Statistische Auswertungen

Standardmethoden und Alternativen
mit ihrer Durchführung in R

Von
Universitätsprofessor
Dr. Rainer Schlittgen

R. Oldenbourg Verlag München Wien

Kapitel 9

Einführung in R

1 Erste Schritte

Besorgen und Installieren

Die freie Software R kann aus dem Internet von einem der CRAN- (Comprehensive R Archive Network-) Server herunter geladen werden. Die Hauptadresse ist:

http://cran.r-project.org/

Es gibt einige andere URLs, die dort durch Anklicken von 'Mirrors' in Erfahrung zu bringen sind.

Am einfachsten läd man sich eine der vorcompilierten binären Versionen herunter. Es gibt solche Distributionen für Linux, Macintosh und Windows. Im folgenden wird davon ausgegangen, dass unter Windows gearbeitet wird. Dann ist die Installation einfach. Die R-Version 1.9, die zur Zeit der Verfassung dieses Manuskriptes aktuell ist, besteht aus einer Datei `rw1090.exe`. Diese ist mit der Maus doppelzuklicken; dann hat man einfach den Anweisungen zu folgen, die auf dem Bildschirm erscheinen.

Starten und Beenden

Wird, wie hier vorausgesetzt, ein Windows-Betriebssystem verwendet, so wurde bei der Installation im Startmenü unter Programme ein Ordner mit der Bezeichnung R angelegt. Darüber kann R gestartet werden. Zudem wird auch im Startmenü ein Icon angelegt, über das R mittels Anklicken gestartet werden kann. Das sind letztlich verschiedene Wege, um die Datei 'rgui.exe' auszuführen.

Nach dem Starten des Programms öffnet sich ein Fenster, die R-Konsole. Hier sind in Blau einige englischsprachige Hinweise und in Rot ein Cursor nach dem Prompt-Zeichen (>) zu sehen.

Wie in den blau gefärbten Hinweisen angegeben, wird R verlassen, indem hinter dem Prompt-Zeichen `q()` eingegeben und durch Betätigen der Enter-Taste abgeschickt wird. Man kann auch auf der Menüleiste (am oberen Bildschirmrand) rechts in der Ecke das Kreuz zum Schließen des R-Fensters anklicken. In beiden Fällen erscheint in einem Dialogfenster die Frage, ob die in der Sitzung erzeugten Daten für ein weiteres Arbeiten in der nächsten Sitzung gespeichert werden sollen. Nur wenn 'yes' angeklickt wird, werden die Daten gespeichert.

Die R-Konsole

Nach dem Starten befindet man sich in der R-Konsole. Ist der Cursor hinter dem Promptzeichen zu sehen, so ist R bereit, Befehle entgegenzunehmen und zu verarbeiten. Ein möglicher Befehl wäre beispielsweise `1+1`. Nach der Eingabe und dem anschließenden Abschicken (Betätigen der Enter-Taste) führt R diesen Befehl aus; das Ergebnis erhält man i.d.R. auf der Konsole angezeigt. Auf dem Bildschirm sieht die vollständige Einheit wie folgt aus:

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

Wird versehentlich ein Befehl abgeschickt, bevor er vollständig ist, so fragt R mittels eines Plus-Zeichens nach der Vervollständigung:

```
>(1+1)*
+ 3
[1] 6
>
```

Es gibt einige Navigationsmöglichkeiten auf der Konsole. So holt man bereits abgeschickte Kommandos über die Taste 'nach oben' zurück. Einen Überblick über die Tastenkombinationen, mit denen man auf der Konsole arbeiten kann, erhält man im Menü 'Help' auf der oberen Bildschirmleiste unter 'Console'.

R kann als Taschenrechner verwendet werden; es lassen sich die gängigen Rechenoperationen durchführen. Die anzuwendenden Befehlskombinationen bedürfen in vielen Fällen keiner Erklärung:

`1 + 2`	Addiert 1 und 2
`2 - 1`	Subtrahiert 1 von 2
`2 * 3`	Multipliziert 2 und 3
`2 / 3`	Dividiert 2 durch 3
`2 ^ 3`	Berechnet 2 hoch 3

Weiterhin gibt es zahlreiche mathematische Funktionen; sie werden entsprechend der üblichen mathematischen Konvention aufgerufen. Beispiele sind:

`sqrt(2)`	Quadratwurzel von 2
`exp(2)`	Exponentialfunktion an der Stelle 2
`log(2)`	Natürlicher Logarithmus von 2

Arbeiten mit einem Editor

Wie aus dem letzten Abschnitt schon deutlich wurde, ist R ein konsolenbasiertes Programm. Befehle sind am Bildschirm einzugeben. Die errechneten Ergebnisse werden, wenn nicht anders verlangt, dort auch wieder ausgegeben. Dann stehen die Befehle nicht mehr unmittelbar zu Verfügung. Gerade wenn umfangreichere Auswertungen durchgeführt werden sollen, ist es aber günstig, einzelne Befehlssequenzen wiederholt ausführen zzu lassen. Dies organisiert man am besten, indem man mit einem Text-Editor arbeitet. Links zu verschiedene Editoren sind auf der CRAN-Seite angegeben. (Zu erreichen über 'Software Other'.) Für Windos sind vor allem 'R-WinEdt' und 'Ultra Edit' zu nennen. 'R-WinEdt' bietet weitergehende Möglichkeiten der Kommunikation mit R. 'Ultra Edit' verfügt zumindest über die Möglichkeit, R-Befehle farbig hervorzuheben.

Werden damit Dateien mit der Dateierweiterung `r` angelegt, so kann der gesamte Inhalt über den Menüpunkt 'File' durch Anklicken von 'Source R code' ausgeführt werden. Einzelne Teile lassen sich über 'Cut & Paste vom Text-Editor in die Konsole bringen. Mit 'R-WinEdt' geht es noch komfortabler. Die Details erfährt man aus der Datei Readme.txt, auf die man durch einen Doppelklick auf 'R-WinEdt' geführt wird.

Variablen

Will man das errechnete Ergebnis eines Befehls zur späteren Weiterverarbeitung im Direktzugriff zur Verfügung haben, kann der auszuführenden Operation ein Name zugewiesen werden. Mit anderen Worten: Es wird eine Variable erzeugt.

Variablen in Computeranwendungen sind von Variablen in der Statistik oder Mathematik zu unterscheiden. So ist eine Variable in R immer ein über den Variablennamen ansprechbarer Wert oder eine Anzahl von Werten, allgemeiner ein Objekt. Was unter einem Variablennamen zwischengespeichert ist, kann sich während der Laufzeit einer R-Sitzung ändern. R unterscheidet zwischen großen und kleinen Buchstaben, Variablennamen sind "case sensitive". Die Variablennamen können aus beliebig vielen Buchstaben und dem Punkt bestehen. Reservierte Namen wie NA, TRUE und FALSE dürfen nicht als Variablennamen verwendet werden.

Variablen werden in R durch ihre erste Wertzuweisung initialisiert. Die Zuweisung eines Wertes an eine Variable geschieht durch den Operator `<-` . (Das Kleiner-Zeichen und das Minus-Zeichen hintereinandergesetzt.) Dabei kann die Variable auch einen noch zu berechnenden Ausdruck zugewiesen bekommen. Nur im einfachsten Fall handelt es sich bei dem Ausdruck in der Zuweisung

```
Variable <- Ausdruck
```

um eine Konstante. Zwei Beispielzuordnungen sind:

`A <- 4.5` A hat den Wert 4.5
`B <- 1+1` B hat den Wert 2.

Hat man bereits mehrere Operationen in R durchgeführt und auch schon einigen davon Namen zugewiesen, möchte man sich oft einen Überblick über die vorhandenen Objekte verschaffen. Dies ist mit dem Befehl `ls()` möglich. Als Ergebnis erhält man eine Liste der vergebenen Namen in alphabetischer Reihenfolge.
Sind etwa die beiden letzten Beispielzuordnungen die einzigen in einer Sitzung gewesen, so ergibt der Befehl `ls()` die Ausgabe

```
[1] "A" "B"
```

Benötigt man eines der Objekte nicht mehr, kann man dieses löschen, indem man den Befehl `rm(name)` eingibt; `name` ist natürlich der Name des zu löschenden Objektes.

Hilfen

Die direkte Hilfe zu einer Funktion wird mit `help(befehl)` aufgerufen. Hierbei muss der in Klammern gesetzte Ausdruck `befehl` durch den Befehl ersetzt werden, über den eine Auskunft gewünscht wird. Alternativ zu `help(...)` kann auch das Kommando `?befehl` verwendet werden.

Es gibt zudem verschiedene menügesteuerte Hilfen. Sie können über den Menüpunkt Help (obere Bildschirmleiste) angefordert werden. In den ersten Punkten des Menüs gibt es einige Informationen zum Arbeiten mit der R-Konsole und zu R. 'R functions (text)...' bildet einen weiteren Zugang zur direkten Hilfe. Hier wird nach Aufruf in einem Dialogfenster nach dem Befehl gefragt, über den Informationen gewünscht sind. Eine alphabetische Liste der Funktionen steht als PDF-Datei unter 'Manuals', 'R Reference Manual' zur Verfügung. Um sie zu nutzen, muss der Acrobat Reader installiert sein. Er ist kostenfrei aus dem Internet herunterzuladen. Einen weiteren Zugang zum Befehlsvorrat von R bietet die Hilfe im html-Format, 'Html help'. Hierbei wird ein Browser geöffnet und eine lokale Datei angezeigt; man muss sich bei Verwendung dieser Hilfe also nicht im Internet anmelden. Dann kann über den Weg 'Packages' (unter Reference), 'base' auf eine alphabetisch geordnete Beschreibung aller verfügbaren Funktionen des Basispaketes zugegriffen werden. Es gibt zwei Vorteile dieser Hilfe. Einer besteht darin, dass man bei Hinweisen auf andere in diesem Zusammenhang wichtige Funktionen direkt durch Anklicken auf diese zugreifen kann, während die Standardhilfe für den "Folgebefehl" wieder neu aufgerufen werden muss. Der andere Vorteil besteht darin, dass bei einer Neuinstallation eines Paketes die Html-Hilfe automatisch aktualisiert wird. Zu den Paketen sei auf den Abschnitt 4 verwiesen.

2 Datentypen und Objekte

Datentypen

Die einfachen Datentypen in R sind

- Zahlen,
- Wahrheitswerte,
- Zeichenketten.

Zahlkonstanten bestehen aus einem optionalen Vorzeichen und beliebig vielen Ziffern, die durch einen Dezimalpunkt in Vor- und Nachkommastellen unterteilt werden. Der angelsächsischen Konvention gemäß wird ausschließlich der Dezimalpunkt anstelle des im deutschen Sprachraum gebräuchlichen Kommas verwendet! Zusätzlich wird von R auch die Exponentialschreibweise unterstützt.

Beispiele für Zahlkonstanten sind 1, 13.5674, .04, -45 , 34e-12, pi. Die Konstante pi ist dabei die bekannte Kreiszahl 3.1415926535897931 (usw.).

Zusätzlich kennt R die Werte +Inf und -Inf, also $\pm\infty$. Dies ergibt sich z.B. immer, wenn eine von null verschiedene Zahl durch null dividiert wird.

