Computational Statistics

An Introduction to R

Günther Sawitzki

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Computational Statistics

An Introduction to R

Computational Statistics

An Introduction to R

Günther Sawitzki

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

Chapman & Hall/CRC
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

International Standard Book Number-13: 978-1-4200-8678-2 (Hardcover)

Library of Congress Cataloging-in-Publication Data

Sawitzki, Günther.
Computational statistics : an introduction to R / Günther Sawitzki.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-4200-8678-2 (hardcover : alk. paper)
1. R (Computer program language) 2. Mathematical statistics--Data processing. I. Title.

QA276.45.R3S29 2009
519.50285--dc22 2008042298

Introduction

This introduction to R is intended as course material to be used in a concise course or for self-instruction. The course is for students with basic knowledge of stochastics. Notions like distribution function, quantile, expected value and variance are presupposed, as well as some familiarity with statistical features corresponding to these notions. Only a condensed summary is included here. Classical distributions, such as binomial, uniform, and Gaussian should be familiar, together with derived distributions and their asymptotic behaviour. An in-depth knowledge of statistics is not required. However, while this course does cover much of statistics, it is not a substitute for a general statistics course. This course concentrates on the "computing" aspects. Statistical concepts and statistical points of view are introduced and discussed. 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.

What Is R?

R is a programming language, and the name of a software system that implements this language [31]. 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 referred to according to the colour of their cover as white ([5]), blue ([2]) and green book ([4]).

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[1], or "Gnu S" `<http://www.r-project.org/>`.

[1] R got its name by accident — the same accident that made the first names of the original authors of R (Ross Ihaka and Robert Gentleman) start with R.

In the meantime, R has become the reference implementation. Essential more precise definitions and — if necessary — even modifications of the language are given 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. In addition to the original elements of S, R has several extensions. Some of these have been introduced in response to recent developments in statistics, some are introduced to open experimental facilities. Advancements in the S language are taken into account.

The most 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 graphics system in R implements a model 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 viewport/object model. Graphical objects in various positions and orientations are mapped in a visual space.
- *Packages:* The original R had a linear command history and a uniform workspace. R 2.x introduced an improved support for "packages" that may have encapsulated information. To support this, language concepts such as "name spaces" and various tools have been introduced.
- *Internationalisation:* 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 localised 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. This more experimental project 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 *`iplots`*, 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 that is targeted at biocomputing.

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 winding path is chosen here instead: case studies and examples, followed by systematic surveys. Practical work should make use of the rich online material that comes with R. Starting points are the "frequently asked questions" (FAQ) `<http://www.cran.r-project.org/faqs.html>`. "An Introduction to R" ([30]) is the "official" introduction. This documentation and other manuals can be downloaded from `<http://www.cran.r-project.org/manuals.html>`.

R comes with an extensive online help system providing function descriptions and examples. Once you have become familiar with R, the online help and manuals will become your first source of information. For off-line reference, we include some examples of information provided by the help system, with kind permission of The R Foundation for Statistical Computing.

Many R functions are included in the basic system. Other functions must be loaded from libraries. Some of these libraries are distributed with the R system. If additional packages are needed, `<http://www.cran.r-project.org/>` is a first source for downloads, and `<http://r-forge.r-project.org/>` for current projects.

The commercial S-Plus is available 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 that need attention. These programming details of S-Plus are not discussed here. Information about S-Plus can be found at `<http://www.insightful.com/>`.

Contents 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. This course can only open the door to R.

Course participants can come from various backgrounds, with different prerequisites. For pupils and younger students, a mere programming course on technical basics may be appropriate. Later, questions about meaningful classification and background will be more important. This is the aim of the present course. The "technical" material provides a skeleton. Beyond this, we try to open the view for statistical questions and to stimulate interest in the background. This course should whet the appetite for the substance that may be offered in a subsequent well-founded statistical course.

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

The appendix provides a collection of R language elements and functions. During the course, it can serve as a quick reference and perhaps as a starting point and orientation to access the rich information material that comes bundled with R. After the course, it may serve as a note pad. Finally, in the long run for practical work, the online help and online manuals for R are the prime sources of information. This appendix is not meant to be comprehensive. If a concise syntax description or example could be given, it is included. In other cases, the online help information should be consulted.

Using a selection of the exercises, the course can be completed within about four days of work. Conceptually, it is an introduction to statistics with the following 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). The course can then be covered in one week, provided follow-up facilities are established to answer questions that have come up, and possibilities are available to follow the interest in the statistical background that may have resulted.

At a more leisurely pace, Chapter 1 with its exercises can be used on its own. This should provide a working base to use R, and more material from the subsequent chapters can be added later as needed. The first chapter is fairly selfcontained, including the necessary basic definitions of statistical terms. The other chapters assume that the reader can look up terms if necessary.

