

Computational Statistics

Introduction to

Günther Sawitzki
StatLab Heidelberg
private version

March 18, 2008

in preparation



Contents

| | |
|--|----------|
| Introduction | 9 |
| 1 Basic Data Analysis 1-1 | |
| 1.1 R Programming Conventions | 1-1 |
| 1.2 Generation of Random Numbers and Patterns | 1-4 |
| 1.2.1 Random Numbers | 1-4 |
| 1.2.2 Patterns | 1-9 |
| 1.3 Case Study: Distribution Diagnostics | 1-10 |
| 1.3.1 Distribution Functions | 1-12 |
| 1.3.2 Histogram and Related Plots | 1-15 |
| Bar Charts | 1-19 |
| 1.3.3 Statistics of Distribution Functions; Kolmogorov-Smirnov-Tests | 1-21 |
| Monte Carlo Confidence Bands | 1-23 |
| 1.3.4 Statistics of Histograms and Related Plots; χ^2 -Tests | 1-29 |
| 1.4 Moments and Quantiles | 1-34 |
| 1.5 R Complements | 1-39 |
| 1.5.1 Random Numbers | 1-39 |
| 1.5.2 Graphical Comparisons | 1-40 |
| 1.5.3 Enhancing Graphical Displays | 1-46 |
| 1.5.4 Functions | 1-47 |
| 1.5.5 R Internals | 1-51 |
| Parse | 1-51 |
| Eval | 1-51 |
| Print | 1-52 |

| | |
|---|------------|
| Executing Files | 1-52 |
| 1.5.6 Packages | 1-52 |
| 1.6 Statistical Summary | 1-54 |
| 1.7 Literature and Additional References | 1-54 |
| 2 Regression | 2-1 |
| 2.1 General Regression Model | 2-1 |
| 2.2 Linear Model | 2-2 |
| 2.2.1 Factors | 2-4 |
| 2.2.2 Least Squares Estimation | 2-5 |
| 2.2.3 More Examples for Linear Models | 2-15 |
| 2.2.4 Model Formulae | 2-16 |
| 2.2.5 Gauss-Markov Estimator | 2-17 |
| 2.3 Variance Decomposition and Analysis of Variance | 2-19 |
| 2.4 Simultaneous Inference | 2-26 |
| 2.4.1 Scheffé's Confidence Bands | 2-26 |
| 2.4.2 Tukey's Confidence Intervals | 2-28 |
| Case Study: Titre Plates | 2-29 |
| 2.5 Beyond Linear Regression | 2-34 |
| Transformations | 2-34 |
| Box-Cox Transformations | 2-34 |
| 2.5.1 Generalized Linear Models | 2-35 |
| 2.5.2 Local Regression | 2-35 |
| 2.6 R Complements | 2-39 |
| 2.6.1 Discretisation | 2-39 |
| 2.6.2 External Data | 2-39 |
| 2.6.3 Testing Software | 2-40 |
| 2.6.4 R Data Types | 2-41 |
| 2.6.5 Classes and Polymorphic Functions | 2-42 |
| 2.6.6 Extractor Functions | 2-42 |
| 2.7 Statistical Summary | 2-43 |
| 2.8 Literature and Additional References | 2-43 |

| | |
|---|------------|
| CONTENTS | 5 |
| 3 Comparison of Distributions | 3-1 |
| 3.1 Shift/Scale Families | 3-4 |
| 3.2 <i>QQ</i> plot, <i>PP</i> plot, and Comparison of Distributions | 3-6 |
| 3.2.1 Kolmogorov Smirnov Tests | 3-11 |
| 3.3 Tests for Shift Alternatives | 3-12 |
| 3.4 Power and Confidence | 3-20 |
| 3.4.1 Theoretical Power and Confidence | 3-20 |
| 3.4.2 Simulated Power and Confidence | 3-23 |
| 3.4.3 Quantile estimation by Simulation | 3-26 |
| 3.5 Qualitative Features of Distributions | 3-28 |
| 3.6 R Complements | 3-29 |
| 3.7 Statistical Summary | 3-31 |
| 3.8 Literature and Additional References | 3-31 |
| 4 Dimensions 1, 2, 3, ..., ∞ | 4-1 |
| 4.1 R Complements | 4-1 |
| 4.2 Dimensions | 4-3 |
| 4.3 Selections | 4-5 |
| 4.4 Projections | 4-6 |
| 4.4.1 Marginal Distributions and Scatter Plot Matrices | 4-7 |
| 4.4.2 Projection Pursuit | 4-12 |
| 4.4.3 Projections for Dimensions 1, 2, 3, ... 7 | 4-14 |
| 4.4.4 Parallel Coordinates | 4-15 |
| 4.5 Sections, Conditional Distributions and Coplots | 4-17 |
| 4.6 Transformations and Dimension Reduction | 4-23 |
| 4.7 Higher Dimensions | 4-28 |
| 4.7.1 Linear Case | 4-28 |
| Partial Residuals and Added Variable Plots | 4-28 |
| 4.7.2 Non-linear Case | 4-30 |
| Example: Cusp Non-linearity | 4-30 |
| 4.7.3 “Curse of Dimension” | 4-35 |
| 4.7.4 Case Study | 4-35 |
| 4.8 High Dimensions | 4-47 |
| 4.9 Statistical Summary | 4-47 |
| 4.10 Literature and Additional References | 4-49 |

| | |
|--|-------------|
| R as a Programming Language and Environment | A-51 |
| A.11 Help and Information | A-51 |
| A.12 Names and Search Paths | A-51 |
| A.13 Customizing | A-52 |
| A.14 Basic Data Types | A-54 |
| A.15 Output for Objects | A-56 |
| A.16 Object Inspection | A-57 |
| A.17 System Inspection | A-58 |
| A.18 Complex Data Types | A-59 |
| A.19 Accessing Components | A-61 |
| A.20 Data Manipulation | A-63 |
| A.21 Operators | A-65 |
| A.22 Functions | A-66 |
| A.23 Debugging and Profiling | A-68 |
| A.24 Control Structures | A-70 |
| A.25 Administration and Customisation | A-72 |
| A.26 Input and Output to Data Streams | A-73 |
| A.27 External Data | A-74 |
| A.28 Libraries, Packages | A-76 |
| A.29 Linear Algebra | A-77 |
| A.30 Model Descriptions | A-78 |
| A.31 Graphic Functions | A-80 |
| A.31.1 High Level Graphics | A-80 |
| A.31.2 Low Level Graphics | A-80 |
| A.31.3 Annotations and Legends | A-81 |
| A.31.4 Graphic Parameters and Layout | A-82 |
| A.32 Basic Statistical Functions | A-83 |
| A.33 Distributions, Random Numbers, Densities... | A-84 |
| A.34 Programming on the Language | A-86 |
| Bibliography | A-87 |

| | |
|---|-------------|
| CONTENTS | 7 |
| Functions by Topic | A-89 |
| Function and Variable Index | A-90 |
| Index | A-94 |
| B | B-1 |
| B.1 Test R function index | B-1 |
| B.2 R styles | B-2 |
| B.3 Tables | B-6 |
| C S help — remove for public version | C-1 |
| C.1 Ch 01 | C-1 |
| D To Do | D-1 |
| D.1 For release | D-1 |
| D.2 misc | D-1 |



Introduction

ch:00

This introduction to R is intended as course material to be used in a compact course or for self-instruction. The course is for students with basic knowledge in stochastics. Notions like distribution function, quantile, expected value and variance are presupposed, as well as some familiarity with statistical features corresponding to these notions. Classical distributions, such as binomial, uniform, and gaussian should be familiar, together with derived distributions and their asymptotic behavior. It is not required that the reader has knowledge in statistics proper. But this knowledge is not taught in this course, on the other hand. This course concentrates on the “computing” aspects. Statistical concepts and statistical points of view are introduced and discussed. But for an in depth discussion, the reader is referred to statistics courses,

A working knowledge in computer usage is presupposed, at least rudimentary knowledge of programming concepts like variables, loops, functions. No extended knowledge in computing is required.

s:01

What is R?

R is a programming language, and the name of a software system which implements this language. The R programming language has been developed for statistics and stochastic simulation. By now, it has become a standard in these fields. To be precise, we should use specific terms: the language is called S, its implementation and the environment are called R. The original authors of S are John M. Chambers, R. A. Becker and A. R. Wilks, AT & T Bell Laboratories, Statistics Research Department. The language and its development are documented in a series of books, commonly called by their cover as white ([3]), blue ([1]) and green book ([2]).

For a long time the AT & T implementation of S has been the reference for the S language. Today, S is available in a commercial version called S-Plus <<http://www.insightful.com/>> (based on the AT & T implementation) and as a free software version R*, or “Gnu S” <<http://www.r-project.org/>>.



* R got its name by accident - the same accident which made the first names of the original authors of R (Ross Ihaka & Robert Gentleman) start with R.

In the mean time, R has become the reference implementation. Essential precisions and - if necessary - even modifications of the language are defined by R. For simplicity, here and in the sequel we use “R language” as a common term even where the precise term should be: the S language using the R implementation.

R is an interpreted programming language. Instructions in R are executed immediately. Besides the original elements of S, R has several extensions. Part of these are introduced in response to recent developments in statistics, part are introduced to open experimental facilities. On the other side, advancements of the S language are taken into account.

The recent (2008) version of R is 2.x. This version is largely compatible with the previous version R 1.x. The essential changes are internal to the system. For initial use, there is no significant difference from R 1.x. For the advanced user, there are three essential innovations:

- *Graphics*: The basic graphic system in R implements a models inspired by pen and paper drawing. A graphics port (paper) is opened, and lines, points or symbols are drawn in this port. In R 2.x there is a second additional graphics system, oriented at a camera/object model. Graphical object in various position and orientation are mapped in a visual space.
- *Packages*: The original R had a linear command history and a uniform work space. R 2.x introduced an improved support for “packages” which may have encapsulated information. To support this, language concepts such as “name spaces” and various tools have been introduced.
- *Internationalization*: Originally, R was based on the English language, and ASCII was the general encoding used. With R 2.x extensive support for other languages and encodings has been introduced. With this, it has become possible to provide localized versions.

Two aspects of R are active areas of recent developments: interactive access, and integration in a networked environment. These and other aspects are part of Omegahat - an attempt to develop a next generation system based on the experiences from R . These more experimental work is accessible at <<http://www.omegahat.org/>>. R does already provide simple possibilities to call functions implemented in other languages like C or FORTRAN. Omegahat extends these possibilities and allows direct access to Java, Perl A Java based graphical interface for R is JGR, accessible at <<http://stats.math.uni-augsburg.de/software/>>. A collection of interactive displays for R is in *ipplots*, available at the same site.

Recent developments related to R are in <<http://r-forge.r-project.org/>>. Many helpful extensions are in <<http://www.bioconductor.org/>>, a site which is targeted at biocomputing.

s:02 R has been developed for practical work in statistics. Usability often has been given priority over abstract design principles. As a consequence, it is not easy to give a systematic introduction to R. A devoured path is chosen instead: case studies and examples, followed by systematic surveys. Practical work should make use of the rich online material which comes with R. Starting points are the “frequently asked questions” (FAQ)

<<http://www.cran.r-project.org/faqs.html>>. “An Introduction to R” ([14]) is the “official” introduction. This documentation and other manuals can be downloaded from <<http://www.cran.r-project.org/manuals.html>>.

Many R functions come with the basic system. Other functions must be loaded from libraries. Some of these libraries come with the R system. If additional packages are needed, <<http://www.cran.r-project.org/>> is a first resource for downloads.

The commercial S-Plus comes in various versions. S-Plus 4.x and S-Plus 2000 use S version 3 and are largely compatible with R. S-Plus 5 is an implementation of S version 4 with some changes which need attention. These programming details of S-Plus are not discussed here. Information about S-Plus is at <<http://www.insightful.com/>>.

s:03 Content and Outline of this Course

In its basic version, R contains more than 1500 functions - too many to introduce in just one course, and too many to learn reasonably. The course can only open the way to R.

Course participants can come from various background, with different prerequisites. For pupils and younger students, a mere programming course on technical basics may be appropriate. Later in study, questions about meaningful classification and background will be more important. This is what the present course aims at. The “technical” material gives a skeleton. Beyond this, we try to open the view for statistical questions and rise interest for the background. The course should sharpen the appetite for the substance which may be offered in a subsequent well-founded statistical course.

The first part of this course material is organized by themes, using example topics to illustrate how R can be used to tackle statistical problems.

An appendix gives a collection of R language elements and functions. During the course, it can help es a quick reference and at may serve as a starting point and orientation to access the rich information material which comes bundled with R. After the course, it may serve as a note pad. Finally, on the long run for practical work, the online help and online manuals of R are the prime sources of information.

Using a selection of the exercises, the course can be completed within about four days of work. Conceptually, it is a four day introduction to statistics with the topic areas

- One sample analysis and distributions
- Regression
- Two sample problems and comparison of distributions
- Multivariate analysis

A generous time slot for exercises is recommended (an additional half day for the introductory exercises, an additional half day for one of the project exercises). With this, the course can be covered in one week, provided follow up facilities are established to answer questions which have come up, and possibilities are available to follow the interest in statistical background which may have resulted.

ToDo: examples of help files included

ToDo:

comment
on litera-
ture

ToDo:

comment
ch.4 exper-
imental

expl:00

Using the course during a term in a weekly class needs more time, since repetitions must be calculated in. Each of the first four chapters will cover about four lectures, plus time for exercises. For this time schedule, a course to cover the statistical background is recommended running in parallel.

For a subsequent self-paced study going into details on R as a programming language, the recommended reading is ([24]).

Typographical Conventions

Examples and input code are formatted so that they can be used with “Cut & Paste” and entered as program input. To allow this, punctuation marks are omitted and the input code is shown without a “prompt”. For example

Example 0.1:

1 + 2

Input

3

Output

may correspond to a screen output of

```
> 1+2
[1] 3
>
```

Depending on the configuration, the prompt ">" may be represented by a different Symbol.

Thanks

Thanks to the R core team for comments and hints. Special thanks to Friedrich Leisch (R core team) and Antony Unwin (Univ. Augsburg).

Literature and Additional References

s:04

R:Rnotes

[14] R Development Core Team (2000-2007): An Introduction to R.
See: <<http://www.r-project.org/manuals.html>>.

R:Rref
[17] R Development Core Team (2000-2007): R Reference Manual.
See: <<http://www.r-project.org/manuals.html>>.

The Omega Development Group (2000): Omega.
See: <<http://www.omegahat.org/>>.

bcw88news
[1] Becker, R.A.; Chambers, J.M.; Wilks, A.R. (1988): The New S Language. New York:
Chapman and Hall.

chh92sns
[3] Chambers, J.M.; Hastie, T.J. (eds) (1992): Statistical Models in S. New York: Chapman
and Hall.

clev93vd
[4] Cleveland, W.F. (1993): Visualizing Data. Summit: Hobart Press.

vr02mass
[25] Venables, W.N.; Ripley, B.D. (2002): Modern Applied Statistics with S. Heidelberg:Springer.
See: <<http://www.stats.ox.ac.uk/pub/MASS4/>>.

vr00pis
[24] Venables, W.N.; Ripley, B.D. (2000): Programming in S. Heidelberg:Springer.
See: <<http://www.stats.ox.ac.uk/pub/MASS3/Sprog>>.

```
$Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/chapters/S00eintro.tex,v $  
$Revision: 1.1 $  
$Date: 2008/01/26 20:27:31 $  
$Name: $  
$Author: j40 $
```

CHAPTER 1

Basic Data Analysis

ch:01

1.1 R Programming Conventions

s:1.0

Like any programming language, R has certain conventions. Here are the first basic rules.

To Do:
discuss
missing
values -
IntroR 2.5

| R <i>conventions</i> | |
|----------------------|--|
| numbers | A point is used as a decimal separator. Numbers can be written in exponential form; the exponential part is introduced by <i>E</i> . Numbers can be complex numbers; the imaginary part is marked by <i>i</i> . <i>Example:</i> 1 2.3 3.4E5 6i+7.8 Numbers can take the values <i>Inf</i> , <i>-Inf</i> , <i>NaN</i> for “not a number” and <i>NA</i> for “not available” = missing. <i>Example:</i> 1/0 results in <i>Inf</i> ; 0/0 results in <i>NaN</i> . |
| Strings | Strings are delimited by " or '. <i>Example:</i> "ABC" 'def' "gh'ij" |

To allow for non trivial examples, we anticipate a detail: in R, *a:b* is a sequence of numbers from *a* to at most *b* in steps of 1 resp. -1.

| R <i>conventions</i> | |
|----------------------|----------|
| | (cont.)→ |

| | |
|---------------------------------|--|
| R conventions (cont.) | |
| Objects | The basic elements in R are objects. Objects can be assigned to classes. <i>Example:</i> The basic objects in R are vectors. |
| Names | R objects can have names, by which they can be accessed. Names begin with a letter or a dot, followed by a sequence of letters, digits, or the special characters _ or . <i>Examples:</i> <code>x</code> <code>y_1</code> Lower and upper case are treated as different. <i>Examples:</i> <code>y87</code> <code>y87</code> |
| Assignments | Assignments have the form <i>Syntax:</i> <code>name <- value</code> or alternatively <code>name = value</code> . <i>Example:</i> <code>a <- 10</code> <code>x <- 1:10</code> |
| Queries | If only the name of an object is entered, the value of the object is returned. <i>Example:</i> <code>x</code> |
| Indices | Vector components are accessed by index. The lowest index is 1. <i>Example:</i> <code>x[3]</code> The indices can be specified directly, or using symbolic names or rules. <i>Examples:</i> <code>x[-3]</code> <code>x[x^2 < 10]</code> <code>a[1]</code> |

| | |
|----------------------------|--|
| help and inspection | |
|----------------------------|--|

(cont.)→

| | |
|---------------------------------------|---|
| <i>help and inspection</i> (cont.) | |
| help | <p>Documentation and additional information about an object can be requested using <code>help</code>.</p> <p><i>Syntax:</i> <code>help(name)</code></p> <p><i>Examples:</i> <code>help(exp)</code> <code>help(x)</code></p> <p>Alternative form <code>?name</code></p> <p><i>Examples:</i> <code>?exp</code> <code>?x</code></p> |
| Inspection | <p><code>help()</code> can only provide information that has been prepared in advance. <code>str()</code> can inspect the actual state of an object and display this information.</p> <p><i>Syntax:</i> <code>str(object, ...)</code></p> <p><i>Examples:</i> <code>str(x)</code></p> |

| | |
|----------------------|--|
| <i>R conventions</i> | |
| Functions | <p>Function calls in R have the form</p> <p><i>Syntax:</i> <code>name(parameter ...)</code></p> <p><i>Example:</i> <code>e_10 <- exp(10)</code></p> <p>This convention even hold when there are no parameters at all.</p> <p><i>Example:</i> To quit R, you call a “quit” function <code>q()</code>.</p> <p>Parameters are treated in a very flexible way. They can have default values, which are used if no explicit parameter value is given.</p> <p><i>Examples:</i> <code>log(x, base = exp(1))</code></p> <p>Functions can be polymorphic. For a polymorphic function, the actual function is determined by the class of the actual parameters.</p> <p><i>Examples:</i> <code>plot(x)</code> <code>plot(x, x^2)</code> <code>summary(x)</code></p> |

(cont.)→

| | |
|--|---|
| <pre>random_numbers runif Uniform@Uniform dunif@dunif (Uni- form) punif@punif (Uni- form) qunif@qunif (Uni- form) runif@runif (Uni- form) Topic dis- tri- bu- tion!Uniform@Uniform</pre> | <p>R conventions (cont.)</p> <p>Operators</p> <p>When applied to vectors, operators operate on each of the vector components.</p> <p><i>Example:</i> For vectors y, z, the product $y*z$ is the vector of component-wise products.</p> <p>Operators are special functions. They can be called in prefix form (function form).</p> <p><i>Example:</i> $+(x, y)$</p> <p>When applied to two operands with different lengths, the smaller operand is repeated cyclically.</p> <p><i>Example:</i> $(1:2)*(1:6)$</p> |
|--|---|

Our subject are statistical methods. In a first step, we apply the methods in simulations, that is we use synthetical data. The generation of these data are largely under our control. This gives us the opportunity to gain experiences with the methods and allows a critical evaluation. Only after that we will use the methods for data analysis.

1.2 Generation of Random Numbers and Patterns

s:1.1

1.2.1 Random Numbers

subs:1.1.1

Random variables with a uniform distribution can be generated by the function `runif()`. Using `help(runif)` or `?runif` we get information how to use this function:

help(runif)

Uniform

The Uniform Distribution

Description

These functions provide information about the uniform distribution on the interval from `min` to `max`. `dunif` gives the density, `punif` gives the distribution function `qunif` gives the quantile function and `runif` generates random deviates.

Usage

```
dunif(x, min=0, max=1, log = FALSE)
punif(q, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
qunif(p, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
runif(n, min=0, max=1)
```

```
.Random.seed@.Random.seed
rnorm@rnorm|textit
```

Arguments

| | |
|-------------------------|---|
| <code>x, q</code> | vector of quantiles. |
| <code>p</code> | vector of probabilities. |
| <code>n</code> | number of observations. If <code>length(n) > 1</code> , the length is taken to be the number required. |
| <code>min, max</code> | lower and upper limits of the distribution. |
| <code>log, log.p</code> | logical; if TRUE, probabilities p are given as log(p). |
| <code>lower.tail</code> | logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$. |

Details

If `min` or `max` are not specified they assume the default values of 0 and 1 respectively.
The uniform distribution has density

$$f(x) = \frac{1}{max - min}$$

for $min \leq x \leq max$.

For the case of $u := min == max$, the limit case of $X \equiv u$ is assumed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`.Random.seed` about random number generation, `rnorm`, etc for other distributions.

Examples

```
u <- runif(20)

## The following relations always hold :
punif(u) == u
dunif(u) == 1

var(runif(10000))#- ~ = 1/12 = .08333
```

```
serial
  plot|textbf
plot
```

The help information tells us: as a parameter for `runif()` we have to supply the number `n` of random variates to generate. As additional parameters for `runif()` we can specify the minimum and the maximum for the range of the random numbers. If we do not specify additional parameters, the default values `min = 0` and `max = 1` are taken. For example, `runif(100)` generates a vector with 100 uniform random numbers with range $(0, 1)$. Calling `runif(100, -10, 10)` generates a vector with 100 uniform random numbers in the range $(-10, 10)$.

The additional parameters can be supplied in the defined order, or specified by name. If the name of the parameter is given, the position can be chosen freely. So instead of `runif(100, -10, 10)` it is possible to use `runif(100, min = -10, max = 10)` or `runif(100, max = 10, min = -10)`. Using the name, it is also possible to set only chosen parameters. For example, if the minimum is not specified, the default value for the minimum is taken: using `runif(100, max = 10)` is equivalent with `runif(100, min = 0, max = 10)`. For better readability, we often write the names of parameters, even if not necessary.

Each execution of `runif()` generates 100 new uniform random numbers. We can store these numbers.

```
x <- runif(100)
```

generates a new vector of random numbers and stores it in the variable `x`.

```
x
```

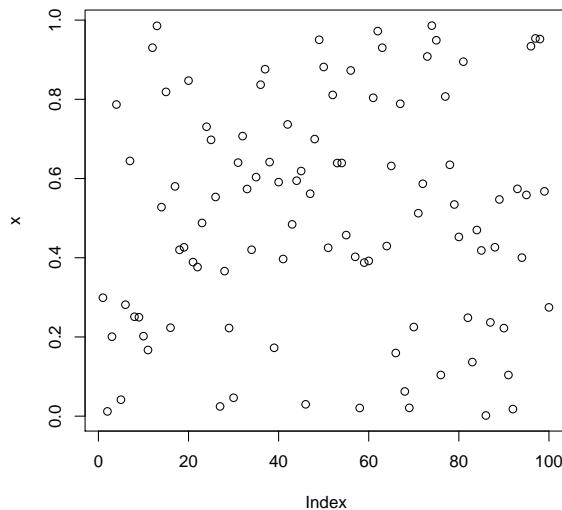
returns the values. By default, it is written to the output, and we can inspect the result. We get a graphical representation, the serial plot `serial plot` - a scatter plot of the entries `x` against its running index, by using

```
plot(x)
```

expl:ch01-Sframe02

Examplex 1.1:

```
x <- runif(100)
plot(x)
```

Input

a:ch01:01-1

| Exercise 1.1 | |
|--------------|--|
| | <p>Try experimenting with these plots and <code>runif()</code>. Do the plots show images of random numbers?</p> <p>To be more precise: do you accept these plots as images of 100 independent realisations of random numbers, distributed uniformly on $(0, 1)$?</p> <p>Repeat your experiments and try to note as precisely as possible the arguments you have for or against (uniform) randomness. What is your conclusion?</p> <p>Walk through your arguments and try to draft a test strategy to analyse a sequence of numbers for (uniform) randomness. Try to formulate your strategy as clear as possible.</p> <p style="text-align: right;">(cont.)→</p> |

```
pseudo
random
num-
bers|see random
numbers|textbf
random_numbers!pseudo
```

| Exercise 1.1 | (cont.) |
|--------------|--|
| | <p><i>Hint:</i> For comparison, you can keep several plots in a window. The code</p> $\text{par(mfrow} = c(2, 3))$ <p>parametrises the graphics system to show six plots simultaneously, arranged row wise as a 2×3 matrix (2 rows, 3 columns).</p> <p>The function <code>par</code> is the central function to control graphics parameters. For more information, see <code>help(par)</code>.</p> |

ToDo:
 consider
 simple
 exercises
 for random
 numbers –
 Antony

We lift the secret*: the numbers are not random, but completely deterministic. In the background, `runif()` builds a deterministic sequence z_i . Various algorithms for this are available. For more information, see `help(.Random.seed)`. In the most simple case, for linear congruence generators, subsequent values z_i, z_{i+1} are generated simply by a linear function. To keep the values in a controlled range, calculation is done modulo an upper bound, that is

$$z_{i+1} = a z_i + b \mod M.$$

The resulting values are re-scaled by

$$\frac{z_i}{M} \cdot (\max - \min) + \min.$$

and returned to us. The sequence so defined can be regular and soon lead to a periodic solution. With appropriate choice of the parameters however, like in the example given in the footnote, it can result in a long period (in the order of magnitude of M) and appear random. But the sequence of numbers is not at all a sequence of independent random numbers, and its distribution is not a uniform distribution on (\min, \max) .

Even knowing the secret, a lot of additional knowledge is needed to prove that the generated sequence does not follow the rules which apply to a sequence of independent identically distributed random numbers from a uniform distribution. Sequences which claim to work like random numbers are called **pseudo random numbers**, if it is important to mark the difference. We use these pseudo random numbers to generate convenient test data sets. Using these test data sets we can analyse how statistical methods perform under known conditions. In his context, we use pseudo random numbers as if we had true random numbers.

On the other side, we can take pseudo random numbers as a challenge: are we capable to detect that they are not independent random numbers? If we can detect the difference, we would try to replace the pseudo random number generator by a better one. But first the challenge goes to us. Are we capable at all to detect that the sequence generated by a linear congruence generator, say, is not random but deterministic? If we cannot: what are the intellectual consequences to draw from this?

* ... at least part of it. The random number generators used in R can be customised and usually more complex than the simple variant introduced here. For our discussion, the family of linear congruence generators suffices as an illustration. The usual reference here is the “minimal standard generator” with $x_{i+1} = (x_i \times 7^5) \mod 2^{31} - 1$.

1.2.2 Patterns

Besides the possibility to generate pseudo random numbers, R provides several possibilities to generate regular sequences. In many cases these can replace loops, which are common in other languages. Here is a first survey:

| R Sequences | |
|--------------------|--|
| : | generates a sequence from <code><begin></code> to at most <code><end></code> . <i>Syntax:</i> <code><begin>:<end></code> <i>Examples:</i> <code>1:10</code> <code>10.1:1.2</code> |
| <code>c()</code> | “combine”. combines arguments to a new vector. <i>Syntax:</i> <code>c(..., recursive = FALSE)</code> <i>Examples:</i> <code>c(1, 2, 3)</code> <code>c(x, y)</code> If the arguments are complex data types, the function will descend recursively if called with <code>recursive = TRUE</code> . |
| <code>seq()</code> | generates general sequences. <i>Syntax:</i> See <code>help(seq)</code> |
| <code>rep()</code> | repeats argument. <i>Syntax:</i> <code>rep(x, times, ...)</code> <i>Examples:</i> <code>rep(x, 3)</code> <code>rep(1:3, c(2, 3, 1))</code> |

a:ch01:01-2

Here “...” denotes a variable list of arguments. We will use this notation frequently.

| Exercise 1.2 | |
|--------------|---|
| | <p>Use</p> $\text{plot}(\sin(1:100))$ <p>to generate a plot of a discretised sine function. (If you do not recognise the sine function at first sight, use <code>plot(sin(1:100), type = "l")</code> to connect the points. Use your strategy from irexercise 1.1. Does your strategy detect that the sine function is not a random sequence?</p> |

Listing the numbers of a data set, as for example the output of a random number

generator, rarely helps to detect underlying structures. Simple unspecific graphical representations like the serial plot have some, but only limited use. Even with clear patterns the information provided by these plots is rarely meaningful. Purposeful representations are needed to investigate distribution properties of data.

1.3 Case Study: Distribution Diagnostics

s:1.2

We need specific strategies to detect the presence or the violation of structures. We use random numbers to illustrate how these strategies can look like. Here we concentrate on the distribution properties. Let us assume the sequence consists of independent random numbers with a common distribution. How do we check whether this distribution is a uniform distribution? We will ignore a rescaling to (\min, \max) - this might be necessary, but is a technical detail which does not affect the investigation substantially. So for now we consider the case $\min = 0; \max = 1$.

Realisations of random variables do not allow to read off the distributions directly. This is our critical problem. We need characterisations of the of the distribution which allow an empirical inspection. Of course we can view the observations as measures. For n observations X_1, \dots, x_n can we define the empirical distribution P_n as $P_n = \sum(1/n)\delta_{x_i}$, where δ_{X_i} is the point measure at X_i . Hence

$$P_n(A) = \#\{i : X_i \in A\}/n.$$

Unfortunately the empirical distribution P_n of a set of independent observations with common distribution P is in general very different from P . Some properties get lost without repair. Infinitesimal properties are among these. So for example P_n is always concentrated on finitely many points. So empirical distributions are always discrete, irrespective of the underlying distribution. To analyse distributions based on data, we need functionals which can be determined based on realisations of random variable and which can be compared to the corresponding functionals of theoretical distributions. One strategy is to restrict to a family of test sets that is treatable empirically.

S01:ex01 Example 1.1 *distribution function*

Instead of the distribution P we consider its distribution function $F = F_P$, where

$$F(x) = P(X \leq x).$$

For an empirical distribution P_n of n observations X_1, \dots, X_n , the corresponding empirical distribution function is

$$F_n(x) = \#\{i : X_i \leq x\}/n.$$

S01:ex02 Example 1.2 *Histogram*

We select disjoint test sets $A_j, j = 1, \dots, J$, covering the range of X . For example for the uniform distribution on $(0, 1)$ we can chose the intervals

$$A_j = \left(\frac{j-1}{J}, \frac{j}{J}\right]$$

as test sets.

Instead of the distribution P we consider the vector $(P(A_j))_{j=1,\dots,J}$. The empirical version is the vector $(P_n(A_j))_{j=1,\dots,J}$.

We will discuss this example in some detail. Some general lessons can be drawn from this example. We will take several passes, moving from a naive approach to an elaborate statistical approach.

We take the opportunity to point out that the histogram depends critically on the choice of the test sets. In particular, if discretisations in the data meet with an unfortunate choice of the test sets, the results may be misleading. As an alternative to the histogram, we can choose to smooth the data.

S01:kdens

Example 1.3 Smoothing We replace each data point by a (local) distribution, that is we blur the data points somewhat. We can do this using weight functions. These weight function are called **kernel** and denoted by K . If the integral over a kernel exists, conventionally the kernel is normalised so that $\int K(x)dx = 1$. Some commonly used kernels are listed in table I.9 and shown in Fig. I.1. For kernels with compact support, the support is chosen to be the interval $[-1, 1]$ (The R convention is to standardise the kernel, so that the standard deviation is 1.)

| Kernel | $K(x)$ |
|--------------------------|------------------------------|
| uniform | $1/2$ |
| triangular | $1 - x $ |
| Epanechnikov (quadratic) | $3/4(1 - x^2)$ |
| biweight | $15/16(1 - x^2)^2$ |
| triweight | $35/32(1 - x^2)^3$ |
| Gauß | $(2\pi)^{-1/2} \exp(-x^2/2)$ |

Table 1.9 Some commonly used kernels.

ta:ch01kernel

By shift and scaling each kernel gives rise to family

$$\frac{1}{h} K\left(\frac{x - x_0}{h}\right).$$

For kernels, the scale factor h is called the kernel **bandwidth**. The kernel scaled by h , is denoted by K_h :

$$K_h(x) = \frac{1}{h} K\left(\frac{x}{h}\right).$$

The function

$$x \mapsto \frac{1}{n} \sum_i K_h(x - X_i)$$

results in a smoothed display which can replace (or enhance) the histogram.

For more information, look for the keywords **smoothing** or **kernel density estimation**.

kernel|textbf
bandwidth|textbf

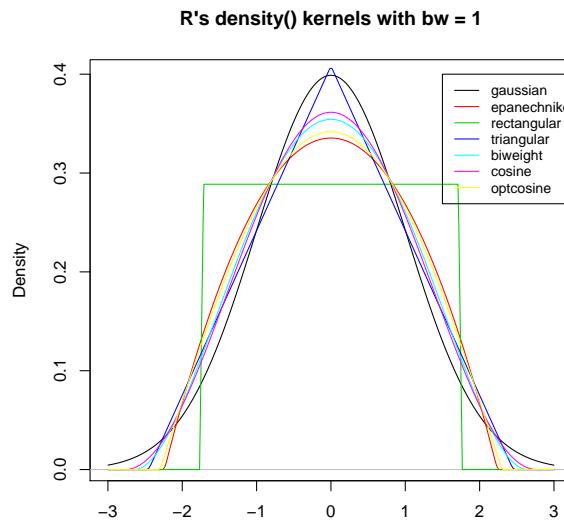
Figure 1.1 *Kernels in R*

fig:Rkernels

1.3.1 First Pass for Example S01:ex01 1.1: Distribution Functions

subs:s01-vf1

To test whether a random sequence has a distribution with distribution function F , we compare F with the empirical distribution function F_n . As a special case, we start with a uniform distribution on $(0, 1)$. In this special case we have $F(x) = F_{unif}(x) = x$ for $0 \leq x \leq 1$. In general, we have to compute the functions F_n and F . A first considerations tells us: F_n is a piecewise constant function with jumps at the observations. So we get a complete image of F_n , if we evaluate F_n at the observations $X_i, i = 1..n$. If $X_{(i)}$ denotes the i th order statistics, we have - up to ties - $F_n(X_{(i)}) = i/n$. We have to compare $F_n(X_{(i)})$ with the theoretical value $F(X_{(i)}) = X_{(i)}$. An R implementation, written with temporary variables, is

```
n <- 100
x <- runif(n)
xsort <- sort(x)
i <- (1:n)
y <- i/n
plot(xsort, y)
```

A line with the theoretical values can be added with

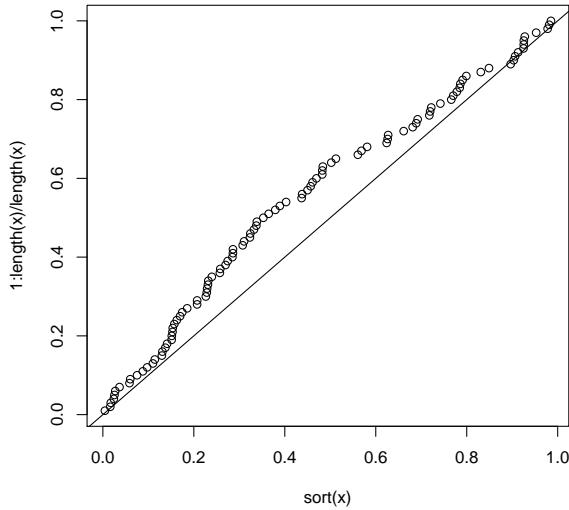
```
abline(0, 1)
```

A more compact implementation using the function `length()` would be:

expl:ch01-Sframe03

Example 1.2:*Input*

```
x <- runif(100)
plot(sort(x), 1:length(x)/length(x))
abline(0, 1)
```



```
plot@plot|textit
plot@plot|textit
```

| <i>R functions</i> | |
|--------------------|--|
| <i>sort()</i> | sorts a vector <i>Example:</i> <code>sort(runif(100))</code> |
| <i>length()</i> | length of a vector <i>Example:</i> <code>length(x)</code> |
| <i>abline()</i> | adds a line to a plot <i>Example:</i> <code>abline(a = 0, b = 2)</code> |

By default, the `plot()` function adds some annotation. To get a graphic speaking for itself we modify the default annotation to show more specific information. We can replace the default parameters of `plot()`. The parameter `main` controls the main title (default: empty). We can add a title, for example

```
plot(sort(x), (1:length(x))/length(x),
     main = "Empirical distribution function\n(X uniform)").
```

plotmath Labels for the axis can be controlled using the parameters `xlab` and `ylab`. More information about this and additional parameters can be requested using `help(plot)`.
expression@expression|textit
plot@plot|textit But the help information only provides a forward reference in these cases. The actual information is given by `help(title)`.

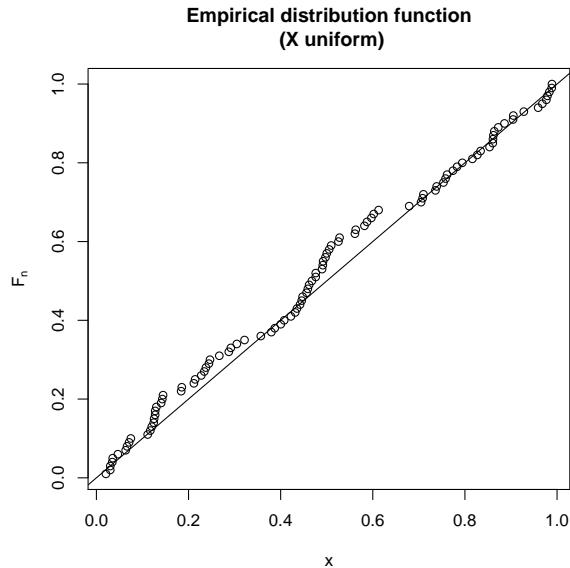
The vertical axis still gives a challenge. Using `ylab = "Fn(x)"` would give a label $F_n(x)$. The usual notation however puts the samplesize as a subscript, as in $F_n(x)$. A hidden property of the levelling functions helps: if a character string is passed as parameter, it is displayed “as is”. If an R expression is passed as parameter, R tries to display it following the usual mathematical conventions. Details are documented in `help(plotmath)` and examples can be seen using `demo(plotmath)`. To convert a character string to an (unevaluated) R expression, use function `expression()`.

expl:ch01-1.3

Example 1.3:

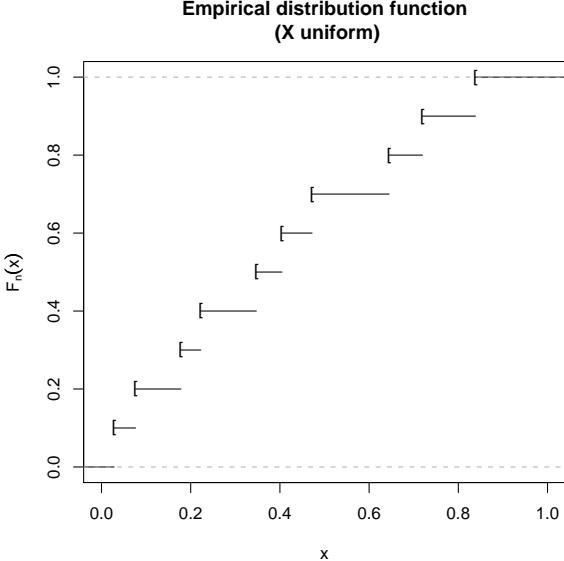
Input

```
x <- runif(100)
plot(sort(x), (1:length(x))/length(x), xlab = "x",
      ylab = expression(F[n]), main = "Empirical distribution function\n(X uniform)")
abline(0, 1)
```



This example serves only for an introduction here. It is not necessary to program distribution functions and their plots from scratch. R already provides a class `ecdf` for empirical distribution functions. If the function `plot()` is applied to an object of class `ecdf`, the “generic” function `plot()` dispatches internally to the special function `plot.ecdf`, and this draws the distribution function in its specific way. We can reduce our example by just calling `plot(ecdf(runif(100)))`.

a:ch01:01-ecdf

| Exercise 1.3 | Extend the function call <code>plot(ecdf(runif(10)))</code> using additional parameters so that the result has the following form:  |
|--------------|---|
|--------------|---|

a:ch01:01-3

| Exercise 1.4 | |
|--------------|--|
| | <p>Using <code>norm()</code> you can generate random variables with a Gaussian distribution. Can you tell a Gaussian distribution from a uniform distribution using a serial plot?</p> <p>Use the empirical distribution function.</p> <p>Can you tell a Gaussian distribution from a uniform distribution using a plot of the empirical distribution function? Können Sie damit gaußverteilte und uniform verteilten unterscheiden? Can you tell the sinus series from a uniform distribution using a plot of the empirical distribution function? from a uniform distribution?</p> <p>What is in each case the sample size needed to recognise the differences reliably?</p> |

1.3.2 First Pass for Example S01:ex02

subs:S01hist1

We select test sets A_j , $j = 1, \dots, J$ to cover the range taken by the values of X . Our strategy: to test whether the data may correspond to a random sequence from a

`rug@rug|textit{distribution} P , we compare the vector $(P(A_j))_{j=1,\dots,J}$ with $(P_n(A_j))_{j=1,\dots,J}$. For the special case of the uniform distribution on $(0, 1)$ we can take the intervals`

$$A_j = \left(\frac{j-1}{J}, \frac{j}{J} \right]$$

as test sets. Then the vector of theoretical probabilities

$$(P(A_j))_{j=1,\dots,J} = (1/J, \dots, 1/J)$$

is to be compared to observed relative frequencies $\frac{\#\{X_i \in A_j\}}{n} \quad j = 1, \dots, J$. A preliminary implementation: we use a ready made function to draw histograms. As a side effect, it returns the relative frequencies. The function `rug()` adds the original data to the plot.

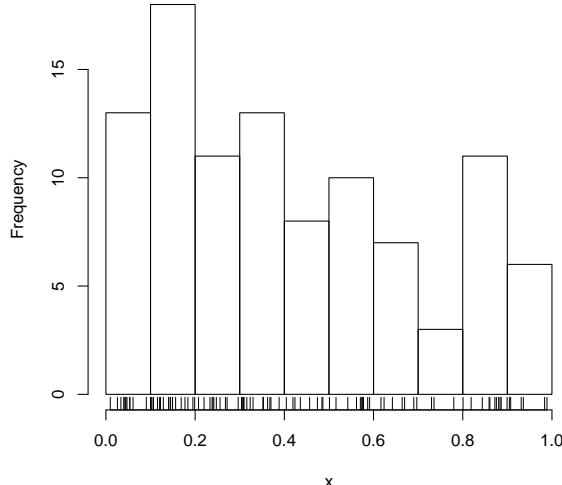
expl:ch01-fights

Example 1.4:

Input

```
x <- runif(100)
hist(x)
rug(x)
```

Histogram of x

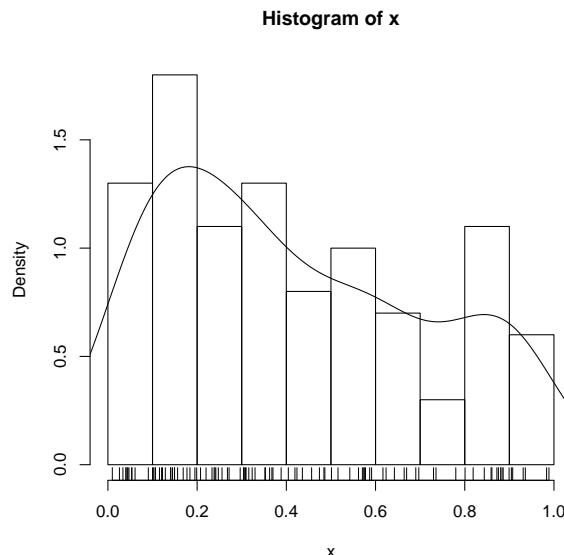


For comparison, we can overlay a kernel density estimation. In contrast to `hist()`, the function `density()` does not draw its result, but prints it. So we have to ask for the graphical output explicitly. To make scale comparable, we use the parameter `probability = TRUE` to get a histogram using a probability scale.

01-fig41histkernel

Example 1.5:

```
hist(x, probability = TRUE)
rug(x)
lines(density(x))
```

Input

Histograms and kernel density estimators each have their particularities. Histograms suffer from their discretisation, which may interfere with possible discretisation of the data and give an aliasing effect. Kernel density estimators do not have these discretisation effect, but they blur the data which may lead to inappropriate boundary effects.

Let us return to the histogram: If we use an assignment

```
whist <- hist(x),
```

the internal information of the histogram is stored as `whist` and be accessed using

```
whist
```

This gives a result like in the following example:

```
expl:ch01-fights
```

data_structures

Example 1.6:

| | |
|---|---------------|
| <pre> x <- runif(100) whist <- hist(x) whist </pre> | <i>Input</i> |
| <pre> \$breaks [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 \$counts [1] 8 16 12 11 11 11 10 8 8 5 \$intensities [1] 0.7999998 1.6000000 1.2000000 1.1000000 1.1000000 1.1000000 [7] 1.0000000 0.8000000 0.8000000 0.5000000 \$density [1] 0.7999998 1.6000000 1.2000000 1.1000000 1.1000000 1.1000000 [7] 1.0000000 0.8000000 0.8000000 0.5000000 \$mids [1] 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95 \$xname [1] "x" \$equidist [1] TRUE attr(,"class") [1] "histogram" </pre> | <i>Output</i> |

Counts gives the cell counts. So these are exactly the numbers we are looking for. The information stored in *whist* consists essentially in five components. Each of these components is a vector. The components have names and can be accessed using these names. So for example

whist\$counts

gives the vector of the cell counts.

| | |
|------------------------|--|
| <i>Data Structures</i> | |
|------------------------|--|

(cont.)→

| <i>Data Structures</i> (cont.) | | hist@hist textit hist@hist textit table@table textit |
|-----------------------------------|--|--|
| vectors | Components of a vector are accessed by their index. All elements of the vector have the same type. <i>Examples:</i> <code>x</code> <code>x[10]</code> | |
| lists | Lists are complex data structures. The components of a list have names. They can be accessed using these names. Components of a list may be of differing types. <i>Examples:</i> <code>whist</code> <code>whist\$counts</code> | |

Additional complex data structures are discussed in appendix (A.18).

The choice of histogram cells is data dependent. There are various conventions and recommendations how to choose cells. The details are controlled by parameters for `hist()`. To select test sets of our choice, we have to inspect the calling structure for `hist()`.

| Exercise 1.5 | |
|--------------|---|
| | <p>Use <code>runif(100)</code> to draw random numbers and generate histograms with 5, 10, 20, 50 cells of equal size. Use repeated samples. Do the histogram plots correspond to what you expect from independent uniform random variates? Try to note your observations in detail.</p> <p>Repeat the experiment with two cells $(0, 0.5], (0.5, 1)$.</p> <pre>hist(runif(100), breaks = c(0, 0.5, 1))</pre> <p>Repeat the experiment with random numbers generated by <code>norm(100)</code> and compare the results from <code>runif(100)</code> and <code>norm(100)</code>.</p> |

Bar Charts

Note: if the data are not quantitative, but categorical (marked by category labels like “excellent, good, satisfactory, ...”, or marked by category number, like “1, 2, 3, ...”), a bar chart may be more appropriate than a histogram. Simple bar charts are already provided by

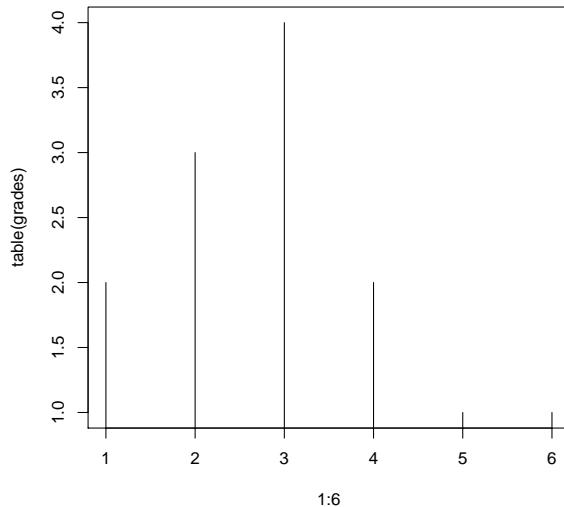
`plot()`, using the parameter `type = h`. To use `plot()`, the raw data must be broken down to frequencies for each level of the variable. This can be done using `table()`.

`expl:ch01-bar`

Example 1.7:

Input

```
grades <- c(2, 1, 3, 4, 2, 2, 3, 5, 1, 3, 4, 3, 6)
plot(1:6, table(grades), type = "h")
```



a:ch01:e3bar

| Exercise 1.6 | Modify the function call so that the resulting plot is as follows: | | | | | | | | | | | | | | |
|--------------|--|-------|-------|---|---|---|---|---|---|---|---|---|---|---|---|
| | <p style="text-align: center;">Grading</p> <table border="1"> <caption>Data for the 'Grading' bar chart</caption> <thead> <tr> <th>Grade</th> <th>Count</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>2</td> </tr> <tr> <td>2</td> <td>3</td> </tr> <tr> <td>3</td> <td>4</td> </tr> <tr> <td>4</td> <td>2</td> </tr> <tr> <td>5</td> <td>1</td> </tr> <tr> <td>6</td> <td>1</td> </tr> </tbody> </table> | Grade | Count | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 2 | 5 | 1 | 6 | 1 |
| Grade | Count | | | | | | | | | | | | | | |
| 1 | 2 | | | | | | | | | | | | | | |
| 2 | 3 | | | | | | | | | | | | | | |
| 3 | 4 | | | | | | | | | | | | | | |
| 4 | 2 | | | | | | | | | | | | | | |
| 5 | 1 | | | | | | | | | | | | | | |
| 6 | 1 | | | | | | | | | | | | | | |

ToDo: remove line caps - bar line is too long

1.3.3 Statistics of Distribution Functions; Kolmogorov-Smirnov-Tests

We now move from a naive approach to a statistical point of view. For independent identically distributed variables (X_1, \dots, X_n) with distribution function F . In a naive approach, we have assumed that $i/n = F_n(X_{(i)}) \approx F(X_{(i)})$. We wanted to use this relation to test our distribution assumption. In particular, for a uniform distribution on $(0, 1)$ this relation reads $i/n \approx F(X_{(i)}) = F(X_{(i)})$.

Seen from a statistical point of view, each $X_{(i)}$ is a random variable. Hence also $F(X_{(i)})$ is a random variable, with values in $[0, 1]$, and we can analyse the distribution if this random variable.

Theorem 1.4 If (X_1, \dots, X_n) are independent random variables with common continuous distribution function F , the $F(X_{(i)})$ has a beta distribution $\beta(i, n - i + 1)$.

Proof. → probability theory. Hint: use

$$X_{(i)} \leq x_\alpha \Leftrightarrow (\#j : X_j \leq x_\alpha) \geq i.$$

For continuous distributions, $(\#j : X_j \leq x_\alpha)$ has a binomial distribution with parameters (n, α) . □

Corollary 1.5

$$E(F(X_{(i)})) = i/(n + 1).$$

ToDo:
this is
first theorem. make
numeration more
logical

a:ch01:beta1

| Exercise 1.7 | |
|--------------|---|
| | Using <code>help(rbeta)</code> you get information about the functions available to work with beta distributions. Generate plot for the densities of the beta distribution for $n = 10, 50, 100$ and $i = n/4, n/2, 3n/4$. Use the function <code>curve()</code> to generate the plots. For more information, see <code>help(curve)</code> . |

So in the average, for independent random variables with uniform distribution on $(0, 1)$ we cannot expect that $X_{(i)} \approx .i/n$, but in the average we get $i/(n + 1)$. Hence the reference line should be drawn with `abline(a = 0, b = n/n+1)`.

| Exercise 1.8 | |
|--------------|--|
| | Draw the distribution function with the correct reference line. |
| * | We use the graphical display for a single sample, not for a run of samples. Is the expected value of $X_{(i)}$ the adequate reference? Are there alternatives which can serve as references? If you see alternatives, give an implementation. |

Monte Carlo Confidence Bands

Simulation can also help as to get an impression of the typical fluctuation. We use random numbers to generate a small number of samples, and compare our sample in question with these simulations. For comparison, we generate envelopes of these simulations and check whether our sample lies within the area delimited by the envelopes. If x is the sample vector to be analysed, with length n , we use this programming idea:

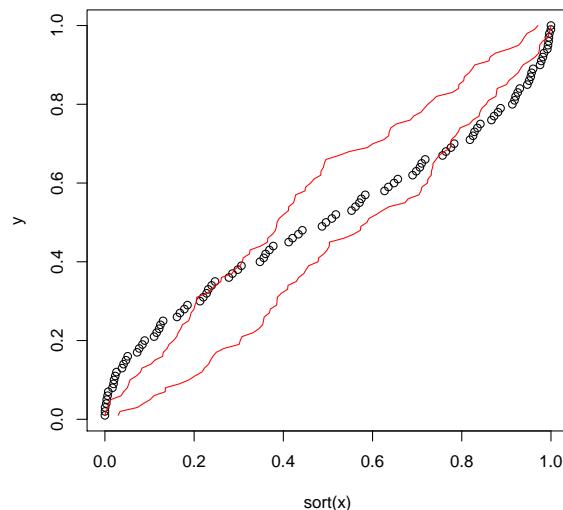
```
expl:ch01-MC06
```

Example 1.8:*Input*

```

x <- (sin(1:100) + 1)/2
y <- (1:length(x))/length(x)
plot(sort(x), y)
nrsamples <- 19
samples <- matrix(data = runif(length(x) * nrsamples),
    nrow = length(x), ncol = nrsamples)
samples <- apply(samples, 2, sort)
envelope <- t(apply(samples, 1, range))
lines(envelope[, 1], y, col = "red")
lines(envelope[, 2], y, col = "red")

```



ToDo: apply and related

This example is taken from [\[25\]](#), which is a rich source of ideas for R programming.

A typical R is illustrated here. R is an interpreted vector oriented language. Single step interpretation is time extensive. Hence operations with few complex steps can be more efficient than operations using several elementary steps.

- Operations on a vector level are more effective than chains of single elementary operations.
- Iterations and loops are avoided and replaced by structured vector operations.

a:ch01:e5

| | |
|---------------------|--|
| Exercise 1.9 | |
| | Make use of <code>help()</code> -function and comment example I.8 step by step. Take a special note of the new functions which are introduced here. |

ToDo:
ch01: add examples
f. outer

| R Iterators | |
|----------------------|---|
| <code>apply()</code> | applies a function to the rows or columns of a matrix. <i>Examples:</i> <code>samples <- apply(samples, 2, sort)</code> sorts by column. |
| <code>outer()</code> | generates a matrix of all pairwise combinations of two vectors, and applies a function to all pairs. |

For the simulation bands, the hypothesis is that the given sample and the simulations come from the same model. If the curve representing our sample exceeds the limits from our simulation, this indicates that this hypothesis is violated. The approach which is sketched here is called a **Monte Carlo Test**. The idea behind this is very general and can be applied in many situations.

a:ch01:e6

| Exercise 1.10 | |
|----------------------|---|
| * | Why 19? <i>Hint:</i> try to take an abstract simplified view of the problem first: let T be a measurable function and $X_0, X_1, \dots, X_{nrsamples}$ independent samples with a common distribution function. What is $P(T(X_0) > T(X_i))$ for all $i > 0$. In a second step, give an abstract formulation for the example above. Then specialise for <code>nrsamples = 19</code> . |

a:ch01:e7

| Exercise 1.11 | |
|----------------------|---|
| * | Estimate the coverage probability of the Monte Carlo band by first generating a band as above (how can you draw the band without first making a plot for a special sample)? Next, generate sim simulation samples of uniform random numbers of sample size 100. Count how many simulations give a sample within the band. You have to make your choice of the number sim of simulations (100? 1000? 999?) for this step. Use this information to estimate the coverage probability. (cont.)→ |

bquote@bquote|

| | |
|----------------------|--|
| Exercise 1.11 | (cont.) |
| | Hint: <code>any()</code> can be used to evaluate a comparison for a full vector. |

ToDo:

work out
as example

As usual, we want to rework the output so that the resulting `plot` contains enough information. For the legend, we can proceed as in section 1.3.1. The simulation count `nrsamples` needs some thought. If only a fixed count (e.g. 19) is considered, we can use the constant in the legend as usual. If the program fragment should be more versatile, we would need the specific simulation count for each instance. Doing this by hand is error prone, and this possible source of errors can be avoided. The function `bquote()` allows to evaluate a variable, or to calculate it for a specific context. To allow evaluating the current number of simulations, we rearrange the statements so that the number of observations is fixed before `plot()` is called.