Zeichenketten sind beliebige Folgen von Ziffern, Buchstaben und Sonderzeichen. Konstante Zeichenketten erkennt man an den umschließenden doppelten Ausführungszeichen. Zeichenketten bilden neben den Zahlen die andere Form von Konstanten.

Beispiele sind etwa `"Hallo"` und `"Dies ist ein Test."`

Wahrheitswerte sind die Zustände wahr und falsch. Sie werden in R durch die Booleschen Konstanten `TRUE` und `FALSE` (kurz `T` bzw. `F`) repräsentiert. Auch numerische Konstanten tragen Wahrheitswerte, und zwar den Wert TRUE, wenn die Konstante von null verschieden ist und FALSE, wenn sie gleich null ist.

`NA`, "Not Available", ist eine Konstante, die keinem dieser Datentypen zugehört. NA wird immer dann verwendet, wenn in einem Datensatz ein Wert nicht verfügbar ist. Wenn eine mathematische Operation oder Funktion keinen sinnvollen Wert berechnen kann, wird `NaN`, "Not a Number" (keine Zahl), ausgegeben.

Vektoren

R-Vektoren sind die elementaren Datenobjekte in R. Ein R-Vektor ist eine Folge von gleichartigen Objekten. Vektoren können mit einem Buchstaben bzw. Namen angesprochen werden. Wesentlich für die Generierung von Vektoren ist die Funktion `c`, mit der eine (Objekt- bzw.) Werteliste zu einem Vektor zusammengefügt werden kann. Ein Vektor kann wie eine Zahl oder Zeichenkette einer Variablen zugewiesen werden:

```
> A <- c(1,2,3,4)
> A
[1] 1 2 3 4
>
```

In dieser Befehlssequenz hat `A` den Datentyp numerischer Vektor und den Inhalt

1,2,3,4. Die Ausgabe von A die typische Gestalt, dass so viele Elemente eines Vektors hintereinander geschrieben werden, wie es die Dimensionierung des Konsolenfensters zulässt, und jede Zeile mit der Angabe des Index des ersten Elementes beginnt. Auch wenn die Ausgabe als eine Zeile erscheint, ist A ein R-Vektor, der sich in vielen Umständen wie ein Spaltenvektor verhält.

Mit dem binären Sequenzoperator, der durch einen Doppelpunkt angegeben wird, kann ein Vektor von aufeinanderfolgenden Zahlen erzeugt werden. Die Zahlen haben den Abstand 1. Der erste Operand gibt den Startwert, der zweite die Obergrenze der Sequenz an. Falls der zweite Operand kleiner ist als der erste, wird eine absteigende Sequenz erzeugt. Wenn die Obergrenze sich nicht um eine ganze Zahl von der Untergrenze unterscheidet, ist die letzte Zahl der Sequenz kleiner als die Obergrenze:

```
> 1:6
[1] 1 2 3 4 5 6
> 6:-1
[1]  6  5  4  3  2  1  0 -1
> 1.2 : 3
[1] 1.2 2.2
```

Neben den numerischen Vektoren gibt es alphanumerische oder 'character'-Vektoren und logische Vektoren. Deren Komponenten sind jeweils vom entsprechenden Datentyp. Es ist nicht möglich, verschiedene Datentypen in einem Vektor zu mischen; wie man an der zweiten Ausgabe sieht, interpretiert R alle Komponenten als Zeichenketten:

```
> c(TRUE,TRUE,FALSE)
[1]  TRUE  TRUE FALSE
> c(TRUE,1,"regen")
[1] "TRUE"  "1"     "regen"
```

Matrizen und Datensätze

R-Vektoren sind wie erwähnt die grundlegenden Datenobjekte in R. *Matrizen* entstehen aus R-Vektoren, indem diesen das Attribut dim und eventuell das optionale Attribut dimnames angeheftet wird. Aus R-Vektoren können mit den folgenden Funktionen Matrizen erzeugt werden:

`matrix(x,z,s)`	Ordnet einem Vektor die Dimension (z,s) zu; m.a.Worten: matrix transformiert einen Vektor x in eine Matrix vom Typ (z,s).
`dim(x)<-c(z,s)`	Transformiert einen Vektor x in eine Matrix vom Typ (z,s).
`diag(x)`	Erzeugt eine Diagonalmatrix, bei der x auf der Diagonale steht.
`cbind(x,y)`	Setzt Vektoren (und Matrizen) nebeneinander zu einer Matrix zusammen.
`rbind(x,y)`	Setzt Vektoren (und Matrizen) untereinander zu einer Matrix zusammen.

`structure(x,y)`	Gibt das eingegebene Objekt x mit den in y gesetzten Attributen zurück.

Nicht nur die Konvertierung eines R-Vektors in eine Matrix, sondern auch die Änderung der Dimension einer Matrix und die Abfrage der Dimension einer Matrix ist mit dim möglich:

```
> matrix(c(1,2,3,4,5,6),2,3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> structure(1:6, dim = 2:3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Einige weitere Beispiele:

```
> x <- 1:12 ; dim(x) <- c(3,4)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> dim(x)
[1] 3 4
> x<-1:3 ; y<-4:6 ; z<-cbind(x,y) ; z
     x y
[1,] 1 4
[2,] 2 5
[3,] 3 6
```

Mit is.matrix stellt man fest, ob ein Objekt eine Matrix ist. Bezogen auf das zuletzt erstellte Objekt z:

```
> is.matrix(z)
[1] TRUE
```

Es muss darauf hingewiesen werden, dass R-Vektoren keine Zeilen- oder Spaltenvektoren im herkömmlichen Sinne sind. Solche sind einfach Matrizen der Dimension (1,n) bzw. (n,1).

Datensätze bilden den Ausgangspunkt der meisten statistischen Aktivitäten. Sie bestehen aus einem rechteckigen Schema von Werten, so dass die Zeilen jeweils eine Beobachtungseinheit repräsentieren. In den Spalten sind jeweils die zu einer Variablen gehörigen Beobachtungen angeordnet. Die Variablen können fantasievolle Namen tragen wie etwa Ozon, Geschlecht oder Zigarette; sie können aber auch einfach mit V01. V02 usw. bezeichnet sein. Bei Datensätzen sind die Spalten mit den Variablenbezeichnungen verknüpft.

Besteht der Datensatz aus mehr als einer Spalte, also aus mehreren Variablen, so interessiert natürlich die Möglichkeit, die Variablen einzeln anzusprechen. Dies kann durch die Angabe der Variablen geschehen, die durch ein $-Zeichen mit dem Namen des Datensatzes verbunden ist, etwa `dat$var`. Alternativ dazu sind die Variablen des Datensatzes mittels `attach` als einfache Datenvektoren zur Verfügung zu stellen.

Beispiel 9.1
Im November 2001 veröffentlichte DMEuro folgende Ergebnisse aus einer europäischen Studie:

Land	:	Hauptsitz der Firma
Inter	:	Internationalität (gemessen als außereuropäischer Umsatzanteil in Prozent)
Finanz	:	Finanzkraft (gemessen in Eigenkapitalquote in Prozent)
Inno	:	Innovationsfähigkeit (gemessen in durchschnittlichen Patenten der vergangenen Jahre)
Marke	:	Markenstärke (Kennzahl gemäss Fortune-Ranking)
Gesamt	:	Gesamtbewertung in der Studie

Dieser Datensatz sei unter dem Namen `firmen` in der Konsole vorhanden. Beide folgende Befehlszeilen geben den Inhalt der Variablen Land aus:

```
> firmen$Land
> attach(firmen); Land
```

■

Faktoren

In vielen Fällen sind die statistischen Daten in Gruppen unterteilt. Beispiele sind Preiserhebungen nach Bundesländern, soziale Schichten, Gruppen von Probanden, bei denen jeweils die gleiche Therapie angewendet wurde. Diese Unterteilung wird typischer Weise durch eine Indikatorvariable gekennzeichnet. Numerische Indikatorvariablen sollten in R zu *Faktoren* gemacht werden. Dies ist für manche Auswertungen essentiell; ohne die Konvertierung zu einem Faktor kann das Ergebnis falsch werden. (Genauer: man führt dann gar nicht die gewünschte Analyse durch.)

Die Transformation eines numerischen Vektors in einen Faktor geschieht mit dem Befehl `factor`. Den Werten, oder wie man auch in der Varianzanalyse sagt, Stufen des Faktors, können Bezeichnungen oder Labels zugeordnet werden.

```
> gruppe <- c(1,1,1,1,1,2,2,2,3,3,3,3,3)
> fgruppe <- factor(gruppe,levels=1:3)
> levels(fgruppe) <- c("niedrig","mittel","hoch")
```

Zeitreihen

Zeitreihen sind Beobachtungen einer Größe, die in gleichabständigen Zeitpunkten vorgenommen werden. Die Zeitpunkte werden oft einfach mit 1, 2, 3,... durchnummeriert. Speziell in ökonomischen Anwendungen haben aber monatlich, vierteljährlich oder jährlich erhobene Werte eine besondere Bedeutung. Daher werden Zeitreihen in R durch den R-Vektor der Beobachtungswerte sowie durch das Attribut `Tsp` gekennzeichnet. In dem Attribut `Tsp` werden die Parameter Startpunkt, Endpunkt und die Anzahl von Beobachtungen pro Zeitintervall (=frequency) gespeichert.

Mit den folgenden Anweisungen wird beispielsweise eine Zeitreihe von Monatswerten erzeugt:

```
> y <- c(901, 689, 827, 677, 522, 406, 441, 393,
+   387, 582, 578, 666, 830, 752, 785, 664, 467, 438, 421, 412,
+   343, 440, 531, 771, 767, 1141, 896, 532, 447, 420, 376, 330,
+   357, 445, 546, 764, 862, 660, 663, 643, 502, 392, 411, 348,
+   387, 385, 411, 638, 796, 853, 737, 546, 530, 446, 431, 362,
+   387, 430, 425, 679, 821, 785, 727, 612, 478, 429, 405, 379,
+   393, 411, 487, 574)
> fdeaths <- ts(y,start =1974, frequency=12)
```

Die Logik dahinter ist die, dass ein Jahr als eine Zeiteinheit angesehen wird. Dieses Zeitintervall wird in 12 gleiche Teile zerlegt. Damit ist man bei Monatswerten.

Das Ausdrucken der Zeitreihe und das Abfragen ihrer Attribute mit `tsp( )` führt dann zu:

```
> fdeaths
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1974  901  689  827  677  522  406  441  393  387  582  578  666
1975  830  752  785  664  467  438  421  412  343  440  531  771
1976  767 1141  896  532  447  420  376  330  357  445  546  764
1977  862  660  663  643  502  392  411  348  387  385  411  638
1978  796  853  737  546  530  446  431  362  387  430  425  679
1979  821  785  727  612  478  429  405  379  393  411  487  574
> tsp(fdeaths)
[1] 1974.000 1979.917   12.000
```

Entsprechend der oben angegebenen Logik wird der Januar an den Beginn des ersten Teilintervalls gesetzt, also an die Stelle 1974+0. Der Februar steht an der Stelle 1974+1/12, dem Beginn des zweiten Teilabschnittes. Dies geht so weiter; beim Dezember ist man schließlich bei 1979.917= 1974+11/12.

3 Operatoren und Funktionen

In einer Kommandozeile der R-Konsole kann nur jeweils ein Befehl stehen. Will man mehr als einen Befehl unterbringen, so kann dies durch Einfügen eines Semikolons zwischen den Befehlen erreicht werden:

```
> 1+2; 3*21
[1] 3
[1] 63
```

Man kann auch eine Datei anlegen, die von R aus ausgeführt wird. In einer solchen Datei mit R-Befehlen dürfen mehrere Operatoren oder auch Funktionen und Zuweisungen stehen. Diese müssen durch einen Zeilenumbruch voneinander getrennt sein. Das Zeichen # beendet die weitere Abarbeitung der Zeile; es wird zur nächsten Zeile übergegangen. Alles, was nach # steht, wird als Kommentar angesehen.

Das ist der günstigere Weg, wenn man etwas komplexere oder aufwendigere Befehlsgruppen ausgeführt haben möchte. Man erstellt in einem Text-Editor die Befehle und speichert sie in einer Datei mit der Dateierweiterung r ab. Dann wird von der R-Konsole aus unter dem Menü 'File' der Punkt 'Source R-Code' angeklickt. Es öffnet sich ein Fenster, in dem die Datei ausgewählt werden kann. Hat man unter Windows die Eigenschaften des zugehörigen Icons so eingestellt, dass der Pfad der Ausführung ein spezielles Verzeichnis ist, so wird dieses in dem Fenster angezeigt.

Ist im Code etwas zu verbessern, so kann auf die übliche Windows-Art zwischen den beiden Anwendungen gewechselt werden. In R lässt sich mittels des Aufwärtspfeils der Befehl, die Datei auszuführen, zurückholen; so braucht man nur die zugehörige Zeile erneut abzuschicken.

Mathematische Operatoren

R unterstützt die folgenden binären mathematischen Operationen:

`a + b`	Addition
`a - b`	Subtraktion
`a * b`	Multiplikation
`a / b`	Division
`a ^ b`	Potenzierung
`a %% b`	Divisionsrest
`a %/% b`	Ganzzahlige Division (Division ohne Rest)

Über die genannten Operatoren können sowohl zwei Konstanten oder Variablen vom Datentyp Zahl, wie auch eine Zahl und ein Vektor sowie zwei Vektoren miteinander verknüpft werden. Sind beide Operanden Zahlen, so ist das Ergebnis wiederum eine Zahl, ist zumindest einer der beiden Operanden ein Vektor, so ist auch das Ergebnis ein Vektor.

Bei der Verknüpfung eines Vektors mit einer Zahl wird jedes Element des Vektors mit der Zahl verknüpft. Ist die Verknüpfung nicht möglich, so wird in den Resultatsvektor an dieser Stelle `NaN` eingefügt; das Ergebnis ist keine Zahl. Bei einer Division durch Null wird der Wert $\pm$`Inf` ausgegeben:

```
> 1:4 + 2
[1] 3 4 5 6
> (2:-1)^0.5
[1] 1.414214 1.000000 0.000000      NaN
> c(-1:1) /0
[1] -Inf  NaN  Inf
```


Bei Durchführung einer mathematischen Operation auf zwei Vektoren x und y besteht der Ergebnisvektor aus dem Vektor der elementweisen Verknüpfungen. Der Ergebnisvektor ist so lang wie der längere der beiden Eingangsvektoren. Ist einer der Operanden kürzer als der andere, so wird er mehrfach hintereinander gesetzt:

```
> 1:4 + 1:4
[1] 2 4 6 8
> 1:4 * 1:4
[1]  1  4  9 16
> 1:4 + 1:8
[1]  2  4  6  8  6  8 10 12
> 1:4 + 1:5
[1] 2 4 6 8 6
```

Im letzten Fall wird eine Warnung ausgegeben, dass die Längen der Vektoren sich nicht um ein ganzzahliges Vielfaches unterscheiden:

```
Warning message:
longer object length
        is not a multiple of shorter object length in: 1:4 + 1:5
```

Vergleichsoperatoren

Neben den oben aufgeführten mathematischen Operationen kennt R eine Reihe von Vergleichsoperatoren, die jeweils einen booleschen Wert, bzw. einen Vektor von booleschen Werten zurückliefern. Für skalare Werte (Zahlen, boolesche Werte und Zeichenketten) sind diese Operatoren wie folgt definiert:

`a < b`	`TRUE`, wenn a kleiner als b ist.
`a <= b`	`TRUE`, wenn a kleiner oder gleich b ist.
`a = = b`	`TRUE`, wenn a gleich b ist.
`a != b`	`TRUE`, wenn a ungleich b ist
`a >= b`	`TRUE`, wenn a größer oder gleich b ist.
`a > b`	`TRUE`, wenn a größer als b ist.

Bei der Durchführung einer Vergleichsoperation auf einen Vektor und einen skalaren Wert wird jedes Element des Vektors mit diesem Wert verglichen und ein entsprechender Resultatsvektor zurückgeliefert:

```
> 1:8 < 4
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

Der Vergleich zweier Vektoren führt wie bei den arithmetischen Operationen zu einem komponentenweisen Vergleich der Werte beider Vektoren.

Boolesche Operatoren

`!a`	Negation
`a & b`	Und
`a | b`	Oder

Der *Negationsoperator* ! negiert einen booleschen Wert, bzw. alle Werte eines Vektors von booleschen Werten:

```
> !(1:8 < 4)
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Die booleschen Operatoren & (und) und | (oder) führen eine boolesche Verknüpfung auf zwei Skalaren, bzw. einem Skalar und einem Vektor, bzw. zwei Vektoren durch. Hierbei werden die nicht-booleschen Datentypen wie folgt interpretiert:

0 hat den Wert FALSE, alle anderen Zahlen haben den Wert TRUE. Die leere Zeichenkette " " hat den Wert FALSE, alle anderen Zeichenketten haben den Wert TRUE. NA hat immer den Wert FALSE. Die Verknüpfung von Vektor und Skalar, bzw. von zwei Vektoren geschieht wie bei den arithmetischen Operatoren durch paarweise Verknüpfung:

```
> A <- 1:8; B <- (A < 6) & (A > 2); B
[1] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE
```

Funktionen

Wie bei jeder anderen Programmiersprache wird auch der R-Sprachumfang durch eine Reihe von Funktionen ergänzt. Einige wurden schon in der bisherigen Beschreibung angegeben. Allgemein ist eine Funktion in R nach dem folgenden Schema aufgebaut:

```
funktionsname(argument1,argument2, .. weitere Argumente,
              option1 = wert, option2 = wert, . . weitere Optionen )
```

Die Reihenfolge der Argumente ist dabei wesentlich; sie ist in der Hilfe dokumentiert. Dies gilt nicht für die Argumente, die in der Form `argument = wert` bzw. `option = wert` angegeben werden. Wenn die optionalen Argumente nicht mit angeführt sind, werden Voreinstellungen verwendet. Auch diese sind in der Hilfe dokumentiert.
So sind `mean(y)` und `mean(y, rm.na=TRUE)` legitime Aufrufe der Funktion `mean`. `rm.na` ist dabei ein Wahrheitswert, der angibt, ob fehlende Werte entfernt werden sollen, bevor das arithmetische Mittel berechnet wird. Weitergehend gibt es noch die Option `trim`. Mit dem Befehl `mean(y,trim=0.1,rm.na=TRUE)` wird das getrimmte arithmetische Mittel der Werte aus y berechnet, wobei die 10% der kleinsten und 10% der größten Werte entfernt werden.

Indizierung

Man kann auf Teile oder einzelne Elemente eines Vektor oder einer Matrix zugreifen, um sie auszuwählen bzw. zu entfernen. Dies geschieht durch die *Indizierung*, die Angabe der Indizes innerhalb eckiger Klammern. Da in der mathematischen und statistischen Literatur die Indizes vornehmlich als Subskripte notiert werden, spricht man hier auch von *Subscription*. Bei Vektoren ergeben sich folgende Möglichkeiten:

Die Indizierung erfolgt gemäß `Vektor[a]`. Bei a kann es sich um einen Vektor mit positiven oder negativen Komponenten handeln, oder um einen Vektor von booleschen Werten.

Ist a ist eine positive Zahl, so wird bei einem Vektor das an der entsprechenden Stelle stehende Element ausgewählt. Das erste Element eines Vektors hat dabei den Index 1. Bei nicht ganzzahligen Indizes wird abgerundet. Ist a ein Vektor von positiven Werten, so werden diese als Liste von Indizes interpretiert und die entsprechenden Elemente werden ausgewählt.

Da das Komma für die Trennung der Dimensionen bei Matrizen dient, darf es nicht zur Trennung von Indizes eines Vektors verwendet werden. Ist der angegebene Index größer als die Länge des Vektors, so ist das Ergebnis der Operation NA:

```
> A <- 3:8
> A[2]
[1] 4
> A[9]
[1] NA
> A[2:4]
[1] 4 5 6
> A[1,3]
Error in A[1, 3] : incorrect number of dimensions
> A[c(2:4,9)]
[1]  4  5  6 NA
```

Wird ein negativer Index angegeben, so ist das Resultat ein Vektor, aus dem das an dieser Stelle befindliche Element entfernt wurde. Ist der Betrag des angegebenen negativen Index größer als die Länge des Vektors, so gibt R eine Fehlermeldung aus. Es gilt die Einschränkung, dass alle Indizes entweder positiv oder negativ sein müssen:

```
> A <- 3:8
> A[-2]
[1] 3 5 6 7 8
> A[-9]
Error: subscript out of bounds
> A[-2:-4]
[1] 3 7 8

> A[c(-1,-3)]
[1] 4 6 7 8
> A[c(-1,3)]
Error: only 0's may mix with negative subscripts
```

Anstelle von numerischen Indizes können auch Vektoren von booleschen Werten verwendet werden. Hierbei werden aus dem indizierten Vektor nur die Elemente ausgewählt, deren Pendant im Indexvektor den Wert TRUE hat. Ist der Indexvektor kürzer als der indizierte Vektor, so findet wie schon bei den arithmetischen Opera-

toren ein 'wrap around' statt:

```
> A <- 1:6
> A[c(T,F,F,T,F,T)]
[1] 1 4 6
> A[c(T,F)]
[1] 1 3 5
```

Weiterhin ist es erlaubt, als Index einen beliebigen Ausdruck zu verwenden, der in einem Vektor von booleschen Werten resultiert:

```
> A <- 1:6; B<-A<4; A[B]
[1] 1 2 3
```

Bei Matrizen gibt es eine Zeilen- und eine Spaltendimension. Es sind also zwei Angaben zu machen; diese werden durch ein Komma getrennt. Dann wird auf die jeweiligen Zeilen bzw. Spalten zugegriffen. Das Komma muss angeführt werden. Werden nur Indizes vor dem Komma spezifiziert, so betrifft die Auswahl die ganze(n) vor dem Komma spezifizierte(n) Zeile(n). Entsprechendes gilt für die Angabe von Indizes nach dem Komma bzgl. der Auswahl die ganze(n) nach dem Komma spezifizierte(n) Spalte(n).

