment

In a simulation experiment, linear trends for the time series according to equations 8.1 and 8.2, are to be tested for significance. For simplification we define two functions for generating time series of type "TSP" and "DSP" with user-specific parameters β_0, β_1 and c:

```
genTSP <- function(time, beta0, beta1)
  as.ts(beta0 + beta1 * time + rnorm(time))
```

and:

```
genDSP <- function(time, c) {
  DSP <- numeric(length(time))
  DSP[1] <- rnorm(1)
  for (tt in time[-1]) DSP[tt] <- DSP[tt-1] + c + rnorm(1)
  as.ts(DSP)
}
```

Now we will test for the number of significant F-values for the linear regression model using a number of simulations for both types of time series. We set the trend to zero, so the result of the function `countSignif(a)` counts in fact the false positive results within the simulation loop.

```
count.signif <- function(N, time, FUN, ...) {
  a <- 0
  for (i in 1:N) {
    x <- FUN(time, ...)
    m <- summary(lm(x  ~ time(x)))
    f <- m$fstatistic
    p.value <- pf(f[1], f[2], f[3], lower=FALSE)
    # cat("p.value", p.value, "\n")
    if (p.value < 0.05) a <- a + 1
  }
  a
}
```

To some readers of this script the above function may possibly appear to be too complicated. Ultimately, the details (handing over a function `FUN` with optional arguments `...` or calculating the p-value via the distribution function of the F-distribution `pf` [1] are not that crucial here.

More important is what the function does: it simulates many (e.g. 1000) time series using a given function `FUN` and counts the number of significant results with $p < 0.05$. For function `genTSP` the portion of false positives is approximately 5%.

```
Nruns <- 100 # or even better 1000 !!!
count.signif(N=Nruns, time=time, FUN=genTSP, beta0=0, beta1=0) / Nruns
```

```
[1] 0.05
```

In the process `genDSP` that portion is much higher than the expected 5%:

```
count.signif(N=Nruns, time=time, FUN=genDSP, c=0) / Nruns
```

```
[1] 0.91
```

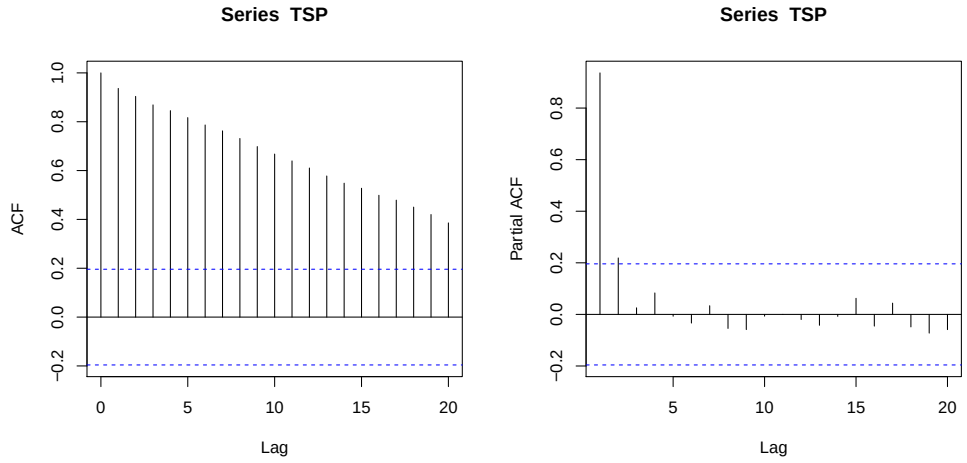
That means that an apparent trend is detected much too often while in reality there is none. This phenomenon is called "spurious regression". The reason for this is that only example "TSP" is a process with a deterministic trend (trend stationary process). The "DSP" series is a so-called difference stationary process, which can be made stationary by differencing (i.e. subtracting successive values) and not by subtracting an (in our case linear) trend.

8.2 Autocorrelation

The correlation between a time series with a time-shifted version of the same time series is called autocorrelation. Usually, the time shift (lag) is being varied and the result is displayed in a tabulary or graphic way (correlogram or autocorrelation plot). The autocorrelation for $lag = 0$ is 1 (one), the other values (with higher lags) express to which amount the value at a given instant x_t depends on its preceding points in time ($x_{t-1}, x_{t-2}, ...$). The dependency can also be indirect, i.e. x_t is dependent on x_{t-2} only because x_{t-1} is dependent on x_{t-2}. When direct dependencies without indirect effects are to be shown, so called partial autocorrelation is used.

[1] see also `https://stat.ethz.ch/pipermail/r-help/2009-April/194121.html`

In the `TSP` data set the autocorrelation plot exhibits a serial dependency of the observations. The partial autocorrelation function however shows that only subsequent values ($lag = 1$) are significantly correlated:

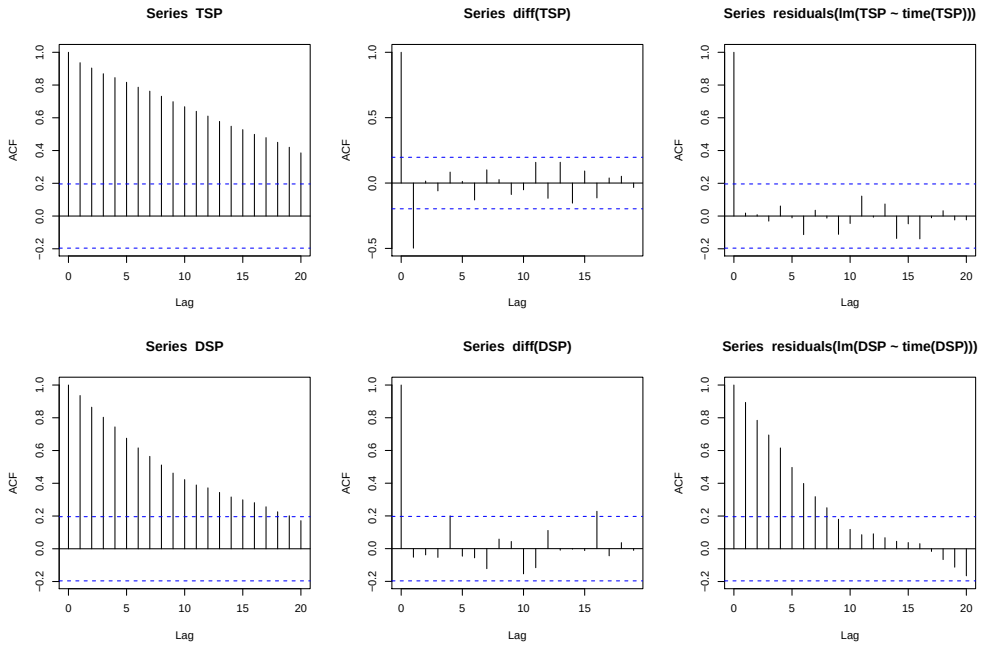


Cross-correlation is used to describe the mutual dependency of two variables, in which the lag can be positive or negative. In a lake, for instance, global radiation determines water temperature, not the other way around.

With the help of typical patterns of the autocorrelation function, different types of time series can be distinguished, e.g. time series of TSP and DSP-type following the equation 8.1 and 8.2 respectively.

The autocorrelation plots of the original time series (left) look very much alike. In the differentiated time series (middle) and the series obtained by subtracting the regression line the differences are easily visible.

```
par(mfrow=c(3,3))
acf(TSP)
acf(diff(TSP))
acf(residuals(lm(TSP~time(TSP))))
acf(DSP)
acf(diff(DSP))
acf(residuals(lm(DSP~time(DSP))))
```



After differencing the time series "TSP" a negative correlation can easily be observed at $lag = 2$ (central figure). As one would expect, once the regression line has been subtracted the resulting residuals are merely white noise (right). In the "DSP" example things are different. Here, differencing results in white noise, while the detrended time series still reveals strong autocorrelation.

8.3 Unit Root Tests

In order to determine whether a time series is of type "DSP" (difference-stationary process) unit root tests can be used. The mathematical theory they are based on cannot be discussed here. However, in real life the ADF-test (augmented Dickey–Fuller test) is used frequently. It is contained within the R-package **tseries**:

```
library(tseries)
adf.test(TSP)

        Augmented Dickey-Fuller Test

data:  TSP
Dickey-Fuller = -4.0145, Lag order = 4, p-value = 0.01134
alternative hypothesis: stationary
```

```
adf.test(DSP)
```

```
        Augmented Dickey-Fuller Test

data:  DSP
Dickey-Fuller = -2.4355, Lag order = 4, p-value = 0.3962
alternative hypothesis: stationary
```

The time series TSP can be made stationary by subtracting a linear trend. This is done automatically by the test.

In the DSP series the null hypothesis (presence of an unit root) cannot be rejected, thus the time series would be regarded as non-stationary. But after differencing there are no objections against stationarity:

```
adf.test(diff(DSP))
```

```
        Augmented Dickey-Fuller Test

data:  diff(DSP)
Dickey-Fuller = -3.9902, Lag order = 4, p-value = 0.01261
alternative hypothesis: stationary
```

The KPSS test (Kwiatkowski-Phillips-Schmidt-Shin test) tests directly for stationarity or trend stationarity:

```
kpss.test(TSP)                 # instationary
kpss.test(TSP, null="Trend")   # stationary after trend removal
kpss.test(DSP)                 # instationary
kpss.test(DSP, null="Trend")   # still instationary
```

8.4 Trend tests

Common trend tests are suitable only for trend-stationary time series, but not for difference-stationary time series, because in these the residuals are autocorrelated. This also holds true for the Mann-Kendall test which is popular in environmental sciences. In its standard formulation it is applicable only for trend-stationary time series.

```
library("Kendall")
MannKendall(TSP)
```

```
tau = 0.894, 2-sided pvalue =< 2.22e-16
```

```
MannKendall(DSP)
```

```
tau = 0.755, 2-sided pvalue =< 2.22e-16
```

8.5 Decomposition into mean, trend and a seasonal component

The traditional approach towards time series analysis is based on the decomposition of time series into different components (classical component model). The most important ones are:

1. trend component,
2. seasonal or cyclical component and
3. stochastic component.

Many of the decomposition methods require normal distribution of the residuals. If this is not the case or, even worse, the variance of a time series changes proportionally with a trend, a transformation might be necessary. For hydrological or biomass data, which frequently show positive skewness (the bulk of values of the density distribution lie left of the mean), a logarithmic transformation is often helpful.

8.5.1 Smoothing methods

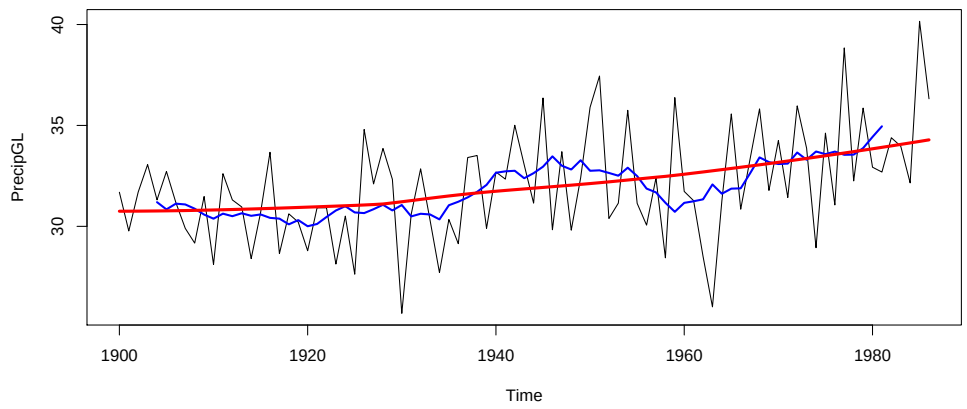
The detection or elimination of a trend can by achieved by fitting curves or by using so-called smoothers. A linear trend, for example, can be detected with a linear regression against time. The residuals then correspond to the time series corrected for the trend. This works not only for trend-stationary time series, but in principle for difference-stationary time series as well. However, as mentioned above, the significance tests that are normally used for linear regression models are likely to give wrong results because of the autocorrelation of the residuals. To sum up: the use of linear regression for trend elimination is fine, but the associated tests may fail, depending on the particular properties of the time series.

Alternatively, trends can be identified by application of moving averages (linear filters), by exponential smoothing or by using so-called "kernel smoothers". Differencing also eliminates trends. To illustrate a linear filter we will use a data set of annual precipitation values from the Great Lakes region from 1900 to 1986, which is contained within the package **Kendall** (McLeod, 2009). For comparison, another possibility is presented too: the LOWESS-Filter (Cleveland, 1981) that is very popular in modern data analysis.

```
library(Kendall)
data(PrecipGL)
tsp(PrecipGL)
```

```
[1] 1900 1986    1
```

```
plot(PrecipGL)
kernel <- rep(1, 10)  # a rectangular kernel, please vary bandwith
lines(filter(PrecipGL, kernel/sum(kernel)), lwd=2, col="blue")
lines(lowess(time(PrecipGL),PrecipGL),lwd=3, col=2)
```



Now, the trend corrected series can be obtained by subtracting the trend, e.g.:

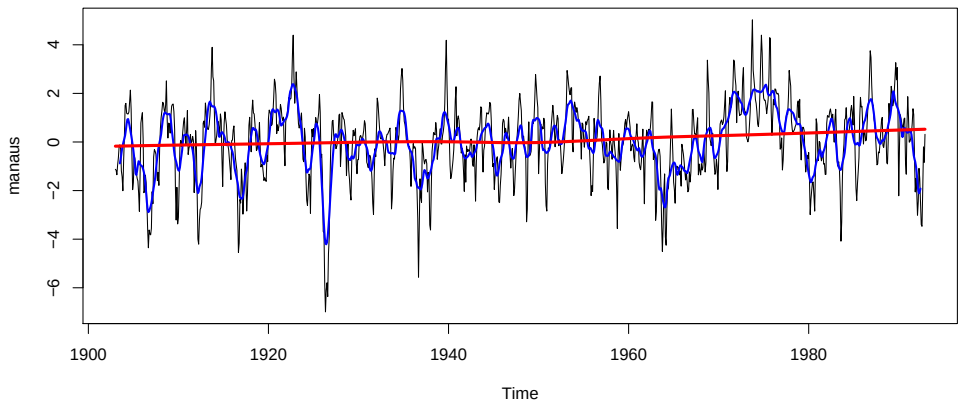
```
smooth <- filter(PrecipGL, kernel/sum(kernel))
## OR:
# smooth <- lowess(time(PrecipGL), PrecipGL)$y
res <- PrecipGL - smooth
```

For a seasonal time series with monthly values KLEIBER and ZEILEIS (2008) recommend a filter with 13 coefficients. The data set used in the following example describes the water level of Rio Negro 18 km upstream from its confluence with the Amazon River. The data set is contained in the package **boot** (CANTY and RIPLEY, 2009):

```
library(boot)
data(manaus)
tsp(manaus)
```

```
[1] 1903.000 1992.917   12.000
```

```
plot(manaus)
lines(filter(manaus, c(0.5, rep(1, 11), 0.5)/12), lwd=2, col="blue")
lines(lowess(time(manaus),manaus),lwd=3, col=2)
```



It is possible to vary the amount of smoothing for the linear filter and the LOWESS-smoother. In addition to LOWESS there is an improved algorithm implemented in R, LOESS (without "W"). Beyond that there are methods that try to achieve the ideal smoothing automatically (e.g. via GCV (*generalized cross validation*)). The *generalized additive models* (GAM) popular in many disciplines belong also to this class of smoothing models. An excellent overview of this subject is given in WOOD (2006), whose homepage [2] features additional tutorials too.