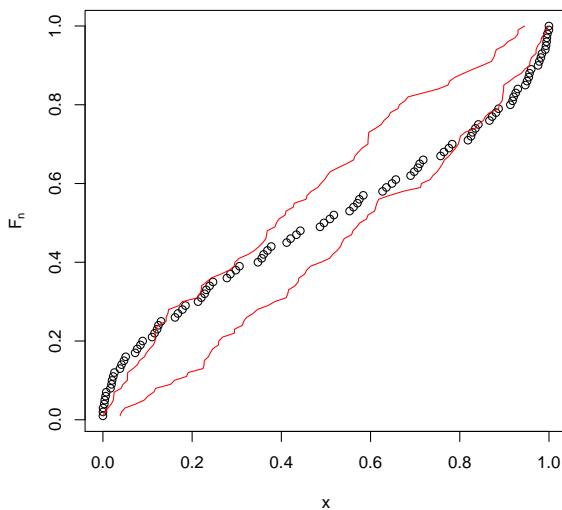
expl:ch01-MC06a

Example 1.9:*Input*

```

x <- (sin(1:100) + 1)/2
y <- (1:length(x))/length(x)
nrsamples <- 19
plot(sort(x), y, main = paste("Monte Carlo Band: ",
  bquote(..(nrsamples)), " Monte Carlo Samples"),
  xlab = "x", ylab = expression(F[n]))
samples <- matrix(data = runif(length(x) * nrsamples),
  nrow = length(x), ncol = nrsamples)
samples <- apply(samples, 2, sort)
envelope <- t(apply(samples, 1, range))
lines(envelope[, 1], y, col = "red")
lines(envelope[, 2], y, col = "red")

```

Monte Carlo Band: 19 Monte Carlo Samples

For each simulation new Monte Carlo samples are drawn. So for each invocation we get different Monte Carlo bands, and the bands here are different from those in the previous example.

For practical work it may be necessary to reduce distribution diagnostic to a simple decision problem. This may allow to produce tables or control charts to decide whether a distribution fits to some hypothesis, or to characterise the deviation from a given model. If we want to use tables or simple numbers, additional restrictions are needed: we need to reduce the information contained in the functions (F_n, F) to simple numbers. One digest is for example the statistic

$$\sup_x |F_n - F|(x).$$

To Do:

add label for examples

If we want to use this statistic as a criterion, we once again are faced with the task to analyse its distribution.

Theorem 1.6 (Kolmogorov, Smirnov) *For a continuous distribution function F , the distribution of*

$$\sup_x |F_n - F|(x)$$

is independent of F (in general, it will depend on n).

Proof. → probability theory. Z.B. [Gänßler & Stute, Lemma 3.3.8]. □

Theorem 1.7 (Kolmogorov) *For a continuous distribution function F and $n \rightarrow \infty$ the statistic*

$$\sqrt{n} \sup |F_n - F|$$

asymptotically has the distribution function

$$F_{\text{Kolmogorov-Smirnov}}(y) = \sum_{m \in \mathbb{Z}} (-1)^m e^{-2m^2 y^2} \quad \text{for } y > 0.$$

Proof. → probability theory. Z.B. [Gänßler & Stute, Formel (3.3.11)]. □

For practical work this means that for a continuos distributions function we can use the following decision strategy: we decide that observations (X_1, \dots, X_n) do not follow the hypothesis of independent identically distributed observations with distribution function F , if $\sup |F_n - F|$ is too large:

$$\sup |F_n - F| > F_{\text{crit}}/\sqrt{n},$$

where F_{crit} is taken form the distribution function of the Kolmogorov-Smirnov statistic for sample size n . This distribution function does not depend on F . In particular, if we chose the upper α quantile $F_{\text{crit}} = F_{\text{Kolmogorov-Smirnov}, 1-\alpha}$, we know that if the hypothesis is valid, this critical value is reached or exceeded at most with probability α . So we can control the error probability of a false rejection of the hypothesis.

Asymptotically, for large n , we can use the Kolmogorov-Smirnov approximation instead of the distribution function. If the model distribution F is not continuous is, additional considerations are necessary.

We want to concentrate on programming issues an do not dive into the details of the Kolmogorov-Smirnov tests. With elementary tools, we can calculate the test statistics $\sup_x |F_n - F|(x)$ for the uniform distribution. By monotony,

$$\sup_x |F_n - F|(x) = \max_{X(i)} |F_n - F|(X(i))$$

and for the uniform distribution

$$\max_{X(i)} |F_n - F|(X(i)) = \max_i |i/n - X(i)|.$$

In R notation, the expression

```
max( abs((1: length(x)) / length(x)) - sort(x)) )
```

is the statistics we are looking for, based on a data vector `x`.

`Test!Median-`

Like many commonly used statistics, this statistic is already implemented, as are functions for the corresponding distribution functions.[†]

`a:ch01:e7ks`

| Exercise 1.12 | |
|---------------|--|
| | <p>Using</p> <pre>help(ks.test)</pre> <p>you get information how to invoke the function <code>ks.test</code>.</p> <p>Which results do you expect if you test the following vectors for a uniform distribution:</p> <pre>1:100 runif(100) sin(1:100) rnorm(100)?</pre> <p>Perform these tests (scale the values so that they fall into the interval $[0, 1]$, or use a uniform distribution on an interval which is adapted to the data.) and discuss the results.</p> |

1.3.4 Statistics of Histograms and Related Plots; χ^2 -Tests

`subs:S01hist2`

Like with the distribution functions, we now move towards a statistical analysis. For simplicity we assume that we have chosen $A_j, j = 1, \dots, J$ covering the range of X . The vector of observations (x_1, \dots, x_n) translates into bin counts n_j

$$n_j = (\#i : X_i \in A_j).$$

If $(X_i)_{i=1,\dots,n}$ are independent with identical distribution P , $(n_j)_{j=1,\dots,J}$ is a random vector following a multinomial distribution with parameters $n, (p_j)_{j=1,\dots,J}$ where $p_j = P(A_j)$. For the special case $J = 2$ we have the binomial distribution. Since we are free in our choice of the sets A_j , we can cover a spectrum of special cases, like

Median test for symmetry:

$$A_1 = \{x < x_{0.5}\} \quad A_2 = \{x \geq x_{0.5}\}$$

Midrange test for concentration:

$$A_1 = \{x_{0.25} \leq x < x_{0.75}\} \quad A_2 = \{x < x_{0.25} \text{ or } x \geq x_{0.75}\}.$$

[†] Different implementations can use other calling structures. The Kolmogorov-Smirnov test is available as `ks.test()`.

Before R version 2.x however this was not included in the base of R, but was contained in special libraries which had to be loaded explicitly. In R1.x for example the library of classical tests needed to be loaded by `library(ctest)`

Test!\$\\chi^2\$ For the general case however we must compare the empirical vector of bin counts n_j with the multinomial distribution, which is not pleasant to calculate. We resort to using approximations. The following approximation goes back to Pearson:

Lemma 1.8 (Pearson): For $(p_j)_{j=1,\dots,J}$, $p_j > 0$ in the limit $n \rightarrow \infty$ the following approximation holds:

$$\begin{aligned} P_{mult}(n_1, \dots, n_J; n, p_1, \dots, p_J) \approx \\ (2\pi n)^{-1/2} \left(\prod_{j=1,\dots,J} p_j \right)^{-1/2} \cdot \\ \exp \left(-1/2 \sum_{j=1,\dots,J} \frac{(n_j - np_j)^2}{np_j} \right. \\ \left. - 1/2 \sum_{j=1,\dots,J} \frac{n_j - np_j}{np_j} \right. \\ \left. + 1/6 \sum_{j=1,\dots,J} \frac{(n_j - np_j)^3}{(np_j)^2} + \dots \right). \end{aligned}$$

ToDo:

check
pearson

Proof. → probability theory. Z.B. [10] p. 285. \square

The first term is controlled by $\widehat{\chi^2} := \sum_{j=1,\dots,J} (n_j - np_j)^2 / np_j$. This term is called the χ^2 statistics. At least asymptotically for $n \rightarrow \infty$ large values of $\widehat{\chi^2}$ give small probabilities. This motivates to use the χ^2 statistics approximatively as a measure for goodness of fit. The name comes from the distribution asymptotics:

Theorem 1.9 (Pearson): for $(p_j)_{j=1,\dots,J}, p_j > 0$ in the limit, as $n \rightarrow \infty$, the statistics

$$\widehat{\chi^2} := \sum_{j=1,\dots,J} \frac{(n_j - np_j)^2}{np_j}$$

has a χ^2 distribution with $J - 1$ degrees of freedom.

As above, to get a formal decision rule can fix a critical value χ^2_{crit} and reject the hypothesis that the observations (X_1, \dots, X_n) come from independent identically distributed uniform random variables if the χ^2 statistics exceeds this value. If we chose the upper α quantile of the χ^2 distribution as a critical value, we know that if the hypothesis holds the value χ^2_{crit} or a higher value is reached at most with probability α . So again, at least asymptotically, we can control the error probability of a false rejection of the hypothesis.

The χ^2 tests are a basic part of R and provided as function `chisq.test()`. They are designed to be used for more general “contingency tables”. We just need a special case. In our case, the table is a (one dimensional) vector of counts for the chosen histogram cells. (See also: the R implementation for more general variants in `library(loglin)`.)

a:ch01:e8chisq

| Exercise 1.13 | |
|---------------|--|
| | <p>Use <code>help(chisq.test)</code> to see the calling structure for χ^2 tests. Apply it to test the hypothesis ($p_j = 1/J$), $J = 5$ on the following vectors of bin counts:</p> $(3 \ 3 \ 3 \ 3 \ 3) \quad (1 \ 2 \ 5 \ 3 \ 3) \quad (0 \ 0 \ 9 \ 0 \ 6).$ |

a:ch01:e8achisq

| Exercise 1.14 | |
|---------------|--|
| | <p>Which results do you expect if you use a χ^2 test to check the following vectors for a uniform distribution:</p> $\begin{aligned} & 1:100 \\ & runif(100) \\ & sin(1:100) \\ & rnorm(100)? \end{aligned}$ <p>Perform these tests and discuss the results.</p> <p><i>Hint:</i> The function <code>chisq.test()</code> expects a frequency table as input. The function <code>table()</code> can be used to generate a frequency table directly (see <code>help(chisq.test)</code>). But you can also use the function <code>hist()</code> which gives <code>counts</code> as one components as a result.</p> |

At first, the approximation for the χ^2 statistics is only valid, if fixed cells are chosen, independent of the information provided by the sample. Practical histogram algorithms however derive the number of cells and cell bounds based on the sample. Often this implies an estimation of parameters of the distribution. Under some conditions however the χ^2 asymptotic still holds, as for example illustrated by the following theorem from [19] Section 6b.2:

Theorem 1.10 (i) Let the cell probabilities be the specified functions $\pi_1(\boldsymbol{\theta}), \dots, \pi_k(\boldsymbol{\theta})$ involving q unknown parameters $(\theta_1, \dots, \theta_q) = \boldsymbol{\theta}'$. Further let

- (a) $\hat{\boldsymbol{\theta}}$ be an efficient estimator of $\boldsymbol{\theta}$ in the sense of (5c.2.6),
- (b) each $\pi_i(\boldsymbol{\theta})$ admit continuous partial derivatives of the first order (only) with respect to θ_j , $j = 1, \dots, q$ or each $\pi_i(\boldsymbol{\theta})$ be a totally differentiable function of $\theta_1, \dots, \theta_q$, and
- (c) the matrix $M = (\pi_r^{-1/2} \partial \pi_r / \partial \theta_s)$ of order $(k \times q)$ computed at the true values of $\boldsymbol{\theta}$ is of rank q . Then the asymptotic distribution of

$$\chi^2 = \sum \frac{(n_i - n\hat{\pi}_i)^2}{n\hat{\pi}_i} = \sum \frac{(0 - E)^2}{E} \quad (1.1) \quad \boxed{\text{eq:?}}$$

is $\chi^2(k - 1 - q)$, where $\hat{\pi}_i = \pi_i(\hat{\boldsymbol{\theta}})$.

~~samples!repeated~~ ~~textbf{P}^{\text{rao73ls}}_{\text{off}}~~ See [19] Section 6b.2. \square

~~Power~~ ~~textbf{P}^{\text{rao73ls}}_{\text{off}}~~

| Exercise 1.15 | |
|----------------------|--|
| * | <p>Sketch comparable test environments for fixed and adaptive choice of histogram cells.</p> <p>For fixed and for adaptive choice of histogram cells draw $s = 1000$ samples of size 50 from <code>runif()</code>. Calculate in both settings the formal the χ^2 statistics and plot its distribution functions.</p> <p>Compare the distribution functions.</p> |

ToDo:

histogram
as density
estimator;
choice of
cells

ToDo:
reference
to kernel
density
estimators

Repeated Samples

So far, we have concentrated on inspecting the distribution of random variables. We can continue this inspection. If (X_1, \dots, X_n) are independent random numbers with identical uniform distribution, for fixed cells the χ^2 statistics is approximatively distributed as χ^2 , and $\kappa := \sqrt{n} \sup |F_n - F|$ has the Kolmogorov-Smirnov distribution.

We can take repeated samples $(X_{1,j}, \dots, X_{n,j})_{j=1..m}$ and from these we can calculate statistics $\widehat{\chi^2}_j$ and κ_j . Given independent, identically distributed random variables to start with, the statistics have a χ^2 resp. a Kolmogorov-Smirnov distribution. Taking these repeated samples we can not only inspect the distribution of the single observations, but the joint distribution of samples, each consisting of n sample elements.

| Exercise 1.16 | |
|----------------------|--|
| | <p>For $n = 10, 50, 100$, draw 300 samples using <code>runif(n)</code>. For each sample, calculate the χ^2 and the Kolmogorov-Smirnov statistic.</p> <p>You have to chose a χ^2 test to take. What is your choice?</p> <p>Plot the distribution functions of these statistics and compare them to the theoretical (asymptotic) distributions.</p> <p>Are there any indications against the assumption of independent uniform random numbers?</p> <p><i>Hint:</i> the functions for the χ^2 and Kolmogorov-Smirnov test keep their internal information as a list. To get the names of the list elements, you can create sample object. For example, use</p> <pre>names(chisq.test(runif(100))).</pre> |

Power

So far in our discussion the uniform distribution the target model distribution, our "hypothesis". We have discussed how various methods will behave, if this hypothesis

a:ch01:e10kChiSq

holds. The distribution properties derived from the model assumption can indicate how to find critical values for formal tests. We reject the hypothesis, if the observed test statistics are too extreme. What “too extreme” means is determined by the distributions we derived from the hypothesis. This leads to decision rules like:

reject the hypothesis, if $F_{\chi^2}(\widehat{\chi^2}) \geq 1 - \alpha$

or

reject the hypothesis, if $F_{Kolmogorov-Smirnov}(\kappa) \geq 1 - \alpha$
for chosen (small) values of α .

In the next step, if we have fixed a formal decision rule formal, we can ask for the power of this rule to come to a rejection if the hypothesis actually does not hold. A more precise analysis of the power of decision rules is a theme in classical lectures in statistics. The possibilities discussed so far however allow to analyse the performance using Monte Carlo strategies.

As framework for simulations we chose a family of alternatives. The uniform distribution blends into the family of beta distributions with densities

$$p_{a,b}(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1} (1-x)^{b-1} \quad \text{for } a > 0, b > 0 \text{ and } 0 < x < 1.$$

We chose alternative distributions from this family. From these we draw repeated samples, and apply our decision rule formally on each of these.

We note whether the procedure leads to a rejection of the hypothesis or not. Give a choice of the sample size n and a number of simulations m , and given a choice of a limiting probability α we get a table

$$(a, b) \mapsto \# \text{ simulations leading to a rejection of the hypothesis.}$$

In particular for the uniform distribution $(a, b) = (1, 1)$ we expect approximately $m \cdot \alpha$ rejections. For distributions other than the uniform hypothesis, a decision procedure is more powerful if the proportion of rejections is higher.

a:ch01:e11ts

| Exercise 1.17 | |
|----------------------|---|
| ** | <p>analysis the power the Kolmogorov-Smirnov test and the χ^2 tests. Select values for n, m and α, and choose 9 pairs for (a, b). What are your arguments for your choices?</p> <p>Use your chosen parameters to draw samples from <code>rbeta()</code>. Apply the Kolmogorov-Smirnov test and a χ^2 test with 10 cells of equal size on $(0, 1)$.</p> |
| | (cont.)→ |

| expected value textb: | Exercise 1.17 | (cont.) |
|--------------------------|----------------------|--|
| | | <p>Chose alternative parameters (a, b) so that you can compare the decision rules along the following lines:</p> <ul style="list-style-type: none"> i) $a = b$ ii) $b = 1$ iii) $a = 1$ <p>and run these simulations.</p> |
| | | <p>Chose alternative parameters (a, b) so that you can compare the decision rules over the range $0 < a, b < 5$.</p> <p>Your conclusions?</p> <p><i>Hint: outer(x, y, fun)</i> applies a function <i>fun()</i> to all pairs of values from <i>x, y</i> and returns the result as a matrix.</p> <p>Using</p> $\text{contour}()$ <p>you can generate a contour plot. See <i>demo("graphic")</i>.</p> |

a:ch01:e12pseudo

| Exercise 1.18 | |
|---------------|---|
| ** | <p>Design a test strategy to unmask “pseudo random numbers” .</p> <p>Test this strategy using simple examples</p> <ul style="list-style-type: none"> i) $x \quad x = 1..100 \mod m$ for convenient m ii) $\sin(x) \quad x = 1..100$ iii) ... <p>Do you tag these sequences as “not random”?</p> <p>Now try to unmask the random number generators provided by R. Can you tag the generated sequences as “not random”?</p> |

1.4 Moments and Quantiles

Distribution functions or densities are no easy objects from a mathematical point of view. In general, the space of functions is infinite dimensional. Finite dimensional geometrical or optimisation arguments cannot be applied directly. To simplify analysis one often resorts to finite dimensional descriptions.

Historically, moments have play an important role: probability distributions are viewed like mass distributions, and the moments are defined in analogy to the moments in mechanics. The first moment, corresponding to the center of gravity, is called **expected value** in statistics.

Definition 1.11 If X is a real valued random variable with distribution P , the expected value of X is defined as

$$E_P(X) := E(X) := \int X dP$$

provided this integral exists.

```
variance|textbf
standard
    deviation|textbf
variance!empirical|textbf
variance!sample|textbf
mean@mean|textit
var@var|textit
sd@sd|textit
```

The second moment and higher moments are usually centred. For the second (central) moment, the **variance**, we have the following definition:

Definition 1.12 If X is a real valued random variable with distribution P , its variance is defined as

$$\text{Var}_P(X) := \text{Var}(X) := \int (X - E(X))^2 dP.$$

provided this integral exists.

The integral do not always give defined values. Hence the moments do not always exists. But if they exist, they give a first information about the distribution. The expected value is often interpreted as the “statistical average” or mean of the distribution. The root of the variance, the **standard deviation**, is interpreted as the “statistical spread”.

The definitions can be applied to empirical distributions as well. This gives a first possibility to estimate the moments of an unknown theoretical distribution from the data. For the expected value, the mean, we have consistency:

$$E_P(E_{P_n}(X)) = E_P(X),$$

That is in the statistical average the empirical mean and the mean of the underlying theoretical distribution coincide (provided they are defined).

For the variance this consistency does not hold, but we have

$$\frac{n}{n-1} E_P(\text{Var}_{P_n}(X)) = \text{Var}_P(X),$$

for $n > 1$. The mathematical background is, that the expected value is a linear operator. It commutes with linear operators. But the variance is a quadratic operator, and that requires a correction if we want consistency. The variance with this consistency correction is often called the **empirical variance** or **sample variance**.

For the estimation of the first two moments of a vector of random numbers, R provides two functions: `mean()` estimates the mean and `var()` estimates the variance, using the sample variance. The function `sd()` gives the standard deviation of a vector.

a:ch01:01-18

| Exercise 1.19 | |
|---------------|---|
| | Generate a sample of random variables with sample size 100 from the distributions with the following densities: (cont.)→ |

`quantile@quant
summary@summar`

| | |
|----------------------|--|
| Exercise 1.19 | (cont.) $p(x) = \begin{cases} 0 & x < 0 \\ 1 & 0 \leq x \leq 1 \\ 0 & x > 1 \end{cases}$ <p>sowie</p> $p(x) = \begin{cases} 0 & x \leq 0 \\ 2 & 0 < x \leq 1/4 \\ 0 & 1/4 < x \leq 3/4 \\ 2 & 3/4 < x \leq 1 \\ 0 & x > 1 \end{cases}$ <p>Estimate the mean, variance and standard deviation in each of these. Repeat the estimation for 1000 samples. analysis the distribution of estimated mean, variance and standard deviation for repeated samples.</p> |
|----------------------|--|

Moments can be calculated using simple arithmetic operations. Their combination (exact or approximate) follows simple laws. However they are quite sensitive. Even shifting minimal probability mass can lead to a breakdown. For the empirical distribution this means: if a proportion of $1 - \varepsilon$ of the observed data follows a model distribution an a proportion of ε from some different distribution, the moments can take any value, even for arbitrarily small values of ε . Quantiles are much more robust against a break down. So at least 50% of the data have to be "outliers" to affect the median, while changes in just one data point can give any value to the mean.

With the availability of sufficient computing power, quantiles have gained more importance as descriptors. Calculating quantiles implicitly requires sorting. Hence it is more complex than the calculation of moments. Rules for combination of quantiles are not as simple as those for moments and often need explicit calculations. But with the technical resources available now this is no essential restriction.

R provides several functions to work with quantiles.

`quantile()` is an elementary function to calculate quantiles. The function `summary()` gives a summary of the distribution information, which includes information about quantiles.

`a:ch01:e19`

| | |
|----------------------|--|
| Exercise 1.20 | |
| | <p>Generate a sample of 100 random variables from the distributions of exercise 1.19.</p> <p>Estimate the median, and the lower and upper quartile.</p> <p style="text-align: right;">(cont.)→</p> |

| | |
|----------------------|--|
| Exercise 1.20 | (cont.) |
| | Repeat the estimation for 1000 samples. analysis the distribution of the estimated median, lower and upper quartile from repeated samples. |

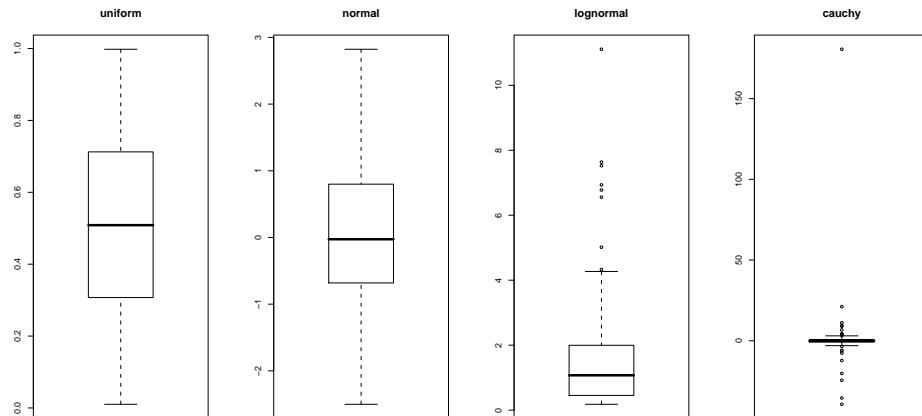
`boxplot()` gives a graphical representation of a quantile based summary. The “box&whisker plot” which is used here has many variations. So for interpretation of a box&whisker plot it is necessary to get detailed information which convention is used. Usually a box is used to mark the central part of the distribution. In the standard version a line is used to mark the median, and a “Box” goes from the median of the upper half to the median of the lower half. Roughly this corresponds to the upper and lower quartile. The finer definition takes care that the information is still reliable if the data contains ties, that is repeated observations with the same values. The “whisker” describes the adjacent area. Outliers are marked in particular.

`expl:ch01-box`

Example 1.10:

Input

```
oldpar <- par(mfrow = c(1, 4))
boxplot(runif(100), main = "uniform")
boxplot(rnorm(100), main = "normal")
boxplot(exp(rnorm(100)), main = "lognormal")
boxplot(rcauchy(100), main = "cauchy")
par(oldpar)
```



Theorem [1.4](#) gives a possibility to determine confidence intervals for quantiles, that are valid independent of the form of the underlying distribution.

To get an upper bound for the p -quantile x_p of a continuous distribution function by an order statistics $X_{(k:n)}$ with confidence level $1 - \alpha$ we look for

$$\min_k : P(X_{(k:n)} \geq x_p) \geq 1 - \alpha.$$

But $X_{(k:n)} \geq x_p \iff F(X_{(k:n)}) \geq p$ and by theorem [thm:s01-beta](#) we have

$$P(X_{(k:n)} \geq x_p) = 1 - F_{beta}(p; k, n - k + 1).$$

So we can determine \min_k directly using the beta distribution, or we use the relation with the binomial distribution and calculate k as

$$\min_k : P_{bin}(X \leq k - 1; n, p) \geq 1 - \alpha.$$

`a:ch01:e4`

| Exercise 1.21 | |
|---------------|--|
| | <p>For continuous distributions and the distribution median X_{med} we have $P(X_i \geq X_{med}) = 0.5$. Hence we can find a k such that</p> $k = \min\{k : P(X_{(k)} \leq X_{med}) < \alpha\}$ <p>and $X_{(k)}$ as an upper bound for the median with confidence level $1 - \alpha$.</p> <p>Use this idea to construct a confidence interval for the median with confidence level $1 - \alpha = 0.9$.</p> |
| | <p>Modify the box & whisker plot to show this interval.</p> <p><i>Hint:</i> You need the distribution function F_X, evaluated at the position marked by the order statistics $X_{(k)}$. The distributions of $F_X(X_{(k)})$ is discussed in I.4.</p> |
| | <p>The box & whisker plot offers an option <code>notch = TRUE</code> to mark confidence intervals. Try to use the documentation to find out how a <code>notch</code> is calculated. Compare your confidence intervals with those marked using <code>notch</code>.</p> |
| * | <p>Use an analogous strategy to get a distribution independent confidence interval for the interquartile distance.</p> |
| *** | <p>Augment the box & whisker plot so that gives information about the scale in a way which is statistically reliable.</p> <p><i>Hint:</i> Why is it not sufficient to mark confidence intervals for the quartiles?</p> |

1.5 R Complements

s:1.3

1.5.1 Random Numbers

If we had independent identical uniform distributed random numbers, we could generate random numbers with many other distributions. For example:

Lemma 1.13 (*Inversion method: If (U_i) is a sequence of independent identically $U[0, 1]$ distributes random variables and F a distribution function, then $(X_i) := (F^{-1}U_i)$ is a sequence of independent identically distributed random variables with distribution F .*)

From an analytical point of view, this lemma is only usable if F^{-1} is known. Numerically, it has a much wider use. Instead of F^{-1} we often use only approximations, sometimes just an inversion table.

The inversion method is a method to use uniformly distributed random numbers to get random numbers with other distributions of interest. Other, some times much more effective methods to get random numbers with specific distributions are discussed in the literature on statistical simulation.

For a range of distributions, transformed random number generators are supplied with R. A list is given in appendix (Seite A-84). For each distribution family there is a series of function. The names for these functions is derived from the short name of the distribution. For the family xyz , $rxyz$ is a function that generates random numbers. $dxyz$ calculates the density or the point measure for this family. $pxyz$ gives the distribution function and $qxyz$ the quantiles[‡].

To Do:
add references

Some selected distributions. For more distributions distributions see A.33 (Seite A-84).

| Distribution | Random Numbers | Density | Distribution Function | Quantiles |
|------------------|---------------------|---------------------|-----------------------|---------------------|
| Binomial | <code>rbinom</code> | <code>dbinom</code> | <code>pbinom</code> | <code>qbinom</code> |
| Hypergeometrisch | <code>rhyper</code> | <code>dhyper</code> | <code>phyper</code> | <code>qhyper</code> |
| Poisson | <code>rpois</code> | <code>dpois</code> | <code>ppois</code> | <code>qpois</code> |
| Gauß | <code>rnorm</code> | <code>dnorm</code> | <code>pnorm</code> | <code>qnorm</code> |
| Exponential | <code>rexp</code> | <code>dexp</code> | <code>pexp</code> | <code>qexp</code> |

[‡] that is confusingly with the notations which are common in statistics we have $p_{xyz} \equiv dxyz$ and $F_{xyz} \equiv pxyz$.

PP
 $\text{plot}|\text{textbf}$ 1.5.2 Graphical Comparisons
probability
 $\text{plot}|\text{textbf}$ Differences for simple geometrical forms are perceived easier than differences between
QQ
 $\text{plot}|\text{textbf}$ general graphs of similar form. So it can be helpful to choose representations leading to
quantile
 $\text{plot}|\text{textbf}$ simple forms such as straight lines. For example to compare two distribution functions
 F, G , instead of comparing the function graphs we can consider the graph of

$$x \mapsto (F(x), G(x)).$$

This graph is called the *PP plot* or *probability plot*. If the distributions coincide, this graph shows a diagonal straight line. Deviations from the diagonal are perceived easily.

Alternatively, we can use the observation scale as a reference. So we consider the graph of

$$p \mapsto (F^{-1}(p), G^{-1}(p))$$

This graph is called the *QQ plot* or *quantile plot*. If the distributions coincide, this graph again shows a diagonal straight line.

For the special case of uniform distributions on $[0, 1]$ on this interval we have $x = F(x) = F^{-1}(x)$, that is the *QQ* plot and the *PP* plot coincide. In this case, they both are just the graph of the distribution function. For non uniform distributions in the *PP* plot the distribution functions are standardised to the probability scale $[0, 1]$, and in the *QQ* plot they are re-scaled to the observation scale.

a:ch01qqpp

| Exercise 1.22 | |
|---------------|---|
| | Generate a <i>PP</i> plot of the $t(\nu)$ distribution against the standard normal distribution in the range $0.01 \leq p \leq 0.99$ for $\nu = 1, 2, 3, \dots$ |
| | Generate a <i>QQ</i> plot the $t(\nu)$ distribution against the standard normal distribution in the range $-3 \leq x \leq 3$ for $\nu = 1, 2, 3, \dots$ |
| | How large must ν be so that the t distribution is barely different from the normal distribution in these plots? |
| | How large must ν be so that the t distribution is barely different from the normal distribution if you compare the graphs of the distribution functions? |

If the distributions are related by an affine transformation in the observation space, the *QQ* plot still shows a straight line; slope and intercept represent the affine transformation. This applies for example to the family of the normal distributions: if F is the standard normal distribution $N(0, 1)$ and $G = N(\mu, \sigma^2)$, the *QQ* plot is a straight line with intercept μ and slope σ .

For the empirical distributions, corollary I.5 applies: instead of i/n we use a reference point which corrects for the skewness of the distribution, so that on the average a straight line is generated. The quantile plot, using this correction for empirical distributions, is provided as function *qqplot()*. For the special case of the normal distribution there is a

corol:ewquantil

variation of `qqplot()` available as `qqnorm()` to compare an empirical distribution with the theoretical normal distribution.

By transforming to probability scale or observation scale the graphical procedures gain power. So for example using a sample size of $n = 50$, to tell the distribution function of the normal distribution from that of a uniform distribution often needs a well trained observer. In the normal *QQ* plot however the uniform samples show up as markedly non-linear, whereas normal sample largely show linear pictures, and the difference becomes obvious.

To illustrate this, we generate random samples:

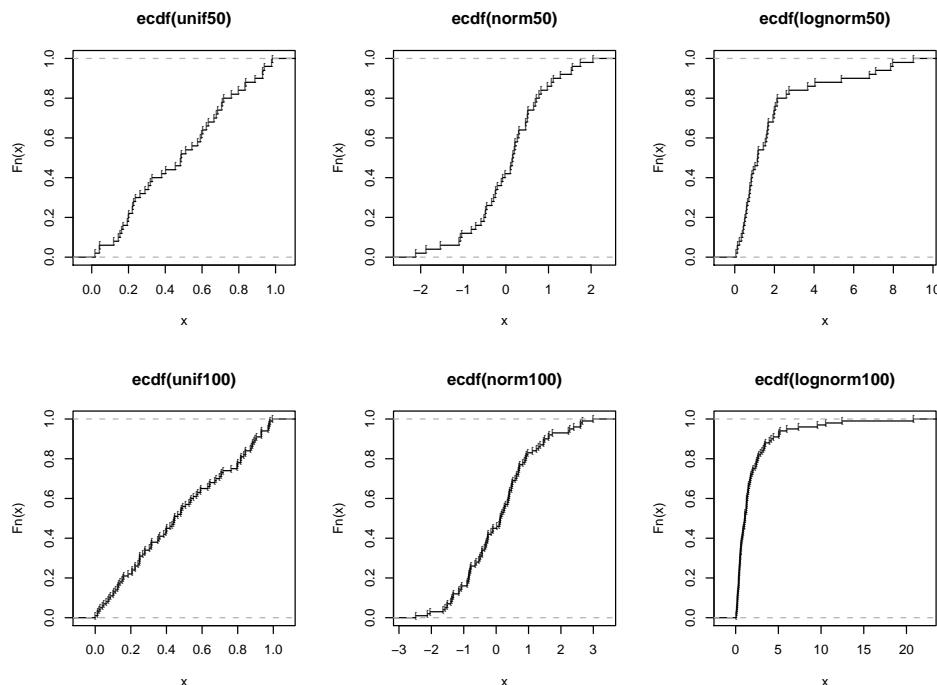
Input

```
unif50 <- runif(50)
unif100 <- runif(100)
norm50 <- rnorm(50)
norm100 <- rnorm(100)
lognorm50 <- exp(rnorm(50))
lognorm100 <- exp(rnorm(100))
```

expl:ch01-compecdf Using these data sets we generate plots of the distribution functions.

Example 1.11:*Input*

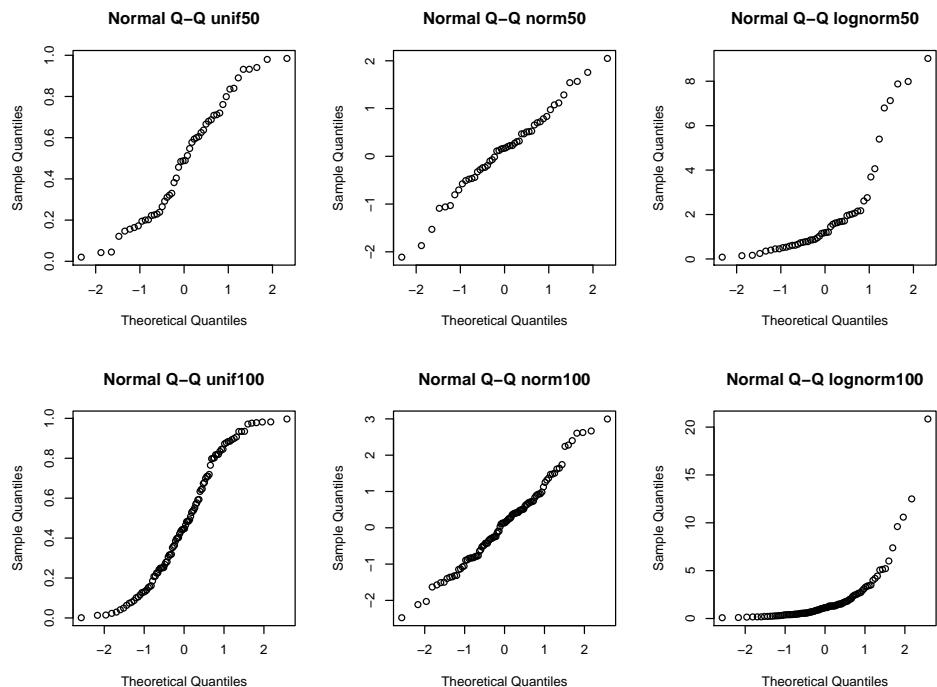
```
oldpar <- par(mfrow = c(2, 3))
plot(ecdf(unif50), pch = "[")
plot(ecdf(norm50), pch = "[")
plot(ecdf(lognorm50), pch = "[")
plot(ecdf(unif100), pch = "[")
plot(ecdf(norm100), pch = "[")
plot(ecdf(lognorm100), pch = "[")
par(oldpar)
```



expl:ch01-compqq For comparison, here are the corresponding *QQ* plots for the same data:

Example 1.12:Input

```
oldpar <- par(mfrow = c(2, 3))
qqnorm(unif50, main = "Normal Q-Q unif50")
qqnorm(norm50, main = "Normal Q-Q norm50")
qqnorm(lognorm50, main = "Normal Q-Q lognorm50")
qqnorm(unif100, main = "Normal Q-Q unif100")
qqnorm(norm100, main = "Normal Q-Q norm100")
qqnorm(lognorm100, main = "Normal Q-Q lognorm100")
par(oldpar)
```



a:ch01:e1vf

Exercise 1.23

Use *PP* plots instead of distribution functions to illustrate the χ^2 - and Kolmogorov-Smirnov approximations.

a:ch01:e2qq

ToDo:
better
irexercise

| Exercise 1.24 | |
|---------------|--|
| | Use <i>QQ</i> plots instead of distribution functions. Can you add confidence regions to these plots with the help of the χ^2 - resp. Kolmogorov-Smirnov statistics? |

To get an impression of the fluctuation, we have to compare the empirical plots with typical plots of a model distribution. A plot matrix is a simple way. Here is an example for the normal *QQ* plot, implemented as a function:

```
Input
```

```
qqnormx <- function(x, nrow = 5, ncol = 5, main = deparse(substitute(x))) {
  oldpar <- par(mfrow = c(nrow, ncol))
  qqnorm(x, main = main)
  for (i in 1:(nrow * ncol - 1)) qqnorm(rnorm(length(x)),
    main = "N(0, 1)")
  par(oldpar)
}
```

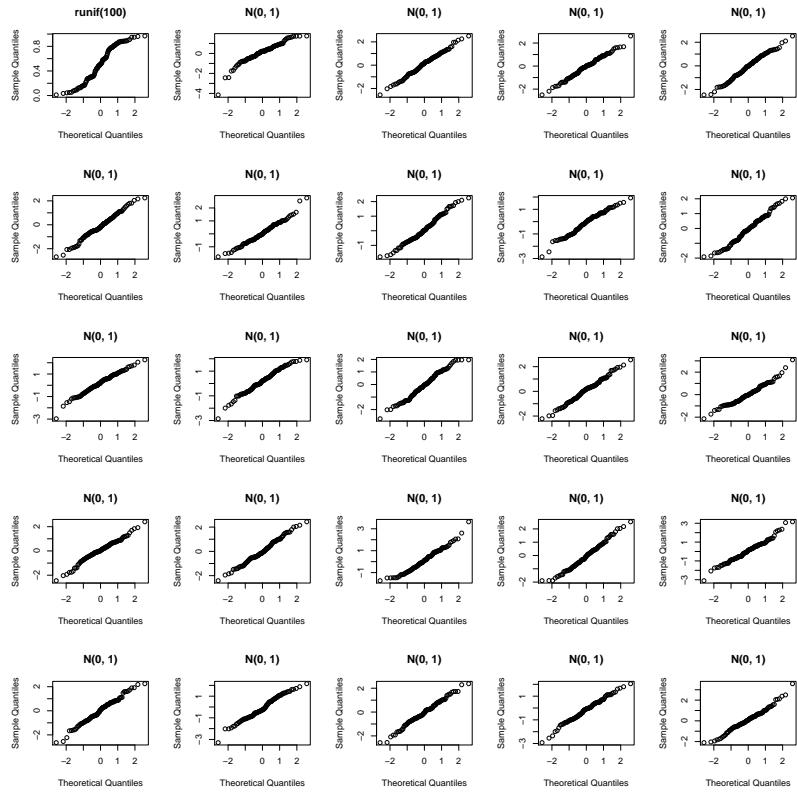
In this example we used a *for* loop. Like all programming languages, R has control structures such as loops or conditional statements. A summary of control structures in R is in appendix A.24.

ToDo:

avoid
loops

Deviations from a linear structure should be considered as fluctuation if they stay within the frame of the simulated examples. If the data set under investigation is extreme in comparison to the simulated examples, this indicates a contradiction to the model assumptions.

expl:ch01-qqmat

Example 1.13:`qqnormx(runif(100))`*Input*

On the long run it is worth to modify the plot functions so that they give information about the fluctuation which is to be expected. In example 1.9 we have constructed Monte Carlo bands for the distribution function. We can generalise this idea for the *PP* plot and the *QQ* plot. All that is needed is to transform the bands to the adequate scale used in the plot.

`a:ch01:e3ppqq`

| Exercise 1.25 | |
|----------------------|---|
| | Use <code>rnorm()</code> to generate with pseudo random numbers for the normal distribution for sample size $n = 10, 20, 50, 100$. For each sample, generate a <i>PP</i> plot and a <i>QQ</i> plot, using the theoretical normal distribution as a reference. |
| | (cont.)→ |

`legend@legend`

| Exercise 1.25 | (cont.) |
|----------------------|--|
| * | <p>Add Monte Carlo bands from the envelop of 19 simulations. Instead of the uniform distribution, you have to use the normal distribution to generate the Monte Carlo bands. Then you have to represent the results in the co-ordinate system of the <i>QQ</i> plots, that is the x axis represents the quantiles of the normal distribution. Hint: inspect the source of <i>qqnorm()</i>.</p> <p>The bands are initially bands for the standard normal distribution. Find bands adjusted in scale and location of the data at hand.</p> |

1.5.3 Complements: Enhancing Graphical Displays

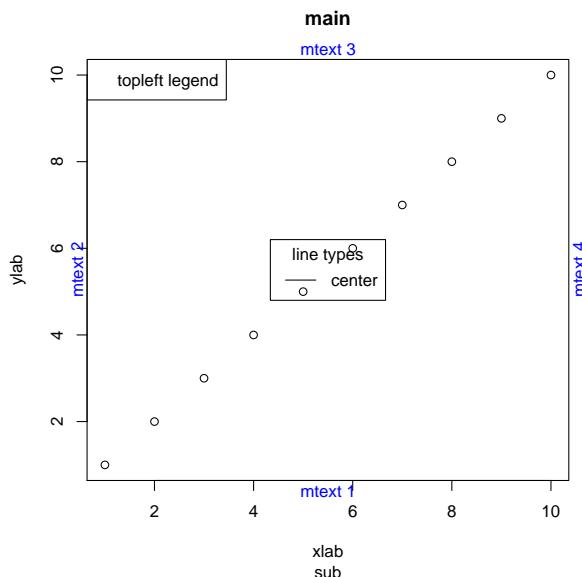
So far the the R graphics has been used in rudimentary form only. For serious work the graphics has to be enhanced to make its information identifiable and readable. This implies adding headers, captions, axis labels etc. In R there are “high level” and “low level” graphic functions. “High level” functions generate a new display. Additionally, they provide possibilities to control general graphics parameters for the use in this display.

The “low level” functions add elements to an existing plot or modify it in some detail. So for example the function `legend()` can add legends inside a plot.

`expl:ch01-mtext`

Example 1.14:

```
function|textbf
plot(1:10, xlab = "xlab", ylab = "ylab", main = "main",
     sub = "sub")
mtext("mtext 1", side = 1, col = "blue")
mtext("mtext 2", side = 2, col = "blue")
mtext("mtext 3", side = 3, col = "blue")
mtext("mtext 4", side = 4, col = "blue")
legend("topleft", legend = "topleft legend")
legend("center", legend = "center", lty = 1:4, title = "line types")
```



a:ch01:e4plot

| Exercise 1.26 | |
|---------------|--|
| | Use <code>help(plot)</code> to inspect the possibilities to customise the plot function. Some information on detail of the parameters is only available if you use <code>help(plot.default)</code> . Modify your latest plot to have a correct main title. |

See also: [14] Ch. 12.

1.5.4 Complements: Functions

R command can be grouped to functions. Functions may be parametrised. Functions provide a flexible way of code reusability.

ToDo:

Add reference to lattice

expl:ch01-envfun Example for a function

Example 1.15:

```
ppdemo <- function(x, samps = 19) {  
  Input _____  
  y <- (1:length(x))/length(x)  
  plot(sort(x), y, xlab = substitute(x), ylab = expression(F[n]),  
       main = "distribution function with Monte Carlo bands (unif.)",  
       type = "s")  
  mtext(paste(samps, "Monte Carlo samples"), side = 3)  
  samples <- matrix(runif(length(x) * samps), nrow = length(x),  
                    ncol = samps)  
  samples <- apply(samples, 2, sort)  
  envelope <- t(apply(samples, 1, range))  
  lines(envelope[, 1], y, type = "s", col = "red")  
  lines(envelope[, 2], y, type = "s", col = "red")  
}
```

ToDo:

copy
function

In `ppdemo()` we used the function `mtext()` to generate axis annotations.

Functions are called using the form `<name>(<actual parameter list>)`.

expl:ch01-ppdemo

syntex

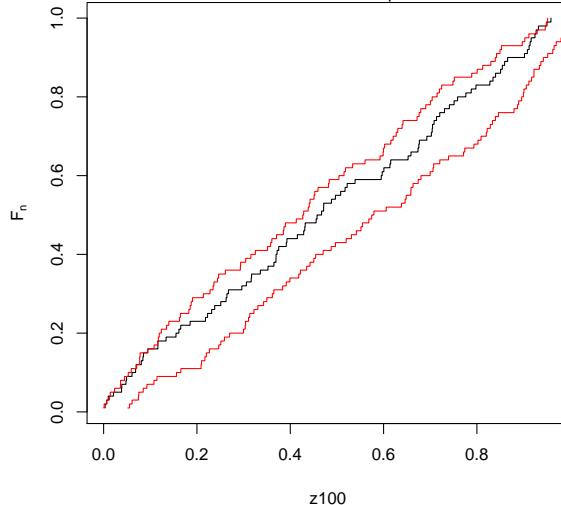
Example 1.16:

```
z100 <- runif(100)
ppdemo(z100)
```

Input

distribution function with Monte Carlo bands (unif.)

19 Monte Carlo samples



If only the name of a function is given as input, the definition of the function is returned, that is the function is listed. Example:

Example 1.17:

```
ppdemo
```

Input

```
function (x, samps = 19)
{
  y <- (1:length(x))/length(x)
  plot(sort(x), y, xlab = substitute(x), ylab = expression(F[n]),
       main = "distribution function with Monte Carlo bands (unif.)",
       type = "s")
  mtext(paste(samps, "Monte Carlo samples"), side = 3)
  samples <- matrix(runif(length(x) * samps), nrow = length(x),
                    ncol = samps)
  samples <- apply(samples, 2, sort)
  envelope <- t(apply(samples, 1, range))
  lines(envelope[, 1], y, type = "s", col = "red")
  lines(envelope[, 2], y, type = "s", col = "red")
}
```

Output

a:ch01:fun

| Exercise 1.27 | |
|---------------|--|
| | Inspect <code>rnorm()</code> with <code>qqplot()</code> and <code>plot()</code> . Rework your programming exercises and write reusable parts as functions. |

ToDo:

more
precise
irexercise

ToDo:

example in
appendix

Parameters for functions are passed by value. Each function receives a copy of the actual parameter values upon call. This guarantees a safe programming environment. On the other side, this leads to memory requirements and a time penalty. In situations where the parameter size is large or time requirements are critical this expenditure can be avoided by direct access to variables which are defined in the function environment. Techniques to achieve this are documented in [R:GentlemanThaka:2000].

Functions in R can be nested, that is one “outer” function can contain other “inner” functions. These inner functions are not visible globally, but only in their containing outer function.

Functions can have objects as results. An object is explicitly returned as a result by `return(obj)`. Results can be returned implicitly as well. If the end of a function is reached without calling `return()` the value the last evaluate expression is returned.

| Input | Output |
|--|--|
| <code>circlearea <- function(r) r^2 * pi</code> | |
| <code>circlearea(1:4)</code> | |
| | [1] 3.141593 12.566371 28.274334 50.265482 |

Results can also be provided so that they are only passed upon request. We have met this technique when studying the histogram. The function call `hist(x)` does not return a result, but has the (intended) side effect to draw a histogram. But if we use `hist()` in an expression, for example in an assignment `xhist <- hist(x)`, we get a description of the histogram returned as the function value. To return results only upon request, instead of `return(obj)` the expression `invisible(obj)` is used.

a:ch01:fun

| Exercise 1.28 | |
|---------------|--|
| | <p>Write as functions:</p> <ul style="list-style-type: none"> • A function <code>ehist</code> showing an augmented histogram. • A function <code>eecdfe</code> showing the empirical distribution shows. • A function <code>eqqnorm</code> showing a <i>QQ</i> plot with the standard normal distribution as comparions. • A function <code>eboxplot</code> showing a Box&Whisker-Plot. • A wrapper function <code>eplot</code> showing a plot matrix with these four. <p>Your functions should call the standard functions (or modify them, if necessary) and guarantee that the plots have an adequate complete annotation.</p> |

While statements in R are processed stepwise and allow inspection of the results step by step, upon call of a function all statements for the function are executed as a block. This can provide a problem for error diagnosis. R provides possibilities to inspect specific functions, in particular to allow stepwise processing of function. For details see appendix A.23 “Debugging and Profiling” on page Seite A-68.

```
parse@parse|textit
eval@eval|textit
print@print|textit
environment@environment|textit
search@search|textit
ls@ls|textit
```

1.5.5 Complements: R Internals

A typical processing step in R handles a command in three sub-steps

- `parse()` analyses a textual input and translates it into an internal representation as an R expression. R expression are R objects of a special kind.
- `eval()` interprets this expression and evaluates it. The result is again an R object.
- `print()` displays the resulting object.

Details have to be added:

Parse

The first step consists of two parts, a scanning which scans the input and splits it into tokens, and the parsing proper which takes the tokens and tries to combine them to syntactically correct expressions. The function `parse()` combines both steps. `parse()` scan work of local files or data streams as well as on external files denoted by an URL reference.

As an inverse function `substitute()` is provided. A typical application is to decode the actual parameters of a function call (using `parse()`) and then to derive an informative annotation (using `substitute()`)

Eval

The function `eval()` evaluates an R expression. For this step the references in expressions have to be translated into values, taking into account the actual environment. Since R is an interpreted system, environmental conditions can change. So the evaluation of one expression can lead to different results, depending on the environment.

Each function defines its own local environment. As functions can be nested, so can be the environments. Environments can also change as additional libraries are loaded into R. The actual local environment can be queried using `environment()`. With `search()` you get a list of the environments which are searched successively to resolve references. `ls()` gives a list of the objects in an environment.

R’s extensibility implies the possibility that names can collide and the translation of names into actual values becomes a problem. As a safeguard, R 2.x provides the possibility to collect names in (guarded) name spaces. In most case this transparent for the user; the name resolution follows a search path which is determined by the chain of environments. To access an object in a specific name space explicitly, the name space can be specified. So for example `base::pi` can be used as an explicit name for the constant `pi` in the name space `base`), and this access will work even if the user has chosen to introduce some variable with name `pi`, using some arbitrary value.

```
print@print|textit
print@print|textit
polymorph
print@print|textit
source@source|Textit
The function print() is implemented as a polymorphic function. To execute print() R eval-
Sweave@Sweave|textituates the class of the objects to print and chooses an appropriate print methods. Details are give
install.packages@install|Ch02_PolyMorpH|textit
in section 2.6.5 Seite 2-42.
update.packages@update.packages|textit
```

Executing Files

The function `source()` is available to use a file as input for R. The file can be local, or specified by an URL reference. Conventionally for names of R source files, the suffix `.R` is used.

The function `Sweave()` allows to interweave documentation and commands. Conventionally for names of `Sweave()` source files the suffix `.Rnw` is used. Details about the format is in the `Sweave()` documentation

[<http://www.ci.tuwien.ac.at/~leisch/Sweave/Sweave-manual-20060104.pdf>](http://www.ci.tuwien.ac.at/~leisch/Sweave/Sweave-manual-20060104.pdf).

1.5.6 Complements: Packages

sub ToDo: lib

package
or library
as basic
term?

Functions, examples, data sets etc. can be bundled in R packages following certain conventions. The conventions may change with different implementations. The conventions documented in R [\[R:Ext\]](#) [18] should be used as a reference. Some package are part of R be default. Packages for special purposes can be found on the internet, for example using [<http://www.cran.r-project.org/src/contrib/PACKAGES.html>](http://www.cran.r-project.org/src/contrib/PACKAGES.html).

Packages which are not bundled with R must be installed into the R system first. In general, system specific commands are supplied for this installation. A more convenient way is to do the installation from inside R using `install.packages()`. If no special source is given, `install.packages` tries to access a prepared address (usually the CRAN address given above). But `install.packages` can load packages from any repository specified. In particular, using the form `install.packages(<package>, repos = NULL)`, `<package>` can be an access path for direct access on your local machine.

The function `update.packages()` compares installed versions with the recent state in the repository and updates the version installed locally if necessary.

Once a package is installed, it can be loaded using

`library(pkgnname)`

After that, the objects (functions, data sets, ...) defined in the package (functions, data sets, ...) are retrievable on the current search path and can be used directly in any expression.

Packages are released using

`detach(pkgnname)`

ToDo:
search
path

After this, the package objects do not appear in the current search path any more.

Technically packages directories, following the R conventions. Usually they are provided in packed compresses form such as `.tar.gz` files. In the beginning, you will install pre-compiled binary version of packages. If necessary, for example to inspect the source code, the source version can be installed by the same functions.

It is worth following the R even as a guideline how to organise your own work. For the organisation of your own work it is worth following the R conventions and to organise related parts as R packages. R provides several tools supporting package administration. The conventions and the tools available to work with packages are documented in [18]. For Unix/Linux/Mac OS X, the main tools are available as commands:

```
R CMD check <directory> # checks a directory for compliance with the R conventions
R CMD build <directory> # generates an R package
```

As a first step: The function `package.skeleton()` helps constructing new packages. `package.skeleton()` does not only generate a package skeleton, but also a help file documenting the next steps to build a loadable package.

Packages must have a file DESCRIPTION with specific information. The Details are documented in [18], and a prototype is generated by `package.skeleton()`. The other components are optional.

```
package.skeleton@package
package.skeleton@package
package.skeleton@package
```

| Name | Kind | Content |
|-------------|-----------|--|
| DESCRIPTION | file | a source description following format conventions. |
| R | directory | R code. Files in this directory should be readable by <code>source()</code> . Recommended name suffix: .R . |
| data | directory | additional data. Files in this directory should be readable by <code>data()</code> . Recommended name suffixes and formats: .R for R code. Alternatively: .r .tab for tables. Alternatively: .txt, .csv .RData for output of <code>save()</code> . Alternatively: .rda. The directory should contain a file <i>00Index</i> with a short survey of the data sets included. |
| exec | directory | additional executable files, for example Perl- or shell scripts. |
| inst | directory | contents is copied (recursively) to the target directory of the installation. In particular, this directory can contain a file CITATION, which is evaluated by the Rfunction <code>citation()</code> . |
| man | directory | documentation in the R documentation format (see: [18] "Writing R extensions", available from < http://www.cran.r-project.org/ >). Recommended name suffix: .Rd |
| src | directory | Fortran, C and other sources. |
| demo | directory | executable examples. This directory should contain a file <i>00Index</i> with descriptions. |

a:ch01:e6lib

| Exercise 1.29 | |
|---------------|--|
| | <p>a:ch01:fun</p> <p>Install the functions from exercise 1.28 as a package. You can prepare the package with <code>package.skeleton()</code>, if you have already defined the functions.</p> <p>Load the package. Verify that you can still load the package with <code>library()</code> if you have restarted the R system.</p> <p><i>Hint:</i> For an object <i>x</i>, the statement <code>prompt(x)</code> generates a skeleton from which you can build a documentation for <i>x</i>.</p> |

1.6 Statistical Summary

In this chapter, our leading example was the statistical analysis of a (univariate) random sample. We used a model framework which is central to statistics. The values the random sample are considered realisations of a random variable, drawn from underlying theoretical distribution. The aim of the statistical analysis was the inference from the empirical distribution of the random sample on the unknown underlying theoretical distribution. This inference can take two forms: we can compare the empirical distribution with some hypothetical distribution. This is the approach of classical statistics. Or we can attempt to extract features of the underlying distribution from the empirical distribution. This is the data analysis approach.