Using the course during a term in a weekly class requires 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 covering the statistical background is recommended, running in parallel with this one.

For a subsequent self-paced study that goes into detail on R as a programming language, the recommended reading is ([51]).

Statistical literature is evolving, and new publications will be available at the time you read this text. Instead of giving a long list of the relevant literature available at the time this text is written, the sections include keywords that can be used to locate up-to-date literature.

For economic reasons, most of the illustrations are printed in black and white. Colour versions are available at the web site accompanying this book:

`http://sintro.r-forge.r-project.org/`.

Additional material and updates will be available at this site.

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:**

```
Input
1 + 2
```

```
Output
3
```

may correspond to a screen output of

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

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

Acknowledgements

Thanks to the R core team for comments and hints. Special thanks to Friedrich Leisch (R core team) and Antony Unwin (Univ. Augsburg) who worked through an early version of this manuscript. Thanks to Rudy Beran, Lucien Birgé, Dianne Cook, Lutz Dümbgen, Jan Johannes, Deepayan Sarkar, Bill Venables, Ali Ünlü and Adalbert Wilhelm for comments and hints.

Thanks to Dagmar Neubauer and Shashi Kumar for helping with the TeX pre-production, and very special thanks to Gerben Wierda for making the necessary tools accessible.

Literature and Additional References

[30] R Development Core Team (2000–2008): An Introduction to R.
See: `<http://www.r-project.org/manuals.html>`.

[34] R Development Core Team (2000–2008): R Reference Manual.
See: `<http://www.r-project.org/manuals.html>`.

The Omega Development Group (2008): Omega.
See: `<http://www.omegahat.org/>`.

[2] Becker, R.A.; Chambers, J.M.; Wilks, A.R. (1988): *The New S Language.*
Chapman & Hall, New York.

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

[6] Cleveland, W.F. (1993): *Visualizing Data.*
Hobart Press, Summit NJ.

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

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

Contents

CHAPTER 1

Basic Data Analysis

1.1 R Programming Conventions

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

R ***Conventions***	

Numbers	A point is used as a decimal separator. Numbers can be written in exponential form; the exponential part is introduced by `E`. Numbers can be complex numbers; the imaginary part is marked by `i`.
	Example: `1`
	`2.3`
	`3.4E5`
	`6i+7.8`
	Numbers can take the values `Inf`, `-Inf`, `NaN` for "not a number" and `NA` for "not available" = missing.
	Example: `1/0` results in `Inf`
	`0/0` results in `NaN`
	`NA` is used as a placeholder for missing numbers.

Strings	Strings are delimited by `"` or `'`.
	Example: `"ABC"`
	`'def'`
	`"gh'ij"`

Comments	Comments start with `#` and go to the end of the current line.

To allow for non-trivial examples, we anticipate a detail: in R, `a:b` is a sequence of numbers. If $a \leq b$, `a:b` is the sequence starting at `a` to at most `b` in steps of 1. If $a < b$, `a:b` is the sequence starting at `a` to at least `b` in steps of -1.

R ***Conventions***	
Objects	The basic elements in R are objects. Objects have types, for example `logical` or `integer`. Objects can have a class attribute specifying more complex type information. *Example:* The basis 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 .
	Examples: `x`
	`y_1`
	Lower- and uppercase are treated as different.
	Examples: `Y87`
	`y87`

Assignments	Assignments have the form *Syntax:* `name <- value` or alternatively `name = value`. *Example:* `a <- 10` `x <- 1:10`
Queries	If only the name of an object is entered, the value of the object is returned. *Example:* `x`
Indices	Vector components are accessed by index. The lowest index is 1. *Example:* `x[3]` The indices can be specified directly, or using symbolic names or rules. *Examples:* `a[1]` the first element `x[-3]` all elements except the third `x[ x^2 < 10]` all elements where $x^2 < 10$

Help and Inspection	

Help	Documentation and additional information about an object can be requested using `help`.
	Syntax: `help(name)`
	Examples: `help(exp)`
	`help(x)`
	Alternative form `?name`
	Examples: `?exp`
	`?x`
	A hypertext (currently HTML) version of R's online documentation is activated by `help.start()`. This allows us to search by topics, and provides a more structured access to information.