```
Matrix[a,], Matrix[,b], Matrix[a,b]
```

Bei a und b kann es sich um einen Vektor mit positiven oder negativen Komponenten oder um einen Vektor von booleschen Werten handeln. Auch hier dürfen innerhalb eines Vektors von Indizes positive und negative Werte nicht gemischt werden. Ansonsten gelten die für Vektoren angegebenen Regeln.

Matrix-Operationen

Matrizen können wie üblich mittels * mit Skalaren multipliziert werden. Das eigentliche Matrizenprodukt ist %*% . Die Transponierung einer Matrix A wird durch t(A) geleistet. Die Inverse einer regulären Matrix erhält man mit solve:

```
> A<-matrix(c(1:6),2,3); B<-t(A); B
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> B %*% A
     [,1] [,2] [,3]
[1,]    5   11   17
[2,]   11   25   39
[3,]   17   39   61
> solve(diag(c(1,1,1))+ (B %*% A))
           [,1]       [,2]       [,3]
[1,]  0.7844828 -0.1637931 -0.1120690
[2,] -0.1637931  0.7155172 -0.4051724
[3,] -0.1120690 -0.4051724  0.3017241
```


Mittels apply lässt sich eine Funktion, die eigentlich für Vektoren gedacht ist, auf allen Zeilen oder Spalten einer Matrix simultan anwenden:

```
> A<-matrix(c(1:6),2,3)
> apply(A,1,sum)
[1]  9 12
> apply(A,2,sum)
[1]  3  7 11
```

apply(A,1,sum) ergibt also die Summe über die Zeilen, apply(A,2,sum) die über die Spalten.

Dass Matrizenoperationen zur Verfügung stehen, führt dazu, dass man möglichst viele der bei einer Berechnung auszuführenden Operationen als Matrix-Operationen schreibt.

Beispiel 9.2
Die folgenden Befehle erzeugen eine Matrix X, bestimmen die Randsummen und den Wert der χ^2-Teststatistik für den Test auf Unabhängigkeit:

```
> X<-1:24 ;  dim(X)<-c(4,6) ; X
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
> r<-apply(X,1,sum) ; u<-apply(X,2,sum)
> E<-r %*% t(u)/sum(X)
> XE<-(X-E)^2
> XE<-scale(XE,center=FALSE,scale=u)
> XE<-scale(t(XE),center=FALSE,scale=r)
> sum(XE)*sum(X)
[1] 1.705138
```

Die Funktion scale skaliert dabei eine Matrix spaltenweise. Wird die Option center auf FALSE gesetzt, so wird sie nicht zentriert. Die Option scale=u verlangt, dass die Matrix spaltenweise durch das entsprechende Element des Vektors u dividiert wird.

Zu dem gleichen Ergebnis gelangt man auch folgendermaßen; dabei werden die ersten beiden Zeilen des letzten Code-Blocks vorausgesetzt:

```
> D1<-solve(diag(u)) # (6,6)-Diagonalmatrix mit den Elementen
                     # 1/Zeilensumme
> D2<-solve(diag(r)) # (4,4)-Diagonalmatrix mit den Elementen
                     # 1/Spaltensumme
> X1<-((X-E)^2)%*%D1 # Die Abweichungsquadrate werden durch die
                     # Zeilensummen dividiert.
> X2<-D2%*%X1        # Zusätzlich werden sie durch die Spalten-
                     # summen dividiert.
> sum(X2)*sum(X)     # Aufaddieren aller Matrixelemente und
```

```
                    # Multiplikation mit dem Stichprobenumfang.
[1] 1.705138
```

■

Speziell ist die Verknüpfung von numerischen R-Vektoren mit Matrizen. Wird ein R-Vektor x mittels * mit einer Matrix a multipliziert, so geschieht das elementweise, und zwar so, dass die erste Zeile der Matrix mit dem ersten Element des R-Vektors durchmultipliziert wird, die zweite Zeile mit dem zweiten Element usw. Das geht nicht, wenn x ein 'normaler' Vektor ist, also eine einzeilige oder einspaltige Matrix:

```
> A<-matrix(c(1:6),2,3)
> c(4,5)*A
     [,1] [,2] [,3]
[1,]    4   12   20
[2,]   10   20   30
```

Will man diese Operation mit einer einspaltigen oder einzeiligen Matrix durchführen, so ist sie zuerst in einen R-Vektor zu verwandeln. Das geschieht mit as.vector:

```
> A<-matrix(c(1:6),2,3); x<-as.vector(A[,1]); x*A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    4    8   12
```

Für etliche Anwendungen ist es günstig, dass das äußere Produkt %o% zur Verfügung steht. Bei zwei Vektoren wird dadurch eine Matrix der komponentenweisen Produkte erzeugt:

```
> x<-c(1:3)
> y<-c(1,3,5,7)
> x%o%y
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    6   10   14
[3,]    3    9   15   21
```

Beispiel 9.3
Mit StudZuf liegt ein Datensatz mit zwei kategorialen Variablen Zufr und Stud vor. Die erste kennzeichnet die Zufriedenheit mit der eigenen Situation aufgrund der zur Verfügung stehenden finanziellen Mittel, die zweite, ob es sich um Vollzeit-, Teilzeit-oder Nebenherstudenten handelt. Um die beiden Variablen auf Unabhängigkeit zu testen, wird mit table eine Kontingenztafel erstellt. Dann werden ebenfalls mit table die univariaten Randverteilungen bestimmt. Das äußere Produkt ergibt alle Produkte der Randhäufigkeiten in der Struktur, die der ursprünglichen Kontingenztabelle entspricht. Damit ist die Bestimmung der Summanden der χ^2-Statistik leicht:

```
t<-table(StudZuf)
f1<-table(Zufr)
f2<-table(Stud)
```



```
e<-f1%o%f2/sum(t)
x2<-sum((t-e)^2/e)
paste("Wert der Teststatistik: ",x2)
paste("P-Wert : ",1-pchisq(x2,8))
```

■

4 Bibliotheken und Programmierung

Bibliotheken

Es gibt neben den Funktionen in dem Basispaket viele zusätzliche Funktionen in weiteren Paketen. Mit dem Aufruf library() erhält man einen Überblick über die aktuell implementierten Pakete.

Zugriff auf die Funktionen in einem Paket bekommt man mit dem Befehl library(Paket); für Paket ist natürlich das spezielle gewünschte Paket anzugeben, also etwa library(stepfun). In der Html-Hilfe sind für die installierten Packages Listen der jeweils implementierten Funktionen mit Erklärungen zu Aufruf und zu den Ergebnissen zu finden.

Der einfachste Weg, um zusätzlich zu den bei der Installation von R mitinstallierten Paketen weitere zu installieren, besteht in dem Öffnen des Menüpunktes 'Packages' von der R-Konsole aus. Hier kann eine der beiden Möglichkeiten 'Install Package(s) from CRAN...' oder 'Install Package(s) from local zip files...'. ausgewählt werden. Hat man z.B. das interessierende Paket, etwa leaps.zip auf der Festplatte im Verzeichnis c:TEMP gespeichert, so ist dies in dem Fenster, das sich beim Anklicken geöffnet hat, auszuwählen. Dann geschieht bei den entsprechend vorbereiteten Paketen alles automatisch. Auch die HTML-Hilfe wird aktualisiert. Die Bibliothek muss nur noch mit library(leaps) geladen werden, um mit den dort enthaltenen Funktionen arbeiten zu können.

Für die andere Variante muss eine Internetverbindung bestehen. Ansonsten sei auf die Hilfe verwiesen, die mit help(install.packages) anzufordern ist. Vorab sollte man die diesbezüglichen Ausführungen in der Datei rw-FAQ lesen. Die Datei befindet sich im Wurzelverzeichnis der R-Installation.

Will man für eine Weile stets mit einer oder mehreren Bibliotheken arbeiten, so kann in der Datei Rprofile im Verzeichnis ../rw1090/etc/ ein entsprechender Eintrag vorgenommen werden. Dort können auch Optionen ausgewählt bzw. gesetzt werden.

Kontroll-Strukturen

Das Schreiben von eigenen Funktionen macht vor allem Sinn, wenn man häufiger die gleiche Anweisungsfolge bei eventuell unterschiedlichen Parameterkonstellationen ausführen will. Dann benötigt man Kontroll-Strukturen, die den Fluss eines

Programms regeln. R kennt zwei Arten von *Kontroll-Strukturen*, nämlich bedingte Anweisungen und Schleifen.

Bedingte Anweisungen werden über die `if`-Anweisung realisiert:

```
if (Bedingung) { Anweisungen }
```

Die in den geschweiften Klammern enthaltenen Anweisungen werden nur ausgeführt, falls die angegebene Bedingung wahr ist, d.h. den Wert TRUE besitzt. Optional kann die `if`-Anweisung auch einen `else`-Zweig enthalten, der ausgeführt wird, wenn die Bedingung falsch ist:

```
if (Bedingung) { Anweisungen } else { Anweisungen }
```

Ein einfaches Beispiel für eine solche `if`-Anweisung ist etwa:

```
> if (x==10) { k <- x+1 } else { k <- x-1 }
```

Die einfachste Form der *Schleife* in R ist die `while`-Anweisung. Auch hier folgt in einer in runde Klammern gesetzten Bedingung die auszuführende(n) Operation(en) in geschweiften Klammern. Diese in den geschweiften Klammern angegebene Anweisung (oder Menge von Anweisungen) wird solange ausgeführt, wie die Bedingung wahr ist.

```
while (Bedingung) { Anweisungen }
```

Die folgende Sequenz verdeutlicht dies beispielhaft:

```
> i <- 1; while (i < 4) { i <- i + 1; print(i) }
[1] 2
[1] 3
[1] 4
```

Diese Schleife wird also genau dreimal durchlaufen. Im letzten Durchlauf bekommt i den Wert 4; daher wird die weitere Bearbeitung gestoppt.

Eine weitere Form der Schleife in R ist die `for`-*Schleife*. Die Syntax der for-Anweisung ist sehr elegant:

```
for (Variable in Vektor) { Anweisungen }
```

Die Laufvariable `Variable` nimmt nacheinander die Werte von `Vektor` an, d.h. beim ersten Schleifendurchlauf hat sie den Wert des ersten Vektorelements, beim zweiten den Wert des zweiten und so weiter:

```
> for(i in 1:5)  print(1:i)
[1] 1
[1] 1 2
[1] 1 2 3
[1] 1 2 3 4
[1] 1 2 3 4 5
```

Besteht die Anweisung nur aus einem einzigen Befehl, so kann wie in diesem Beispiel auf die geschweiften Klammern verzichtet werden.

Die Anweisung `break` führt zum unmittelbaren Verlassen einer Schleife, Die Anweisung `next` zum Beenden des aktuellen Schleifendurchlaufs.

Eigene Funktionen

Um eine Funktion zu schreiben, sind die Befehle mit einem einfachen Editor als Text (ohne jede Formatierung) zu schreiben. Hierfür gibt es die Möglichkeit, von R aus einen Editor zu starten. Dies geschieht mit der Funktion `edit` oder mit `fix`:

```
fix(Funktionsname)
```

Über eine Option kann ein beliebiger Text-Editor ausgewählt werden. In dem Editor wird dann die Funktion geschrieben und als 'Source-Code' in einer externen Datei mit der Dateierweiterung `r` abgespeichert. Die externe Datei kann auch mehrere selbstdefinierte Funktionen enthalten.

Eine Funktion wird nun mittels

```
MeineFunktion <- function(Argumente) { Anweisungen }
```

definiert. Diese Funktion steht nicht automatisch zur Verfügung. Sie muss vielmehr erst mit dem Befehl