Both approaches are closely related. The essential tool for both was analysis of the empirical distribution function.

1.7 Literature and Additional References

R:Ext

[18]R Development Core Team (2000-2005): Writing R Extensions.

See: <<http://www.r-project.org/manuals.html>>.

gnst77wth

[6] Gänßler, P; Stute, W.: Wahrscheinlichkeitstheorie Heidelberg: Springer 1977.

R:Gentleman+Ihaka:2000

[7] Gentleman, R.; Ihaka, R.: Lexical Scope and Statistical Computing. Journal of Computational and Graphical Statistics 9 (2000) 491–508.

Generated by SWeave from:

```
Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/Rnw/S01base.Rnw.tex,v
Revision: 1.15
Date: 2008/03/18 09:01:42
name:
Author: j40
$Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/Rnw/S01base.Rnw.tex,v $
$Revision: 1.15 $
$Date: 2008/03/18 09:01:42 $
$name: $
$Author: j40 $
```

CHAPTER 2

Regression

ch:02

2.1 General Regression Model

ToDo:
add more
regression
graphics

The (controlled) trial is a common paradigm from experimental sciences: under experimental conditions x measurement result y is recorded. This is composed of a systematic effect $m(x)$ and a measurement error ε .

$$y = m(x) + \varepsilon.$$

This is a particular view of the data. Instead of looking at the joint behaviour of n x and y , an asymmetric view is taken: x is the “source”, y (or a change in y) the effect. In this view, the “source” x is considered as given or under direct control; y is influenced indirectly via the experimental conditions. Experimental conditions should be chosen so the ε , the measurement error, is kept as small as possible and vanishes on average: there should be no systematic error.

From a statistical point of view the main difference of the roles of x and y is that for y the stochastic behaviour is modelled using ε , while x is considered “given” and no stochastics is allotted for x in the model.

To get a manageable frame, we consider the case where x can be represented as vector of real variables, $x \in \mathbb{R}^p$, and that the measurements gives one dimensional real values, $y \in \mathbb{R}$. In a stochastic model we can express the idea formally. One possible formalisation is to represent the measurement error ε as a random variable. Assuming additionally that the expected value of ε exist, the assumption that that the measurement error vanishes on average can be formalised as $E(\varepsilon) = 0$.

To analyse the systematic effect m we consider series of experiments. The index $i, i = 1, \dots, n$, an experiment in this series, and the model is

$$\begin{aligned} y_i &= m(x_i) + \varepsilon_i \quad i = 1, \dots, n \\ \text{with} \quad x_i &\in \mathbb{R}^p \\ E(\varepsilon_i) &= 0. \end{aligned}$$

Now the statistical problem is:

estimate the function m from the measurement results y_i at measurement conditions x_i . The keywords “curve estimation” or “Regression” lead to abundant statistical literature about this problem. The following discussion can only be an introduction.

We concentrate mainly on linear regression. This is a very simplified version of the general regression problem. But central aspects of regression can already be illustrating using the linear case. In practical applications, linear models are widely used, and general models are more restricted to sophisticated applications.

response|textbf To have a unified terminology, we call y_i the **response** and the components of x_{ij} with $j = 1, \dots, p$ are the **regressors**. The function m is called the **model function**.

ToDo: `move`, `fit`

linear

regressors

here

2.2 Linear Model

ToBe: `linear|textbf`

linear

compare

We begin with the regression model - now in vector notation * -

sion|seeregression,

fit, **error**

term

ToDo:

`matrix|textbf`

discuss

difference

to setting

to ch 01

Additionally we assume that m is linear. Then there exists (at least one) a vector $\beta \in \mathbb{R}^p$, so that

$$m(X) = X\beta.$$

The regression problem is now reduced to the exercise to estimate β from the information (Y, X) .

The modified regression model

$$\begin{aligned} Y &= X\beta + \varepsilon \\ &\quad Y \text{ with values in } \mathbb{R}^n \\ &\quad X \in \mathbb{R}^{n \times p} \\ &\quad \beta \in \mathbb{R}^p \\ &\quad E(\varepsilon) = 0 \end{aligned} \tag{2.1}$$

eq:regrmod

is called a **linear model** or **linear regression**. The matrix X which combines the values of the regressors, hence the information about the experimental conditions, is called the **design matrix** of the model.

expl:02slr

Example 2.1 (Simple Linear Regression) If the experimental condition is characterised by value of one real variable, which links to the experimental result via a linear model function

$$m(x) = a + b \cdot x,$$

so we can write a series of experiments with experimental result y_i give experimental condition x_i in co-ordinates as

$$y_i = a + b \cdot x_i + \varepsilon_i. \tag{2.3}$$

eq:02slrcoord

In matrix notation we can concentrate the series of experiments as linear model

$$Y = \underbrace{\begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}}_X \cdot \underbrace{\begin{pmatrix} a \\ b \end{pmatrix}}_\beta + \varepsilon. \tag{2.4}$$

eq:02slrmat

* We change conventions and notations whenever this change is helpful. The confusion is part of the conventions: in some common conventions, capital letters denote random variable. In other conventions, they denote functions, in still others they denote vectors. We have to leave the resolution to the reader.

Simple linear regression is one example for linear models. The one way classification is the other basic example.

expl:oneway

Example 2.2 (One Way Classification) *For comparison of k treatments (in particular for the special case $k = 2$) we use indicator variables which are combined in a matrix. The indicator variable for treatment i is in column i . Usually we have repeated observations $j = 1, \dots, n_i$ under treatment i , giving a total of $n = \sum_{i=1}^k n_i$ observations. The model*

$$Y = \underbrace{\begin{pmatrix} 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}}_X \cdot \underbrace{\begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_k \end{pmatrix}}_\beta + \varepsilon. \quad (2.5) \quad \text{eq:02oneway0mat}$$

corresponds in co-ordinates

$$y_{ij} = \mu_i + \varepsilon_{ij}. \quad (2.6) \quad \text{eq:02oneway0coord}$$

This is the typical model to test the hypothesis “no difference” $\mu_1 = \dots = \mu_k$ against the alternative that there is a difference in mean between the treatments.

The same relation can be represented if the measurement values are interpreted as a sum of a basic value μ_0 and an additional treatment effect $\mu'_i = \mu_i - \mu_0$. In co-ordinates, this reads

$$y_{ij} = \mu_0 + \mu'_i + \varepsilon_{ij}. \quad (2.7) \quad \text{eq:02oneway1coord}$$

In matrix notation, this is

$$Y = \underbrace{\begin{pmatrix} 1 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 1 & \dots & 0 \\ 1 & 0 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \dots & 1 \end{pmatrix}}_{X'} \cdot \underbrace{\begin{pmatrix} \mu_0 \\ \mu'_1 \\ \mu'_2 \\ \vdots \\ \mu'_k \end{pmatrix}}_{\beta'} + \varepsilon. \quad (2.8) \quad \text{eq:02oneway1mat}$$

Wilkinson-Rogers notation Example 2.2 illustrates that the representation of a problem as a linear model need not be unique. (2.5) and (2.8) are equivalent representations and only the application background can decide which one to prefer.

For mathematical analysis the design matrix X is an essential tool. For data analysis we use R to generate this matrix (implicitly) for us. R can understand a special notation, the **Wilkinson-Rogers notation**, for model description. With this notation we write

$$y \sim x.$$

The error term in this model is not written.

By default, a constant term is assumed. For the one way classification we get model (2.8). If we do not want a constant term (in the regression case for a regression through the origin; in the one way classification the model (2.5) where we do not include an overall mean), we have in co-ordinates

$$y_i = b \cdot x_i + \varepsilon_i.$$

In the Wilkinson-Rogers notation we have to set the constant term to zero explicitly:

$$y \sim 0 + x.$$

Addition regressors can be included using the operator $+$. So $y \sim u + v$ corresponds to the model (in co-ordinates)

$$y_i = a + b \cdot u_i + c \cdot v_i + \varepsilon_i.$$

We return to this notation in sections 2.2.4 and 2.3.

There is rich literature on linear models. The book “The Theory of Linear Models” of Bent Jørgensen [11] deserves special recommendation. It largely covers the mathematical background of this chapter and contain numerous illustrating examples.

2.2.1 Factors

subs:factor-1

Using the notation for the description of designs and models, a case oriented description of a design can be machine translated into a design matrix for a linear model in canonical form. Sometimes the translation needs a little bit of coaching. Consider for example a data set

```
y <- c( 1.1, 1.2, 2.4, 2.3, 1.8, 1.9)
x <- c( 1, 1, 2, 2, 3, 3).
```

The vector x can mean a quantitative vector for the regression model

$$y_i = a + b x_i + \varepsilon_i$$

to be used as a regressor. Or it can be intended for one way classification, the model of one way analysis of variance,

$$y_{ix} = \mu + \alpha_x + \varepsilon_{ix}$$

to be used as indicator for a treatment group. To mark the difference, vectors in R can be defined to be **factors**. Numeric vectors that are no factors are treated as quantitative variable, as in the first example. Factors are treated indicators and expanded into indicator variables when constructing the design matrix. So

$$y \sim x$$

results in the regression model, but

$$y \sim \text{factor}(x)$$

gives the variance model for the one way layout.

Specifying a parameter `ordered = TRUE` when calling `factor()` marks the variable as ordered. The variable specified will be treated as on an ordinal scale.

$$y \sim \text{factor}(x, \text{ ordered} = \text{TRUE})$$

Without this specification, factors are considered to be on a categorial scale.

```
factor@factor|textit
scale!ordinal
scale!categorical
levels|seefactor,
  levels|textbf
factor!levels
Gauß-Markov
  estimator|textbf
least
  squares
  estimator|textbf
lm@lm|textit
```

The values used for factors (both for categorial and ordered) may be mere names and will be substituted internally by numbers. If numerical values are used, they need not be in an ordered sequence.

So

$$\text{factor}(c(2, 2, 5, 5, 4, 4))$$

results in a vector having three factor values 1, 2, 3, with the names "2", "5" and "4". An example using a factor denoted by names would be for example

$$y \sim \text{factor}(c("Tmt1", "Tmt1", "Tmt2", "Tmt2", "Tmt3", "Tmt3"))$$

The values which can be taken by a factor are called **levels** of the factor. This is an attribute of the factor which can be accessed with `levels()`, as for example in

```
levels(factor(c(2, 2, 5, 5, 4, 4)))
levels(factor(c("Tmt1", "Tmt1", "Tmt2", "Tmt2", "Tmt3", "Tmt3")))
```

2.2.2 Least Squares Estimation

A first idea for estimation in a linear model can be gained from the following relation: given X , we have $E(Y) = X\beta$, hence $X^T E(Y) = X^T X\beta$ and so $(X^T X)^{-1} X^T E(Y) = \beta$. Here X^T means the transposed of the matrix X and $(X^T X)^{-1}$ the (generalised) inverse of $(X^T X)$. This equation motivates the following estimator:

$$\hat{\beta} = (X^T X)^{-1} X^T Y. \quad (2.9)$$

Using the model relation $Y = X\beta + \varepsilon$ ^{2.2.2} and `E(\varepsilon) = 0`, we get

$$E(\hat{\beta}) = E((X^T X)^{-1} X^T (X\beta + \varepsilon)) = \beta, \quad (2.10)$$

`eq:02-kqs`

`eq:02-unbiased`

that is $\hat{\beta}$ is an unbiased estimator for β . It is a topic in statistics lectures to discuss whether there are other qualities of this estimator beyond consistency. The Gauß-Markov theorem is a theorem from statistics characterising the estimator. We came back to this estimator frequently and give it a name for reference: **Gauß-Markov estimator**. In case of a linear model, such as the regression model, this estimator has a series of optimality properties. For example, this estimator minimises the mean quadratic deviation, that is it is a **least squares estimator** in this model.

The least squares estimator for linear models is implemented `lm()`.

We generate an example data set to be used for illustration.

`eq:linmod`

```
fit|textbf
x <- 1:100
err <- rnorm(100, mean = 0, sd = 10)
y <- 2.5 * x + err
```

We get the least squares estimator using

Example 2.1:

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

Coefficients:
(Intercept)          x
-4.633            2.548
```

a:ch02:lm01

| Exercise 2.1 | |
|--------------|---|
| | When we generated the data, we did not use a constant term. The model specified for estimation however did not exclude the constant term. Repeat the estimation using the model without a constant term. Compare the results. |

The estimator $\hat{\beta}$ immediately yields an estimation \hat{m} for the function m in our original model:

$$\hat{m}(x) = x^\top \cdot \hat{\beta}.$$

The evaluation at a point x results in values $\hat{y} := \hat{m}(x)$, the **fit** at point x . The evaluation at the measurement points results in the vector the fitted values $\hat{Y} = X\hat{\beta}$.

In our example the fit gives a regression line. Using `plot()` can plot the data points. If we store the result of the regression, can we use it with `abline()` to add the regression line.

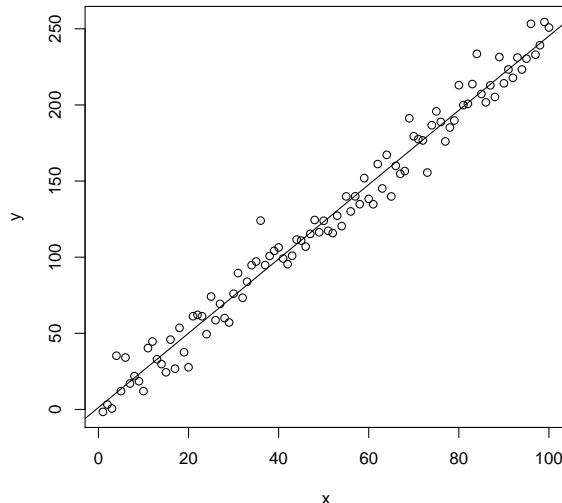
expl:02-lmplot

Example 2.2:

```
lmres <- lm(y ~ x)
plot(x, y)
abline(lmres)
```

Input

hat
matrix|textbf



Function `abline()` is a function to draw lines, using various parametrisations. For more information, see `help(abline)`.

Equation (2.9) tells us how to calculate the fit at the measurement points.

$$\hat{Y} = X(X^\top X)^{-1} X^\top \cdot Y. \quad (2.11)$$

eq:02-fit

The matrix

$$H := X(X^\top X)^{-1} X^\top \quad (2.12)$$

eq:02-hat

is called the **hat matrix**[†]. It is the main tool to analyse the Gauß-Markov estimator for a given design matrix X . The design matrix, and hence the hat matrix, depends only on the experimental conditions, but not on the result of the experiment. The fit on the other side always refers to a specific outcome of the experiment, the random sample of observed values Y .

The linear model contains a term ε , representing the measurement error or the experimental fluctuation. We cannot observe this error directly. If we could, we would subtract it and get an exact information about the model function. Indirect inference is all that is available for us.

The values the random observation Y in general is different from the fit \hat{Y} . The difference

$$R_X(Y) := Y - \hat{Y}$$

To Do:
mention
unbiased

[†] it puts the hat on top of Y .

`residual` is called **residual**. The residual can be seen as an estimator for the non-observable error term `model!simple_linear!variance!residual`. Residuals do not exactly match the error terms. This would only be the case if the estimation variance!residual were `textbf{0}`. For the general case the relation

$$\begin{aligned} R_X(Y) &= Y - \hat{Y} \\ &= (I - H)Y \\ &= (I - H)(X\beta + \varepsilon) \\ &= (I - H)\varepsilon, \end{aligned} \tag{2.13} \quad \boxed{\text{eq:02-ewerr}}$$

shows that the residuals are linear combinations of the error terms. We have to infer back from these linear combinations on the error term.

If the variance of the error terms does exist, the variance matrix Σ of the error terms $Var(\varepsilon) = \Sigma$ determines the variance of the residuals:

$$\begin{aligned} Var(R_X(Y)) &= Var((I - H)\varepsilon) \\ &= (I - H)\Sigma(I - H)^\top. \end{aligned} \tag{2.14} \quad \boxed{\text{eq:02-varerr}}$$

So far we only presumed that there is no systematic error. This was formalised as assumption

$$E(\varepsilon) = 0.$$

We speak of a **simple linear model**, if we have additionally:

$$\begin{aligned} (\varepsilon_i)_{i=1,\dots,n} &\quad \text{are independent} \\ Var(\varepsilon_i) &= \sigma^2 \quad \text{for a } \sigma \text{ not depending on } i. \end{aligned}$$

For a linear model we try to estimate the parameter vector β . The variance structure of the vector of error terms introduces nuisance parameters which complicate the estimation. For a simple linear model this nuisance reduces to just one unknown nuisance parameter σ . Equations like eq:02-varerr can be simplified, because now $\Sigma = \sigma^2 I$ and the parameter σ can be pulled out from the formula. We can estimate this parameter from the residuals, because **residual variance**

$$s^2 := \frac{1}{n - Rk(X)} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 \tag{2.15}$$

is an unbiased estimator for σ^2 . We write $\widehat{\sigma^2} := s^2$. (Taking the root is no linear operation and does not preserve the expected value. The residual standard deviation $\sqrt{s^2}$ is no unbiased estimator for σ .) Plugged into equation (2.9), the residual variance estimator gives an estimator for the variance/covariance matrix of the estimator for β , because in the simple model we have

$$Var(\hat{\beta}) = \sigma^2 X^\top X \tag{2.16}$$

which can be estimated by $s^2 X^\top X$.

The standard output in example 2.1 shows only minimal information about the estimator. More information about the estimator, residuals and derived statistics are returned if we ask for a summary.

`bsp:02.summary`

Example 2.3:

| <pre>summary(lm(y ~ x))</pre> <p>Call: lm(formula = y ~ x)</p> <p>Residuals:</p> <table border="0"> <tr><td>Min</td><td>1Q</td><td>Median</td><td>3Q</td><td>Max</td></tr> <tr><td>-25.8173</td><td>-6.5451</td><td>0.0719</td><td>8.5949</td><td>26.4990</td></tr> </table> <p>Coefficients:</p> <table border="0"> <thead> <tr> <th></th> <th>Estimate</th> <th>Std. Error</th> <th>t value</th> <th>Pr(> t)</th> </tr> </thead> <tbody> <tr> <td>(Intercept)</td> <td>-4.63276</td> <td>2.32045</td> <td>-1.996</td> <td>0.0487 *</td> </tr> <tr> <td>x</td> <td>2.54799</td> <td>0.03989</td> <td>63.872</td> <td><2e-16 ***</td> </tr> </tbody> </table> <p>---</p> <p>Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1</p> <p>Residual standard error: 11.52 on 98 degrees of freedom Multiple R-squared: 0.9765, Adjusted R-squared: 0.9763 F-statistic: 4080 on 1 and 98 DF, p-value: < 2.2e-16</p> | Min | 1Q | Median | 3Q | Max | -25.8173 | -6.5451 | 0.0719 | 8.5949 | 26.4990 | | Estimate | Std. Error | t value | Pr(> t) | (Intercept) | -4.63276 | 2.32045 | -1.996 | 0.0487 * | x | 2.54799 | 0.03989 | 63.872 | <2e-16 *** | <i>Input</i> <hr/> <i>Output</i> |
|--|----------|------------|---------|------------|-----|----------|---------|--------|--------|---------|--|----------|------------|---------|----------|-------------|----------|---------|--------|----------|---|---------|---------|--------|------------|-------------------------------------|
| Min | 1Q | Median | 3Q | Max | | | | | | | | | | | | | | | | | | | | | | |
| -25.8173 | -6.5451 | 0.0719 | 8.5949 | 26.4990 | | | | | | | | | | | | | | | | | | | | | | |
| | Estimate | Std. Error | t value | Pr(> t) | | | | | | | | | | | | | | | | | | | | | | |
| (Intercept) | -4.63276 | 2.32045 | -1.996 | 0.0487 * | | | | | | | | | | | | | | | | | | | | | | |
| x | 2.54799 | 0.03989 | 63.872 | <2e-16 *** | | | | | | | | | | | | | | | | | | | | | | |

a:ch02:1

| Exercise 2.2 | |
|--------------|--|
| | <p>analysis the output of <code>lm()</code> shown in example 2.3 (page 2-8). Which of the terms can you interpret? Write down your interpretations. For which terms do you need more information?</p> <p>Generate a commented version of the output.</p> |

In section 2.3 we will present the theoretical background needed to interpret the remaining terms.

Calling `lm()` always returns a result - if it is appropriate for the data, but it will also return a linear result if the linear model is not adequate. We need additional diagnostics to tell us whether the model is reliable and usable.

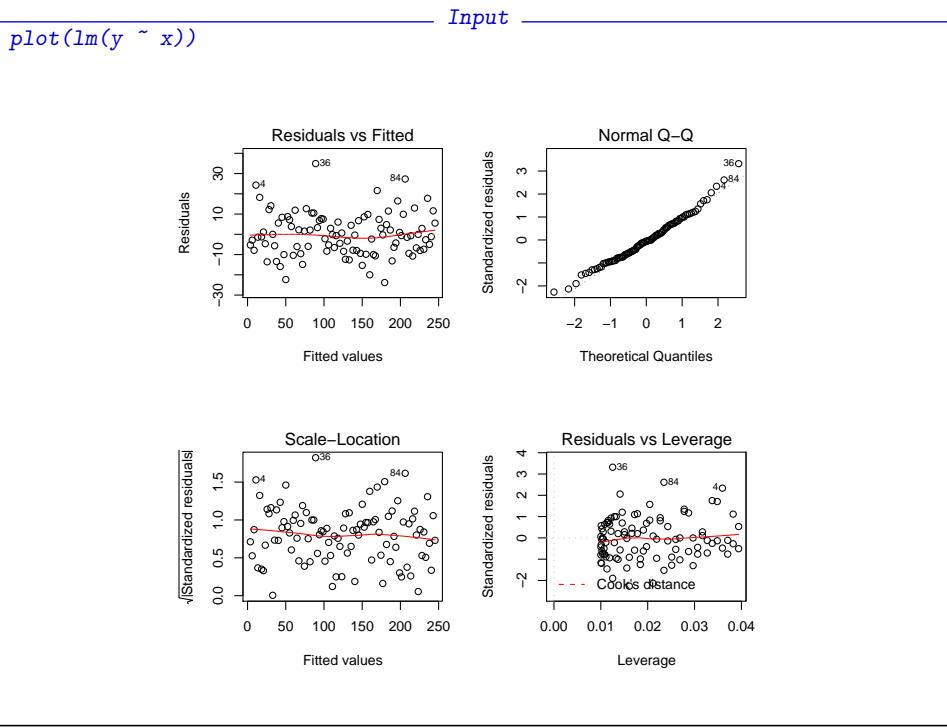
a:ch02:1a

| Exercise 2.3 | |
|--------------|--|
| | <p>Let $yy < -2.5 * x + 0.01x^2 + err$. What are the results you get if you do a regression using the (incorrect) Regression model $yy \sim x$? Do you get any hints that this model is not adequate?</p> |

The function `lm()` does not only give an estimation for the linear model, but provides a se-

ries of diagnostics that can help to judge whether the model assumptions are acceptable. A representation using `plot()` shows some aspects.

Example 2.4:



The top left plot shows the residuals against the fit. It gives a first survey.

The distribution of the fitted values depends on the design. Unless the design is homogeneous, you cannot expect the residual plot to be homogeneous.

The residuals should approximately look like scatter plot of independent variables. The distribution of the residuals should not vary with the fit. If systematic structures show up in this plot, it is a warning that the model or the model assumptions may not be satisfied.

The previous discussion allows to be more precise: the residuals should be linear combinations as in (2.13) of independent identically distributed variables. If the model assumptions are satisfied, the variance is given by (2.14).

In a one dimensional situation, a plot of the residuals against the regressor would be sufficient. For p regressors, the graphical representation becomes difficult. The plot of the residuals against the fit however generalises to higher dimensions.

If we start with distribution assumptions about the error terms, we can derive distribution properties of the estimator and of the residuals. The most powerful statements are possible, if

the error terms are independent identically distributed with a common normal distribution. In that case the plot on the upper right should look approximately like a “normal probability plot” of normal random variates, where again “approximately” means: up to transformation with the matrix $I - H$.

a:ch02:1b The two remaining plots are special diagnostics for linear models (see `help(plot.lm)`).

```
lm@lm|textbf
print.lm@print.lm
  (lm)
Topic
  re-
  gres-
  sion!lm@lm
aov@aov|textit
as.data.frame@as.data.fra
```

| | |
|---------------------|---|
| Exercise 2.4 | |
| | a:ch02:1a Use <code>plot()</code> to inspect the results of exercise 2.3. Does it give you indications that the linear model is not appropriate? Which indications? |

`plot()` provides additional diagnostic plots for linear models. These must be requested explicitly using the parameter `which`.

help(lm)

lm *Fitting Linear Models*

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
  method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
  singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

- | | |
|----------------|---|
| formula | a symbolic description of the model to be fit. The details of model specification are given below. |
| data | an optional data frame, list or environment (or object coercible by <code>as.data.frame</code> to a data frame) containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(formula)</code> , typically the environment from which <code>lm</code> is called. |
| subset | an optional vector specifying a subset of observations to be used in the fitting process. |
| weights | an optional vector of weights to be used in the fitting process. Should be <code>NULL</code> or a numeric vector. If non- <code>NULL</code> , weighted least squares is used with weights <code>weights</code> (that is, minimizing <code>sum(w*e^2)</code>); otherwise ordinary least squares is used. |

| | |
|---|--|
| <code>options@options textit</code> | a function which indicates what should happen when the data contain NAs. The default is set by the <code>na.action</code> setting of <code>options</code> , and is <code>na.fail</code> if that is unset. The “factory-fresh” default is <code>na.omit</code> . Another possible value is <code>NULL</code> , no action. Value <code>na.exclude</code> can be useful. |
| <code>na.fail@na.fail textit</code> | |
| <code>na.omit@na.omit textit</code> | |
| <code>na.exclude@na.exclude textit</code> | |
| <code>model.matrix.default@model.matrix.default textit</code> | |
| <code>offset@offset textit</code> | |
| <code>model.offset@model.offset textit</code> | the method to be used; for fitting, currently only <code>method = "qr"</code> is supported; <code>method = "model.frame"</code> returns the model frame (the same as with <code>model = TRUE</code> , see below). |
| <code>method@method textit</code> | |
| <code>offset@offset textit</code> | |
| <code>model.matrix@model.matrix textit</code> | |
| <code>aov@aov textit</code> | logicals. If <code>TRUE</code> the corresponding components of the fit (the model frame, the model matrix, the response, the QR decomposition) are returned. |
| <code>model, x, y, qr@formula textit</code> | |
| <code>lm.fit@lm.fit textit</code> | |
| <code>class@class textit</code> | |
| <code>singular.ok</code> | logical. If <code>FALSE</code> (the default in S but not in R) a singular fit is an error. |
| <code>contrasts</code> | an optional list. See the <code>contrasts.arg</code> of <code>model.matrix.default</code> . |
| <code>offset</code> | this can be used to specify an <i>a priori</i> known component to be included in the linear predictor during fitting. This should be <code>NULL</code> or a numeric vector of length either one or equal to the number of cases. One or more <code>offset</code> terms can be included in the formula instead or as well, and if both are specified their sum is used. See <code>model.offset</code> . |
| <code>...</code> | additional arguments to be passed to the low level regression fitting functions (see below). |

Details

Models for `lm` are specified symbolically. A typical model has the form `response ~ terms` where `response` is the (numeric) response vector and `terms` is a series of terms which specifies a linear predictor for `response`. A `terms` specification of the form `first + second` indicates all the terms in `first` together with all the terms in `second` with duplicates removed. A specification of the form `first:second` indicates the set of terms obtained by taking the interactions of all terms in `first` with all terms in `second`. The specification `first*second` indicates the *cross* of `first` and `second`. This is the same as `first + second + first:second`.

If the formula includes an `offset`, this is evaluated and subtracted from the response.

If `response` is a matrix a linear model is fitted separately by least-squares to each column of the matrix.

See `model.matrix` for some further details. The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a `terms` object as the formula (see `aov` and `demo(glm.vr)` for an example).

A formula has an implied intercept term. To remove this use either `y ~ x - 1` or `y ~ 0 + x`. See `formula` for more details of allowed formulae.

`lm` calls the lower level functions `lm.fit`, etc, see below, for the actual numerical computations. For programming only, you may consider doing likewise.

All of `weights`, `subset` and `offset` are evaluated in the same way as variables in `formula`, that is first in `data` and then in the environment of `formula`.

Value

`lm` returns an object of class `"lm"` or for multiple responses of class `c("mlm", "lm")`.

The functions `summary` and `anova` are used to obtain and print a summary and analysis of variance table of the results. The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` extract various useful features of the value returned by `lm`.

An object of class "lm" is a list containing at least the following components:

| | |
|----------------------------|--|
| <code>coefficients</code> | a named vector of coefficients |
| <code>residuals</code> | the residuals, that is response minus fitted values. |
| <code>fitted.values</code> | the fitted mean values. |
| <code>rank</code> | the numeric rank of the fitted linear model. |
| <code>weights</code> | (only for weighted fits) the specified weights. |
| <code>df.residual</code> | the residual degrees of freedom. |
| <code>call</code> | the matched call. |
| <code>terms</code> | the <code>terms</code> object used. |
| <code>contrasts</code> | (only where relevant) the contrasts used. |
| <code>xlevels</code> | (only where relevant) a record of the levels of the factors used in fitting. |
| <code>offset</code> | the offset used (missing if none were used). |
| <code>y</code> | if requested, the response used. |
| <code>x</code> | if requested, the model matrix used. |
| <code>model</code> | if requested (the default), the model frame used. |

In addition, non-null fits will have components `assign`, `effects` and (unless not requested) `qr` relating to the linear fit, for use by extractor functions such as `summary` and `effects`.

Using time series

Considerable care is needed when using `lm` with time series.

Unless `na.action = NULL`, the time series attributes are stripped from the variables before the regression is done. (This is necessary as omitting NAs would invalidate the time series attributes, and if NAs are omitted in the middle of the series the result would no longer be a regular time series.)

Even if the time series attributes are retained, they are not used to line up series, so that the time shift of a lagged or differenced regressor would be ignored. It is good practice to prepare a `data` argument by `ts.intersect(..., dframe = TRUE)`, then apply a suitable `na.action` to that data frame and call `lm` with `na.action = NULL` so that residuals and fitted values are time series.

Note

Offsets specified by `offset` will not be included in predictions by `predict.lm`, whereas those specified by an offset term in the formula will be.

Author(s)

The design was inspired by the S function of the same name described in Chambers (1992). The implementation of model formula by Ross Ihaka was based on Wilkinson & Rogers (1973).

```
anova@anova|textit
terms@terms|textit
effects@effects|textit
ts.intersect@ts.intersect
predict.lm@predict.lm|tex
```

```

summary.lm@summary.lm|textit
anova.lm@anova.lm|textit
aov@aov|textit
coef@coef|textit Chambers, J. M. (1992) Linear models. Chapter 4 of Statistical Models in S eds J. M.
effects@effects|textit Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.
residuals@residuals|textit Wilkinson, G. N. and Rogers, C. E. (1973) Symbolic descriptions of factorial models for
vcov@vcov|textit analysis of variance. Applied Statistics, 22, 392–9.
predict.lm@predict.lm|textit
predict@predict|textit
confint@confint|textit
lm.influence@lm.influence|textit
glm@glm|textit
lm.fit@lm.fit|textit summary.lm for summaries and anova.lm for the ANOVA table; aov for a different interface.
lm.wfit@lm.wfit|textit
The generic functions coef, effects, residuals, fitted, vcov.
predict.lm (via predict) for prediction, including confidence and prediction intervals;
confint for confidence intervals of parameters.
lm.influence for regression diagnostics, and glm for generalized linear models.
The underlying low level functions, lm.fit for plain, and lm.wfit for weighted
regression fitting.

```

Examples

```

## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2,10,20, labels=c("Ctl","Trt"))
weight <- c(ctl, trt)
anova(lm.D9 <- lm(weight ~ group))
summary(lm.D90 <- lm(weight ~ group - 1))# omitting intercept
summary(resid(lm.D9) - resid(lm.D90)) #- residuals almost identical

opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
plot(lm.D9, las = 1)      # Residuals, Fitted, ...
par(opar)

## model frame :
stopifnot(identical(lm(weight ~ group, method = "model.frame"),
model.frame(lm.D9)))

```

To be added to the help information: in the formula notation, with *first-second* includes the variables indicated by the first term, but excludes those indicated by second. For more information on the formula notation, see *help(formula)*. A summary is given in appendix A.53 (Seite A-78).

The hat matrix is a particularity of linear models. Fit and residuum however are general concepts and can be applied for all kind of estimations. Clients are often satisfied seeing a fit (or the estimation). For serious clients, and for statisticians, the residuals often contain more valuable information. They indicate what is not yet covered by the model or the estimation.

2.2.3 More Examples for Linear Models

```
design
matrix|textbf
```

The matrix X is called the **design matrix** of the model. It can be the matrix of row vectors x_i representing the measurement conditions for experiment i . But the design matrix is not restricted to this special case. The seemingly simple class of linear models comprises many import special cases. For example:

Simple Linear Regression:

$$y_i = a + b x_i + \varepsilon_i \quad \text{with } x_i \in \mathbb{R}, a, b \in \mathbb{R}$$

can be written as a linear model

$$X = (1 \ x)$$

here $1 = (1, \dots, 1)^\top \in \mathbb{R}^n$.

Polynomial Regression:

$$y_i = a + b_1 x_i + b_2 x_i^2 + \dots + b_k x_i^k + \varepsilon_i \quad \text{with } x_i \in \mathbb{R}, a, b_j \in \mathbb{R}$$

can be written as a linear model

$$X = (1 \ x \ x^2 \ \dots \ x^k)$$

where $x^j = (x_1^j \dots x_n^j)^\top$.

In analogy, a multitude a models can be written as linear model, using other transformations.

Analysis of Variance: One Way Layout

Observations are taken under m experimental conditions, with n_j observations are taken under condition $j, j = 1, \dots, m$. The measurement is the sum of a baseline effect μ , a contribution α_j that is specific for condition j , and a measurement error

$$y_{ij} = \mu + \alpha_j + \varepsilon_{ij} \quad \text{with } \mu, \alpha_j \in \mathbb{R}, j = 1, \dots, n_i.$$

This can be written as a linear model With $n = \sum n_j$ and

$$X = (1 \ I_1 \ \dots \ I_m),$$

where I_j is an indicator variable marking group j .[‡]

Analysis of Covariance

Like for the analysis of variance, differences between treatment groups are analysed. But in addition to the treatment, additional influence factors are considered and must be adjusted for. Under experimental condition j , the observation i in this group depends on additional influence factors x_{ij} for observation ij .

$$y_{ij} = \mu + \alpha_j + b x_{ij} + \varepsilon_{ij} \quad \text{with } \mu, \alpha_j \in \mathbb{R}.$$

[‡] For analysis of variance, the convention is to use the first index to mark the group, and the last index to count the observations per group. Index names are given in alphabetical sequence. Following this convention, in comparison to our notation the roles of i and j are swapped.

2.2.4 Model Formulae

R allows to specify models by giving the rules for building the design matrix. The syntax for writing these rules is sketched in the description of `lm()`. We will have a closer look at this formula notation. This kind of model specification does not only apply to linear models, but can also be used for more general non linear models. The model specification is stored as an attribute with attribute name `formula`. It can be manipulated using `formula()`.

Examples

| | |
|--|--|
| $y \sim 1 + x$ | corresponds to $y_i = (1 \ x_i)(\beta_1 \ \beta_2)^\top + \varepsilon$ |
| $y \sim x$ | short for $y \sim 1+x$ (a constant term is assumed implicitly) |
| $y \sim 0 + x$ | corresponds to $y_i = x_i\beta + \varepsilon$ |
| $\log(y) \sim x_1 + x_2$ | corresponds to $\log(y_i) = (1 \ x_{i1} \ x_{i2})(\beta_1 \ \beta_2 \ \beta_3)^\top + \varepsilon$ (a constant term is assumed implicitly) |
| <code>lm(y ~ poly(x, 4), data = Experiment)</code> | analyses the data set "Experiment" with a linear model for polynomial regression of degree 4 in x . |

There are important special cases for factorial designs:

| | |
|----------------|---|
| $y \sim A$ | one way analysis of variance with factor A , |
| $y \sim A + x$ | analysis of covariance with factor A and regression co-variable x , |
| $y \sim A + B$ | two factor crossed design with factors A and B without interaction, |
| $y \sim A * B$ | two factor crossed design with factors A and B and all interactions (combinations of the levels of A and B), |
| $y \sim A/B$ | two factor hierarchical layout with factor A and subfactor B . |

a:ch02:2 A table of all operators for model specification formulae is in appendix [A.53](#) (Seite [A-78](#)).

| Exercise 2.5 | |
|--------------|--|
| | Write the four models from section 2.2.3 using the R formula notation. <small>subs:02-BspLinMod</small> |
| | For each of these models, generate an example data set by simulation, and apply <code>lm()</code> to the example. Compare the estimators returned by <code>lm()</code> with the parameters you have used in the simulations. |

The model formula used by `lm()` is returned as entry in the function result and recovered from

there. From the formula, R implicitly generates a design matrix. Function `model.matrix()` can be used to inspect design matrix.

`model.matrix@model.matrix`

a:ch02:3

| Exercise 2.6 | |
|--------------|---|
| | <p>Generate three vectors of random variables with a $N(\mu_j, 1)$ distribution, $\mu_j = j$, $j = 1, 3, 9$, each of length 10, and combine these into a vector y.</p> <p>Generate a vector x with the values j, $j = 1, 3, 9$, each repeated 10 times.</p> <p>Calculate the Gauß-Markov estimator in the linear models</p> $y \sim x \text{ and } y \sim \text{factor}(x).$ <p>Inspect the results as a table using <code>summary()</code> and graphically using <code>plot()</code>. Compare the results, and give a written report.</p> |

2.2.5 Gauss-Markov Estimator and Residuals

We have a closer look on the Gauß-Markov estimator. Knowledge from linear Algebra, long thinking or other sources tell us:

rem:2.1

Remark 2.3

- (1) The design matrix X defines a mapping $\mathbb{R}^p \rightarrow \mathbb{R}^n$ with $\beta \mapsto X\beta$.
Let \mathcal{M}_X , $\mathcal{M}_X \subset \mathbb{R}^n$ be the image space of this mapping. \mathcal{M}_X is the vector space generated by the column vectors from X .
- (2) If the model assumptions are satisfied, $E(Y) \in \mathcal{M}_X$.
- (3) $\hat{Y} = \pi_{\mathcal{M}_X}(Y)$, where $\pi_{\mathcal{M}_X} : \mathbb{R}^n \rightarrow \mathcal{M}_X$ is the (Euclidean) orthogonal projection is.
- (4) $\hat{\beta} = \arg \min_{\beta} |Y - \hat{Y}_{\beta}|^2$ where $\hat{Y}_{\beta} = X\beta$.

The characterisation (3) of the Gauß-Markov estimator as an orthogonal projection often helps understanding. The fit is the orthogonal projection of the observation vector on the space of expected values of the model (and hence minimises the quadratic distance). The vector of residuals is the orthogonal complement.

In statistics, the estimator is analyses systematically, and the characterisation given above is just one starting point. Some properties of the estimator can be easily derived using knowledge from probability theory, such as the following lemma:

th:S02.1

Theorem 2.4 Let Z be a random variable with values in \mathbb{R}^n , with $N(0, \sigma^2 I_{n \times n})$ distribution, and let $\mathbb{R}^n = L_0 \oplus \dots \oplus L_r$ be an orthogonal decomposition. Let $\pi_i = \pi_{L_i}$ be the orthogonal projection onto L_i , $i = 0, \dots, r$. Then the following holds:

- (i) $\pi_0(Z), \dots, \pi_r(Z)$ are independent random variables with normal distributions.
- (ii) $\frac{|\pi_i(Z)|^2}{\sigma^2} \sim \chi^2(\dim L_i)$ for $i = 0, \dots, r$.

Proof. → probability theory. See z.B. [Jørgensen 1993, 2.5 Theorem 3]. \square

Using $\varepsilon = Y - X\beta$ this allows to derive the theoretical distributions for the estimator $\hat{\beta}$ and the residuals $Y - \hat{Y}$.

In particular, for simple linear models, the residual variance can be used to calculate the variance (resp. standard deviation) for each component $\hat{\beta}_k$. The corresponding t statistics and the p -value for the test of the hypothesis $\hat{\beta}_k = 0$ are given in the output of `summary()`.

ToDo:

studentised
residuals,
pointwise
confidence
intervals

ToDo:
std error,
t-test,
point-wise
confidence

| Exercise 2.7 | What is the distribution of $ R_X(Y) ^2 = Y - \hat{Y} ^2$, if ε has a $N(0, \sigma^2 I)$ distribution? |
|--------------|--|
|--------------|--|

At first sight $|R_X(Y)|^2 = |Y - \hat{Y}|^2$, seems an appropriate gauge to judge the quality of a model: small values indicate a good fit, large values indicate a poor fit. However this has to be taken with caution. On one hand this value depends on linear scale factors. On the other hand, the dimension of the spaces involved has to be taken into account.

ToDo:

zum
einen..zum
anderen

What happens if additional regressors are taken into the model? We have already seen, that “linear” includes the possibility to model non-linear relations, for example by taking transformed variables into the design matrix. The characterisation (3) in remark [Rem:2.1](#) tells us, that effectively only the vector space spanned by the design matrix is relevant. Here we can see limits for the Gauß-Markov estimator in linear models: if many transformed variables are taken into the model, or generally if the image space determined by the design matrix becomes too large, an over-fitting will result. In the extreme we may get $\hat{Y} = Y$. So all residuals are zero, but the estimation is not useful.

We use $|R_X(Y)|^2 / \dim(L_X)$, where L_X is the orthogonal complement of \mathcal{M}_X in \mathbb{R}^n (so $\dim(L_X) = n - \dim(\mathcal{M}_X)$) to compensate for the number of dimensions.

| Exercise 2.8 | |
|--------------|---|
| | Modify the output of <code>plot.lm()</code> for the linear model so that instead of the Tukey-Anscombe plot the studentised residuals are plotted against the fit. |
| * | Enhance the <i>QQ</i> -Plot by Monte Carlo bands for independent normal errors. <i>Hint:</i> You cannot generate the bands directly from a normal distribution - you need the distribution of the residuals, not the distribution of the errors. |

ToDo:

Check
revise
Tukey-
Anscombe

| Exercise 2.9 | |
|--------------|---|
| | <p>Write a procedure that calculates the Gauß-Markov estimator for the simple linear regression $y_i = a + bx_i + \varepsilon_i$ with $x_i \in \mathbb{R}, a, b \in \mathbb{R}$</p> <p>and shows four plots:</p> <ul style="list-style-type: none"> • response against regressor, with estimated straight line • studentised residuals against Fit • distribution function of the studentised residuals in a <i>QQ</i> plot with confidence bands • histogram of the studentised residuals |

2.3 Variance Decomposition and Analysis of Variance

s:2.2

If a simple linear model with normal errors applies, the t tests are useful to solve one dimensional problems (tests or confidence intervals for single parameters, point-wise confidence intervals). To solve simultaneous or higher dimensional problems, we need other tools. Instead of differences or mean values which are forming the base of t tests, we use norm based distances (as, for example quadratic distances) which generalise to higher dimensions.

The interpretation of the Gauß-Markov estimator as orthogonal projection (Remark 2.3.3) shows a possibility to compare models: for X, X' design matrices with $\mathcal{M}_{X'} \subset \mathcal{M}_X$, we consider the decomposition $\mathbb{R}^n = L_0 \oplus \dots \oplus L_r$ with $L_0 := \mathcal{M}_{X'}$, and the orthogonal complements $L_1 := \mathcal{M}_X \ominus \mathcal{M}_{X'}, L_2 := \mathbb{R}^n \ominus \mathcal{M}_X$. As above π denotes the projection.

$$F := \frac{\frac{1}{\dim(L_1)} |\pi_{\mathcal{M}_X} Y - \pi_{\mathcal{M}_{X'}} Y|^2}{\frac{1}{\dim(L_2)} |Y - \pi_{\mathcal{M}_X} Y|^2}.$$

This statistic, the F statistic (in honour of R.A. Fisher) is the basis for the *analysis of variance*, a classical strategy to compare models.

The idea generalises to chains of models. Let $\mathcal{M}_0 \subset \dots \subset \mathcal{M}_r = \mathbb{R}^n$ be a chain of vector spaces, and $L_0 := \mathcal{M}_0, L_i := \mathcal{M}_i \ominus \mathcal{M}_{i-1}$ for $i = 1, \dots, r$ the stepwise orthogonal complements. With the notation as above,

$$\frac{\frac{1}{\dim(L_{i-1})} |\pi_{\mathcal{M}_i} Y - \pi_{\mathcal{M}_{i-1}} Y|^2}{\frac{1}{\dim(L_i)} |Y - \pi_{\mathcal{M}_i} Y|^2}$$

is a test statistic which can be used as a test for the model \mathcal{M}_{i-1} in comparison to the refined \mathcal{M}_i .

a:ch02:11

| Exercise 2.10 | |
|---------------|---|
| | What is distribution of F , if $E(Y) \in \mathcal{M}_{X'}$ applies and ε is distributed as $N(0, \sigma^2 I)$? |

a:ch02:12

| Exercise 2.11 | |
|---------------|--|
| | <p>Give an explicit formula for the F statistics for analysis of variance in the one way layout</p> $y_{ij} = \mu + \alpha_j + \varepsilon_{ij}$ <p>in comparison to the homogeneous model</p> $y_{ij} = \mu + \varepsilon_{ij}.$ |

The analysis of variance gives another representation and interpretation of linear models. For comparison with example 2.3, For example comparison with 2.3, in the analysis of variance representation is

bsp:02.aovsummary

Example 2.5:

| Input | Output | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------------|--|---------|------------|---------------|---------|--------|---|--------|--------|--------|---------------|-----------|----|-------|-----|--|-----|--|--|--|--|----------------|---|-------|------------|----------|--|----------|---------|---|--|
| <code>summary(aov(lmres))</code> | <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">Df</th> <th style="width: 10%;">Sum Sq</th> <th style="width: 10%;">Mean Sq</th> <th style="width: 10%;">F value</th> <th style="width: 10%;">Pr(>F)</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>540969</td> <td>540969</td> <td>4079.6</td> <td>< 2.2e-16 ***</td> </tr> <tr> <td>Residuals</td> <td>98</td> <td>12995</td> <td>133</td> <td></td> </tr> <tr> <td>---</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Signif. codes:</td> <td>0</td> <td>'***'</td> <td>0.001 '**'</td> <td>0.01 '*'</td> </tr> <tr> <td></td> <td>0.05 '.'</td> <td>0.1 ' '</td> <td>1</td> <td></td> </tr> </tbody> </table> | Df | Sum Sq | Mean Sq | F value | Pr(>F) | 1 | 540969 | 540969 | 4079.6 | < 2.2e-16 *** | Residuals | 98 | 12995 | 133 | | --- | | | | | Signif. codes: | 0 | '***' | 0.001 '**' | 0.01 '*' | | 0.05 '.' | 0.1 ' ' | 1 | |
| Df | Sum Sq | Mean Sq | F value | Pr(>F) | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 540969 | 540969 | 4079.6 | < 2.2e-16 *** | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Residuals | 98 | 12995 | 133 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| --- | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Signif. codes: | 0 | '***' | 0.001 '**' | 0.01 '*' | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | 0.05 '.' | 0.1 ' ' | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

a:ch02:13

| Exercise 2.12 | |
|---------------|--|
| | Analyse the output of <code>lm()</code> shown in example 2.3 (page 2-8). Which terms can you interpret now? Give a written report. For which terms do you need more information? |

One more item you find in the output is labelled “R-squared”. The term which is given here is an estimator for the fraction of $Var(Y)$ which is explained by the model:

$$R^2 = \frac{mss}{mss + rss}$$

with $mss := \frac{1}{n} \sum (\hat{Y}_i - \bar{\hat{Y}})^2$ and $rss := \frac{1}{n} \sum (R_X(Y)_i - \bar{R_X(Y)})^2$. The notation R^2 stems from simple linear regression. In that text, conventionally the correlation $Cor(X, Y)$ is denoted by R , and $R^2 = Cor(X, Y)^2$. R^2 does not take into account the number of the estimated parameters. As a consequence, it can easily be too optimistic. The term “adjusted R-squared” is re-weighted to take into account the degrees of freedom.

private note > > >

```

ans$df <- c(p, rdf, NCOL(Qr$qr))
if (p != attr(z$terms, "intercept")) {
  df.int <- if (attr(z$terms, "intercept"))
    1
  else 0
  ans$r.squared <- mss/(mss + rss)
  ans$adj.r.squared <- 1 - (1 - ans$r.squared) * ((n -
    df.int)/rdf)
  ans$fstatistic <- c(value = (mss/(p - df.int))/resvar,
    numdf = p - df.int, dendf = rdf)
}
else ans$r.squared <- ans$adj.r.squared <- 0
ans$cov.unscaled <- R
dimnames(ans$cov.unscaled) <- dimnames(ans$coefficients)[c(1,
  1)]
if (correlation) {
  ans$correlation <- (R * resvar)/outer(se, se)
  dimnames(ans$correlation) <- dimnames(ans$cov.unscaled)
  ans$symbolic.cor <- symbolic.cor
}

```

<<< private note

[help\(anova\)](#)

[anova](#)

Anova Tables

Description

Compute analysis of variance (or deviance) tables for one or more fitted model objects.

Usage

```
anova(object, ...)
```

Arguments

- | | |
|---------------|---|
| object | an object containing the results returned by a model fitting function (e.g., lm or glm). |
| ... | additional objects of the same type. |

```

coefficients@coefficients|textit
Value
effects@effects|textit
fitted.values@fitted.values|textit
This (generic) function returns an object of class anova. These objects represent analysis-
residuals@residuals|textit
of variance and analysis-of-deviance tables. When given a single argument it produces a
summary@summary|textit
drop1@drop1|textit
table which tests whether the model terms are significant.
add1@add1|textit
When given a sequence of objects, anova tests the models against one another in the order
contrast|textbf
specified.
lm@lm|textit
aov@aov|textit
The print method for anova objects prints tables in a “pretty” form.

```

Warning

The comparison between two or more models will only be valid if they are fitted to the same dataset. This may be a problem if there are missing values and R's default of **na.action = na.omit** is used.

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*, Wadsworth & Brooks/Cole.

See Also

coefficients, effects, fitted.values, residuals, summary, drop1, add1.

Models for the analysis of variance can be specified as descriptive rules. The syntax used is the same we have used for regression. If terms on the right hand side of the model description are factors, **lm()** will automatically use a variance analysis model instead of a regression model.

ToDo:
clean up
discussion
of factors

The model description determines the linear spaces that contain the expected values for the model and its sub-models. The analysis of variance is not determined uniquely by the model description. The spaces may allow for various orthogonal decompositions (for example, depending on the order in which sub.models are considered). Moreover the specification of the factors for a design determines a system of generators for the spaces. The factors need not be orthogonal, not even independent .

This holds for all linear models. In regression, dependency is more an exception. With factorial designs, dependency is a frequent phenomenon. One way analysis, written in co-ordinates, may illustrate this problem: with

$$y_{ij} = \mu + \alpha_j + \varepsilon_{ij} \quad \text{with } \mu, \alpha_j \in \mathbb{R}$$

ToDo:
discuss
choice of
models
- model
dependent
estimation
[a:ch02:14]
ToDo:
check
contrasts

for $n_j > 0$ the decomposition into μ and α_j is not unique. The underlying reason is: the global factor μ defines the vector space spanned by the constant vector 1, and this is contained in the space spanned by the group indicators.

The model formula defines a design matrix X and hence a model space. An additional matrix C is used to extend the matrix and to specify a variance decomposition uniquely. The effective design matrix is then $[1 \ X \ C]$; C is called the **contrast matrix**. The functions for analysis of variance like **lm()** or **aov()** allow for an explicit specification of contrasts.

The function **anova()** operates like a special formatting of the output and is used analogous to **summary()**, for example using the form **anova(lm())**.

| Exercise 2.13 | <p>The file “micronuclei” contains a data set from a mutagenicity test. Cell cultures (50 units each) have been observed in a control group and under 5 chemical treatments. The effect of the substances is to break up the chromosomes and to build micronuclei. Recorded is the size of the micronuclei (relative to the parent nucleus).</p> <p>Read the file “micronuclei” and calculate mean and variance for each group.</p> <p>Hint: You can read the file with <code>data()</code>. For files with table format there is special function <code>read.table()</code>. Use <code>help()</code> to get information about both functions.</p> <p>Some selected statistical functions (such as <code>mean</code>) are in table A.32 in the appendix. § A.12</p> <p>Compare the results between the groups. Are treatment effects detectable?</p> <p>Hint: try to formulate the exercise as a one way analysis of variance. You must reshape the data set first, for example with the help of <code>c()</code>.</p> |
|---------------|---|
|---------------|---|

a:ch02:15

ToCheck:
micronu-
lei still
available?

~~d~~
~~e~~ **ToDo:** ref
appendix:
reshape
()

| Exercise 2.14 | |
|----------------------|---|
| * | Write a function <code>oneway()</code> , the takes a data table as an argument and performs a one way analysis of variance as a test on difference between the columns. |
| * | Enhance <code>oneway()</code> by adding the necessary diagnostic plots. Which diagnostics are necessary? |

a:ch02:16kiwi

| Exercise 2.15 | <p>The industrial enterprise Kiwi Inc.[§] wants to develop a new helicopter for the market. The helicopter design is rated by the time it stays in air before it touches ground[¶] from a fixed starting height (ca. 2m). A design drawing is in figure 2.1, Seite 2-25. What are the factors that can affect the variability the flight(sink) time? What are the factors that can affect the mean flight duration?</p> <p>Fig:KiwiHopp_KiwiHopp</p> <p>Perform 30 test flight with a prototype by and measure the time in 1/100s. (You will have to co-operate in pairs to carry out the measurements.) Would you consider the recorded times as normally distributed? The requirement is that the mean flight duration reaches at least 2.4s. Does the prototype satisfy the requirement?</p> <p>Your task is to select a design for production. The variants under discussion are:</p> <p>(cont.)→</p> |
|---------------|---|
| | |
| | |

[§] Following an idea of Alan Lee, Univ. Auckland, New Zealand

¶ Kiwis can not fly.

| Exercise 2.15 | (cont.) |
|---------------|--|
| | <p>rotor width 45mm rotor width 35mm rotor width 45mm with an additional fold for stabilisation rotor width 35mm with an additional fold for stabilisation.</p> <p>Your budget allows for about 40 test flights. (If you need more test flights, you should give good arguments for this.) Build 4 prototypes, perform the test flights and record the times. Find the design that achieves maximum flight duration. Generate a report. The report should contain the following details:</p> <ul style="list-style-type: none"> • a list of the observed data and a description of the experimental procedure. • suitable plots of the data for each of the designs • an analysis of variance • a clear summary of your conclusions. <p><i>Additional hints:</i> Randomise the sequence of your experiments. Reduce the variation by providing uniform conditions for the experiment (same height, same launch technique etc.).</p> |
| | The fold will result in additional production cost. Give an estimate of the gain which can be achieved by this additional cost. |

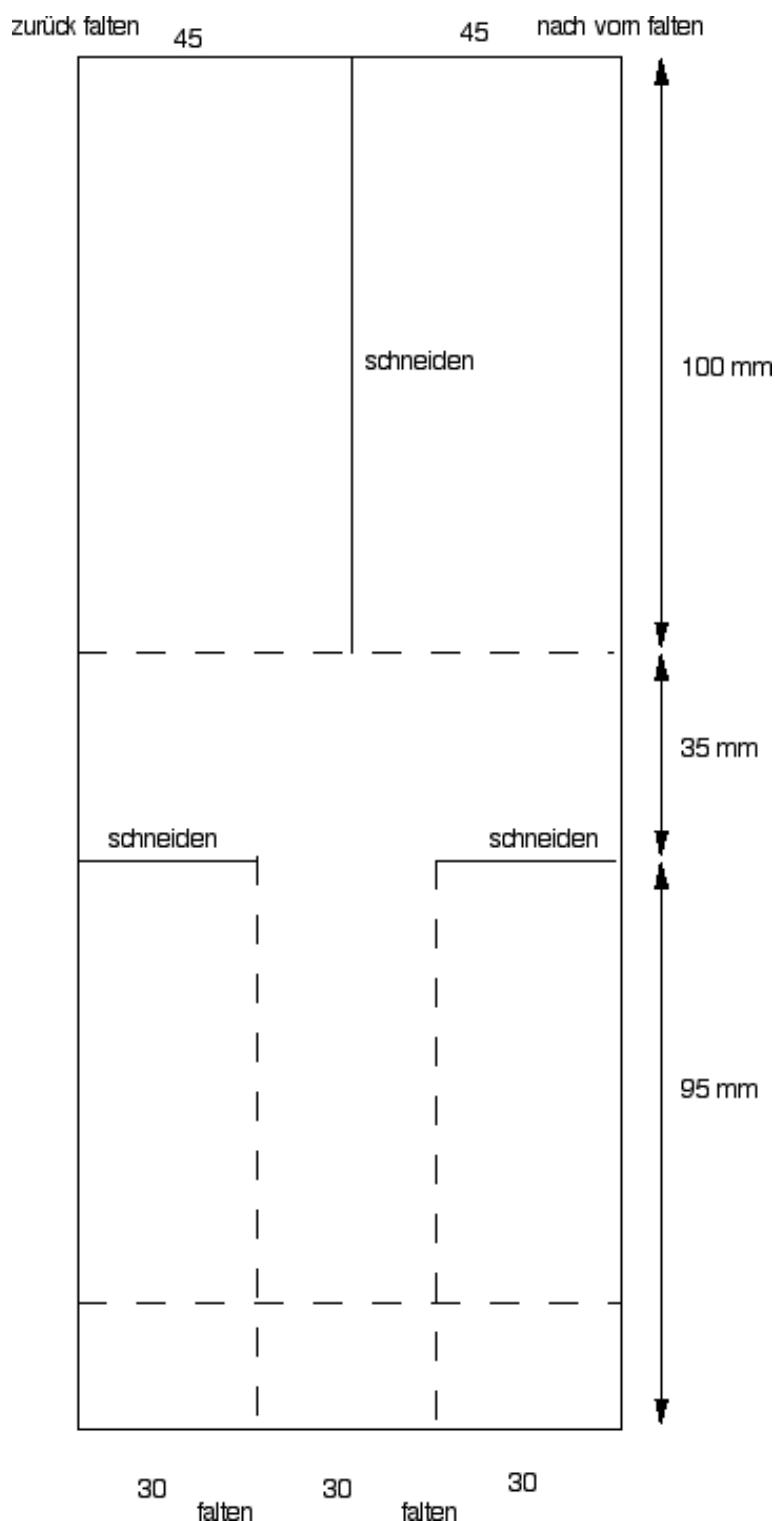
Figure 2.1 *KiwiHopp*

fig:KiwiHopp