8.5.2 Automatic time series decomposition

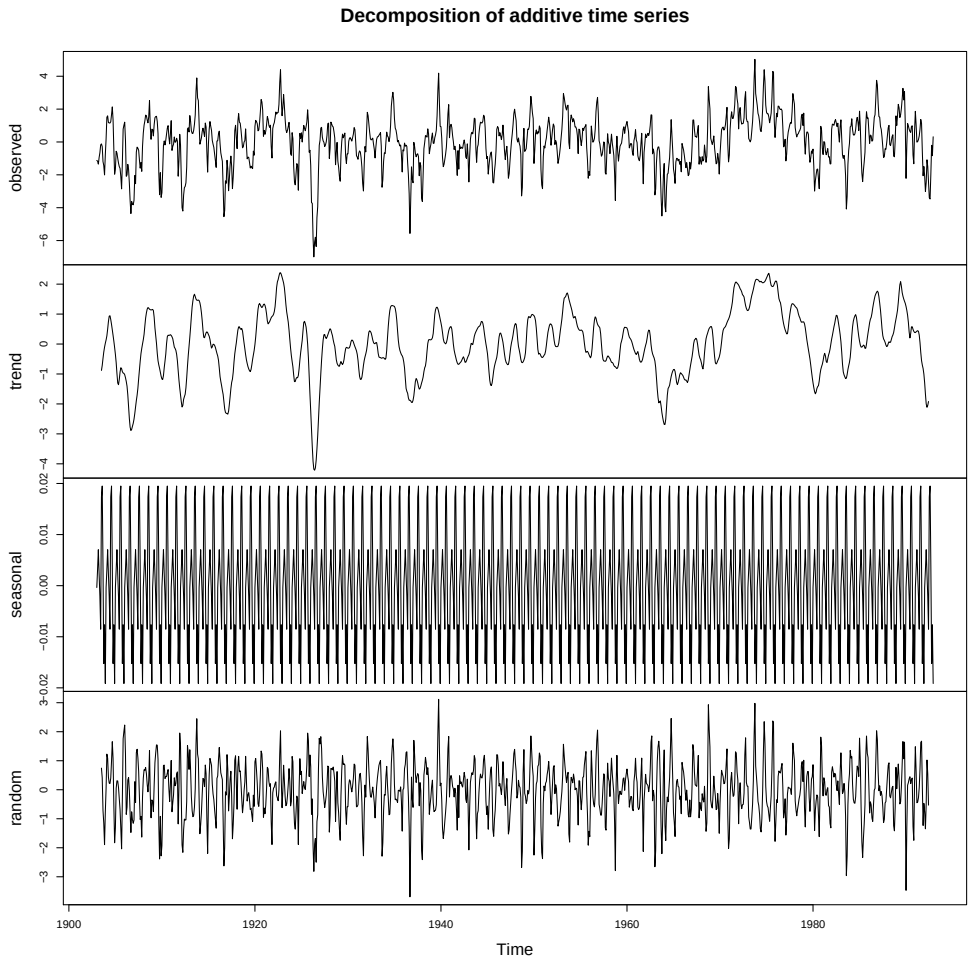
The function `decompose` implements the classical approach towards time series decomposition with the help of simple symmetric moving average filters.

[2] `http://www.maths.bath.ac.uk/~sw283/`

```
manaus_dec <- decompose(manaus)
str(manaus_dec)
```

```
List of 6
 $ x       : Time-Series [1:1080] from 1903 to 1993: -1.124 -1.164 -1.349 -0.945 
 $ seasonal: Time-Series [1:1080] from 1903 to 1993: -0.00036 0.00312 0.00704 0.0
 $ trend   : Time-Series [1:1080] from 1903 to 1993: NA NA NA NA NA ...
 $ random  : Time-Series [1:1080] from 1903 to 1993: NA NA NA NA NA ...
 $ figure  : num [1:12] -0.00036 0.00312 0.00704 0.00228 -0.00213 ...
 $ type    : chr "additive"
 - attr(*, "class")= chr "decomposed.ts"
```

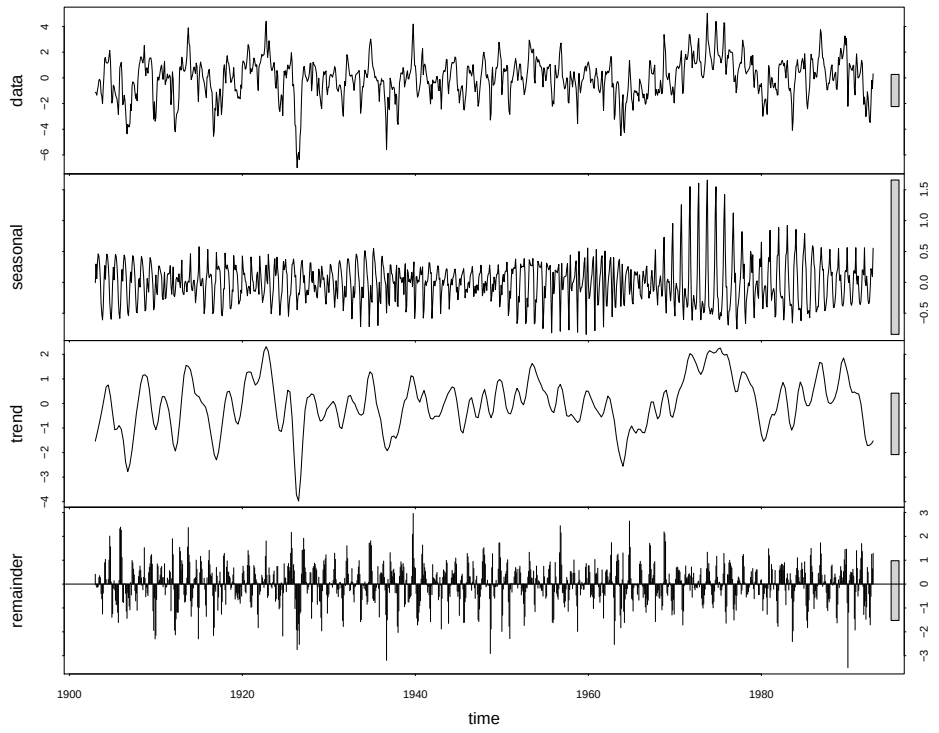
```
plot(manaus_dec)
```



In this data set the seasonal component possesses an additive character, but a multiplicative component would be possible too (`type="multiplicative"`).

The function `stl` (seasonal time series decomposition (CLEVELAND *et al.*, 1990)) uses a LOESS filter:

```
manaus_stl <- stl(manaus, s.window=13)
plot(manaus_stl)
```



8.5.3 Periodogram analysis

In nature periodic phenomena are found frequently, for instance the annual courses of global radiation and temperature, the development cycles of vegetation throughout the year or the beating of a water fleas' (*Daphnia*) legs. At this point only the application of the harmonic analysis (Fourier analysis) for the approximation of periodic data series is to be presented.

In principle every time series can be described as a sum of sine and a cosine functions with different periods (Fourier series).

$$x_t = a_0 + \sum_{p=1}^{N/2-1} \left(a_p \cos(2\pi pt/N) + b_p \sin(2\pi pt/N)\right) + a_{N/2}\cos(\pi t), \qquad t = 1 \ldots N \tag{8.3}$$

Here, a_0 is the mean value of the time series, a_p, b_p are the coefficients of the Fourier series and N is the length of the time series or the period. Similar to linear regression models the coefficients can be determined with a system of linear equations:

$$a_0 = \bar{x} \tag{8.4}$$

$$a_{N/2} = \sum_{t=1}^{N} (-1)^t x_t / N \tag{8.5}$$

$$a_p = 2\frac{\sum_{t=1}^{N} x_t \cos(2\pi pt/N)}{N} \tag{8.6}$$

$$b_p = 2\frac{\sum_{t=1}^{N} x_t \sin(2\pi pt/N)}{N} \tag{8.7}$$

Beyond that there is an especially powerful method called "fast Fourier transform" (FFT), which can calculate the coefficients a_0, a_p, b_p very efficiently with the help of complex numbers.

Equation 8.3 can also be transformed into a form with only one cosine term:

$$x_t = a_0 + \sum(R_p \cdot \cos(2\pi pt/N + \Phi_p) \tag{8.8}$$

with:

$$R_p = \sqrt{a_p^2 + b_p^2} \tag{8.9}$$

$$\Phi_p = \arctan(-b_p/a_p) \tag{8.10}$$

This offers the advantage that the amplitudes R_i and the phase shifts Φ_i of the particular frequencies $2\pi/N$, $4\pi/N, \ldots, \pi$) can be read off directly. The periodogram can be derived by plotting out $R_p^2/2$a, the proportion of variance of the pth harmonic term, against the frequency $\omega_p = 2\pi p/N$. Smoothing might be necessary in certain circumstances.

Because harmonic analysis breaks down the process into the proportions of variance, a time series can be synthesized based upon selected proportions of frequency.

8.5.4 Implementation in R

In order to simplify the analysis it is a good idea here to create two auxiliary functions. One functions is used in determining the coefficients a_p and b_p (eq. 8.4 to 8.7), the second one synthesizes the harmonic function according to equation 8.3. Below several possibilities are presented, e.g. the classical way an a function using the FFT contained within R. The synthesis can also be accomplished in various ways, e.g. the classical way via equation 8.3 or 8.8 or using the inverse FFT (not shown here). In R it is convenient to use matrix multiplication instead of loops. In practice, however, one function for each step is sufficient, e.g. analysis via FFT (`harmonic.fft`) and synthesis via the classic way with matrix multiplication (`synth.harmonic`):

```
## classic method of harmonic analysis
harmonic.classic <- function(x, pmax=length(x)/2) {
 n  <- length(x)
 t  <- 0:(n-1)
 a0 <- mean(x)
 a  <- numeric(pmax)
 b  <- numeric(pmax)
 for (p in 1:pmax) {
   k    <- 2 * pi * p * t / n
   a[p] <- sum(x * cos(k))
   b[p] <- sum(x * sin(k))
 }
 list(a0=a0, a=2*a/n, b=2*b/n)
}
```

```
## fast fourier version of harmonic analysis
harmonic.fft <- function(x) {
  n  <- length(x)
  pf <- fft(x)      # Fast Fourier Transform
  a0 <- Re(pf[1])/n # first element = mean
  pf <- pf[-1]      # drop first element
  a  <-  2*Re(pf)/n # Real part of complex
  b  <- -2*Im(pf)/n # Imaginary part
  list(a0=a0, a=a, b=b)
}
### ================================================

## synthesis of a harmonic function
## (classic method)
synth.harmonic.classic <- function(t, fpar, n, ord) {
  a <- fpar$a; b <- fpar$b; a0 <- fpar$a0
  x <- a0
  for (p in ord) {
    k <- 2 * pi * p * t/n
    x <- x + a[p] * cos(k) + b[p] * sin(k)
  }
  x
}
## synthesis of a harmonic function
## version with amplitude (R) and phase (Phi)
synth.harmonic.amplitude <- function(t, fpar, n, ord) {
  a   <- fpar$a; b <- fpar$b; a0 <- fpar$a0
  R   <- sqrt(a * a + b * b)
  Phi <- atan2(-b, a)
  x   <- a0
  for (p in ord) {
    x <- x + R[p] * cos(2 * pi * p * t/n + Phi[p])
  }
  x
}
## synthesis of a harmonic function
## classic method with matrices
synth.harmonic <- function(x, fpar, n, ord) {
  a <-  fpar$a; b <- fpar$b; a0 <- fpar$a0
  k <- (2 * pi * x/n) %*% t(ord)
  y <-    a0 + cos(k) %*% a[ord] +
               sin(k) %*% b[ord]
  y
}
```

A data set appropriate for testing can be created with sine and cosine functions, with a normally distributed error term for adding a little “noise”:

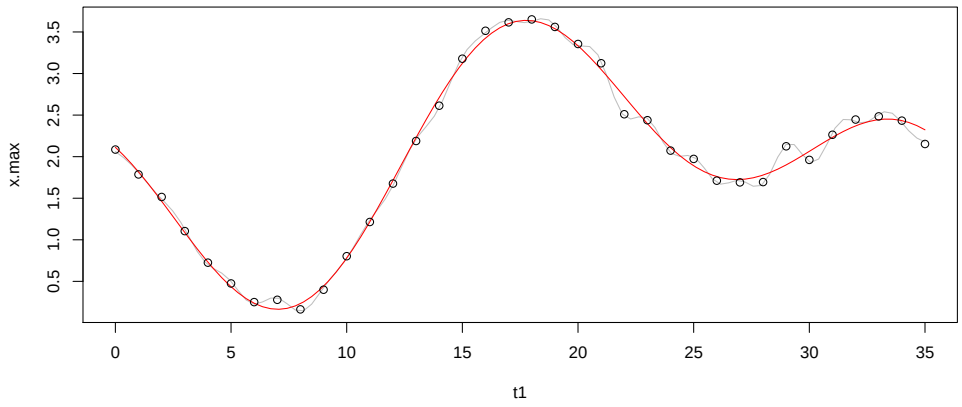
```
n <- 36
t <- 0:(n-1)
x <- 2 + sin(t*4*pi/n+2) + sin(t*2*pi/n + 4) + rnorm(n, sd=0.1)
```

Now, the estimation of the Fourier parameters is carried out with:

```
fpar <- harmonic.fft(x)
```

Afterwards, calling up `fpar` displays the calculated coefficients. Synthetic time series with the maximum order (or order 2 which is optimal in this case) can be received with:

```
t1<-seq(min(t), max(t), length=100)
x.max <-  synth.harmonic(t1, fpar, n, ord=1:(n/2))
x.1 <-  synth.harmonic(t1, fpar, n, ord=1:2)
plot(t1, x.max, col="gray", type="l")
lines(t1, x.1, col="red")
points(t,x)
```



`t1` is a user-defined domain of definition for which the function is to be plotted, `fpar` contains the Fourier coefficients and `n` is the period (number of values of the original data set). The vector `ord` indicates the harmonic order which is to be used in the synthesizing the function. Not all the orders from 1 to $n/2$ need to be used. Specific orders can be selected separately, too.

The calculation could also be performed outside R in a way similar to the one using `synth.harmonic` with the help of the coefficients a_0, a_p, b_p and equation 8.3 in a spreadsheet program (e.g. Excel).

8.5.5 Exercise

Annual courses of temperature or global radiation serve as driving forces of ecosystems and of course also of ecological models. A simple possibility is to represent them with harmonic functions of low order.

Exercise: Find a harmonic function that describes the mean annual course of global radiation in $\mathrm{J\,cm^{-2}d^{-1}}$. Use the 1981 to 1990 data set of Wahnsdorf weather station near Dresden (source: World Radiation Data Center[3]).

[3] `http://wrdc-mgo.nrel.gov/`

Solution

First, the auxiliary functions for harmonic analysis need to be typed in or read in. Afterwards, the data are imported from a text file, which needs to contain the column names in the first line (`header=TRUE`). We display the variable names, copy column `igl` to variable `x` and create a new variable representing time `t`, whereby we can bypass date calculation which can at times prove to be somewhat tricky.

