 — der statistische Taschenrechner für Anfänger und Profis: eine kleine Einführung

Hans Peter Wolf

Version: 28.06.2006

Inhaltsverzeichnis

1 Hintergrund, Installation und erste Schritte mit R

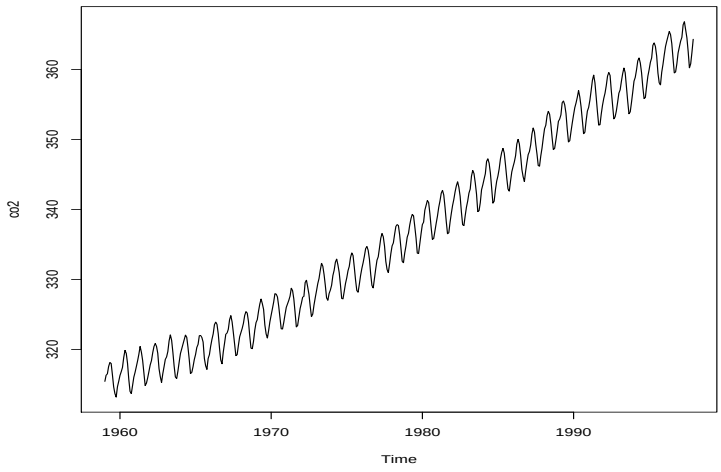
Statistiker und werdende Statistiker, die in einer vielseitigen Umgebung ihre statistischen Vorstellungen realisieren möchten, finden in R ein mächtiges Werkzeug. Die wesentlichen Konzepte von R werden hier in aller Kürze vorgestellt. Abschnitten, die in der ersten Sitzung überlesen werden sollten, ist ein Kurvenzeichen[1] vorangestellt: 🍥

Entstehungsgeschichte. Den Ursprung von R bildet S, eine Sprache wie auch eine Arbeitsumgebung, die in den 80-er Jahren bei `AT&T` Becker, Chambers und Wilks für Statistiker entwickelt haben. Interaktivität, Graphik, einfacher Umgang mit Daten sowie Erweiterungsfähigkeiten waren schon damals die hervorstechenden Eigenschaften.[2] Als kommerzielles Produkt von MathSoft (heute Insightful) vertrieben erfuhr S unter dem Namen S-Plus eine große Verbreitung und Reifung. Zur Realisierung eigener Vorstellungen schufen in den 90-er Jahren *R*oss Ihaka und *R*obert Gentleman einen Interpreter namens *R*, für den sie die S-Syntax übernahmen.[3] Diese Grundsatzentscheidung sowie kostenlose Erhältlichkeit, Offenheit und Erweiterbarkeit sind verantwortlich für die rasante Entwicklung des *R-Projektes.* Seit 1997 wird R zentral durch das *R Core Team* vorangetrieben. Durch die heute erreichte Qualität schneidet R bei einen Vergleich mit kommerziellen Produkten sehr gut ab.[4] Alle wesentlichen Informationen über R einschließlich Dokumentationen, Quellcodes und lauffähigen Programmen lassen sich über die Projekt-Homepage `http://cran.r-project.org` finden.

Ein Beispiel. Betrachten wir ein Beispiel. Wir wollen zu dem Datensatz `co2`, der CO_2-Konzentrationen im Zeitablauf zeigt, die Werte graphisch darstellen und einige typische Statistiken berechnen. Aus Sicht eines Statistikers erfordert diese Aufgabe drei Schritte: *Daten bereitstellen*, *Zeichnung anfertigen* und *Statistiken berechnen.* Folgerichtig sind in der R-Umgebung drei Kommandos einzugeben:

in 1:
```
data(co2)
plot(co2,type="l")
summary(co2)
```

R erzeugt nach der Eingabe der Befehle selbstständig eine Graphik ...



... und wirft Ergebnisse aus:

out 1:
```
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
313.2   323.5   335.2   337.1   350.3   366.8
```

[1] Diese Idee wie auch das Kurvenzeichen gehen auf D.E. Knuth zurück.
[2] Literatur: `http://cm.bell-labs.com/cm/ms/departments/sia/jmc/pub.html`
[3] Literatur: `http://cran.r-project.org/doc/html/interface98-paper/paper_5.html`
[4] Vgl.: `http://www.sciviews.org/other/benchmark.htm`

Installation. Damit der Leser dieses Beispiel ausprobieren kann, muss er sich R beschaffen und starten. R ist für WINDOWS leicht installierbar und zu bekommen über die Seite `http://cran.r-project.org`. Der direkte Zugriff auf die selbstentpackende Software lautet:

```
http://cran.at.r-project.org/bin/windows/base/R-2.3.1-win32.exe
```

Dieses Programm ist unter Windows herunterzuladen und zu aktivieren.
Zum Start von R ist nach Installation nur die neue R-Ikone anzuklicken. Unter WINDOWS öffnet sich ein Fenster zur Kommunikation mit R.

Arbeitsweise. Die drei Kommandos sind über Tastatur zu übermitteln. Hierbei erfährt der Neuling gleich die Arbeitsweise: Der Interpreter zeigt dem Anwender ein Eingabeaufforderungszeichen `>` (auch `Prompt` genannt). Hinter diesem wird das gewünschte Kommando eingegeben und durch Druck der Eingabetaste (`Return`) an das System übergeben. Berechnungsergebnisse werden unmittelbar unter der Eingabezeile ausgegeben, Graphiken erscheinen in einem isolierten Fenster.

Fehler. Fehler in einer noch nicht abgeschickten Zeile lassen sich vor der Übergabe beheben. Sind eingegebene Kommandos syntaktisch inkorrekt oder nicht auszuwerten, antwortet das System mit einer Fehlermeldung, und es erscheint wieder das Prompt-Zeichen als Eingabeaufforderung. Nun kann der Anwender erneut sein Glück wagen. Zur Erleichterung der Tipparbeit lassen sich durch Betätigung der Cursor-Tasten (↑, ↓) die letzten Eingaben in die Eingabezeile kopieren. Als Demonstration sei eine fehlerhafte Anweisung zur Mittelwertberechnung gezeigt:

in 2:
```
mean(co2))
```

Hierauf antwortet R mit:

out 2:
```
Error: syntax error
```

Na, wo liegt der Fehler? Es folgt die korrekte Anweisung:

in 3:
```
mean(co2)
```

... und das korrekte Ergebnis:

out 3:
```
[1] 337.0535
```

Arbeitsende. Die Arbeit mit R wird beendet durch Eingabe des Befehls `q()`.
Während des Beendigungsprozesses kann der Anwender entscheiden, ob er den Zustand der Umgebung speichern möchte. Die Speicherung der Arbeitsumgebung (*workspace*) erlaubt die spätere Fortführung einer angefangenen Arbeit. Hierbei werden neu geschriebene Funktionen und neu geschaffene Datenobjekte, die Zwischenergebnisse repräsentieren können, kompakt als Datei abgelegt. Bei dem nächsten Start von R wird die alte Umgebung automatisch geladen.

Zusammenfassung. Fassen wir die Aktionen zusammen:

Aktion / Kommando	Beschreibung
Installation	Download und Durchführung der Installation
Klick auf R*-Ikone*	Start von R
`data(co2)`	Bereitstellung des Datensatzes CO_2
`plot(co2,type="l")`	Plot von CO_2
`summary(co2)`	Berechnung einiger zusammenfassender Statistiken
`mean(co2)`	Berechnung des Mittelwertes
`q()`	Beendigung von R

Übersicht 1: erste Schritte mit R

Damit hat der Leser einen ersten Einblick bekommen.

2 Daten einlesen, Statistiken berechnen, Plots erstellen

Die Analyse von Datensätzen erfordert den Umgang mit Daten. Dazu werden in diesem Abschnitt verschiedene einfache Objekte zur Datenspeicherung vorgestellt und gezeigt, wie man mit diesen hantiert. Es werden elementare Berechnungen vorgeführt und einfache Plots erstellt.

2.1 Daten einlesen und hantieren

2.1.1 Zuweisung

Daten lassen sich in der Arbeitsumgebung durch eine *Zuweisung* auf einer Variablen ablegen. Durch diesen Prozess entsteht ein neues *Objekt*, auf das über den Objektnamen zugegriffen werden kann. Eine Zuweisung wird durch einen Pfeil, bestehend aus den beiden Zeichen `<-` ausgedrückt. Rechts vom Pfeil werden die Inhalte (Daten) der Zuweisung festgelegt, links vom Pfeil steht der Name, unter dem man die Daten ablegt und wiederfinden kann.
Die Arbeitsumgebung (`.GlobalEnv`) ist der zentrale Ort für die Objekte der Anwender. Von R bereitgestellte Objekte befinden sich in Paketen. Weiterhin existieren für den normalen Anwender meist unbemerkt lokale Umgebungen, in denen zum Beispiel lokale Objekte von Funktionen aufbewahrt werden. Hieraus erklärt sich, dass uns bei der Lektüre von Einführungen neben der einfachen Zuweisung noch andere Zuweisungstypen begegnen.