```
predict@predict{textit
2.4 Simultaneous Inference
predict@predict{textit
predict.lm@predict.lm{textit
predict@predict{textit Scheffé's Confidence Bands
```

In principle, the least squares estimator estimates all components of the parameter vector simultaneously. On the other hand, the optimality statements in the Gauß-Markov theorem refer only to one dimensional linear statistics. But multivariate confidence statements are possible. The confidence set for confidence level $1 - \alpha$ derived from the F distribution has the form

$$\{\hat{\beta} \in \mathbb{R}^k : (\sum_{j=1}^k (\hat{\beta}_j - \beta_j)^2 \|x_j\|^2 / k) / \hat{\sigma}^2 \leq F_{1-\alpha}(k, n-k)\},$$

that the confidence set is an ellipsoid. Instead of thinking of a quadratic form, we can see the ellipse as the arc delimited by all tangents of the ellipsoid. This translates the (single) quadratic condition on the points in the confidence set into (infinitely many) linear conditions. This geometric relation is the core for the following theorem:

Theorem 2.5 Let $\mathcal{L} \subset \mathbb{R}^k$ be a linear subspace of dimension d ; $EY = Xb$ with $\text{rk}(X) = p < n$. Then

$$P\{\ell^t \beta \in \ell^t \hat{\beta} \pm (dF_{d,n-\alpha}^\alpha)^{1/2} s(\ell^t (X^t X)^{-1} \ell)^{1/2} \forall \ell \in \mathcal{L}\} = (1 - \alpha).$$

Proof. [12] 2.2, p. 48 \square

This is a simultaneous confidence set for all linear combinations from \mathcal{L} . Translated into a test, this result gives a simultaneous test for all linear Hypothesis from \mathcal{L} . In the special case $d = 1$ this simultaneous Scheffé test reduces to the usual F test. In general, it is not possible to evaluate several tests on the same data material without deflating the confidence level. The F Test is an exception. After a global F test it is possible to test these linear combinations or contrasts individually, without violating the level of the test.

In the case of simple linear regression, we get a confidence ellipsoid in the parameter space, that is the space with co-ordinates *intercept, slope*. If we look at the lines or surfaces defined by parameter values in this confidence ellipsoid, we get a hyperboloid as confidence set for the regression in the regressor/response space.

In the regressor/response space, that is the space of experimental conditions and observations, the interest is often not in a confidence set for the regression, but in a prognoses region (or tolerance region) for observations to come. To get a tolerance area, the stochastic variation by the error term affecting additional observations must be added to the estimation uncertainty. So the tolerance area is increased in comparison to the estimation hyperboloid for the regression. Confidence sets for the regression and tolerance regions for observation can be calculated using function `predict()`. The following figure shows both regions. The function `predict()` is a generic function. For linear models, it calls `predict.lm()`. `predict()` allows to specify new supporting points where the fit is calculated using the estimated model parameters. Variables are matched here by name. `newdata` has to be a data frame `data.frame` with component names corresponding to those of the original variables.

ToDo:
improve
Scheffe

We prepare a data set as an example.

Input

```
n <- 100
sigma <- 1
```

```
x <- (1:n)/n - 0.5
err <- rnorm(n)
y <- 2.5 * x + sigma * err
lmxy <- lm(y ~ x)
```

To get better control of the graphics, we calculate the plot limits and supporting points first.

ToDo:
check plot

Input

```
plotlim <- function(x) {
  xlim <- range(x)
  del <- xlim[2] - xlim[1]
  if (del > 0)
    xlim <- xlim + c(-0.1 * del, 0.1 * del)
  else xlim <- xlim + c(-0.1, 0.1)
  return(xlim)
}
xlim <- plotlim(x)
ylim <- plotlim(y)
newx <- data.frame(x = seq(xlim[1], xlim[2], 1/(2 *
  n)))
```

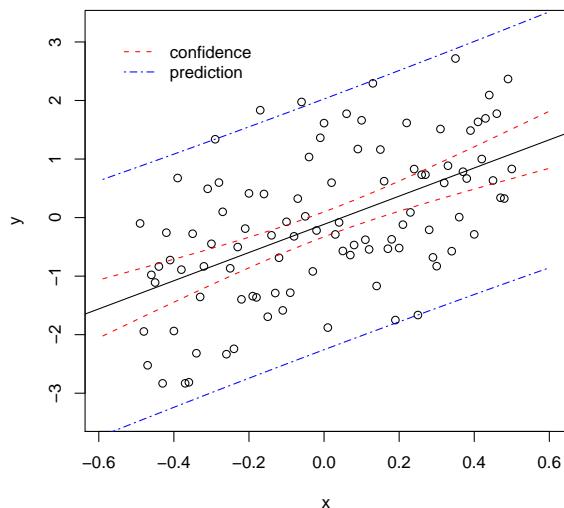
expl:02-lmkonf For these nodes, we calculate the confidence bands and draw them.

contrast|textbf

Example 2.6:

```
Input
plot(x, y, xlim = xlim, ylim = ylim)
abline(lmxy)
pred.w.plim <- predict(lmxy, newdata = newx, interval = "prediction")
pred.w.clim <- predict(lmxy, newdata = newx, interval = "confidence")
matplot(newx$x, cbind(pred.w.clim[, -1], pred.w.plim[, -1]),
       lty = c(2, 2, 6, 6), col = c(2, 2, 4, 4),
       type = "l", add = TRUE)
title(main = "Simultaneous Confidence")
legend("topleft", lty = c(2, 6), legend = c("confidence",
                                             "prediction"), col = c(2, 4), inset = 0.05, bty = "n")
```

Simultaneous Confidence

**ToDo:**

Scheffe:

fix low-
er/upper
plotting
bound

2.4.2 Tukey's Confidence Intervals**ToDo:**

Comment
on Tukey
vs Schafte

Geometrically the confidence ellipsoid is delimited by its (infinitely many) tangential spaces. Translated into tests this means that infinitely many linear tests are performed simultaneously. In many applications it is possible to formulate more specific questions. For example, to compare k treatments with treatment effects β_i , $i = 1, \dots, k$ we may be interested only in the hypotheses $\beta_i - \beta_{i_0} = 0$. These reduced question can be formulated in the framework of linear models. When specified, they can lead to more powerful tests than the omnibus simultaneous tests. Families of hypotheses of this kind can again be specified as **contrasts**. R supports the use of contrasts for the analysis of variance.

Case Study: Titre Plates

A typical tool in biology and medicine are titre plates which are used for example in cell culture experiments. The plate has small wells in a rectangular grid. Some substrate can be applied to the plate as a whole. Then on each well, a test substance can be applied. Typically, this is done using multi pipettes for a row or a column at once (figure 2.2).



Figure 2.2 *Titre Plates*. With multi pipettes, substances can be applied for a row or a column at once.

fig:microtitre

The experiments are often done in series. We will just use one plate from a series as our data example.

Input

```
p35 <- read.delim("../data/p35.tab")
```

In this experiment, data had been recorded in matrix form. For the analysis with `lm()` we have to transform the data from a matrix form in a long form which has a treatment label in one column. The column *H* of the plate in our experiment contains no treatment, but serves only for comparison between plates in the series for quality control.

Input

```
s35 <- stack(p35[, 3:9])
s35 <- data.frame(y = s35$values, Tmt = s35$ind,
Lane = rep(1:12, length.out = dim(s35)[1]))
lmres <- lm(y ~ 0 + Tmt, data = s35)
```

The summary as a linear model gives *t* tests for the individual coefficients.

Input

```
summary(lmres)
```

Output

```
Call:
lm(formula = y ~ 0 + Tmt, data = s35)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----|----|--------|----|-----|
|-----|----|--------|----|-----|

```

anova@anova|textit
`-0.084833 -0.016354  0.009125  0.022729  0.073083
glht@glht|textit

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
TmtA     0.19383   0.01035   18.73   <2e-16 ***
TmtB     0.24892   0.01035   24.06   <2e-16 ***
TmtC     0.23783   0.01035   22.99   <2e-16 ***
TmtD     0.24117   0.01035   23.31   <2e-16 ***
TmtE     0.24392   0.01035   23.57   <2e-16 ***
TmtF     0.23558   0.01035   22.77   <2e-16 ***
TmtG     0.22367   0.01035   21.62   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.03584 on 77 degrees of freedom
Multiple R-squared:  0.9787,    Adjusted R-squared:  0.9768 
F-statistic: 506.2 on 7 and 77 DF,  p-value: < 2.2e-16

```

For this example, tests for individual coefficients are not adequate. `anova()` gives a summary which is tailored to the needs of an analysis of variance.

Input

`anova(lmres)`

Output

Analysis of Variance Table

```

Response: y
          Df Sum Sq Mean Sq F value    Pr(>F)
Tmt       7 4.5513  0.6502  506.15 < 2.2e-16 ***
Residuals 77 0.0989  0.0013
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

```

Provided the assumptions of the Gauß-linear model are satisfied, this summary says that the treatment effect is significant. The next question that arises immediately is which of the treatments differ significantly, that is we are interested in the contrasts that describe the differences between treatments. Without violating the level of the test, we can use Tukey's approach to post-hoc test. To can use the function `glht()` for tests of generalised linear hypotheses, which is provided in the package `multcomp` for multiple testing.

expl:02-lhtres

Examplex 2.7:

plot@plot|textit

| Input | | | | | | |
|---|--|-----------------|--------|---------|----|--|
| library(multcomp) | lhtres <- glht(lmres, linfct = mcp(Tmt = "Tukey")) | summary(lhtres) | | | | |
| Output | | | | | | |
| Simultaneous Tests for General Linear Hypotheses | | | | | | |
| Multiple Comparisons of Means: Tukey Contrasts | | | | | | |
| Fit: lm(formula = y ~ 0 + Tmt, data = s35) | | | | | | |
| Linear Hypotheses: | | | | | | |
| B - A == 0 | 0.055083 | 0.014632 | 3.765 | 0.00599 | ** | |
| C - A == 0 | 0.044000 | 0.014632 | 3.007 | 0.05251 | . | |
| D - A == 0 | 0.047333 | 0.014632 | 3.235 | 0.02870 | * | |
| E - A == 0 | 0.050083 | 0.014632 | 3.423 | 0.01651 | * | |
| F - A == 0 | 0.041750 | 0.014632 | 2.853 | 0.07794 | . | |
| G - A == 0 | 0.029833 | 0.014632 | 2.039 | 0.39942 | . | |
| C - B == 0 | -0.011083 | 0.014632 | -0.757 | 0.98820 | . | |
| D - B == 0 | -0.007750 | 0.014632 | -0.530 | 0.99832 | . | |
| E - B == 0 | -0.005000 | 0.014632 | -0.342 | 0.99986 | . | |
| F - B == 0 | -0.013333 | 0.014632 | -0.911 | 0.96970 | . | |
| G - B == 0 | -0.025250 | 0.014632 | -1.726 | 0.60127 | . | |
| D - C == 0 | 0.003333 | 0.014632 | 0.228 | 0.99999 | . | |
| E - C == 0 | 0.006083 | 0.014632 | 0.416 | 0.99958 | . | |
| F - C == 0 | -0.002250 | 0.014632 | -0.154 | 1.00000 | . | |
| G - C == 0 | -0.014167 | 0.014632 | -0.968 | 0.95930 | . | |
| E - D == 0 | 0.002750 | 0.014632 | 0.188 | 1.00000 | . | |
| F - D == 0 | -0.005583 | 0.014632 | -0.382 | 0.99974 | . | |
| G - D == 0 | -0.017500 | 0.014632 | -1.196 | 0.89359 | . | |
| F - E == 0 | -0.008333 | 0.014632 | -0.570 | 0.99748 | . | |
| G - E == 0 | -0.020250 | 0.014632 | -1.384 | 0.80864 | . | |
| G - F == 0 | -0.011917 | 0.014632 | -0.814 | 0.98279 | . | |
| --- | | | | | | |
| Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 | | | | | | |
| (Adjusted p values reported) | | | | | | |

Under the assumptions of the model the significance of the differences between A and B , the differences between A D are established.

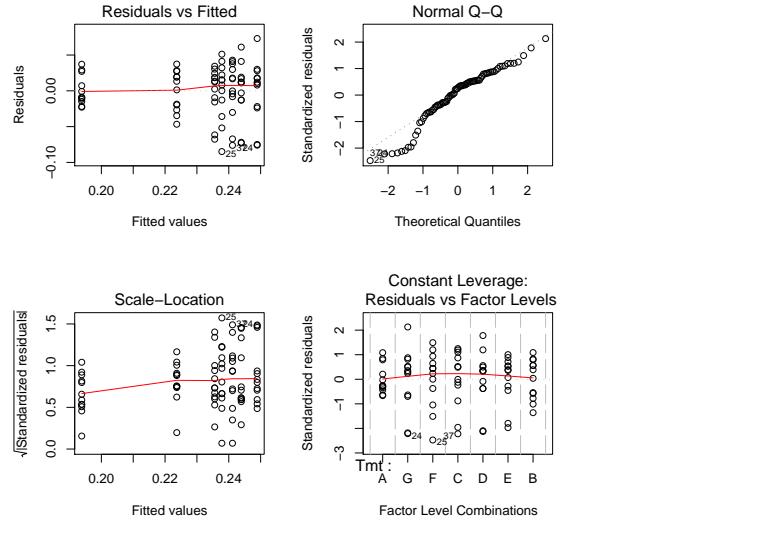
To check the assumptions, we can use the residuals which can be inspected with `plot()`.

expl:02-plotlhtres

2-32

Input

REGRESSION

`oldpar <- par(mfrow = c(2, 2))``par(oldpar)``plot(lm28:``par(oldpar)`

The distribution shows a clear deviation from the normal distribution, in particular at low values. We can inspect these selectively.

expl:02-s35studres

Example 2.9:

Input

```
library(MASS)
s35$studres <- studres(lmres)
s35[s35$studres < -1, ]
```

Output

| y | Tmt | Lane | studres |
|----|-------|------|--------------|
| 13 | 0.174 | B | 1 -2.239390 |
| 24 | 0.173 | B | 12 -2.271296 |
| 25 | 0.153 | C | 1 -2.559766 |
| 33 | 0.202 | C | 9 -1.044865 |
| 36 | 0.186 | C | 12 -1.523409 |
| 37 | 0.165 | D | 1 -2.279286 |
| 48 | 0.174 | D | 12 -1.994858 |
| 49 | 0.171 | E | 1 -2.175828 |
| 60 | 0.172 | E | 12 -2.144169 |
| 61 | 0.168 | F | 1 -2.007887 |
| 72 | 0.174 | F | 12 -1.821449 |
| 73 | 0.177 | G | 1 -1.367608 |
| 84 | 0.189 | G | 12 -1.010381 |

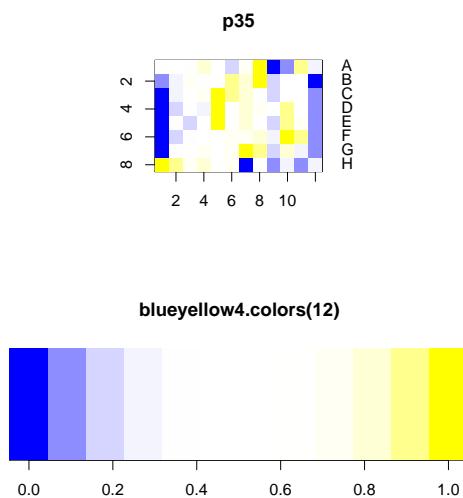
The pattern is conspicuous. Nearly all small values occur at the boundary of the plate.

We could as well have detected this pattern by visual inspection:

expl:02-bertin

2-34 REGRESSION

```
Input  
a35 <- as.matrix(p35[3:10])  
Eac2triplex apply(a35, 2, rank)  
library(bertin)  
oldpar <- par(mfrow = c(2, 1))  
image(t(a35rk), col = blueyellow4.colors(12), main = "p35")  
colramp(blueyellow4.colors(12), 12, horizontal = TRUE)  
par(oldpar)
```



For independent errors we would expect a random distribution of the range values over the rows. The concentration of the extreme values in the extreme columns shows an inhomogeneity in the experimental production process.

ToDo:

rows,
columns,
lanes?

So for this example we can report that apparently there is a difference between treatment A and particular other treatments. The finding has to be seen with precaution: the model assumptions are not satisfied. There is a noticeable inhomogeneity between the rows. More important than the findings on the treatment differences may be to follow the hint to look into the production process for the sources of this inhomogeneity.

We would not be able to detect a similar effect for the rows. Since the treatments are applied row-wise, treatment effect and column effect are confounded, and the effects are not identifiable.

2.5 Beyond Linear Regression

s:2.3

Transformations

Box-Cox Transformations

subs:3.4.1

Finding scale and location parameters can be understood as attempt to transform the distribution to some reference form. However scale and location cover only linear transformations.

The *Box -Cox transformations*

$$y^{(\lambda)} = \begin{cases} \frac{y^\lambda - 1}{\lambda} & \text{for } \lambda \neq 0, \\ \log(y) & \text{for } \lambda = 0 \end{cases}$$

Box-Cox
transformation | textbf
boxcox@boxcox | textit

are a family which is scaled to imbed the logarithm transformation smoothly in the power transformations. The function `boxcox()` in `library(MASS)` can be used to choose λ .

Using transformations, linear models can sometimes be used to cover non-linear situations.

Generalised linear models are an extension of linear models that allow to cover certain transformations within the model. For more information, see [25].

2.5.1 Generalized Linear Models

We want to proceed to practical work. But at this point we should take the time for an outlook how to overcome the limiting assumptions of linear models. Linear models count among the statistical models which are investigates best. Theory and algorithms are far advanced. So it is inviting to try how this class of models can be extended while still allowing to use the theoretical and algorithmical know-how.

We formulated the linear model as

$$\begin{aligned} Y &= m(X) + \varepsilon \\ &\quad Y \text{ with values in } \mathbb{R}^n \\ &\quad X \in \mathbb{R}^{n \times p} \\ &\quad E(\varepsilon) = 0 \\ &\quad \text{with } m(X) = X\beta, \quad \beta \in \mathbb{R}^p. \end{aligned}$$

An important extension is to remove the linearity assumption. As an intermediate step, we do not suppose any more that m is linear, but only that it can be factored using a linear function. This results in a generalised linear model

$$\begin{aligned} Y &= m(X) + \varepsilon \\ &\quad Y \text{ with values in } \mathbb{R}^n \\ &\quad X \in \mathbb{R}^{n \times p} \\ &\quad E(\varepsilon) = 0 \\ &\quad m(X) = \bar{m}(\eta) \text{ with } \eta = X\beta, \quad \beta \in \mathbb{R}^p. \end{aligned}$$

The next generalisation at hand is to allow for a transformation for Y . Many more generalisation have been discussed. A small number of them has proven tractable. These remaining models are called generalised linear models. Generalised linear models have extensive support in R. For most of the functions in R for linear model, there is a corresponding function for generalised linear models. For more information see `help(glm)`.

2.5.2 Local Regression

We now make a big jump. We have discussed linear models. We now that we can represent non-linear functions using linear models, but the term entering the functions must be specified off

loess@loess|textbf hand. Too many terms lead to over-fitting; so it is not the best idea to leave it to the algorithm
Topic for linear models to do the model selection. Statistical treatment of a regression problem with
smooth!loess@loess mild assumptions on the model function remains a problem.

Topic **loess!loess@loess** A partial approach for a solution comes from analysis. In analysis, it is a standard technique to use local approximations for functions. The analogous approach is to use a localised variant instead of a global estimation. We still assume that

$$\begin{aligned} Y &= m(X) + \varepsilon \quad Y \in \mathbb{R}^n \\ X &\in \mathbb{R}^{n \times p} \\ E(\varepsilon) &= 0, \end{aligned}$$

but assume linearity only locally:

$$m(x) \approx x' \beta_{x_0} \quad \beta_{x_0} \in \mathbb{R}^p \text{ and } x \approx x_0.$$

For practical work, abstract asymptotics is not sufficient. We have to specify what \approx means. This can be done with reference to some scales (for example $x \approx x_0$ if $|x - x_0| < 3$) or with reference to the design (z.B. $x \approx x_0$ if $\#\{i : |x - x_i| \leq |x - x_0| < n/3\}$). Up to date implementations use fine variant that cannot be discussed here. For an illustration of concepts, a coarse image may be sufficient:

Localised Gauß-Markov estimator:

For $x \in \mathbb{R}^p$, find

$$\delta = \min_d : (\#\{i : |x - x_i| \leq d\}) \geq n \cdot f$$

where f is a chosen fraction (for example 0.5).

Calculate the Gauß-Markov estimator $\hat{\beta}_x$, using only those observations for which $|x - x_i| \leq \delta$. Estimate

$$\hat{m}(x) = x' \hat{\beta}_x.$$

This coarsening ignores all measurement points which have a distance of more than δ . More subtle methods use a weight function to reduce the influence of distant measurement points progressively.

help(loess)

loess

Local Polynomial Regression Fitting

Description

Fit a polynomial surface determined by one or more numerical predictors, using local fitting.

Usage

```
loess(formula, data, weights, subset, na.action, model = FALSE,
      span = 0.75, enp.target, degree = 2,
      parametric = FALSE, drop.square = FALSE, normalize = TRUE,
      family = c("gaussian", "symmetric"),
      method = c("loess", "model.frame"),
      control = loess.control(...), ...)
```

Arguments

| | |
|--------------------|--|
| formula | a formula specifying the numeric response and one to four numeric predictors (best specified via an interaction, but can also be specified additively). |
| data | an optional data frame, list or environment (or object coercible by <code>as.data.frame</code> to a data frame) containing the variables in the model. If not found in data , the variables are taken from <code>environment(formula)</code> , typically the environment from which <code>loess</code> is called. |
| weights | optional weights for each case. |
| subset | an optional specification of a subset of the data to be used. |
| na.action | the action to be taken with missing values in the response or predictors. The default is given by <code>getOption("na.action")</code> . |
| model | should the model frame be returned? |
| span | the parameter α which controls the degree of smoothing. |
| enp.target | an alternative way to specify span , as the approximate equivalent number of parameters to be used. |
| degree | the degree of the polynomials to be used, up to 2. |
| parametric | should any terms be fitted globally rather than locally? Terms can be specified by name, number or as a logical vector of the same length as the number of predictors. |
| drop.square | for fits with more than one predictor and degree =2, should the quadratic term (and cross-terms) be dropped for particular predictors? Terms are specified in the same way as for parametric . |
| normalize | should the predictors be normalized to a common scale if there is more than one? The normalization used is to set the 10% trimmed standard deviation to one. Set to false for spatial coordinate predictors and others known to be a common scale. |
| family | if "gaussian" fitting is by least-squares, and if "symmetric" a redescending M estimator is used with Tukey's biweight function. |
| method | fit the model or just extract the model frame. |
| control | control parameters: see <code>loess.control</code> . |
| ... | control parameters can also be supplied directly. |

Details

Fitting is done locally. That is, for the fit at point x , the fit is made using points in a neighbourhood of x , weighted by their distance from x (with differences in 'parametric' variables being ignored when computing the distance). The size of the neighbourhood is controlled by α (set by **span** or **enp.target**). For $\alpha < 1$, the neighbourhood includes proportion α of the points, and these have tricubic weighting (proportional to $(1 - (\text{dist}/\text{maxdist})^3)^3$). For $\alpha > 1$, all points are used, with the 'maximum distance' assumed to be $\alpha^{1/p}$ times the actual maximum distance for p explanatory variables.

For the default family, fitting is by (weighted) least squares. For **family**="symmetric" a few iterations of an M-estimation procedure with Tukey's biweight are used. Be aware that as the initial value is the least-squares fit, this need not be a very resistant fit.

It can be important to tune the control list to achieve acceptable speed. See `loess.control` for details.

```
as.data.frame@as.data.frame
loess.control@loess.control
loess.control@loess.control
```

```
loess.control@loess@textit
predict.loess@predict.loess@textit
lowess@lowess@textit
An object of class "loess".
```

Note

As this is based on the `cloess` package available at `netlib`, it is similar to but not identical to the `loess` function of S. In particular, conditioning is not implemented.

The memory usage of this implementation of `loess` is roughly quadratic in the number of points, with 1000 points taking about 10Mb.

Author(s)

B.D. Ripley, based on the `cloess` package of Cleveland, Grosse and Shyu available at <http://www.netlib.org/a/>.

References

W.S. Cleveland, E. Grosse and W.M. Shyu (1992) Local regression models. Chapter 8 of *Statistical Models in S* eds J.M. Chambers and T.J. Hastie, Wadsworth & Brooks/Cole.

See Also

`loess.control`, `predict.loess`.
`lowess`, the ancestor of `loess` (with different defaults!).

Examples

```
cars.lo <- loess(dist ~ speed, cars)
predict(cars.lo, data.frame(speed = seq(5, 30, 1)), se = TRUE)
# to allow extrapolation
cars.lo2 <- loess(dist ~ speed, cars,
control = loess.control(surface = "direct"))
predict(cars.lo2, data.frame(speed = seq(5, 30, 1)), se = TRUE)
```

While linear regression is forced by the model assumptions to give a linear image (or an image with linear parametrisation) under all circumstances, a localised variant can represent non linearities properly. The analysis of this family of methods gives a branch of its own in statistics: nonparametric regression.

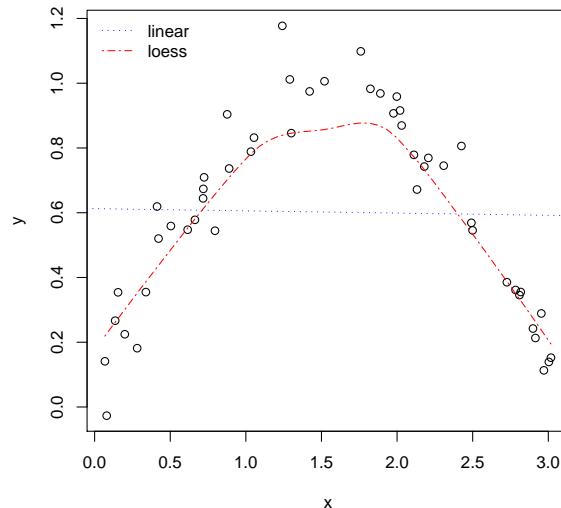
We prepare an example:

| | |
|---------------|---|
| expl:02-loess | <i>Input</i> |
| | x <- runif(50) * pi y <- sin(x) + rnorm(50)/10 |

Example 2.11:

```
plot(x, y)
abline(lm(y ~ x), lty = 3, col = "blue")
lines(loess.smooth(x, y), lty = 6, col = "red")
legend("topleft", legend = c("linear", "loess"),
       lty = c(3, 6), col = c("blue", "red"), bty = "n")
```

save@save|textit
load@load|textit



2.6 R Complements

s:2.4

2.6.1 Complements: Discretisation

In analogy to the construction of histograms we can discretise the data. We did this already with respect to the regressors when we discussed the helicopter example. The response can be discretised as well. This convert a regression problem into a contingency table problem. We will not delve into this possibility here.

To Do:
ch02: Add example

2.6.2 Complements: External Data

Data, as well as any other R objects can be written into an external file using `save()`, and read from there again using `load()`. Data are stored in these files in compressed form. The R storage files are exchangeable between different R systems.

| <code>data@data text stack@stack te</code> | <i>Data In/Output for R</i> | |
|--|---------------------------------|--|
| | <code>save()</code> | Store data in an external file. Syntax: <code>save(<names of the objects to store>, file = <file name>, ...)</code> |
| | <code>load()</code> | Load data from an external file. Syntax: <code>load(file = <file name>, ...)</code> |

Often data are collected or prepared with other systems. R provides a series of functions to read data in various formats. See appendix section [A.9.7](#) on page Seite [A-74](#). More information is given in the manual “Data Import/Export” ([15]).

The function `data()` bundles various access routines, provided access paths and file names follow the R conventions.

Often imported data have to be post-processed to align the format to the calling conventions of the R functions which will be applied.

For example, `lm` expects the regressors as separate variables. On the other side, for factorial designs it is usual to collect the results in a table where the factor-levels appear as row or column labels. The function `stack()` transforms tables into columns.

ToDo:

syntax:
`stack`,
`unstack`,
`split`, `cut`

2.6.3 Complement: Testing Software

All algorithms, like the algorithms for linear models and their variants, should treated with the same precautions as any mathematical publication or quotation. Unfortunately even simple programs soon have a semantic complexity far beyond that of many mathematical proof. The usual strategies like re-computing or stepwise execution forbid them selves. Selective testing must take the place of complete examination. An example how to design test strategies is documented in [Sawitzki, 1994].

Scrutiny is necessary for the implementation as well as for the underlying abstract algorithm.

| Exercise 2.16 | |
|---------------|--|
| | For this series of exercises, let $y_i = a + bx_i + \varepsilon_i$ with $\varepsilon_i \text{ iid } \sim N(0, \sigma^2)$ and $x_i = i, i = 1, \dots, 10$. |
| | <p>Chose a strategy to inspect <code>lm()</code> with regard to the parameter space (a, b, σ^2).</p> <p>Are there apparent cellular decompositions for the parameters a, b, σ^2?</p> <p>What are the trivial cases? What are the asymptotics that apply?</p> <p>Choose test points in the interior of each cell and on the boundaries.</p> <p>Perform these tests and summarise the results.</p> |
| | (cont.)→ |

| | | |
|----------------------|--|--|
| Exercise 2.16 | (cont.) | bde@mode textit storage.mode@storage.mode typeof@typeof textit typeof@typeof textit attr@attr textit |
| | What are the symmetries/anti-symmetries that apply? Check for these symmetries. | typeof@typeof textit typeof@typeof textit attr@attr textit |
| | Which invariant or covariate behaviour applies? Check for these invariant or covariate behaviour. | |

a:ch02:221

| | |
|----------------------|---|
| Exercise 2.17 | |
| | For this series of exercises, let $y_i = a + bx_i + \varepsilon_i$ with ε_i iid $\sim N(0, \sigma^2)$. |
| | What are the extremal designs (x_i)? Check the behaviour of <code>lm()</code> for four extremal designs. |
| | Perform the tests from the last exercise, now with variable design. Summarise your results. |

a:ch02:23

| | |
|----------------------|--|
| Exercise 2.18 | |
| | For this series of exercises, let $y_i = a + bx_i + \varepsilon_i$ with ε_i iid $\sim N(0, \sigma^2)$. |
| | Modify <code>lm()</code> to give a fail-safe function for simple linear models which checks deviations from the model assumptions as well. |

2.6.4 R Data Types

R is an interpreted programming language. It tried to allow a flexible handling of definitions and specifications for the user. For performance reasons, R attempts to evaluate expressions and terms as late as possible. This requires some restriction for the language which makes R differ from other programming languages.

R does not have abstract data types. A data type is defined by its instances, the variables.

The data type of a variable is dynamic: in the same context the same name can refer to different variable values and different variable types at different times.

At any time however any variable has a well defined type. The R type system is best understood with reference to its historical development and the corresponding functions. Initially the type was described by `mode()` (z.B. “numeric”) and `storage.mode()` (z.B. “integer” or “real”).

Essentially both functions have been replaced by `typeof()`. A summary of the type reported by `typeof()` is in [16].

More complex data types are derived from the basic types documented in [16] by attaching attributes to the variables. This is done using function `attr()` which can also be applied to inspect attributes. So for example a matrix or an array just special vectors, which are special by having a `dim` attribute. The `class` attribute served to denote and name a class explicitly.

Inspection and conversion functions are supplied for the main types: `is.(typ)()` tests for a type, `as.(typ)()` converts the type.

ToDo:
conditional,
function
parameters

polymorph *2.6.5 Classes and Polymorphic Functions*
ch02:polymorph | textit
unclass@unclass | textit

inherits@inherits | textit
 plot@plot | textit
 R The special attribute **class** is used here. The name of the type (or the “class”) is stored as **print@print** | textit
class attribute. Multiple class membership is possible. In this case, **class** is a vector of class **UseMethod@UseMethod** | textit
names. So for example an ordered factor has the attribute **class = c("ordered", "factor")**.
UseMethod@UseMethod | textit
polymorph For class management, the functions **class()**, **unclass()**, **inherits()** are provided.

methods@methods | textit
lm@lm | textit Class membership in R is based on trust. R does not check, whether a data structures is consistent with its acclaimed class.

Functions like **plot()**, **print()** and many other inspect the class or type of its argument and branch to specialised functions as appropriate. This is called function polymorphism. If you list a polymorphic function, first you get only a clue the a Dispatch-function **UseMethod()** is called.
 Example:

expl:02-plot

Example 2.12:

| | |
|--------------|---|
| Input | Output |
| plot | |
| | function (x, y, ...) |
| | { |
| | if (is.function(x) && is.null(attr(x, "class"))) { |
| | if (missing(y)) |
| | y <- NULL |
| | hasylab <- function(...) ! all(is.na(pmatch(names(list(...)), |
| | "ylab")) |
| | if (hasylab(...)) |
| | plot.function (x, y, ...) |
| | else plot.function (x, y, ylab = paste(deparse(substitute(x)), |
| | "(x)" , ...) |
| | } |
| | else UseMethod("plot") |
| | } |
| | <environment: namespace:graphics> |

UseMethod() tries to identify the class of the first argument that was used in the function call. Then it tried to find a specialisation of the function, a method, for this class and finally it calls this function . for **polymorphic** functions the available methods are listed with **methods()**, z.B. **methods(plot)**.

2.6.6 Extractor Functions

Functions like **lm()** yield complex data with rich information. In a purely object oriented environment, access methods and data would be encapsulated. In R is object orientation is

implemented rudimentary and with varying models. This partially reflects the development.
For sufficiently general structures, access methods as in section 2.6.5 are available. For the objects as returned by `lm()`, a series of extractor functions all provide access to components in various forms.

`lm@lm|textit`

| <i>Extractor Functions for lm</i> | |
|-----------------------------------|--|
| <code>coef()</code> | extracts the estimated coefficients |
| <code>effects()</code> | extracts successive orthogonal components |
| <code>residuals()</code> | raw residuals |
| <code>stdres()</code> | (in <code>library(MASS)</code>) standardised residuals |
| <code>studres()</code> | (in <code>library(MASS)</code>) externally standardised residuals |
| <code>fitted()</code> | |
| <code>vcov()</code> | variance/covariance matrix of the estimated parameter |
| <code>predict()</code> | confidence and tolerance intervals |
| <code>confint()</code> | confidence intervals for parameter |
| <code>influence()</code> | extract influence diagnostics |
| <code>model.matrix()</code> | derives the design matrix |

2.7 Statistical Summary

The leading example in this chapter was the statistical analysis of a functional relationship. The models considered are finite in the sense that only a finite dimension function space was taken into account to describe the relation between regressors and response. The stochastic component in these models was restricted to s (one dimensional) random variable. The notion of dimension here deserves further attention. For one, we have the regressor dimension. This is the dimension the space of the observed or derived parameters. Not all parameters are identifiable or estimable. More precisely, this dimension is the vector space dimension of the chosen model space. Models are indicated by parameters in this space. The parameter can be unknown or hypothetical. In any case we have considered the parameters as deterministic. On the other side we have the stochastic component, represented by the error term. In this chapter, we assumed homogeneous errors. So the error term in principle is one dimensional, but taken from a space of distributions. For the special case of simple Gauß-linear models, the distributions are specified by two parameters, the expected value and the variance. We freed us from the expected value by the assumption that the model would cover all systematic effects on the average, hence the expected value is null. The variance remains as an unknown nuisance parameter. We avoided the problems coming from this nuisance parameter by restricting us to problems where the nuisance parameter could be replaced by an estimated value and thus be eliminated.

2.8 Literature and Additional References

`s:s02-bib`

`[chh92sms`
`[3]` Chambers, J.M.; Hastie, T.J. (eds.) (1992): Statistical Models in S. New York: Chapman & Hall.

[**jor93tlm**] Jørgensen, B. (1993): The Theory of Linear Models. NewYork: Chapman & Hall.

[**R:Rlang**] R Development Core Team (2004): The R language definition.

[**gs94nrel1**] Sawitzki, G. (1994): Numerical Reliability of Data Analysis Systems. Computational Statistics & Data Analysis 18.2 (1994) 269-286. <<http://www.statlab.uni-heidelberg.de/reports/>>.

[**gs94nrel2**] Sawitzki, G. (1994):Report on the Numerical Reliability of Data Analysis Systems. Computational Statistics & Data Analysis/SSN 18.2 (1994) 289-301. <<http://www.statlab.uni-heidelberg.de/reports/>>.

Generated by Sweave from:

```
Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/Rnw/S02reg.Rnw.tex,v  
Revision: 1.18  
Date: 2008/03/18 09:01:43  
name:  
Author: j40  
$Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/Rnw/S02reg.Rnw.tex,v $  
$Revision: 1.18 $  
$Date: 2008/03/18 09:01:43 $  
$name: $  
$Author: j40 $
```

CHAPTER 3

Comparison of Distributions

ch:03

We begin with the construction of a small gadget that will provide us with example data. The base is a small reaction timer. We draw a “random” point, wait for a mouse click, and record the position of the mouse pointer. To get a stable image for repeated activations, we fix the co-ordinate system..

ch03:Sframe03.1

Example 3.1:

Input

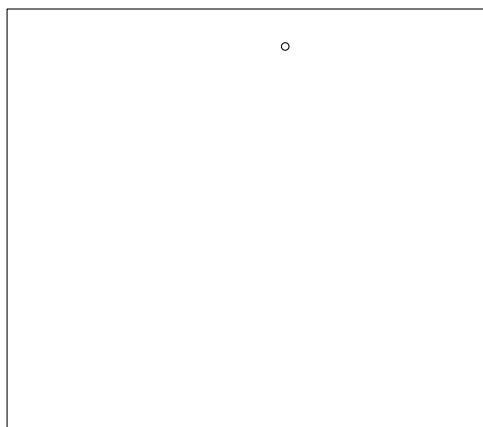
```
plot(x = runif(1), y = runif(1), xlim = c(0, 1),
      ylim = c(0, 1), main = "Please click on the circle",
      xlab = "", ylab = "", axes = FALSE, frame.plot = TRUE)
locator(1)
```

Output

```
$x  
[1] 0.6324415
```

```
$y  
[1] 0.3336268
```

Please click on the circle



Now we wrap up the base timer. We record the co-ordinates, try to measure the reaction time, and return the results as a list.

ch03:Sframe03.2

Example 3.2:

```
click1 <- function() {  
  Input  
  x <- runif(1)  
  y <- runif(1)  
  plot(x = x, y = y, xlim = c(0, 1), ylim = c(0,  
    1), main = "Please click on the circle",  
    xlab = "", ylab = "", axes = FALSE, frame.plot = TRUE)  
  clicktime <- system.time(xyclick <- locator(1))  
  list(timestamp = Sys.time(), x = x, y = y, xclick = xyclick$x,  
    yclick = xyclick$y, tclick = clicktime[3])  
}
```

For later processing we can integrate the list in a `data.frame` and extend this `data.frame` stepwise using `rbind`.

Example 3.3:

```
dx <- as.data.frame(click1())  
dx <- rbind(dx, data.frame(click1()))  
dx  
  
Output  
timestamp      x      y   xclick   yclick  
elapsed 2008-03-18 19:19:37 0.25199 0.27131 0.63846 0.33838  
elapsed1 2008-03-18 19:19:38 0.28412 0.72227 0.63846 0.33838  
      tclick  
elapsed    0.038  
elapsed1   0.378
```

a:03-click

Exercise 3.1

Define a function `click(runs)` that repeats `click1()` a chosen number `runs` of times and returns the result as a `data.frame`. An additional first timing should be considered as a “warming up” and not included in the following evaluation.

Select a number `runs`. Give reasons for your choice of `runs`. Execute `click(runs)` and store the result in a file using `write.table()`.

Display the distribution the component `tclick()` with the methods from chapter 1 (distribution function, histogram, box&whisker plot).

shift family **3.1 Shift/Scale Families**

stochastically

smaller

comparison of distributions can be a demanding task. The mathematical space of distributions is no longer a number space any more nor a finite dimensional vector space. Properly the room in which distributions reside is a space of probability measures. In simple cases, like for distributions on \mathbb{R} , can we reduce everything to distribution functions and come to a tractable function space. However even here a comparison can provide big problems. We do not have any simple order relation to base the comparison upon.

a:03-clickr1

| Exercise 3.2 | |
|--------------|--|
| | <p>a:03-click</p> <p>Perform exercise 3.1 using the right hand and then again using the left hand. Compare the empirical distributions of the timing data returned by <code>tclick()</code>.</p> <p>The recorded data contain also information about the positions. Define a distance measure <code>dist</code> for the deviation. Give reasons for your definition. Perform a right/left comparison for <code>dist</code>.</p> |

We concentrate on the comparison of two distributions only, for example that of the results of two treatments. And we take a simple case: we assume that the observations are independent identically distributed. We use the index notation which is usual for the comparison of treatments in the two sample case.

Y_{ij} independent identically distributed with distribution function F_i

$i = 1, 2$ treatments

$j = 1, \dots, n_i$ observations in treatment group i .

How do we compare the observations in the treatment groups $i = 1, 2$? The (simple) linear models

$$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$$

consider only the case where the difference amounts to a shift $\Delta = \alpha_1 - \alpha_2$.

Notations: For a distribution with distribution function F the family

$$F_a(x) = F(x - a)$$

is called the **shift family** for F . a is called the shift or location parameter.

Speaking in terms of probabilities, the treatment can shift probability mass in quite different way than what can be achieved by an additive term. We need more general ways to compare distribution than those provided by shift families.

Notation: A distribution with distribution function F_1 is **stochastically smaller** than a distribution with distribution function F_2 ($F_1 \prec F_2$), if F_1 takes rather smaller values than as F_2 . This means that F_1 increases sooner.

$$F_1(x) \geq F_2(x) \quad \forall x$$

and

$$F_1(x) > F_2(x) \text{ for at least one } x.$$

For shift families we have: If $a < 0$, then $F_a \prec F$. The shift results in a parallel shift of the distribution functions.

A typical result of the click experiment (exercise 3.1 is given in figure 3.1. The response times for the right side are stochastically smaller than those for the left side. But the distributions do not belong to a common shift family, because the distribution functions are not parallel.

`a:03-click` `fig:03clickcdf`
`scale shift`
`family|textbf`

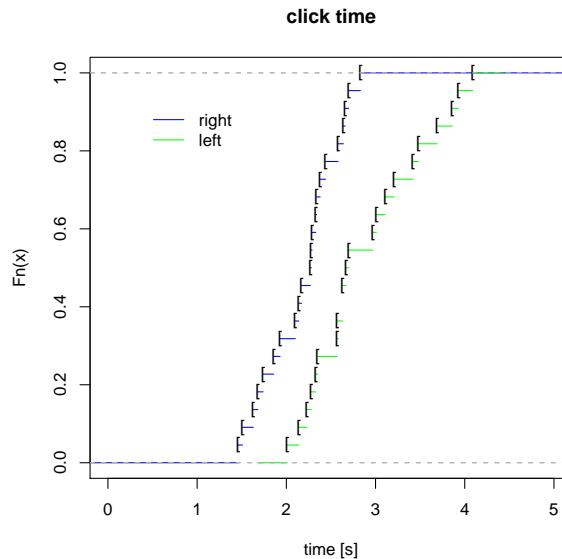


Figure 3.1 Distribution functions for the right/left click time

`fig:03clickcdf`

`a:ch03:1`

| Exercise 3.3 | |
|--------------|--|
| | How does a PP plot for F_1 against F_2 look like if $F_1 \prec F_2$? How does a QQ plot for F_1 against F_2 look like if $F_1 \prec F_2$? |

Unfortunately the stochastic order defined here is only of a limited value. It does not define a complete order. For shift families it is sufficient. But counter examples can be constructed by minor extensions of shift families.

Notations: For a distribution with distribution function F the family with

$$F_{a,b}(x) = F\left(\frac{x-a}{b}\right)$$

is called the *scale shift family* for F .

`a:ch03:2`

| | |
|--|--|
| <pre>qqnorm@qqnorm qqplot@qqplot </pre> Exercise 3.4 | <p>The scale shift family for the $N(0, 1)$ distribution are the $N(\mu, \sigma^2)$ distributions. Which $N(\mu, \sigma^2)$ distributions are stochastically smaller than the $N(0, 1)$ distribution? Which are stochastically larger? Which distributions have an undefined order relation to $N(0, 1)$?</p> |
|--|--|

The ordering by scale and location which comes from the linear theory, and stochastic order are different concepts, and both aspects must be considered separately. Many statistical methods concentrate on aspects which are motivated by scale shift families. Differences going beyond what can be formulated in terms of scale and shift often need special attention.

In chapter 2 we have considered a typical situation for linear models. In principle we dealt with scale shift families. The (stochastic) stochastic scale parameter in these models is but a nuisance parameter that can be eliminated. For this, we used an estimator for this stochastic scale parameter, the residual variance, which we cancelled out ultimately. As a particularity for Gauß-linear models we get independent estimators for the expected value and the variance. This helps us in Gauß-linear models to derive statistics which are independent from the nuisance scale parameter.

In general case however we have a spectrum of problems:

- shift alternatives
- shift/scale alternatives
- stochastic ordering
- general alternatives

Test and estimation methods often concentrate on just one aspect of the problem, the location. The stochastic scale parameter is here but a nuisance parameter. Differences in the shift parameter lead to stochastically monotone relations. Differences in the stochastic scale parameter are not so ranged in so simply and test statistics must first be cleared from this nuisance parameter, if not only the shift parameter but also the stochastic scale parameter may vary.

3.2 QQ plot, PP plot, and Comparison of Distributions

s:3.2

As means to compare distribution we already encountered the *PP* plot and the *QQ* plot. As long as we stay within one shift scale family, the *QQ* plot has at least one advantage over the *PP* plot:

Remark 3.1 If F_1, F_2 are distribution functions from a common shift scale family, the *QQ* plot of F_1 against F_2 is a straight line.

In particular for the Gaussian distributions the *QQ* plot against $N(0, 1)$ is a valuable tool. Any normal distribution gives a straight line in this plot. The *QQ* plot is already implemented for this situation as function `qqnorm()`.

For the comparison of two samples the corresponding function `qqplot()` can be used: if we denote the empirical quantiles by $Y_{1,(i:n)}$ resp. $Y_{2,(i:n)}$, the plot is the graph $(Y_{1,(i:n)}, Y_{2,(i:n)})_{i=1\dots n}$ in case of equal sample size. If the sample sizes differ, R generates nodes by linear interpolation, with the smaller of the two sample sizes determining the number of nodes.

The *PP* plot does not have equivariance properties comparable to those of the *QQ*. If we want to eliminate scale and shift parameters, we have to transform the data first. Mathematical theory however is simple for the *PP* plot. In particular there is a corresponding Kolmogorov-Smirnov test (see section 3.2.1).

On the other hand, the equivariance properties of the *QQ* plots are paid for with structural deficits. In low density regions, few data points control the plot. As a consequence, the plot show high variance. On the other hand, in these regions quantiles that are near as quantiles are far apart in value space: high variance combines unfavourably with high variability, and correspondingly the *QQ* plot shows large fluctuation. For most text book distributions this means that the *QQ* plot is not very useful if it comes to the tails. The *PP* plot does not have this kind of scale deficits, but then it does not have the equivariance properties of the the *QQ* plots. To cope with this, the *PP* plot is generally applied to variable which are standardised in an appropriate way.

help(qqplot)

| | |
|---------------------|--------------------------------|
| <code>qqnorm</code> | <i>Quantile-Quantile Plots</i> |
|---------------------|--------------------------------|

Description

`qqnorm` is a generic function the default method of which produces a normal QQ plot of the values in `y`. `qqline` adds a line to a normal quantile-quantile plot which passes through the first and third quartiles.

`qqplot` produces a QQ plot of two datasets.

Graphical parameters may be given as arguments to `qqnorm`, `qqplot` and `qqline`.

Usage

```
qqnorm(y, ...)
## Default S3 method:
qqnorm(y, ylim, main = "Normal Q-Q Plot",
      xlab = "Theoretical Quantiles", ylab = "Sample Quantiles",
      plot.it = TRUE, datax = FALSE, ...)

qqline(y, datax = FALSE, ...)

qqplot(x, y, plot.it = TRUE, xlab = deparse(substitute(x)),
       ylab = deparse(substitute(y)), ...)
```

Arguments

| | |
|---|--|
| <code>x</code> | The first sample for <code>qqplot</code> . |
| <code>y</code> | The second or only data sample. |
| <code>xlab</code> , <code>ylab</code> , <code>main</code> | plot labels. The <code>xlab</code> and <code>ylab</code> refer to the <code>y</code> and <code>x</code> axes respectively if <code>datax = TRUE</code> . |

```
qqnorm@qqnorm|textbf
qqline@qqline
  (qqnorm)
qqplot@qqplot
  (qqnorm)
Topic
  hplot!qqnorm@qqnorm
Topic
  dis-
  tri-
  bu-
  tion!qqnorm@qqnorm
```

```
NA@NA|textit
ppoints@ppoints|textit
  plot.it
    datax
    ylim, ...
```

logical. Should the result be plotted?
 logical. Should data values be on the x-axis?
 graphical parameters.

Value

For `qqnorm` and `qqplot`, a list with components

- `x` The x coordinates of the points that were/would be plotted
- `y` The original y vector, i.e., the corresponding y coordinates *including NAs*.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`ppoints`, used by `qqnorm` to generate approximations to expected order statistics for a normal distribution.

Examples

```
y <- rt(200, df = 5)
qqnorm(y); qqline(y, col = 2)
qqplot(y, rt(300, df = 5))

qqnorm(precip, ylab = "Precipitation [in/yr] for 70 US cities")
```

a:03-clickqq

| Exercise 3.5 | |
|--------------|---|
| | Use the <i>QQ</i> plot to compare the results of the right/left <i>click</i> experiments. Summarise the results. |
| | Combine the right/left <i>tclick</i> data to a vector. Compare the <i>QQ</i> plot with that of Monte Carlo samples taken from the joined vector. Hint: You can draw random samples with <code>sample()</code> . With <code>par(mfrow = c(2, 2))</code> you arrange the display area so that it shows four plots at a time. |
| ** | For <code>sample()</code> use <code>replace = FALSE</code> . How do you have to apply <code>sample()</code> now to split the joint vector into two vectors with Monte Carlo samples? What differences do you expect in comparison to <code>replace = TRUE</code> ? |

a:ch03:3

| Exercise 3.6 | |
|--------------|---|
| | Find scale and shift parameters for the right/left <i>click</i> data so that after using these parameters for transformation the groups match as good as possible. Describe the differences using these parameters. Use a model formulation in terms of a linear model. |
| | Use the function <code>boxplot()</code> , to display quartiles and tail behaviour. Compare the information with the information you derived from the scale and shift parameters. <i>Hint:</i> What corresponds to the shift (or location) parameter? Was corresponds to the stochastic scale parameter? |

If representations such as visual representations in displays or numeric representations in summary statistics are affine invariant, scale and shift parameters can be ignored. If representations are not affine invariant are, it is often helpful to estimate scale and shift parameters first, then standardise the distributions, and only then to inspect the standardised distributions.

The problem that comes potentially with this is that we have to take into account the stochastic behaviour of the scale and shift parameter estimation. The usual way out is to be cautious and use “conservative” tests and robust estimators. The following function tries to transformation scale and location to match a standard normal distribution.

Input

```
ScaleShiftStd <- function(x) {
  xq <- quantile(x[!is.na(x)], c(0.25, 0.75))
  y <- qnorm(c(0.25, 0.75))
  slope <- diff(y)/diff(xq)
  (x - median(x, na.rm = FALSE)) * slope
}
```

For direct comparison of distributions, we resume to techniques discussed in the first chapter. In principle what has been said there about the comparison to a theoretical distribution can be carried over to the comparison of two distributions, for example to comparison of two treatments. The statistical statements however have to adapted. In chapter one we had one fixed hypothetically known distribution to be compared with an empirical distribution. This is sometimes called the *one sample case*. Now we are in the *two sample case*. We have to compare two empirical distributions, with the understanding that the proper target of comparison are the underlying unknown distributions.

The idea of Monte Carlo band that we used in the one sample case does not have an immediate counterpart. We want to compare two distributions, but we have no distinguished model distribution to draw reference samples from.

However we can modify the idea and construct conditional Monte Carlo bands. Conditioning here means: the construction is dependent on the observed sample values. Assume we have two samples Y_{11}, \dots, Y_{1n_1} and Y_{21}, \dots, Y_{2n_2} of independent and within the group identically F_1 resp. F_2 distributed observations. If there is no difference between the groups, $(Y_{11}, \dots, Y_{1n_1}, Y_{21}, \dots, Y_{2n_2})$ is a iid random sample from the common distribution $F = F_1 = F_2$ with sample size $n = n_1 + n_2$. For an iid random sample, each permutation of the indices would arise with equal probability.

`bootstrap|textbf{t}` This motivates the following procedure: we permute the tuple $(Y_{11}, \dots, Y_{1n_1}, Y_{21}, \dots, Y_{2n_2})$ and after permutation we assign the first n_1 values to the first group, the remaining values to the second.