```
> source("c:/myr/myfunc.r")
```

initialisiert werden. Hierbei ist angenommen, dass die Funktion MeineFunktion in der Datei `myfunc.r` abgespeichert wurde.

Beispiel 9.4 (Standardfehler des Median)

Eine Funktion zur Bestimmung des Standardfehlers des Median mittels Bootstrap kann folgendermaßen aussehen:

```
sdmedian <- function(x,B)
 {
  n <- length(x)
  m <- rep(0,B)
  for(b in 1:B)
   {
    x1 <- sample(x,n,replace=TRUE)
    m[b] <- median(x1)
   }
  sd(m)
 }
```

Diese Funktion kann nun in der Datei `sdmed.r` abgespeichert werden. Mit

```
> source("c:/d/myr/sdmed.r")
```

steht sie dann für die aktuelle Sitzung zur Verfügung.
Es wird nun mittels `y<-rnorm(49)` ein Vektor von 49 standardnormalverteilten Zufallszahlen erzeugt und die Funktion auf diesen Vektor angewendet:

```
> source("c:/myr/sdmed.r")
> y <- rnorm(49)
> sdmedian(y,1000)
[1] 0.1521678
```

Für den Standardfehler des arithmetischen Mittels erhält man bei diesem Stichpro-

benumfang übrigens mit $\mathrm{Var}(\bar{Y}) = \sigma^2/n$ den Wert 1/7=0.143. ∎

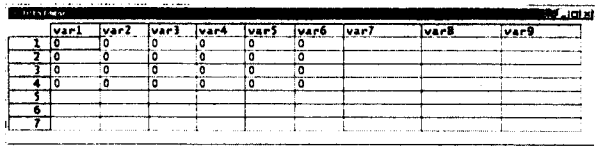
5 Einlesen und Exportieren von Daten

Datensätze liegen in den unterschiedlichsten Formaten vor. Viele externe Formate können in R eingelesen werden; das ist allerdings nicht für alle Formate möglich. Hier sollen die wichtigsten Möglichkeiten beschrieben werden. Eine detaillierte Beschreibung findet man in dem R-Hilfe-Menü unter 'Manuals', 'R-Data Import/Export'.

Will man an der Konsole einen kleinen Datensatz eingeben, so ist es am einfachsten, eine Matrix von Nullen zu erzeugen und diese aufzufüllen. Mit dem Aufruf von `data.entry(mat)` wird dann ein Datenblatt geöffnet, in das die gewünschten Werte eingetragen werden können:

```
A<-matrix(0,4,6)
data.entry(A)
```

Das geöffnete Datenblatt hat dann die in der Abbildung wiedergegebene Gestalt.



Aus einer externen ASCII-Datei können Daten mit `scan` in einen Vektor eingelesen werden:

```
> scan("c:\\daten\\koerpertemp.txt")
Read 65 items
 [1] 35.72 35.94 36.06 36.11 36.17 36.17 36.17 36.22 36.28 36.33
[11] 36.33 36.33 36.33 36.39 36.39 36.44 36.44 36.44 36.50 36.56
[21] 36.56 36.56 36.56 36.61 36.61 36.67 36.67 36.67 36.67 36.67
[31] 36.67 36.72 36.72 36.78 36.78 36.78 36.78 36.83 36.83 36.89
[41] 36.89 36.89 36.89 36.94 36.94 37.00 37.00 37.00 37.00 37.00
[51] 37.00 37.06 37.06 37.11 37.11 37.11 37.17 37.22 37.22 37.22
[61] 37.28 37.33 37.39 37.44 37.50
```

Zu beachten ist, dass die Backslashes innerhalb von Anführungsstrichen doppelt eingegeben werden müssen. Auch unter Windows kann die Pfadangabe mit einem einfachen 'Slash' anstelle eines doppelten Backslashes erfolgen. Somit ist

```
> scan("c:/daten/koerpertemp.txt")
```

zu dem obigen Aufruf äquivalent. Da dies einfacher ist, wird im Folgenden diese Möglichkeit verwendet.

Mit `scan` werden die Werte hintereinander in einen Vektor geschrieben. Sind in einer ASCII-Datei Daten verschiedener Variablen gespeichert, so können sie gleich in der

Form eines Datensatz eingelesen werden, wenn die ASCII-Datei schon die Struktur hat, die der Datensatz dann bekommen soll. Das bedeutet speziell, dass die zu einer Variablen gehörigen Werte jeweils spaltenweise angeordnet sind; die Werte einer Zeile gehören zu einer Beobachtungseinheit. Die Angaben sind durch Leerzeichen voneinander getrennt. In der ersten Zeile dieser Datei seien die Variablennamen eingetragen (durch Leerzeichen separiert). Dann erfolgt das Einlesen mittels der Funktion `read.table`:

```
> firmen <- read.table("c:/daten/firmen2001.dat",header=TRUE)
```

Sind in der ersten Zeile der Datei die Namen der Variablen nicht angegeben, so lautet die Option `header=FALSE`.

Es gibt zahlreiche Datensätze, die als Beispielmaterial schon in dem Basismodul mitgeliefert werden; weitere sind in fast allen Packages enthalten. Diese können mit dem Befehl `data` eingelesen werden:

```
> data(cars)
> summary(cars)
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  
> attach(cars)
> summary(speed)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    4.0    12.0    15.0    15.4    19.0    25.0 
```

Externe Repräsentationen von R-Objekten werden mit `save` abgespeichert. Mit der Option `ascii = TRUE` wird ein Datensatz in eine ASCII-Datei geschrieben. Die Voreinstellung ist `ascii = FALSE`; dies ergibt eine kompaktere Form der Speicherung. Solche Dateien können mit dem Befehl `load` wieder geladen werden.

In dem folgenden Beispiel wird zwischendurch der Datensatz `Y` mittels `rm(Y)` aus dem aktiven Arbeitsbereich entfernt; `rm` ist günstig, wenn man zwischendurch etwas aufräumen will.

```
> y<-rnorm(20)
> y<-matrix(y,4,5)
> y<-data.frame(y)
> y
          X1         X2         X3         X4         X5
1 -1.0331633  0.9611738 -1.1029188  0.3391295 -0.7100169
2 -0.2150846  0.9401090  0.1533617 -0.3523658  0.1325400
3  1.3525974 -1.1968621  1.0002396 -2.1669665  0.4348876
4  0.3616053  1.3718655 -1.7790348  1.3811907  1.4269337
```

```
> save(y,file="c:/d/myr/y.RData" )
> rm(y)
> y
Error: Object "y" not found
> load("c:/d/myr/y.RData")
> y
          X1         X2         X3         X4         X5
1 -1.0331633  0.9611738 -1.1029188  0.3391295 -0.7100169
2 -0.2150846  0.9401090  0.1533617 -0.3523658  0.1325400
3  1.3525974 -1.1968621  1.0002396 -2.1669665  0.4348876
4  0.3616053  1.3718655 -1.7790348  1.3811907  1.4269337
```

6 Grafik

Es gibt etliche Typen von Grafiken, die auf einfache Weise Datensätze bzw. Vektoren oder Matrizen darstellen. Die Grafikfunktionen produzieren dabei Grafik-Typen, die von dem jeweiligen Objekt-Typ der Eingabe abhängen. Diese Grafikfunktionen gehören zu den sogenannten High-Level-Grafikfunktionen. Der Aufruf einer dieser Funktionen führt dazu, dass eine bereits vorhandene Grafik durch eine neue ersetzt wird. Um in einem Grafik-Fenster mehr als eine Grafik unterzubringen, kann der Befehl `par` eingesetzt werden. So wird durch `par(mfrow=c(1,2))` die Grafik senkrecht geteilt.

Univariate Daten

Es stehen die im folgenden aufgeführten Grafik-Typen zur Darstellung univariater Daten bzw. Vektoren zur Verfügung.

High-Level-Grafikfunktionen für univariate Daten

`barplot`	Säulendiagramm.
`boxplot`	Box-Plot.
`dotchart`	Erzeugt ein Cleveland-Dot-Chart.
`hist`	Histogramm unter Verwendung von `barplot`.
`pie`	Kreisdiagramm.
`plot`	Je nach Argument ein Indexplot, ein Stabdiagramm oder eine empirische Verteilungsfunktion.
`qqnorm`	Quantildiagramm für die Normalverteilung.
`qqplot`	Quantil-Quantil-Diagramm.
`stripchart`	Eindimensionales Streudiagramm

Das *Säulendiagramm* `barplot` verlangt als Eingabe einen Vektor von Anteilen oder Anzahlen. Die Säulen werden bei der Voreinstellung vertikal gezeichnet und unterschiedlich eingefärbt. Bezeichnungen muss man mittels eines Vektors `names.arg` selber hinzufügen:

```
> barplot(c(5,3,7),names.arg=c("A","B","C"))
```

Die Funktion `boxplot` zur Erstellung eines *Box-and-Whisker-Plots* erwartet als Eingabe einen Datensatz oder eine Liste von Variablen bzw. Vektoren. Bei einer Matrix werden alle Elemente der Matrix in einem einzigen Boxplot dargestellt. Variablennamen erscheinen beim Boxplot automatisch als Labels unterhalb der x-Achse. Gibt es keine explizite Angabe von Variablennamen, etwa weil `X` eine zum Datensatz konvertierte Matrix ist, so bekommen die Variablen die voreingestellten Werte V1, V2, ...

Sei zum Beispiel `dat` ein Datensatz mit zwei Variablen `V1` und `V2`. Dann erzeugen die beiden folgenden Aufrufe die gleiche Grafik:

```
> boxplot(dat)
> boxplot(V1,V2)
```

Für einen *Cleveland-Dot-Chart* ist die Eingabe `x` ein Vektor numerischer Werte (NAs sind zugelassen). Es können auch Labels definiert werden. Sie erscheinen dann am linken Rand des Dot-Charts:

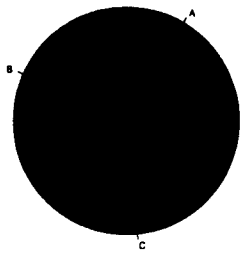
```
> dotchart(c(1:5),labels=c("A","B","C","D","E"))
```

`hist` bietet ein einfaches *Histogramm*. Hier wird neben dem Datenvektor die Angabe der Anzahl der Klassen erwartet. Erst durch Setzen von `include.lowest= TRUE` werden Werte berücksichtigt, die mit der untersten Klassengrenze übereinstimmen. Es kann auch ein Vektor von Klassengrenzen angegeben werden. Dann ist allerdings der gesamte Wertebereich abzudecken; die unterste Klassengrenze muss kleiner oder gleich dem kleinsten Wert in `x` sein und die oberste größer oder gleich dem größten x-Wert. Beispiele sind:

```
> hist(runif(500), breaks=10)
> hist(runif(500), breaks=10, include.lowest=TRUE)
> hist(runif(500), breaks=c(0,0.1,0.25,0.5,0.75,1))
```

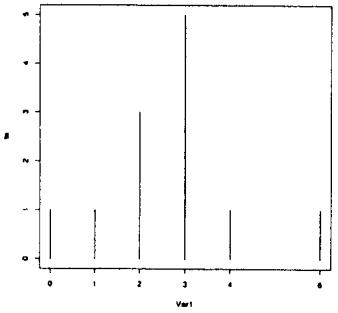
`pie` stellt einen Eingabevektor in einem *Kreis-* oder *Tortendiagramm* dar. Dazu werden die Werte in Anteile der Gesamtsumme aller Werte transformiert und bekommen einen entsprechendes Segment des Kreises zugeordnet. Negative Werte sind folglich nicht zugelassen. Für die Darstellung werden die Kreissegmente der Reihe nach durchnummeriert. Mittels des zusätzlichen Arguments `labels` können die Segmente auch mit Namen etc. versehen werden. Zusätzlich können die Kreissegmente mit `col` eingefärbt werden.