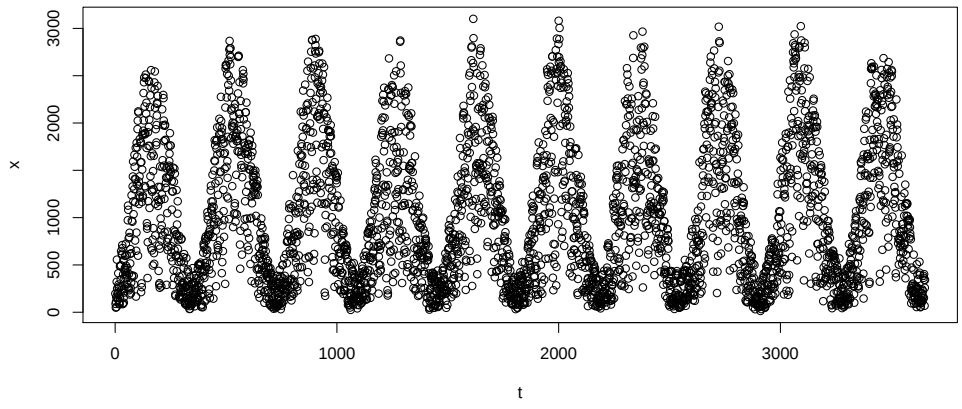
```
dat <- read.table("http://www.simecol.de/data/igl8190_dd.txt", header=TRUE)
names(dat)
```

```
[1] "date"         "igl"          "interpoliert"
```

```
x   <- dat$igl
t   <- 1:length(x)
```

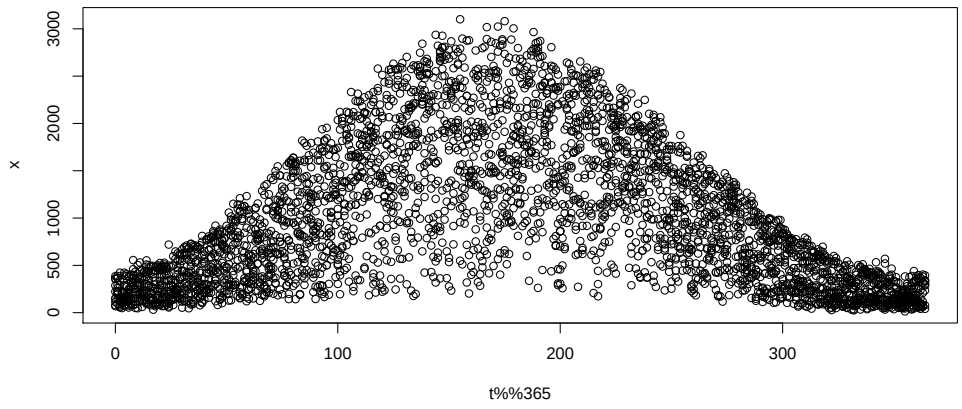
Subsequently we plot the data as a time series spanning several years:

```
plot(t, x)
```



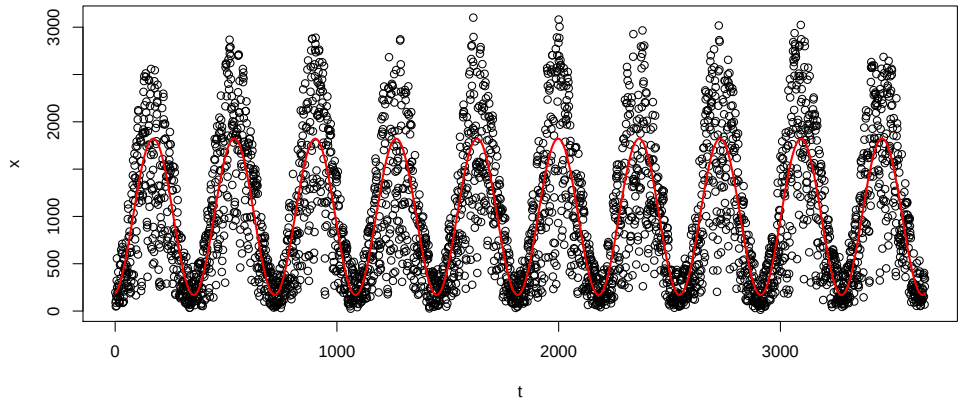
Alternatively, all the years can be plotted on top of each other, with the operator `%%` being the modulo operator (remainder of an integer-type number division. In this case day 366 would be plotted at $t = 1$.

```
plot(t %% 365, x)
```



After providing us with some orientation, we will now proceed to the actual analyis (using FFT this time) and plotting the results.

```
fpar <- harmonic.fft(x)
plot(t, x)
lines(synth.harmonic.classic(t,fpar,length(x),ord=10), col="red", lwd=2)
```

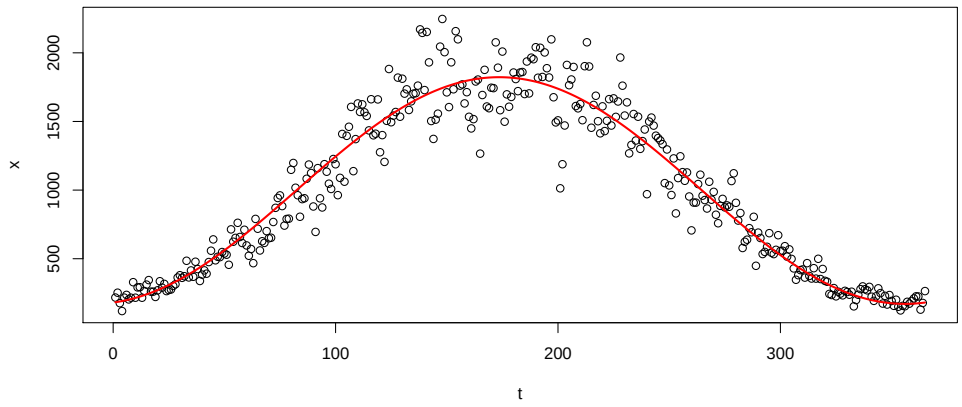


As the data set we use contains 10 years, `ord=10` represents an annual cycle. We start by playing around a little with the Fourier order and displaying the results, e.g. for the orders `ord=1`, `ord=1:10` and `ord=1:100`.

Solution 2

An alternative possibility is to calculate the 10 years mean for each of the 365 days and to calculate a 1st order harmonic function afterwards. The mean value can be calculated using an external spreadsheet programm or the very powerful R-Funktion `aggregate`, which expects its arguments to be either lists or data frames. The first argument characterizes the data to be analyzed, the second one the grouping and the third one the function to be used.

```
meanyear <- aggregate(list(x=x), list(yr=t%%365), mean)
x <- meanyear$x
t <- 1:length(x)
plot(t, x)
fpar <- harmonic.fft(x)
lines(synth.harmonic.classic(t,fpar,length(x),ord=1), col="red", lwd=2)
```



For further use the calculated function can now be written in a closed form. As we know the coefficients are located in the list `fpar`. If only the seasonal cycle ($p = 1$) is to be shown, what follows from equation 8.3 is:

```
cat(fpar$a0, fpar$a[1], fpar$b[1], "\n")
```

```
996.746 -815.988 126.741
```

```
plot(t,x)
x1 <- 997 - 816 * cos(2*pi*1*t/365) + 127 * sin(2*pi*1*t/365)
lines(t,x1)
```

Written out as a mathematical function that is:

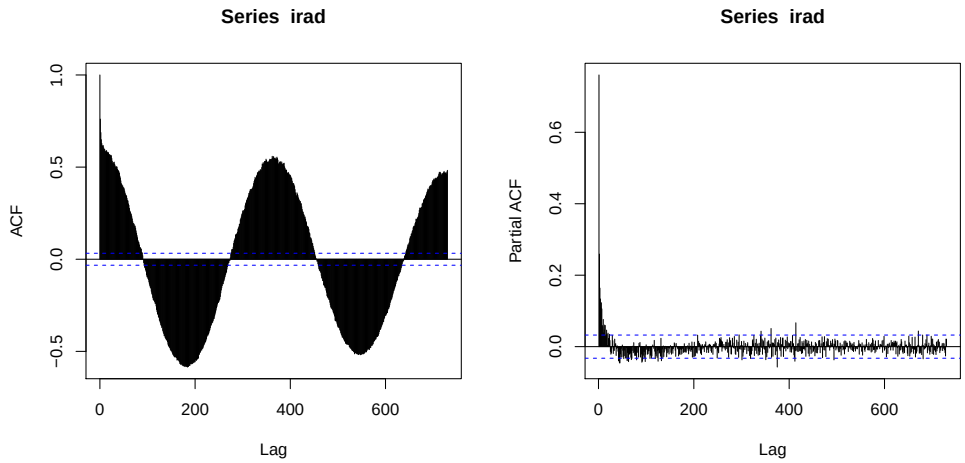
$$x = 997 - 816 \cdot \cos(2\pi \cdot t/365) + 126 \cdot \sin(2\pi \cdot t/365) \tag{8.11}$$

Because of the annual cycle it is more or less obvious which Fourier orders are needed in our example. Generally, this is unknown beforehand and has to be derived from periodogram or frequency analysis (see SCHLITTGEN and STREITBERG, 1989; BOX *et al.*, 1994).

8.5.6 Frequency spectra

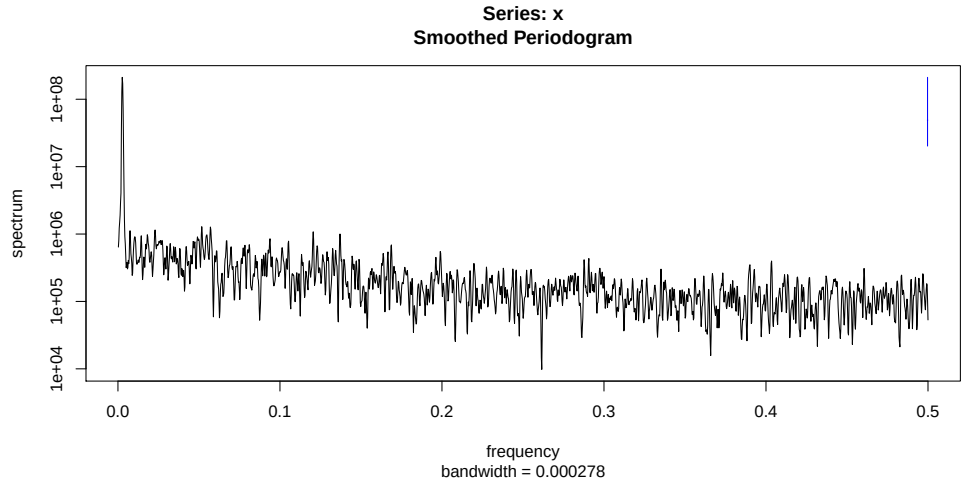
It goes without saying that R features a predefined function to this end. First, we convert the series into a time series and have a look at the autocorrelograms:

```
dat <- read.table("http://www.simecol.de/data/igl8190_dd.txt", header=TRUE)
x   <- dat$igl
irad <- ts(dat$igl)
par(mfrow=c(1,2))
acf(irad, lag.max=2*365)
pacf(irad, lag.max=2*365)
```



What follows is a spectral analysis with the parameter `spans` defining the window of the smoothing function (in this case a modified Daniell smoother):

```
sp <- spectrum(irad, spans=c(2, 2))
```

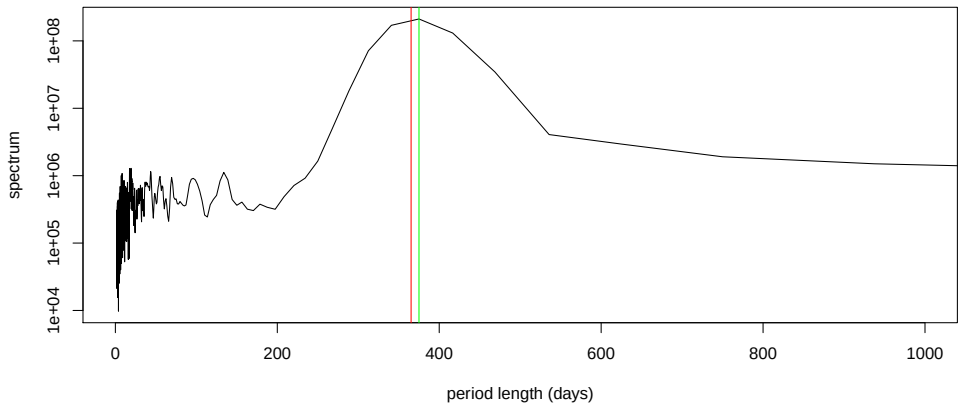


Instead of plotting it against the frequency domain the spectrum can be also plotted against the period (1/freq), with the y-axis displayed logarithmically here.

```
with(sp, plot(1/freq, spec, type="l", xlim=c(0,1000),
  ylab="spectrum", log="y", xlab="period length (days)"))
abline(v=365.25, col="red")
(smax <- 1/sp$freq[sp$spec == max(sp$spec)])
```

```
[1] 375
```

```
abline(v = smax, col = "green")  # spectral maximum is 375 days
```



The lines mark the expected and the empirical maximum at 365.25 and 375 days respectively.

8.5.7 More exercises

1. Caclulate the amplitude and phase shift of the global radiation function.
2. Does the result match your expectations (calendar)?
3. Present the function in the style of equation 8.8 and check the result graphically.
4. Compare the 10th order global radiation function from solution 1 with the 1st order function from solution 2.
5. Plot the function and the data with your preferred graphics or spreadsheet software.
6. Try to describe the epilimnion temperature of a lake with a harmonic function (data set `t_epi7.txt`). How many Fourier orders are required?

8.6 ARIMA Modelling

The main focus of ARIMA modelling[4] (BOX *et al.*, 1994) is prediction. Periodogram analysis and ARIMA models are both based upon the autocorrelation function. They are simply two different points of view. The idea is to convert time series with distinct trends or cycles to time series that are approximately stationary. Simple and seasonal difference filters are of particular importance here and may be applied multiple times successively. In order to reverse the differentiation used for model identification, integration is used for the prediction step.

A challenging problem in ARIMA modelling is the specification of the model order, that is the decision which AR and MA orders are to be used. In the BOX-JENKINS approach the autocorrelation function (`acf`) and the partial autocorrelation function (`pacf`) are used. Often, characteristic patterns can be observed, which may then serve as a first indicator as to which order the process in question has. Furthermore, it is recommended to fit several alternative models and to evaluate them using the mean squared error and the significance of the parameters ("Overfitting", BOX *et al.*, 1994).

[4] Autoregressive Integrated Moving Average

The autocorrelation function is suited especially for determining pure MA processes and the partial autocorrelation function for AR processes. Nevertheless, according to SCHLITTGEN and STREITBERG (1989) the use of ACF and PACF for specifying mixed ARMA processes is a "delicate problem".

8.6.1 Moving average processes

A stochastic process is termed moving average process of order q (in short: MA(q)-process), if it can be expressed by:

$$X_t = \varepsilon_t - \beta_1 \varepsilon_{t-1} - \cdots - \beta_q \varepsilon_{t-q}$$

In this context (ε_t) is a white noise process. This means that the actual value of X_t depends solely on "random fluctuations" (ε_t) in the past.

8.6.2 Autoregressive processes

A stochastic process (X_t) is termed autoregressive process of order p (in short: AR(p)-process), if it satisfies the relation:

$$X_t = \alpha_1 X_{t-1} + \cdots + \alpha_p X_{t-p} + \varepsilon_t$$

(ε_t) is a white noise process here. Thus, the AR-process formally corresponds to a multiple linear regression, in which the actual value can be understood as a function of the preceding values.

8.6.3 ARIMA processes

A process that is composed of an AR(p)- and a MA(q)-fraction is termed ARMA(p,q)-process. If it has been subject to one or more differencing steps it is called ARIMA(p,d,q)-process. If not only the immediate history is considered, but a seasonal shift as well, the resulting processes are ARIMA(p,d,q)(P,D,Q)-processes, which can be called SARIMA (seasonal ARIMA) too.

8.6.4 Fitting ARIMA models

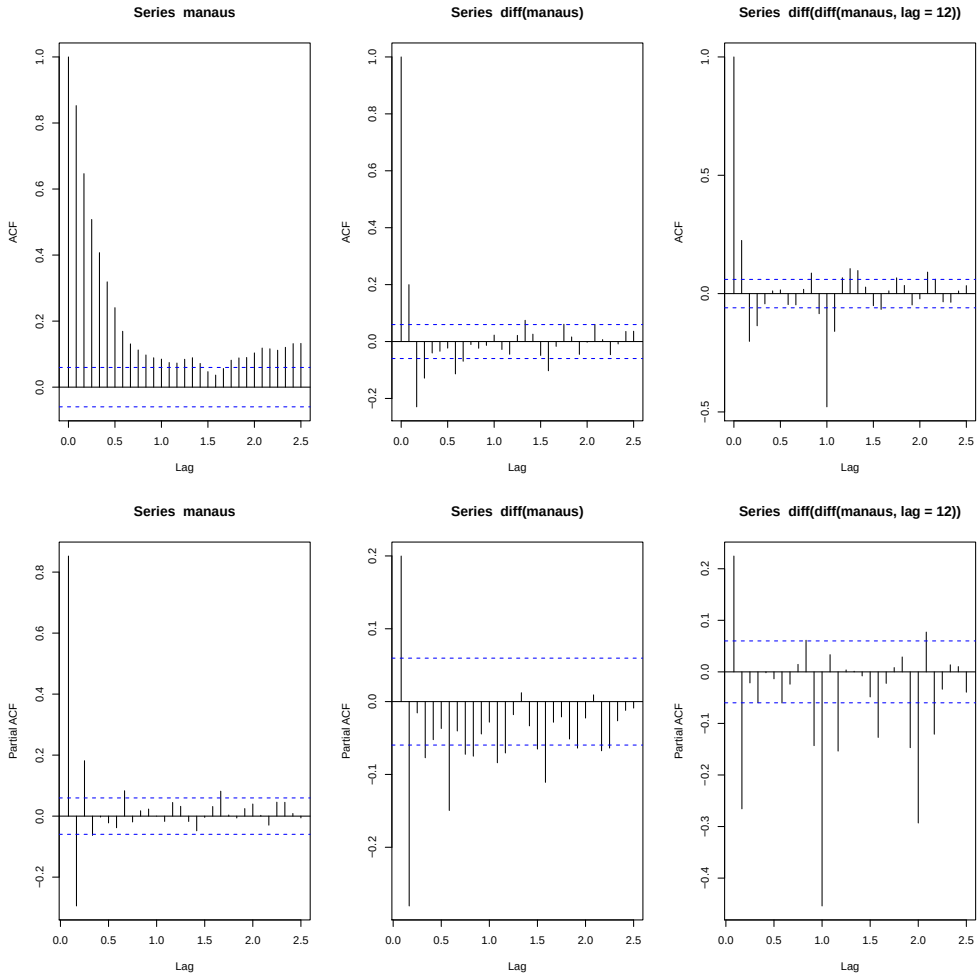
Identification is the process of determining the parameter values for a model specified before. To this end different methods are available, all of which are based on the maximum likelihood criterion. The main problem is the selection (specification) of the optimal model. Different criteria are suggested for decision making, e.g. "overfitting" and subsequent model simplification using visual criteria or a very detailed interpretation of autocorrelograms (e.g. by BOX *et al.*, 1994).