The permutation group soon is so big that an exhaustive evaluation becomes impossible. As a way out we use a random sample of permutations. We values generated by this procedure are used to generate Monte Carlo bands.

ToDo: ch03:5

ch03: stan-

dardise

scale

ToDo:

ch03: work

out

ToDo:

ch03: irex-

ercise:

power

| Exercise 3.7 | |
|--------------|---|
| | Modify the functions for the <i>PP</i> plot and the <i>QQ</i> plot so that Monte Carlo bands for the comparison of two samples are added. <i>Hint:</i> Use the function <code>sample()</code> to generate random permutations. |

At larger sample size, the mere administration overhead to generate permutations may become to big. Instead of generation permutations, drawing samples with replacement from the n values $(Y_{11}, \dots, Y_{1n_1}, Y_{12}, \dots, Y_{1n_2})$ **with replacement** may be more economical and used as a substitute. This approximative solution is known as *bootstrap approximation**.

Since there are only finitely many permutations for a given sample size, for small sample sizes we can enumerate all permutations. We choose bands such that a sufficiently high proportion of all curves are captures within the bands (for example more than 95 %). permutations which only differ within the groups give the same bands. Hence we need not check all $n!$ permutations, but only the $\binom{n}{n_1}$ selections for the assignment to the groups.

| Exercise 3.8 | |
|--------------|--|
| ** | Augment the <i>PP</i> plot and <i>QQ</i> plot for the <i>click</i> experiments by permutation bands that cover 95 % of the permutations. |
| * | Generate new plots from the <i>PP</i> -Plots and <i>QQ</i> plots by adding Monte Carlo bands from permutations. Use the envelop of 19 Monte Carlo samples. <i>Hint:</i> Use function <code>sample()</code> to draw a random sample of sample size n_1 from $x = (Y_{11}, \dots, Y_{1n_1}, Y_{12}, \dots, Y_{1n_2})$. |
| | <i>Hint:</i> See <code>help(sample)</code> . |

a:ch03:6

| Exercise 3.9 | |
|--------------|---|
| * | Try to compare the properties of permutation bands, Monte Carlo bands and Bootstrap bands on the hypothesis where $F_1 = F_2$. |

If not the distributions are to be compared, but only single specified parameters, an analogous strategy can be used. For example, if we focus on shift alternatives (that is F_1 and F_2 are

* Caution: there are arbitrarily wild definitions of bootstrap. Always try to get a concise mathematical formulation when speaking about bootstrap.

a:ch03:7

from a shift family, $F_1(x) = F_2(x - a)$ for some a , we can take the mean (or the median) as characteristic parameter. The procedure given above can be used analogously to test the hypothesis that the distributions are not different ($a = 0$), based on the data.

a:ch03:8

| Exercise 3.10 | |
|----------------------|--|
| * | <p>Formulate the strategies given above for intervals of single test statistics (example: mean) instead of bands.</p> <p><i>Hint:</i> Instead of the two mean values for both groups, can you use a single one dimensional statistics?</p> |

3.2.1 Kolmogorov Smirnov Tests

subs:3.4.2

In chapter 1 we have seen the Kolmogorov-Smirnov test for comparison of a random sample $(X_i)_{i=1,\dots,n}$ and the corresponding empirical distribution function F_n with a (fixed, given) distribution F . The critical test statistics was

$$\sup |F_n - F|.$$

We can modify this test slightly to compare two empirical distributions. Instead of a model distribution F we now have a second empirical distribution G_m from observations $(Y_j)_{j=1,\dots,m}$ based on some underlying (unknown) distribution G . The critical test statistics is now

$$\sup |F_n - G_m|.$$

The test based on this statistics is in the 2 sample Kolmogorov-Smirnov test which is presented in most text books. This test corresponds to the *PP* plot and allows the construction of bands for the *PP* plot.

We also can use simulation to determine bands. In contrast to the one sample case we do not have a given distribution from which to simulate. Under the hypothesis, that the distributions F and G do not differ for independent observations the joined vector $(X_1, \dots, X_n, Y_1, \dots, Y_m)$ is vector of $n + m$ independent random numbers with identical distribution $F = G$. Given a data set, this relation can be used or simulation. Using a permutation π of the indices from the vector $Z = (X_1, \dots, X_n, Y_1, \dots, Y_m)$ a new vector Z' with $Z'_i = Z_{\pi(i)}$ is generated. The first n components are used as simulated values $(X'_i)_{i=1,\dots,n}$, the remaining m components as simulated values $(Y'_j)_{j=1,\dots,m}$.

a:ch03:14

| Exercise 3.11 | |
|----------------------|--|
| * | Implement this algorithm and enhance the <i>PP</i> plot by adding simulated <i>PP</i> plots for a for a small number (19?) of permutations. |
| | Determine the permutation distribution of $\sup F_n - G_m $ from the simulation and calculate this statistics for the original data. Can you use this comparison to define a test procedure? |
| | The Kolmogorov-Smirnov test as implemented uses an Approximation for the two sample case. In our simulation we know that we simulate under the hypothesis, that is the hypothesis holds. Inspect the distribution of the nominal level under the simulated conditions. |

```
testit
a:ch02:21
t.test t.test|textbf
```

Topic

ht-

est!t.test

Exercise 3.12

Use the *QQ* plot, for a pairwise comparison of the results of the helicopter experiment from chapter 2. Summarise your Resultate.

```
a:ch02:22
```

Exercise 3.13

Inspect the implementation of *qqnorm()*. Implement an analogous function for the *PP* plot and apply it to the helicopter data.

3.3 Tests for Shift Alternatives

```
s:3.3
```

If we make addition distribution assumptions, we can choose other decision rules. For these however the distribution assumptions may be critical. The dependency on these distribution assumptions can be relaxed or avoided, if we can guarantee appropriate assumptions by procedure. The *F* test which we met in the last chapter is an example for a distribution based method. For the Two sample case this test can be modified as *t* test, which also gives an indication of the direction of the difference. (The square of the *t* statistics is a *F* statistics.)

[help\(t.test\)](#)

`t.test`

Student's t-Test

Description

Performs one and two sample t-tests on vectors of data.

Usage

```
t.test(x, ...)

## Default S3 method:
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)

## S3 method for class 'formula':
t.test(formula, data, subset, na.action, ...)
```

Arguments

| | |
|--------------------|--|
| x | a numeric vector of data values. |
| y | an optional numeric vector data values. |
| alternative | a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. |
| mu | a number indicating the true value of the mean (or difference in means if you are performing a two sample test). |
| paired | a logical indicating whether you want a paired t-test. |
| var.equal | a logical variable indicating whether to treat the two variances as being equal. If TRUE then the pooled variance is used to estimate the variance otherwise the Welch (or Satterthwaite) approximation to the degrees of freedom is used. |
| conf.level | confidence level of the interval. |
| formula | a formula of the form lhs ~ rhs where lhs is a numeric variable giving the data values and rhs a factor with two levels giving the corresponding groups. |
| data | an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula . By default the variables are taken from environment(formula) . |
| subset | an optional vector specifying a subset of observations to be used. |
| na.action | a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action") . |
| ... | further arguments to be passed to or from methods. |

Details

The formula interface is only applicable for the 2-sample tests.

If **paired** is TRUE then both **x** and **y** must be specified and they must be the same length. Missing values are removed (in pairs if **paired** is TRUE). If **var.equal** is TRUE then the pooled estimate of the variance is used. By default, if **var.equal** is FALSE then the variance is estimated separately for both groups and the Welch modification to the degrees of freedom is used.

If the input data are effectively constant (compared to the larger of the two means) an error is generated.

Value

A list with class "htest" containing the following components:

| | |
|------------------|---|
| statistic | the value of the t-statistic. |
| parameter | the degrees of freedom for the t-statistic. |
| p.value | the p-value for the test. |
| conf.int | a confidence interval for the mean appropriate to the specified alternative hypothesis. |

| | |
|--|---|
| <code>prop.test@prop.test\$estimate</code> | the estimated mean or difference in means depending on whether it was a one-sample test or a two-sample test. |
| <code>null.value</code> | the specified hypothesized value of the mean or mean difference depending on whether it was a one-sample test or a two-sample test. |
| <code>alternative</code> | a character string describing the alternative hypothesis. |
| <code>method</code> | a character string indicating what type of t-test was performed. |
| <code>data.name</code> | a character string giving the name(s) of the data. |

See Also`prop.test`*Examples*

```
t.test(1:10,y=c(7:20))      # P = .00001855
t.test(1:10,y=c(7:20, 200)) # P = .1245    -- NOT significant anymore

## Classical example: Student's sleep data
plot(extra ~ group, data = sleep)
## Traditional interface
with(sleep, t.test(extra[group == 1], extra[group == 2]))
## Formula interface
t.test(extra ~ group, data = sleep)
```

In its text book for the t test supposed, that we have independent identically distributed samples from normal distributions. In fact, weaker assumptions are sufficient. Writing the t test statistics as

$$t = \frac{\widehat{\mu}_1 - \widehat{\mu}_2}{\sqrt{Var(\widehat{\mu}_1 - \widehat{\mu}_2)}} \quad (3.1)$$

eq:03tstat

we see that t is t distributed if $\widehat{\mu}_1 - \widehat{\mu}_2$ has a normal, $(Var(\widehat{\mu}_1 - \widehat{\mu}_2))$ has a χ^2 distribution, and both terms are independent. The central limit theorem guarantees that under mild conditions $\widehat{\mu}_1 - \widehat{\mu}_2$ at least asymptotically has a normal distribution. Similarly under not too strict conditions $(Var(\widehat{\mu}_1 - \widehat{\mu}_2))$ behaves asymptotically as χ^2 distributed. If both terms are independent, t is approximatively t distributed.

a03-tsim

| Exercise 3.14 | |
|---------------|---|
| * | <p>Use a simulation to inspect the distribution of \bar{Y}, $Var(\bar{Y})$ and the t statistics for \bar{Y} from a uniform distribution $U[0, 1]$ with sample size $n = 1, \dots, 10$. Compare the distributions from the simulation with the corresponding normal-, χ^2- resp. t distribution.</p> <p>Use a simulation to inspect the distribution of \bar{Y}, $Var(\bar{Y})$ and the t statistics for \bar{Y} from a mixture, consisting at 90% from a $N(0, 1)$- and at 10% from a $N(0, 10)$ distribution, with sample size $n = 1, \dots, 10$. Compare the distributions from the simulation with the corresponding normal-, χ^2- resp. t distribution.</p> |

The t test has some robustness which can give it approximative validity. There are however ways to free us completely from the normal distribution assumption. If we proceed as in the F test resp. t test, but use ranks instead of the original data, we get test procedures that are distribution independent (at least as long as not ties occur). The Wilcoxon test is a distribution independent variant of the t test. Theoretically corresponds exactly to a t test, applied to the (jointly) rank transformed data. Like the t test, this test only designed to test the null hypothesis (no difference) against a shift alternative. For the practical application, arithmetic simplifications can be used. This may hide the relation between the usual formula for the t test and for the Wilcoxon test.

To apply the Wilcoxon test, on the one hand the test statistics has to be calculated. On the other hand, to determine the critical values, the distribution function has to be evaluated. If all observations are distinct, this function depends only on n_1 and n_2 , and fairly simple algorithms are available. These are provided in the R base and used by `wilcox.test()`. However if there are ties in the data, that is there are values occurring more than once, the distribution depends on the special pattern of these ties and the calculation is laborious. `wilcox.test()` resorted to approximations in this case. For an exact evaluation (in contrast to approximative), the necessary algorithms are available as well. To use them, you need `library(coin)`. The exact variant of the Wilcoxon tests for example is implemented as `wilcox_test()`.

It is a classical branch in statistics to characterise distribution independent methods based on ranks and to compare them with independent methods based on Monte Carlo calculations and their variations. You find literature for this area under the key words “rank tests” and “distribution free methods”. Additional R functions are provided in `library(coin)` as well as in some specialised libraries.

Of course information loss is a concern. If we restrict to the data and make no or weak assumptions on the distributions, we have less information than in a model with explicit distribution assumptions. If we reduce the data to ranks, we may give away additional information. This loss in information can be measured for instance in terms of asymptotic relative efficiency, that is the (asymptotic) sample size proportion which is needed to get a comparative power using the competing test. Under the assumption of normality, the asymptotic relative efficiency of Wilcoxon test is 94%. So if the assumption of a normal distribution holds, the (optimal) t only needs 94% of the sample size needed by the Wilcoxon test. 6% of the sample size are the costs for the reduction to ranks. But if the normal distribution does not apply, the t test possibly can break down, while the Wilcoxon test stay a valid test for the shift alternative.

[help\(wilcox.test\)](#)

[wilcox.test](#)

Wilcoxon Rank Sum and Signed Rank Tests

Description

Performs one and two sample Wilcoxon tests on vectors of data; the latter is also known as ‘Mann-Whitney’ test.

```
Test!Wilcoxon
wilcox.test@wilcox.test|t|
tie
wilcox.test@wilcox.test|t|
exact|test
test!exact
wilcox_test@wilcox_test|t|
wilcox.test@wilcox.test|t|
Topic
ht-
ToDo: !wilcox.test@wilco
add power
function
```

ToDo:
clarify: in-
formation
 $<->$ as-
sumptions

ToDo:
add power
simula-
tion - see
Rnw.tex
source

```
model.frame@model.frame|textit
Usage
  wilcox.test(x, ...)

  ## Default S3 method:
  wilcox.test(x, y = NULL,
              alternative = c("two.sided", "less", "greater"),
              mu = 0, paired = FALSE, exact = NULL, correct = TRUE,
              conf.int = FALSE, conf.level = 0.95, ...)

  ## S3 method for class 'formula':
  wilcox.test(formula, data, subset, na.action, ...)
```

Arguments

| | |
|--------------------|---|
| x | numeric vector of data values. Non-finite (e.g. infinite or missing) values will be omitted. |
| y | an optional numeric vector of data values. |
| alternative | a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. |
| mu | a number specifying an optional location parameter. |
| paired | a logical indicating whether you want a paired test. |
| exact | a logical indicating whether an exact p-value should be computed. |
| correct | a logical indicating whether to apply continuity correction in the normal approximation for the p-value. |
| conf.int | a logical indicating whether a confidence interval should be computed. |
| conf.level | confidence level of the interval. |
| formula | a formula of the form lhs ~ rhs where lhs is a numeric variable giving the data values and rhs a factor with two levels giving the corresponding groups. |
| data | an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula . By default the variables are taken from environment(formula) . |
| subset | an optional vector specifying a subset of observations to be used. |
| na.action | a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action") . |
| ... | further arguments to be passed to or from methods. |

Details

The formula interface is only applicable for the 2-sample tests.

If only **x** is given, or if both **x** and **y** are given and **paired** is **TRUE**, a Wilcoxon signed rank test of the null that the distribution of **x** (in the one sample case) or of **x-y** (in the paired two sample case) is symmetric about **mu** is performed.

Otherwise, if both **x** and **y** are given and **paired** is **FALSE**, a Wilcoxon rank sum test (equivalent to the Mann-Whitney test: see the Note) is carried out. In this case, the null

hypothesis is that the distributions of `x` and `y` differ by a location shift of `mu` and the alternative is that they differ by some other location shift.

By default (if `exact` is not specified), an exact p-value is computed if the samples contain less than 50 finite values and there are no ties. Otherwise, a normal approximation is used. Optionally (if argument `conf.int` is true), a nonparametric confidence interval and an estimator for the pseudomedian (one-sample case) or for the difference of the location parameters `x-y` is computed. (The pseudomedian of a distribution F is the median of the distribution of $(u+v)/2$, where u and v are independent, each with distribution F . If F is symmetric, then the pseudomedian and median coincide. See Hollander & Wolfe (1973), page 34.) If exact p-values are available, an exact confidence interval is obtained by the algorithm described in Bauer (1972), and the Hodges-Lehmann estimator is employed. Otherwise, the returned confidence interval and point estimate are based on normal approximations.

With small samples it may not be possible to achieve very high confidence interval coverages. If this happens a warning will be given and an interval with lower coverage will be substituted.

Value

A list with class "htest" containing the following components:

| | |
|--------------------------|---|
| <code>statistic</code> | the value of the test statistic with a name describing it. |
| <code>parameter</code> | the parameter(s) for the exact distribution of the test statistic. |
| <code>p.value</code> | the p-value for the test. |
| <code>null.value</code> | the location parameter <code>mu</code> . |
| <code>alternative</code> | a character string describing the alternative hypothesis. |
| <code>method</code> | the type of test applied. |
| <code>data.name</code> | a character string giving the names of the data. |
| <code>conf.int</code> | a confidence interval for the location parameter. (Only present if argument <code>conf.int = TRUE</code> .) |
| <code>estimate</code> | an estimate of the location parameter. (Only present if argument <code>conf.int = TRUE</code> .) |

Warning

This function can use large amounts of memory and stack (and even crash R if the stack limit is exceeded) if `exact = TRUE` and one sample is large (several thousands or more).

Note

The literature is not unanimous about the definitions of the Wilcoxon rank sum and Mann-Whitney tests. The two most common definitions correspond to the sum of the ranks of the first sample with the minimum value subtracted or not: R subtracts and S-PLUS does not, giving a value which is larger by $m(m+1)/2$ for a first sample of size m . (It seems Wilcoxon's original paper used the unadjusted sum of the ranks but subsequent tables subtracted the minimum.)

R's value can also be computed as the number of all pairs $(x[i], y[j])$ for which $y[j]$ is not greater than $x[i]$, the most common definition of the Mann-Whitney test.

```

psignrank@psignrank\textit{References}
pwilcox@pwilcox\textit{References}
wilcox.exact@wilcox\textit{exact}
kruskal.test@kruskal\textit{test}
t.test@t.test\textit{t.test}

Myles Hollander & Douglas A. Wolfe (1973), Nonparametric statistical inference. New York: John Wiley & Sons. Pages 27–33 (one-sample), 68–75 (two-sample).

Or second edition (1999).
David F. Bauer (1972), Constructing confidence sets using rank statistics. Journal of the American Statistical Association 67, 687–690.

```

See Also

psignrank, **pwilcox**.

wilcox.exact in **exactRankTests** covers much of the same ground, but also produces exact p-values in the presence of ties.

kruskal.test for testing homogeneity in location parameters in the case of two or more samples; **t.test** for an alternative under normality assumptions [or large samples]

Examples

```

## One-sample test.
## Hollander & Wolfe (1973), 29f.
## Hamilton depression scale factor measurements in 9 patients with
## mixed anxiety and depression, taken at the first (x) and second
## (y) visit after initiation of a therapy (administration of a
## tranquilizer).
x <- c(1.83, 0.50, 1.62, 2.48, 1.68, 1.88, 1.55, 3.06, 1.30)
y <- c(0.878, 0.647, 0.598, 2.05, 1.06, 1.29, 1.06, 3.14, 1.29)
wilcox.test(x, y, paired = TRUE, alternative = "greater")
wilcox.test(y - x, alternative = "less")      # The same.
wilcox.test(y - x, alternative = "less",
            exact = FALSE, correct = FALSE) # H&W large sample
                                         # approximation

## Two-sample test.
## Hollander & Wolfe (1973), 69f.
## Permeability constants of the human chorioamnion (a placental
## membrane) at term (x) and between 12 to 26 weeks gestational
## age (y). The alternative of interest is greater permeability
## of the human chorioamnion for the term pregnancy.
x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
y <- c(1.15, 0.88, 0.90, 0.74, 1.21)
wilcox.test(x, y, alternative = "g")          # greater
wilcox.test(x, y, alternative = "greater",
            exact = FALSE, correct = FALSE) # H&W large sample
                                         # approximation

wilcox.test(rnorm(10), rnorm(10, 2), conf.int = TRUE)

## Formula interface.
boxplot(Ozone ~ Month, data = airquality)

```

```
wilcox.test(Ozone ~ Month, data = airquality,
subset = Month %in% c(5, 8))
```

a:ch03:11

| | |
|----------------------|--|
| Exercise 3.15 | |
| | Use the Wilcoxon test, to compare the results of the right/left <code>click</code> experiment. |

a:ch03:12

| | |
|----------------------|---|
| Exercise 3.16 | |
| *** | <p>In the the right/left <code>click</code> experiment several effects contribute to the response time. Some problems:</p> <ul style="list-style-type: none"> • The response time comprises reaction time, time for the large scale movement of the mouse, time for fine adjustment etc. • For the right/left movement in general a swivel of the hand is sufficient. For forward/backward movement in general a movement of the arm is necessary. It is not to be expected that both movements have a comparable statistical behaviour. • Subsequent record my be affected by a training effect, or by a tiring effect. <p>Can you modify the experiment or the evaluation so that differences in the reaction time component can be investigated?</p> <p>Can you modify the experiment or the evaluation so that differences in the precision of the position of the click can be investigated?</p> <p>Inspect and document for yourself the right/left differences in reaction time and precision. Summarise your results as a report..</p> |

a:ch03:13

| | |
|----------------------|---|
| Exercise 3.17 | |
| | <p>Use the shift/scale families of $N(0, 1)$ and $t(3)$ and design a setting to compare the performance of the Wilcoxon test with that of the t test for each of these families.</p> <p>Perform the comparison in a simulation with sample sizes $n_1 = n_2 = 10, 20, 50, 100$ and summarise your results.</p> <p>Do an analogous comparison using simulation data from the lognormal distribution.</p> |

3.4 Power and Confidence

3.4.1 Theoretical Power and Confidence

We can use the t tests to illustrate how tests are constructed in general. The test uses a test statistics, here the t test statistics for comparison of two groups (3.1). We know the distribution of this statistics: in case of the t test, given independent errors with normal distribution and equal variance the test statistics has a $t(n_1 + n_2 - 2)$ distribution. For given level α we can use the distribution function to read off limits which are exceeded resp missed only with a probability of α . If we use both limits we get a two sided region with error probability 2α .

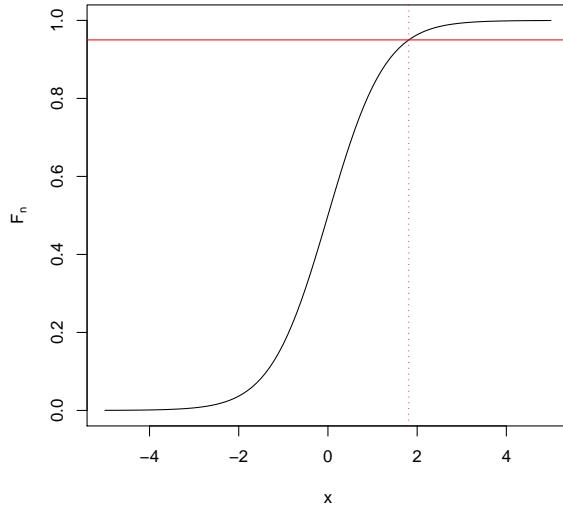
ToDo:
Simulation,
Power

fig:ch03testcrit

Example 3.4:

Input

```
n1 <- 6
n2 <- 6
df <- n1 + n2 - 2
alpha <- 0.05
curve(pt(x, df = df), from = -5, to = 5, ylab = expression(F[n]))
abline(h = 1 - alpha, col = "red")
abline(v = qt(1 - alpha, df = df), lty = 3, col = "red")
```



If, for example, we want to test the hypothesis $\mu_1 = \mu_2$ against the alternative $\mu_1 > \mu_2$, we use the region above the upper of these limits as rejection region. We knot that if the hypothesis applies, a random test statistics will fall into this region with probability of at most α .

ToDo:
introduce
noncentral
t in ch. 2

For the t test we know more. Under the model assumptions of independent errors with normal distribution and equal variance, the t test statistics always has a t distribution. On the

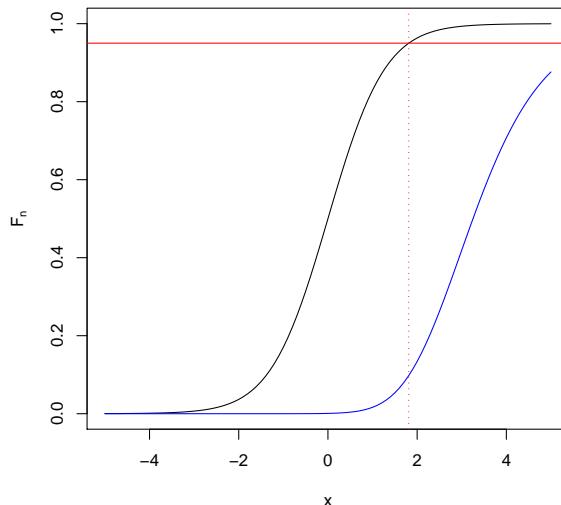
hypothesis it has a t distribution with non centrality parameter 0, that is it has a central t distribution. On the alternative where $\mu_1 \neq \mu_2$, but still for normal distribution with equal variance, we have a t distribution with non centrality parameter $(\mu_1 - \mu_2)\sigma^{-1}\sqrt{n_1 n_2}/(n_1 + n_2)$. Under model assumptions this gives the power of the test for each alternative, that is the probability to reject the hypothesis, if a specific alternative applies.

To Do:
one sided
or two
sided?

Example 3.5:

Input

```
n1 <- 6
n2 <- 6
df <- n1 + n2 - 2
alpha <- 0.05
curve(pt(x, df = df), from = -5, to = 5, ylab = expression(F[n]))
abline(h = 1 - alpha, col = "red")
abline(v = qt(1 - alpha, df = df), lty = 3, col = "red")
n1 <- 5
n2 <- 5
n <- n1 + n2
theta <- 2
ncp <- theta * sqrt(n1 * n2/(n1 + n2))
curve(pt(x, df = df, ncp = ncp), add = TRUE, col = "blue")
```



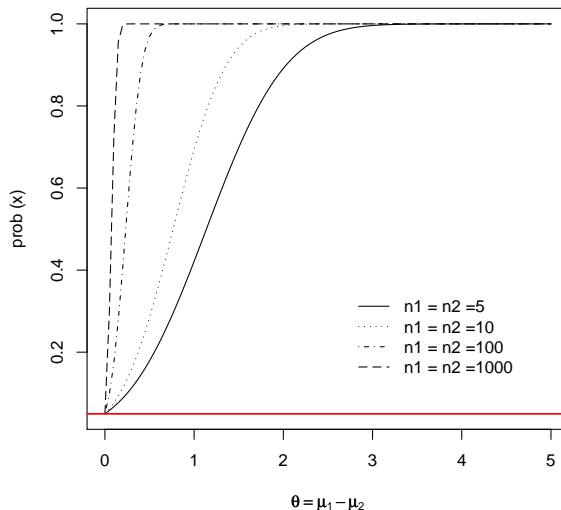
We can display the power of the test by plotting the rejection probability as a function of $(\mu_1 - \mu_2)\sigma^{-1}$.[†]

[†] Conventionally the term “power function” is only used for the rejection probability on the alternative, that is for example for $(\mu_1 - \mu_2) > 0$.

```

power.t.test@power.t.test$textit
Example 3.6: Input
tpower <- function(n1, n2, alpha, ...) {
  df <- n1 + n2 - 2
  tlim <- qt(1 - alpha, df = df)
  prob <- function(theta) {
    pt(tlim, df = df, ncp = theta * sqrt(n1 *
      n2/(n1 + n2)), lower.tail = FALSE)
  }
  curve(prob, 0, 5, xlab = expression(theta ==
    mu[1] - mu[2]), ...)
  abline(h = alpha, col = "red")
}
tpower(5, 5, 0.05)
tpower(10, 10, 0.05, add = TRUE, lty = 3)
tpower(100, 100, 0.05, add = TRUE, lty = 4)
tpower(1000, 1000, 0.05, add = TRUE, lty = 5)
legend("bottomright", lty = c(1, 3, 4, 5), legend = c("n1 = n2 =5",
  "n1 = n2 =10", "n1 = n2 =100", "n1 = n2 =1000"),
  inset = 0.1, bty = "n")

```



This relation can also be used to calculate the sample size needed to reject the hypothesis at with a probability α if it applies (false rejection), but to reject the hypothesis if some specified alternative applies (power or correct rejection). Function `power.t.test()` is already prepared to do this calculation.

ch03powert

Example 3.7:

| | |
|---|--------------|
| <code>power.t.test(delta = 2, power = 0.8, sig.level = 0.01, type = "two.sample", alternative = "one.sided")</code> | <i>Input</i> |
|---|--------------|

| | |
|--|---------------|
| <code>Two-sample t test power calculation</code> | <i>Output</i> |
|--|---------------|

```
n = 6.553292
delta = 2
sd = 1
sig.level = 0.01
power = 0.8
alternative = one.sided
```

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

3.4.2 Simulated Power and Confidence

If the theoretical properties of a test are known, this is the best way to analyse its properties. In an environment as R we have the possibility to assess the power even if theoretical results are not available or not accessible. For chosen alternatives can we generate random samples, perform tests and find the relative proportion of samples that lead to a rejection. If we generate $nsimul$ independent random samples with identical distribution, the number of rejections is a random variable with binomial distribution and

$$\hat{p} = \frac{\#rejections}{nsimul}$$

is an estimator for the rejection probability.

As an example we see how the t test performs if the data have a lognormal distribution. We compare two groups, each with sample size $n_1 = n_2 = 10$. We generate data under the hypothesis first:

ch03tlognH

```
prop.test@prop.test;textit
Example 3.8: _____ Input _____
nsimul <- 300
n1 <- 10
n2 <- 10
alpha <- 0.01
x <- 0
for (i in 1:nsimul) {
  if (t.test(exp(rnorm(n1)), exp(rnorm(n2)), alternative = "less",
    var.equal = TRUE)$p.value < alpha) {
    x <- x + 1
  }
}
p <- x/nsimul
cat("estim p", p)

_____ Output _____
estim p 0.01333333
```

The function `prop.test()` does not only calculate the estimator for the binomial fraction, but also a confidence set.

```
Example 3.9: _____ Input _____
prop.test(n = nsimul, x = x)

_____ Output _____
1-sample proportions test with continuity correction

data: x out of nsimul, null probability 0.5
X-squared = 282.27, df = 1, p-value < 2.2e-16
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.004279730 0.036111807
sample estimates:
  p
0.01333333
```

Analogous four example under the alternative with $\log(x_2)$ distributed as $N(\mu_2, 1)$ with $\mu_2 = 1$.

ch03tlognA

| | |
|---|---|
| Example 3.10: | <i>Input</i> |
| <pre> nsimul <- 300 n1 <- 10 n2 <- 10 alpha <- 0.01 x <- 0 for (i in 1:nsimul) { if (t.test(exp(rnorm(n1)), exp(rnorm(n2, mean = 1)), alternative = "less", var.equal = TRUE)\$p.value < alpha) { x <- x + 1 } } p <- x/nsimul cat("estim p", p) </pre> | <i>Output</i> |
| estim p 0.18 | |
| <i>Input</i> | <i>Output</i> |
| <pre>prop.test(n = nsimul, x = x)</pre> | <pre> 1-sample proportions test with continuity correction data: x out of nsimul, null probability 0.5 X-squared = 121.6033, df = 1, p-value < 2.2e-16 alternative hypothesis: true p is not equal to 0.5 95 percent confidence interval: 0.1391668 0.2292316 sample estimates: p 0.18 </pre> |

In `library(binom)` several tools are provided for a differentiated analysis of the binomial distribution.

The confidence intervals in this example show that a simulation sample size of $nsimul = 300$ only gives rough results. For simulations, we want better control over the simulation precision. We can choose the simulation sample size to improve the precision as we like. A sample size calculation can be done with `power.prop.test()`. In experiments, an exact sample size calculation may be necessary. For simulations, often an estimation of the sample size is sufficient.

With $\hat{p} := Z/n$ as estimator for a probability p we have $E(\hat{p}) = p$ and $Var(\hat{p}) = p(1 - p)/n$. If p is actually the value of the parameter, an error has to be judged relative to this target parameter. At $p = 50\%$ an error of $\pm 1\%$ has to be judged differently than at $p = 99\%$. The relative error, the *coefficient of variation*, is

$$\frac{\sqrt{Var(\hat{p})}}{E(\hat{p})} = \sqrt{\frac{1-p}{np}}.$$

```

power.prop.test@power.prop
coefficient
of
variation|textbf

```

To get a coefficient of variation of at most η , we need a sample size

$$n \geq \frac{1-p}{p\eta^2}.$$

If n and p are in an order of magnitude where a normal approximation applies, we have an approximative confidence interval with limits

$$\hat{p} \pm \phi_{1-\alpha/2} \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \text{ for a confidence level } 1-\alpha$$

If the length of the confidence interval should not exceed ηp , we need a sample size

$$n \geq \frac{\phi_{1-\alpha/2}^2(1-p)}{p\eta^2}.$$

As usual, the choice of α is up to us. For example for $\alpha = 1\%$ with $\phi_{1-\alpha/2} = 2.575829$ we get the values in table 3.28. If we work with higher quantiles, we will try to bound the error relative to $1-p$. Examples are in table 3.28.

ToDo: add
just gray
formats in
tables

`tab:simulneta`

Table 3.28 Requires sample size for two sided confidence intervals with relative length $\leq \eta$

| p | $1-p$ | $n(\alpha = 10\%)$ | $\eta = 0.1$ | $n(\alpha = 1\%)$ | $\eta = 0.1$ | $\eta = 0.01$ |
|-------|-------|--------------------|---------------|-------------------|---------------|---------------|
| | | $\eta = 0.1$ | $\eta = 0.01$ | $\eta = 0.1$ | $\eta = 0.01$ | |
| 0.500 | 0.500 | 271 | 27 055 | 663 | 66 349 | |
| 0.250 | 0.750 | 812 | 81 166 | 1990 | 199 047 | |
| 0.100 | 0.900 | 2435 | 243 499 | 5971 | 597 141 | |
| 0.010 | 0.990 | 26 785 | 2 678 488 | 65 685 | 6 568 547 | |
| 0.001 | 0.999 | 270 283 | 27 028 379 | 662 826 | 662 822 617 | |

It is worth to memorise the rough numbers: To estimate a probability in the range of magnitude of $50\% \pm 5\%$ with 90% confidence, about 300 simulations are necessary. To estimate a value of about 99% up to $\pm 0.1\%$, we need 30000 simulations.

3.4.3 Quantile estimation by Simulation

The other side of the problem above is to estimate a quantile from a random sample. We already know that for a random variable X with continuous distribution function F the variable $F(X)$ has a uniform distribution on $[0, 1]$. For the quantile estimation we need the distribution function, evaluated at the order statistics. We already met this in chapter ??, where we had theorem ??.

`thm:simulF`

Theorem 3.2 For independent observations $X_i, i = 1, \dots, n$ from a continuous distribution function F , let $X_{(k:n)}$ denote the k th order statistics. Then

$$F(X_{(k:n)}) \sim P_{beta}(\cdot; k, n-k+1).$$

We repeat:

Remark 3.3 In general the beta distribution is skewed. The expected value of the $\text{beta}(k, n - k + 1)$ distribution is $k/(n + 1)$. For an unbiased estimation of the quantile x_p use $X_{(k:n)}$ with $k/(n + 1) = p$. The “plug in” approximation $k/n = p$ gives a biased estimation.

The theorem can be applied directly to get an upper or lower estimate for quantiles. As a special case we can try to use the minimum of the observed values $X_{(1:n)}$ as lower estimate for the p quantile. The confidence level is

$$P(X_{(1)} \leq F_p) = P(F(X_{(k)}) \leq p) = I_p(1, n),$$

where I is the incomplete beta integral. For the special parameters $(1, n)$ the beta density simplifies to $n(1 - p)^{n-1}$ and we get the incomplete beta integral $I_p(1, n) = 1 - (1 - p)^n$. Hence

$$P(X_{(1)} \leq F_p) = 1 - (1 - p)^n$$

and we can guarantee a confidence level $1 - \alpha$, if

$$n \geq \frac{\ln \alpha}{\ln(1 - p)}.$$

Similarly, the observed maximum value can be used as an upper estimate for the p -quantile. By symmetry we get a confidence level of $1 - \alpha$, if

$$n \geq \frac{\ln \alpha}{\ln p}.$$

Examples are given in table 3.29. tab:simulnquant

Table 3.29 Required sample size for estimation of a quantile with confidence level $\geq 1 - \alpha$

| p | | n | | | |
|--------------------|--------------------|-----------------|----------------|----------------|------------------|
| $X_{(1)} \leq F_p$ | $X_{(n)} \geq F_p$ | $\alpha = 10\%$ | $\alpha = 5\%$ | $\alpha = 1\%$ | $\alpha = 0.5\%$ |
| 0.500 | 0.500 | 4 | 5 | 7 | 8 |
| 0.250 | 0.750 | 9 | 11 | 17 | 19 |
| 0.100 | 0.900 | 22 | 29 | 44 | 51 |
| 0.010 | 0.990 | 230 | 299 | 459 | 528 |
| 0.001 | 0.999 | 2302 | 2995 | 4603 | 5296 |

Again it is worth to memorise the rough numbers: to get a one sided estimate for a 1% (99%)-quantile of a continuous distribution function with a confidence of 99%, about 500 simulations are needed.

We can combine one sided limits to intervals. The corresponding result to calculate the probability of intervals is in corollary ???: cor:beta2s

Corollary 3.4 For the k_1 th and $k_1 + k_2$ th order statistics, the interval $(X_{(k_1:n)}, X_{(k_1+k_2:n)})$ is a confidence interval for the p quantile with coverage probability

$$I_p(k_1, n - k_1 + 1) - I_p(k_1 + k_2, n - k_1 - k_2 + 1).$$

cor:sumul2s

The simulation sample sized for quantile estimation are considerably smaller than those needed for the estimation of comparable probabilities. In retrospect, this is not surprising: the question whether an observation exceeds some specific quantile is simpler the task to estimate the p -value. The sample size can often reduces even more drastically if the question at hand is reduced to a test problem.

Without additional distribution assumptions this gives a first possibility to chose the size of a simulation. In special situation, cute ideas can allow for a drastic reduction of the simulation sample size. But for a start, the calculations given above provide a basis for determining the simulation sample size.

3.5 Qualitative Features of Distributions

ToDo:
Qualitative
features

3.6 R Complements

s:3.4



3.7 Statistical Summary

As a leading example in this chapter we use the comparison of treatments. In simple cases, the samples under the treatments to be compared differ only by a shift in mean. In this case the problems can be reduced to the approaches of chapter 2. In this reduced case the distributions coincide when centred at the mean. For the general case which we touched here the simplification does not work. An important example is the analysis of therapy studies. If a treatment has a homogenous effect, we can analyse it using the methods of chapter 2. But often under treatment we see a split into “responders” and “non responders”. This goes beyond the models sketched in chapter 2 and requires more general approaches mentioned in this chapter 3.

We restricted our discussion to the comparison of two samples. Practical work often leads to other problems. For example, a typical question is to compare a new treatment with a known reference treatment, where only sample observation is available for the new treatment, but extensive prior information is available about the reference treatment. Or a reference treatment is to be compared with a series of alternative treatments. These problems go beyond the scope of our introduction. Here we can only refer to advanced literature, for example [12].

3.8 Literature and Additional References

- [25] Venables, W.N.; Ripley, B.D. (2002): Modern Applied Statistics with S. Heidelberg: Springer
- [24] Venables, W.N.; Ripley, B.D. (2000): S Programming. Heidelberg: Springer
- [12] Miller, R. G. (1981): Simultaneous Statistical Inference. Heidelberg: Springer

```
Generated by Sweave from:
Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/Rnw/S03vergl.Rnw.tex,v
Revision: 1.17
Date: 2008/03/18 09:01:45
name:
Author: j40
$Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/Rnw/S03vergl.Rnw.tex,v $
$Revision: 1.17 $
$Date: 2008/03/18 09:01:45 $
$name: $
$Author: j40 $
```



CHAPTER 4

Dimensions 1, 2, 3, ..., ∞

ch:04

ToDo:
add mds

4.1 R Complements

In this chapter we begin with complements on R in order to concentrate on statistical questions for the remainder. We have a look on the graphical possibilities that are at our disposition. The basic graphics model of R is oriented at to possibilities of a plotter as an output device. The graphics follow the possibilities available when drawing with a pen. Besides the one and two dimensional possibilities which we have seen so far, there are possibilities to display a function which is defined over a grid. Basically three functions are available.

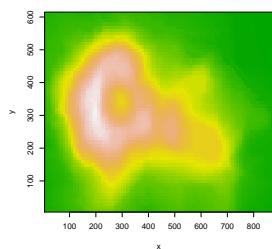
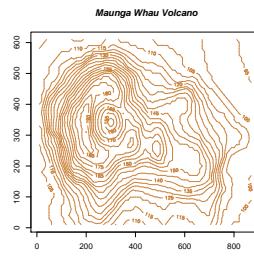
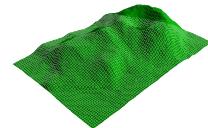
| <i>3d Graphic</i> | |
|-------------------|--|
| <i>image()</i> | gives the values of a variable <i>z</i> in grey levels or colour coding. |
| <i>contour()</i> | gives the contours of a variable <i>z</i> . |
| <i>persp()</i> | gives a perspective plot of a variable <i>z</i> . |

ch04-SframeVolcano *image()* and *contour()* can also be used to give an overlay on other plots.

```
print@print|textit
Example 4.1:
```

Input

```
x <- 10 * (1:nrow(volcano))
y <- 10 * (1:ncol(volcano))
image(x, y, volcano, col = terrain.colors(100), axes = FALSE)
axis(1, at = seq(100, 800, by = 100))
axis(2, at = seq(100, 600, by = 100))
box()
title(main = "Maunga Whau Volcano", font.main = 4)
contour(x, y, volcano, levels = seq(90, 200, by = 5),
        col = "peru", main = "Maunga Whau Volcano", font.main = 4)
z <- 2 * volcano
x <- 10 * (1:nrow(z))
y <- 10 * (1:ncol(z))
persp(x, y, z, theta = 135, phi = 30, col = "green3",
      scale = FALSE, ltheta = -120, shade = 0.75, border = NA,
      box = FALSE)
```

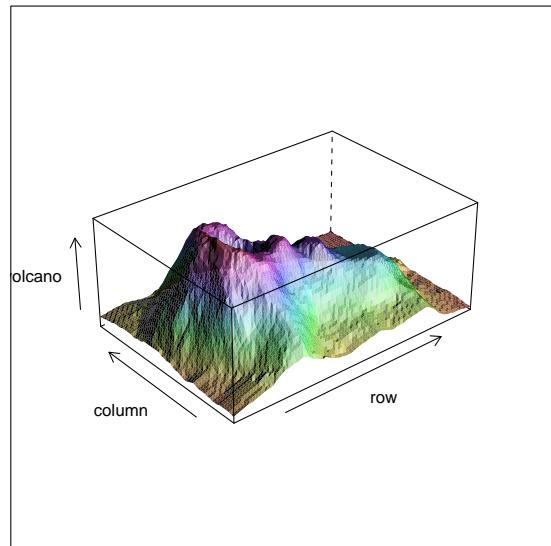
*image()**contour()**persp()*

The basic graphic system is easy to use, but limited in possibilities. A newer graphic system conceptually works with objects and a camera model. The graphic objects can be combined and post-processed. The display takes part in a separate step. Simple 2d graphs can be post-processed. For a 3d display distance, point of view and focal length can be chose like when using a camera. The object oriented graphic system consists of a library *grid* with the elementary operations required, and a higher level library *lattice* which gives new implementation of the displays known from the basic graphics and adds additional displays.

xpl:ch04-wireframe For the output of lattice objects the function *print()* is used.

Example 4.2:

```
library(lattice)
print(wireframe(volcano, shade = TRUE, aspect = c(61/87,
0.4), light.source = c(10, 0, 10)))
```



The basic graphic system and lattice graphics are separate graphics systems. Unfortunately they use different notations for comparable functions, and comparable displays have different representations. A small translation aid is given in table 4.4. Some convenience functions to combine both graphics systems are provided in library `gridBase`. An extensive introduction to both graphics systems is [13].

In a wide range of scientific visualisations, OpenGL is used as a common standard. OpenGL functions are accessible in R using the library `xgll`. There are however certain differences between common requirements for graphics, and the specific requirements of statistical graphics. As far as the representation of functions is concerned, statistical graphics is comparable with the requirements usual in analysis. The small difference is that functions in statistics are often piecewise constant or only piecewise continuous, while for example in analysis continuous or even in differentiable functions are the rule rather than the exception. When it comes to displaying data, the situation changes drastically. Usually, statistical data are discrete. Smoothness properties that simplify display of analytical data are not available for statistical data. So visualisations adapted to the needs of statistics are required.

To Do:
 discuss
 alpha
 channel

4.2 Dimensions

If we move from one dimension to higher dimensions, there are challenges for theoretical considerations as well as for graphical displays. Again, linear models can serve as leading or warning examples.

| Basic Graphics | | Lattice |
|-------------------------------|-----------------------------|----------------------------|
| <code>barplot()</code> | bar chart | <code>barchart()</code> |
| <code>boxplot()</code> | box and whisker plot | <code>bwplot()</code> |
| | 3 dimensional scatter plot | <code>cloud()</code> |
| <code>contour</code> | contour plot | <code>contourplot()</code> |
| <code>coplot</code> | conditional scatter plots | <code>contourplot()</code> |
| <code>curve(density())</code> | density estimator | <code>densityplot()</code> |
| <code>dotchartt()</code> | dot plot | <code>dotplot()</code> |
| <code>hist()</code> | dot plot | <code>histogram()</code> |
| <code>image()</code> | colour map plots | <code>splom()</code> |
| | parallel coordinate plots | <code>parallel()</code> |
| <code>pairs()</code> | scatter plot matrices | <code>wireframe()</code> |
| <code>persp()</code> | three dimensional surface | <code>wireframe()</code> |
| <code>plot()</code> | scatter plot | <code>xyplot()</code> |
| <code>qqnorm()</code> | theoretical $Q - Q$ -plot | <code>qqmath()</code> |
| <code>qqplot()</code> | empirical $Q - Q$ -plot | <code>qq()</code> |
| <code>stripchart()</code> | one dimensional scatterplot | <code>stripplot()</code> |

Table 4.4 *Basic Graphics and Lattice Graphics*

ta:ch04lattice

The challenge may result from serious problems. Even under regularity conditions, the distribution on higher dimensional spaces can be complex beyond limits. The classification identification problems for functions and spaces from analysis and geometry give a foretaste of what has to be coped with for the analysis of probability distributions.

Besides, there are home made problems: own goals, which are only generated by choices made in a first step.

An example for home made problems can be illustrated for linear models. The interpretation of the Gauß-Markov estimator as a linear projection shows that only seemingly the coefficients for single regressors are estimated. In fact a vector in the vector space spanned by the regressors is estimated; the attribution to individual regressors is mere linear algebra. The allocation to individual components does not depend on the influence of the individual regressor, but on the joint geometry of all the regressors. Only if the regressors form an orthogonal basis there is a direct interpretation of the coefficients. For example if we duplicate the list of regressors in a linear model, the space does not change. The calculation in coordinates become more complicated because the regressors do in no case form a basis now, but from an abstract point of vie, the information is unchanged. However if there is no exact duplicate, but minute deviations (by minimal “error”, rounding, transformations), the situation changes drastically. With respect to the Gauß-Markov estimator, only the space spanned by the regressors is relevant, and even minimal changes in the duplicate copy can duplicate the dimension of this space. This is a example for a home made problem.

This and other examples are a reason enough to analyse the relation between the variables more precisely. In the case of regression for example this does not only refer to the relation between response and regressors, but, as illustrated by the recent example, the relations between the regressors.

**linking|textbf
brushing|textbf**

To keep the relation to the regression problems and resort to experiences in this area, we use a formal imbedding of the regression problem in a more general framework. In regression, we had a singled out variable, the response. The distribution of this variable was to be modelled as a function of the remaining variables, the regressors X . Now we comprise response and regressor to a joined data vector $Z = (Y; X)$ and will discuss the joined distribution of Z . We retrieve the regression problem in this more general framework: for the regression problem we were looking for an estimator for the mean value function m in the model

$$Y = m(X) + \varepsilon.$$

In the general framework we consider the joint distribution of X and Y . The regression model becomes

$$Y = E(Y|X) + \varepsilon$$

and for now we have the identification $m(X) = E(Y|X)$.

If we are indeed interested in the a pristine regression model, additional work has to be done. An estimation of the conditional expected value $E(Y|X)$ is not the same as the estimation of a regression function $m(X)$. In the regression problem we have made no assumptions about the distribution of X . To infer from $E(Y|X)$ (or a estimator of this) to $m(X)$ we must establish that the estimation is independent of the distribution assumptions about X . for our temporary purposes the distinction is irrelevant. We can allow us a temporary ignorance.

So the general frame in this chapter is:

we analyse data $(Z_i)_{i=1,\dots,n}$, where the single observations take values in \mathbb{R}^q .

If we have essentially linear structures, we often can analyse higher dimensional structures with methods that have been developed for one dimensional models. We may have to modify these methods, or use them stepwise taking residuals. But they may help us to detect the essential structures. They fail however if higher dimensionality combines with non linearity. In that case, more specific methods are needed.

4.3 Selections

Originally a selection indicated a selection of observations. For the graphical display a selection is associated with a specification of display attributes (such as colour, plot character, line width). All variable values that belong to observations in the selection are displayed with these attributes. This allows to trace (“**linking**”) by selection in various plots. With linking, selections can help to detect structures in linked plots.

In practical data analysis, selections are varied dynamically (“**brushing**”), to group corresponding observations. Brushing with linked plots is an important tool in interactive data analysis. In statistical terms, selections are used for model selection. They correspond to local models: instead of using a global, possibly rather complex for the data, for each selection a (hopefully simpler) model is used which only applies to the data in this selection.

Unfortunately linking is not supported directly by R. We ourselves have to ensure that selections are displayed using their accordant attributes. Moreover the syntactic representation of

| Name | Variable | Unit, Remarks |
|------|---|---------------------------------|
| rw | relative body weight | |
| fpg | plasma glucose (after overnight fast) | [mg/100 ml] |
| ga | glucose level integrated over about 3 hours tolerance test | [mg/100 ml \times h] |
| ina | insulin level integrated over about 3 hours tolerance test | [μ U/100 ml \times h] |
| sspg | plasma glucose (steady state) | [mg/100 ml] |
| cc | classification | chemical, normal, open diabetes |

Table 4.5 *Diabetes data set: variables***diabvars**

selections is not unified. In function calls, selections can be represented as a **selection** parameter, or by a **group**-Variable, or as a condition in a formula expression. So in each case we have to resume to ad-hoc solutions.

R is mainly restricted to static selections. Brushing is only possible in a rudimentary form.

Selections are illustrated in context in the following sections.

4.4 Projections

As a first example we look at a data set published in a comparison of different kind of diabetes ([\[20\]](#)). The data set is provided for example in `library(locfit) as data(chemdiab)`. The variables refer to laboratory values on glucose metabolism (see table 4.5).

Input

```
library(locfit)
data(chemdiab)
```

We get a first summary with

Input

```
summary(chemdiab)
```

| Output | | |
|----------------|---------------|----------------|
| rw | fpg | ga |
| Min. :0.7100 | Min. : 70.0 | Min. : 269.0 |
| 1st Qu.:0.8800 | 1st Qu.: 90.0 | 1st Qu.: 352.0 |
| Median :0.9800 | Median : 97.0 | Median : 413.0 |
| Mean :0.9773 | Mean :122.0 | Mean : 543.6 |
| 3rd Qu.:1.0800 | 3rd Qu.:112.0 | 3rd Qu.: 558.0 |

```

Max.    :1.2000   Max.    :353.0   Max.    :1568.0
      ina          sspg            cc
Min.    :10.0     Min.    :29.0   Chemical_Diabetic:36
1st Qu.:118.0   1st Qu.:100.0  Normal       :76
Median  :156.0   Median  :159.0  Overt_Diabetic  :33
Mean    :186.1   Mean    :184.2
3rd Qu.:221.0   3rd Qu.:257.0
Max.    :748.0   Max.    :480.0

```

```

distribution!marginal|text
scatter
plot
matrix|textbf
pairs@pairs|textit
pairs@pairs|textbf
Topic
hplot!pairs@pairs

```

As in the original publication we omit the relative weight from our discussion. The chemical classification *cc* is derived from the metabolic data. So it does not contain any information on its own. For orientation, we use it as classification marker, that is we use the selections *cc* = *Chemical_Diabetic*, *Normal*, *Overt_Diabetic*. Essentially the data set is four dimensional with the variables *fpg*, *ga*, *ina*, *sspg*.

4.4.1 Marginal Distributions and Scatter Plot Matrices

We can try to analyse multi dimensional distributions by looking at the two dimensional *marginal distributions* for all pairs of variables. The corresponding graphical display is called a *scatter plot matrix*. In R, it is implemented as function *pairs()*.

help(pairs)

| | |
|--------------|-----------------------------|
| pairs | <i>Scatterplot Matrices</i> |
|--------------|-----------------------------|

Description

A matrix of scatterplots is produced.

Usage

```

pairs(x, ...)

## S3 method for class 'formula':
pairs(formula, data = NULL, ..., subset,
      na.action = stats::na.pass)

## Default S3 method:
pairs(x, labels, panel = points, ...,
      lower.panel = panel, upper.panel = panel,
      diag.panel = NULL, text.panel = textPanel,
      label.pos = 0.5 + has.diag/3,
      cex.labels = NULL, font.labels = 1,
      row1attop = TRUE, gap = 1)

```

`data.matrix@data.matrix$textit`
Arguments

| | |
|---------------------------------------|---|
| <code>x</code> | the coordinates of points given as numeric columns of a matrix or data frame. Logical and factor columns are converted to numeric in the same way that <code>data.matrix</code> does. |
| <code>formula</code> | a formula, such as <code>~ x + y + z</code> . Each term will give a separate variable in the pairs plot, so terms should be numeric vectors. (A response will be interpreted as another variable, but not treated specially, so it is confusing to use one.) |
| <code>data</code> | a <code>data.frame</code> (or list) from which the variables in <code>formula</code> should be taken. |
| <code>subset</code> | an optional vector specifying a subset of observations to be used for plotting. |
| <code>na.action</code> | a function which indicates what should happen when the data contain <code>NAs</code> . The default is to pass missing values on to the panel functions, but <code>na.action = na.omit</code> will cause cases with missing values in any of the variables to be omitted entirely. |
| <code>labels</code> | the names of the variables. |
| <code>panel</code> | <code>function(x, y, ...)</code> which is used to plot the contents of each panel of the display. |
| <code>...</code> | arguments to be passed to or from methods. Also, graphical parameters can be given as can arguments to <code>plot</code> such as <code>main</code> . <code>par("oma")</code> will be set appropriately unless specified. |
| <code>lower.panel, upper.panel</code> | separate panel functions to be used below and above the diagonal respectively. |
| <code>diag.panel</code> | optional <code>function(x, ...)</code> to be applied on the diagonals. |
| <code>text.panel</code> | optional <code>function(x, y, labels, cex, font, ...)</code> to be applied on the diagonals. |
| <code>label.pos</code> | y position of labels in the text panel. |
| <code>cex.labels, font.labels</code> | graphics parameters for the text panel. |
| <code>row1attop</code> | logical. Should the layout be matrix-like with row 1 at the top, or graph-like with row 1 at the bottom? |
| <code>gap</code> | Distance between subplots, in margin lines. |

Details

The ij th scatterplot contains `x[, i]` plotted against `x[, j]`. The “scatterplot” can be customised by setting panel functions to appear as something completely different. The off-diagonal panel functions are passed the appropriate columns of `x` as `x` and `y`: the diagonal panel function (if any) is passed a single column, and the `text.panel` function is passed a single (`x`, `y`) location and the column name.

The graphical parameters `pch` and `col` can be used to specify a vector of plotting symbols and colors to be used in the plots.

The graphical parameter `oma` will be set by `pairs.default` unless supplied as an argument.

A panel function should not attempt to start a new plot, but just plot within a given coordinate system: thus `plot` and `boxplot` are not panel functions.

By default, missing values are passed to the panel functions and will often be ignored within a panel. However, for the formula method and `na.action = na.omit`, all cases which contain a missing values for any of the variables are omitted completely (including when the scales are selected). (The latter was the default behaviour prior to R 2.0.0.)