Beispielsweise speichern wir die Größe eines Preises ohne Umsatzsteuer als Zahl auf einer Variablen mit dem Namen `preis.netto`. Nach dieser Zuweisung können wir den abgelegten Wert über den Namen `preis.netto` zugreifen. Weisen wir den Steuersatz der Variablen `steuersatz` zu, erhalten wir den Betrag inklusive Steuer, wie folgt:

in 4:
```
preis.netto <- 200
steuersatz <- 0.16
preis.netto * (1.00 + steuersatz)
```

Diese Befehle liefern:

out 4:
```
[1] 232
```

Namenskonventionen. Als Objektnamen sind fast alle Anordnungen aus Buchstaben, dem Punkt `"."` und ab R-1.9.0 auch dem Unterstrich zugelassen. Weiter dürfen Namen vom ersten Zeichen abgesehen Ziffern enthalten. Deutsche Umlaute sind zwar auch erlaubt: Doch sollte man besser auf diese verzichten, da Umlaute immer wieder Schwierigkeiten verursachen. Übrigens werden große und kleine Buchstaben als unterschiedlich behandelt.

2.1.2 Eindimensionale Datensätze

Statistische Datensätze bestehen immer aus mehreren Zahlen sowie Zusatzinformationen. Deshalb können in R unter einem Namen ganze Datensätze abgelegt werden. R bietet für verschiedene Situationen unterschiedliche Möglichkeiten, um den zuzuweisenden Inhalt einer Zuweisung elegant festzulegen. Beispielsweise lässt sich mit dem *Sequenz-Generator* `":"` eine Folge von aufeinander folgenden ganzen Zahlen generieren und die Funktion `c()` (*concatenate*) fügt einzelne Elemente (zum Beispiel Zahlen) zu einem Vektor zusammen.

Beispiel. In preußischen Regimenten wurden *Todesfälle durch Hufschlag* über mehrere Jahre untersucht. Dabei wurde festgestellt, dass in 119 Fällen kein tödlicher Unfall, in 58 einer, in 17 zwei Unfälle, in 4 Fällen drei, in einem Fall vier und in einem Fall fünf Unfälle zu beklagen waren. Wir legen mit `c()` und `:` die Häufigkeit von bestimmten Anzahlen von Todesfällen auf den Variablen `h.anz.tote` und `anz.tote` ab. Wir wollen den Mittelwert der Todesfälle pro Regiment ermitteln. Sind $x_1, \ldots, x_n$ die einzelnen Beobachtungen, $a_1, \ldots, a_K$ die verschiedenen Ausprägungen und $n_1, \ldots, n_K$ die Häufigkeiten der Ausprägungen, erhalten wir den Mittelwert nach der Formel:

$$\overline{x} = \frac{1}{n}(x_1 + \cdots + x_n) = \frac{1}{n}(n_1 a_1 + \cdots + n_K a_K) = \frac{n_1}{n} \cdot a_1 + \cdots + \frac{n_K}{n} \cdot a_K$$

Mit der Funktion `sum()` zur Summation von Vektorelementen können wir deshalb eingeben:

in 5:
```
anz.tote <- 0:5
h.anz.tote <- c(119,58,17,4,1,1)
h.anz.tote.relativ <- h.anz.tote/sum(h.anz.tote)
sum(anz.tote*h.anz.tote.relativ)
```

out 5:
```
[1] 0.565
```

Indexzugriff. Wollen wir für die Mittelung nur die Fälle mit Todesfolge verwenden, dann müssen wir auf einen Teilvektor zugreifen. Hierfür bietet R Indexzugriffe mittels Indexklammern an. An den Positionen 2 bis 6, kurz `[2:6]`, befinden sich in den Vektoren die Einträge mit tödlichen Unfällen. Somit lässt sich das arithmetische Mittel ohne die Klasse "0-Tote" berechnet durch:

in 6:
```
sum(anz.tote[2:6]*h.anz.tote[2:6]/sum(h.anz.tote[2:6]))
```

out 6:
```
[1] 1.395062
```

R kennt verschiedene Arten der Indizierung. Weiter unter wird dieses Thema noch einmal aufgegriffen. Beispielsweise können wir über negative Indexwerte die Elemente benennen, die wir bei der Berechnung ausschließen wollen. Die folgende Anweisungszeile erarbeitet zur Demonstration ebenfalls das Ergebnis 1.395062.

in 7:
```
sum( anz.tote[-1]*h.anz.tote.relativ[-1] )
```

2.1.3 Tabellen und Matrizen

Ein Statistiker will die Häufigkeitstabelle natürlich unter einem einzigen Namen ablegen. Hierzu bedarf es einer Matrix-Struktur. Mit Hilfe der Funktion `cbind()` (*column* ***bind***) lassen sich Vektoren zu einer Matrix zusammenbinden.

in 8:
```
TdH <- cbind(anz.tote,h.anz.tote)
```

out 8:
```
     anz.tote h.anz.tote
[1,]        0        119
[2,]        1         58
[3,]        2         17
[4,]        3          4
[5,]        4          1
[6,]        5          1
```

Über Indexoperationen erhalten wir die einzelnen Spalten zurück, so dass wir wieder das Mittel berechnen können. Indexzugriffe besitzen für Matrizen die Struktur `[Zeileninidizes, Spaltenindizes]`. Einträge vor dem Komma in den Indexklammern beziehen sich auf Zeilen, Angaben nach dem Komma auf Spalten.

in 9:
```
sum(TdH[,1]*TdH[,2]/sum(TdH[,2]))
```

Mit der Funktion `matrix()` lässt sich `TdH` alternativ konstruieren. Die Spaltennamen können mit Hilfe von `colnames()` ergänzt werden.

in 10:
```
TdH <- matrix(c(anz.tote,h.anz.tote),nrow=6,ncol=2)
colnames(TdH)<-c("anz.tote", "h.anz.tote")
```

2.1.4 Datensätze in Dateien

In der Regel müssen Datensätze nicht eingetippt werden, sondern liegen in Dateiform vor. Zur Demonstration speichern wir zunächst die gerade erzeugten Objekte mit der Funktion `cat()` ab und lesen sie anschließend mit `scan()` wieder ein. Das Kommentarzeichen von R ist der *Lattenzaun:* #. Zeichen nach dem # werden von R nicht beachtet.

in 11:
```
# lege  anz.tote  in der Datei  anz-tot.txt  ab
cat(anz.tote, file="anz-tot.txt")
#
# lese die Datei  anz-tot.txt  auf das Objekt  x  ein
x<-scan(file="anz-tot.txt")
#
# lege  h.anz.tote in der Datei hanz-tot.txt ab
cat(h.anz.tote, file="h.anz-tot.txt")
#
# lese die Datei  h.anz-tot.txt  auf das Objekt  h.x  ein
h.x<-scan(file="hanz-tot.txt")
#
# berechne Mittel mit  x  und  h.x
sum(x*h.x/sum(h.x))
```

Wieder erhalten wir den schon bekannten Mittelwert. Gelingt uns Ex- und Import auch mit unserer Tabelle? Dazu greifen wir auf die Funktionen `write.table()` und `read.table()` zurück. Das neue Objekt `xy` gehört übrigens in die Klasse `data.frame`, ein Spezialfall einer *Liste*. Später erfährt der Leser hierzu mehr.

in 12:
```
# lege  TdH  in der Datei  tdh.txt  ab
write.table(TdH,file="tdh.txt")
#
# lese  tdh.txt  auf das Objekt  xy  ein
xy<-read.table(file="tdh.txt")
#
# berechne mit der Matrix  xy  den Mittelwert
sum(xy[,1]*xy[,2]/sum(xy[,2]))
```

2.1.5 Mehrdimensionale Datensätze

Bei statistischen Untersuchungen werden in der Regel mehrere Merkmale erhoben, so dass mehrdimensionale Datensätze den Ausgangspunkt für Analysen bilden. Wird mit einem Fragebogen nach den Merkmalen *Geschlecht, Größe, Gewicht und Wohnort* gefragt, könnte der Datensatz in einer Datei mit Namen `fragebogen.dat`, wie folgt, aussehen:

```
Geschlecht;Gewicht;Groesse;Wohnort
M;180;70;Berlin
W;175;65;Bremen
W;181;68;Berlin
M;170;72;Hamburg
...
```

In der ersten Zeile stehen Merkmalsnamen und in jeder weiteren die Ausprägungen eines Merkmalsträgers, wobei die einzelnen Ausprägungen durch `;`-Zeichen voneinander getrennt sind. Diese Datei können wir ohne Probleme einlesen und dem Objekt auf `fb.daten` ablegen:

in 13:
```
fb.daten<-read.table("fragebogen.dat",header=TRUE,sep=";")
```

out 13:
```
   Geschlecht Gewicht Groesse  Wohnort
1           M     180      70   Berlin
2           W     175      65   Bremen
3           W     181      68   Berlin
4           M     170      72  Hamburg
```