Inspection	`help()` can only provide information that has been prepared in advance. `str()` can inspect the actual state of an object and display this information. *Syntax:* `str(object, ...)` *Examples:* `str(x)`

R ***Conventions***	
Functions	Function calls in R have the form: *Syntax:* `name(argument` ...) *Example:* `e_10 <- exp(10)` This convention holds even when there are no arguments at all. *Example:* To quit R, you call a "quit" function `q()`. Function arguments are treated in a very flexible way. They can have default values, which are used if no explicit argument value is given. *Examples:* `log(x, base = exp(1))`

(cont.)→

R ***Conventions*** (cont.)	
	Functions can be ***polymorphic***. For a polymorphic function, the actual function is determined by the class of the actual arguments. *Examples:* `plot(x)` # a one-dimensional serial plot `plot(x, x^2)` # a two-dimensional scatter plot `summary(x)`
Operators	When applied to vectors, operators operate on each of the vector components. *Example:* For vectors `y, z`, the product `y*z` is the vector of component-wise products. Operators are special functions. They can be called in prefix form (function form). *Example:* `"+"(x, y)` When applied to two operands with different lengths, the smaller operand is repeated cyclically. *Example:* `(1:2)*(1:6)`

1.2 Generation of Random Numbers and Patterns

Our subject is statistical methods. As a first step, we apply the methods in simulations, that is, we use synthetic data. Generating these data is largely under our control. This gives us the opportunity to gain experience with the methods and allows a critical evaluation. Only then will we use the methods for data analysis.

1.2.1 Random Numbers

Random variables with a uniform distribution can be generated by the function `runif()` Using `help(runif)` or `?runif` we get information on 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)
```

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. Must be finite.
`log, log.p`	logical; if TRUE, probabilities p are given as log(p).
`lower.tail`	logical; if TRUE (default), probabilities are $P[X \le x]$, 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 $min \le x \le max$.
For the case of $u := min == max$, the limit case of $X \equiv u$ is assumed, although there is no density in that case and `dunif` will return `NaN` (the error condition).
`runif` will not generate either of the extreme values unless `max = min` or `max-min` is small compared to `min`, and in particular not for the default arguments.

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
```

The help information tells us that as an argument for `runif()` we have to supply the number `n` of random variates to generate. As additional arguments for `runif()` we can specify the minimum and the maximum for the range of the random numbers. If we do not specify additional arguments, 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 arguments can be supplied in the defined order, or specified by name. If the name of the argument 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 arguments. 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 arguments, even if it is 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
```

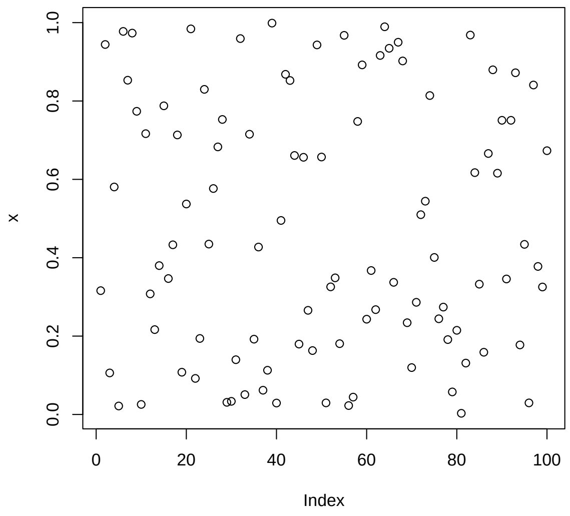
returns the values. By default, it is written to the output, and we can inspect the result. We get a graphical representation, the ***serial plot***, a scatterplot of the entries `x` against its running index, by using