Author(s)

Enhancements for R 1.0.0 contributed by Dr. Jens Oehlschlaegel-Akiyoshi and R-core members.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Examples

```

pairs(iris[1:4], main = "Anderson's Iris Data -- 3 species",
      pch = 21, bg = c("red", "green3", "blue") [unclass(iris$Species)])

## formula method
pairs(~ Fertility + Education + Catholic, data = swiss,
      subset = Education < 20, main = "Swiss data, Education < 20")

pairs(USJudgeRatings)

## put histograms on the diagonal
panel.hist <- function(x, ...)
{
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5) )
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks; nB <- length(breaks)
  y <- h$counts; y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col="cyan", ...)
}
pairs(USJudgeRatings[1:5], panel=panel.smooth,
      cex = 1.5, pch = 24, bg="light blue",
      diag.panel=panel.hist, cex.labels = 2, font.labels=2)

## put (absolute) correlations on the upper panels,
## with size proportional to the correlations.
panel.cor <- function(x, y, digits=2, prefix="", cex.cor)
{
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- abs(cor(x, y))

```

4-10

DIMENSIONS 1, 2, 3, ..., ∞

```
txt <- format(c(r, 0.123456789), digits=digits)[1]
txt <- paste(prefix, txt, sep="")
if(missing(cex.cor)) cex <- 0.8/strwidth(txt)
text(0.5, 0.5, txt, cex = cex * r)
}
pairs(USJudgeRatings, lower.panel=panel.smooth, upper.panel=panel.cor)
```

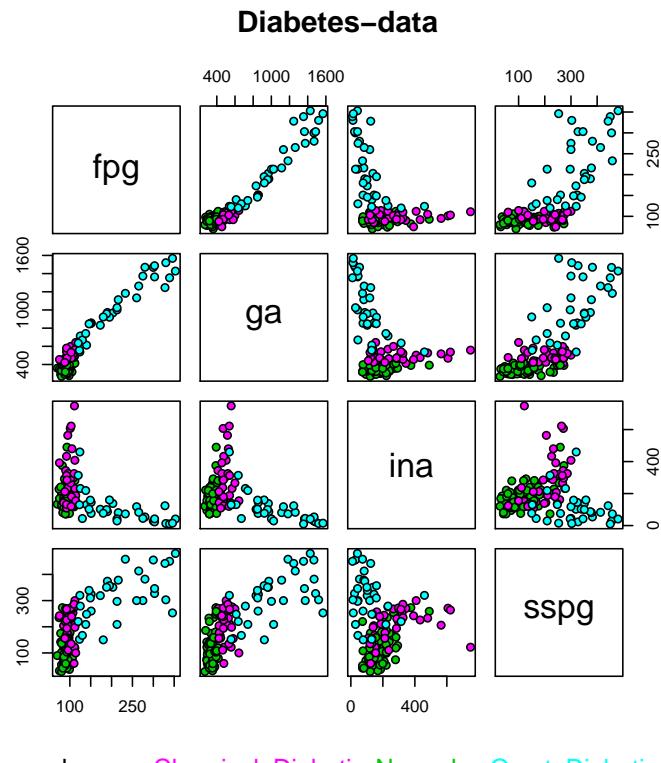
We use the chemical diabetes classes *cc* to define selections. We assign a colour to each of these selections. This is the linking attribute that allows us to trace relations between the plots. To provide a documentation of this linking, we have to control the graphics and modify the plots. Using the parameter *oma* we generate an outer margin for a caption.

expl:ch04-diabetes

Example 4.3:

```
Input
pairs(~fpg + ga + ina + sspg, data = chemdiab, pch = 21,
      main = "Diabetes-data", bg = c("magenta", "green3",
      "cyan")[unclass(chemdiab$cc)], oma = c(8,
      8, 8, 8)
mtext(c("Colour codes:", levels(chemdiab$cc)), col = c("black",
      "magenta", "green3", "cyan"), at = c(0.1, 0.4,
      0.6, 0.8), side = 1, line = 2)
```

```
pairs@pairs|textit
```



The `pairs()` function only controls the “layout” of the matrix, the selection and placement of the projections. The display in the individual plot tiles can be controlled in the function call. The defaults give the names of the variables in the diagonal tiles, and pair-wise scatter plots in the off diagonal tiles.

a:ch04:4

ToDo:
 supply
 better
 support
 for legend,
 colour key

| Exercise 4.1 | |
|--------------|---|
| | <p>Generate a scatter plot matrix for the diabetes data set that show a histogram of the variables in the diagonal tiles.</p> <p><i>Hint:</i> See <code>help(pairs)</code>.</p> |

Certain aspects of the distribution can be read easily from the marginal distributions. Other geometrical structures are barely or not at all recognisable from the marginal distribution.

For example, there is an apparent linear relation between the glucose level after a fast `fpg` and the integrate glucose level under tolerance test. This can be seen from the two dimension marginal distributions, and it can be analysed using the methods for linear models.

This apparent relation propagates to the relations to the other variables `ina`, `sspg`. For this reason, `fpg` is not considered in detail in the original paper. What remains to analyse are the variables `ga`, `ina`, `sspg`. The three dimensional structure of this part of the data set is not easily detected from the marginal distributions.

4.4.2 Projection Pursuit

Geometrical relations or stochastic dependencies that are not aligned along co-ordinate axes are in general not covered by marginal distributions. We can generalise the idea. Instead of two dimensional marginal distributions, we can use arbitrary projections. We use `library(lattice)` which implements a camera oriented graphic model.

The `grid` graphics and the `lattice` package provide support for multivariate representations. `grid` forms the basis. The original R graphics system implements a model that takes pen and paper as a model. A graphic port (paper) is opened, and lines, points or symbols are drawn. `grid` is a second graphics system which follows a camera/object model. Graphical objects are in various locations and orientations are mapped in a visual space. `lattice` builds upon `grid`. In <<http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/>> gives the underlying ideas for visualisation of multidimensional data that are implemented in `lattice`.

The graphics thus generated is output with `print()`. The parameter `split` allows to divide up the output area. Unfortunately linking is broken here: `cloud()` can generate a caption, but this shows the colour scale which was in effect when the graphics started, not the colour scale effectively used in the output. Once again we have to interfere with the system, this time to adjust the colour table.

Input

```

library("lattice")
diabcloud <- function(y, where, more = TRUE, ...) {
  print(cloud(ga ~ ina + spsg, data = chemdiab,
              groups = cc, screen = list(x = -90, y = y),
              distance = 0.4, zoom = 0.6, auto.key = TRUE,
              ...), split = c(where, 3, 2), more = more)
}
supsym <- trellis.par.get("superpose.symbol")
supsymold <- supersym
supsym$col = c("magenta", "green3", "cyan")
trellis.par.set(superpose.symbol = supersym)
diabcloud(y = 70, where = c(1, 1))

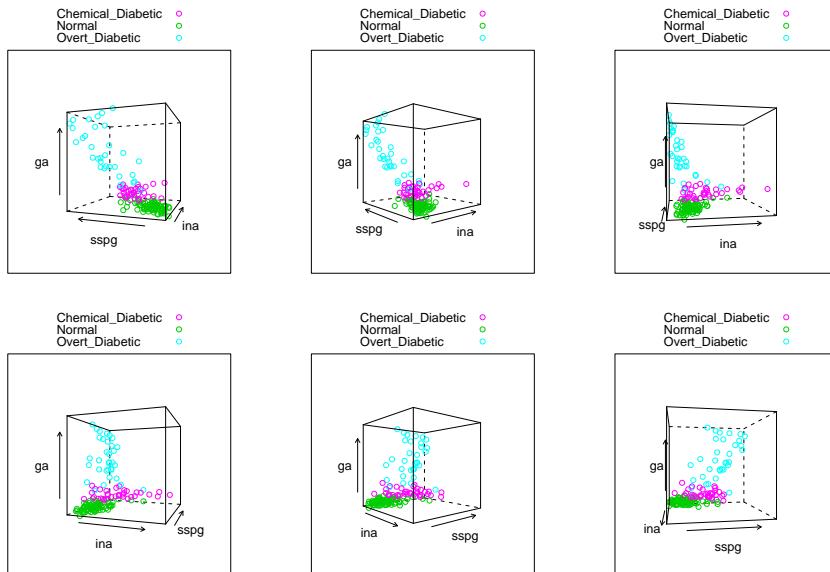
```

PROJECTIONS

4-13

```
diabcloud(y = 40, where = c(2, 1))
diabcloud(y = 10, where = c(3, 1))
diabcloud(y = -20, where = c(1, 2))
diabcloud(y = -50, where = c(2, 2))
diabcloud(y = -80, where = c(3, 2), more = FALSE)
trellis.par.set(superpose.symbol = supersymold)
rm(diabcloud, supersymold, supersym)
```

projection
pursuit|textbf



a:ch04:1

| Exercise 4.2 | |
|--------------|---|
| | <p>Modify this example so that you get an impression of the three dimensional structure.</p> <p>What is the difference between open diabetes and chemical diabetes?</p> <p>How does the normal group compare to both diabetes groups?</p> |

As with series of projections it is often not simple to identify a three dimensional structure. Animation can help. There is support for animation in *library(rggobi)*. This however requires ggobi, accessible in in <http://www.ggobi.org/>, as additional software.

What has been done here ad-hoc can be done systematically and generalised for arbitrary dimensions: for a data set in \mathbb{R}^q one looks for “interesting” projections. For this, an index is defined that should indicate how interesting a projection is. Then a search is started to maximise this index. A family of statistical methods based on this idea can be found under the key word **projection pursuit**. *ggobi* contains implementations of projection pursuit for several indices. These can be accessed using the R functions in *library(rggobi)*.

4.4.3 Projections for Dimensions 1, 2, 3, ... 7

Projection methods try to identify structures of lower dimension in a higher dimensional data set. The dimensions that can be identified are limited. If we project a structure with a dimension larger than the projections target, a typical projection covers the target space and gives no information any more.

How many dimensions can we capture? The graphical display in the plane gives for first a bound set as low as two dimensions. We can represent two dimensional structures directly using Cartesian co-ordinates in the xy plane (at least locally. The global structure might require additional dimensions.) Perception can reconstruct three dimensional structures with the help of visual depth cues (such as a shade) or from sequences of 2d images. Animation can give the impression of changing 3d scenes and we reach four dimensions.

Additional information channels, as for example colour encoding can elevate the dimension slightly, but effectively we stay with four to seven dimensions for a display.

Combination of several displays helps little beyond this limit. If we combine several displays, for example generalising the idea of a scatter plot matrix, we lose the capability to generate more complex structures by our perception. Instead we have to work out the more complex structures by active comparison. The ability to perform simultaneous comparison of structures is limited. The same, is the number of displays which can be presented simultaneous on a page medium such as a screen or paper is limited.

ToDo:

Does
memory
help to in-
crease the
bound?

Four dimensions is easy. Going beyond seven is hard.

For illustration, we use a classical experiment on colour perception. An observer (usually a person well trained in working with colours) is presented a target colour, and control over three light sources (red, green blue). The task is to adjust the controllable light sources to match the target colour. The classical experiment was performed with 34 target colours varying in frequency in steps of 10nm, with 20 observers.

To start with, we can view a single observation as a four dimensional datum (target colour, intensities for red, green, blue). We can take the three marginals of the target colour by any of the intensities. This gives us [fig:04color](#) colour matching curves which are found in many text books on colour perception. (See [fig:04color](#) top row). We use colour encoding by target colour here as an additional cue. Several notes are necessary. First, the classical experiments have been made with a black background, at that time. Second, the colour cues are a rough approximation, not the exact target colour. Third, not all colours can be mixed from the primaries in this experiment. The observer had the possibility to set the control to negative values which would have the effect to change the target colour by blending. Fourth, as in all real data, there are measurement errors. One observation is off.

ToDo:

color
version R

The basic data set has four dimensional observations. The target colour is already encoded as colour cue. We can combine the three response dimensions in 3d scatter plot. An example is in figure [fig:04color](#), bottom row middle. It tells us that essentially we have a one dimensional data structure, imbedded non linearly in 3d space.

From the experience with linear models we know that the data scatter plot only shows part of the information. The other information is contained in the residuals. Taking residuals from one dimensional non linear regression for the three components (figure [fig:04color](#), middle row) gives us another three dimensional view on the information contained in the data. Again using a (non-marginal) projection from this 3d structure tells us that again we have a non linear structure:

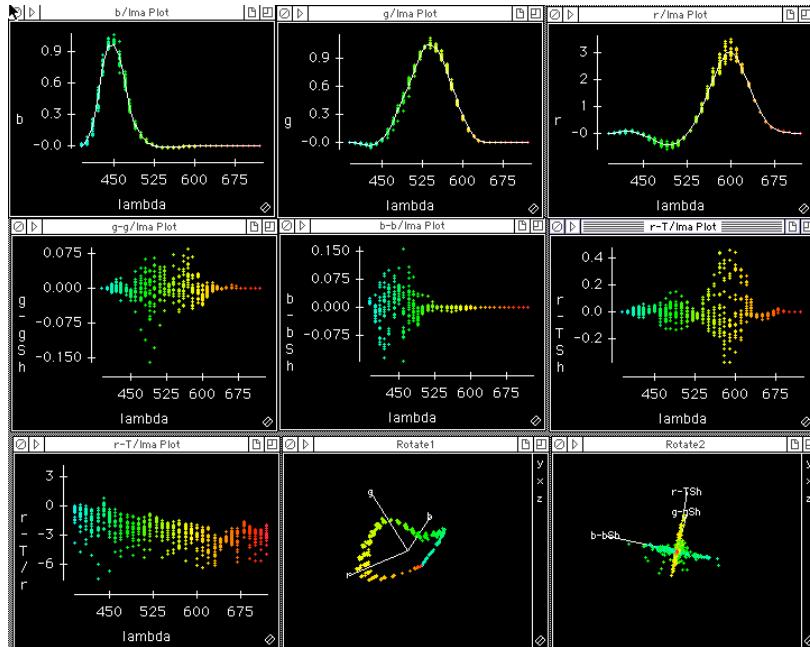


Figure 4.1 Colour perception data. Displays prepared with Data Desk

fig:04color

there is a disk, essentially in the plane characterised by red/green colours, and rod like structure orthogonal to this, essentially characterised by blue/yellow colours.

We only used basic data analysis approaches to reveal this structure. Work on colour perception has proceeded. The conception nowadays is that there are essentially two systems in early colour perception, one combining the red/green channels, the other one based on the blue channel. We did not use this structural information. But the information we get from the residuals just fit.

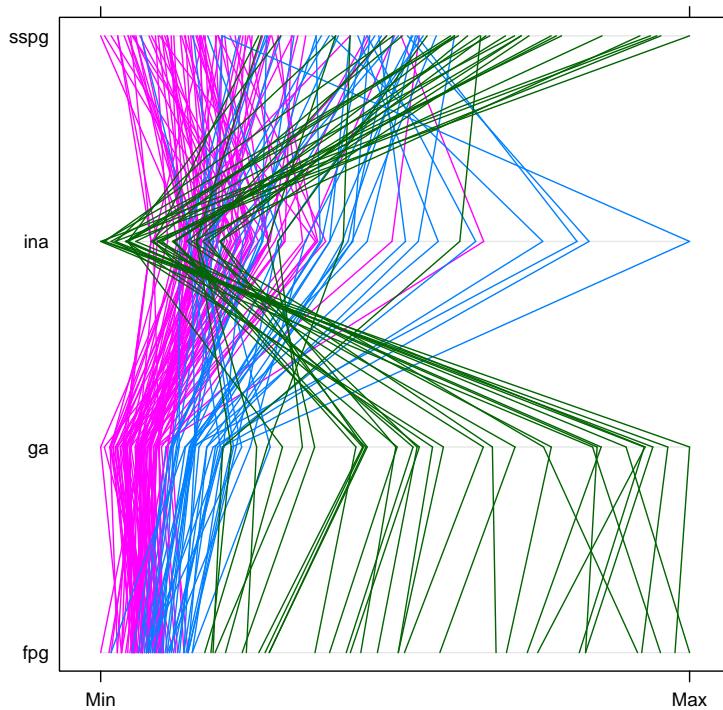
4.4.4 Parallel Coordinates

The graphical display (in Cartesian co-ordinates) are initially restricted to one and two dimensional projections. But even for representation in a plane, the restriction to two dimensions is not predetermined, but a consequence of our choice to display the data in Cartesian co-ordinates. Plot matrices break this dimension barrier by combining Cartesian co-ordinate systems.

Parallel co-ordinates us a parallel orientation of axes, not an orthogonal orientation. For frequencies with categorical variables this is a common display: (possibly stacked) bar charts use parallel co-ordinates. Function `parallel()` in `library(lattice)` implements parallel co-ordinates for quantitative variables as well. The marks on these axes that correspond to one case are joined by a zigzag line. This form of parallel co-ordinates was suggested by A. Inselberg [\[9\]](#).

```
library("lattice")
print(parallel(chemdiab[2:5], groups = chemdiab$cc))
```

Input

**ToDo:**

add legend
end con
[a;ch04:2]
trial colour

The information is the same as that in previous plots of this data set. Using a different display give a new look on the structure in this data set.

| Exercise 4.3 | |
|--------------|---|
| | For the <code>chemdiab</code> data set, prepare a (written!) report about the relation between the variables that you can recognise in the parallel co-ordinate plot. |
| | Instead of using <code>chemdiab[2:5]</code> you can specify the variables explicitly as <code>chemdiab[c(2, 3, 4, 5)]</code> . This gives you control over the order of the variables. Compare two different sequences of the variables and note (in writing!) your observations. Which sequence of variables gives the simplest display? Which relations between the variables are visible in both? Which relations appear only in one of the arrangements? |

4.5 Sections, Conditional Distributions and Coplots

s:4.3

From an abstract point of view, sections are conditional distributions of the type $P(\cdot \mid X = x)$. But they are only reliable where the section defines a condition which has positive probability. To make the idea to restrict the view on conditional distributions applicable to data, we thicken the sections. Instead of considering conditional distributions of the type $P(\cdot \mid X = x)$ we consider $P(\cdot \mid \|X - x\| < \varepsilon)$, where ε possibly can vary with x . In graphical representations of data this requires a series of plots showing only the part of the data set specified by the condition.

Statistically projections lead to marginal distributions and sections to conditional distributions. In a certain sense, sections and projections are complementary: projections show structural features of low dimension. Sections are helpful to detect structural features of low codimension. For data analysis, both can be combined. The interplay of projections and sections is discussed in [5].

Like the dimension boundaries for projections there are boundaries for the codimension when using sections. We can only catch structures of small codimension. If the codimension is too large, a typical section is empty, hence it has no information.

As a first tool, R provides the possibility analyse two variables **conditioned** on one or more additional variables. As a graphical display **coplot()** serves for this purpose. It is a variant of the plot matrix and shows in each tile the scatter plot of two variables, given the condition.

The coplot can be inspected for pattern. If the variables shown are stochastically independent of the conditioning variables, all plot elements show the same shape. The variables shown and the conditioning variables can then be de-coupled.

If the general shape coincides, but location and size vary, this hints to a (not necessarily linear) shift/scale relation. Additive models or variants of these can be used to model the relation between the variables shown and conditioning variables.

If the shape changes with varying condition, a major dependency structure or interaction my apply that needs more precise modelling.

help(coplot)

coplot

Conditioning Plots

Description

This function produces two variants of the **conditioning plots** discussed in the reference below.

Usage

```
coplot(formula, data, given.values, panel = points, rows, columns,
       show.given = TRUE, col = par("fg"), pch = par("pch"),
       bar.bg = c(num = gray(0.8), fac = gray(0.95)),
       xlab = c(x.name, paste("Given :", a.name)),
```

```
conditioned|textbf
coplot@coplot|textit
coplot@coplot|textbf
co.intervals@co.intervals
  (coplot)
Topic
  hplot!coplot@coplot
Topic
  aplot!coplot@coplot
```

```
function@function|textit  ylab = c(y.name, paste("Given :", b.name)),
  subscripts = FALSE,
  axlabels = function(f) abbreviate(levels(f)),
  number = 6, overlap = 0.5, xlim, ylim, ...)
co.intervals(x, number = 6, overlap = 0.5)
```

Arguments

| | |
|---------------------|--|
| formula | a formula describing the form of conditioning plot. A formula of the form $y \sim x a$ indicates that plots of y versus x should be produced conditional on the variable a . A formula of the form $y \sim x a * b$ indicates that plots of y versus x should be produced conditional on the two variables a and b . |
| | All three or four variables may be either numeric or factors. When x or y are factors, the result is almost as if <code>as.numeric()</code> was applied, whereas for factor a or b , the conditioning (and its graphics if <code>show.given</code> is true) are adapted. |
| data | a data frame containing values for any variables in the formula. By default the environment where <code>coplot</code> was called from is used. |
| given.values | a value or list of two values which determine how the conditioning on a and b is to take place. |
| | When there is no b (i.e., conditioning only on a), usually this is a matrix with two columns each row of which gives an interval, to be conditioned on, but it can also be a single vector of numbers or a set of factor levels (if the variable being conditioned on is a factor). In this case (no b), the result of <code>co.intervals</code> can be used directly as <code>given.values</code> argument. |
| panel | a <code>function(x, y, col, pch, ...)</code> which gives the action to be carried out in each panel of the display. The default is <code>points</code> . |
| rows | the panels of the plot are laid out in a <code>rows</code> by <code>columns</code> array. <code>rows</code> gives the number of rows in the array. |
| columns | the number of columns in the panel layout array. |
| show.given | logical (possibly of length 2 for 2 conditioning variables): should conditioning plots be shown for the corresponding conditioning variables (default <code>TRUE</code>) |
| col | a vector of colors to be used to plot the points. If too short, the values are recycled. |
| pch | a vector of plotting symbols or characters. If too short, the values are recycled. |
| bar.bg | a named vector with components "num" and "fac" giving the background colors for the (shingle) bars, for numeric and factor conditioning variables respectively. |
| xlab | character; labels to use for the x axis and the first conditioning variable. If only one label is given, it is used for the x axis and the default label is used for the conditioning variable. |
| ylab | character; labels to use for the y axis and any second conditioning variable. |

| | |
|-------------------------|---|
| <code>subscripts</code> | logical: if true the panel function is given an additional (third) argument <code>subscripts</code> giving the subscripts of the data passed to that panel. |
| <code>axlabels</code> | function for creating axis (tick) labels when <code>x</code> or <code>y</code> are factors. |
| <code>number</code> | integer; the number of conditioning intervals, for <code>a</code> and <code>b</code> , possibly of length 2. It is only used if the corresponding conditioning variable is not a factor. |
| <code>overlap</code> | numeric < 1; the fraction of overlap of the conditioning variables, possibly of length 2 for <code>x</code> and <code>y</code> direction. When <code>overlap</code> < 0, there will be <i>gaps</i> between the data slices. |
| <code>xlim</code> | the range for the <code>x</code> axis. |
| <code>ylim</code> | the range for the <code>y</code> axis. |
| <code>...</code> | additional arguments to the panel function. |
| <code>x</code> | a numeric vector. |

```

factor@factor|textit
par@par|textit
mtext@mtext|textit
title@title|textit
matrix@matrix|textit
range@range|textit
pairs@pairs|textit
panel.smooth@panel.smooth
points@points|textit

```

Details

In the case of a single conditioning variable `a`, when both `rows` and `columns` are unspecified, a “close to square” layout is chosen with `columns >= rows`.

In the case of multiple `rows`, the *order* of the panel plots is from the bottom and from the left (corresponding to increasing `a`, typically).

A panel function should not attempt to start a new plot, but just plot within a given coordinate system: thus `plot` and `boxplot` are not panel functions.

As from R 2.0.0 the rendering of arguments `xlab` and `ylab` is not controlled by `par` arguments `cex.lab` and `font.lab` even though they are plotted by `mtext` rather than `title`.

Value

`co.intervals(., number, .)` returns a (`number` × 2) `matrix`, say `ci`, where `ci[k,]` is the range of `x` values for the `k`-th interval.

References

- Chambers, J. M. (1992) *Data for models*. Chapter 3 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.
 Cleveland, W. S. (1993) *Visualizing Data*. New Jersey: Summit Press.

See Also

`pairs`, `panel.smooth`, `points`.

Examples

```

## Tonga Trench Earthquakes
coplot(lat ~ long | depth, data = quakes)
given.depth <- co.intervals(quakes$depth, number=4, overlap=.1)

```

```

coplot(lat ~ long | depth, data = quakes, given.v=given.depth, rows=1)

## Conditioning on 2 variables:
ll.dm <- lat ~ long | depth * mag
coplot(ll.dm, data = quakes)
coplot(ll.dm, data = quakes, number=c(4,7), show.given=c(TRUE,FALSE))
coplot(ll.dm, data = quakes, number=c(3,7),
       overlap=c(-.5,.1)) # negative overlap DROPS values

## given two factors
Index <- seq(length=nrow(warpbreaks)) # to get nicer default labels
coplot(breaks ~ Index | wool * tension, data = warpbreaks, show.given = 0:1)
coplot(breaks ~ Index | wool * tension, data = warpbreaks,
       col = "red", bg = "pink", pch = 21, bar.bg = c(fac = "light blue"))

## Example with empty panels:
with(data.frame(state.x77), {
  coplot(Life.Exp ~ Income | Illiteracy * state.region, number = 3,
         panel = function(x, y, ...) panel.smooth(x, y, span = .8, ...))
  ## y ~ factor -- not really sensical, but 'show off':
  coplot(Life.Exp ~ state.region | Income * state.division,
         panel = panel.smooth)
})

```

We illustrate coplots with the “Quakes” data set. This data set gives the geographical longitude and latitude of a series of earth quakes near the Fiji island, together with the depth of the seismic centre. We use the geographical longitude and latitude as variables for marginal projection, and the depth as covariate defining the sections.

ToDo:
or change
plot co-
ordinates?

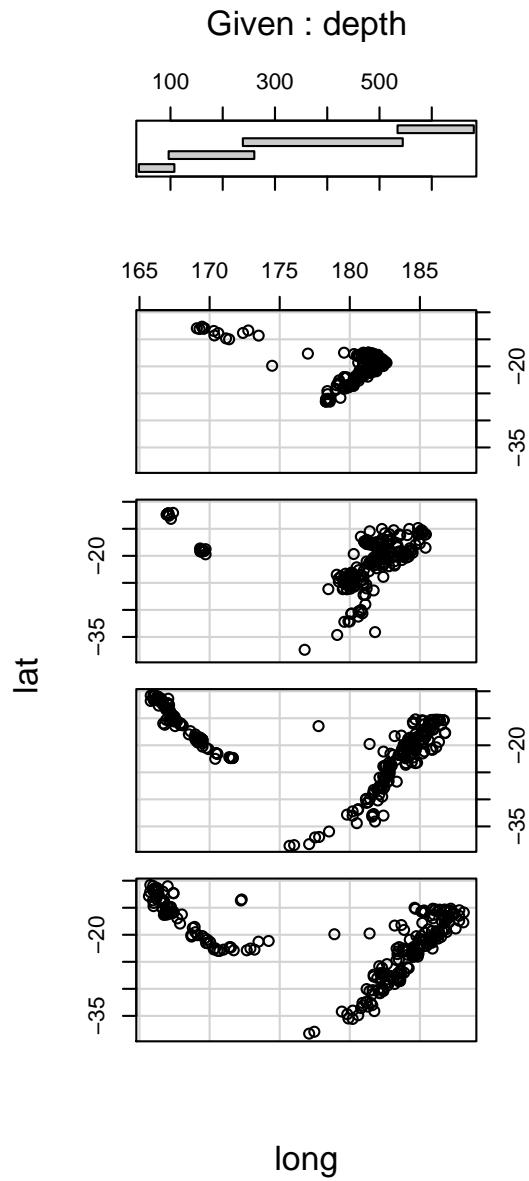
We re-code the depth, so that in graphical representations large depth point downwards.

Input

```

quakes$depth <- -quakes$depth
given.depth <- co.intervals(quakes$depth, number = 4,
                           overlap = 0.1)
coplot(lat ~ long | depth, data = quakes, given.values = given.depth,
       columns = 1)

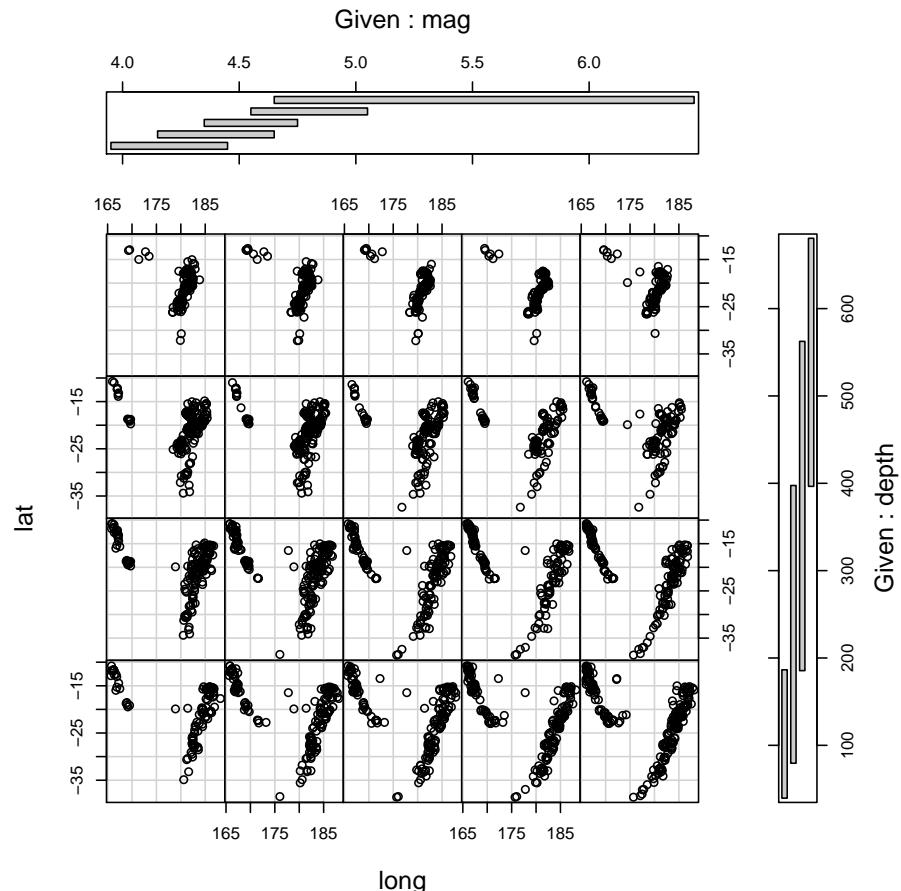
```



Analogue for two covariables, the depth and the magnitude of the earth quake.

Input

```
coplot(lat ~ long | mag * depth, data = quakes, number = c(5,
4))
```



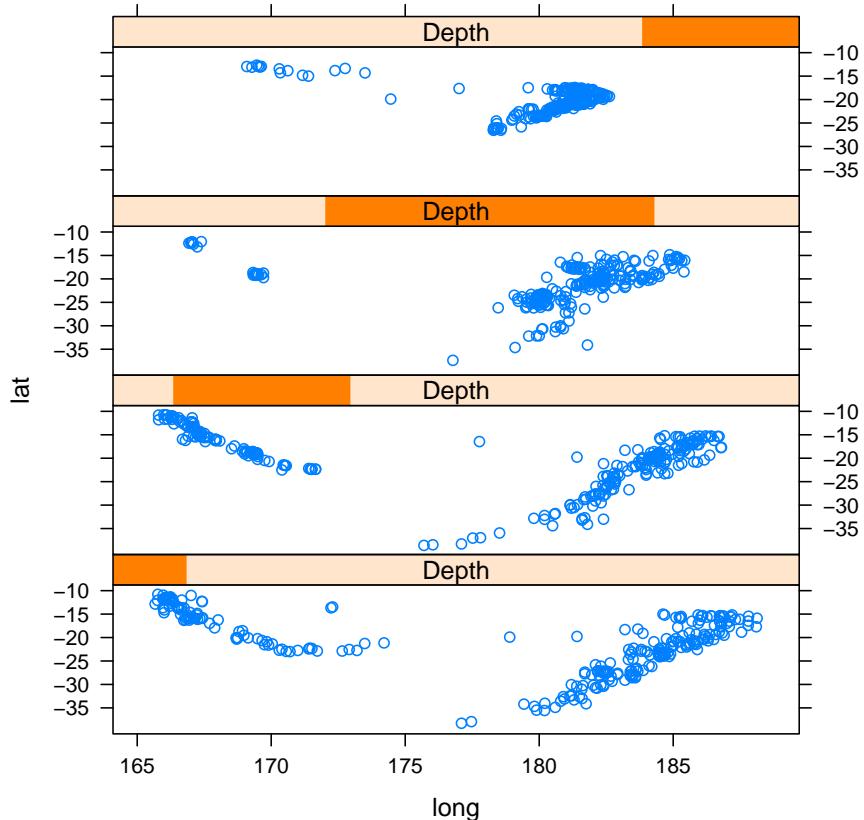
a:ch04:5

| Exercise 4.4 | |
|--------------|--|
| | Analyse the “quakes”-data set. Summarise your results in a report. Try to specify a formal model. |
| | How is the geographic position related to the depth? |
| | Can you identify relations between depth and magnitude of the earth quake? (You may have to choose a different model formula in <code>coplot()</code> .) |

ToDo:
Comment
 on trellis

The idea of the coplots is generalised in the trellis displays (see [4]). Trellis displays are implemented in R in `library("lattice")`. Input [\[4\]](#)

```
library("lattice")
Depth <- equal.count(quakes$depth, number = 4, overlap = 0.1)
print(xyplot(lat ~ long | Depth, data = quakes, columns = 1,
            layout = c(1, 4)))
```



To Do:
add mag

4.6 Transformations and Dimension Reduction

Variables often come in a scale that is suggested by the measurement process or by conventions of the trade. This does not necessarily correspond to a scale that would be suggested by the subject matter or that is best used for statistical modelling. This scale has some arbitrariness:

- For an acoustic perception experiment for example the strength of the stimulus can be described by energy or by acoustic pressure [phone]. One scale is transformed to the other by a logarithm. The Weber-Fechner law in psychology says that for the human perception the (logarithmic) phone scale is appropriate.
- Fuel consumption is reported in the USA in miles per gallon, in Europe as litres per 100 km. Up to conversion constants, one variable is the inverse of the other. When we want to model fuel consumption, the description in litres per 100 km seems to lead to simpler statistical models. Analysis in terms of miles per gallon can become rather complicated.

To Do: Ref
to curse of
dimension
To Do:
Vari-
ance bias
dilemma

The choice of an appropriate scale for the variables can be a critical step in the analysis. Sometimes it is helpful to first introduce transformations and constructed variables, and then to reduce dimensions and determine the effective variables in a second step.

```
plot@plot|textit{Co-ordinate systems are not pre-set canonically. This already applies to univariate problems. For univariate problems, transformations of co-ordinate systems are relatively simple. Modelling of the error distribution on the one hand and transformation the data to a standard distribution are in a certain way exchangeable. In higher dimension situations, suitable transformation families sometimes are not available or not accessible, and the structure of a problem may critically depend on the choice of suitable co-ordinates. A choice of the co-ordinate representation based on the subject matter often is preferable to an automatic selection.
```

We illustrate this with Anderson's Iris-data set. The data set has five dimensions: four quantitative variables (length and width of petal sepal of iris blossoms) and a categorical variable (the species: iris setosa canadensis, iris versicolor, iris virginica)*. The question is to find a classification of the species by the four quantitative variables .

Table 4.11 *Iris species*.



The structure is similar to that of the diabetes data set `chemdiab`. The classification by `iris$Species` however is here a given (external) classification, in contrast to the derived classification `chemdiab$cc`. While in the case of `chemdiab` a general description was looked for, the aim is now to find a classification rule that classifies `iris$Species` by means of the other variables.

In this example, the species define the selection groups which are of interest.

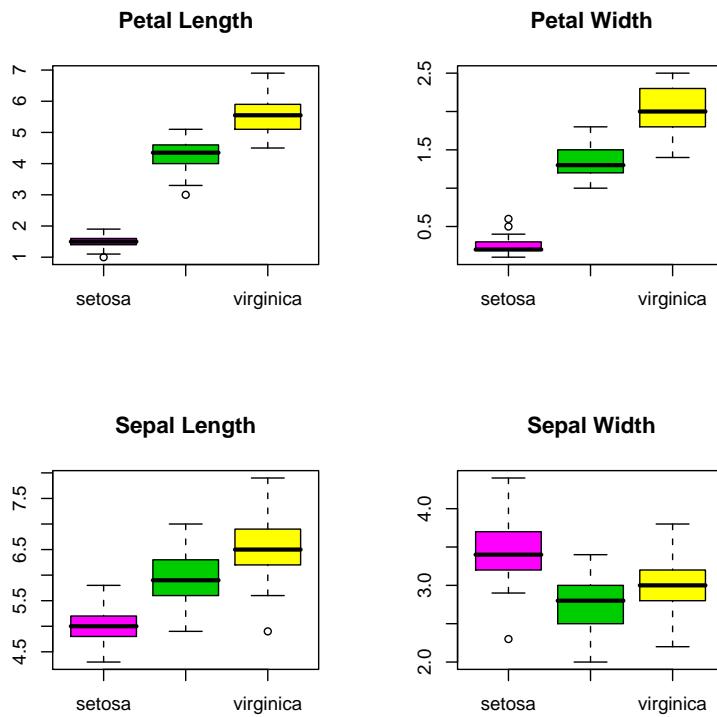
To get a first impression we can view the four quantitative variables by species. The standard conventions of R make this cumbersome. Since species is a categorical variable, this cause R to switch from a scatter plot display to box&whisker plots when using `plot()`.

```
oldpar <- par(mfrow = c(2, 2))
plot(iris$Species, iris$Petal.Length, ylab = "",
     main = "Petal Length", col = c("magenta", "green3",
     "yellow"))
plot(iris$Species, iris$Petal.Width, ylab = "", main = "Petal Width",
     col = c("magenta", "green3", "yellow"))
plot(iris$Species, iris$Sepal.Length, ylab = "",
     main = "Sepal Length", col = c("magenta", "green3",
     "yellow"))
plot(iris$Species, iris$Sepal.Width, ylab = "", main = "Sepal Width",
```

* Photos: The Species Iris Group of North America. With kind permission

```
col = c("magenta", "green3", "yellow")
par(oldpar)
```

```
stripplot@stripplot|text{
```

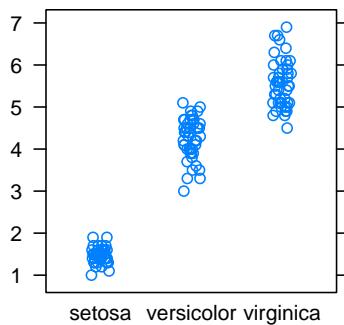
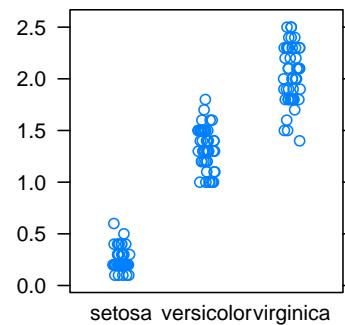
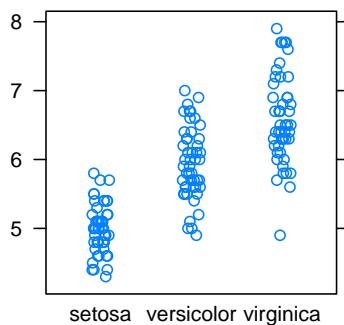
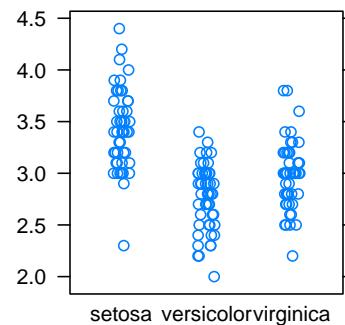


We could modify the R functions to show a scatter plot of the individual variables by group. Instead we resume to *grid* and *lattice* and use the function *stripplot()*. Since given the measurement precision in this experiment the same value is recorded in multiple cases, we jitter the data to present them separately.

Input

```
library("lattice")
print(stripplot(Petal.Length ~ Species, data = iris,
    jitter = TRUE, ylab = "", main = "Petal Length"),
    split = c(1, 1, 2, 2), more = TRUE)
print(stripplot(Petal.Width ~ Species, data = iris,
    jitter = TRUE, ylab = "", main = "Petal Width"),
    split = c(2, 1, 2, 2), more = TRUE)
print(stripplot(Sepal.Length ~ Species, data = iris,
    jitter = TRUE, ylab = "", main = "Sepal Length"),
    split = c(1, 2, 2, 2), more = TRUE)
print(stripplot(Sepal.Width ~ Species, data = iris,
    jitter = TRUE, ylab = "", main = "Sepal Width"),
    split = c(2, 2, 2, 2))
```

lida@lida|textit

Petal Length**Petal Width****Sepal Length****Sepal Width****ToDo:**

add color

ToDo:**Remark**[a:ch04:111
on small]

setosa

The one dimensional marginal distributions give not enough hints how to separate the three groups.

A two dimensional display does not help much further.

| Exercise 4.5 | |
|--------------|--|
| | Use the methods from section 4.4 and 4.5 to inspect the data set. Can you see classification rules which give a classification of the three species to a large extent? <small>ch04projekt4.5nen</small> |

Using formal methods such as linear discriminant analysis (for example `lida()` in `library(MASS)`), a classification based on the morphometric variables can be found. Separating the species is not trivial.

The original variables only represent an aspect of the data that is technically most accessible. From a biological point of view, a different parametrisation would be chosen. The variables

reflect size and form of the leaves. A first approximation would be

$$\text{area} = \text{length} \cdot \text{width} \quad (4.1)$$

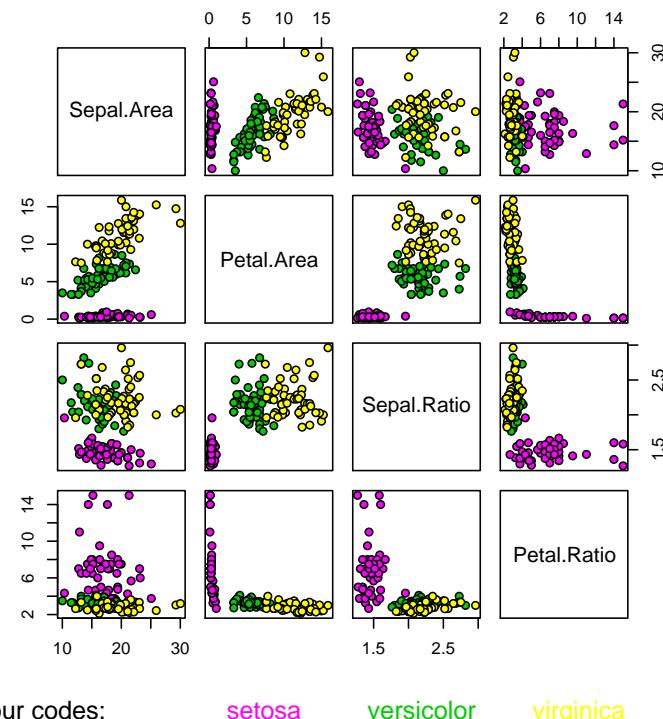
$$\text{aspectratio} = \text{length}/\text{width}. \quad (4.2)$$

This gives the display

```
Input
iris$Sepal.Area <- iris$Sepal.Length * iris$Sepal.Width
iris$Petal.Area <- iris$Petal.Length * iris$Petal.Width
iris$Sepal.Ratio <- iris$Sepal.Length/iris$Sepal.Width
iris$Petal.Ratio <- iris$Petal.Length/iris$Petal.Width
pairs(iris[6:9], main = "Anderson's Iris Data -- 3 species",
      pch = 21, bg = c("magenta", "green3", "yellow") [unclass(iris$Species)],
      oma = c(8, 8, 8, 8))
mtext(c("Colour codes:", levels(iris$Species)), col = c("black",
      "magenta", "green3", "yellow"), at = c(0.1, 0.4,
      0.6, 0.8), side = 1, line = 2)
```

ToDo:
check iris
inverse
ratio

Anderson's Iris Data -- 3 species



In the marginal distribution the species are separated almost completely, with two borderline cases. In these more biological co-ordinates we see that for classification the area and aspect

ToDo: use
attach

ratio of the petal is sufficient. Any more complicated formal procedure has at first to better this trivial classification rule.

Even an exhaustive search, for example using projection pursuit, only catches projections, that is it only catches special linear combination of the variables. For the iris data, we first introduced new variables, the areas and aspect ratios. These are non linear transformations. Only in a second step the classifying variables are identified and the dimension is reduced drastically. In genuinely multivariate problems it is quite common that initially a dimension extension is necessary to solve the problem. Dimension reduction only makes sense if the describing variable are sufficiently complex to lead to a solution.

4.7 Higher Dimensions

4.7.1 Linear Case

If we have essentially linear structures, so can we often can analyse higher dimensional structures with methods that have been designed for one dimensional models. We may have to modify the methods or use them in iterations, taking residuals at each step. But nevertheless they may help us to recognise essential features.

In chapter 2 we have already introduced linear models for arbitrary regressor dimensions p . However chapter 2 presupposes that the model for the statistical analysis is specified off hand, that is the information from the data material does not influence the selection of the model, but only decisions within the framework of the model chosen.

In higher dimensional problems, it is quite common that the model is still to be specified. A common special case is the selection of regressors: the variables are candidates, among which a (preferably small) number of regressors are to be chosen.

Do more complicated models provide an essential improvements over a simple model? Which parameters resp. which derived variables should be included in the model? The lesson from linear models is that not the value of individual parameters determine the contribution in a model, but that space determined by the parameters is the essential factor. Adapted strategies are needed here. We can start with simple models and ask whether additional parameter give an essential contribution. Additional parameters will improve the fit, but on the other hand the variance of the estimation will increase. We can as well start with a relatively complex model and ask whether we can omit parameters. This may increase the residual error, but we may gain reliability of the estimations.

ToDo:
formal
variance
bias

For an abstract linear model, both strategies lead to a comparison of two model spaces $\mathcal{M}_{X'} \subset \mathcal{M}_X$. The corresponding estimators are $\pi_{\mathcal{M}_{X'}}(Y)$ and $\pi_{\mathcal{M}_X}(Y)$. The relation between both becomes clear if we choose the orthogonal decomposition $\mathcal{M}_X = \mathcal{M}_{X'} \oplus L_X := M_0, L_X := \mathcal{M}_X \ominus \mathcal{M}_{X'}$ of \mathcal{M}_X . Then $\pi_{\mathcal{M}_X}(Y) = \pi_{\mathcal{M}_{X'}}(Y) + \pi_{L_X}(Y)$.

Partial Residuals and Added Variable Plots

In regression $\mathcal{M}_{X'}$ and \mathcal{M}_X are spaces which are spanned by the vectors of regressor variables. In our situation we are interested in the special case

$$X' = \text{span}(X_1, \dots, X_{p'}); X = \text{span}(X_1, \dots, X_p)$$

with $p > p'$. Here L_X is spanned by the vectors

$$R_{p'+1} = X_{p'+1} - \pi_{\mathcal{M}'_X}(X_{p'+1}), \dots, R_p = X_p - \pi_{\mathcal{M}'_X}(X_p).$$

If we (formally) do a linear regression of the additional regressors by the regressors already contained in X' , the resulting residuals generate L_X . An additional regression of Y by these residuals yields the term $\pi_{L_X}(Y)$ that describes the difference between the models. By construction we know that $\pi_{\mathcal{M}'_X}(Y)$ is orthogonal to L_X . So this part is mapped to zero by the second regression. We can as well eliminate this part and restrict us to the regression of $Y' = Y - \pi_{\mathcal{M}'_X}(Y)$ after $R_{p'+1}, \dots, R_p$.

The strategy is simple: we check whether additional parameters should be included in the model. Instead of the scatter plot matrix of the original data we look at the scatter plots of the (formal) residuals from this simple model. These scatter plots are called **added variable plots**.

To point out the difference to the scatter plot matrix of the original data: linear structures in the scatter plot the original data are a clear hint for linear dependency. Non linear structures, such as a triangular shape in some of the scatter plots can reflect a corresponding dependency structure. But it can also be an artefact, based on the distribution and correlation structure of the regressors. In general, they do not have a simple interpretation. In contrast, the representations in the matrix the added variable plots are adjusted for linear effects of the preceding variables. As a consequence, they depend on the order in which variables are taken into the model. They correct for linear effects resulting from the correlation of preceding variables. This avoids several artefacts, and the added variable plots are easier to interpret.

a:ch04:12

```
added
variable
plots|textbf
principal
component
analysis|textbf
prcomp@prcomp|textit
mva@mva|textit
```

| Exercise 4.6 | |
|--------------|--|
| | <p>Modify the following function <code>pairslm()</code> so that it calculates the residuals of the regression of all original variables in matrix <code>x</code> by regression after the new variable <code>x\$fit</code> and produce a scatter plot matrix of these residuals.</p> <pre>pairslm <- function(model, x, ...) { x\$fit <- lm(model, x)\$fit; pairs(x, ...)} </pre> <p>Add title, legends etc..</p> <p>Use the "trees" data set as an example.</p> |

We discussed the transition from p' to $p' + 1$ variable. The scatter plot matrix allows to get a quick overview of a (not too large) number of candidates (three potential additional regressors in our case). The transition from p to $p - 1$ for the elimination of a variable, is in a certain sense dual to this. It corresponds to stepwise elimination, the other basic strategy.

Instead of using individual variables step by step, it can be more efficient to select combination of several variables and use these synthetic variables in a model. Methods to achieve this are discussed under the heading **principal component analysis** and supported by function `prcomp()` in `library(mva)`. We will return to this in a later example (Seite 4-43).

The experience with the linear models teaches us, that the marginal relations are only half of the truth. Instead of using the regressors individually, we have to use a stepwise orthogonalisation. Component-wise interpretation of estimators becomes questionable. The estimation may depend on the order in which regressors are used.

In more complex situations, formal methods may lead astray. Technical skills are needed. Unfortunately knowledge about how interventions affect the validity of formal methods is rather limited. This makes it more important to judge chosen strategies critically using simulations.

4.7.2 Non-linear Case

Non-linear relations in higher dimension pose a challenge. Besides a good collection of methods we need a repertoire of examples showing which structures can occur and where we have to pay attention. The following example, the cusp singularity is one of these examples: it is one of the most simple structures that can occur in higher dimensions. The base is here a two dimension structure, a surface, that is not trivially imbedded in a three dimension space. The interesting feature is here the bifurcation from a unimodal to a bimodal situation.

Example: Cusp Non-linearity

The most simple example can be illustrated with reference to physical applications. In physical systems, probability distributions often are related to energy levels; (local) Minima of the energy correspond to modes of the distribution. A typical relation is: if the energy behaves as $\varphi(y)$, after standardisation the distribution behaves like $e^{-\varphi(y)}$. If $\varphi(y)$ is quadratic at the minimum, up to scale transformation, we get distributions from the normal distributions family.

Differential topology teaches us that the qualitative image persist under small perturbations or variations. At least locally the energy stays approximatively quadratic, and the normal distributions stay at least approximatively a suitable distribution family.

The behaviour changes drastically if the behaviour behaves locally like y^4 . Small perturbations can make the potential locally a quadratic function. But they can as well lead to breaking the local minimum into two minima. The typical image has the form

$$\varphi(y; u, v) = y^4 + u \cdot y^2 + v \cdot y. \quad (4.3)$$

f:cusp0

The variations are controlled by the parameters u, v . A dynamic interpretation helps to understand the situation: imagine that u, v are external parameters which can change. We know this scenario from magnetic Hysteresis: y gives the magnetisation in a certain direction, u takes the role of temperature; v that of an external magnetic field. At high temperature, magnetisation follows the external magnetic field directly. At lower temperatures, the material shows memory. The magnetisation does not only depend on the outer magnetic field, but also on the previous magnetisation.

We know similar “memory effect” in other areas. Imagine a market with prices y , costs v and a “pressure of competition” u . With sufficient competition the prices will follow the cost, more or less, given other conditions unchanged. In a monopoly situation, prices seem to have a memory: once they have risen, they will only become lower if the costs reduce drastically.

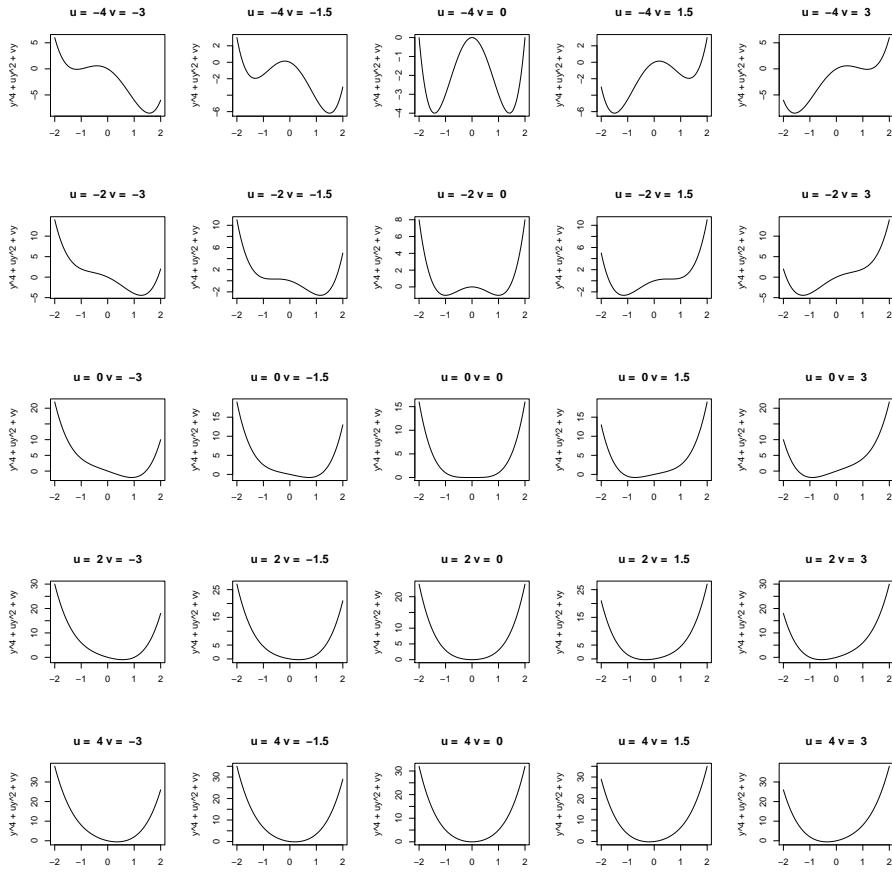
The “unfolding” of the potential y^4 given in formula 4.3 has a characteristic form. From

$$\varphi'(y; u, v) = 4y^3 + 2u \cdot y + v = 0 \quad (4.4)$$

f:cusp1

we get the critical points (see Abb. 4.3). See figure 4.3 on page 4-32.

fig:ch04cuspsurf1

Figure 4.2 Unfolding of y^4 : $\varphi(y; u, v) = y^4 + u \cdot y^2 + v \cdot y$

ToDo: adjust scale, orientation

A projection to the u, v gives a cusp (Fig. 4.4). For parameter values in the interior of this cusp the potential has two local minima, outside of the cusp there is only one extreme value.

ch04cuspine

ToDo: 2/3

The distributions corresponding to these potentials are – up to scale transformation for normalisation –

$$p(y; u, v) \propto e^{-(y^4 + u \cdot y^2 + v \cdot y)}.$$

The structure of the potentials carries over to the corresponding distributions.