While these hints are still valid, today AIC-based model selection is preferred over a solely visual comparison of models. Given the availability of todays' computing power it is even possible to test the models in question (selected by acf, pacf) automatically. We will again use the `manaus` data set to do this.

```
library(boot)
data(manaus)
```

We begin by taking a look at the correlograms of the time series and its differences. In the case of the "double" differentiation we first differentiate seasonally (season-wise) and afterwards with `lag=1`:

```
par(mfrow=c(2,3))
acf(manaus)
acf(diff(manaus))
acf(diff(diff(manaus, lag=12)))
pacf(manaus)
pacf(diff(manaus))
pacf(diff(diff(manaus, lag=12)))
```



Without going into detail too deeply, we see that by all means differentiation is necessary and one or two AR-parameters and probably MA-parameters and seasonal parameters will be required.

The following short script (according to KLEIBER and ZEILEIS, 2008, modified) fits all models with 0 to 1 or 2 parameters respectively for all components. The number of differentiations is fixed to lags 1 for the immediate predecessor and 12 for the seasonal component. Note! Defining the models that are to be tested requires process knowledge and some good instinct.

```
dat <- manaus
pars <- expand.grid(ar=0:2, diff=1, ma=0:2, sar=0:1, sdiff=1, sma=0:1)
aic  <- numeric(nrow(pars))
```

```
 for (i in seq(along=aic))
   aic[i] <- AIC(arima(dat, unlist(pars[i, 1:3]), unlist(pars[i, 4:6])),
     k=log(length(dat)))
 ndx_best  <- which.min(aic)
 (pars_best <- pars[ndx_best,])
```

```
   ar diff ma sar sdiff sma
26  1    1  2   0     1   1
```

```
 ## and now we refit the 'best' (i.e. the most parsimonious) model again
 (m <- arima(dat, unlist(pars_best[1:3]), unlist(pars_best[4:6])))
```

```
Call:
arima(x = dat, order = unlist(pars_best[1:3]), seasonal = unlist(pars_best[4:6]))

Coefficients:
         ar1      ma1      ma2     sma1
      0.7424  -0.5647  -0.4353  -0.9906
s.e.  0.0231   0.0312   0.0308   0.0338

sigma^2 estimated as 0.5766:  log likelihood = -1248.63,  aic = 2507.27
```

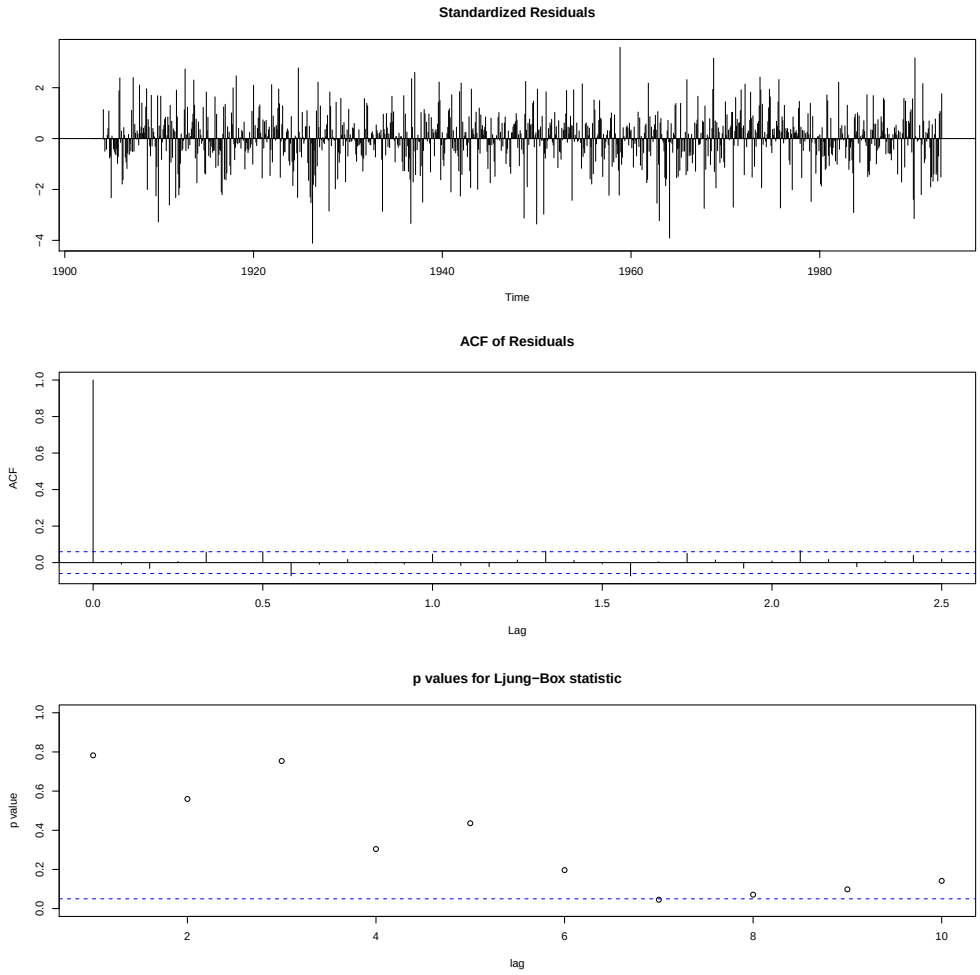
We see that the best model is one of S*ARIMA*$(1,1,2)(0,1,1)_{12}$ type. The level of significance can be obtained with the help of the standard errors.

The package **forecast** (HYNDMAN and KHANDAKAR, 2008) contains a fully automatic function that delivers a similar, though not perfectly identical result in our example:

```
 library(forecast)
 auto.arima(dat)
```

The identified model can be verified with `tsdiag`:

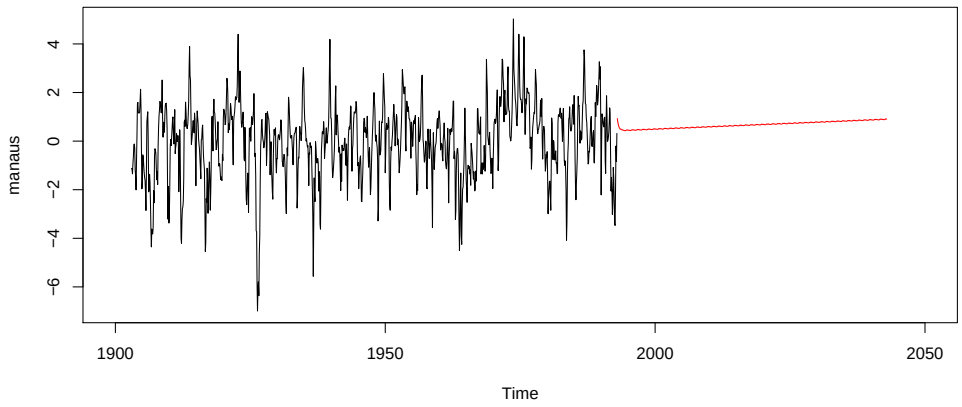
```
 tsdiag(m)
```



One can see that the residuals show no obvious pattern (and thus appear stationary) and that the ACF does not considerably exceed the significance thresholds (except for $lag = 0$ of course). The p-values of the Ljung-Box statistics are greater than 0.05, i.e. the residuals do not differ significantly from white noise.

The last step is the application of the identified model for a prediction. The function `predict` can be used for this purpose, for example for a period of 50 years:

```
pr <- predict(m, n.ahead=50*12)
plot(manaus, xlim=c(1900, 2050))
lines(pr$pred, col="red")
```



The package **forecast** features even more powerful functions and more "beautiful" graphics.

8.6.5 Exercises

1. Change the number of ARIMA-parameters to be fitted in the above example and evaluate the results.
2. Try to fit ARIMA models for the TSP and TSD time series.

8.7 Identification of structural breaks

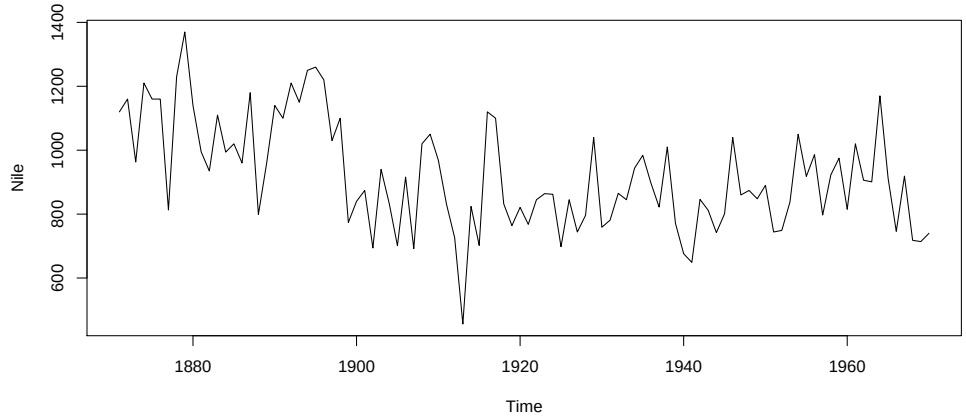
If one or more statistical parameters are not constant over the whole length of a time series it is called a structural break. For instance a location parameter (e.g. the mean), a trend or another distribution parameter (such as variance or covariance) may change.

8.7.1 Testing for strctural breaks

The package **strucchange** (ZEILEIS *et al.*, 2002) implements a number of tests for identification of structural changes or parameter instability of time series. Generally, two approaches are available: fluctuation tests and F-based tests. Fluctuation tests try to detect the structural instability with cumulative or moving sums (CUSUMs and MOSUMs).

The Nile data set (DURBIN and KOOPMAN, 2001, and literature cited there) contains measurements of annual discharge of the Nile at Aswan from 1871 to 1970:

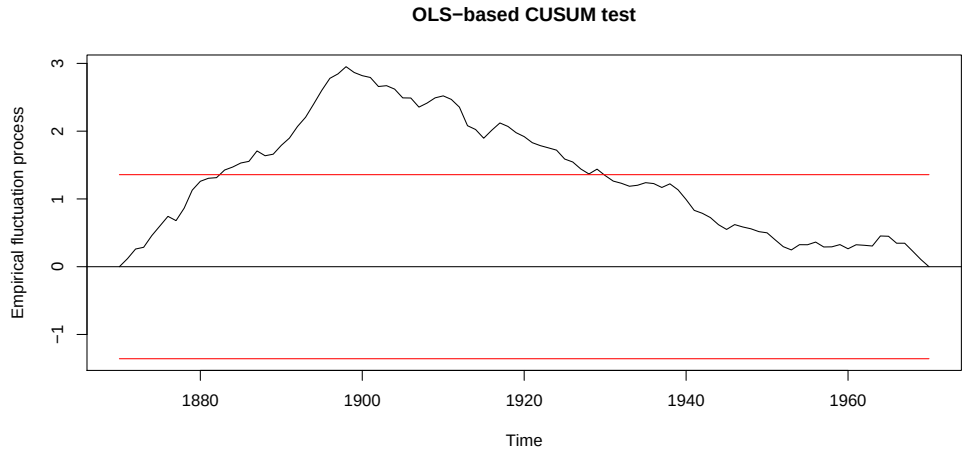
```
library(strucchange)
data("Nile")
plot(Nile)
```



Now we want to test if there is a structural break with respect to the location parameter, i.e. if there are periods with different discharge. We will use an *ordinary least squares* OLS-CUSUM test. The family of OLS-CUSUM and MOSUM tests is flexible and can be applied to various types of problems. The following example tests the simplest case by comparison with a null model. The functions needed are `efp` (*empirical fluctuation model*) and `sctest` (*structural change test*):

```
ocus <- efp(Nile ~ 1, type = "OLS-CUSUM")
plot(ocus)
sctest(ocus)
```

```
        OLS-based CUSUM test

data:  ocus
S0 = 2.9518, p-value = 5.409e-08
```



Instead of simple testing for stability of the mean with a null model (`~1`) time-shifted signals, among others, are possible on the right side too (see the example in KLEIBER and ZEILEIS, 2008, p. 171). Covariates can be specified as well.

8.7.2 Breakpoint analysis

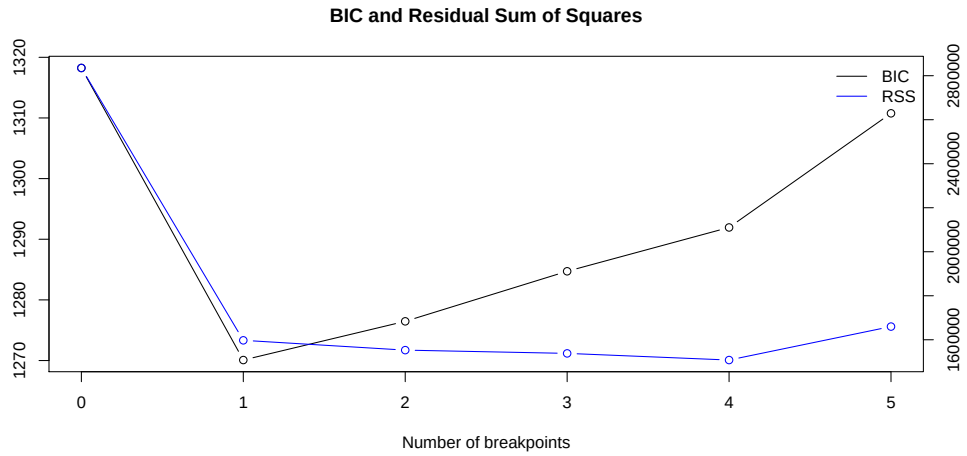
The purpose of breakpoint analysis (BAI and PERRON, 2003; ZEILEIS *et al.*, 2002, 2003) is to identify if there are structural breaks in a time series with respect to a specified linear model, and if yes, their number and location. That way it can be found out whether and where mean value or a trend change. What is special about the method presented here is that a model with an optimal number and location of breaks can be found with a BIC-based model selection.