Damit steht dieses Objekt für statistische Analysen zur Verfügung. Einzelne Merkmale und einzelne Merkmalsträger können wir wieder über Indexzugriffe extrahieren:

in 14:
```
fb.daten[c(1,3),2:3]
```

out 14:
```
   Gewicht Groesse
1     180      70
3     181      68
```

Weiter lassen sich beispielsweise die Frauen auswählen durch:

in 15:
```
frauen<-fb.daten[,1]=="W"
fb.daten[frauen,]
```

out 15:
```
   Geschlecht Gewicht Groesse Wohnort
2           W     175      65  Bremen
3           W     181      68  Berlin
```

Oft möchte man mit den Ausprägungs-Vektoren der einzelnen Merkmale hantieren. Dazu kann das Objekt `fb.daten` *geöffnet,* und die einzelnen Vektoren können danach über die Spaltennamen angesprochen werden:

in 16:
```
attach(fb.daten)
Gewicht
```

Hierdurch erhalten wir:

out 16:
```
 [1] 180 175 181 170
```

2.1.6 Der Gebrauch von Funktionen

Bisher haben wir eine Reihe von Funktionen intuitiv und ohne weitere konzeptionelle Ausführungen eingesetzt, so dass nun ein paar Bemerkungen angebracht sind. Funktionen werden wie Datenobjekte über Namen angesprochen. Hinter dem Namen folgt eine linke runde Klammer, eine rechte runde Klammer schließt den Aufruf ab. Zwischen den Klammern können Argumente benannt werden. Mehrere Argumente müssen durch Kommata voneinander getrennt werden. Die Argumente besitzen Namen, wie zum Beispiel `file`. Soll ein bestimmtes Argument beim Aufruf einen speziellen Wert bekommen, kann dieses über den Namen benannt werden, wie zum Beispiel durch `file="tdh.txt"`. Entspricht die Reihenfolge der Argumente beim Funktionsaufruf der Reihenfolge in der Funktionsdefinition, können die Namen fehlen. Argumente besitzen zum Teil voreingestellte Werte. So führt ein fehlendes `file`-Argument bei der Funktion `scan()` zur Verwendung der Voreinstellung `""` und der Wirkung, dass Werte über Tastatur eingegeben werden müssen. Da die Menge der Funktionen, deren Argumente und die Wirkungsweisen überaus vielfältig sind, ist der Anwender auf eine die online-Hilfe angewiesen.

2.1.7 Die eingebaute Hilfe

`help.search`. Zu einem englischen Begriff finden wir über die Funktion `help.search()` alle erreichbaren Funktionen, in deren Dokumentation es eine Verbindung zu diesem Begriff gibt. Weiter können mit der Funktion `apropos()` Funktionen gefunden werden, deren Namen ein Zeichenmuster aufweist. Der Aufruf

in 17:
```
help.search("mean")
```

liefert uns eine längere Liste von Funktionen, bei denen Mittelwerte bedeutsam sind. Natürlich enthält diese Liste auch die Funktion `mean()` zur Berechnung des arithmetischen Mittels. `apropos("mean")` zeigt uns eine Liste von Objekten, in deren Namen die Zeichenfolge `mean` vorkommt.

`help`. Die Help-Seiten einzelner Funktionen zeigt uns die Funktion `help()`. Die Dokumentation zu `mean()` erhalten wir beispielsweise durch den Aufruf:

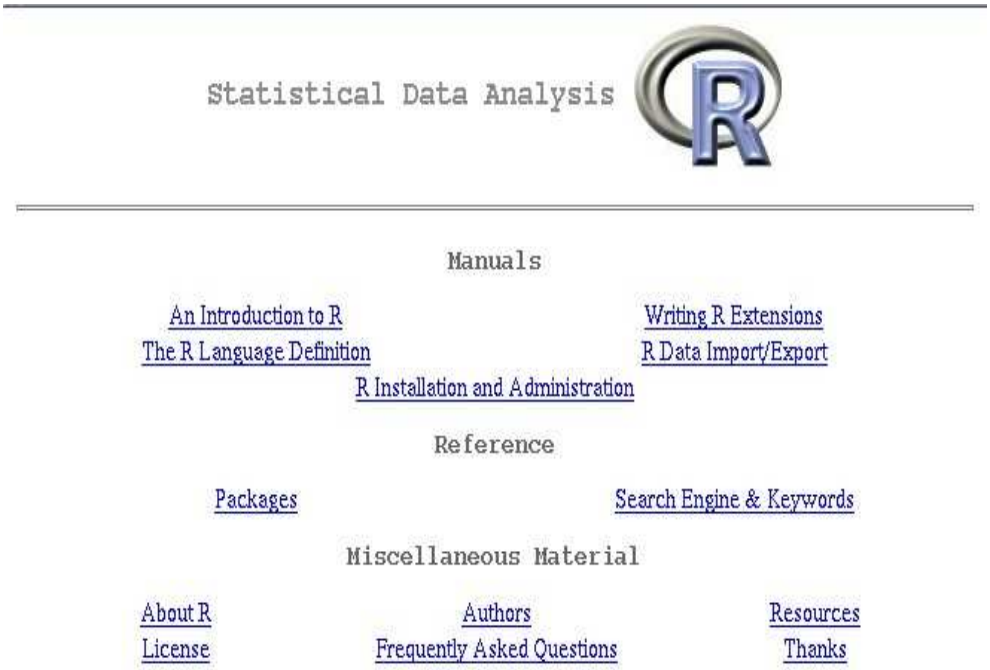
in 18:
```
help("mean")
```

Schreibfaule können alternativ zu `help("mean")` auch `help(mean)` oder auch nur `?mean` eingeben und erhalten denselben Text angezeigt.
Der Hilfetext zerteilt sich in Bemerkungen zur Verwendung, zur Syntax, zu den Inputgrößen, zum Ergebnis, nähere Details, Beispiele und Hinweise auf andere Funktionen. So erfahren wir aus der Hilfe zu `mean()`, dass wir vor der Mittelwertbildung über das Argument `trim` die `trim*100` Prozent kleinsten und größten Werte aus dem Datensatz entfernen, also *getrimmte Mittel* berechnen können.

`help.start`. Falls die Hilfe-Seiten auch im `html`-Format installiert sind, lässt sich ein Browser als Fenster zur Hilfe einsetzen. Für den Explorer als Browser unter Windows ist dafür folgendes Initialisierungskommande erforderlich:

in 19:
```
help.start(browser="explorer")
```

Im Browser erhalten wir bei Erfolg folgendes Navigationsfenster:



Von dieser Übersichtsseite ausgehend starten wir über `Search Engine & Keywords` die lokale Suchmaschine. Über `->packages ->base` gelangt man zu der Hilfe des Basispakets, über `->packages ->graphics` zu der der Graphikroutinen und über `->packages ->stats` zu den Hilfeseiten des Paketes mit Statistikroutinen. Zusammen findet man Dokumentationen zu mehr als 1500 Objekten. Es ist also wichtig, möglichst genau zu wissen, wonach man sucht. Ein erneuter Aufruf von `help(mean)` bewirkt jetzt die Anzeige der Hilfeseite von `mean()` im Browser.

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An R object. Currently there are methods for numeric data frames, numeric vectors and dates. A complex vector is allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Value

For a data frame, a named vector with the appropriate method being applied column by column.
If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed.
If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

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

See Also

`weighted.mean`, `mean.POSIXct`

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))