```
> pie(c(1:3),labels=c("A","B","C"),col=c("red","blue","green"))
```



Die Funktion `plot` ist sehr mächtig. Im folgenden Abschnitt wird weiter darauf eingegangen. Wird als Argument einfach ein Variablenname eingegeben, so produziert `plot` einen *Indexplot*. Dabei werden die einzelnen Werte einfach hintereinander als Punkte dargestellt. Mit `plot` lässt sich auch leicht ein *Stabdiagramm* darstellen. Dazu produziert man mit der Funktion `table` aus den Werten der Variablen eine einfache Häufigkeitstabelle und lässt sich diese mit `plot` darstellen.

```
> Var1<-c(0,2,3,4,1,2,3,3,3,2,3,6); ta<-table(Var1); plot(ta)
```



Weiter dient `plot` zur Darstellung der empirischen Verteilungsfunktion. Ein Weg dazu geht über das R-Package 'stepfun'. Ist dieses geladen (vgl. den Abschnitt 4), so wird mit der dort existierenden Funktion `ecdf` die empirische Verteilungsfunktion entsprechend folgender Befehle angefordert:

```
> a <- c(1,2.2,3.5,4)
> e <- ecdf(a)
> plot(e, do.points=FALSE, verticals=TRUE)
```

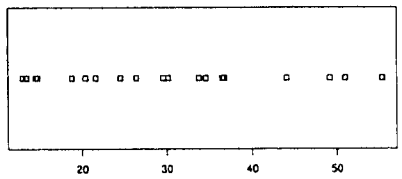
`qqplot` bietet ein empirisches *QQ-Diagramm*. Auch diese Funktion ist einfach aufzurufen. Als Argumente werden zwei Vektoren angegeben. Sie müssen nicht gleich lang sein. Es werden die durch den kürzeren Vektor bestimmten Anteile als p-Werte für die Quantile genommen. Zu beachten ist hier, dass in R Quantile stets über lineare Interpolation ermittelt werden:


```
> x <- rnorm(10,mean=0,sd=1); y <- rnorm(15,mean=0.5,sd=1)
> qqplot(x,y)
```

Mit `stripchart` kann man *eindimensionale Streudiagramme* erstellen. Dabei können auch mehrere Variablen in einen Chart eingezeichnet werden. Das stellt eine gute Alternative zu Boxplots dar, wenn die Umfänge der Datensätze kleiner sind. So ergibt die Befehlszeile

```
> stripchart(firmen$Finanz)
```

die folgende Grafik:



Bivariate und höherdimensionale Daten

Für die Darstellung bivariater und höherdimensionaler Daten gibt es in R die folgenden Funktionen:

High-Level-Grafikfunktionen für bivariate und höherdimensionale Daten

`contour`	Contourdiagramm.
`coplot`	Diese Funktion erzeugt zwei Varianten von bedingten Streudiagrammen.
`image`	High density image plot functions.
`matplot`	Darstellung der Spalten einer Matrix gegen die Spalten einer anderen.
`pairs`	Streudiagramm-Matrix.
`persp`	Dreidimensionale perspektivische Darstellung.
`plot`	Streudiagramm, Linienzüge etc.
`stars`	Sternendiagramm multivariater Daten.

Hier soll nur auf die wichtigsten dieser High-Level-Grafikfunktionen eingegangen werden.

`plot` ist die Universalfunktion zur Darstellung bivariater Daten als *Streudiagramm* und für *Linienzüge*. Der Aufruf ist denkbar einfach: Sind `x` und `y` gleichlange Vektoren, so wird mit

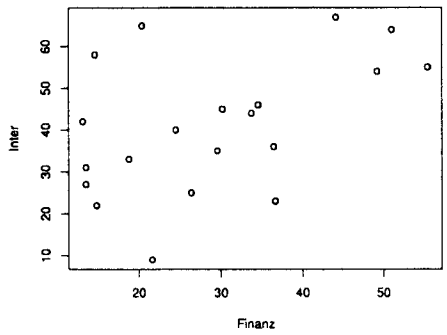
```
> plot(x,y,type="p")
```

ein Streudiagramm der (x,y)-Werte erstellt. Die Angabe von `type="p"` verlangt dabei die Darstellung als Punkte. Möchte man einen Linienzug haben, so ist `type="l"` anzugeben.

Konkret wird der Datensatz `firmen` betracht, vgl. Kapitel 1. Für die Variablen Finanzkraft und Internationalität wird ein *Streudiagramm* erstellt:

```
> firmen<-read.table("c:/daten/firmen.dat",header=T)
> attach(firmen)
> plot(Finanz,Inter,type="p")
```

Das Ergebnis ist in der folgenden Grafik wiedergegeben.



Ein wichtiger Aspekt ist die Achsenbezeichnung. Automatisch werden die Variablennamen gewählt, die für die Platzhalter x und y eingesetzt werden. Möchte man das ändern, so ist `xlab` als Parameter für die x-Achse zu definieren; entsprechend steht `ylab` für die y-Achse.

Die vollen Möglichkeiten von `plot` sind ausufernd. Einige Optionen, die analog zu `type` angegeben werden können, sind:

Parameter	Effekt	Beispiel
`col`	Farbe	`col="red"`
`lwd`	Strichstärke	`lwd=2`
`main`	Gesamttitel	`main="Gesamttitel"`
`xlab`	Titel für die x-Achse	`xlab="X"`
`ylab`	Titel für die y-Achse	`ylab="Y"`
`pch`	Symbol für Punkte	`pch="o"`
`cex`	Symbolgröße	`cex=2`

Alternativ zur direkten Angabe der Optionen können diese auch mittels eines vorangestellten par-Befehls gesetzt werden. `par` wurde bereits zum Beginn des Abschnittes zur Unterteilung des Grafik-Objektes erwähnt. Eine *Streudiagramm-Matrix* eines multivariaten Datensatzes wird mittels `pairs` erzeugt. Als Argument ist ein Datensatz oder eine Matrix anzugeben. Dann ist die einfache Form des Aufrufes: `pairs(x)`. Auch diese Grafik lässt sich mit den oben angegebenen Parametern weiter verschönern.

Ergänzen und Verschönern von Grafiken

Ist eine der oben angegebenen Grafiken erstellt, so können Linien, Punkte und Bezeichnungen hinzugefügt werden. Dies geschieht mit den sogenannten Low-Level-Grafikfunktionen. Ihr Aufruf erzeugt keine neue Grafik, sondern verändert eine bereits vorhandene.

Low-Level-Grafikfunktionen

`abline`	Fügt Linien zu dem aktuellen Plot hinzu. Anzugeben sind Achsenabschnitt und Steigung, welche die Gerade bestimmen
`arrows`	Hinzufügen von Pfeilen zwischen Punktepaaren.
`legend`	Fügt eine Legende zu dem aktuellen Plot hinzu.
`lines`	Hinzufügen von Linien zu dem aktuellen Plot.
`par`	Setzen und Abfragen von Grafikparametern.
`points`	Hinzufügen von Punkten zu dem aktuellen Plot.
`polygon`	Hinzufügen eines Polygonzuges, auch ausgefüllt.
`rug`	Hinzufügen eines Rug ('ergänzendes Stabdiagramm') zu einer Grafik.
`segments`	Hinzufügen von Linien-Segmenten zwischen Punktepaaren.
`text`	Hinzufügen von Textsymbolen zum aktiven Plot.
`title`	Hinzufügen eines Titels zum aktiven Plot.

Die Funktion `abline` ist vor allem günstig, um Regressionslinien zu Streudiagrammen oder QQ-Diagrammen hinzuzufügen. Verlangt werden als Argumente der Achsenabschnitt `a` und die Steigung `b`. Mit weiteren Argumenten können die Farbe der ab-Geraden sowie die Strichstärke eingestellt werden. Für ein Beispiel sei auf den folgenden Abschnitt verwiesen.

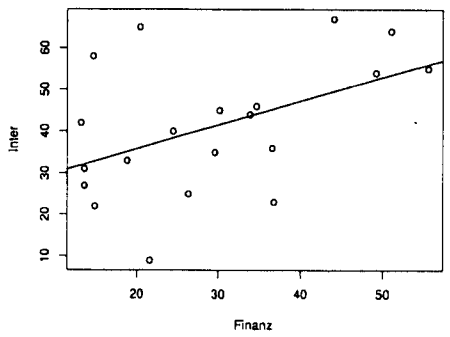
Linienzüge werden durch die Funktion `lines` mittels der Angabe von Punkten erzeugt. Im Aufruf `lines(x,y)` sind `x` und `y` zwei gleichlange Vektoren. Sollen mehr als ein Linienzug einem Plot hinzugefügt werden, so ist `lines` mehrmals aufzurufen.

Im oben angegebenen Streudiagramm der Finanzkraft und Internationalität von Firmen soll die Regressionsgerade des Regressionsansatzes

$$Inter = \beta_0 + \beta_1 Finanz + \varepsilon$$

eingefügt werden. Dazu werden zunächst die Regressionskoeffizienten berechnet. Dann brauchen jeweils nur Vektoren mit den Anfangs- und Endkoordinaten angegeben zu werden:

```
> r <-lm(Inter~Finanz)
> beta<-coefficients(r)
> lines(c(10,60),c(beta[1]+beta[2]*10,beta[1]+beta[2]*60))
```

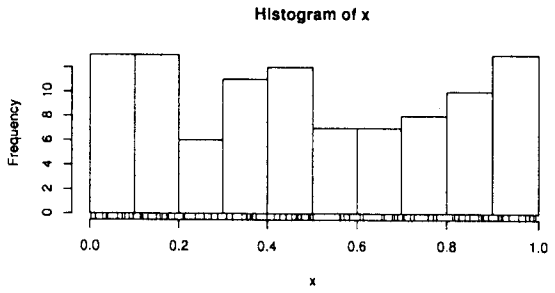


Das Hinzufügen von Punkten mittels `points` erfolgt gemäß der gleichen Logik.

Unter einem *Rug* wird ein am Rande angebrachtes Stabdiagramm verstanden. Das ist z.B. sinnvoll, wenn man eine übersichtliche Darstellung mittels eines Histogramms haben und dennoch nicht auf eine detailliertere Angabe verzichten möchte, wo die Werte tatsächlich liegen. Beispielsweise führt die Befehlssequenz

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

zu der folgenden Grafik.



Als Titel kann der Grafik nun 'Hübsches Bild' gegeben werden. Der Text ist dabei in Anführungszeichen zu setzen: `title("Hübsches Bild")`.

Weiterer Text wird mit `text` in die Grafik eingefügt: `text(1,5,"Beginn")` Der angegebene Punkt ist dann das Zentrum des Textfeldes.

Mit der Anweisung `postscript(file="myfile.ps")` wird die anschließend erzeugte Grafik als Postskript-Datei in der Datei myfile.ps gespeichert. Mit `dev.off()` wird das Abspeichern beendet. Alternativ kann auch einfach ein Rechtsklick auf das Grafikfenster vorgenommen und dort die Auswahl zwischen Bitmap und Postskript getroffen werden.

7 Statistische Modelle in R

R stellt eine miteinander verzahnte Suite von Möglichkeiten bereit, mit denen die Anpassung statistischer Modelle sehr vereinfacht wird. Der Output ist zunächst minimal; um Details zu bekommen, müssen im Anschluss Extractor-Funktionen eingesetzt werden.

Den Prototypen eines statistischen Modells bildet das lineare Regressionsmodell mit unabhängigen, homoskedastischen Fehlern

$$Y_v = \sum_{i=0}^{p-1} \beta_i x_{iv} + \varepsilon_v, \quad v = 1, \ldots, n,$$

wobei die Fehler ε_v normalverteilt sind, $\varepsilon_v \sim \mathcal{N}(0, \sigma^2)$. Häufig ist x_{0v} konstant eins; dadurch wird dann der Achsenabschnitt (intercept) festgelegt.

In R wird die Struktur eines Modells gemäß einer Formel wie der folgenden spezifiziert:

```
Responsevariable ~ erklärende Variablen,
```

wobei der Operator " ~ " interpretiert wird als 'wird modelliert als Funktion von'.

Ein paar Beispiele zeigen, wie die Formulierung im Prinzip lautet:

Formel	Modell
`y~x`	Ein einfaches lineares Regressionsmodell der Form $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, wobei der Term für den Achsenabschnitt aufgrund der Voreinstellung implizit enthalten ist.
`y~-1+x`	Ein einfaches lineares Regressionsmodell ohne Achsenabschnitt; m.a.W. eine Regression durch den Ursprung.
`y~poly(x,2)`	Polynomiale Regression des Grades 2 von **y** auf **x**.
`y~A`	Einwegsvarianzanalyse von **y**, bei der die Faktorstufen durch `A` bestimmt sind.
`y~A+B+B:A`	Zweifaktormodell mit Wechselwirkungen.

Arithmetische Symbole, die auf der rechten Seite der Modellformulierungen auftauchen, haben hier eine modellspezifische Interpretation:

\+ Einbeziehung der Variablen in das Modell;

\- Ausschluss der Variablen aus dem Modell;

* Einschluss der Variablen sowie ihrer Interaktion der Variablen in das Modell;

: Einschluss der Interaktion der Variablen in das Modell;

/ Verlangt, dass die Variable als hierarchisch untergeordnete berücksichtigt wird;

| Zeigt eine Bedingung an. (So ist y ~ x|z zu lesen als 'y als Funktion von x gegeben z'.)

Die grundlegende Funktion zur Anpassung normaler multipler linearer Modelle ist `lm()`; eine von allen Optionen befreite Version des Aufrufs hat die einfache Struktur:

```
fitted.model <- lm(formula)
```

Um Informationen über das angepasste Modell zu extrahieren, stehen die folgenden generischen Funktionen zur Verfügung:

add1	coef	effects	kappa	predict	residuals
alias	deviance	family	labels	print	step
anova	drop1	formula	plot	proj	summary

Auch wenn die Varianzanalyse unter die linearen Modelle subsumiert werden kann, gibt es wegen der großen Bedeutung eine eigenständige Funktion:

```
fitted.model <- aov(formula, data=data.frame)
```

Die meisten der oben angegebenen Funktionen zur Extraktion von Informationen über das angepasste Modell sind auch hier anwendbar.

Die Funktion `update()` bringt eine Vereinfachung, wenn ein neues Modell angepasst werden soll, das sich von alten nur geringfügig unterscheidet. Ihr Aufruf lautet

```
new.model <- update(old.model, new.formula)
```

In dieser Funktion kann der Punkt verwendet werden, um zu spezifizieren, 'was bereits im Modell enthalten ist'. Ist also das Ausgangsmodell

```
model1 <- lm(y~x*z),
```

so wird durch den folgenden Aufruf die Interaktion entfernt und nur die beiden Variablen x und z werden berücksichtigt:

```
model2 <- update(model1, ~.-x:z).
```

Beispiele sind unter den R-Codes im Text zu finden.

8 Tabellarische Überblicke über wichtige Funktionen

Die wichtigsten Funktionen sind nach inhaltlichen Gesichtspunkten in Tabellen zusammengefasst. Diese sind mathematische Funktionen, statistische Funktionen, Funktionen zur Erzeugung und Bearbeitung von Vektoren und Matrizen sowie eine Tabelle mit Verteilungen. Als letzte Übersicht sind diese und noch einige weitere Funktionen alphabetisch zusammengefasst. Diese Aufstellungen sind nicht vollständig. Es werden auch nicht alle optionalen Argumente der Funktionen angegeben. Für weitere Funktionen und zum genauen Gebrauch der angegebenen ist die R-Hilfe zu konsultieren.

Tabelle 9.5: Mathematische Funktionen

Funktionsname	Funktionsbeschreibung
`abs(x)`	Beträge der Elemente von x.
`ceiling(x)`	Aufrunden der Werte in x (auf ganze Zahlen).
`choose(n,x)`	Binomialkoeffizient n über x.
`cos(x)`	Kosinus von x (x in Bogenmaß).
`exp(x)`	Exponentialfunktion für die in x enthaltenen Werte.
`floor(x)`	Abrunden der Werte in x (auf ganze Zahlen).
`log(x)`	Logarithmus zur Basis e (=2.718282.....).
`log10(x)`	Logarithmus zur Basis 10.
`round(x)`	Rundet die in x enthaltenen Werte. Über das optionale Argument `digits = d` kann die Nachkommastelle bestimmt werden, bei der gerundet werden soll.
`sign(x)`	Bestimmt die Vorzeichen der Elemente von x.
`sin(x)`	Sinus von x (x in Bogenmaß).
`sqrt(x)`	Quadratwurzel der Elemente von x.
`tan(x)`	Tangens von x (x in Bogenmaß).
`trunc(x)`	Abrunden der Werte von x (auf ganze Zahlen).

Tabelle 9.6: Statistische Funktionen

Funktionsname	Funktionsbeschreibung
`cor(x,y)`	Korrelationskoeffizient der Vektoren x und y. Wenn x eine Matrix ist und y weggelassen wird, werden die Korrelationen der Spalten von x berechnet. Wird zusätzlich y angegeben, so werden die Korrelationen der Spalten von x mit denen von y bestimmt.
`cut(x,breaks)`	Teilt den Wertebereich von x in Intervalle mit den in breaks angegebenen Intervallgrenzen auf und kodiert die Werte von x entsprechend der Klasse, in die sie fallen. Die Klasse mit den kleinsten Werten korrespondiert mit dem Wert 1 usw.
`length(x)`	Länge des Vektors x bzw. Anzahl der Elemente der Matrix x.
`lm(y~x)`	Führt eine Regression mit der abhängigen Variablen y und der unabhängigen Variablen x durch. Dabei wird ein konstantes Glied berücksichtigt.
`mad(x)`	Median der absoluten Abweichungen der Elemente von x vom Median von x.
`max(x)`	Maximum der Elemente von x.
`mean(x)`	Arithmetisches Mittel der Elemente von x.
`median(x)`	Median der Elemente von x.
`min(x)`	Minimum der Elemente von x.
`quantile(x,probs=p)`	Bestimmt die empirischen Quantile von x zu den in p angegebenen Anteilen. Dabei wird linear interpoliert.

`range(x)`	Gibt den Vektor zurück, der aus dem kleinsten und dem größten der in x enthaltenen Werte besteht.
`rank(x)`	Gibt die Rangwerte eines numerischen Vektors x wieder. Bei Bindungen oder Ties werden mittlere Ränge berechnet.
`rle(x)`	Bestimmung der Runs, der aufeinander folgenden gleichen Werte in x.
`scale(x,..)`	Zentriert und/oder skaliert die numerische Matrix x spaltenweise. Die Auswahl nur einer der Operationen geschieht, indem bei `center=TRUE, scale=TRUE` einer der beiden `TRUE` auf `FALSE` gesetzt wird. Es können auch Vektoren angegeben werden, die zur Zentrierung bzw. Skalierung verwendet werden.
`sd(x)`	Standardabweichung der in x enthaltenen Daten.
`stem(x)`	Stem-and-Leaf-Diagramm des Datenvektors x.
`sum(x)`	Summe aller in dem Vektor oder der Matrix x enthaltenen Elemente.
`summary(x)`	Je nach Art der Eingabe gibt diese Funktion eine Übersicht über die wesentlichen statistischen Charakteristika von x aus: univariater Datensatz, Vektor: 5-Zahlen-Zusammenfassung, ohne Umfang der Daten, aber mit arithmetischem Mittel. Ergebnis der Funktion `lm`: Übersicht über das Regressionsergebnis
`table(x,..)`	Häufigkeits- oder Kontingenztafel der Variable x bzw. zweier Variablen x und y.
`tapply(x,y,fun)`	Berechnung der durch `fun` spezifizierten Funktion für eine Maßzahl für Untergruppen; die Gruppenzugehörigkeit ist durch Gruppenindikatoren in y spezifiziert.
`var(x,..)`	Varianz oder Kovarianz der Variablen x bzw. der Variablen x und y.

Tabelle 9.7: Erzeugung und Bearbeitung von Matrizen und Vektoren

Funktionsname	Funktionsbeschreibung
`append(x, v, a)`	Anfügen von Werten v an einen Vektor x nach der durch a bestimmten Stelle, etwa `a=length(x)`
`apply(x,[i],function)`	Zeilen- (i=1) oder spaltenweises (i=2) Anwenden der Funktion `function` auf die Matrix x.
`c(x1,x2,..)`	Werte `x1,x2,..` zu einem Vektor zusammenfassen. Die Angabe kann auch gemäß `xu:xo` erfolgen; dann wird xu wiederholt um 1 erhöht, solange xo nicht überschritten wird.
`cbind(x,y)`	Setzt Vektoren (und Matrizen) nebeneinander zu einer Matrix zusammen.

`cumsum(y)`	Gibt einen Vektor zurück, dessen Elemente die kumulierten Summen der Elemente des Vektors y sind.
`diag(d)`	Macht aus dem Vektor d eine Diagonalmatrix.
`rbind(x,y)`	Setzt Vektoren (und Matrizen) untereinander zu einer Matrix zusammen.
`rep(x,times)`	Wiederholtes Aneinanderfügen des Vektors x entsprechend der in `times` angegebenen Anzahl.
`replace(x,list,val)`	Ersetzt die Werte in x mit den Indizes, die in `list` aufgeführt sind, durch die in val angegebenen Werte.
`seq(from, to)`	Erzeugt eine jeweils um 1 wachsende Folge, bei der der mit `to` angegebene Wert nicht überschritten wird. Die Schrittweite kann mittels des optionalen Arguments `by=i` verändert werden. Mit dem optionalen Argument `length=k` werden k-2 äquidistante Zwischenwerte bestimmt und zusammen mit `from` und `to` ausgegeben.
`sort(x)`	Sortiert den numerischen Vektor x (partiell) aufsteigend (oder absteigend).
`t(x)`	Bestimmt die Transponierte der Matrix x.

Tabelle 9.8: Wahrscheinlichkeitsverteilungen

Verteilung	R-Name	Argumente
Binomial	`binom`	`size, prob`
Cauchy	`cauchy`	`location, scale`
Chiquadrat	`chisq`	`df, ncp`
Exponential	`exp`	`rate`
F	`f`	`df1, df1, ncp`
Gamma	`gamma`	`shape, scale`
Geometrische	`geom`	`prob`
Gleich	`unif`	`min, max`
Hypergeometrische	`hyper`	`m, n, k`
Lognormal	`lnorm`	`meanlog, sdlog`
Logistische	`logis`	`location, scale`
Negative Binomial	`nbinom`	`size, prob`
Normal	`norm`	`mean, sd`
Poisson	`pois`	`lambda`
Student's t	`t`	`df, ncp`
Weibull	`weibull`	`shape, scale`
Wilcoxon	`wilcox`	`m, n`

Bei den Verteilungen ist den in der mittleren Spalte angegebenen Namen ein Buchstabe voranzustellen, und zwar einer der Buchstaben `d, p, q` oder `r`. Dabei bedeuten:

d	Dichte- oder Wahrscheinlichkeitsfunktion	p	Verteilungsfunktion
q	Inverse der Verteilungsfunktion	r	Zufallszahlen

Die angegebenen Argumente ergeben sich als Parameter der Verteilungen. Dafür gibt es Voreinstellungen, die wirksam werden, wenn diese Parameter nicht aufgeführt werden. Obligatorisch an erster Stelle muss jeweils eines der Argumente stehen:

Bei `d, p`: Vektor von Werten;
Bei `q`: Vektor von Anteilen;
Bei `r`: Anzahl der zu erzeugenden Zufallszahlen .

Somit wird durch `pnormal(c(-2,3.1))` der Vektor der Werte der Verteilungsfunktion der Normalverteilung mit $\mu = 0$ und $\sigma = 1$ an den beiden Stellen -2 und 3.1 bestimmt. `qnormal(0.2,mean=2,sd=3)` gibt das 0.2-Quantil der Normalverteilung mit $\mu = 2$ und $\sigma = 3$ zurück.

Tabelle 9.9: Alphabetische Liste der wichtigsten Funktionen

Funktionsname	Funktionsbeschreibung
`abs(x)`	Beträge der Elemente von `x`
`acos(x)`	Arcuscosinus der Elemente von `x`; diese müssen zwischen -1 und +1 liegen. Für andere Werte wird ein fehlender Wert (NA) ausgegeben.
`all(x)`	Überprüfen, ob alle Elemente von `x` TRUE sind.
`any(x)`	Überprüfen, ob eines der Elemente von `x` TRUE ist.
`append(x, v, a)`	Anfügen von Werten `v` an einen Vektor `x` nach der durch `a` bestimmten Stelle, etwa `a=length(x)`.
`apply(x,i,function)`	Zeilen- (`i=1`) oder spaltenweises (`i=2`) oder zeilen- und spaltenweises (`i=c(1,2)`) Anwenden der Funktion `function` auf die Matrix `x`.
`asin(x)`	Arcussinus der Elemente von `x`; diese müssen zwischen -1 und +1 liegen. Für andere Werte wird ein fehlender Wert (NA) ausgegeben.
`assign("x",...)`	Zuweisen von Werten zu der Variablen `x`; die Punkte stehen für einen einzelnen Wert oder eine Folge von Werten.
`atan(x)`	Arcustangens der Elemente von `x`.
`c(x1,x2,..)`	Werte `x1,x2,..` zu einem Vektor zusammenfassen. Die Angabe kann auch gemäß `xu:xo` erfolgen; dann wird `xu` wiederholt um 1 erhöht, solange `xo` nicht überschritten wird.
`cbind(x,y)`	Setzt Vektoren (und Matrizen) nebeneinander zu einer Matrix zusammen.
`ceiling(x)`	Aufrunden der Werte in `x` (auf ganze Zahlen).
`choose(n,x)`	Binomialkoeffizient `n` über `x`.

`cor(x,y)`	Korrelationskoeffizient der Vektoren `x` und `y`. Wenn `x` eine Matrix ist und `y` weggelassen wird, werden die Korrelationen der Spalten von `x` berechnet. Wird zusätzlich `y` angegeben, so werden die Korrelationen der Spalten von `x` mit denen von `y` bestimmt.
`cos(x)`	Cosinus der Elemente von `x`.
`cumprod(y)`	Gibt einen Vektor zurück, dessen Elemente die kumulativen Produkte der Elemente des Vektors `y` sind.
`cumsum(y)`	Gibt einen Vektor zurück, dessen Elemente die kumulierten Summen der Elemente des Vektors `y` sind.
`cut(x,breaks)`	Teilt den Wertebereich von `x` in Intervalle mit dem in `breaks` angegebenen Intervallgrenzen auf und kodiert die Werte von `x` entsprechend der Klasse, in die sie fallen. Die Klasse mit den kleinsten Werten korrespondiert mit den Wert 1 usw.
`diag(d)`	Macht aus dem Vektor `d` eine Diagonalmatrix.
`dim(x)`	Bestimmt die Dimension einer Matrix. Für weitere Funktionalitäten siehe Seite 297.
`exp(x)`	Exponentialfunktion für die in `x` enthaltenen Werte.
`floor(x)`	Abrunden der Werte in `x` (auf ganze Zahlen).
`identical(x, y)`	Test, ob zwei Vektoren bzw. Matrizen exakt gleich sind. Die Ausgabe ist `TRUE`, wenn dies gilt und `FALSE` in jedem anderen Fall.
`inverse.rle(r)`	Rekonstruiert einen Datenvektor aus der Tabelle der Runs `r`, der Angaben wie oft das jeweilige Element von dieser Stelle an hintereinander vorkommt, bis es durch ein anderes abgelöst wird.
`is.na(x)`	Überprüfen auf fehlende Werte in `x`. An jeder Stelle, an der ein Wert vorhanden ist, wird `FALSE` ausgegeben, sonst `TRUE`.
`length(x)`	Länge des Vektors `x` bzw. Anzahl der Elemente der Matrix `x`.
`lm(y~x)`	Führt eine Regression mit der abhängigen Variablen `y` und der unabhängigen Variablen `x` durch. Dabei wird ein konstantes Glied berücksichtigt.
`log(x)`	Logarithmus zur Basis e (=2.718282.....).
`log10(x)`	Logarithmus zur Basis 10.
`log2(x)`	Logarithmus zur Basis 2.
`mad(x)`	Median der absoluten Abweichungen der Elemente von `x` vom Median von `x`.
`match(x, table)`	Gibt die Positionen in `table` an, in denen die in `x` enthaltenen Werte vorkommen.
`max(x)`	Maximum der Elemente von `x`.
`mean(x)`	Arithmetisches Mittel der Elemente von `x`.

`median(x)`	Median der Elemente von x.
`min(x)`	Minimum der Elemente von x.
`objects()`	Namen aller Variablen der aktuellen R-Sitzung
`order(x,...)`	Sortiert den Vektor x mit der Möglichkeit, nur Teile zu sortieren und zwischen auf- und absteigener Sortierung zu wählen.
`prod(x)`	Gibt das Produkt aller Werte wider, die in dem Vektor x enthalten sind.
`quantile(x,probs=p)`	Bestimmt die empirischen Quantile des Vektors x zu den in p angegebenen Anteilen. Dabei wird linear interpoliert.
`quit(), q()`	Beenden der R-Sitzung.
`range(x)`	Gibt den Vektor zurück, der aus dem kleinsten und dem größten der in x enthaltenen Werte besteht.
`rank(x)`	Gibt die Rangwerte eines numerischen Vektors wieder. Bei Bindungen oder Ties werden mittlere Ränge berechnet.
`rbind(x,y)`	Setzt Vektoren (und Matrizen) untereinander zu einer Matrix zusammen.
`rep(x,times)`	Wiederholtes Aneinanderfügen des Vektors x entsprechend der in `times` angegebenen Anzahl.
`replace(x,list,val)`	Ersetzt die Werte in x mit den Indizes, die in `list` aufgeführt sind, durch die in val angegebenen Werte.
`rev(x)`	Dreht den Vektor x um, so dass das erste Element das letzte wird usw.
`rle(x)`	Tabelle der runs von x, d.h. der Angaben wie oft das jeweilige Element in x hintereinander vorkommt, bis es durch ein anderes abgelöst wird.
`rm(x,y,...,list=c)`	Löschen der Variablen x und y (und von weiteren, in ... angeführten), oder die in dem Charaktervektor `list` enthalten sind. Auch eine Kombination von beiden Angaben ist möglich.
`round(x)`	Rundet die in x enthaltenen Werte. Über das optionale Argument `digits = d` kann die Nachkommastelle bestimmt werden, bei der gerundet werden soll.
`scale(x,..)`	Zentriert und/oder skaliert die numerische Matrix x spaltenweise. Die Auswahl nur einer der Operationen geschieht, indem bei `center=TRUE` , `scale=TRUE` einer der beiden `TRUE` auf `FALSE` gesetzt wird. Es können auch Vektoren angegeben werden, die zur Zentrierung bzw. Skalierung verwendet werden.
`scan("datei")`	Einlesen von Daten aus einer externen Datei. `datei` kann eine vollständig spezifizierte Pfadangabe enthalten. (Mit doppelten Backslashes!)

`seq(from, to)`	Erzeugt eine jeweils um 1 wachsende Folge, bei der der mit `to` angegebene Wert nicht überschritten wird. Die Schrittweite `i` kann mittels des optionalen Arguments `by=i` verändert werden. Mit dem optionalen Argument `length=k` werden k-2 äquidistante Zwischenwerte bestimmt und zusammen mit from und to ausgegeben.
`sign(x)`	Bestimmt die Vorzeichen der Elemente von x.
`sin(x)`	Sinus der Elemente von x.
`solve(x)`	Inverse der regulären Matrix x.
`sort(x)`	Sortiert den numerischen Vektor x (partiell) aufsteigend (oder absteigend).
`source(x)`	Einbinden von R-Code, der in der in x angegebenen Datei steht.
`sqrt(x)`	Quadratwurzel der Elemente von x.
`sd(x)`	Standardabweichung der in x enthaltenen Daten.
`stem(x)`	Erstellen eines Stem-and Leaf-Diagramms für einen Datenvektor x.
`substring(x,...)`	Extrahiert oder ersetzt einen Teil der Zeichenkette x.
`sum(x)`	Summe aller in dem Vektor oder der Matrix x enthaltenen Elemente.
`summary(x)`	Je nach Art der Eingabe gibt diese Funktion eine Übersicht über die wesentlichen statistischen Charakteristika von x aus: univariater Datensatz, Vektor: 5-Zahlenzusammenfassung, ohne Umfang der Daten, aber mit arithmetischem Mittel. Ergebnis der Funktion `lm`: Übersicht über das Regressionsergebnis.
`t(x)`	Bestimmt die Transponierte der Matrix x.
`table(x,..)`	Häufigkeits- oder Kontingenztafel der Variable x bzw. zweier Variablen x und y.
`tan(x)`	Tangens der Werte von x.
`tapply(x,y,fun)`	Berechnung der durch `fun` spezifizierten Maßzahl für Untergruppen; die Gruppenzugehörigkeit ist durch Gruppenindikatoren in y spezifiziert.
`trunc(x)`	Abrunden der Werte von x (auf ganze Zahlen).
`var(x,..)`	Varianz oder Kovarianz der Variablen x bzw. zweier Variablen x und y.