Helmut Strasser

Notizen zu R

24. September 2003

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Homepage und Installation

Zu Beginn verweisen wir auf die Homepage von R:

`http://www.r-project.org`

Von dieser Webseite aus kann R kostenlos geladen und installiert werden. Das selbstinstallierende File

`http://www.ci.tuwien.ac.at/R/bin/windows/base/SetupR.exe`

ist 15.5 MB groß. Studierenden, die keinen Breitbandzugang zum Internet haben, bieten wir eine CD an, von der aus R installiert werden kann. Bestellungen für die CD nehmen wir per Email entgegen.

1.2 Dokumentation

Das offizielle Einführungsskript ist:

`http://www.ci.tuwien.ac.at/R/doc/manuals/R-intro.pdf`

Die vorliegenden Notizen verstehen sich als Einführung in das offizielle Einführungsskript.

1.3 Literatur

Das führende Buch über Statistik mit R (bzw. S-Plus, die kommerzielle Variante) ist:

`http://www.stats.ox.ac.uk/pub/MASS3/`

1.4 Installation und Start

Man installiert R nach den Anweisungen, die mitgeliefert werden. Anschließend startet man R, indem man

```
RGui.exe
```

aufruft. R arbeitet immer in einem bestimmten Arbeitsverzeichnis (workdir), welches mit einem Menupunkt gesetzt werden kann.

1.5 Hilfesystem

R besitzt ein ausführliches Online-Hilfesystem. Information über das Hilfesystem erhält man durch

```
help("help")
```

Kapitel 2

R als Rechenmaschine

Im folgenden geben wir eine Einführung in die einfachsten Anwendungen und Grundlagen von R. Um weitere Informationen über die verwendeten Befehle und ihre Syntax zu erhalten, verwendet man das Hilfesystem von R.

2.1 Erste Schritte

Wir starten R.

R rechnet mit Zahlen wie ein Taschenrechner:

```
> 5+7
[1] 12

> 3*11
[1] 33

> 5^2
[1] 25

> 1/100
[1] 0.01

> 2*(5-7/2)^(1/3)
[1] 2.289428
```

Man kann aber auch mit Variablen rechnen. Wenn man Variablennamen Zahlen zuweist, dann gelten die gleichen Rechengesetze wie für Zahlen:

```
> a <- 5
> b <- 7
> c <- a+b
> c
[1] 12
```

Man kann auch aussagekräftige Variablennamen verwenden:

```
> Ertrag <- 100
> Kosten <- 80
> Gewinn <- Ertrag-Kosten
> Gewinn
[1] 20
```

2.2 R-Vektoren

Eine weitere wichtige Datenstruktur in R sind R-Vektoren. Ein R-Vektor ist ganz einfach eine Folge von gleichartigen Objekten, im einfachste Fall von Zahlen. In der üblichen Sprache der Informatik sind R-Vektoren eindimensionale Felder.

Einen R-Vektor erzeugt man am einfachsten durch Aufzählung seiner Komponenten:

```
> a <- c(1,2,3,4,5)
> a
[1] 1 2 3 4 5

> length(a)
[1] 5
```

Komponenten eines R-Vektors erhält man durch Indizierung:

```
> a[1]
[1] 1

> a[5]
[1] 5

> a[1:3]
[1] 1 2 3

> a[c(1,3,5)]
[1] 1 3 5
```

Es gibt viele Möglichkeiten, R-Vektoren mit besonderen Eigenschaften zu erzeugen, ohne die Komponenten aufzählen zu müssen. Zum Beispiel durch Wiederholung:

```
> rep(0,times=5)
[1] 0 0 0 0 0

> rep(c(1,2),times=3)
[1] 1 2 1 2 1 2
```