The situation appears mostly harmless: the parameter space (the space of the regressors) $x = (u, v)$ has only two dimensions. The distribution is one dimensional with a smooth density. But

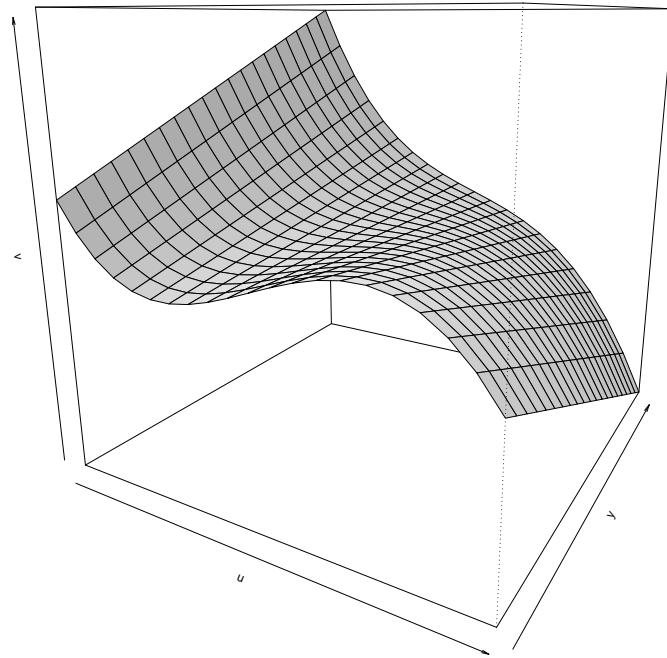


Figure 4.3 Critical points $\varphi'(y; u, v) = 4y^3 + 2u \cdot y + v = 0$

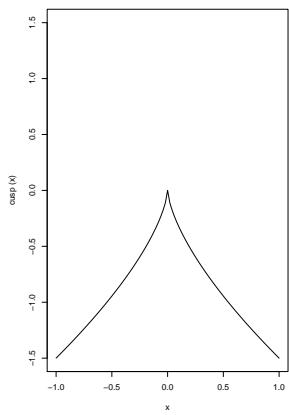


Figure 4.4 Boundary between uni- and bimodality in (u, v) space

HIGHER DIMENSIONS

4-33

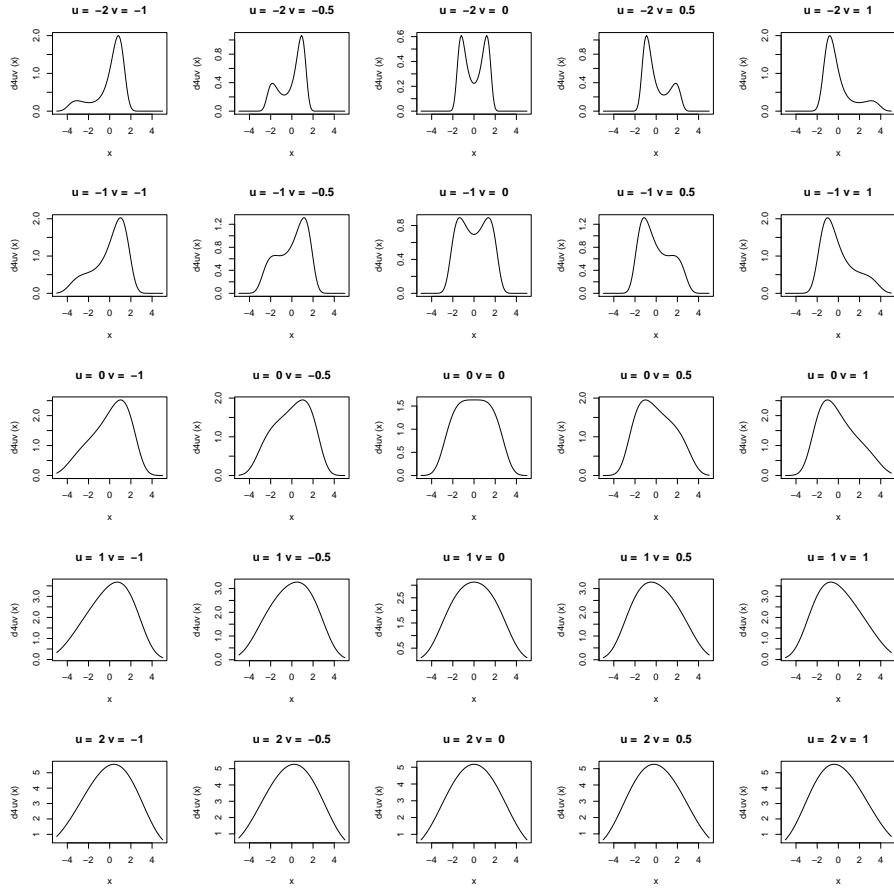


Figure 4.5 $p(y; u, v) \propto e^{-(y^4 + u \cdot y^2 + v \cdot y)}$

the situation cannot be captured sufficiently by linear methods. The typical non-linear effect will not be recognised if you are not prepared for it. Only overall picture in three dimensions convey the underlying structure.

a:ch04:13

This simple example is a challenge. How can we diagnose a structure of this kind?

To Do:
add Mel-
lourne
data

| Exercise 4.7 | |
|--------------|--|
| | <p>Write a function <code>dx4exp(x, u, v)</code> to calculate the centred probability densities for (4.5). You have to integrate the density from (4.5) to determine the norming constant and calculate the expected value to centre the density. For both, you can use a numerical integration with <code>integrate()</code>.</p> <p style="text-align: right;">(cont.) →</p> |

| Exercise 4.7 | (cont.) |
|--------------|--|
| *** | <p>For values u, v on a grid in $u = -2 \dots 2$ and $v = -1 \dots 1$ simulate 100 random numbers from $dx4exp(x, u, v)$. Use the methods from chapter 2 to analyse the data.</p> <p>Can you detect hints indicating the non-linear structure?</p> <p>Is the bimodality recognisable?</p> <p>How much of the structure can you identify?</p> |

In non-linear relations, joint dependency can have a large importance. In general the requires prudence in modelling. Non-linear relations can be hidden in projections. Artefact of the (linear) projection can give an image that does not correspond to the original relations.

4.7.3 “Curse of Dimension”

curse of
dimension|textbf

With the lack of adapted co-ordinate systems, an extensive search for interesting projections or sections is necessary. The number of possibilities increases rapidly with the dimension. For illustration: to identify a cube, at least the corner points must be detected. In d dimensions there are 2^d corner points. The number increases exponentially with the dimension. This is an aspect of the problem known as **curse of dimension**.

Seen from another point of view: if we look at the data points which are extremal in at least one variable dimension, in the one dimensional case we get two points. In d dimensions typically we get 2^d extremal points. If we do not look at co-ordinate directions, but in arbitrary directions, typically any point is extremal, if d gets large.

A third aspect: in d dimensional space almost every point is isolated. Localisations, like those discussed in section 2.5, will break down. If we take a neighbourhood around some point covering a proportion p of the data range, for example $p = 10\%$ of the span of the variables, in one dimension typically we cover a proportion of p of the data points. In d dimensions this is this only a fraction in the order of p^d . So for example in 6 dimensions we need several million data points in order to have not trivial environments.

4.7.4 Case Study: Body Fat

ToDo:
reduce
body fat
case study
drastically

As a running example we use here the fat data set. This data set is published repeatedly in the literature and in R among others available in library **UsingR**.

Target of the investigation related to this data set is the determination of the proportion of body fat. The most reliable method is to use a water be to determine the average density of the tissue and to infer from the to the body fat proportion. This measurement is rather intricate. Can it be replaced by body parameters **fatvars** that allow a more simple measurement? The parameters at disposition are summarised in table 4.15.

Anhand the Übersicht in table 4.15 sehen we gleich, that metrische Angaben and US-Maße gemischt are. Damit for uns the Interpretation einfacher is, stellen we all Angaben auf metrische values um.

Input

```
library("UsingR")
data(fat)
fat$weightkg <- fat$weight * 0.453
fat$heightcm <- fat$height * 2.54
fat$ffweightkg <- fat$ffweight * 0.453
```

ToDo:
reduce
prelim-
inary
informa-
tion on
fat

The variables **body.fat** and **body.fat.siri** are derived from the observed vales of **density**. These formulas reflect assumptions about the mean density of and of fat free tissue. With these assumptions the fat proportion can be calculated (or rather: estimated) from the **density**. Both formulas use a density dependent factor $1/\text{density}$. Up to (given or assumed) constants is this the relevant term for us (and not **density**).

The first step is a critical inspection and cleaning of the data set. This is almost always necessary, not only for higher dimensional data sets. In for higher dimensional data sets though we often have redundancies that allow for consistency checks and possibly for corrections. In our case

pairs@pairs|textit

| Name | Variable | Unit, Remarks |
|---------------|--|---------------|
| case | Case Number | |
| body.fat | Percent body fat using Brozek's equation, $457/Density - 414.2$ | |
| body.fat.siri | Percent body fat using Siri's equation, $495/Density - 450$ | |
| density | Density | $[g/cm^2]$ |
| age | Age | $[yrs]$ |
| weight | Weight | $[lbs]$ |
| height | Height | $[inches]$ |
| BMI | Adiposity index = $Weight/Height^2$ | $[kg/m^2]$ |
| ffweight | Fat Free Weight = $(1 - fraction of body fat) * Weight$, using Brozek's formula | $[lbs]$ |
| neck | Neck circumference | $[cm]$ |
| chest | Chest circumference | $[cm]$ |
| abdomen | Abdomen circumference "at the umbilicus and level with the iliac crest" | $[cm]$ |
| hip | Hip circumference | $[cm]$ |
| thigh | Thigh circumference | $[cm]$ |
| knee | Knee circumference | $[cm]$ |
| ankle | Ankle circumference | $[cm]$ |
| bicep | Extended biceps circumference | $[cm]$ |
| forearm | Forearm circumference | $[cm]$ |
| wrist | Wrist circumference "distal to the styloid processes" | $[cm]$ |

Table 4.15 Fat data set: variables

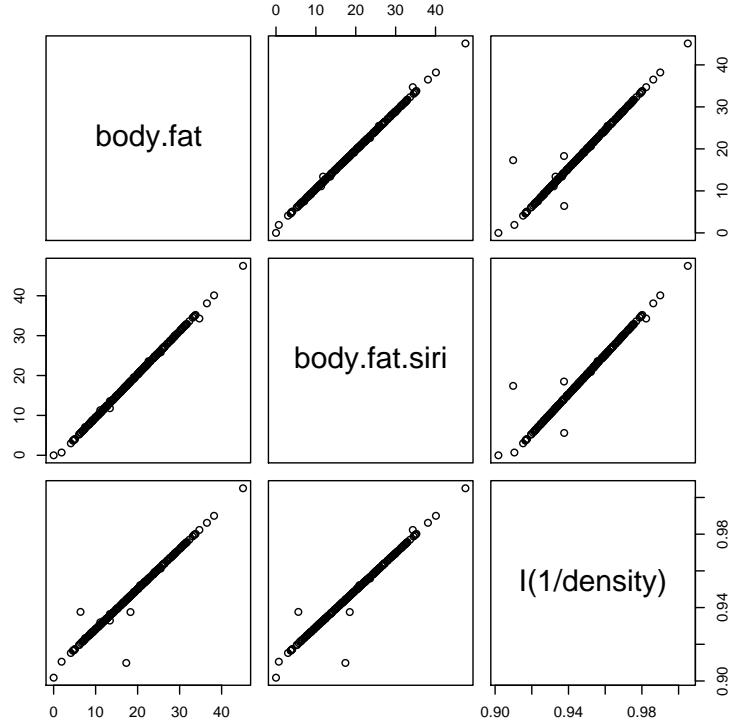
fatvars

the variables `body.fat`, `body.fat.siri`, `ffweight` and `BMI` are derived variables that are in a deterministic relation to other variables.

We first look at the group `body.fat`, `body.fat.siri`, `1/density`. The pair-wise scatter plots should show a straight line. `pairs()` provides good service. We use in the formula based variation here. To mark that `1/density` should be calculate, and the `/` operator is not to be understood as a formula operator, we have to mark up this term.

Input

```
pairs(~body.fat + body.fat.siri + I(1/density), data = fat)
```



The inconsistent values and outliers are obvious. Unfortunately in R it is not simple to mark values in a scatter plot matrix.

a:ch04:14

| Exercise 4.8 | Use functions <code>plot()</code> and <code>identify()</code> , to generate the following output: |
|--------------|--|
| | <p>The four plots show the following data points:</p> <ul style="list-style-type: none"> body fat vs 1/density: Points include (0.91, 0.18), (0.92, 0.96), (0.93, 0.76), (0.94, 0.68), (0.95, 0.48), and (0.98, 216). body fat siri vs 1/density: Points include (0.91, 0.18), (0.92, 0.96), (0.93, 0.76), (0.94, 0.68), (0.95, 0.48), and (0.98, 160). ffweightkg vs (1-body.fat/100)*weightkg: Points include (55, 55), (60, 65), (70, 75), (80, 85), (90, 95), and (100, 110). BMI vs weightkg/(fat\$heightcm/100)^2: Points include (40, 25), (45, 30), (50, 35), (55, 40), (60, 45), (65, 50), (70, 55), (75, 60), (80, 65), (85, 70), (90, 75), (95, 80), (100, 85), (110, 90), (120, 95), (130, 100), (140, 105), (150, 110), and (160, 115). |

If an obvious correction is possible, it should be done here. Of course the correction needs to be noted in the report of the analysis. case 42 is simple: a height of 0.73m for a weight 63.5kg is not plausible and not consistent with BMI 29.9. From the BMI we can recalculate the height. Instead of the value of 29.5 inch the height should presumably 69.5 inch.

Input

```
fat$height[42] <- 69.5
fat$heightcm[42] <- fat$height[42] * 2.54
```

Case 216 is a discretionary. The density is extremely low, the BMI extremely high. On the other hand, the body measurements fit to these extremes. This case can be an outlier which would distort the analysis. But it could also be an observation which is particularly informative. We note it as a particularity.

After this preliminary inspection we clean up the data set. We remove the variables that contain no additional information or have been replaced. As a target variable, we use `body.fat`. However we keep the variable `density` for later purposes.

Input

```
fat$weight <- NULL
fat$height <- NULL
fat$ffweight <- NULL
fat$ffweightkg <- NULL
fat$body.fat.siri <- NULL
```

`lm@lm|textit`

Some indices are in common use to describe the body constitution (see Figure 4.6). Some time ago, the rule of thumb was ‘optimal weight = height -100’. Today the “body mass index” BMI = weight/ height² is usual. (Commercially available scales determine the body fat by electrical impedance. This variable is not contained in the fat data.)

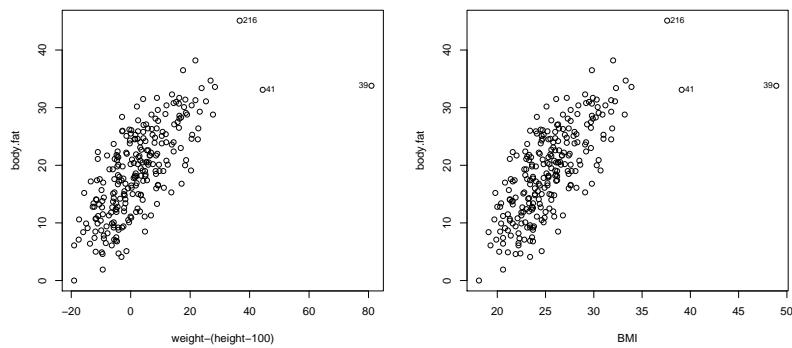


Figure 4.6 *Body fat against conventional indices*

`fat01scat`

We can use the conventional indices in a linear model. We exclude the obvious outliers and possible leverage points. This can be achieved with the `subset` parameter of `lm()`.

For the rule of thumb ‘optimal weight = height -100’ we get:

Input

```
lm.height <- lm(body.fat ~ I(weightkg - (heightcm -
  100)), data = fat, subset = -c(39, 41, 216))
summary(lm.height)
```

Output

Call:

```
lm(formula = body.fat ~ I(weightkg - (heightcm - 100)), data = fat,
  subset = -c(39, 41, 216))
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|----------|----------|---------|----------|
| -11.90734 | -3.68697 | -0.05303 | 3.65458 | 12.28000 |

Coefficients:

| | Estimate | Std. Error | t value |
|-------------|----------|------------|---------|
| (Intercept) | 17.70722 | 0.33296 | 53.18 |

4-40

DIMENSIONS 1, 2, 3, ..., ∞

```
I(weightkg - (heightcm - 100)) 0.54557   0.03283  16.62
                                         Pr(>|t|)
(Intercept)                      <2e-16 ***
I(weightkg - (heightcm - 100))  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 5.166 on 247 degrees of freedom
Multiple R-squared: 0.5279, Adjusted R-squared: 0.526
F-statistic: 276.2 on 1 and 247 DF, p-value: < 2.2e-16

The regression of *body.fat* for *BMI* results in:

Input

```
lm.BMI <- lm(body.fat ~ BMI, data = fat, subset = -c(39,
41, 216))
summary(lm.BMI)
```

Output

Call:
lm(formula = body.fat ~ BMI, data = fat, subset = -c(39, 41,
216))

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|----------|----------|---------|----------|
| -12.49460 | -3.53561 | -0.05228 | 3.69129 | 11.72720 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|-------------|----------|------------|---------|------------|
| (Intercept) | -25.6130 | 2.6212 | -9.772 | <2e-16 *** |
| BMI | 1.7564 | 0.1031 | 17.042 | <2e-16 *** |
| --- | | | | |

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.097 on 247 degrees of freedom
Multiple R-squared: 0.5404, Adjusted R-squared: 0.5385
F-statistic: 290.4 on 1 and 247 DF, p-value: < 2.2e-16

The fit of $R^2 = 0.53$ resp. $R^2 = 0.54$ is poor in both cases.

Even using all data points and all regressors, we get at most $R^2 = 0.75$:

Input

```
lm.fullres <- lm(body.fat ~ age + BMI + neck + chest +
abdomen + hip + thigh + knee + ankle + bicep +
forearm + wrist + weightkg + heightcm, data = fat)
summary(lm.fullres)
```

Output

Call:
lm(formula = body.fat ~ age + BMI + neck + chest + abdomen +

HIGHER DIMENSIONS

4-41

```
hip + thigh + knee + ankle + bicep + forearm + wrist + weightkg +
heightcm, data = fat)
```

regsubsets@regsubsets|tex

Residuals:

| | Min | 1Q | Median | 3Q | Max |
|--|----------|---------|---------|--------|--------|
| | -10.0761 | -2.6118 | -0.1055 | 2.8993 | 9.2691 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|------------|------------|---------|-------------|
| (Intercept) | -50.804727 | 36.489198 | -1.392 | 0.16513 |
| age | 0.061005 | 0.029862 | 2.043 | 0.04217 * |
| BMI | 0.782993 | 0.733562 | 1.067 | 0.28688 |
| neck | -0.439082 | 0.218157 | -2.013 | 0.04528 * |
| chest | -0.040915 | 0.098266 | -0.416 | 0.67751 |
| abdomen | 0.866361 | 0.085550 | 10.127 | < 2e-16 *** |
| hip | -0.206231 | 0.136298 | -1.513 | 0.13159 |
| thigh | 0.246127 | 0.135373 | 1.818 | 0.07031 . |
| knee | -0.005706 | 0.229564 | -0.025 | 0.98019 |
| ankle | 0.135779 | 0.208314 | 0.652 | 0.51516 |
| bicep | 0.149100 | 0.159807 | 0.933 | 0.35177 |
| forearm | 0.409032 | 0.186022 | 2.199 | 0.02886 * |
| wrist | -1.514111 | 0.493759 | -3.066 | 0.00242 ** |
| weightkg | -0.389753 | 0.221592 | -1.759 | 0.07989 . |
| heightcm | 0.187196 | 0.199854 | 0.937 | 0.34989 |
| --- | | | | |
| Signif. codes: | 0 *** | 0.001 ** | 0.01 * | 0.05 . |
| | 0.1 | ' | ' | 1 |

Residual standard error: 3.991 on 237 degrees of freedom
 Multiple R-squared: 0.7497, Adjusted R-squared: 0.7349
 F-statistic: 50.7 on 14 and 237 DF, p-value: < 2.2e-16

This is a model with 15 coefficients. The model is so complex that it is hardly to interpret, and one would try to reduce the model. Instead of looking “by hand” for simpler models, this process can be automated. This is done for example by function `regsubsets()` in `library(leaps)`. The quadratic error (resp. the coefficient of determination R^2) has to be modified: the quadratic error is trivially minimised if we take all regressors into the model, so the complete model will always be the optimal model in terms of the coefficient of determination. For model selection one uses variants of the quadratic error (resp. the coefficient of determination R^2) that are adjusted for the number of parameters.

a:ch04:141

| Exercise 4.9 | |
|--------------|--|
| * | <pre>Use library(leaps) lm.reg <- regsubsets(body.fat age + BMI + neck + chest + abdomen + hip + thigh + knee + ankle + bicep + forearm + wrist + weightkg + heightcm, data = fat) and inspect the result with summary(lm.reg) plot(lm.reg, scale = "r2") plot(lm.reg, scale = "bic") plot(lm.reg, scale = "Cp") Hint: see help(plot.regsubsets)</pre> |
| * | Use the function <code>leaps()</code> for model selection. |

But the tools which we discussed in chapter 2 have tarnished now. The statistical statements in the summary are only valid, if model resp. hypothesis are stated independent of the data material. If a model is determined based on the data material, the distribution of the estimated coefficients is unclear. We do not know how to define confidence intervals or how to perform tests. The software does not know that we are in a process of model selection and just returns the probabilities that apply for a fixed model under the assumption of a normal distribution.

The diagnostic becomes useless as well: the central limit theorem ensures that under weak independence conditions the residuals approximately have a normal distribution if we have a large number of terms, even if the normal distribution assumption does not hold for the errors themselves.

We are in a dead end.

We illustrate a different approach that carries further. For this we step back to the beginning of the analysis, after the preliminary inspection and data correction. To avoid running into the problem that the statistical distributions are influenced by preceding model selection steps, we split the data set. We use one part a training set, used for model selection and to play with various alternatives. The rest is reserved as evaluation set. The information from this part is only used after model selection for the statistical analysis.

A closer look shows that the model selection is most critical for the estimation of the error, not so much for parameter estimation. If the error is estimated based on the data used for model selection, we tend to under-estimate the error. The evaluation part serves for a reliable estimation of the error and for residual diagnostics. This is a limited task. We reserve only a smaller proportion of the data for this.

Input

```
sel <- runif(dim(fat)[1])
fat$train <- sel < 2/3
rm(sel)
```

We discard the outliers from the training part.

Input

```
fat$train[c(39, 41, 216)] <- FALSE
summary(fat$train)
```

| | Output | | prcomp@prcomp textit |
|---------|--------|------|----------------------|
| Mode | FALSE | TRUE | |
| logical | 87 | 165 | |

Our target variable is `body.fat`, or alternatively `1/density`.

We try to structure the variables with respect to the subject matter. For the density we have a definition from physics

$$\text{density} = \frac{\text{weight}}{\text{volume}}.$$

Among the variables that are candidate regressor we have just one variable that directly refers to weight (`weight` resp. `weightkg`), a series of variables, referring to body geometry, and `age`.

From the observed density and the observed weight we can retrieve the volume. We add this to our variable set. Since we have just one weight observation per person, there is no space for person related statistics.

Input

```
fat$vol <- fat$weightkg/fat$density
```

Next we try to model the volume `fat$vol` in terms of the more convenient variables on the body geometry. The body geometry reported are linear values. In a rough approximation, we can derive volume related variables. The only length information we have is in `height`. For the lack of better information we assume that that all part of the body have a length which is proportional to the height. We are aiming for a linear model. Hence linear factors can be neglected since they are estimated by the model anyhow. If we take cylindrical approximations for the parts of the body, up to linear factors we have

Input

```
fat$neckvol <- fat$neck^2 * fat$heightcm
fat$chestvol <- fat$chest^2 * fat$heightcm
fat$abdomenvol <- fat$abdomen^2 * fat$heightcm
fat$hipvol <- fat$hip^2 * fat$heightcm
fat$thighvol <- fat$thigh^2 * fat$heightcm
fat$kneevol <- fat$knee^2 * fat$heightcm
fat$anklevol <- fat$ankle^2 * fat$heightcm
fat$bicepvol <- fat$bicep^2 * fat$heightcm
fat$forearmvol <- fat$forearm^2 * fat$heightcm
fat$wristvol <- fat$wrist^2 * fat$heightcm
```

Next we inspect the internal structure of the volume oriented regressor candidates, using the training data set only. We use the function `prcomp()`, which for any given set of variables yields stepwise best linear predictors.

For the approximative volumes of the body parts, the principal components are:

Input

```
pcfatvol <- prcomp(fat[, 20:29], subset = fat$train)
round(pcfatvol$rotation, 3)
```

| | Output | | | | | | |
|------------|--------|--------|--------|--------|--------|--------|--------|
| | PC1 | PC2 | PC3 | PC4 | PC5 | PC6 | PC7 |
| neckvol | 0.054 | -0.017 | 0.063 | -0.070 | 0.570 | 0.045 | 0.787 |
| chestvol | 0.548 | 0.457 | 0.691 | 0.012 | -0.113 | -0.004 | -0.003 |
| abdomenvol | 0.650 | 0.290 | -0.697 | -0.059 | 0.049 | -0.011 | -0.017 |
| hipvol | 0.483 | -0.748 | 0.099 | 0.437 | -0.031 | 0.069 | 0.003 |
| thighvol | 0.185 | -0.370 | 0.074 | -0.870 | -0.244 | -0.017 | 0.079 |
| kneevol | 0.056 | -0.082 | 0.055 | -0.061 | 0.336 | -0.822 | -0.278 |
| anklevol | 0.016 | -0.032 | 0.034 | -0.005 | 0.140 | -0.236 | -0.047 |
| bicepvol | 0.051 | -0.047 | 0.079 | -0.180 | 0.552 | 0.508 | -0.531 |
| forearmvol | 0.024 | -0.019 | 0.074 | -0.086 | 0.388 | 0.019 | -0.104 |
| wristvol | 0.009 | -0.005 | 0.017 | 0.002 | 0.110 | -0.053 | 0.044 |
| | PC8 | PC9 | PC10 | | | | |
| neckvol | -0.169 | 0.067 | 0.093 | | | | |
| chestvol | -0.012 | 0.005 | 0.002 | | | | |
| abdomenvol | 0.016 | -0.015 | -0.002 | | | | |
| hipvol | 0.018 | 0.011 | 0.004 | | | | |
| thighvol | 0.012 | -0.015 | -0.019 | | | | |
| kneevol | -0.159 | 0.299 | 0.056 | | | | |
| anklevol | -0.120 | -0.949 | 0.073 | | | | |
| bicepvol | -0.333 | 0.030 | 0.010 | | | | |
| forearmvol | 0.906 | -0.049 | 0.044 | | | | |
| wristvol | 0.003 | -0.048 | -0.990 | | | | |

The pattern of the signs for the loadings gives hints about the internal structure. The first principal component $PC1$ is a linear combination of variables which essentially describes the torso. The second principal component contrasts the upper part of the torso (chest to abdomen) with the lower torso. The third contrasts the abdomen to the rest of the torso.

a:ch04:15

| | |
|----------------------|--|
| Exercise 4.10 | |
| | Draw a sketch that shows which body geometry features are represented by the next principal component $PC4, \dots, PC10$. |

ToDo:

discuss
scale dependency

To Do: use principal components in modelling. pc regression?

The attempt to represent the derived total volume by the approximative volumes of the body parts results in a high coefficient of determination.

```
lm.vol <- lm(vol ~ neckvol + chestvol + abdomenvol +  
    hipvol + thighvol + kneevol + anklevol + bicevpvol +  
    forearmvol + wristvol, data = fat, subset = fat$train)  
summary(lm.vol)
```

```
Call:  
lm(formula = vol ~ neckvol + chestvol + abdomenvol + hipvol +  
    thighvol + kneevol + anklevol + bicepvol + forearmvol + wristvol,  
    data = fat, subset = fat$train)
```

Residuals:

| | Min | 1Q | Median | 3Q | Max |
|--|---------|---------|--------|--------|--------|
| | -9.1317 | -0.9130 | 0.1768 | 1.1286 | 4.4042 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|------------|------------|---------|--------------|
| (Intercept) | 1.012e+00 | 1.465e+00 | 0.691 | 0.490552 |
| neckvol | 3.318e-05 | 9.612e-06 | 3.452 | 0.000719 *** |
| chestvol | 8.948e-06 | 1.287e-06 | 6.952 | 9.61e-11 *** |
| abdomenvol | 1.151e-05 | 1.207e-06 | 9.530 | < 2e-16 *** |
| hipvol | 9.427e-06 | 1.913e-06 | 4.928 | 2.12e-06 *** |
| thighvol | 9.827e-06 | 3.264e-06 | 3.010 | 0.003049 ** |
| kneevol | 4.224e-06 | 9.610e-06 | 0.440 | 0.660907 |
| anklevol | 4.285e-05 | 1.387e-05 | 3.090 | 0.002377 ** |
| bicepvol | 3.381e-05 | 8.023e-06 | 4.214 | 4.26e-05 *** |
| forearmvol | 1.246e-05 | 9.793e-06 | 1.272 | 0.205265 |
| wristvol | -5.996e-05 | 4.016e-05 | -1.493 | 0.137497 |
| --- | | | | |
| Signif. codes: | 0 *** | 0.001 ** | 0.01 * | 0.05 . |
| | 0.1 ' ' | 1 | | |

Residual standard error: 1.889 on 154 degrees of freedom
 Multiple R-squared: 0.9773, Adjusted R-squared: 0.9759
 F-statistic: 664.4 on 10 and 154 DF, p-value: < 2.2e-16

Taking the linear variables into the mode, for the training part we only get a neglectable improvement of the coefficient of determination.

Again we can use the functions from `library(leaps)` to search for “optimal” models.

Input

```
library(leaps)
l1 <- leaps(x = fat[, c(6:15, 20:29)], y = fat$vol)
```

If we want to model the proportion of body fat directly, not the volume, we can construct the corresponding auxiliary variables.

Input

```
fat$neckvolf <- fat$neckvol/fat$weightkg
fat$chestvolf <- fat$chestvol/fat$weightkg
fat$abdomenvolf <- fat$abdomenvol/fat$weightkg
fat$hipvolf <- fat$hipvol/fat$weightkg
fat$thighvolf <- fat$thighvol/fat$weightkg
fat$kneevolf <- fat$kneevol/fat$weightkg
fat$anklevolf <- fat$anklevol/fat$weightkg
fat$bicepvolf <- fat$bicepvol/fat$weightkg
fat$forearmvolf <- fat$forearmvol/fat$weightkg
fat$wristvolf <- fat$wristvol/fat$weightkg
```

We begin with a simple model. We use only one variable (`abdomenvolf`) from the group of variables describing the torso, and one variable (`wristvolf`) from the higher principal components. With this simple model, we achieve about the same quality as previously with the full set of variables.

```

predict.lm@predict.lm|textit
lm@lm|textit  lm.volf <- lm(body.fat ~ abdomenvolf + wristvolf,
                                Input _____
                                data = fat, subset = fat$train)
                                summary(lm.volf)

_____
                                Output _____
Call:
lm(formula = body.fat ~ abdomenvolf + wristvolf, data = fat,
subset = fat$train)

Residuals:
      Min        1Q    Median        3Q       Max
-9.13915 -3.15102 -0.06347  2.98484  8.59909

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.9316867  6.1596386 -0.151   0.88
abdomenvolf  0.0023648  0.0001835 12.889 < 2e-16 ***
wristvolf   -0.0331976  0.0049903 -6.652 4.22e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.093 on 162 degrees of freedom
Multiple R-squared:  0.6936,    Adjusted R-squared:  0.6898
F-statistic: 183.3 on 2 and 162 DF,  p-value: < 2.2e-16

```

So the recommendation would be: if you do not have a bath tube with you to determine the density, take a measurement on weight, height, abdomen and wrist.

| Exercise 4.11 | |
|---------------|---|
| * | Extend the variables by other volume related variables in the model given above. Do you gain precision? |
| ** | Try to include the variable <code>age</code> in the model. How do you exactly include <code>age</code> in the model? |
| ** | The function <code>mvr()</code> in <code>library(pls)</code> is available to perform a regression based on principal components. Use this function for regression. What is the difference between this estimation and usual least squares regression? |

For model construction, we used only the training part of the data. The quality of the model derived now can be checked using the evaluation part. This can be done using function `predict.lm()` which applies a model estimated with `lm()` to a new data set with analogous structure, for example

```

fat.eval <- fat[fat$train == FALSE, ]
pred <- predict.lm(lm.volf, fat.eval, se.fit = TRUE)
_____
                                Input _____

```

a:ch04:17

| Exercise 4.12 | | <small>page@image textit</small> |
|----------------------|---|----------------------------------|
| * | Estimate the precision of the model using the evaluation part of the data. | |
| * | Carry out a regression diagnostics of the model derived, using the evaluation part of the data. | |

4.8 High Dimensions

In small dimensions, we can present an analyse many problems completely. Higher dimension often require to design special strategies for analysis. Formal applications of standard methods soon meets its limitations.

Higher dimensions, such as dimension 10 through 100, are common in many application fields. But even problems in high dimensions occur day-to-day. We have to see that dimension is a question of modelling, not a mere question of the problem. Digital video (DV PAL) for example records images in a format of 720×576 . A single image with three colours hence gives a vector in $720 \times 576 \times 3 = 1244160$ dimensional space. A second of video has 25 of these frames. If we have to work with image data, it is our choice to see image processing as a problem in dimension $d = 1244160$, or as sequence of 1244160 (not independent!) observations in dimension $d = 1$.

Going from dimension $d = 1244160$ to $d = 1$ we shift information that is implicit in the dimensions to structural information, which has to be modelled then.

As a remark: practical application takes a middle course. The image is decomposed into blocks, for example of size 64×64 . Pixels within a block are handled simultaneously; the blocks are handled sequentially. You can see it the next time the TV has a glitch.

In high dimensional problems, the statistics is often not apparent at all. It is hidden in the hardware as "imbedded system".

The figure 4.7 is an example from an analysis with R for a high dimensional data set of cDNA micro array data ([23]). A single observation in this data set consists of measurements on 4227 probes, each with 4 partial recordings (`fg.green`, `fg.red`, `bg.green`, `bg.red`). The essential function, which is used for visualisation here is `image()`. It is used here to represent a variable `z` using a colour table against two co-ordinates `x`, `y` which give the position of a spot on a slide. The image shows one observation. The four channels for the partial recordings are put on different displays side by side. The colours code the result of a pre-analysis - the red point mark problem areas on the cDNA-Chip. In this case, the pattern of the highlighted selection could identify a specific problem in the production process.

ToDo:
cDNA:
details

Theme oriented surveys about R packages, in particular for multivariate problems, are to be found in <<http://cran.r-project.org/src/contrib/Views/>>.

4.9 Statistical Summary

The analysis of multivariate data can only be touched in this context. Multivariate problems already occur implicitly for regression problems (see chapter 2). For simple regression problems, the multivariate aspects only referred to deterministic parameters. In the general case we have to analyse a multivariate statistical distribution. The introduction has to stop here, and more is reserved to continuing lectures.

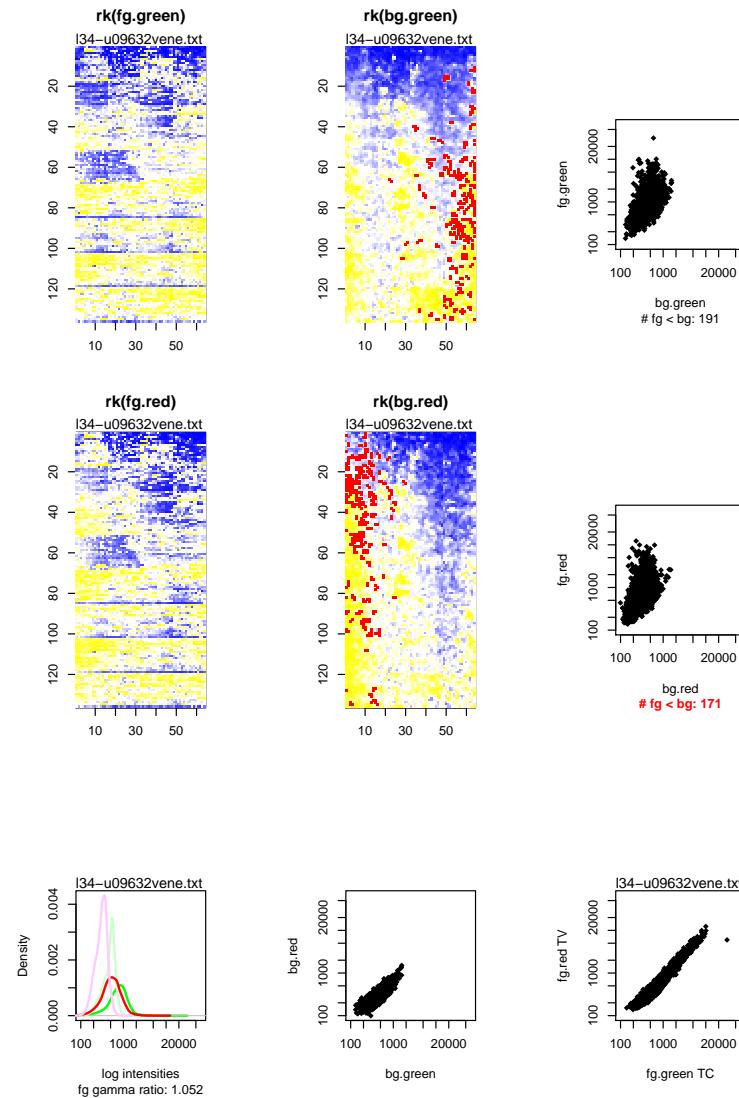
Figure 4.7 A 4227×4 observation from a micro array experiment

fig:cDna

4.10 Literature and Additional References

Generated by Sweave from:

```
+ Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/Rnw/S04multiv.Rnw.tex,v +
+ Revision: 1.20 +
+ Date: 2008/03/18 09:01:46 +
+ name: +
+ Author: j40 +
+Source : /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/Rnw/S04multiv.Rnw.tex,v +
+Revision : 1.20 +
+Date : 2008/03/18 09:01:46 +
+name : +
+Author : j40 +
```



R as a Programming Language and Environment

ch:0A

R is an interpreted expression language. Expressions are composed of objects and operators.

A.11 Help and Information

| R <i>help</i> | |
|----------------------|--|
| <i>help()</i> | Information about an object/a function <i>Example:</i> <code>help(help)</code> |
| <i>args()</i> | Shows arguments of a function |
| <i>example()</i> | Executes examples, if available <i>Example:</i> <code>example(plot)</code> |
| <i>help.search()</i> | Searches for information about an object/a function |
| <i>apropos()</i> | Localises by keyword |
| <i>demos()</i> | Executes demos for a topic area <i>Example:</i> <code>demos(graphics)</code> <i>demos()</i> lists all topic areas which provide a demo |

A.12 Names and Search Paths

Objects are identified by names. By the name objects are searched in a chain of search areas. The search areas in effect can be inspected with *search()*.

| R <i>Search Paths</i> | |
|-----------------------|---|
| <i>search()</i> | List the search areas in effect, beginning with <i>.GlobalEnv</i> down to the base package <i>package:base</i> . <i>Example:</i> <code>search()</code> |

(cont.)→

| | |
|---|---|
| <code>mode@mode text length@length typeof@typeof </code> | <p>R Search Paths (cont.)</p> |
| | <p>searchpaths() List the access paths for the search areas in effect <i>Example:</i> <code>searchpaths()</code></p> |
| | <p>objects() List the objects in a search path <i>Examples:</i> <code>objects()</code> <code>objects("package:base")</code></p> |
| | <p>ls() List the objects in a search path <i>Examples:</i> <code>ls()</code> <code>ls("package:base")</code></p> |
| | <p>ls.str() List the objects and their structure in a search path <i>Examples:</i> <code>ls.str()</code> <code>lsf.str("package:base")</code></p> |
| | <p>find() Localises by keyword. Does also find overlaid entries <i>Syntax:</i> <code>find(what, mode = "any", numeric = FALSE, simple.words = TRUE)</code></p> |
| | <p>apropos() Localises by keyword. Does also find overlaid entries <i>Syntax:</i> <code>apropos(what, where = FALSE, ignore.case = TRUE, mode = "any")</code></p> |

Functions can be nested. This may occur at definition time as well as at execution time. This requires an extension of the search paths. The dynamic identification of objects uses environments to resolve local or global variables in functions.

| | |
|--|---|
| | <p>R Search Paths (cont.)</p> |
| | <p>environment() Current environments <i>Example:</i> <code>environment()</code></p> |
| | <p>sys.parent() Preceding environments <i>Example:</i> <code>sys.parent(1)</code></p> |

Objects have two implicit attributes that can be queried with `mode()` and `length()`. The function `typeof()` gives the (internal) storage modus of an object.

A `class` attribute gives the class of an object.

A.13 Customizing

R offers a series of possibilities to configure the system, so that certain commands are exe-

cuted upon start or termination. When starting, the files `.Rprofile` and `.RData` are read and executed if available. Details can be system specific. The appropriate information is given by `help(Startup)`.

A.14 Basic Data Types

s:A.1

| <i>Basic R Data Types</i> | |
|---------------------------|---|
| <i>numeric</i> | <i>real</i> or <i>integer</i> . In R: real numbers are always in double precision. Single precision is supported for external call to other languages with .C or .FORTRAN. Functions can <i>mode()</i> and <i>typedef()</i> can show the storage modus (single, double ...), depending on the implementations. <i>Examples:</i> 1.0 2 3.14E0 |
| <i>complex</i> | Complex, in Cartesian co-ordinates <i>Example:</i> 1.0+0i |
| <i>logical</i> | TRUE, FALSE. In R, T and F are predefined variables provided as an alternative. In S-Plus, T and F basic objects. |
| <i>character</i> | Character strings. Delimiter are alternatively " or '. <i>Example:</i> "T", 'klm' |
| <i>list</i> | General list structure. List elements can be of different types. <i>Example:</i> list(1:10, "Hello") |
| <i>function</i> | R function <i>Example:</i> sin |
| <i>NULL</i> | Special case: empty object <i>Example:</i> NULL |

In addition to TRUE and FALSE there are three special values for exceptional situations:

| <i>Special Constants</i> | |
|--------------------------|--|
| <i>TRUE</i> | Alternative: <i>T</i> . Type: logical. |
| <i>FALSE</i> | Alternative: <i>F</i> . Type: logical. |
| <i>NA</i> | "not available". Type: logical. NA is different from TRUE and FALSE |
| <i>NaN</i> | "not a valid numeric value". Implementation dependent. Should follow the IEEE Standard 754. Type: numeric. <i>Example:</i> 0/0 |

(cont.)→

| | |
|-------------------------------------|---|
| <i>Special Constants</i> (cont.) | |
| <i>Inf</i> | Infinite. Implementation dependent. Should follow the IEEE Standard 754. Type: numeric. <i>Example:</i> 1/0 |

polymorphic

A.15 Output for Objects

The object attributes and can be queried or displayed using output routines. The output routines generally are *polymorphic*, that is they come with variants adapted to the given object type.

| R <i>Inspection</i> | |
|---------------------|--|
| <i>print()</i> | Standard-output |
| <i>structure()</i> | output, optional with attributes |
| <i>summary()</i> | Standard output as summary, in particular for model fits |
| <i>plot()</i> | standard graphic output |

A.16 Object Inspection

The following table summarises the most important information possibilities about objects.

| <i>Object Inspection</i> | |
|------------------------------------|--|
| <code>str()</code> | Shows the internal structure of an object in compact form. Syntax: <code>str(<object>)</code> |
| <code>structure()</code> | Shows the internal structure of an object. Attributes for the display can be passed as parameter. Example: <code>structure(1:6, dim = 2:3)</code> Syntax: <code>structure(<object>, ...)</code> |
| <code>class()</code> | Object class. For newer objects, the class is stored as an attribute. In older S or R versions, the class is determined implicitly by type and other attributes. |
| <code>mode()</code> | Mode (type) of an object. |
| <code>storage.mode()</code> | Storage mode of an object. |
| <code>typeof()</code> | Mode of an object. May be different from the storage mode. Depending on the implementation a numerical variable for example can be stored in double precision (the default) or in single precision.) |
| <code>length()</code> | Length = number of elements |
| <code>attributes()</code> | Reads/sets attributes of an object, such as names, dimensions, classes. |
| <code>names()</code> | Names attribute for elements of an object, for example a vector. Syntax: <code>names(<obj>)</code> Gives the <code>names</code> attribute of <code><obj></code> . <code>names(<obj>)<-<charvec></code> sets the <code>names</code> attribute. Example: <code>x<-values names(x)<- <charvec></code> |

```
trellis.par.set@trellis.par.set@textit
lattice.options@lattice.options@textit
```

The following table summarises the most important information possibilities about the general system environment.

| <i>System Inspection</i> | |
|--------------------------|---|
| <code>search()</code> | Current search path |
| <code>ls()</code> | Objects in current search path |
| <code>methods()</code> | Generic methods Syntax: <code>methods(<fun>)</code> shows specialised functions for <code><fun></code> , <code>methods(class = <c>)</code> the class specific functions for class <code><c></code> . Examples: <code>methods(plot)</code> <code>methods(class = lm)</code> |
| <code>data()</code> | Accessible data |
| <code>library()</code> | Accessible libraries |
| <code>help()</code> | General help system |
| <code>options()</code> | Global options |
| <code>par()</code> | Parameter settings for the Graphic system |

The options of the `lattice` systems can be controlled with `trellis.par.set()` resp. `lattice.options()`.

R is anchored in the host operating system. Some variables such as access paths, encoding etc. are imported from there.

| <i>System Environment</i> | |
|---------------------------|---|
| <code>getwd()</code> | Get current working directory |
| <code>setwd()</code> | Set current working directory |
| <code>dir()</code> | List files in the current working directory |
| <code>system()</code> | calls system functions |

A.18 Complex Data Types

The interpretation of basic types or derived types can be specified by one or more `class` attributes. Polymorphic functions such as `print` or `plot` evaluate this attribute and call a variant for this class if available (see [2.6.5 Seite 2-42](#)).

For the storage of dates and times, special classes are provided. For more information on these data types see

`help(DateTimeClasses)`.

R is vector based. Individual constants or values just are vectors the special length 1. They do not get a special treatment.

| <i>Compound Data Types</i> | |
|----------------------------|--|
| vectors | Basic R data types |
| Matrices | Vectors with two dimensional layout |
| Arrays | <p>Vectors with higher dimensional layout</p> <p><code>dim()</code> defines a dimension attribute</p> <p><i>Example:</i> <code>x <- runif(100)</code> <code>dim(x) <- c(5, 5, 4)</code></p> <p><code>array()</code> generates a new vector with specified dimension structure</p> <p><i>Example:</i> <code>z <- array(0, c(4, 3, 2))</code></p> <p><code>rbind()</code> combines by rows</p> <p><code>cbind()</code> combines by columns</p> |
| factors | <p>special case for categorical data</p> <p><code>factor()</code> converts a vector into a factor</p> <p><i>See also</i> section 2.2.1</p> <p><code>ordered()</code> converts a vector into a factor with ordered levels. This is a shortcut for <code>factor(x, ..., ordered = TRUE)</code></p> <p><code>levels()</code> returns the levels of a factor</p> <p><i>Example:</i> <code>x <- c("a", "b", "a", "c", "a")</code> <code>xf <- factor(x)</code> <code>levels(xf)</code> results in <code>[1] "a" "b" "c"</code></p> <p><code>tapply()</code> applies a function separately for all levels of factors in a list</p> |

(cont.)→

| <i>Compound Data Types</i> (cont.) | |
|---------------------------------------|---|
| Lists | <p>analogous to vectors, with elements of possibly different types</p> <p><code>list()</code> generates a list</p> <p>Syntax: <code>list(<components>)</code></p> <p><code>[[]]</code> access to components of a list by index</p> <p><code><list\$component></code> access by names</p> <p>Example: <code>l <- list(name = "xyz", age = 22, fak = "math")</code> <code>>l[[2]]</code> <code>22</code> <code>>l\$age</code> <code>22</code></p> |
| data frames | <p>data frames analogous to arrays resp. lists, with column-wise uniform type and uniform column length</p> <p><code>data.frame()</code> analogous to <code>list()</code>, but restrictions have to be satisfied</p> <p><code>attach()</code> attaches a data to the current search list. For access to components the component name will be sufficient.</p> <p><code>detach()</code></p> |

A.19 Accessing Components

A.2

The length of vectors is a dynamic attribute. It is extended or shortened as needed. In particular, an implicit ‘recycling rule’ applies: if a vector does not have the length necessary for some operation, it is repeated periodically up to the length required.

Vector components can be accessed by index. The indices can be specified explicitly or in form of an expression rule.

| <i>Accessing Components</i> | |
|-----------------------------------|--|
| <code>x[<indices>]</code> | Indicated components of <i>x</i> <i>Example:</i> <code>x[1:3]</code> |
| <code>x[-<indices>]</code> | <i>x</i> omitting indicated components <i>Example:</i> <code>x[-3]</code> <i>x</i> omitting 3. components |
| <code>x[<condition>]</code> | components of <i>x</i> , for which the <code><condition></code> holds. <i>Example:</i> <code>x[x < 0.5]</code> |

vectors (and other objects) can be mapped to higher dimensional constructs. The image is described by additional attributes. By convention the imbedding goes by column, that is the first index varies first (FORTRAN convention). Operators and functions can evaluate the dimension attributes.

| <i>R Index Access</i> | |
|-------------------------|---|
| <code>dim()</code> | Gets or sets dimensions of an object <i>Example:</i> <code>x <- 1:12 ; dim(x) <- c(3, 4)</code> |
| <code>dimnames()</code> | Gets or sets names for the dimensions of an object |
| <code>nrow()</code> | gives the number of rows = dimension 1 |
| <code>ncol()</code> | gives the number of columns = dimension 2 |
| <code>matrix()</code> | generates a matrix with given specifications <i>Syntax:</i> <code>matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)</code> <i>See also</i> <code>example(ch01-MC06-ch01-MC06)</code> <code>array()</code> (page II-23) |
| <code>array()</code> | generated a possibly higher dimensional matrix <i>Example:</i> <code>array(x, dim = length(x), dimnames = NULL)</code> |

| R <i>Array Access</i> | |
|--|-----------------------------|
| <code>cbind()</code> <code>rbind()</code> | combines columns resp. rows |
| <code>split()</code> | splits a vector by factors |
| <code>table()</code> | generates a table of counts |

| R <i>Iterators</i> | |
|--------------------------|---|
| <code>apply()</code> | applies a function to the rows or columns of a matrix Syntax: <code>apply(x, MARGIN, FUNCTION, ...)</code> MARGIN = 1: rows, MARGIN = 2: columns See also <code>example(ch01-MC06)</code> <small>(page II-23)</small> |
| <code>lapply()</code> | applies a function to the elements of a list Syntax: <code>lapply(X, FUN, ...)</code> |
| <code>sapply()</code> | applies a function to the elements of a list, of a vector or a matrix. If possible, dimension names are carried over. Syntax: <code>sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)</code> |
| <code>tapply()</code> | applies a function to components of an object depending on a list of controlling factors. |
| <code>by()</code> | object-orientierte Variante of <code>tapply</code> Syntax: <code>by(data, INDICES, FUN, ...)</code> |
| <code>aggregate()</code> | calculates statistics for subsets Syntax: <code>aggregate(x, ...)</code> |
| <code>replicate()</code> | evaluates an expression repeatedly (for example with generating random numbers for simulation). Syntax: <code>replicate(n, expr, simplify = TRUE)</code> |
| <code>outer()</code> | generates a matrix with all pair-wise combinations from two vectors, and applies a function to each pair. Syntax: <code>outer(vec1, vec2, FUNCTION, ...)</code> |

A.20 Data Manipulation

| <i>Transformations</i> | |
|---------------------------|---------------------|
| <code>seq()</code> | generate a sequence |
| <code>abbreviate()</code> | |

| <i>Transformations</i> | |
|---------------------------|---|
| <code>duplicated()</code> | checks for duplicate or multiple values |
| <code>unique()</code> | generates a vector without multiple values |
| <code>match()</code> | gives first position of a value in a vector |
| <code>pmatch()</code> | partial matching |

| <i>Character String Transformations</i> | |
|---|--|
| <code>casefold()</code> | translates characters, in particular from upper to lower case or vice versa. |
| <code>tolower()</code> | translates to lower case |
| <code>toupper()</code> | translates to upper case |
| <code>chartr()</code> | translates characters in a character vector |
| <code>substring()</code> | |

| <i>Transformations</i> | |
|----------------------------|---|
| <code>table()</code> | generates a table of counts |
| <code>expand.grid()</code> | generates a data frame with all combinations of the factors given |
| <code>reshape()</code> | convert between a cross classification table (column per variable) and a long table (variables in rows, with additional indicator column) |
| <code>merge()</code> | merges data frames |

| <i>Transformations</i> | |
|------------------------|--|
| | |

ToDo:
 extend reshape data frames
 Harrell
 4.2.7
ToDo:
 add merge data frames
 Harrell
 4.2.5ff

| <i>Transformations</i> | |
|------------------------|---|
| <code>t()</code> | transposes rows and columns Syntax: <code>t(x)</code> |
| <code>aperm()</code> | generalised permutation Syntax: <code>aperm(x, perm)</code> where <code>perm</code> is a permutation of the indices of <code>x</code> . |
| <code>split()</code> | splits a vector by a factor |
| <code>unsplit()</code> | combines components to a vector |

ToDo:

move to
cbind etc

A.21 Operators

s:A.4

Expressions in R can be composed of objects and operators. The following table of operators is ordered by precedence (highest rank on top). See *help(Syntax)*.

| <i>Basic R operators</i> | |
|--------------------------|--|
| \$ | components-selection <i>Example:</i> <code>list\$item</code> |
| [[| indexing, access to elements <i>Example:</i> <code>x[i]</code> |
| ^ | exponentiation (right to left) <i>Example:</i> <code>x^3</code> |
| - | unary minus |
| : | sequence generation <i>Examples:</i> <code>1:5</code> <code>5:1</code> |
| %<name>% | special operators. Can also be user defined. <i>Examples:</i> <code>"%deg2%"<-function(a, b) a + b^2</code> <code>2 %deg2% 4</code> |
| * / | multiplication, division |
| + - | addition, subtraction |
| < > < = > = == ! = | comparison operators |
| ! | negation |
| & && | and, or &&, are “shortcut” operators |
| <- -> | assignment |

If operand do not have the same length, the shorter operand is repeated cyclically.

Operators of the form %<name>% can be defined by the user. The definition follows the rules for function definitions.

Expressions can be write as a sequence with separation semicolons. Expression groups can be combined by {...}.

function|**ToDs:A.5**

superas-

signment

introR

10.5

A.22 Functions

Functions are special objects. Functions can return objects as results.

| <i>R Function Declarations</i> | |
|--------------------------------|--|
| Declarations | <code>function (<formal parameter list>) <expression></code> <i>Example:</i> <code>fak <- function(n) prod(1:n)</code> |
| Formal parameter | <code><parameter name></code> <code><parameter name> = <default value></code> |
| Formal parameter list | list of formal parameter, separated by commas <i>Examples:</i> <code>n, mean = 0, sd = 1</code> |
| ... | Variable parameter list. Variable parameter lists can be propagated to imbedded functions. <i>Example:</i> <code>mean.of.all <- function (...)mean(c(...))</code> |
| Funktion result | <code>return <value></code> stops function evaluation and returns value <code><value></code> as last expression in a function declaration: returns value |
| Function result | <code><Variable><-<value></code> returns value. In general, assignments operate only on local copies of the variable. The assignment with <code><-</code> however looks for the target in the complete search chain. |

| <i>R Function Call</i> | |
|-------------------------------|--|
| Function call | <code><name>(<Current parameter list>)</code> <i>Example:</i> <code>fak(3)</code> |
| Current parameter list | values are matched by position. Deviating from this, names can be used to control the matching. Initial parts of the names suffice (Exception: after a variable parameter list, names must be given completely). Function <code>missing()</code> can be used to check, whether for a corresponding actual parameter is missing for a formal parameter. <i>Syntax:</i> <code><list of values></code> <code><parameter name> = <values></code> <i>Example:</i> <code>rnorm(10, sd = 2)</code> |

Parameter for functions are passed by. This helps consistency, but involves overhead for memory

management and copying. If this overhead needs to be avoided, the information provided by [environment@environment | R: Gentleman+Ihaka:2000](#) `environment()` allows direct access to variables. Techniques to use this are described in [7].

Special case: Functions with names of the form `xxx<-` extend the assignment function. *Example:*

```
"inc<-" <-function (x, value) x+value
x <- 10
inc(x)<- 3
x
```

In R assignment functions the value argument **must** be called “value”.