Here, again, we can only demonstrate one of the simplest cases, the constancy of the location parameter (i.e. the mean). The function `breakpoints` serves for systematical testing and evaluation of a number of structural break models. As the result you receive candidates for the structural breaks and their RSS (*residual sums of squares*) plus BIC (*Bayes Information Criterion*). The full result can be obtained with `summary` and the behavior of `RSS` and `BIC` can be visualized with the plotting function:

```
bp.nile <- breakpoints(Nile ~ 1)
summary(bp.nile)
```

```
         Optimal (m+1)-segment partition:

Call:
breakpoints.formula(formula = Nile ~ 1)

Breakpoints at observation number:

m = 1      28
m = 2      28       83
m = 3      28    68 83
m = 4      28 45 68 83
m = 5   15 30 45 68 83

Corresponding to breakdates:

m = 1        1898
m = 2        1898           1953
m = 3        1898      1938 1953
m = 4        1898 1915 1938 1953
m = 5   1885 1900 1915 1938 1953

Fit:

m   0           1           2           3           4           5
RSS 2835156.750 1597457.194 1552923.616 1538096.513 1507888.476 1659993.500
BIC    1318.242    1270.084    1276.467    1284.718    1291.944    1310.765
```

```
## the BIC also chooses one breakpoint
plot(bp.nile)
```

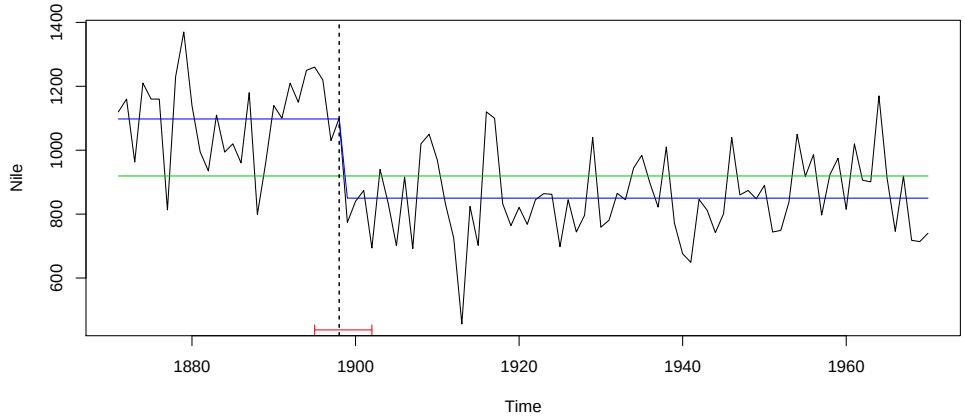


We can discover that the most suitable model here is indeed the model with exactly one structural break. For visualizing the breakpoints a linear model can be fitted piecewise. For this purpose, we use the function `breakfactor` that creates a so-called dummy variable with codes for each segment. Other generic functions serve for illustrating the location `lines(bp.nile)` and confidence interval (`confint(bp.nile)`, `lines(ci.nile)`) of the structural break:

```
## fit null hypothesis model and model with 1 breakpoint
fm0 <- lm(Nile ~ 1)
fm1 <- lm(Nile ~ breakfactor(bp.nile,  breaks = 1))
plot(Nile)
lines(ts(fitted(fm0),  start = 1871),  col = 3)
lines(ts(fitted(fm1),  start = 1871),  col = 4)
lines(bp.nile)
## confidence interval
ci.nile <- confint(bp.nile)
ci.nile
```

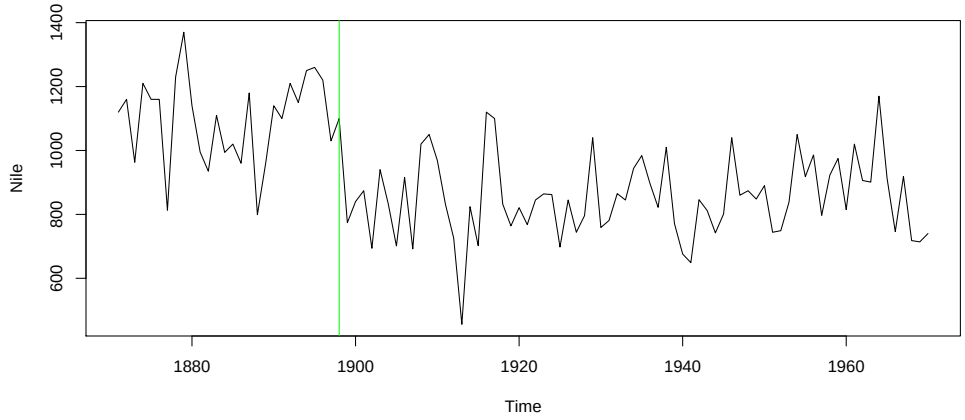
```
        Confidence intervals for breakpoints
        of optimal 2-segment partition:

Call:
confint.breakpointsfull(object = bp.nile)

Breakpoints at observation number:
  2.5 % breakpoints 97.5 %
1    25          28     32

Corresponding to breakdates:
  2.5 % breakpoints 97.5 %
1  1895        1898   1902
```

```
lines(ci.nile)
```



Of course the results can be analyzed with R's standard functions instead of the "generic" functions too. Here, as an example, the location of the structural break:

```
plot(Nile)
dat <- data.frame(time = time(Nile), Q = as.vector(Nile))
abline(v=dat$time[bp.nile$breakpoints],  col="green")
```



In order to check whether the model with the structural break is better than the null model we can use either a likelihood ratio test (i.e. a pairwise ANOVA in R) or a comparison by means of the AIC[5]:

```
anova(fm0, fm1)
```

```
Analysis of Variance Table

Model 1: Nile ~ 1
Model 2: Nile ~ breakfactor(bp.nile, breaks = 1)
  Res.Df     RSS Df Sum of Sq     F    Pr(>F)
1     99 2835157
2     98 1597457  1   1237700 75.93 7.439e-14 ***
```

[5] Akaike Information Criterion

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
 # Vergleich per AIC
 AIC(fm0,fm1)
```

```
    df      AIC
fm0  2 1313.031
fm1  3 1257.663
```

We can see that the model with the structural break in the year 1898 is significantly better than the null model.

Of course, further diagnostics and analyses can follow for the fitted models, e.g. a diagnostic comparison of autocorrelation and spectral density or a Q-Q-plot to see whether the residuals are normally distributed:

```
 acf(residuals(fm0))
 acf(residuals(fm1))
 spectrum(residuals(fm0), log = "no",  spans = c(5, 5))
 spectrum(residuals(fm1), log = "no",  spans = c(5, 5))
 qqnorm(residuals(fm0))
 qqnorm(residuals(fm1))
```

8.7.3 Exercise

The following example, which features an original data set from IHLE *et al.* (2005), is about how the abundance of the cyanobacteria genus *Microcystis* changes in the sediment of a highly eutrophic reservoir. The hypothesis was, broadly speaking, that there is a habitat change between a benthic and a pelagic mode of life. Structural break methods were used to determine the dates of such events which were in turn examined for coincidence with other variables.

The analysis is in line with the previous example. A small difference is that the time series is not equidistant, but in the here this does not matter much as the analysis is restricted to the location parameters. But instead of the class `ts` we have to use use class `zoo` for non-equidistant time series.

Test this example and discuss the results with the help of the publication of IHLE *et al.* (2005):

```
 dat    <- read.table("http://www.simecol.de/data/cells_sediment.txt", header = T
 time   <- as.Date(as.character(dat$date), format = "%d.%m.%Y")
 ## zoo: Z's ordered observations, loaded by strucchange
 Cells <- zoo(dat$cells, time)
 par(mfrow=c(2, 1))
 bp <- breakpoints(Cells ~ 1, h = 5)
 plot(bp)
 fm1 <- lm(Cells ~ breakfactor(bp))
 plot(Cells, type = "p")
 lines(fitted(fm1), col = "red")
 abline(v = time[bp$breakpoints], col = "blue")
 (ci <- confint(bp))
 sctest(Cells ~ 1, type = "OLS-CUSUM")
```

9 Multivariate statistics

The purpose of multivariate statistics is to analyze multiple variables simultaneously. This can be useful to get an overview of a large data set or to formulate preliminary hypotheses, which are to be examined in more detail later on (exploratory data analysis). On the other hand, multivariate statistics is applied if the effect in question is distributed over several variables, e.g. over the presence of multiple species, several peaks in a chemical analysis or a to compare differences at several places of a DNA molecule.

In general, one starts with a rectangular data structure, e.g. a matrix or a data frame, and refers to the rows as observations, sites, cases or objects, and to the columns as variables, properties or species. Now, the purpose of multivariate methods is to uncover structures in this matrix (similarities, differences and correlations, to be precise), to reduce the number of dimensions as far as possible, and to present the result numerically or graphically. The following chapter will give a short overview of important methods before demonstrating examples and their implementation in R. A detailed view at the background can be found in textbooks like LEGENDRE and LEGENDRE (1998) or LEYER and WESCHE (2007).

9.1 Basic concepts

The basic concepts that apply here are covariance and correlation, respectively distance and similarity.

A there variety of different measures of distance and similarity exist, so it is helpful to first specity their properties, by using a so-called axiomatic definition.

For a measure of distance d between the multidimensional points x_i and x_j the following conditions apply:

1. $d(x_i,x_j) \geq 0$ (distances are similar or equal to zero),
2. $d(x_i,x_j) = d(x_j,x_i)$ (the distance from A to B is the same as from B to A),
3. $d(x_i,x_i) = 0$ (the distance from a given point to itself is zero).

Beyond that, a distance measure is termed metric, if:

- $d = 0$ applies in the case of equality only, and
- the triangle inequality (the indirect route is longer than the direct route) applies too.

A measure of similarity s can be defined in a similar way:

1. $s(x_i,x_j) \leq s_{max}$
2. $s(x_i,x_j) = s(x_j,x_i)$
3. $s(x_i,x_i) = s_{max}$

and it is metric, if:

- s_{max} applies only in the case of equality and
- the triangle inequality applies too.

Similarity and distance can be transformed into each other. There are different possibilities to that end, e.g.:

$s = 1 - d/d_{max} \quad d = 1 - s/s_{max}$
$s = exp(-d) \quad d = -ln(s - s_{min})$

In many cases a simplification results from limiting the range of values of the similarity measures to the interval $(0,1)$, as done by some authors. Accordingly, the values $d = 1 - s$ are called "measure of dissimilarity", whereas to the more general "distance measures" are within the interval $(0,\infty)$.

From the variety of distance measures the following five may be most important for us:

- Euclidean distance (shortest connection between 2 points in space),
- Manhattan distance (around the corner, as in Manhattans grid-like streets),
- Chi-square distance for comparison of frequencies,
- Mahalanobis distance (takes covariance into account),
- Bray-Curtis Dissimilarity Index (created specifically for comparison of species lists in ecology).

Euclidean distance

At first, the Euclidean distance (shortest connection according to the Pythagorean theorem) appears to be the only natural criterion:

$$d = \sqrt{\sum (x_{ij} - x_{ik})^2}$$

In practice, however, this doesn't always hold true, as we often try to compare different units of measurement or different processes, with each other. If, for example, the nitrogen and phosphorous concentration of some water is given in mgl^{-1}, then phosphorous usually shows vanishingly small values, so the difference in nitrogen concentration comes to dominate the calculation of distance.

Thus, it is absolutely necessary to make the different measures and ranges of values comparable (scaling). Standardization is one possibility to do so:

$$x_i' = \frac{x_i - \bar{x}}{s}$$

i.e. the subtraction of the mean value and the division by the standard deviation. It should be noted that standardization is not always advisable. Let's assume we have an ecological survey with common and rare species. Then occurrence of one single individual of a species should be weighted less than the mass occurrence of a dominant species.

If a weighting is to be implies by different values (e.g. peak heights or abundances), centering is used:

$$x_i' = x_i - \bar{x}$$

Other possibilities of data preparation are:

- transformation of the data, e.g. by applying powers, roots or logarithms,
- ranking, i.e. assignment of 1, 2, 3, ... to the ordered values,
- special formulas for distances or similarities for binary (yes/no) or nominal data types.

Manhattan distance

In visual terms, the Manhattan distance (or city block distance) corresponds to a distance "around the corner", i.e. it is the sum of all distances over all dimensions:

$$d = \sum |x_{ij} - x_{ik}|$$

Mahalanobis distance

The Mahalanobis distance is a special measure, that takes into account the covariance (i.e. the mutual dependency) of the regarded dimensions. Let's for example assume we have four variables, e.g. nitrogen, phosphorous and phytoplankton concentration in a lake and the lake's depth. The first three variables are criteria of the trophic status and normally correlated among each other, because all of them are dependent on external loads. By applying a multivariate analysis to such a data set, the trophic status is given an almost threefold influence on the statistics, while the lake's depth occurs only once. Thus, small yet possibly existing differences between nitrogen, phosphorous and phytoplankton will get lost in the analysis.

At this point the Mahalanobis distance can help. It takes interdependency into account and eliminates correlation between the variables. However, minor effects and and random errors get also increased weight and may falsely influence the result.

On the other hand the Mahalanobis distance has the advantage that scaling (making different scales comparable) is made unnecessary by including the structure of covariance.

Chi-square distance

Chi-square distance (χ^2 distance) serves for comparison of frequencies, as they are used in correspondence analysis (see CA and CCA for instance). Here, the Chi-square distance (χ^2) measures to which extent an observed frequency deviates from an expected frequency. It is in general defined as:

$$\chi^2 = \sum_{i=1}^{n} \frac{(B_i - E_i)^2}{E_i}$$

where B_i is the observed and E_i is the expected frequency. By calculating the square root it becomes clear that χ is basically a weighted Euclidean distance, in which the weight is the reciprocal of the expected frequency. With regard to correspondence analysis (CA), this means that every frequency refers to the total sum of the values in the columns or rows of the table. Variables with large values (e.g. common species) will thus be weighted less, while variables with smaller frequencies will be emphasized.

Applying the genaral formula to a species list, we get (in the notation of LEYER and WESCHE (2007)):

$$D_{\chi^2_{1,2}} = \sqrt{x_{++}} \sqrt{\sum_{k=1}^{m} \frac{1}{x_{+k}} \left(\frac{x_{1k}}{x_{1+}} - \frac{x_{2k}}{x_{2+}} \right)^2} \tag{9.1}$$

Table 9.1: Example for the calculation of the Bray-Curtis coefficient from species abundances from (LEYER and WESCHE, 2007)

	1	2	3	4	5	6	B	C	w
Object 1	7	3	2	0	4	0	16		
Object 2	4	4	0	0	6	5		19	
Min	4	3	0	0	4	0			11

Here x_{+k} is the sum of all values for species k, x_{1+} is the sum of all values for observation 1 and x_{++} is the sum of all values contained in the table.

Bray-Curtis distance

The Bray-Curtis distance is a non-metrical (i.e. violating the triangle rule) distance measure used for the comparison of abundances. The numerator counts the sum of **differences** between abundances, the denominator the sum of **all sums** of abundances of the compared samples (e.g. sampling sites):

$$d = \frac{\sum |x_{ij} - x_{ik}|}{\sum (x_{ij} + x_{ik})} \tag{9.2}$$

Alternatively, the Bray-Curtis coefficient (S_{bc}) can be obtained as the double ratio of the species present in two observations (w) and the sum of species present only in one of both observations (B or C, resp.):

$$s_{bc} = \frac{2w}{B+C} \tag{9.3}$$

Instead of Bray Curtis similarity, it can also be expressed as Bray-Curtis dissimilarity:

$$d_{bc} = 1 - s_{bc} \tag{9.4}$$

The following example is to illustrate the application. We assume there are 2 objects (e.g. sampling sites), in which 6 species were found in total (see table 9.1).

First we calculate the Bray-Curtis coefficient (`sbc`) or Bray-Curtis dissimilarity (`1-sbc`) according to equation 9.3:

```
ob1 <- c(7, 3, 2, 0, 4, 0); ob2 <- c(4, 4, 0, 0, 6, 5)
B <- sum(ob1); C <- sum(ob2); w  <- sum(pmin(ob1, ob2))
sbc <- 2*w/(B+C)
sbc
```

```
[1] 0.6285714
```

```
1 - sbc
```

```
[1] 0.3714286
```

as well as according to equation 9.4 (for comparison):

```
sum(abs(ob1-ob2))/sum(ob1+ob2)
```

```
[1] 0.3714286
```

and finally with the ready-to-use function `vegdist` from the **vegan** package:

```
library(vegan)
vegdist(rbind(ob1, ob2))
```

```
          ob1
ob2 0.3714286
```

In all three cases we receive the same result. The `vegdist` function, however, is usually used to create a complex distance matrix for the pairwise comparison of a multitude of objects.

9.2 Ordination methods

The purpose of ordination methods is to bring measurements into an order. Such an "order" does not need to be limited to one dimension, but can be multidimensional. The number of dimensions is smaller or equal to the original number of variables. Usually it will be tried to reduce the information present in the data to as few dimensions as possible (dimensional reduction).

In the past, a large number of different methods was developed, which can roughly be divided into single-matrix methods (termed *unconstrained ordination*, too) and two-matrix methods (*constrained ordination*).

As the name suggests, single matrix methods work with a single matrix, which might for example be a species list or a matrix with physical and chemical data. In the two-matrix methods methods a distinction is made between a dependent matrix (e.g. a species list) and an independent or explanatory matrix (e.g. environmental factors). In the following we will therefore often just call them "species matrix" and "environmental matrix", but the same methods can also be applied to other data sets, e.g. chemical data, results of simulation experiments or marketing data.

9.2.1 PCA: Principal Components Analysis

The aim of principal components analysis is to place new coordinate axes into a multidimensional space (created by the variables) in such a way that the main part of the information is found in as few dimensions (coordinate axes) as possible. In these terms, "information" is measured as the total variance of the data from which as much as possible is to be assigned to a small number of new artificial coordinate axes (the principal components). The method is the most fundamental method of dimensionality reduction and is based on turning the original coordinate system using matrix operations (Eigenvalue decomposition).

The working basis of the method and a useful by-product of the analysis is the covariance or correlation matrix. Details can be found in the textbooks (see for example VENABLES and RIPLEY, 2002).

A particular advantage of principal components analysis is that the variables and objects can be displayed in a combined plot (a biplot). Another advantage is that the PCA is easily understandable and therefore interpretable. Beyond that, a PCA of a data set can serve as a starting point for further analysis, e.g. for

cluster analyses with a dimension-reduced data set, as a starting point for Nonmetric Multidimensional Scaling (NMDS) or for performing principal component regression.

The main disadvantages are that the PCA relies on Euclidean distance, and one often does not succeed in bringing a sufficient portion of the information into one plane.

9.2.2 CA: Correspondence Analysis

The correspondence analysis has been developed several times under different names and can be calculated in different ways. The classical method is based on the so-called gradient analysis (how can species be ordered along an environmental gradient) and was also called *weighted averaging* (see LEYER and WESCHE, 2007, for a description of the classical method).

In modern software the CA is being calculated similarly to the PCA with the help of eigenvalue decomposition. The only difference is the use of the Chi-square distance instead of the Euclidean distance.

In short, the PCA is most suited for the analysis of metric data where Euclidean distances makes sense, the CA for the analysis of frequency data.

9.2.3 PCO: Principal Coordinate Analysis

PCA as well as as CA are both limited to one specific measure of distance. The PCO methods (or PCoA) are an advancement insofar as they allow for the use of other distance measures (or dissimilarity measures) than the Euclidean distance. For that reason, they are more flexible, but the interpretation can be tricky because of the many different variants. PCO is the classical form of the metrical multidimensional scaling (MDS).

The R functions `dist` from the `stats` package or `vegdist` from the `vegan` package allow other distance measures besides Euclidean and Chi-square, e.g. Bray-Curtis or Canberra distance. Beyond that, individual distance matrices (based for example upon Renkonen's coefficient) can be handed over to the PCO.

9.2.4 Nonmetric Multidimensional Scaling (NMDS)

PCO is the method that creates the best **distortion-free** geometric projection of higher dimensional data to a lower dimension. Sometimes, however, an even better representation of the structure of similarity can be achieved when a certain distortion is accepted. In contrast to the search for coordinate axes with a maximum portion of variance, nonmetric methods strive for a representation (mapping) in which the correlation of the distances is as large as possible.

Different iterative methods have been developed to achieve this aim, e.g. Nonlinear Mapping (Sammon) or Kruskal's Nonmetric Multidimensional Scaling (NMDS). In both methods the quality of the mapping is expressed by a "stress" value, but they are calculated in different ways. In the **vegan** aund **MASS** packages the so-called "stress 1" is given (table 9.2):

$$S_1 = \sqrt{\frac{\sum_{i \neq j} (\theta(d_{ij}) - \tilde{d}_{ij})^2}{\sum_{i \neq j} \tilde{d}_{ij}^2}}$$

with $\tilde{d}_{ij}$ = ordination distance and $\theta(d_{ij})$ = observed dissimilarity. Another way of evaluating the results is the Shepard diagram. It compares the distances displayed in the NMDS with the original distances. The NMDS fit is plotted as stepcurve.

As the NMDS works iteratively, we receive different results depending on the initial situation. It is impossible to find *the* best fit, only local minima can be found. As a consequence, one either has to perform several runs with different starting configurations, e.g. with the help of random numbers, or starts with a configuration that is known to work well, e.g. because it has been calculated with a PCO. By default, the function `isoMDS` performs an NMDS with PCO as starting configuration. IN contrast (and even better) function `metaMDS` from the `vegan` package can be used for a multiple automatic MDS with random initialization.

Table 9.2: Guideline stress values for the function `isoMDS` and the reference values of the SPSS procedure ALSCAL for comparison

	Stress 1 (R)	Stress 2 (SPSS-ALSCAL)
low	0.2	0.4
sufficient	0.1	0.2
good	0.05	0.1
excellent	0.025	0.05
perfect	0	0

9.2.5 CCA und RDA: Canonical Correspondence Analysis and Redundancy Analysis

Redundancy analysis and canonical correspondence analysis are comparable to multiple linear regression, but, in contrast, not only multiple independent variables (x-values, explanatory variables), but also a complex matrix of dependent variables (y-values, e.g. a complete species matrix) can be related to each other.

Accordingly, RDA and CCA are two-matrix methods, i.e. a regression between multiple y as a function of multiple x. They are called *constrained ordination* methods, because a distinction is made between the dimensions that can explained by the environmental matrix (*constrained axes*) and the other dimensions (*unconstrained axes*).

Apart from their different origin, both methods are relatively similar to each other. One difference is, however, that RDA uses the Euclidean distance, whereas the CCA uses the Chi-square distance. Thus, it becomes clear that the PCA is a special case of RDA, and CA is a special case of CCA, if the explanatory "environmental" matrices are omitted.

CCA is very widespread in vegetation ecology and is recommended as a standard method in that field, besides a special variant of it called DCA (Detrended Correspondence Analysis). A great advantage of these methods is that they allow for the simultaneous analysis and plotting of environmental factors and species lists, another is that there is an extraordinary amount of experience in interpreting the results of these methods.

Initially, though, there may be problems understanding that only the explainable portion of the information (called "Inertia" or mass) is displayed. In addition, RDA and CCA sometimes produce unexpected artifacts (so-called arc effects or horse-shoe effects). Even in DCA, which was developed to that purpose, these artifacts are fixed only incompletely (and according to some authors in a way hardly comprehensible). The DCA can be found as `decorana` in R.

9.3 Vector Fitting

"Vector fitting" is a method of subsequently fitting environmental data to an ordination obtained from a one-matrix method (such as CA, PCA or NMDS).

As this method is suitable for expressing the influence of environmental factors to a species composition, an NMDS with subsequent vector fitting is a simpler and sometimes more comprehensible alternative to CCA.

9.4 Randomization Tests

Ordination methods at first only deliver a graphic representation. In many cases, however, it is desirable to extract information on the significance of the displayed relations. In the context of multivariate statistics most often randomization methods are used to that end. The principle they are based on is that a suitable test criterion is calculated and then the observed configuration is compared to a variety of random configurations. Depending on the problem there are different randomization tests, e.g.:

- The Mantel test, that compares two distance matrices (e.g. species list and environmental matrix) and tests the hypothesis that there is a correlation between these distance matrices (if there is a correlation between environmental factors and the colonization, for instance).
- Procrustes analyses, that allows to describe similarities and differences of two ordinations (congruence analysis). In particular, the Procrustes test tests if two ordinations differ significantly. It may be used as an alternative to Mantel tests.
- ANOSIM is a resampling analogue to ANOVA which tests for the hypothesis that there is a difference between two or more sampling series (e.g. after an anthropogenic interference). The ANOSIM test is rank-based. A problem is that it is not robust against different group sizes and group heterogeneity, which can under certain circumstances lead to falsely significant results (OKSANEN, 2010).
- For two-matrix methods (e.g. CCA or RDA), permutation tests for the significance of the constraints (the influencing environmental variables) are available. In R it can easily be called using the function `anova` with a resulting object of a CCA or RDA.
- The Multiple Response Permutation Procedure (MRPP) is similar to ANOSIM in terms of expressiveness, but works with the original distances instead. The method detects differences in the location (species abundance) and variation (diversity).
- The ADONIS test is a relatively new and now already popular method for multivariate analysis of variance of distance matrices (ANDERSON, 2001). The method tests, how complete species lists depend upon environmental factors specified in a model formula. In that process it often is important to limit the permutation to appropriate strata, for instance if samples were taken at different times of the season.

In R these methods can be found in the functions called `mantel`, `anosim`, `mrpp` and `adonis`, all of them being part of the **vegan** package. Beyond that, the BIOENV method known particularly from the statistics package PRIMER (in Ras function `bioenv`) offers another method for identifying the "best subset of explanatory variables" for a species list.

Multivariate resampling tests allow for direct comparison of species compositions (or other multivariate data sets) without the need for a calculation of indices of any kind (e.g. diversity indices). The usual statistical

prerequisites have to be met, though, especially representativeness and independence of samples. What can be seen is that multivariate resampling tests can often relatively easy detect significant differences. However, the question remains what, after all, the meaning of differences in species composition is. A subject-specific interpretation and evaluation of the findings is, in this case, even more important than normally.

9.5 Classification methods

The aim of cluster analysis is to identify groups of similar objects. In contrast to ordination methods (PCA, PCO, NMDS etc.) the groups are not to be visualized in their location relative to each over, but their distances are to be displayed in a tree diagrem. While ordination methods do not always succeed in reducing a given data structure to only a few dimensions (e.g. a two-dimensional plot), cluster methods can handle arbitrary high-dimensional and non-correlated data. Therefore, it is often advisable to combine ordination with cluster analysis.

Basically, there are hierarchical, non-hierarchical, agglomerating and disaggregating (divisive) methods of cluster analysis. In the following only two methods are presented, agglomerative hierarchical clustering and a non-hierarchical divisive method.

9.5.0.1 Hierarchical Cluster Analysis

Hierarchical cluster analysis starts with individual objects, and creates clusters by combining the objects closest to each other. The algorithm is:

1. Find the smallest distance,
2. merge objects with the smallest distance into a new object,
3. refresh the distance matrix (distances to the new objects),
4. go to step 1, if there are more than 2 objects left.

In step 3 the problem arises of how to determine the distance between an individual observation and a cluster created before. Accordingly, there is a large number of possibilities and making a decision which one to use can be hard at first view.

Single Linkage	nearest neighbour
Complete Linkage	most distant element
Average Linkage	average distance
Median (weighted/unweighted)	distance of medians
Centroid (weighted/unweighted)	centroid
WARD (weighted/unweighted)	minimum variance
flexible strategies	adjustable behaviour

While *single linkage* uses the shortest distance between the nearest elements of a cluster, in the *complete linkage* approach the elements farthest away from each other determine the distance. The WARD method is a special method. It calculates the distance in a way that keeps the variance of the created clusters as small as possible. All other methods employ different kinds of mean distances between the clusters.

Single linkage often produces very widely branched clusters and so-called chain-effects, whereas *complete linkage* usually aggregates clusters very late. As a general rule the best compromise can be achieved with the WARD strategy or a mean value method. Above that, it is no bad idea to run different methods successively and to observe the stability of the configuration (are individuals always assigned to the same cluster?).

9.5.0.2 Non-hierarchical k-Means Cluster analysis

In non-hierarchical cluster analysis the number of groups to be created is defined beforehand. The clusters are being aggregated either from a starting configuration or iteratively with the help of a random initialization. The clustering success is then mesured as sum of squares or by means of a "stress" value.

The number of cluster is either derived from the specific scientific question, or it is set based on a hierarchical cluster analysis. K-means clustering can be used if a very large number of individuals needs to be classified and a tree diagram would be too big and confusing. Another useful application is the combined application of k-means clustering and a plotting method, e.g. an NMDS.

9.6 Examples and Implementation in R

Despite the fact that some of the methods presented before work quite differently, it makes sense to present their use and implementation in context. Besides the methods presented here (which come from the packages **stats**, **MASS** and **vegan**), additional methods can be found in packages `cluster`, `knn`, `ade4`, `cclust`, `daisy` and others.

The **vegan** package has undergone a very exciting development, and now contains a comprehensive set of advanced functions tailored directly to the specific needs of ecologists.

Most notable among these enhancements are, for instance, the great extension of the function `metaMDS`, the direct linking of environmental data with NMDS graphics (vector fitting and surface fitting) and modern randomization tests. These allow to perform ANOVA-like analyses with entire species lists (similarity matrices).

It is particularly pleasing that all these function are now well documented. An easily understandable and highly recommended tutorial[1] and additional material on multivariate statistics can be found on the homepage[2] of the first author of **vegan**, Jari Oksanen.

Complementary to this also the approach pursued by the "French School" of multivariate statistics would be worth to be considered (see e.g. DRAY and DUFOUR, 2007), especially as there are also nice graphical user interfaces (THIOULOUSE and DRAY, 2007).

9.6.1 Example 1: Lake Data Set

The following example shows some criteria of several lakes in Brandenburg and Mecklenburg from KOSCHEL *et al.* (1987). The example is intended to be small, so basically there would be no need for multivariate statistics. It has, however, the advantage that the results can easily be understood and compared with our expectations.

[1] `http://cc.oulu.fi/~jarioksa/opetus/metodi/vegantutor.pdf`
[2] `http://cc.oulu.fi/~jarioksa/`

Lakes	z (m)	t (a)	P (μg/l)	N (mg/l)	Chl (μg/l)	PP ($gC\,m^{-2}a^{-1}$)	SD m
S	23.7	40	2.5	0.2	0.7	95	8.4
NN	5.9	10	2	0.2	1.1	140	7.4
NS	7.1	10	2.5	0.1	0.9	145	6.5
BL	25.2	17	50	0.1	6.1	210	3.8
SL	7.8	2	30	0.1	4.7	200	3.7
DA	5	4	100	0.5	14.9	250	1.9
HS	6.3	4	1150	0.75	17.5	420	1.6

Principal components

First, we will perform a principal component analysis and try to interpret the results, and afterwards run a PCO. Finally, we will apply NMDS, once with a PCO as a starting configuration (default in `isoMDS`) and once with a random starting configuration.

The data set's first column holds an identifier of the object (lake name). Via `row.names` it will be used as row name of the matrix and is removed from the actual data set afterwards. The function `scale` does a standardization and `prcomp` runs the analysis itself:

```
library(vegan)
library(MASS)
dat <- read.table("http://www.simecol.de/data/seen_bb.txt",header=TRUE)
# first column contains row names
row.names(dat) <- dat[,1]
# remove the first column
dat <- dat[,-1]
# principal components (with standardized data)
pca <- prcomp(scale(dat))
# proportion of variance (in percent)
summary(pca)
```

```
Importance of components:
                          PC1    PC2     PC3    PC4     PC5     PC6        PC7
Standard deviation     2.2007 1.1411 0.76855 0.5006 0.11502 0.01502 4.169e-17
Proportion of Variance 0.6919 0.1860 0.08438 0.0358 0.00189 0.00003 0.000e+00
Cumulative Proportion  0.6919 0.8779 0.96228 0.9981 0.99997 1.00000 1.000e+00
```

```
biplot(pca, choices=c(1,2))
```

Interpretation aid

The first look goes to proportions of variance stated by `summary(pca)`. They decide whether the reduction of dimensions was successful and how many principal components need to be considered (with the principal components being sorted according to their proportion of variance). In our example 88% of all variance fall upon the first two principal components. That means that a plot of PC1 and PC2 contains already 88% of the information.

The plot created with `biplot` shows objects (lakes) and variables simultaneously. Depending on the algorithm used, the plot may be mirrored along the x or y axis. So what is important is only the relation of the objects and variables to each other, not if they are at the right or left, top or bottom.

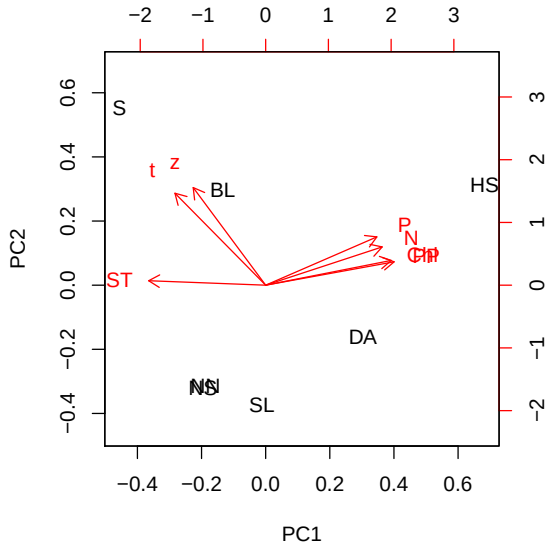


Figure 9.1: PCA of the lake data set.

First the plots are interpreted with regard to the objects, e.g.:

- Nehmitzsee Nord' (NN) is similar to Nehmitzsee Süd (NS), they form a cluster,
- Stechlinsee (S) and Dagowsee (DG) are entirely different.

When interpreting the variables it has to be noted that only long arrows lie in the represented plane. Thus, only long arrows can be interpreted, short arrows point to dimensions not contained in the plot:

- nitrogen (N), phosphorous (P), chlorophyll (Chl) and primary production (PP) are correlated,
- a high concentration of chlorophyll means low secchi depth (SD, depth of visibility),
- phosphorous and mean depth (z) are not correlated (at least not in our data set).

A combined interpretation is possible, too. The respective objects have to be projected onto the arrows rectangularly:

- Stechlinsee has the largest depth (z) and the longest residence time (t).
- Haussee (HS) has the highest nutrient and chlorophyll concentrations as well as the highest planktonic primary production.

Principal Coordinate Analysis

The PCO can be done in a similar way, but the possibility to use any measure of distance does not offer any advantage over PCA in the example at hand, as the Euclidean distance used here is suitable for physical and chemical data.

The difference to the example presented before is that not the original data, but a distance matrix is handed over as input to the analysis. It can be obtained with the function `dist` (uses the Euclidean distance by default) or the extended function `vegdist`, respectively (which gives the Bray-Curtis dissimilarity by default, but can also deliver the Euclidean distance, among others):

```
pco <- cmdscale(vegdist(dat, method="euclidean"), k=2, eig=TRUE)
plot(pco$points, type="n")
text(pco$points, row.names(dat))
```

The parameter `k` defines the number of dimensions to be extracted, parameter `eig` allows the return of the eigenvalues, which are important for plotting.

NMDS

With the help of the function `isoMDS` we get an ordination in which the configuration of the PCO is used as a starting point. Please set `trace` to `TRUE` in order to display intermediate results.