Die folgenden Beispiele zeigen weitere Möglichkeiten und sind selbsterklärend:

```
> 5:12
[1]  5  6  7  8  9 10 11 12

> seq(from=5,to=12)
```

```
[1]  5  6  7  8  9 10 11 12

> seq(from=5,to=12,by=2)
[1]  5  7  9 11

> seq(from=5,to=12,length=3)
[1]  5.0  8.5 12.0

> seq(from=5,to=12,length=7)
[1]  5.000000  6.166667  7.333333  8.500000  9.666667
[6] 10.833333 12.000000
```

R kann auch mit R-Vektoren rechnen:

```
> a <- 1:5
> 3*a
[1]  3  6  9 12 15

> -a
[1] -1 -2 -3 -4 -5

> 1/a
[1] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000
```

Klar ist, wie gleich lange R-Vektoren bei Rechenoperationen verarbeitet werden:

```
> a <- 1:5
> b <- 3:7
> a+b
[1]  4  6  8 10 12

> a*b
[1]  3  8 15 24 35

> a/b
[1] 0.3333333 0.5000000 0.6000000 0.6666667 0.7142857

> a^b
[1]     1    16   243  4096 78125
```

Eine besondere Rechenoperation ist das Skalarprodukt:

```
> a%*%b
     [,1]
[1,]   85
> b%*%a
     [,1]
[1,]   85
```

Davon zu unterscheiden ist das äußere Produkt:

```
> c(1,2,3)%o%c(5,6,7)
```

```
     [,1] [,2] [,3]
[1,]    5    6    7
[2,]   10   12   14
[3,]   15   18   21
```

Bei verschieden langen R-Vektoren wird der kürzere R-Vektor so oft wiederholt, bis der längere R-Vektor abgearbeitet ist:

```
> a <- c(1,2)
> b <- 10:15
> a+b
[1] 11 13 13 15 15 17

> a*b
[1] 10 22 12 26 14 30
```

2.3 Matrizen

Eine Matrix ist eine Liste von gleichartigen Objekten, die als Rechteck angeordnet sind:

```
> a=matrix(1:12,nrow=3,ncol=4)
> a
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Zeilenvektoren und Spaltenvektoren sind Matrizen, bei denen ein der beiden Dimensionen die Länge 1 hat. Es ist wichtig zu verstehen, daß Zeilenvektoren und Spaltenvektoren nicht R-Vektoren, sondern Matrizen sind !

2.3.1 Arrays (für Freaks)

Um mit R frustrationsfrei umgehen zu können, muß man ein wenig Einfühlungsvermögen einwickeln. Dazu ist es nützlich, eine vernünftige Illusion darüber zu bekommen, was „wirklich" passiert.

Vektoren sind ein elementarer Grundbaustein von R. Wie alle Objekte in R, haben Vektoren gewisse Eigenschaften, und zwar grundsätzlich **length** und **mode**:

```
> a <- c(1,2,3)
> mode(a)
[1] "numeric"
> length(a)
[1] 3
```

Eine einzelne Zahl ist ein R-Vektor der Länge 1.

Vektoren können eine weitere Eigenschaft haben, welche **dim**, also Dimension heißt, und einfach ein Vektor von Zahlen ist. Ein Vektor mit einem Dimensionsvektor ist ein Feld (**array**).

Bevor wir umständliche Definitionen formulieren, sehen wir uns ein Beispiel an:

```
> a <- 1:6
> dim(a)
NULL
> dim(a) <- c(2,3)
> dim(a)
[1] 2 3
> a
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> a[2,2]
[1] 4
```

Wir haben den Vektor **a** durch Setzen des Dimensionsvektors zu einem Array gemacht:

```
> is.array(a)
[1] TRUE
```

Entfernt man den Dimensionsvektor, dann wird unser Array wieder zu einem Vektor:

```
> dim(a) <- NULL
> dim(a)
NULL
> a
[1] 1 2 3 4 5 6
```

Zwei Feinheiten sind noch zu erwähnen.

Erstens kann ein Array immer auch als Vektor behandelt werden:

```
> a <- 1:6
> dim(a) <- c(2,3)
> a
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> a[2,2]
[1] 4
> a[4]
[1] 4
```

Zweitens gibt es eindimensionale Arrays, die sich von R-Vektoren nur dadurch unterscheiden, daß sie einen Dimensionsvektor der Läange 1 haben:

```
> a <- 1:6
> is.array(a)
[1] FALSE
> dim(a)=6
> a
[1] 1 2 3 4 5 6
> is.array(a)
```

```
[1] TRUE
```

Wir fassen zusammen:

- Matrizen sind zweidimensionale Arrays, dh. ihr Dimensionsvektor ist ein Vektor der Länge 2.
- Zeilenvektoren und Spaltenvektoren sind Matrizen, also zweidimensionale Felder (ihre Dimensionsvektor hat die Länge 2), aber einer der Komponenten des Dimensionsvektors hat den Wert 1.
- Eindimensionale Felder haben als Dimensionsvektor eine Zahl, die ihrer Länge entspricht.
- R-Vektoren haben keine Dimension.

2.3.2 Erzeugung von Matrizen

Um eine Matrix zu erzeugen, muß man den R-Vektor aller Elemente angeben und erklären, wieviele Zeilen und Spalten die Matrix haben soll:

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

Etwas übersichtlicher (aber keineswegs kürzer) geht das so:

```
> a <- 1:12
> b <- matrix(a,nrow=3,ncol=4)
> b
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Besitzt der R-Vektor weniger Elemente als die geplante Matrix, dann wird der R-Vektor solange wiederholt, bis die Matrix voll ist:

```
> m <- matrix(c(1,2,3),nrow=3,ncol=3)
> m
      [,1] [,2] [,3]
 [1,]    1    1    1
 [2,]    2    2    2
 [3,]    3    3    3
```

Die Länge einer Matrix

```
> length(m)
[1] 9
```

ist die Anzahl aller Elemente. Die Anzahl der Zeilen und Spalten erhält man durch

```
> dim(m)
[1] 3 3
```

Ein andere Möglichkeit, eine Matrix durch Festlegen ihrer Elemente zu definieren, besteht durch den Matrixeditor:

```
> m <- edit(matrix(0,nrow=3,ncol=4))
```

Spezielle Matrizen kann man manchmal einfacher herstellen, z.B. eine Diagonalmatrix:

```
> diag(c(1,2,3,4))
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    2    0    0
[3,]    0    0    3    0
[4,]    0    0    0    4
```

oder eine Einheitsmatrix:

```
> diag(5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
```

2.3.3 Rechnen mit Matrizen

Für Matrizen gelten völlig andere Rechengesetze als für R-Vektoren. In R sind für Matrizen nur Rechenoperationen möglich, die auf Grund der Zeilen- und Spaltenanzahl der beteiligten Matrizen erlaubt sind.

Es gibt zunächst die Rechenoperationen, die elementweise ausgeführt werden, wie +, -, *, /, ^ usw. Diese Rechenoperationen sind nur möglich, wenn die beteiligten Matrizen gleich groß sind.

Darüber hinaus stehen die besonderen Rechenoperationen der Matrizenrechnung zu Verfügung, insbesondere die Matrixmultiplikation %*%:

```
> a <- matrix(1:6,nrow=2,ncol=3)
> a
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> b <- matrix(1:6,nrow=3,ncol=2)
> b
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
> a%*%b
     [,1] [,2]
[1,]   22   49
[2,]   28   64

> b%*%a
     [,1] [,2] [,3]
[1,]    9   19   29
[2,]   12   26   40
[3,]   15   33   51
> a*b
Error in a * b : non-conformable arrays
```