Debugging
Profiling

A.23 Debugging and Profiling

```
debug@debug|textit
browser@browser|textit
debug@debug|textit R provides a collection of tools for identification of errors. These are particularly helpful in
Topic connection with functions. browser() can be used to switch to a browser mode. In this mode,
de- the usual R instructions can be used. Besides this, there is a small number of special instructions.
bug- With debug(), the browser mode is activated automatically upon entry to a function. The
ging!browser@browser browser mode is marked by a special prompt Browse[xx]>.
Topic
de- n goes to the next instruction, if the function is under control of debug. Continuous
bug- goes to the next instruction (also if browser has been called directly).
Topic
de- cont Continuous with the expression evaluation.
bug- c Short for cont. Continuous the expression evaluation.
ging!traceback@traceback
```

where Shows call nesting.

Q Stop execution and jumps back to base state.

| Debug Help | |
|-------------------|--|
| browser() | Suspends execution and enters the browser mode. Syntax: browser() |
| recover() | recover() shows a list of the current call hierarchy. An entry from this list can be chosen for inspection by browser() . With c you leave the browser and return to recover . With 0 you leave recover() Syntax: recover() Hint: With options(error = recover) , error handling for a function is directed to call browser() automatically in case of an error. |
| debug() | Marks a function for debugger control. On subsequent calls to the function, the debugger is activated and switches to browser mode. Syntax: debug(<function>) |
| undebug() | Cancels debugger control for a function. Syntax: undebug(<function>) |
| trace() | Marks a function for trace control. On subsequent calls to the function, the call is signaled together with its arguments. Syntax: trace(<function>) |

(cont.)→

| | |
|------------------------------|--|
| Debug Help (cont.) | |
| untrace() | Cancels trace control for a function. Syntax: <code>untrace(<function>)</code> |
| traceback() | In case of error inside of a function the current calling stack is stored in a variable <code>.Traceback</code> . <code>traceback()</code> evaluates this variable and displays its content. Syntax: <code>traceback()</code> |
| try() | Allows for user defined error handling. Syntax: <code>traceback(<expression>)</code> |

To measure execution time in selected code ranges, R provides a “profiling”. However this is only available if R has been compiled with the appropriate options. The options innovated at compiling can be queried using `capabilities()`.

| | |
|-----------------------------|--|
| Profiling-Hilfen | |
| <code>system.time()</code> | Returns the execution time of an expression. This function is available in all implementations. Syntax: <code>system.time(<expr>, <gcFirst>)</code> |
| <code>Rprof()</code> | Records active functions periodically. This function is only available, if R has been compiled for “profiling”. With <code>memory.profiling = TRUE</code> , in addition to the timing the memory usage is recorded periodically. This option is only available if R has been compiled correspondingly. Syntax: <code>Rprof(filename = "Rprof.out", append = FALSE, interval = 0.02, memory.profiling = FALSE)</code> |
| <code>Rprofmem()</code> | Records memory requirements on demand. This function is only available if R has been compiled for “memory profiling”. Syntax: <code>Rprofmem(filename = "Rprofmem.out", append = FALSE, threshold = 0)</code> |
| <code>summaryRprof()</code> | Summarizes the output of <code>Rprof()</code> and reports the times by function. Syntax: <code>summaryRprof(filename = "Rprof.out", chunksize = 5000, memory = c("none", "both", "tseries", "stats"), index = 2, diff = TRUE, exclude = NULL)</code> |

`function()`
[A:ctl]

A.24 Control Structures

| R Control Structures | |
|----------------------|---|
| if | conditional execution Syntax: <code>if (<log. expression 1>) <expression2></code> The logical expression1 may return only one logical value. For vector oriented access use <code>ifelse</code> . Syntax: <code>if (<log. expression1>) <expression2> else <expression3></code> |
| ifelse | Element wise conditional execution Syntax: <code>ifelse(<log. expression1>, <expression2>, <expression3>)</code> evaluates the logical expression1 elementwise on a vector, and returns expression2 if the evaluation gives true, else expression3) Example: <code>trimmedX <- ifelse (abs(x)<2, X, 2)</code> |
| switch | evaluates an expression and executes an instruction based on the result Syntax: <code>switch(<expression1>, ...)</code> expression1 must return a numeric value or a character string. ... is an explicit list of alternative actions. Example: <code>centre <- function (x , type) { switch(type, mean = mean(x), median = median(x), trimmed = mean(x, trim = .1)}</code> |
| for | iteration (loop) Syntax: <code>for (<name> in <expression1>) <expression2></code> |
| repeat | iteration. Must be terminated explicitly, for example with <code>break</code> . Syntax: <code>repeat <expression></code> Example: <code>pars<-init repeat { res<- get.resid (data, pars) if (converged(res)) break pars<-new.fit (data, pars)}</code> |
| while | Conditional repetitions Syntax: <code>while (<log. expression>) <expression></code> Example: <code>pars<-init; res <- get.resid (data, pars) while (!converged(res)) { pars<- new.fit(data, pars) res<- get.resid}</code> |

(cont.)→

| | |
|--|--|
| <i>R Control Structures</i> (cont.) | |
| <i>break</i> | terminates the current loop |
| <i>next</i> | terminates the current loop cycle and advances to next |

A.25 Administration and Customisation

s:A.7

| <i>objects()</i> <i>ls()</i> | list the objects in the current search path |
|---------------------------------|---|
| <i>rm()</i> | removes indicated objects Syntax: <i>rm(⟨object list⟩)</i> |

A.26 Input and Output to Data Streams

s:A.8

| R <i>Input/Output</i> | |
|-----------------------|---|
| <code>write()</code> | Writes data to a file. <i>Syntax:</i> <code>write(val, file)</code> <i>Example:</i> <code>write(x, file = "data")</code> |
| <code>source()</code> | Executes the R instruction from the file indicated. <i>Syntax:</i> <code>source("<file name> ")</code> <i>Example:</i> <code>source("cmnds.R")</code> |
| <code>sink()</code> | Redirect output in the file specified. <i>Syntax:</i> <code>sink("<file name>")</code> <i>Example:</i> <code>sink()</code> redirect the output back to the console. |
| <code>dump()</code> | Writes the commands defining an object. The object can be regenerated from this output using <code>source()</code> <i>Syntax:</i> <code>dump(list, file = "<dumpdata.R>", append = FALSE)</code> |

`edit@edit|textit` **A.27 External Data**

```
data@data@data.entry|textit
save@save|textit
load@load|textit To edit or enter data using a spreadsheet model, R provides edit() (previously called data.entry()).
data@data|textit For exchange, the data formats have to be harmonised between all parties. For import from
data bases or other systems, several packages are available, for example stataread for Stata,
foreign for SAS, Minitab and SPSS, RODBC for SQL. For more information, see the manual
“Data Import/Export” ([15]).
```

Within R, prepared data are usually provided as *data frames*. If additional objects such as functions or parameters are necessary, they can be made accessible in bundled form as packages (see exercise A.28 (Seite A-76)).

For the exchange from R to R, a special exchange format can be used. Files in this format can be generated with `save()` and conventionally have the name suffix `.Rda`. These files can be loaded again using `load()`.

A general purpose function to load data us `data()`. Depending on the suffix of the input file name, `data()` branches for several special cases. Besides `.Rda` usual suffixes for data input files are `.tab` or `.txt`. The online help function `help(data)` gives additional information.

| <i>Data Input/Output for R</i> | |
|--------------------------------|--|
| <code>save()</code> | stores data in an external file. <i>Syntax:</i> <code>save(<names of the objects to be stored>, file = <file name>, ...)</code> |
| <code>load()</code> | loads data from an external file. <i>Syntax:</i> <code>load(file = <file name>, ...)</code> |
| <code>data()</code> | loads data. <code>data()</code> can handle various file formats, if the access paths and file-names the follow R conventions. <i>Syntax:</i> <code>data(..., list = character(0), package = c(.packages(), .Autoloaded), lib.loc = .lib.loc)</code> <i>Example:</i> <code>data(crimes) # lädt die data set 'crimes'</code> |

For the flexible exchange with other programs in general text based files are provided. Some conventions can make exchange easier:

- in table form,
- only ASCII characters (z.B. no umlaut!)
- variables arranged in columns
- columns separated by tabulator stops
- possibly a column header in row 1
- possibly a row label in column 1

For reading the function `read.table()` is provided, and for writing, there is `write.table()`. Besides `read.table()` there are several variants which are adapted to usual data formats. These are documented under `help(read.table)`.

| <i>Input and Output of Data for Exchange</i> | |
|--|--|
| <code>read.table()</code> | <p>read data tables</p> <p>Syntax: <code>read.table(file, header = FALSE, sep = "\t", ...)</code></p> <p>Examples: <code>read.table(<file name>, header = TRUE, sep = '\t')</code> headers in row 1, row labels in column 1 <code>read.table(<file name>, header = TRUE, sep = '\t')</code> now row number, headers in row 1,</p> |
| <code>write.table()</code> | <p>writes data table</p> <p>Syntax: <code>write.table(file, header = FALSE, sep = '\t', ...)</code></p> <p>Examples: <code>write.table(<data frame>, <file name>, header = TRUE, sep = '\t')</code> headers in row 1, row labels in column 1 <code>write.table(<data frame>, <file name>, header = TRUE, sep = '\t')</code> now row number, headers in row 1,</p> |

```

read.table@read.table|tex
write.table@write.table|tex
read.table@read.table|tex
DateTimeClasses
Datum!↳see
    DateTimeClasses
Zeit!↳see DateTimeClasses
scan@scan|textit
read.fwf@read.fwf|textit

```

By default, `read.table()` converts data to `factor` variables if possible. This behaviour can be modified with the parameter `as.is` when calling of `read.table()`. This modification is for example necessary to read date and time information as for example in the following example from [8]:

```

# date col in all numeric format yyyyymmdd
df <- read.table("laketemp.txt", header = TRUE)
as.Date(as.character(df$date), "%Y-%m-%d")
# first two cols in format mm/dd/yy hh:mm:ss
# Note as.is = in read.table to force character
library("chron")
df <- read.table("oxygen.txt", header = TRUE,
as.is = 1:2)
chron(df$date, df$time)

```

For sequential reading, `scan()` if provided. Files with data in fixed format (by character columns) can be read with `read.fwf()`.

A.28 Libraries, Packages

External information can be stored in (text) files and Packages. Libraries and packages follow special R conventions. “Libraries” are collections of “packages”. In general, additional function are provided as packages. Packages are loaded with
library().
Data sets contained in the package are then included in the search path and can be listed with
data()
(no arguments).
example:

```
library(nls)
data()
data(Puromycin)
```

| <i>packages</i> | |
|---------------------------|--|
| library() | loads add-on package Syntax: library(package, ...) See also section I.5.6 |
| require() | loads add-on package; gives warning on error. Syntax: require(package, ...) |
| detach() | releases add-on package and removes it from the search path. Syntax: detach(<name>) |
| install.packages() | Installs add-on package in <lib> , downloading it from the archive CRAN or from specified archives Syntax: install.packages(pkgs, lib, CRAN =getOption("CRAN"), ...) |
| package.manager() | if implemented: interface for management of installed packages. Syntax: package.manager() |
| package.skeleton() | creates a skeleton for a new package. Syntax: package.skeleton(name = "<anRpackage>", list, ...) |

Detailed information for building R packages is in in “Writing R Extensions” ([\[18\]](#)). R:Ext

A.29 Linear Algebra

For linear algebra, the most important functions are widely standardised and implemented in C libraries such as BLAS/ATLAS and Lapack. R makes use of these libraries and provides an interface to the most important functions.

| <i>Linear Algebra</i> | |
|----------------------------|---|
| <code>eigen()</code> | computes eigenvalues and eigenvectors of real or complex matrices |
| <code>svd()</code> | singular value decomposition of a matrix |
| <code>qr()</code> | QR decomposition of a matrix |
| <code>determinant()</code> | determinant of a matrix |
| <code>solve()</code> | solves linear equations |

If possible, statistical functions should be used and direct access to the linear algebra functions should be avoided.

A.30 Model Descriptions

s:A.10

Mathematically, linear statistical models can be specified by a design matrix X and written generally as

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \varepsilon,$$

where the matrix \mathbf{X} has to be specified.

R allows to specify models by giving the rules how to build the design matrix.

| Operator | Syntax | Meaning | Example |
|---------------------|-----------------------------|--|--|
| \sim | $Y \sim M$ | Y depends on M ; | $Y \sim X$ results in $E(Y) = a + bX$ |
| $+$ | $M_1 + M_2$ | include M_1 and M_2 | $Y \sim X + Z$ $E(Y) = a + bX + cZ$ |
| $-$ | $M_1 - M_2$ | include, M_1 , but exclude M_2 | $Y \sim X - 1$ $E(Y) = bX$ |
| : | $M_1 : M_2$ | tensor product, that is all combinations of levels of M_1 and M_2 | |
| $\% \text{ in } \%$ | $M_1 \% \text{ in } \% M_2$ | modified tensor product | $a + b \% \text{ in } \% a$ corresponds to $a + a : b$ |
| * | $M_1 * M_2$ | “crossed” | $M_1 + M_2$ corresponds to $M_1 + M_2 + M_1 : M_2$ |
| / | M_1 / M_2 | “nested”: $M_1 + M_2 \% \text{ in } \% M_1$ | |
| n | M^n | M with all “interactions” up to level n | |
| $I()$ | $I(M)$ | interpret M . Terms in M retain their original meaning; the result determines the model. | $Y \sim (1 + I(X^2))$ corresponds to $E(Y) = a + bX^2$ |

Table A.53 Wilkinson-Rogers Notation for Linear Models

ta:wrn

The model specification is also possible for generalised (not linear) models.

examples

$$\begin{aligned} y &\sim 1 + x \\ y &\sim x \end{aligned} \quad \begin{aligned} \text{corresponds to } y_i &= (\mathbf{1} \ x_i)(\boldsymbol{\beta}_1 \ \boldsymbol{\beta}_2)^\top + \varepsilon \\ &\text{short for } y \sim 1 + x \\ &\text{(a constant term is assumed implicitly)} \end{aligned}$$

| | | |
|--------------------------|---|------------------------|
| $y \sim 0 + x$ | corresponds to $y_i = x_i \cdot \beta + \varepsilon$ | update@update textit |
| $\log(y) \sim x_1 + x_2$ | corresponds to $\log(y_i) = (1 \ x_{i1} \ x_{i2})(\beta_1 \ \beta_2 \ \beta_3)^\top + \varepsilon$ (a constant term is assumed implicitly) | |
| $y \sim A$ | one way analysis of variance with factor A | |
| $y \sim A + x$ | Covariance analysis with factor A and covariable x | |
| $y \sim A * B$ | two factor crossed layout with factors A and B | |
| $y \sim A/B$ | two factor hierarchical layout with factor A and sub-fakcor B | |

For an economic transition between models, for example for model comparison, the function `update()` is available.

| <i>Model Administration</i> | |
|-----------------------------|--|
| <code>formula()</code> | extracts a model formula from an object |
| <code>terms()</code> | extracts terms of the model formula from an object |
| <code>contrasts()</code> | specifies contrasts |
| <code>update()</code> | update and re-fit, or change a model |
| <code>model.matrix()</code> | generate the design matrix for a model |

Example:

`lm(y ~ poly(x, 4), data = experiment)`
analyses the data set “experiment” with a linear model for polynomiale regression of degree 4.

ToDo:
discuss
update

| <i>Standard Analysis</i> | |
|--------------------------|--|
| <code>lm()</code> | linear model See also chapter ch:02 |
| <code>glm()</code> | generalised linear model |
| <code>nls()</code> | non-linear least squares |
| <code>nlm()</code> | general non-linear minimisation |
| <code>update()</code> | update and re-fit, or change a model |
| <code>anova()</code> | analysis of variance |

A.31 Graphic Functions

s:A.11

R provides two graphic systems: The basic graphic system of R implements a model, that is oriented at pen and paper drawing. The lattice graphic system is an additional second graphic system, that is oriented at a camera/object model. For information about lattice see `help(lattice)`, for a survey about the functions in lattice see `library(help = lattice)`. Informationen about the basic graphic system follow here.

Graphic functions fall essentially in three groups:

- “high level” functions. These define a new output.
- “low level” functions. These modify an existing output.
- parametrisations. These modify the settings of the graphic system.

A.31.1 High Level Graphics

| “high level” | |
|------------------------|--|
| <code>plot()</code> | generic graphic output |
| <code>pairs()</code> | pair-wise scatter plots |
| <code>coplot()</code> | scatter plots, conditioned on covariables |
| <code>qqplot()</code> | <i>QQ</i> -Plot |
| <code>qqnorm()</code> | Gaußian <i>QQ</i> -Plot |
| <code>qqline()</code> | adds a line to a Gaußian <i>QQ</i> -Plot, passing through the first and third quantile. |
| <code>hist()</code> | histogram See also section I.3.2, Seite I-29 |
| <code>boxplot()</code> | box&whisker plot |
| <code>dotplot()</code> | |
| <code>curve()</code> | evaluates a function or an expression and draws a curve. <i>Example:</i> <code>curve(dnorm, from = -3, to = 3)</code> |
| <code>image()</code> | colour coded z against x, y |
| <code>contour()</code> | contour plot of z against x, y |
| <code>persp()</code> | 3D surface |

A.31.2 Low Level Graphics

Most high-level functions have a parameter `add`. If the function is called with `add = FALSE`, it can be used to add elements to an existing plot. Moreover there are several low-level functions which suppose that a there is already a defined plot environment.

| <i>"low level"</i> | | annotation legend captions |
|------------------------|---|----------------------------------|
| <code>points()</code> | generic function. Marks points at specified positions.. <i>Syntax:</i> <code>points(x, ...)</code> | |
| <code>lines()</code> | generic function. Joins points at specified positions. <i>Syntax:</i> <code>lines(x, ...)</code> | |
| <code>abline</code> | Adds a line (in several representations) to a plot. <i>Syntax:</i> <code>abline(a, b, ...)</code> | |
| <code>polygon()</code> | adds polygon with specified vertices. | |
| <code>axis()</code> | Adds axis. | |

Besides this, R has rudimentary possibilities for interaction with graphics.

| <i>Interactions</i> | |
|------------------------|---|
| <code>locator()</code> | determines the position of mouse clicks. A current graphics display has to be defined before <code>locator()</code> is used. <i>Example:</i> <code>plot(runif(19))</code> <code>locator(n = 3, type = "l")</code> |

A.31.3 Annotations and Legends

The high-level-function generally offer the possibilities to add standard annotations by using parameters.

```
main = main title, above the plot
sub = plot caption, below the plot
xlab = label for the x axis
ylab = label for the y axis.
```

For documentation, see `help(plot.default)`.

This is complemented by low-level functions.

| <i>"low level"</i> | |
|----------------------|---|
| <code>title()</code> | adds main title, analogous to high-level parameter main. <i>Syntax:</i> <code>title(main = NULL, sub = NULL, xlab = NULL, ylab = NULL, ...)</code> |
| <code>text</code> | adds text at specified coordinates. <i>Syntax:</i> <code>text(x, y = NULL, text, ...)</code> |

(cont.)→

| | | |
|--|-------------------------------|--|
| <code>expression@exp bquote@bquote[</code> | "low level" (cont.) | |
| | legend() | adds a legend block. <i>Syntax:</i> <code>legend(x, y = NULL, text, ...)</code> |
| | mtext() | adds text to margin. <i>Syntax:</i> <code>mtext(text, side = 3, ...)</code> . The margins are denoted by by 1 = bottom, 2 = left, 3 = top, 4 = right) |

R gives (limited) possibilities for mathematical typesetting. If the text parameter is a character string, it is taken directly. If the text parameter is an (unevaluated) R expression, R tries to render the expression as usual in a mathematical formula. R expressions can be generated using the functions `expression()` or `bquote()`.

Example:

```
text(x, y, expression(paste(bquote("(",
atop(n, x), ")"),
.(p)^x, .(q)^{n-x})))
```

`demo(plotmath)` gives several examples for mathematical typesetting in plots.

A.31.4 Graphic Parameters and Layout

| Parametrisations | |
|-------------------------|---|
| par() | sets parameters for the basic ggraphic system <i>Syntax:</i> see <code>help(par)</code> <i>Example:</i> <code>par(mfrow = c(m, n))</code> splits the graphic area in <i>m</i> rows and <i>n</i> columns, to be filled row-wise. <code>par(mfcol = c(m, n))</code> fills the area column by column. |
| split.screen() | splits the graphic area in parts <i>Syntax:</i> <code>split.screen(figs, screen, erase = TRUE)</code> . If <i>figs</i> is a pair of two arguments, these will fix the number of rows and columns.. If <i>figs</i> is a matrix, each row gives the co-ordinates of a graphic area in relative coordinates [0 ... 1]. <code>split.screen()</code> can be nested. |
| screen() | Selects graphic area for the next graphical output. <i>Syntax:</i> <code>screen(n = cur.screen, new = TRUE)</code> . |
| layout() | Divides the graphic area. This function is not compatible with other layout functions. |

A.32 Basic Statistical Functions

s:A.12

| <i>Statistical Functions</i> | |
|------------------------------------|---|
| <i>sum()</i> | sums up components of a vector |
| <i>cumsum()</i> | calculates cumulated sums |
| <i>prod()</i> | multiplies components of a vector |
| <i>cumprod()</i> | calculates cumulated products |
| <i>length()</i> | length of an object, for example a vector |
| <i>max()</i> <i>min()</i> | maximum, minimum. See also <i>pmax</i> , <i>pmin</i> |
| <i>range()</i> | minimum and maximum |
| <i>cummax()</i> <i>cummin()</i> | cumulated maximum, minimum |
| <i>quantile()</i> | sample quantile. for theoretical distributions, use <i>qxxxx</i> , z.B. <i>qnorm</i> |
| <i>median()</i> | median |
| <i>mean()</i> | mean including trimmed mean |
| <i>var()</i> | variance, variance / covariance matrix |
| <i>sort()</i> <i>rev()</i> | sorting |
| <i>order()</i> | returns a permutation for sorting |
| <i>rev()</i> | reverse sorting |
| <i>rank()</i> | ranks in sample |

random_numbers!reproducible
s:A.13

A.33 Distributions, Random Numbers, Densities...

The base generator for uniform random numbers is administered by `Random`. Several types of generators are available as base generator. **for serious simulation it is strongly recommended to read the recommendations of Marsaglia et al.** (See `help(.Random.seed)`). All non-uniform random number generators are derived from the current base generator. A survey of most important non-uniform random number generators, their distribution functions and their quantiles is given at the end of this section.

| R Random Numbers | |
|---------------------------|---|
| <code>.Random.seed</code> | <code>.Random.seed</code> is a global variable that hold the current state of the random number generator. This variable can be stored and later be restored with <code>set.seed()</code> . |
| <code>set.seed()</code> | initialises the random number generator <code>Syntax: set.seed(seed, kind = NULL)</code> |
| <code>RngKind()</code> | <code>RngKind()</code> gives the name of the current base generator. <code>RngKind (<name>)</code> sets a basic random number generator. <code>Syntax: RngKind() RngKind(<name>)</code> <code>Example: RngKind("Wichmann-Hill") RngKind("Marsaglia-Multicarry") RngKind("Super-Duper")</code> |
| <code>sample()</code> | <code>sample()</code> drwas a sample from the values given in vector <code>x</code> angegebenen values, with or without replacement (controlled by the value of <code>replace</code>). Size is by default the length of <code>x</code> . Optionally <code>prob</code> can be a vector of probabilities for the values of <code>x</code> . <code>Syntax: sample(x, size, replace = FALSE, prob)</code> <code>Example: Random permutation: sample(x) val<-c("H", "T") prob<-c(0.3, 0.7) sample(val, 10, replace = T, prob)</code> |

If simulations shall be reproducible, the random number generator must be set to a well-defined state. An example is the following statement sequence: `save.seed <- .Random.seed`
`save.kind <- RNGkind()`

with `set.seed(save.seed, save.kind)` the state of the random number generator is restored if necessary.

The individual function names for the common not-uniform generators and distribution functions is combined from a prefix and the short name of the distribution. General pattern: `xxxx` is the short name

- `rxxxx` generates random numbers
- `dxxxx` density or probability
- `pxxxx` distribution function
- `qxxxx` quantiles

example:

- `x<-runif(100)` generates 100 random variable with U(0, 1) distribution
- `qf(0.95, 10, 2)` calculates the 95% quantile of the F(10, 2) distribution.

| <i>Distributions</i> | <i>Short Name</i> | <i>Parameter and Default-values</i> |
|-------------------------|-----------------------|--------------------------------------|
| Beta | <code>beta</code> | <code>shape1, shape2, ncp = 0</code> |
| Binomial | <code>binom</code> | <code>size, prob</code> |
| Cauchy | <code>cauchy</code> | <code>location = 0, scale = 1</code> |
| χ^2 | <code>chisq</code> | <code>df, ncp = 0</code> |
| Exponential | <code>exp</code> | <code>rate = 1</code> |
| F | <code>f</code> | <code>df1, df2 (ncp = 0)</code> |
| Gamma | <code>gamma</code> | <code>shape, scale = 1</code> |
| Gauß | <code>norm</code> | <code>mean = 0, sd = 1</code> |
| Geometric | <code>geom</code> | <code>prob</code> |
| Hypergeometric | <code>hyper</code> | <code>m, n, k</code> |
| Lognormal | <code>lnorm</code> | <code>meanlog = 0, sdlog = 1</code> |
| Logistisch | <code>logis</code> | <code>location = 0, scale = 1</code> |
| Negativ-Binomial | <code>nbinom</code> | <code>size, prob</code> |
| Poisson | <code>pois</code> | <code>lambda</code> |
| Student's t | <code>t</code> | <code>df</code> |
| Tukey Studentised Range | <code>tukey</code> | |
| Uniform | <code>unif</code> | <code>min = 0, max = 1</code> |
| Wilcoxon Signed Rank | <code>signrank</code> | <code>n</code> |
| Wilcoxon Rank Sum | <code>wilcox</code> | <code>m, n</code> |
| Weibull | <code>weibull</code> | <code>shape, scale = 1</code> |

A.34 Programming on the Language

The language elements of R are objects, as are data or functions. They can be read and changed like these.

| <i>Conversions</i> | |
|---------------------------|---|
| <code>parse()</code> | Converts input into a list of R expressions. <code>parse</code> executes the parse, but does not evaluate the expression. |
| <code>deparse()</code> | Converts an R expression given in internal representation into a character string. |
| <code>expression()</code> | generates an R expression in internal representations. <i>Example:</i> <code>integrate <- expression(integral(fun, lims))</code> <i>See also</i> I.3.1: mathematical typesetting in plot annotations |
| <code>substitute()</code> | R expression with evaluation of all defined terms. |
| <code>bquote()</code> | R expression with selective evaluation. Terms in <code>.()</code> are evaluated. <i>Examples:</i> <code>n<-10; bquote(n^2 == .(n*n))</code> |

| <i>evaluation</i> | |
|---------------------|--------------------------|
| <code>eval()</code> | evaluates an expression. |

```
$Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/Rnw/S0A1.Rnw.tex,v $
$Revision: 1.18 $
$Date: 2008/03/18 09:01:48 $
$name: $
$Author: j40 $
```

References

- [1] Richard A. Becker, John M. Chambers, and Allan R. Wilks. *The New S Language*. Chapman & Hall, London, 1988.
- [2] John M. Chambers. *Programming with Data*. Springer, New York, 1998. ISBN 0-387-98503-4.
- [3] John M. Chambers and Trevor J. Hastie. *Statistical Models in S*. Chapman & Hall, London, 1992.
- [4] William S. Cleveland. *Visualizing Data*. AT&T Bell Laboratories, Murray Hill, 1993.
- [5] George W. Furnas and Andreas Buja. Prosection views: dimensional inference through sections and projections. *J. Comput. Graph. Statist.*, 3(4):323–385, 1994.
- [6] Peter Gänßler and Winfried Stute. *Wahrscheinlichkeitstheorie*. Springer, 1977.
- [7] Robert Gentleman and Ross Ihaka. Lexical scope and statistical computing. *Journal of Computational and Graphical Statistics*, 9:491–508, 2000.
- [8] Gabor Grothendieck and Thomas Petzoldt. R help desk: Date and time classes in R. *R News*, 4(1):29–32, June 2004.
- [9] Alfred Inselberg, Tuval Chomut, and Mordechai Reif. Convexity algorithms in parallel coordinates. *J. Assoc. Comput. Mach.*, 34(4):765–801, 1987.
- [10] N.L. Johnson and S. Kotz. *Discrete Distributions*. Wiley, New York, 1970.
- [11] Bent Jørgensen. *The Theory of Linear Models*. Chapman & Hall, New York-London, 1993.
- [12] R. G. Miller. *Simultaneous Statistical Inference*. Springer, New York, 1981.
- [13] Paul Murrell. *R Graphics*. Chapman & Hall/CRC, Boca Raton, Fla. [u.a.], 2006.
- [14] R Development Core Team. An introduction to R. Technical report, R Project, 2007.
- [15] R Development Core Team. R data import/export. Technical report, R Project, 2007.
- [16] R Development Core Team. The R language definition. Technical report, R Project, 2007.
- [17] R Development Core Team. The R reference index. Technical report, R Project, 2007.
- [18] R Development Core Team. Writing R extensions. Technical report, R Project, 2008.
- [19] C. Radhakrishna Rao. *Linear Statistical Inference and Its Applications*. Wiley, 2 edition, 1973.
- [20] G. M. Reaven and R. G. Miller. An attempt to define the nature of chemical diabetes using a multidimensional analysis. *Diabetologia*, 16:17–24, 1979.
- [21] Günther Sawitzki. Numerical reliability of data analysis systems. *Computational Statistics & Data Analysis*, 18(2):269–286, 1994.
- [22] Günther Sawitzki. Report on the numerical reliability of data analysis systems. *Computational Statistics & Data Analysis*, 18(2):289 – 301, 1994.
- [23] Günther Sawitzki. Quality control and early diagnostics for cDNA microarrays. *R News*, 2(1):6–10, March 2002.
- [24] William N. Venables and Brian D. Ripley. *S Programming*. Springer, 2000. ISBN 0-387-98966-8.
- [25] William N. Venables and Brian D. Ripley. *Modern Applied Statistics with S*. Springer, Heidelberg, 4 edition, 2002.



Functions by Topic

Topic **aplot**

coplot, 4-17

Topic **debugging**

browser, A-68

debug, A-68

recover, A-68

traceback, A-68

Topic **distribution**

qqnorm, 3-7

Uniform, 1-4, C-1

Topic **hplot**

coplot, 4-17

pairs, 4-7

qqnorm, 3-7

Topic **htest**

t.test, 3-12

wilcox.test, 3-15

Topic **loess**

loess, 2-36

Topic **models**

anova, 2-21

Topic **regression**

anova, 2-21

lm, 2-11

Topic **smooth**

loess, 2-36

Function and Variable Index

.Random.seed, 1-5, C-2
[, B-2

aa1 itfun (*alias1*), B-1
abbreviate, A-63
abline, 1-13
add1, 2-22
aggregate, A-62
anova, 2-13, 2-21, 2-30, A-79
anova.lm, 2-14
aov, 2-11, 2-12, 2-14, 2-22
aperm, A-64
apply, 1-25, A-62
apropos, A-51, A-52
args, A-51
array, A-59, A-61
as.data.frame, 2-11, 2-37
attach, A-60
attr, 2-41
attributes, A-57
axis, A-81

barchart, 4-4
barplot, 4-4
boxcox, 2-35
boxplot, 1-37, 4-4, A-80
bquote, 1-26, A-82, A-86
browser, A-68
bwplot, 4-4
by, A-62

c, 1-9
casefold, A-63
cbind, A-59, A-62
chartr, A-63
chisq.test, 1-30
citation, 1-53
class, 2-12, 2-42, A-57
cloud, 4-4, 4-12
co.intervals (*coplot*), 4-17

coef, 2-14, 2-43
coefficients, 2-22
confint, 2-14, 2-43
contour, 4-1, 4-2, A-80
contourplot, 4-4
contrasts, A-79
coplot, 4-17, 4-17, A-80
cummax, A-83
cummin, A-83
cumprod, A-83
cumsum, A-83
curve, 1-23, A-80

data, 1-53, 2-40, A-58, A-74, A-76
data.entry, A-74
data.frame, A-60
data.matrix, 4-8
debug, A-68
demo, A-51
density, 1-16
densityplot, 4-4
deparse, A-86
detach, A-60, A-76
determinant, A-77
dim, A-59, A-61
dimnames, A-61
dir, A-58
dotchartt, 4-4
dotplot, 4-4, A-80
drop1, 2-22
dump, A-73
dunif (*Uniform*), 1-4, C-1
duplicated, A-63

edit, A-74
effects, 2-13, 2-14, 2-22, 2-43
eigen, A-77
environment, 1-51, A-52, A-67
eval, 1-51, A-86
example, A-51

expand.grid, *A-63*
expression, *1-14, A-82, A-86*

factor, *2-5, 4-19, A-59*
find, *A-52*
fitted, *2-14, 2-43*
fitted.values, *2-22*
formula, *2-12, A-79*
fun1, *B-1*
function, *4-18*

getwd, *A-58*
glht, *2-30*
glm, *2-14, A-79*

help, *A-51, A-58*
help.search, *A-51*
hist, *1-16, 1-19, 4-4, A-80*
histogram, *4-4*

identify, *4-38*
image, *4-1, 4-2, 4-4, 4-47, A-80*
influence, *2-43*
inherits, *2-42*
install.packages, *1-52, A-76*
integrate, *4-33*

kruskal.test, *3-18*
ks.test, *1-29*

lapply, *A-62*
lattice.options, *A-58*
lda, *4-26*
leaps, *4-42*
legend, *1-46, A-82*
length, *1-13, A-52, A-57, A-83*
levels, *A-59*
library, *A-58, A-76*
lines, *A-81*
list, *A-60*
lm, *2-5, 2-11, 2-22, 2-42, 2-43, 4-39, 4-46, A-79*
lm.fit, *2-12, 2-14*
lm.influence, *2-14*
lm.wfit, *2-14*
load, *2-39, 2-40, A-74*
locator, *A-81*
loess, **2-36**

loess.control, *2-37, 2-38*
lowess, *2-38*
ls, *1-51, A-52, A-58, A-72*
ls.str, *A-52*

match, *A-63*
matrix, *4-19, A-61*
max, *A-83*
mean, *1-35, A-83*
median, *A-83*
merge, *A-63*
methods, *2-42, A-58*
min, *A-83*
missing, *A-66*
mode, *2-41, A-52, A-54, A-57*
model.frame, *3-13, 3-16*
model.matrix, *2-12, 2-17, 2-43, A-79*
model.matrix.default, *2-12*
model.offset, *2-12*
mtext, *1-48, 4-19, A-82*
mvr, *4-46*

NA, *3-8*
na.exclude, *2-12*
na.fail, *2-12*
na.omit, *2-12*
names, *A-57*
ncol, *A-61*
nlm, *A-79*
nls, *A-79*
nrow, *A-61*

objects, *A-52, A-72*
offset, *2-12*
options, *2-12, A-58*
order, *A-83*
ordered, *A-59*
outer, *1-25, A-62*

package.manager, *A-76*
package.skeleton, *1-53, 1-54, A-76*
pairs, *4-4, 4-7, 4-7, 4-11, 4-19, 4-36, A-80*
panel.smooth, *4-19*
par, *4-19, A-58*
parallel, *4-4*
parse, *1-51, A-86*
persp, *4-1, 4-2, 4-4, A-80*
plot, *1-13, 1-14, 2-31, 2-42, 4-4, 4-24, 4-38, A-56, A-80*

pmatch, *A-63*
 points, *4-19, A-81*
 polygon, *A-81*
 power.prop.test, *3-25*
 power.t.test, *3-22*
 ppoints, *3-8*
 prcomp, *4-29, 4-43*
 predict, *2-14, 2-26, 2-43*
 predict.lm, *2-13, 2-14, 2-26, 4-46*
 predict.loess, *2-38*
 print, *1-51, 1-52, 2-42, 4-2, 4-12, A-56*
 print.anova (*anova*), *2-21*
 print.lm (*lm*), *2-11*
 prod, *A-83*
 prop.test, *3-14, 3-24*
 psignrank, *3-18*
 punif (*Uniform*), *1-4, C-1*
 pwilcox, *3-18*

 q, *1-3*
 qq, *4-4*
 qqline (*qqnorm*), *3-7, A-80*
 qqmath, *4-4*
 qqnorm, *3-6, 3-7, 4-4, A-80*
 qqplot, *3-6, 4-4, A-80*
 qqplot (*qqnorm*), *3-7*
 qr, *A-77*
 quantile, *1-36, A-83*
 qunif (*Uniform*), *1-4, C-1*

 range, *4-19, A-83*
 rank, *A-83*
 rbind, *A-59, A-62*
 read.fwf, *A-75*
 read.table, *A-75*
 recover, *A-68*
 regsubsets, *4-41*
 rep, *1-9*
 replicate, *A-62*
 require, *A-76*
 reshape, *A-63*
 residuals, *2-14, 2-22, 2-43*
 rev, *A-83*
 rm, *A-72*
 RngKind, *A-84*
 rnorm, *1-5, C-2*
 Rprof, *A-69*
 Rprofmem, *A-69*
 rug, *1-16*

 runif, *1-4*
 runif (*Uniform*), *1-4, C-1*

 sample, *A-84*
 sapply, *A-62*
 save, *1-53, 2-39, 2-40, A-74*
 scan, *A-75*
 screen, *A-82*
 sd, *1-35*
 search, *1-51, A-51, A-58*
 searchpaths, *A-52*
 seq, *1-9, A-63*
 set.seed, *A-84*
 setwd, *A-58*
 sink, *A-73*
 solve, *A-77*
 some fun, *B-1*
 sort, *1-13, A-83*
 source, *1-52, 1-53, A-73*
 split, *A-62, A-64*
 split.screen, *A-82*
 splom, *4-4*
 stack, *2-40*
 stdres, *2-43*
 storage.mode, *2-41, A-57*
 str, *A-57*
 stripchart, *4-4*
 stripplot, *4-4, 4-25*
 structure, *A-56, A-57*
 studres, *2-43*
 substitute, *A-86*
 substring, *A-63*
 sum, *A-83*
 summary, *1-36, 2-22, A-56*
 summary.lm, *2-14*
 summaryRprof, *A-69*
 svd, *A-77*
 Sweave, *1-52*
 sys.parent, *A-52*
 system, *A-58*
 system.time, *A-69*

 t, *A-64*
 t.test, *3-12, 3-18*
 table, *1-19, A-62, A-63*
 tapply, *A-59, A-62*
 terms, *2-13, A-79*
 title, *4-19, A-81*
 tolower, *A-63*

`toupper`, *A-63*
`trace`, *A-68*
`traceback`, *A-69*
`trellis.par.set`, *A-58*
`try`, *A-69*
`ts.intersect`, *2-13*
`typedef`, *A-54*
`typeof`, *2-41, A-52, A-57*

`unclass`, *2-42*
`undebug`, *A-68*
`Uniform`, **1-4, C-1**
`unique`, *A-63*
`unsplit`, *A-64*
`untrace`, *A-69*
`update`, *A-79*
`update.packages`, *1-52*
`UseMethod`, *2-42*

`var`, *1-35, A-83*
`vcov`, *2-14, 2-43*

`wilcox.exact`, *3-18*
`wilcox.test`, **3-15, 3-15**
`wilcox_test`, *3-15*
`wireframe`, *4-4*
`write`, *A-73*
`write.table`, *A-75*

`xyplot`, *4-4*

Index

PP plot, **1-40**

QQ plot, **1-40**

ToCheck

 A.34: micronuclei still available?, 2-23

ToDo

 A.34: 2/3, 4-31

 A.34: Add reference to lattice, 1-47

 A.34: Check revise Tukey-Anscombe, 2-18

 A.34: Comment on Tukey vs Schaffee, 2-28

 A.34: Comment on trellis, 4-22

 A.34: Does memory help to increase the bound?, 4-14

 A.34: Qualitative features, 3-28

 A.34: Ref to curse of dimension, 4-23

 A.34: Remark on small setosa, 4-26

 A.34: Scheffe: fix lower/upper plotting bound, 2-28

 A.34: Simulation, Power, 3-20

 A.34: Variance bias dilemma, 4-23

 A.34: add Melbourne data, 4-33

 A.34: add color, 4-26

 A.34: add label for examples, 1-27

 A.34: add legend, control colour, 4-16

 A.34: add mag, 4-23

 A.34: add mds, 4-1

 A.34: add merge data frames Harrell 4.2.5ff, A-63

 A.34: add more regression graphics, 2-1

 A.34: add power function, 3-15

 A.34: add power simulation - see Rnw.tex source, 3-15

 A.34: add references, 1-39

 A.34: adjust gray formats in tables, 3-26

 A.34: adjust scale, orientation, 4-31

 A.34: apply and related, 1-24

 A.34: avoid loops, 1-44

 A.34: better irexercise, 1-43

A.34: cDNA: details, 4-47

A.34: ch01: add examples f. outer, 1-25

A.34: ch02: Add example, 2-39

A.34: ch03: irexercise: power, 3-10

A.34: ch03: standardise scale, 3-10

A.34: ch03: work out, 3-10

A.34: check contrasts, 2-22

A.34: check iris inverse ratio, 4-27

A.34: check pearson, 1-30

A.34: check plot, 2-27

A.34: clarify: information <-> assumptions, 3-15

A.34: clean up discussion of factors, 2-22

A.34: clean up usae of libraries and packages, A-76

A.34: color R version, 4-14

A.34: comment ch.4 experimental, 12

A.34: comment on literature, 12

A.34: compare fit, error term, 2-2

A.34: conditional, function parameters, 2-41

A.34: consider simple exercises for random numbers – Antony, 1-8

A.34: copy function syntax, 1-48

A.34: define studentised residuals , 2-18

A.34: discuss alpha channel, 4-3

A.34: discuss choice of models - model dependent estimation, 2-22

A.34: discuss difference to setting to ch 01, 2-2

A.34: discuss missing values - IntroR 2.5, 1-1

A.34: discuss scale dependency, 4-44

A.34: discuss update, A-79

A.34: example in appendix, 1-50

A.34: examples of help files included, 11

A.34: extend reshape data frames Harrell 4.2.7, A-63

A.34: formal variance bias, 4-28

A.34: formats: add frame. This may be long. longtable environment?, B-8

A.34: formats: avoid page break, B-6
A.34: formats: show text in tables ragged right. Add hyphenation, B-6
A.34: histogram as density estimator; choice of cells, 1-32
A.34: improve Scheffe, 2-26
A.34: introduce noncentral t in ch. 2, 3-20
A.34: mention unbiased, 2-7
A.34: more precise irexercise, 1-50
A.34: move fit, residuals here, 2-2
A.34: move to cbind etc, A-64
A.34: one sided or two sided?, 3-21
A.34: or change plot co-ordinates?, 4-20
A.34: package or library as basic term?, 1-52
A.34: reduce body fat case study drastically, 4-35
A.34: reduce preliminary information on fat, 4-35
A.34: ref appendix: reshape functions, 2-23
A.34: reference to kernel density estimators, 1-32
A.34: remove line caps - bar line is too long, 1-21
A.34: rows, columns, lanes?, 2-34
A.34: search path, 1-52
A.34: std error, t-test, point-wise confidence, 2-18
A.34: studentised residuals, pointwise confidence intervals, 2-18
A.34: superassignment introR 10.5, A-66
A.34: supply better support for legend, colour key, 4-11
A.34: syntax: stack, unstack, split, cut, 2-40
A.34: this is first theorem. make numeration more logical, 1-21
A.34: use attach, 4-27
A.34: use principal components in modelling. pc regression?, 4-44
A.34: work out as example, 1-26
A.34: zum einen..zum anderen, 2-18

`aa2 itd (alias2)`, B-1
added variable plots, **4-29**
analysis of variance, **2-19**
annotation, A-81

bandwidth, **1-11**
bootstrap, **3-10**
Box-Cox transformation, **2-35**
brushing, **4-5**

captions, A-81
coefficient of variation, **3-25**
conditioned, **4-17**
contrast, **2-22**, **2-28**
curse of dimension, **4-35**

data structures, 1-18, A-59
DateTimeClasses, A-75
Datum
 see DateTimeClasses, A-75
Debugging, 1-50, A-68
design matrix, **2-2**, **2-15**
distribution
 marginal, **4-7**

entry, **B-9**
exact test, 3-15
expected value, **1-34**

factor, **2-4**
 levels, 2-5
fit, **2-6**
function, **1-47**, A-66–A-70
 polymorphic, see polymorphic

Gauß-Markov estimator, **2-5**

hat matrix, **2-7**

kernel, **1-11**

least squares estimator, **2-5**
legend, A-81
linking, **4-5**
`locfit`, **4-6**
`loglin`, **1-30**

MASS, **2-43**
model
 linear, 2-2
 simple linear, **2-8**

- model function, **2-2**
- mva**, *4-29*
- one sample case, **3-9**
- parameter
 - default, *1-3*
- plot, *1-6*
- plotmath, *1-14*
- polymorph, *1-52*, *2-42*
- polymorphic, *1-3*, *A-56*
- Power, **1-32**
- principal component analysis, **4-29**
- probability plot, **1-40**
- Profiling, *A-68*
- projection pursuit, **4-13**
- quantile plot, **1-40**
- random numbers, *1-4*
 - pseudo, *1-8*
 - reproducible, *A-84*
- regression
 - linear, *2-2*
- regressor, **2-2**
- residual, **2-8**
- response, **2-2**
- samples
 - repeated, **1-32**
- scale
 - categorical, *2-5*
 - ordinal, *2-5*
- scale shift family, **3-5**
- scatter plot matrix, **4-7**
- serial plot, **1-6**
- shift family, **3-4**
- standard deviation, **1-35**
- stochastically smaller, **3-4**
- substitute, **B-8**
- Test
 - χ^2 , *1-30*
 - Kolmogorov-Smirnov, *1-29*
 - Median-, *1-29*
 - Monte Carlo, **1-25**
 - Wilcoxon, *3-15*
- test

CHAPTER B

S formats. Remove for public version

```
aa1 it-
fun@aa1
itfun
(alias1)
aa2 itd@aa2
itd
(alias2)
some_ fun
fun1
```

```
expl:fo1
expl:fo2
```

Example B.1 Nur *expl*

Example B.1 Nach dem Schunk

Vor dem Schunk

Schunk:fo1

expl:fo1 **expl:fo1** **expl:fo2** B.I. Schunk:fo1 **Schunk:fo1** B.I.

B.1 Test R function index

```
aa1 itfun()
aa2 itd
```

T_EX Example Start: exa2

T_EX Example Input: exa2

\ixr{fun1}

T_EX Example Output: exa2

exa2

T_EX Example End: exa2

T_EX Example Start: exa3

T_EX Example Input: exa3

B-2

\ixr[fun2]{fun3}

TEX Example Output: exa3

exa3 fun2]fun3 _____
TeX Example End: exa3

B.2 R styles

Sinput

TEX Example Start: exa4

TEX Example Input: exa4

Before private note
\begin{irprivnote}
A private note
\end{irprivnote}
After private note

TEX Example Output: exa4

Before private note

Private note:
A private note

exa4

After private note _____
TeX Example End: exa4

TEX Example Start: exa5

TEX Example Input: exa5

\begin{Sinput}
>plot(x) # "1+1"
\end{Sinput}

TEX Example Output: exa5

exa5

Input

TEX Example End: exa5

Soutput

TEX Example Start: exa6

TEX Example Input: exa6

```
\begin{Soutput}
abc <- "1+1"
\end{Soutput}
```

TEX Example Output: exa6

exa6

Output

TEX Example End: exa6

Sinput and Soutput in an Schunk

TEX Example Start: exa7

TEX Example Input: exa7

```
\begin{Schunk}
\begin{Sinput}
>plot(x) # "1+1"
\end{Sinput}
\begin{Soutput}
abc <- "1+1"
\end{Soutput}
\end{Schunk}
```

B-4

T_EX Example Output: exa7

plot(x) # "1+1"

Input

exa7

T_EX Example End: exa7

irin

T_EX Example Start: exa8

T_EX Example Input: exa8

```
\begin{irin}
abc$d <- "1+1" #R input code block
\end{irin}
```

T_EX Example Output: exa8

exa8

abc\$d <- "1+1" #R input code block

T_EX Example End: exa8

irout

T_EX Example Start: exa9

T_EX Example Input: exa9

```
\begin{irout}
abc <- "1+1" #R output code block
\end{irout}
```

T_EX Example Output: exa9

exa9 abc <- "1+1" #R output code block

T_EX Example End: exa9

Scode

T_EX Example Start: exa10

T_EX Example Input: exa10

\begin{Scode}
abc <- "1+1" # R output code block in Scode environment
\end{Scode}

T_EX Example Output: exa10

exa10 abc <- "1+1" # R output code block in Scode environment

T_EX Example End: exa10

T_EX Example Start: exa11

T_EX Example Input: exa11

A sample \ircode{abc <- "1+1"} inline code

T_EX Example Output: exa11

exa11 A sample abc <- "1+1" inline code

T_EX Example End: exa11

B.3 Tables

ToDo:

formats:
show text
in tables
ragged right. Add hyphenation
ToDo: formats: avoid page break

Test:

TeX Example Start: exa12

TeX Example Input: exa12

```
\begin{irexercise}{1}
&abc\\
de&fg\\
\hline
\end{irexercise}
```

TeX Example Output: exa12

1

| Exercise B.1 | |
|--------------|-----------|
| de | abc fg |

exa12

TeX Example End: exa12

TeX Example Start: exa13

TeX Example Input: exa13

```
\begin{tabthree}{Verteilungen}{Kurzname}{Defaults}
Beta & beta & \rcode{shape1, , shape2, , ncp=0}\\\hline
Binomial & binom & \rcode{size, prob} \\hline
Cauchy & cauchy & \rcode{location = 0, scale = 1}\\\hline
$\chi^2$ & chisq & \rcode{df, ncp=0} \\hline
Exponential & exp& \rcode{rate = 1} \\hline
$F$ & $f$ & \rcode{df1, df2, ncp = 0} \\hline
Gamma & gamma & \rcode{shape, scale = 1}\\\hline
Gauß & norm & \rcode{mean = 0, sd = 1}\\\hline
Geometrisch & geom & \rcode{prob} \\hline
Hypergeometrisch & hyper & \rcode{m, n, k} \\hline
Lognormal & lnorm & \rcode{meanlog = 0, sdlog = 1} \\hline
```

```

Logistisch & logis & \ircode{location = 0, scale = 1} \\ \hline
Negativ-Binomial & nbinom & \ircode{size, prob} \\ \hline
Poisson & pois & \ircode{lambda} \\ \hline
Student's t & t & \ircode{df} \\ \hline
Tukey Studentised Range & tukey & \\ \hline
Uniform & unif & \ircode{min = 0, max = 1} \\ \hline
Wilcoxon Signed Rank & signrank & \ircode{n} \\ \hline
Wilcoxon Rank Sum & wilcox & \ircode{m, n} \\ \hline
Weibull & weibull & \ircode{shape, scale = 1}
\\
\end{tabthree}

```

TeX Example Output: exa13

| <i>Verteilungen</i> | <i>Kurzname</i> | <i>Defaults</i> |
|-------------------------|-----------------|--------------------------------|
| Beta | beta | <i>shape1, shape2, ncp=0</i> |
| Binomial | binom | <i>size, prob</i> |
| Cauchy | cauchy | <i>location = 0, scale = 1</i> |
| χ^2 | chisq | <i>df, ncp=0</i> |
| Exponential | exp | <i>rate = 1</i> |
| F | <i>f</i> | <i>df1, df2, ncp = 0</i> |
| Gamma | gamma | <i>shape, scale = 1</i> |
| Gauß | norm | <i>mean = 0, sd = 1</i> |
| Geometrisch | geom | <i>prob</i> |
| Hypergeometrisch | hyper | <i>m, n, k</i> |
| Lognormal | lnorm | <i>meanlog = 0, sdlog = 1</i> |
| Logistisch | logis | <i>location = 0, scale = 1</i> |
| Negativ-Binomial | nbinom | <i>size, prob</i> |
| Poisson | pois | <i>lambda</i> |
| Student's t | t | <i>df</i> |
| Tukey Studentised Range | tukey | |
| Uniform | unif | <i>min = 0, max = 1</i> |
| Wilcoxon Signed Rank | signrank | <i>n</i> |
| Wilcoxon Rank Sum | wilcox | <i>m, n</i> |
| Weibull | weibull | <i>shape, scale = 1</i> |

exa13

TeX Example End: exa13

substitute|textbf

T_EX Example Start: exa14

T_EX Example Input: exa14

```
%\noindent\begin{longtable}{|p{.98\linewidth}|}
\begin{Help}
\begin{Helpinput}
help(lm)
\end{Helpinput}
\todo{formats: add frame. This may be long. longtable environment?}
\begin{Helpoutput}{help(lm)}{lm} \hfill package:base \hfill
... see help test file
\end{Helpoutput}
\end{Help}
%\end{longtable}
```

R Documentation\\Fitting L

T_EX Example Output: exa14

ToDo:
formats:
add frame.
This may
be **exa14**.
longtable
environment?

help(lm) help(lm)lm
Fitting Linear Models

package:base

R Documentation

... see help test file

T_EX Example End: exa14

T_EX Example Start: exa15

T_EX Example Input: exa15

An index \itd[{\substitute}]{entry} definition with substitute.\\

T_EX Example Output: exa15

An index *entry* definition with substitute.

exa15

T_EX Example End: exa15

entry|textbf
used

T_EX Example Start: exa16

T_EX Example Input: exa16

An index \itd{entry} definition without substitute.\\"

T_EX Example Output: exa16

An index *entry* definition without substitute.

exa16

T_EX Example End: exa16

T_EX Example Start: exa17

T_EX Example Input: exa17

A \itx{used} index entry\\"

T_EX Example Output: exa17

A *used* index entry

exa17

T_EX Example End: exa17



CHAPTER C

S help — remove for public version

C.1 Ch 01

help(runif)

Uniform

The Uniform Distribution

Description

These functions provide information about the uniform distribution on the interval from **min** to **max**. **dunif** gives the density, **punif** gives the distribution function **qunif** gives the quantile function and **runif** generates random deviates.

Usage

```
dunif(x, min=0, max=1, log = FALSE)
punif(q, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
qunif(p, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
runif(n, min=0, max=1)
```

Arguments

| | |
|-------------------|---|
| x,q | vector of quantiles. |
| p | vector of probabilities. |
| n | number of observations. If length(n) > 1 , the length is taken to be the number required. |
| min,max | lower and upper limits of the distribution. |
| log, log.p | logical; if TRUE, probabilities p are given as log(p). |
| lower.tail | logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$. |

Uniform@Uniform|textbf
dunif@dunif
(Uni-
form)
punif@punif
(Uni-
form)
qunif@qunif
(Uni-
form)
runif@runif
(Uni-
form)
Topic
dis-
tri-
bu-
tion!Uniform@Uniform

```
.Random.seed@.Random.seed|textit
rnorm@rnorm|textit
```

If `min` or `max` are not specified they assume the default values of `0` and `1` respectively.
 The uniform distribution has density

$$f(x) = \frac{1}{max - min}$$

for $min \leq x \leq max$.

For the case of $u := min == max$, the limit case of $X \equiv u$ is assumed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`.Random.seed` about random number generation, `rnorm`, etc for other distributions.

Examples

```
u <- runif(20)

## The following relations always hold :
punif(u) == u
dunif(u) == 1

var(runif(10000))#- ~ = 1/12 = .08333
```

CHAPTER D

To Do

D.1 For release

Fix colour error at help page(?)
Fix runoff text in too long input lines: ex:1.3
Check/redo image sizes
Fix " -> β in help(loess) family

D.2 misc

discuss name extension
in Sframe: remove vertical space after Beispiel.
summar f. typeof
add margin.table, addmargins Verzani p. 72-73
add stripchart Verzani p. 79 add jitter Verzani p.90
Check ' ToDo' entries in index!!!
Fix refs at exercises
Fine tune graphics size
Add bibliography by chapter
Center email/URL on title page
Grid/Lattice Check lattice vs par usage
Classes?
Check libraries/packages
Fix Bibliography to german header & special words
Check missing fonts: grep 'LaTeX Font Info' *.log

```
$Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/chapters/Rintro_main.tex,v $
$Revision: 1.9 $
$Date: 2008/02/12 10:05:02 $
$Name:  $
$Author: j40 $

$Source: /u/math/j40/cvsroot/lectures/src/SIntro/Rintro/Rintro_temp.tex,v $
$Revision: 1.3 $
$Date: 2008/03/18 09:01:42 $
$Name:  $
$Author: j40 $
```