data(USArrests)
mean(USArrests, trim = 0.2)
```

[Package Contents]

2.1.8 Interaktive Dateneingabe.

In der heutigen Zeit sind die Computernutzer an interaktive Oberflächen gewöhnt. Falls Datensätze per Hand eingegeben werden, kann dieses außerhalb von R mit Hilfe eines komfortablen Editors geschehen. Für kleinere Eingaben oder Korrekturen bietet R die Funktion `data.entry()`, mit der sich ein schlichtes Fenster zur Eingabe von Matrizen nutzen lässt.

2.1.9 Zusammenfassung – Eingabe

Fassen wir die neuen Aktionen wieder in einer Tabelle zusammen.

Aktion / Kommando	Beschreibung: liefert /generiert ...
`5:10`	generiert die ganzen Zahlen von 5 bis 10
`c(-10,5,-5, 0)`	generiert den Zahlenvektor: $(-10, 5, -5, 0)$
`x[pos]`	liefert den durch `pos` definierten Teilvektor von `x`
`cbind(x,y)`	generiert aus (Spalten) `x` und `y` eine Matrix
`rbind(x,y)`	generiert aus (Zeilen) `x` und `y` eine Matrix
`mat[z,sp]`	liefert Zeile(n) `z` und Spalte(n) `sp` der Matrix `mat`
`dim(mat)`	liefert Zeilen- und Spaltenanzahl der Matrix `mat`
`length(x)`	liefert Anzahl der Elemente von `x`
`cat(x, file="my-data")`	legt Inhalt von `x` in Datei `my-data` ab
`scan("my-data")`	liest aus Datei `my-data` Zahlenwerte
`write.table(x,file="my-tab")`	schreibt Tabelle `x` in die Datei `my-tab`
`read.table("my-tab")`	liest aus Datei `my-tab` eine gespeicherte Tabelle
`data(co2)`	stellt den in R vorrätigen Datensatz `co2` bereit
`scan()`	liest Zahlen über Tastatur
`data.entry(mat)`	erlaubt interaktive Veränderung von `mat`
`help(help)`	zeigt die Hilfe zu dem Objekt `help`
`help.search("mean")`	sucht Objekte bzgl. eines Stichwortes `mean`
`help.start()`	startet Online-Hilfenzugriff per Browser
`apropos()`	sucht Objekte mit vorgegebenen Muster

Übersicht 2: Funktionen zur Erstellung von und zum Umgang mit Objekten

2.2 R als Rechnenmaschine

2.2.1 Taschenrechnerei

Einfache arithmetische Berechnungen können fast wie gesprochen eingetippt wereden. Denn Addition, Subtraktion, Multiplikation, Division und Exponentiation werden durch Operatoren angeboten und sind durch die gebräuchlichen Zeichen `+,-,*,/,^` umgesetzt. Weitere Operatoren können folgender Tabelle entnommen werden. Die waagerechten Striche in der Tabelle grenzen Prioritätsklassen voneinander ab. Weiter oben stehende Einträge haben vor weiter unten befindlichen Vorfahrt. Da Multiplikation und Division oberhalb von der Addition residieren, gilt beispielsweise Punkt- vor Strichrechnung. Runde Klammern setzen natürlich die Prioritäten außer Kraft.
Einen herausragenden Vorteil gibt es gegenüber einem Taschenrechner: Viele Operatoren arbeiten auch auf Vektoren und Matrizen! Wird beispielsweise ein +-Zeichen zwischen gleichlange Vektoren gesetzt, werden die korrespondierenden Vektorelemente addiert, und es entsteht ein neuer Vektor gleicher Länge. Die Operatoren `+,-,*,/,^` arbeiten elementweise!

Symbol	Beispiel	Beschreibung
`^`	`x ^ y`	Exponentiation `x` hoch `y`
`-`	`- x`	Vorzeichen von `x`
`:`	`x : y`	Sequenzoperator, erzeugt Folge `x, x+1, ..., y` für `x < y` und `x, x-1, ..., y` für `x > y`
`/`	`x / y`	Division von `x` und `y`
`*`	`x * y`	Multiplikation von `x` und `y`
`-`	`x - y`	Subtraktion von `x` und `y`
`+`	`x + y`	Addition von `x` und `y`

Übersicht 3: verschiedene Operatoren

Zur Demonstration wollen wir die Standardabweichung des Datensatzes *Tote-durch-Hufschlag* nach der Formel

$$s = \sqrt{\sum_{j=1} \frac{n_j}{n-1} \cdot \left(a_j - \overline{x}\right)^2}$$

zu Fuß ausrechnen. n_j sind wieder die Häufigkeiten der Ausprägungen a_j.

in 20:
```
anz.tote <- 0:5
h.anz.tote <- c(119,58,17,4,1,1)
n <- sum(h.anz.tote)
x.quer<-sum(  h.anz.tote/n * anz.tote )
std.abw<-sum(h.anz.tote/(n-1)*(anz.tote-x.quer)^2)^0.5
```

und erhalten

out 20:
```
[1] 0.8362245
```

Lange Eingaben. Wie in der letzten Zeile können sich sehr lange Eingabezeilen ergeben. Wird sogar eine zweite Zeile erforderlich, kann auch nur der erste, unvollständige Teil der Anweisung durch RETURN übergeben werden. Dann antwortet der Interpreter mit einem +-Zeichen als Hinweis, dass für die syntaktisch korrekte Beendigung noch mehr einzugeben ist. Der Anwender kann nun hinter dem Fortführungsprompt mit der Eingabe fortfahren.

2.2.2 Statistische und mathematische Berechnungen

Schon bei der Mittelwertberechnung kam die Funktion `sum()` zum Einsatz. Die große Fülle der von R angebotenen Funktionen gestattet fast alle Berechnungsaufträge schnell einzugeben, die uns im statistischen Alltag begegnen. Viele heißen genau so, wie wir es von der Mathematik her kennen. Auch entspricht die Syntax weitgehend allgemeinen Vorstellungen, so dass für den Anfang eine große Übersichtsliste völlig ausreicht. Einzelheiten sind über die Online-Hilfe abrufbar.

Funktion	Beschreibung
`min(x)`	Minimum von `x`
`max(x)`	Maximum von `x`
`mean(x)`	Mittel von `x`
`median(x)`	Median von `x`
`range(x)`	Extrempunkte von `x`
`sd(x)`	Standardabweichung von `x`
`var(x)`	Stichprobenvarianz von `x`
`summary(x)`	zusammenfassende Statistiken von `x`
`fivenum(x)`	5-Punkte-Zusammenfassung von `x`
`IQR(x)`	Interquartilrange von `x`
`mad(x)`	mittlere absolute Abweichung `x`
`quantile(x,p)`	p-Quantil von `x`
`cor(x,y)`	Korrelation von `x` und `y`
`rank(x)`	Ränge der Elemente von `x`
`order(x)`	Sortierindex zu dem Vektor `x`
`sample(x)`	Permutation von `x`
`length(x)`	Anzahl der Werte von `x`

Übersicht 4: statistische Funktionen

Neben statistischen Funktionen sind verschiedene elementare mathematische Funktionen im Lieferumfang enthalten.

Funktion	Beschreibung
`sin(x)`	$\sin(x)$
`cos(x)`	$\cos(x)$
`tan(x)`	$\tan(x)$
`cot(x)`	$\cot(x)$
`exp(x)`	$\exp(x)$
`sum(x)`	$\sum_i x_i$
`cumsum(x)`	$\sum_j^i x_j$
`diff(x)`	Vektor der Differenzen von `x`
`prod(x)`	$\prod_i x_i$
`cumprod(x)`	$\prod_j^i x_j$

Übersicht 5: mathematische Funktionen

2.2.3 Verteilungsmodelle

Was wäre die Statistik ohne Wahrscheinlichkeitsverteilungen? Für eine Reihe stetiger und diskreter Verteilungen lassen sich Werte der Dichtefunktion, Werte der Verteilungsfunktion, Quantile sowie Zufallszahlen ermitteln. Die folgenden Tabelle zeigt eine Reihe direkt verfügbarer Verteilungen. Die Namen der R-Funktionen ergeben sich durch Vorhängen eines der Buchstaben `d`, `p`, `q`, `r` vor den in der Tabelle abgedruckten Verteilungsnamen mit folgenden Bedeutungen:

- `d` für *density* liefert Dichte- oder Wahrscheinlichkeitsfunktionswerte – `dexp(1:3)` berechnet die Werte der Dichte der Exponentialverteilung mit Parameter 1 an den Stellen 1, 2 und 3
- `p` für *probability* liefert Verteilungsfunktionswerte – `ppois(5,2)` berechnet die Wahrscheinlichkeit, dass eine Poisson-verteilte Zufallsvariable mit $\lambda = 2$ sich in $\{0, 1, 2, \ldots, 5\}$ realisiert.
- `q` für *quantile* liefert Quantile – `qnorm(0.95, 0,1)` berechnet den Prozentpunkt einer standardnormalverteilten Zufallsvariable zur Wahrscheinlichkeit 0.95
- `r` für *random* liefert Zufallszahlen – `runif(10)` generiert 10 Zufallszahlen aus einer Gleichverteilung von 0 bis 1.

Verteilungs-name	Verteilung	notwendige Argumente	optionale Argumente	Defaulteinstellungen
beta	Beta	`shape1, shape2`		
binom	Binomial	`size, prob`		
cauchy	Cauchy		`location, scale`	0,1
chisq	Chi-Quadrat	`df`		
exp	Exponential		`rate`	1
f	F	`df1, df2`		
gamma	Gamma	`shape`		
geom	Geometrisch	`prob`		
hyper	Hypergeom.	`m, n, k`		
lnorm	Log-Normal		`meanlog, sdlog`	0,1
logis	Logistisch	`location`	`scale`	0,1
nbinom	Pascal	`size, prob`		
norm	Normal	`mean, sd`		0,1
pois	Poisson	`lambda`		
t	t	`df`		
unif	Rechteck		`min, max`	0,1
weibull	Weibull	`shape`	`scale`	-,1

Übersicht 6: Verteilungsmodelle