```
mds <- isoMDS(vegdist(dat, method="euclidean"), trace=FALSE)
mds$stress
```

```
[1] 1.670176e-14
```

```
plot(mds$points, type="n")
text(mds$points, row.names(dat))
```

An MDS can also be run with a random start (`initMDS`) and it is possible to scale the results with `postMDS` to make them easier to interpret.

```
dist <- vegdist(dat, method="euclidean")
mds <- isoMDS(dist, initMDS(dist), maxit=200, trace=FALSE)
mds <- postMDS(mds, dist)
mds$stress
```

```
[1] 0.0006361459
```

```
plot(mds$points, type="n")
text(mds$points, row.names(dat))
```

If a random starting configuration is used, it is usually advisable to run several ordinations and to save the best results separately. The function `metaMDS` from the `vegan` package performs such an MDS with multiple random starts automatically and returns the best result of all these trials. But as `metaMDS` is tailored for ecological data, a suitable is shown a little below in the text.

A Shepard plot can be received by:

```
stressplot(mds, dist(dat))
mds$stress
```

```
[1] 0.0006361459
```

and in order to illustrate a bad fit, let's create also a pure random data set with 10 dimensions and uniformly distributed random numbers $U(0,1)$ for comparison:

```
set.seed(123)
rdat <- matrix(runif(100), 10)
mds  <- isoMDS(d<-vegdist(rdat, method="euclid"), trace=FALSE)
stressplot(mds, d, pch=16)
mds$stress
```

```
[1] 10.22633
```

The Shepard diagram plots the ordination distances against the original distances. The fit is shown as a stepped curve. The closer the points are to the stepped curve, the better the fit. The coefficients of determination (R^2) are almost always very high and should thus be treated with caution, for details see OKSANEN (2010).

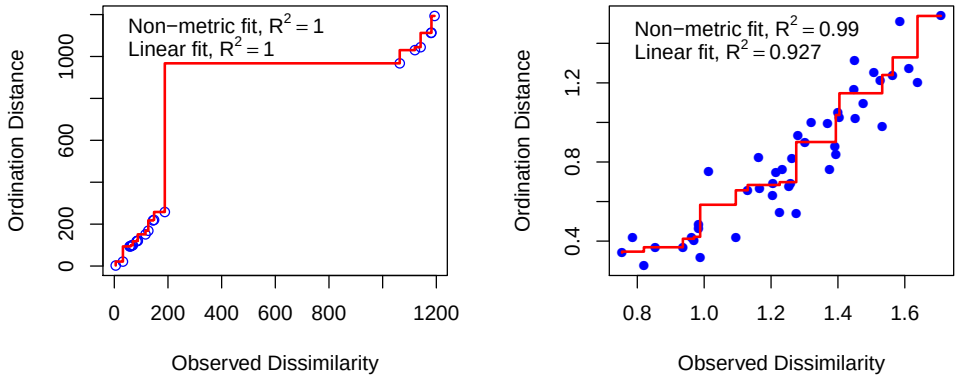


Figure 9.2: Shepard plot for the lake data set (left) and for uniformly distributed random numbers.

Hierarchical Cluster Analysis

Hierarchical cluster analysis is easy to apply. As the calculation of the distance is done repeatedly during clustering, the function for distance calculation (`dist` or `vegdist`) has to be handed over directly.

Because of the different measurement units the lake data set needs to be standardized. The Euclidean distance is the distance measure of choice, as the data are metric and the Ward method is an appropriate agglomeration algorithm. For comparison we will use three more agglomeration methods:

```
par(mfrow=c(2,2))
plot(hclust(dist(scale(dat), method="euclid"), method="complete"), main="Complet
plot(hclust(dist(scale(dat), method="euclid"), method="single"), main="Single")
plot(hclust(dist(scale(dat), method="euclid"), method="average"), main="Average"
plot(hclust(dist(scale(dat), method="euclid"), method="ward"), main="Ward")
```

Cluster Analysis with Mahalanobis Distance

If the Mahalanobis distance is to be used it does not suffice to calculate it in the beginning and to then hand over the distance matrix to `hclust`, for the distance calculation is needed during clustering, too. At time of writing neither `dist` nor `vegdist` did contain Mahalanobis distance. However, a perfectly valid alternative would be to transform the original coordinates in a way that the Euclidean distance of the transformed objects is the same as the Mahalanobis distance of the non-transformed objects. This can be done with an eigenvalue decomposition, and is easiest to do via PCA (without standardization). Afterwards, all principal components are re-scaled to mean zero and unit standard deviation, what in consequence gives greater weight to the otherwise unimportant higher components. For our lake example this means that rather unimportant differences (and measurement errors) receive a very high importance, possibly resulting in an unexpected picture (fig. 9.4, left):

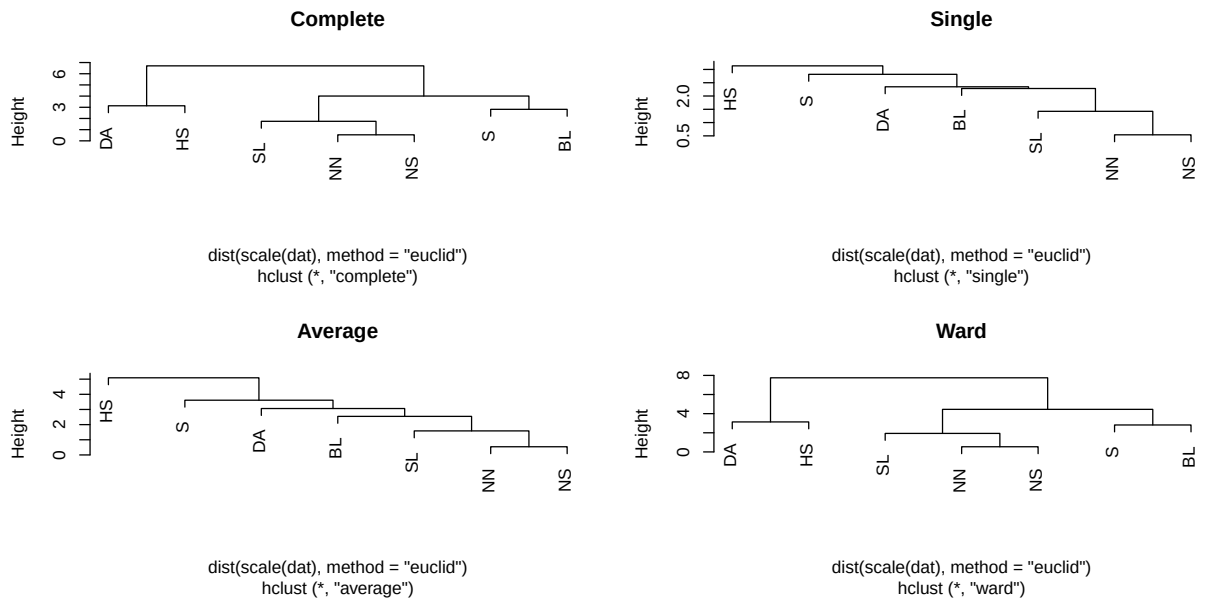


Figure 9.3: Cluster analysis for the lake data set.

```
pc <- prcomp(dat)
pcdata <- as.data.frame(scale(pc$x))
cl <- hclust(dist(pcdata), method="ward")
plot(cl, main="PC 1...7")
```

To demonstrate that the unexpected behavior is caused by the last components which were given more weight, we willingly omit the last 2 components and receive the expected picture (fig. 9.4, right):

```
plot(hclust(dist(pcdata[,1:4]), method="ward"), main="PC 1...4")
```

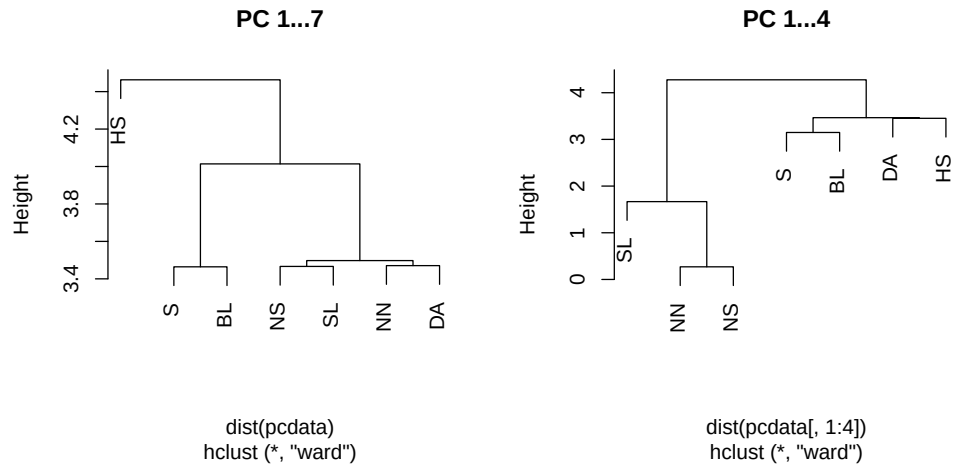


Figure 9.4: Cluster anlysis with all (left) and only the first four principal components (right).

What can be seen here is that the Mahalanobis distance is better suited for problems in which other methods cannot achieve a sufficient separation, as might be the case in the analysis of peaks in a chemical analysis, for

instance. The bottom line is that the choice of a suitable measure of distance should be done very carefully, as radically different results can occur, according to one's choice.

Combination of NMDS and k-means Clustering

NDMS can be usefully combined with cluster analysis, e.g. by highlighting the groups created in different colours in plots. Particularly simple is a combination of NDMS with non-hierarchical k-means clustering. In k-means clustering he number of clusters to be created is defined beforehand, e.g. `centers = 3`, depending on the result of hierarchical clustering (fig. 9.5):

```
dat <- read.table("http://www.simecol.de/data/seen_bb.txt",header=TRUE)
row.names(dat) <- dat[,1]
dat <- dat[,-1]
mds <- isoMDS(vegdist(dat, method="bray"))
```

```
initial  value 1.486552
iter   5 value 0.589242
iter  10 value 0.012067
iter  10 value 0.000134
iter  10 value 0.000000
final  value 0.000000
converged
```

```
km <- kmeans(dat, centers = 3)
plot(mds$points, type="n")
text(mds$points, row.names(dat), col=km$cluster)
```

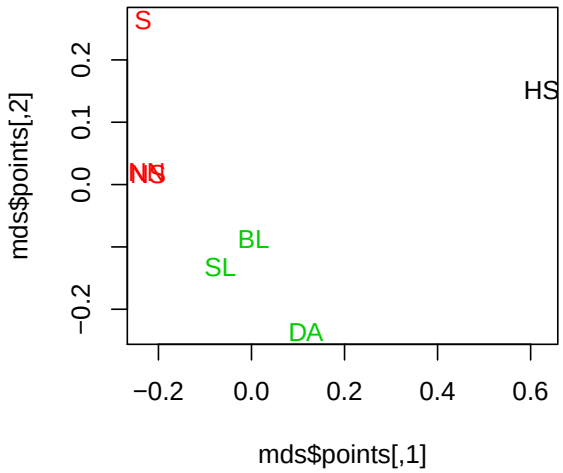


Figure 9.5: Combination of NMDS and k-Means Clustering.

9.6.2 Example 2: A Data Set from the Vegan Package

The following section is a short summary of some examples contained within the online help and the **vegan** tutorial (OKSANEN, 2010). The data set consists of two matrices, a species list (community matrix) `varespec` and the environmental matrix `varechem`. It is a test data set contained directly within the **vegan** package and has been taken from a publication by VÄRE *et al.* (1995):

```
library(vegan)
data(varechem)
data(varespec)
```

NMDS

An MDS can be got fully automatic by:

```
mds <- metaMDS(varespec)
plot(mds, type = "t")
```

in which the built-in automatism does the following steps:

1. if ”necessary“, a square root transformation and a Wisconsin double standardization are performed (initially for each species: abundances by maximum abundance of the species, afterwards for sampling sites: abundance by total abundance),
2. the Bray-Curtis dissimilarity is applied,
3. several random starts are performed, and the result is compared to the best solution with a Procrustes test,
4. to make interpretation easier a rotation is done, so the greatest variance of the site-scores lies on axis 1,
5. A scaling is done, so 1 unit equals 50% of the community similarity and replicate “similarity”,
6. the species scores are added to the final configuration as weighted means of the environmental variables. This yields a biplot.

Although, by all means, most steps make a lot of sense, I strongly recommend to take full responsibility for transformation and standardization in order to achieve reprodcuible results, to determine yourself if you want standardization and transformation:

```
mds <- metaMDS(varespec, distance = "bray", autotransform = FALSE)
```

or with square root and Wisconsin transformation:

```
mds <- metaMDS(wisconsin(sqrt(varespec)), distance = "bray",
  autotransform = FALSE, trace=FALSE)
mds
```

```
Call:
metaMDS(comm = wisconsin(sqrt(varespec)), distance = "bray",      autotransform =

global Multidimensional Scaling using monoMDS
```

```
Data:     wisconsin(sqrt(varespec))
Distance: bray

Dimensions: 2
Stress:     0.1825658
Stress type 1, weak ties
Two convergent solutions found after 13 tries
Scaling: centring, PC rotation, halfchange scaling
Species: expanded scores based on 'wisconsin(sqrt(varespec))'
```

Intermediate results can be displayed using the trace function (highly recommended), or can be turned off (to prevent this tutorial from becoming too long).

To see if a transformation is reasonable on could simply "ask" the automatism.

Graphical output can be produced by using the generic plotting function or a special function `ordiplot`. function `ordiplot`

```
plot(mds)
plot(mds, type="t")
plot(mds, display="sites")
plot(mds, display="species")
```

To find a detailed description see the online help and in particular the **vegan** tutorial (OKSANEN, 2010).