```
plot(x)
```

***Example* 1.1: A Simple Plot**

Input

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



Exercise 1.1	
	Try experimenting with these plots and ***`runif()`***. Do the plots show images of random numbers? To be more precise: do you accept these plots as images of 100 independent realisations of random numbers, distributed uniformly on $(0, 1)$? Repeat your experiments and try to note as precisely as possible the arguments you have for or against (uniform) randomness. What is your conclusion? 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 clearly as possible. (cont.)→

Exercise 1.1	(cont.)
	Hint: For comparison, you can keep several plots in a window. The code `par(mfrow = c(2, 3))` parametrises the graphics system to show six plots simultaneously, arranged rowwise as a 2×3 matrix (2 rows, 3 columns). The function `par` is the central function to control graphics parameters. For more information, see `help(par)`.

We reveal the secret[1]: the numbers are not random, but completely deterministic. In the background, `runif()` builds a deterministic sequence z_i using iterated functions. In the simplest 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. To start, an initial value z_0, the ***seed***, must be given. Computationally there is just one variable, `.Random.seed`, which holds the current state z_i for $i = 0, 1, \ldots$, and is managed by the system. The successive values returned are in the hands of the user.

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).

This is the simplest case. Various other algorithms for random number generation are available, but all follow the same scheme. For more information on available random number generators, see `help(.Random.seed)`. See also Appendix A.21 (page A-228).

Even knowing the secret, a lot of additional knowledge is needed to prove that the generated sequence does not follow the rules that apply to a sequence of independent identically distributed random numbers from a uniform distribution. Sequences that 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

[1] ... at least part of it. The random number generators used in R can be customised and are 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 $x_{i+1} = (x_i \times 7^5) \mod 2^{31} - 1$.

under known conditions. In this context, we use pseudo-random numbers as if we had true random numbers.

On the other hand, we can take pseudo-random numbers as a challenge: are we capable of detecting 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 we have a challenge. Are we capable at all of detecting 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?

1.2.2 Patterns

Besides the possibility of generating pseudo-random numbers, R provides several possibilities of generating regular sequences. In many cases these can replace loops, which are common in other languages. Here is an initial survey:

R ***Sequences***	
`:`	generates a sequence from ⟨`begin`⟩ to at most ⟨`end`⟩. *Syntax:* ⟨`begin`⟩`:`⟨`end`⟩ *Examples:* `1:10` `10.1:1.2`
`c()`	"combine". combines arguments to a new vector. *Syntax:* `c(..., recursive = FALSE)` *Examples:* `c(1, 2, 3)` `c(x, y)` If the arguments are complex data types, the function will descend recursively if called with `recursive = TRUE`.
`seq()`	generates general sequences. *Syntax:* See `help(seq)`
`rep()`	repeats an argument. *Syntax:* `rep(x, times, ...)` *Examples:* `rep(x, 3)` `rep(1:3, c(2, 3, 1))`

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

Exercise 1.2	
	Use `plot(sin(1:100))` to generate a plot of a discretised sine function. Use your strategy from Exercise 1.1. Does your strategy detect that the sine function is not a random sequence? *Hint:* If you do not recognise the sine function at first sight, use `plot(sin (1:100), type = "l")` to connect the points.

Listing the numbers of a data set, as for example the output of a random number generator, rarely helps 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. The line plot suggested by the hint in Exercise 1.2 already illustrates an analysis beyond scatterplot. It uses a crude interpolation.

1.3 Case Study: Distribution Diagnostics

We need specific strategies to detect the presence or the violation of structures. We use random numbers to illustrate how these strategies can look. Here we concentrate on the distribution properties. Let us assume that 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 re-scaling to $(\min, \max)$, which might be necessary, but it is a technical detail that does not affect the investigation substantially. So for now we consider the case $\min = 0; \max = 1$.

Realisations of random variables do not allow us to read off the distributions directly. This is our critical problem. We need characterisations of the distribution which allow an empirical inspection. Of course we can view the observations as measures. For n observations $X_1, \ldots, X_n$ we can 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 the original distribution P. Some properties get lost without repair. Infinitesimal properties are among these. So for example P_n is always concentrated on finitely many points; empirical distributions are always discrete, irrespective of the underlying distribution. To analyse distributions based on data, we need functionals that can be determined based on realisations of random variable and that can be compared to the corresponding functionals of theoretical distributions. One strategy is to restrict ourselves to a family of test sets that is treatable empirically.

Example 1.2: 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 $X_1, \ldots, X_n$, the corresponding empirical distribution function is

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

Example 1.3: Histogram

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

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

as test sets.

Instead of the distribution P we consider the vector $\big(P(A_j)\big)_{j=1,\ldots,J}$. Its graphical representation is called a ***histogram***. The empirical version is the vector $\big(P_n(A_j)\big)_{j=1,\ldots,J}$.

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 here 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.

Example 1.4: 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 functions are called ***kernels*** and denoted by K. We require that the integral over a kernel exists, and conventionally the kernel is normalised so that $\int K(x)dx = 1$. Some commonly used kernels are listed in Table 1.9 and shown in Figure 1.1. 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.)

By shift and scaling each kernel gives rise to the family

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

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

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

The function

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

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

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

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	$(2)^{-1/2}\ exp(-x^2/2)$

Table 1.9 *Some commonly used kernels. See Figure 1.1.*

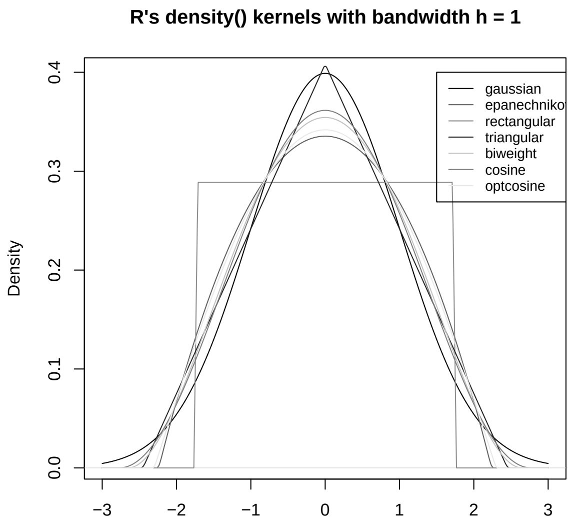


Figure 1.1 *Kernels in* R*. See Example 1.4 on page 11 and Table 1.9. See Colour Figure 1.*