Damit erhält man den Wert der Wahrscheinlichkeitsfunktion einer mit $n = 12$ und $p = 0.5$ binomialverteilten Zufallsvariable an der Stelle 5 durch den Befehl `dbinom(5,12,0.5)`. Den Wert der Verteilungsfunktion liefert der Aufruf `pbinom(5,12,0.5)`. Hier eine kurze Demonstration, bei der die Funktionen `print()` zum angemessenen Ausdruck eines Objektes und `cat()` zur Textausgabe benutzt wurden.

in 21:
```
cat("dbinom(5,12,0.5) "); print(dbinom(5,12,0.5))
cat("pbinom(5,12,0.5) "); print(pbinom(5,12,0.5))
cat(" Approximation durch Normalverteilung:\n")
cat("pnorm(5.5,12*0.5,sqrt(12*0.5*0.7)) ")
print(pnorm(5.5,12*0.5,sqrt(12*0.5*0.7)))
```

out 21:
```
dbinom(5,12,0.5) [1] 0.1933594
pbinom(5,12,0.5) [1] 0.387207
 Approximation durch Normalverteilung:
pnorm(5.5,12*0.5,sqrt(12*0.5*0.7)) [1] 0.4036251
```

An der Stelle des Wertes 5 kann ebenso ein Vektor stehen. So liefert beispielsweise `dbinom(c(3,5,10),12,0.5)` die Werte der Wahrscheinlichkeitsfunktion an den Stellen 3, 5 und 10.

`sample()`. Zum Abschluss sei noch einmal die Funktion `sample()` hervorgehoben, mit der aus einer Menge von Elementen eine Stichprobe gezogen werden kann. Die Syntax lässt sich aus einem Beispiel-Einsatz erkennen: Es liegen die Körpergrößen von 20 Studierenden vor. Nun soll zur Demonstration von Stichprobeneffekten aus dieser Stichprobe eine Stichprobe vom Umfang 10 mit Zurücklegen gezogen werden.

in 22:
```
KG<-c(171,173,176,170,168,175,198,170,177,198,170,173,201,
      168,205,176,184,183,184,180)
stichprobe<-sample(x=KG,size=10,replace=T)
cat("Datensatz - Zusammenfassung\n")
print(summary(KG))
cat("Stichprobe\n")
print(stichprobe)
cat("Stichprobe - Zusammenfassung\n")
summary(stichprobe)
```

Wir erhalten:

out 22:
```
Datensatz - Zusammenfassung
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  168.0   170.8   176.0   180.0   184.0   205.0
Stichprobe
 [1] 176 205 170 201 168 170 176 176 168 198
Stichprobe - Zusammenfassung
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  168.0   170.0   176.0   180.8   192.5   205.0
```

Erneute Anfragen an den Zufallszahlengenerator liefern neue Zufallszahlen. Um einen Strom von Zufallszahlen zu rekonstruieren, lässt sich der Startpunkt des Generators mit der Funktion `set.seed(seed)` setzen. `seed` muss hierbei eine ganze Zahl sein. Beispielsweise kann die oben vorgeführte Stichprobenziehung durch vorherige Setzung `set.seed(13)` exakt wiederholt werden.

2.2.4 Kategoriale Daten

Häufigkeitstabellen. Die Analyse kategorialer Daten beginnt mit der Erstellung einer Häufigkeitstabelle. Bei der Betrachtung eines einzigen Merkmals entsteht aus der Urliste eine eindimensionale Datenstruktur, die sich gut als Vektor speichern lässt. Für zwei Merkmale benötigen wir eine zweidimensionale Tabelle, deren Speicherung eine Matrix erfordert. R bietet für Häufigkeitstabellen die Objektklasse `table`. Vertreter dieser Klasse generiert die Funktion `table()` durch Auszählen. Relative Häufigkeiten erhalten wir durch Division durch die Beobachtungsanzahl, die Funktion `margin.table()` berechnet Summenspalten und -zeilen und `prop.table()` Zeilen- bzw. Spaltenverteilungen. Zur Demonstration erzeugen wir einen künstlichen Datensatz zu den Merkmalen Augenfarbe und Geschlecht und wenden die beschriebenen Funktionen an. Das Protokoll dürfte selbsterklärend sein.
Wir erzeugen eine Tabelle:

in 23:
```
set.seed(13)
geschlecht <-sample(c("M","W"), 20, T)
augen.farbe<-sample(c("blau","gruen","grau","braun"),20,T)
con.table  <-table(geschlecht,augen.farbe)
```

Kontingenztabelle mit relativen Häufigkeiten liefert eine einfache Division:

in 24:
```
con.table/sum(con.table)
```

out 24:
```
          augen.farbe
geschlecht blau braun grau gruen
         M 0.10 0.10  0.25 0.20
         W 0.10 0.15  0.05 0.05
```

Den rechten und unteren Rand bekommen wir mittels `margin.table`:

in 25:
```
margin.table(con.table,1)
```

out 25:
```
geschlecht
 M  W
13  7
```

in 26:
```
margin.table(con.table,2)
```

out 26:
```
augen.farbe
 blau braun  grau gruen
    4     5     6     5
```

Die Verteilung innerhalb der Zeilen berechnet `prop.table`:

in 27:
```
prop.table(con.table,1)
```

out 27:
```
          augen.farbe
geschlecht blau      braun     grau      gruen
         M 0.1538462 0.1538462 0.3846154 0.3076923
         W 0.2857143 0.4285714 0.1428571 0.1428571
```

2.3 Graphiken

R überzeugt viele Neulinge durch die Leichtigkeit und Vielseitigkeit, mit der wir die unterschiedlichsten Graphiken erzeugen können. Besonders hervorzuheben ist, dass R eine Reihe von High-Level-Plot-Funktionen anbietet, die komplette Graphiken produzieren. Die Wirkung dieser Funktionen kann durch optionale Parameter näher spezifiziert werden. Den so erstellten Graphiken lassen sich mit Low-Level-Plot-Funktionen weitere graphische Elemente hinzufügen. Zusätzlich können verschiedene Einstellungen der graphischen Umgebung global gesetzt werden. Um sich beeindrucken zu lassen, sei empfohlen, die graphische Demo durch das Kommando `demo(graphics)` zu starten und die angezeigten Graphiken zu studieren. In jedem Schritt der Demo wird dem Anwender eine Kommandosequenz gezeigt, die dann durch einen RETURN-Druck zur Ausführung kommt.

2.3.1 Graphische High-Level-Routinen

Häufig nachgefragte statistische Graphiken müssen wie auf Knopfdruck generierbar sein. Hierfür bietet R eine Reihe High-Level-Plot-Funktionen an. Wie folgender Tabelle entnommen werden kann, lassen sich mit einem Kommando Kuchen- und Balkendiagramme, Boxplots, Histogramme, Scatterplots usw. anfertigen.

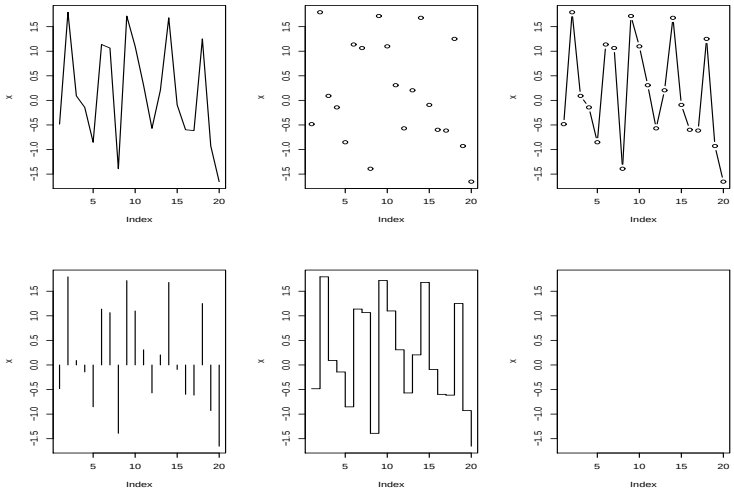
Funktion	erzeugt Darstellung:
`barplot(x)`	Balkendiagramm
`boxplot(x)`	Boxplot(s)
`curve(x^3,-2,2)`	zeichnet Funktion x^3 im Bereich [-2,2]
`dotchart(x)`	Dotchart
`hist(x)`	Histogramm
`pairs(x)`	paarweise Scatterplots
`pie(x)`	Kuchendiagramm
`plot(x)`	Scatterplot: `x` gegen seinen Index
`plot(x,y)`	Scatterplot: `y` gegen `x`
`qqplot(x,y)`	Quantile von `y` gegen die von `x`
`qqnorm(x)`	Quantile von `x` gegen die der Normalverteilung
`stripchart(x)`	Stripchart von `x`

Übersicht 7: High-Level-Plotfunktionen

Die Beschreibungen der Tabelle vermitteln einen kurzen Überblick über einfach erstellbare, typisch statistische Graphiken. Für umfassende Erklärungen zu den einzelnen Funktionen sei der Leser zunächst an die Online-Hilfe verwiesen. Dort erfahren wir, welche Wirkungen die optionalen Parameter besitzen.

`plot()`. Anhand der Variation des Arguments `type` von `plot()` sei ein kleiner Einblick in die Gestaltungsvielfalt gegeben.

in 28:
```
data(co2); x<-co2[1:20]
par(mfrow=c(2,3))
plot(x,type="l"); plot(x,type="p"); plot(x,type="b")
plot(x,type="h"); plot(x,type="s"); plot(x,type="n")
par(mfrow=c(1,1))
```



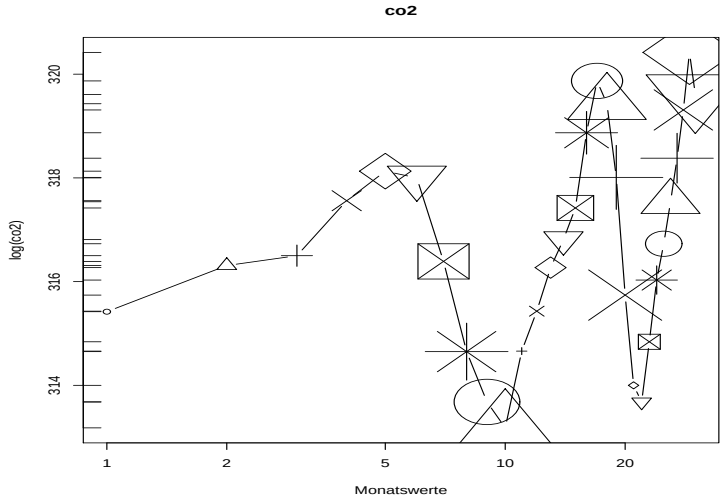
Optionale Argumente. Damit sind die Gestaltungsmöglichkeiten noch nicht ausgereizt. Denn einerseits werden je nach Klasse die an die Funktion `plot` übergebenen Daten unterschiedlich dargestellt. Ist `x` ein einfacher Vektor, werden die Elemente von `x` gegen den entsprechenden Indexvektor geplottet. `plot(x,y)` zeichnet einen Scatterplot der Vektoren `x` und `y`. Auf Basis einer zweispaltigen Matrix `xy` erstellt `plot(xy)` einen Plot der Werte der zweiten gegen die der ersten Spalte. Andererseits werden von `plot` noch andere Argumente akzeptiert, siehe dazu die folgende kleine Auswahl:

Parameter	Beispiel	Beschreibung
`type`	`type="p"`	Darstellung der Daten durch Punkte
	`type="l"`	Darstellung der Daten durch Linien
	`type="b"`	Darstellung der Daten durch Punkte und Linien
	`type="n"`	Daten sollen gar nicht darstellt werden
`xlim`	`xlim=c(1,100)`	Grenzen der x-Achse (z.B.: 1 und 100)