```
mds <- metaMDS(varespec, distance = "bray", autotransform = FALSE, k = 3)
ordirgl(mds, type = "t", col = "yellow")
orgltext(mds, text = names(varespec), display = "species",col = "cyan")
axes3d()
```

In this context the parameter `k=3` allows a three-dimensional NMDS

Vector Fitting

The effect of environmental variables can be studied with the help of so-called *vector fitting* (fig. 9.6), which includes a permutation test (e.g. with 1000 randomizations):

```
mds <- metaMDS(varespec, trace = FALSE)
ef <- envfit(mds, varechem, permu = 1000)
ef
```

```
***VECTORS

             NMDS1     NMDS2     r2   Pr(>r)
N        -0.057256 -0.998360 0.2537 0.046953 *
P         0.619656  0.784874 0.1938 0.088911 .
K         0.766386  0.642380 0.1809 0.116883
Ca        0.685135  0.728416 0.4119 0.003996 **
Mg        0.632466  0.774588 0.4270 0.000999 ***
S         0.191305  0.981531 0.1752 0.128871
Al       -0.871641  0.490144 0.5269 0.000999 ***
```

```
Fe         -0.936060  0.351841 0.4450 0.005994 **
Mn          0.798728 -0.601692 0.5231 0.001998 **
Zn          0.617519  0.786556 0.1879 0.105894
Mo         -0.903088  0.429455 0.0609 0.515485
Baresoil    0.924939 -0.380116 0.2508 0.045954 *
Humdepth    0.932860 -0.360239 0.5200 0.000999 ***
pH         -0.648031  0.761614 0.2308 0.065934 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
P values based on 1000 permutations.
```

```
plot(mds, type = "t")
plot(ef, p.max = 0.1)
```

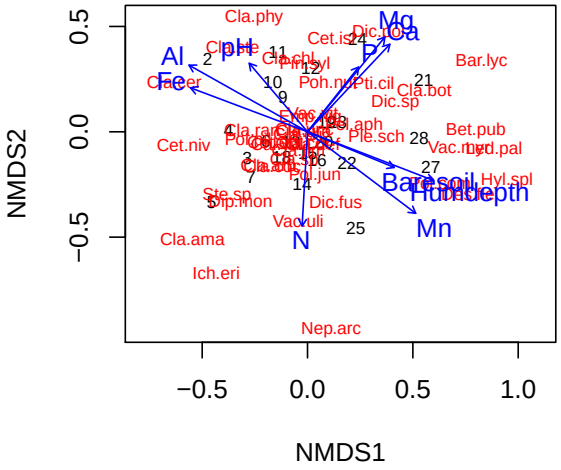


Figure 9.6: NMDS of the varespec data set wit fitted environmental vectors.

The result is a triplot. In the above case initially (and naively) all possible environmental variables were fitted, but in the plot only variables with a p-value < 0.1 were drawn. It is better to specify the variables we are interested in beforehand, and to describe them with a model formula:

```
ef <- envfit(mds ~ Al + Ca, varechem, permu = 1000)
ef
plot(mds, type = "t")
plot(ef)
```

A problem of vector fitting is that in doing so a linear relationship between the environmental vectors and the ordination is assumed. This is not always the case, the more so as the NMDS creates a non-metrical distortion.

The function `ordisurf` (*surface fitting*) applies a GAM approach (generalized additive model) to represent a possible non-linearity. In the following example vector fitting and surface fitting are presented for the variables Al and Ca (fig. 9.7):

```
ef <- envfit(mds ~ Al + Ca, varechem)
plot(mds, display = "sites", type = "t")
plot(ef)
tmp <- with(varechem, ordisurf(mds, Al, add = TRUE))
with(varechem, ordisurf(mds, Ca, add = TRUE,
col = "green4"))
```

```
Family: gaussian
Link function: identity

Formula:
y ~ s(x1, x2, k = 10, bs = "tp", fx = FALSE)
<environment: 0x0000000018b7c3d8>

Estimated degrees of freedom:
4.72  total = 5.72

REML score: 156.6552
```

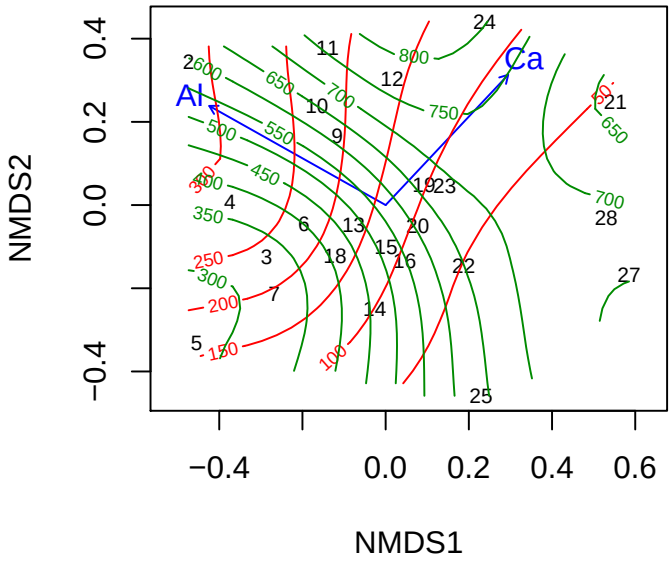


Figure 9.7: NMDS of the varespec data set with fitted environmental vectors and GAM isolines for Al (red) and Ca (green).

A three-dimensional plot can be achieved with `vis.gam(tmp)`.

Mantel Test

The Mantel test can be used to examine if there is a relation between an environmental factor and a species list:

```
veg.dist <- vegdist(varespec)
env.dist <- vegdist(scale(varechem), "euclid")
mantel(veg.dist, env.dist)
```

```
Mantel statistic based on Pearson's product-moment correlation

Call:
mantel(xdis = veg.dist, ydis = env.dist)

Mantel statistic r: 0.3047
      Significance: 0.001

Upper quantiles of permutations (null model):
  90%   95% 97.5%   99%
0.112 0.147 0.175 0.205

Based on 999 permutations
```

Please note that the Bray-Curtis dissimilarity was used for the species list (default) and the Euclidean distance (after scaling) for the environmental factors. Instead of standardized environmental variables it is possible to use a distance matrix of the most important principal components.

For visualization the distances can be plotted directly against each other:

```
plot(veg.dist, env.dist)
```

at which the relationship should look more or less linear or monotonous.

BIOENV

Bioenv is a method used to select the best subset of environmental variables, in which the Euclidean distance has the highest rank correlation with the species dissimilarity matrix:

```
sol <- bioenv(wisconsin(varespec) ~ log(N) +
P + K + Ca + pH + Al, varechem)
summary(sol)
```

```
                    size correlation
P                      1      0.2513
P Al                   2      0.4004
P Ca Al                3      0.4005
P Ca pH Al             4      0.3619
log(N) P Ca pH Al      5      0.3216
log(N) P K Ca pH Al    6      0.2822
```

Thus, in our example that subset is P and Al or P, Ca and Al, respectively.

CCA

Canonical Correspondence Analysis is a two-matrix method, which on the one hand uses the Chi-square distance, and on the other hand produces two kinds of axes. One type are the so-called *constrained* axes,

which contains only the portion of total information (named Total Inertia) that can be explained by the environmental variables. The other type are the remaining *unconstrained* axes. So it has to be noted that, in contrast to NMDS, the plot normally contains only that portion of the species list that can be explained by the environmental matrix.

```
vare.cca <- cca(varespec, varechem)
vare.cca
```

```
Call: cca(X = varespec, Y = varechem)

              Inertia Proportion Rank
Total          2.0832     1.0000
Constrained    1.4415     0.6920   14
Unconstrained  0.6417     0.3080    9
Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
    CCA1     CCA2     CCA3     CCA4     CCA5     CCA6     CCA7     CCA8
0.438870 0.291775 0.162847 0.142130 0.117952 0.089029 0.070295 0.058359
    CCA9    CCA10    CCA11    CCA12    CCA13    CCA14
0.031141 0.013294 0.008364 0.006538 0.006156 0.004733

Eigenvalues for unconstrained axes:
     CA1      CA2      CA3      CA4      CA5      CA6      CA7      CA8
0.197765 0.141926 0.101174 0.070787 0.053303 0.033299 0.018868 0.015104
     CA9
0.009488
```

```
plot(vare.cca)
```

The number of *constrained axes* is equal to the number of environmental variables. This means that a high number of environmental variables leads to more and more degrees of freedom and the CCA approaches the CA, ultimately. Many constraints mean "no constraints", practically. Therefore, it makes sense to specify possible explanatory variables a priori.

```
vare.cca <- cca(varespec ~ P + Ca + Al, varechem)
vare.cca
```

```
Call: cca(formula = varespec ~ P + Ca + Al, data = varechem)

              Inertia Proportion Rank
Total          2.0832     1.0000
Constrained    0.5243     0.2517    3
Unconstrained  1.5589     0.7483   20
Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
   CCA1    CCA2    CCA3
0.34534 0.14890 0.03007
```

```
Eigenvalues for unconstrained axes:
    CA1     CA2     CA3     CA4     CA5     CA6     CA7     CA8
0.36940 0.22296 0.20489 0.17934 0.13416 0.10079 0.08534 0.07717
(Showed only 8 of all 20 unconstrained eigenvalues)
```

```
plot(vare.cca)
```

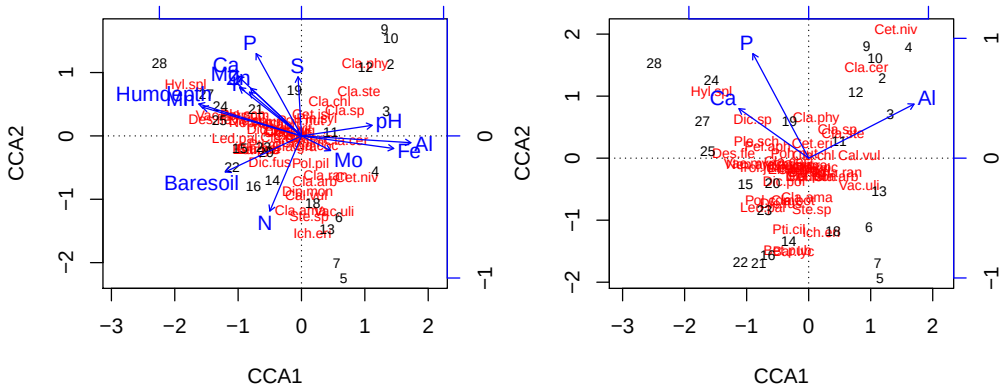


Figure 9.8: CCA of the varespec data set with the complete environmental matrix as constraint (left) or P, Ca and Al only (right).

An RDA is performed following the same pattern. As this method makes use of the Euclidean distance for the species matrix, it is nowadays advisable in exceptional cases only, ans should rather be applied for problems with two matrices with metric data.

Bibliography

ADLER, J., 2010: R in a Nutshell. O' Reiley.

ANDERSON, M., 2001: A new method for non-parametric multivariate analysis of variance. Austral Ecology **26**: 32–46.

BAI, J. and P. PERRON, 2003: Computation and analysis of multiple structural change models. J. Appl. Econ. **18:**: 1–22.

BATES, D. M. and D. G. WATTS, 1988: Nonlinear Regression and its Applications. Wiley and Sons, New York.

BOX, G. E., G. M. JENKINS and G. C. REINSEL, 1994: Time Series Analysis: Forecasting and Control. Prentice Hall Inc., Englewood Cliffs, NJ, third edition.

BRUN, R., P. REICHERT and H. KÜNSCH, 2001: Practical identifiability analysis of large environmental simulation models. Water Resources Research **37**: 1015–1030.

CANTY, A. and B. RIPLEY, 2009: **boot**: Bootstrap R (S-Plus) Functions. R package version 1.2-41.

CLEVELAND, R. B., W. S. CLEVELAND, J. MCRAE and I. TERPENNING, 1990: STL: A seasonal-trend decomposition procedure based on loess. Journal of Official Statistics **6**: 3–73.

CLEVELAND, W. S., 1981: LOWESS: A program for smoothing scatterplots by robust locally weighted regression. The American Statistician **35**: 54.

CRAWLEY, M. J., 2002: Statistical Computing. An Introduction to Data Analysis Using S-PLUS. Wiley, Chichester.

CRAWLEY, M. J., 2012: The R book. John Wiley & Sons.

DALGAARD, P., 2002: Introductory Statistics with R. Springer.

DALGAARD, P., 2008: Introductory Statistics with R. Springer, second edition.

DRAY, S. and A.-B. DUFOUR, 2007: The **ade4** package: Implementing the duality diagram for ecologists. Journal of Statistical Software **22**. URL `http://www.jstatsoft.org/v22/i04/`.

DURBIN, J. and S. J. KOOPMAN, 2001: Time Series Analysis by State Space Methods. Oxford University Press. URL `http://www.ssfpack.com/DKbook.html`.

GRIMM, H. and R.-D. RECKNAGEL, 1985: Grundkurs Biostatistik. Gustav Fischer Verlag, Jena.

HÅKANSON, L. and R. H. PETERS, 1995: Predictive Limnology: methods for predictive modelling. SPB Academic Publishing, Amsterdam.

HALL, D. J., 1964: An experimental approach to the dynamics of a natural population of daphnia galeata mendotae. Ecology **45**: 94–112.

HARRELL, F. E., 2001: Regression Modeling Strategies, with Applications to Linear Models, Survival Analysis and Logistic Regression. Springer. URL `http://biostat.mc.vanderbilt.edu/twiki/bin/view/Main/RmS`, iSBN 0-387-95232-2.

HYNDMAN, R. J. and Y. KHANDAKAR, 2008: Automatic time series forecasting: The forecast package for r. Journal of Statistical Software **27**: 1–22. URL `http://www.jstatsoft.org/v27/i03`.

IHLE, T., S. JÄHNICHEN and J. BENNDORF, 2005: Wax and wane of *Microcystis* and microcystins in lake sediments: a case study in Quitzdorf Reservoir (Germany). J. Phycol. **41**: 479–488.

JÄHNICHEN, S., T. PETZOLDT and J. BENNDORF, 2001: Evidence for control of microcystin dynamics in Bautzen Reservoir (Germany) by cyanobacterial population growth rates and dissolved inorganic carbon. Archiv für Hydrobiologie **150**: 177–196.

JOHNSON, G., JERALD and K. S. OMLAND, 2004: Model selection in ecology and evolution. Trends in Ecology and Evolution **19**: 101–108.

KLEIBER, C. and A. ZEILEIS, 2008: Applied Econometrics with R. Springer.

KÖHLER, W., G. SCHACHTEL and P. VOLESKE, 2002: Biostatistik. Eine Einführung in die Biometrie für Biologen und Agrarwissenschaftler. Springer-Verlag, Berlin, Heidelberg, second edition.

KOSCHEL, R., J. BENNDORF, J. PROFT and F. RECKNAGEL, 1987: Model-assisted evaluation of alternative hypothesis to explain the self-protection mechanism of lakes due to calcite precipitation. Ecol. Model. **39**: 59–65.

LEGENDRE, P. and L. LEGENDRE, 1998: Numerical Ecology. Elsevier, second edition.

LEYER, I. and K. WESCHE, 2007: Multivariate Statistik in der Ökologie eine Einführung. Springer.

MCLEOD, A., 2009: Kendall: Kendall rank correlation and Mann-Kendall trend test. URL `http://CRAN.R-project.org/package=Kendall`, r package version 2.1.

OKSANEN, J., 2010: Multivariate Analysis of Ecological Communities in R: **vegan** Tutorial. URL `http://cc.oulu.fi/~jarioksa/opetus/metodi/vegantutor.pdf`.

QIAN, S. S., 2009: Environmental and Ecological Statistics with R. Chapman and Hall, Boca Raton. URL `http://www.duke.edu/~song/eeswithr.htm`.

SACHS, L., 1992: Angewandte Statistik. Springer-Verlag, Berlin, 7th edition.

SCHLITTGEN, R. and B. H. J. STREITBERG, 1989: Zeitreihenanalyse. R. Oldenbourg Verlag, Wien. 3. Auflage.

SHUMWAY, R. H. and D. S. STOFFER, 2006: Time Series Analysis and Its Applications: With R Examples. Springer.

SOETAERT, K. and T. PETZOLDT, 2010: Inverse modelling, sensitivity and Monte Carlo analysis in R using package FME. Journal of Statistical Software **33**: 1–28. URL `http://www.jstatsoft.org/v33/i03`.

THIOULOUSE, J. and S. DRAY, 2007: Interactive multivariate data analysis in R with the **ade4** and **ade4TkGUI** packages. Journal of Statistical Software **22**. URL `http://www.jstatsoft.org/v22/i06/`.

VENABLES, W. N. and B. D. RIPLEY, 2002: Modern Applied Statistics with S. Springer, New-York, fourth edition.

VENABLES, W. N., D. M. SMITH and THE R CORE TEAM, 2012: An Introduction to R: A Programming Environment for Data Analysis and Graphics. ISBN 3-900051-12-7, http://www.R-project.org.

VENABLES, W. N., D. M. SMITH and THE R DEVELOPMENT CORE TEAM, 2001: An Introduction to R: A Programming Environment for Data Analysis and Graphics, Version 1.4.0. www.r-project.org.

VOLLENWEIDER, R. A. and J. KEREKES, 1980: OECD cooperative programme for monitoring of inland waters. (Eutrophication control). Synthesis report, Organisation for Economic Co-operation and Development, Paris.

VÄRE, H., R. OHTONEN and J. OKSANEN, 1995: Effects of reindeer grazing on understorey vegetation in dry pinus sylvestris forests. Journal of Vegetation Science **6**: 523–530.

WOOD, S. N., 2006: Generalized Additive Models: An Introduction with R. Chapman & Hall.

ZAR, J. H., 1996: Biostatistical Analysis. Prentice-Hall, Upper Saddle River, NJ, third edition.

ZEILEIS, A., C. KLEIBER, W. KRÄMER and K. HORNIK, 2003: Testing and dating of structural changes in practice. Computational Statistics and Data Analysis **44**: 109–123.

ZEILEIS, A., F. LEISCH, K. HORNIK and C. KLEIBER, 2002: `strucchange`: An R package for testing for structural change in linear regression models. Journal of Statistical Software **7**: 1–38. URL `http://www.jstatsoft.org/v07/i02/`.