`ylim`	`ylim=c(0,1)`	Grenzen der y-Achse (z.B.: 0 und 1)
`xlab`	`xlab="x-Achse"`	Beschriftung der x-Achse
`ylab`	`ylab="y-Achse"`	Beschriftung der y-Achse
`log`	`log="xy"`	x- und y-Achse logarithmisch
`bty`	`bty="n"`	Art der Umrahmung
`lty`	`bty=1`	Linentyp (aus 1,...,6)
`pch`	`pch=1`	Zentralsymbol für Punkte (aus 0:255)
`cex`	`cex=2`	Größe für Zentralsymbole (aus 1:25)
`col`	`col="red"`	Farbe der Darstellung
`main`	`main="Testplot"`	Überschrift für das Bild

Übersicht 8: Parameter von `plot`

Die aufgelisteten Parameter können mit den gewünschten Werten durch Kommata getrennt nach den Daten der Funktion `plot()` übergeben werden. Diese Parameter lassen sich, wenn sie dort Sinn machen, auch für andere High-Level-Plot-Funktionen verwenden.
Es folgt ein Beispiel, von dem ausgehend der Leser weitere Fingerübungen unternehmen kann.

in 29:
```
y<-co2[1:30]
plot(y,main="co2",log="x",cex=1:10,pch=1:8,
     xlab="Monatswerte",ylab="log(co2)",type="b")
rug(y,side=2)
```



Die Funktion `rug()` ergänzt am Rand zu den einzelnen y-Werten kleine Ticks. Hierdurch bekommen wir einen Eindruck von der eindimensionalen Verteilung der y-Werte und haben damit gleichzeitig einen Vertreter der Low-Level-Routinen kennengelernt.
Gerade für Balkendiagramme besteht die Notwendigkeit völlig unterschiedlicher Designs. Deshalb werfen wir noch einen Blick auf die Argumente der Funktion `barplot.default()`, die von der Funktion `barplot()` aktiviert wird.

in 30:
```
args(barplot.default)
```

out 30:
```
function (height, width = 1, space = NULL, names.arg = NULL,
    legend.text = NULL, beside = FALSE, horiz = FALSE, density = NULL,
    angle = 45, col = heat.colors(NR), border = par("fg"), main = NULL,
    sub = NULL, xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
    xpd = TRUE, axes = TRUE, axisnames = TRUE, cex.axis = par("cex.axis"),
    cex.names = par("cex.axis"), inside = TRUE, plot = TRUE,
    axis.lty = 0, offset = 0, ...)
```

Über `space` wird beispielsweise der Abstand zwischen den Balken gesteuert, `names.arg` ermöglicht die Übergabe von Balkenbezeichnungen und `horiz=T` induziert waagerechte Balken.

2.3.2 Graphische Low-Level-Routinen

Zu einem Plot kann es Ergänzungswünsche geben. Dazu helfen folgende Funktionen:

Funktion	Beispiel	Beschreibung: zeichnet/fügt ein ...
`abline`	`abline(2,3)`	zeichnet Geraden (hier: $y = 2 + 3x$)
	`abline(h=1)`	horizontale Linie $f(x) = 1$
	`abline(v=5)`	vertikale Linie $f(y) = 5$
`lines`	`lines(x,y)`	Polygonzug durch die Punkte $(x \mid y)$
`segments`	`segments(x1,y1,x2,y2)`	Strecken von $(x1 \mid y1)$ nach $(x2 \mid y2)$
`points`	`points(x,y)`	die Punkte $(x \mid y)$
`title`	`title("Entwicklung")`	Überschriften
`text`	`text(x,y,textxy)`	Texte `textxy` an den Stellen $(x \mid y)$
`symbols`	`symbols(x,y,circles=z)`	Kreise um $(x \mid y)$ vom Umfang `z`
`legend`	`legend()`	Legende
`rug`	`rug(x)`	Repräsentation der Punkte durch Ticks

Übersicht 9: Low-Level-Plotfunktionen

Auch hier lautet unser Vorschlag: Studiere die zugehörigen Hilfen und experimentiere! Für interaktive Graphiken suche man nach den Funktionen `locator()` und `identify()`.

2.3.3 Graphische Parameter und Bildspeicherung

`par()` Graphiken werden in R mit Hilfe einer mehr als 65 Einträge enthaltenden Parameterliste verwaltet. Die Werte werden mit der Funktion `par()` abgefragt. Zum Beispiel enthält der Eintrag `usr` (feststellbar über: `par()$usr`) die aktuellen Koordinaten des in der Darstellung gezeigten Weltausschnittes, ein anderer die eingestellte Farbe (`par()$col`). Manche der Parameter können in den graphischen Routinen gesetzt werden; in diesem Fall gilt die Setzung nur für den entsprechenden Aufruf. Viele können aber auch global über die Funktion `par()` verändert werden, jetzt wirkt die Änderung bis zur nächsten globalen Setzung. Der aufmerksame Leser wird schon einen Einsatz der Funktion `par()` identifiziert haben. Mit `par(mfrow=c(zeilen,spalten))` wird die Ausgabefläche schachbrettartig in `zeilen`$\times$`spalten` Felder aufgeteilt, in denen jeweils eine Graphik platziert werden kann. Die Anweisung `par(mfrow=c(1,1))` stellt den Ausgangszustand wieder her. Ein zweiter Weg zur Aufteilung der Ausgabefläche führt über die Funktion `split.screen()`, deren Wirkungsweise hier nicht weiter ausgeführt wird – befrage hierzu die Online-Hilfe.

Speicherung von Bildern. Ein gelungenes Bild will man sicher ausdrucken oder an anderer Stelle einbringen. Zu diesem Zweck lässt sich von einem erstellten Bild mit der Funktion `dev.copy()` eine Kopie anfertigen. Dieser Kopierfunktion sind der gewünschte *device* als Ziel sowie weitere Parameter anzugeben, siehe genaueres über `help(dev.copy)`. Welche Devices zur Verfügung stehen, kann über `help(Devices)` erfragt werden. Soll beispielsweise ein einbindbares Postscript-Dokument erstellt werden, kann die Anweisung lauten:

in 31:
```
dev.copy(postscript,"testbild.eps", width = 4.0,
         height = 3.0,horizontal = FALSE)
dev.off()
```

Es ist zu beachten, dass die Anweisung `dev.off()` unbedingt notwendig ist, damit der Dateierstellungsprozess abgeschlossen werden kann. Durch Öffnung eines passenden graphischen Device können alternativ die Graphikbefehle sofort in dem entsprechenden Format erstellt werden und zum Beispiel in einer JPEG-, PNG- oder PICTEX-Code-Datei münden.

3 Spezialitäten aus dem F-Kurs

Für den interessierten Anfänger wollen wir noch ein paar Ergänzungen zur Verbesserung des konzeptionellen Verständnisses ergänzen.

3.1 Matrizenerstellung.

Neben der Funktion `cbind()` bietet R die Funktionen `rbind()` und `matrix()` zum Bau von Matrizen an. `rbind()` fügt Zeilen (*rows*) zu einer Matrix zusammen. Die Funktion `matrix()` legt die angegebene Daten als Matrix mit einer Zeilen- und Spaltenanzahl ab, die durch das zweite und dritte Argument angegeben wird. Soll die Matrix nicht spalten-, sondern zeilenweise gefüllt werden, muss im Aufruf `byrow=TRUE` übermittelt werden. Übrigens kann die Anzahl der Elemente einer Matrix wie auch die eines Vektors mit Hilfe der Funktion `length()` festgestellt werden. `length()` behandelt eine Matrix also wie einen Vektor.
Hier ein paar Versuche:

in 32:
```
anz.tote <- 0:5
h.anz.tote <- c(119,58,17,4,1,1)
xy <- rbind(anz.tote,h.anz.tote)
```

out 32:
```
               [,1] [,2] [,3] [,4] [,5] [,6]
  anz.tote      0    1    2    3    4    5
  h.anz.tote  119   58   17    4    1    1
```

in 33:
```
matrix(xy,3,6,FALSE)
```

out 33:
```
       [,1] [,2] [,3] [,4] [,5] [,6]
  [1,]    0   58    3    1    0   58
  [2,]  119    2    4    5  119    2
  [3,]    1   17    4    1    1   17
```

in 34:
```
matrix(data=xy,nrow=3,ncol=6,byrow=TRUE)
```

out 34:
```
       [,1] [,2] [,3] [,4] [,5] [,6]
  [1,]    0  119    1   58    2   17
  [2,]    3    4    4    1    5    1
  [3,]    0  119    1   58    2   17
```

in 35:
```
length(xy)
```

out 35:
```
 [1] 12
```

3.2 Indizierung – ein zweiter Blick

Indizierung von Vektoren. Wie schon berichtet lassen sich Teile von Datenobjekten durch Indexzugriffe extrahieren. Für Vektoren gibt es dafür vier verschiedene Arten:

- Definition der gewünschten Elemente durch Angabe ihrer Indizes,
- Festlegung der nicht gewünschten Elemente durch negatvie Angabe deren Indizes,
- Extraktion der Elemente, die eine bestimmte Bedingung erfüllen,
- Auswahl der gewünschten Elemente über Elementnamen.

Es folgt ein künstliches Beispiel.

in 36:
```
vec<-c(119,58,17,4,1,1)
vec[c(1,3,5)]
vec[-(4:6)]
vec[ vec > 20 ]
names(vec)<-c("Anz=0",
 "Anz=1","Anz=2","Anz=3",
 "Anz=4","Anz=5")
vec[ c("Anz=1","Anz=4")]
```

out 36:
```
[1] 119  17   1
[1] 119  58  17
[1] 119  58


Anz=1 Anz=4
   58     1
```

Indizierung von Matrizen. Für Matrizen lassen sich die Regeln zur Indizierung von Vektoren und zur Extraktion von Zeilen wie auch von Spalten einsetzen. Stehen in den Indexklammern eine Auswahlvorschrift für die Zeilen, dann ein Komma und zum Schluss eine Vorschrift für die Spalten, wird die so definierte Untermatrix abgeliefert. Häufig werden Zeilen einer Matrix mit Hilfe der Werte einer bestimmten Spalte ausgewählt.

in 37:
```
xy <- cbind(0:5,c(119,58,17,4,1,1))
xy[1:3,2:1]
```

liefert

out 37:
```
     [,1] [,2]
[1,]  119    0
[2,]   58    1
[3,]   17    2
```

Hier ein zweites Beispiel:

in 38:
```
xy[ xy[,2]>20 , ]
```

Durch die Bedingung werden die ersten beiden Zeilen ausgewählt und wir erhalten:

out 38:
```
     [,1] [,2]
[1,]    0  119
[2,]    1   58
```

Falls bei einem solchem Zugriff nur eine Zeile verbleibt, entsteht automatisch ein Vektor. Gegen den eventuellen Dimensionsverlust hilft der zusätzliche Parameter `drop` im Index:

out 38:
```
xy[ bedingung , , drop=FALSE].
```

Enthalten die Indexklammern kein Komma, wird auch eine Matrix als Vektor interpretiert und als Ergebnis ein Vektor abgeliefert. Durch Indizierung einer Matrix mit einer 2-spaltigen Indexmatrix entsteht ebenfalls ein Vektor; jede 2-elementige Zeile der Indexmatrix wird in diesem Fall als Zeilen-Spalten-Angabe verwendet.

Indizierung von Listen. Die Objekteklasse der Listen wird zwar erst weiter unten vorgestellt, doch sei hier schon eine Bemerkung zu deren Indizierung gemacht. Teillisten erhält man aus Listen wie Teilvektoren aus Vektoren unter Verwendung einfacher Indexklammern. Besitzen die Elemente von Listen Namen, dann bekommen wir den Inhalt eines Listenelementes durch Anfügen eines $-Zeichens und des Elementnamens an den Objektnamen. Alternativ liefert ein Zugriff mittels doppelter eckiger Klammern den Inhalt des Listenelementes, dessen Nummer angegeben ist. Ein Beispiel mag dieses verdeutlichen.

in 39:
```
fb.daten[[4]]
fb.daten$Gewicht
fb.daten[["Groesse"]]
fb.daten[3:2]
```

out 39:
```
[1] 180 175 181 170
[1] 180 175 181 170
[1] 70 65 68 72
  Groesse Gewicht
1      70     180
2      65     175
3      68     181
4      72     170
```

3.3 Häufig verwendete R-Funktionen

Es gibt unter den R-Funktionen einige, die sehr oft eingesetzt werden, bisher hier aber noch keine Erwähnung gefunden haben. Deshalb wird mit diesem Abschnitt eine Liste mit *hot functions* eingefügt.

Funktion	Beispiel	Beschreibung
`%%`	`10 %% 3`	ganzzahlige Division → 3
`floor()`	`floor(1.3)`	rundet ab → 1
`ceiling()`	`ceiling(1.3)`	rundet auf → 2
`round()`	`round(c(1.3,1.7))`	rundet → 1 2
`==`	`x==y`	elementweiser Vergleich
`&`	`l1&l2`	und-Verknüpfung von `l1` und `l2`
`|`	`l1|l2`	oder-Verknüpfung von `l1` und `l2`
`!`	`!l1`	Negation von `l1`
`rowMeans()`	`rowMeans(mat)`	berechnet Zeilenmittel der Matrix `mat`
`colMeans()`	`colMeans(mat)`	berechnet Spaltenmittel der Matrix `mat`
`rowSums()`	`rowSums(mat)`	berechnet Zeilensummen der Matrix `mat`
`colSums()`	`colSums(mat)`	berechnet Spaltensummen der Matrix `mat`
`apply()`	`apply(mat,1,mean)`	berechnet Zeilenmittel der Matrix `mat`
	`apply(mat,2,sum)`	berechnet Spaltensummen der Matrix `mat`
`t()`	`t(mat)`	transponiert Matrix `mat`
`%*%`	`A%*%B`	setzt Matrixmultiplikation um: `AB`
`solve()`	`solve(A)`	invertiert Matrix `A`
	`solve(A,b)`	löst Gleichungssystem `A`·`b`=`x`
`choose()`	`choose(5,3)`	Binomialkoeffizient $\binom{5}{3}$
`rev()`	`rev(x)`	entspricht `x[length(x):1]`
`which()`	`which(3 == (1:10))`	liefert Position von Wahrheitswert `TRUE` → 3
`seq()`	`seq(x)`	liefert Indexvektor zu `x`
	`seq(0,10,length=20)`	liefert 20 äquidistante Zahlen von 0 bis 10
`rep()`	`rep(1:2, 5)`	wiederholt `1:2` genau 5-mal

Übersicht 10: hot R functions

3.4 Konvertierungen und Anpassungen.

Liegen Daten als Zeichenketten vor, werden diese mit Hilfe der Funktion `as.numeric()` in numerische Objekte überführt. Nicht wandelbare Zeichenketten werden durch den Eintrag `NA` – kurz für: Not Available – gekennzeichnet. Die Rückumwandlung erledigt `as.character()`.

in 40:
```
x.name <- c("Anzahl ist 0",as.character(1:5))
as.numeric(x.name)
```

out 40:
```
[1] NA  1  2  3  4  5
Warning message:
NAs introduced by coercion
```

Gelingt eine Umwandlung nicht, wird – wie man sieht – eine Warnung ausgegeben.

Treffen bei Operationen ungleiche Typen oder verschieden große Objekte aufeinander, versucht R oft zweckmäßige Konvertierungen und Anpassungen durchzuführen. Zum Beispiel liefert `c(1,"A")` den Vektor `("1","A")` ab, und in `1+TRUE` wird `TRUE` durch den Wert 1 ersetzt. Werden zwei verschieden lange Vektoren addiert, wird der kürzere durch Wiederholung der Werte geeignet verlängert und dann erst addiert. Zur Warnung wird eine entsprechende Meldung ausgegeben.

Aktion / Kommando	Beschreibung: liefert /generiert ...
`matrix(x,z.z, sp.z)`	erstellt aus `x` eine `z.z`$\times$`sp.z`-Matrix
`as.character(num.vec)`	überführt numerischen in character-Vektor
`as.numeric(char.vec)`	überführt character- in numerischen Vektor

Übersicht 11: einige Funktionen zur Konvertierung

3.5 Konstanten und fehlende Werte

R kennt einige Konstanten. Die Konstanten können wie Zahlen eingesetzt werden.

Konstante	Beschreibung
`pi`	π
`TRUE, T`	Wahrheitswert: wahr
`FALSE, F`	Wahrheitswert: falsch
`NA`	nicht verfügbares Element
`NULL`	Vektor der Länge 0
`NaN`	not a number
`Inf`	∞
`-Inf`	$-\infty$

Übersicht 12: Konstanten

Besonders wichtig für Statistiker ist die Konstante `NA`. `NA` charakterisiert einen fehlenden Wert. Verschiedene Routinen können mit Datenobjekten, die `NA`s enthalten, erfolgreich umgehen. Zum Auffinden von fehlenden Werten hilft die Funktion `is.na()`. Mit dieser lassen sich zum Beispiel die fehlenden Werte von `x` durch `0` ersetzen: `x[is.na(x)]<-0`.

3.6 Fragen des Umgebungsmanagements

Auflisten und Löschen. Während der Arbeit sammeln sich schnell vielerlei Objekte an. Wie behält man da die Übersicht? Die Funktion `ls()` listet ohne weitere Argumente die Objekte der Umgebung auf. Mit `rm()` und `remove()` lassen sich unnütze Objekte wieder entfernen. Eine umfangreichere Auflistung von Objekten erledigt `ls.str()`.

Speicherungsfragen. Die Funktion `dump()` – Beispiel:

in 41: `dump("my.objekt",file="my.file")`

– gestattet es, ein beliebiges Objekt (Beispiel: `my.objekt`) so in einer Datei im Textformat abzulegen, dass es mit dem Aufruf

in 42: `source("my.file")`

wieder eingelesen werden kann. Die komplette Arbeitsumgebung kann durch den Befehl `save.image("my.env")` unter dem Namen `my.env` so abgelegt werden, so dass sie später wieder mit `load("my.env")` geladen werden kann.

Systemparameter. R besitzt Systemparameter, die mit der Funktions `options()` abgefragt und gesetzt werden können. Dazu gehört die Genauigkeit der Zahlenausgabe (`digits`), die Anzahl der ausgebbaren Zeichen pro Zeile (`width`) wie auch das Promptzeichen (`prompt`).

3.7 Datenobjekttypen

Datenobjekte sind in R Datenobjektklassen zugeordnet. Mit der Erzeugung eines Datenobjektes wird diese Klasse festgelegt. Bisher haben wir die speziellen Klassen `matrix` und `data.frame` kennengelernt. Anhand der Klasseneigenschaft lässt sich feststellen, ob und in welcher Form bestimmte Operationen wie Indizierungen oder die Erstellung einer `summary()` durchführbar sind. Objekte einer Klasse besitzen oft spezielle Attribute und zeichnen sich durch eine spezielle Struktur und spezielle Eigenschaften der Daten aus. Die Klasse eines Objektes zeigt die Funktion `class()`.
In der Klasse `matrix` sind die Daten tabellarisch in Zeilen und Spalten angeordnet und alle Bausteine oder Tabellenelemente vom gleichen Typ oder *mode*. In der Klasse `matrix` finden wir Zahlen-Matrizen (`mode`: *numerical*), Matrizen aus Zeichenketten (`mode`: *character*) und Matrizen mit `TRUE-FALSE`-Einträgen (`mode`: *logical*). Der `mode` eines Objektes reflektiert den Speicherbedarf seiner Bausteine und kann mit der Funktion `mode()` festgestellt werden. Wird einem Vektorobjekt in Form eines `dim`-Attribut eine Zeilen- und eine Spaltenanzahl hierzugefügt, ändert sich die Struktur, und es entsteht ein Objekt der Klasse `matrix`.
Diesem Gedanken folgend müßte es auch eine Klasse `vector` geben, in die Vektoren aus Elementen mit gleichem `mode` gehören. Pragmatischerweise liefert für Vektoren die Funktion `class()` jedoch nicht das Ergebnis `vector`, sondern den genauen Typ der Speicherung.
Verschiedenartige Objekte, also Objekte von unterschiedlichen `mode`s, lassen sich in einer Liste (*list*) zusammenfassen – wir erhalten dann ein Objekt der Klasse `list`. Oben haben wir mit `cbind()` gleichlange Vektoren vom gleichen `mode` zu einem Matrixobjekt zusammengefügt. Verschieden lange Zahlenvektoren können nur zu einer Liste zusammengebunden werden. Dieses gilt auch für die Zusammenfassung gleich langer, jedoch verschiedenartiger Vektoren. Da diese Situation bei Erhebungsdaten sehr oft vorkommt, bietet R als Mischtyp die Klasse *data.frame* an. Mit `fb.daten` ist uns ein `data.frame` begegnet. `data.frame`s lassen sich wie Listen oder wie Matrizen indizieren.
Objekte können mehrere Attribute besitzen. Diese dienen zum Beispiel zur Benennung einzelner Datenelemente, der Zeilen- und Spalten von Matrizen oder der Elemente von Listen. Neben Vektoren und Matrizen kennt R kompliziertere Klassen wie Regressionsergebnisse oder Zeitreihen (Klasse `ts`). Zeitreihen verfügen über ein Attribut, über das Informationen über die Zeitpunkte vermerkt werden.

Weitere Objekttypen. Zur Beschreibung statistischer Modelle sei noch die Klasse `formula` hervorgehoben. In dem einfachsten Fall schreiben wir `y~x` und meinen, dass `y` durch `x` linear erklärt werden soll. In praxi können wir – falls `x` und `y` als Vektoren vorliegen – die vielseitige Funktion `plot` in der Form `plot(y~x)` aktivieren und erhalten einen Scatterplot. `lm(y~x)` passt uns an die Daten eine Gerade der Form $y = ax + b$ an.

Ganz kurz erwähnt sei, dass es noch weitere Objekttypen gibt. So sind Funktionen auch Objekte und besitzen den mode `function`. Daneben kennt R Objekte vom mode `expression`, das sind syntaktisch korrekte Ausdrücke, die ausgewertet werden können. In dem Beispiel zum Plot einer kubischen Funktion (`curve(x^3)`), wird das Argument `x^3` als `expression` hantiert.

Objektorientierung Die Wahl der Bezeichnungen *Objekt, Klasse* ist nicht zufälliger Natur, sondern von der objektorientierten Programmierung übernommen. Nach dieser Weltsicht stehen die Objekte und nicht die Funktionen im Vordergrund. Wollen wir zum Beispiel ein Datenobjekt graphisch darstellen oder zusammenfassende Statistiken berechnen, dann soll es – am besten an dem Objekt angeheftet – *Methoden* zum Plotten und zur Berechnung der passenden Statistiken geben, die wir durch einen Ausruf von `plot` oder `summary` aktivieren können. Die Ausführung geschieht nach dem Motto, dass das Objekt schon weiss, wie passend mit ihm umzugehen ist. In R wird diese Sicht implementiert, indem Objekte Klassen zugeordnet werden und für Klassen je nach Notwendigkeit eigene Funktionen – zum Beispiel zum plotten – entworfen worden sind. Zur Aktivierung der speziellen *Methode* muss der Anwender nur die *generische* Funktion – zum Beispiel `plot()` – mit dem Objekt als Argument aufrufen. Über die generische Funktion wird dann die spezielle Funktion oder die Default-Methode ausfindig gemacht und gestartet.

3.8 FAQ

Es heißt: *Stelle nie eine Frage, die in der FAQ-Liste behandelt wird.* Wir wollen deshalb die Empfehlung aussprechen: Wenn irgend etwas unklar ist, blättere zunächst diese Seiten noch einmal durch, werfe dann einen Blick in die FAQ-Liste und befrage erst danach einen Experten. Bevor die Frage aufkommt, sei beigesteuert: Die FAQ-Liste ist zum Beispiel zu finden über die oberste Ebene der Browser-Hilfe oder via Internet:

```
http://cran.r-project.org/doc/FAQ/R-FAQ.html
```

4 Literatur

- P. Dalgaard (2002): Introductory Statistics with R. Springer
- U. Ligges (2004): Programmieren mit R. Springer
- W.N. Venables, B. Ripley (1994,2002): Modern Applied Statistics with S-Plus. Springer
- W.N. Venables, D. M. Smith (2004): An Introduction to R
  `http://cran.at.r-project.org/doc/manuals/R-intro.pdf`
- H.P. Wolf (2004): Funktionsdefinitionen in R — diskutiert am Beispiel: Funktionsschnittpunkte
  `http://www.wiwi.uni-bielefeld.de/~wolf/lehre/ss04/liptex/fnsinR.pdf`
- Verschiedene Hinweise: `http://cran.at.r-project.org/other-docs.html`