Reguläre Matrizen invertiert man mit **solve**:

```
> a <- matrix(rnorm(9),nrow=3,ncol=3)
> a
           [,1]        [,2]       [,3]
[1,] -0.1705125  1.20435144 -0.1412469
[2,] -0.5342134 -0.05923893  0.4288122
[3,]  0.8236884  1.12844936 -0.5861073

> round(a%*%solve(a))
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

2.3.4 Matrizen und Vektoren

Im folgenden sprechen wir über das komplizierte Verhältnis zwischen Matrizen und R-Vektoren. Dabei muß nochmals klargestellt werden, daß R-Vektoren in diesem (von R verwendeten Sprachgebrauch) nicht Zeilenvektoren oder Spaltenvektoren sind. Unter Zeilenvektoren und Spaltenvektoren versteht man spezielle Matrizen, also zweidimensionale Felder, bei denen eine Komponente der Dimension den Wert 1 hat. R-Vektoren sind etwas anderes, und eindimensionale Felder noch einmal etwas anderes.

Um erklären zu können, was R tut, wenn es mit Feldern und Vektoren rechnet, muß man (leider) zwischen R-Vektoren (ohne Dimension) und 1-dimensionalen Feldern unterscheiden. Es gibt drei Fälle:

1. Matrix und R-Vektor
2. Matrix und 1-dimensionales Feld
3. Matrix und Matrix

Es ist ziemlich sinnlos, aufzählen zu wollen, was R hier im einzelnen tut. Das meiste ist auch unerklärlich: So wird z.B. die Operation $\% * \%$ in den Fällen 1. und 2. gleichbehandelt, während die Operation $*$ in den Fällen 2. und 3. das Gleiche tut.

Wir beschränken uns daher auf ein typisches Beispiel.

Wenn man in Rechenausdrücken Matrizen und R-Vektoren (die keine Matrizen sind) mischt, dann entstehen Besonderheiten, die anfangs verwirrend sind, die allerdings bei raffiniertem Einsatz eine große Flexibilität ergeben.

Erinnern wir uns zunächst an die Eigenschaften von Zeilen- und Spaltenvektoren. Das sind bekanntlich Matrizen, also zweidimensionale Felder, bei denen eine Komponente der Dimension den Wert 1 hat. Also:

```
> a <- matrix(c(1,2,3),nrow=3,ncol=1)
> a
     [,1]
[1,]    1
[2,]    2
[3,]    3
```

ist ein Spaltenvektor, und

```
> b <- matrix(c(4,5,6),nrow=1,ncol=3)
> b
     [,1] [,2] [,3]
[1,]    4    5    6
```

ist ein Zeilenvektor. Die Matrixmultiplikation ergibt, in Abhängigkeit von der Reihenfolge der Faktoren,

```
> a%*%b
     [,1] [,2] [,3]
[1,]    4    5    6
[2,]    8   10   12
[3,]   12   15   18

> b%*%a
     [,1]
[1,]   32
```

genau das, was die Matrizenrechnung vorschreibt.

Wichtig: Man beachte, daß dieses Ergebnis anders lauten würde, wenn **a** und **b** nur R-Vektoren, aber keine Matrizen wären ! (Siehe: Skalarprodukt)

Es ist dagegen nicht möglich, die Matrix

```
> c <- matrix(11:19,3,3)
> c
     [,1] [,2] [,3]
[1,]   11   14   17
[2,]   12   15   18
[3,]   13   16   19
```

elementweise mit dem Spaltenvektor **a** oder dem Zeilenvektor **b** zu multiplizieren.

Wenn jedoch an Stelle von Zeilen- oder Spaltenvektoren R-Vektoren verwendet werden, dann treten

die Spielregeln der Matrizenrechnung außer Kraft, und die Regeln der R-Vektorenrechnung gelten. Sehen wir uns ein Beispiel an.

Wir wollen die Zeilen der Matrix **c** mit den Komponenten des Vektors **a** multiplizieren, und zwar die erste Zeile mit der ersten Komponenten, usw. Zu diesem Zweck wandeln wir den Spaltenvektor **a** in einen R-Vektor um:

```
> a <- as.vector(a)
```

und schon kann man multiplizieren:

```
> c*a
     [,1] [,2] [,3]
[1,]   11   14   17
[2,]   24   30   36
[3,]   39   48   57
```

Die Rechenoperation wird elementweise durchgeführt, wobei eine Spalte der Matrix nach der anderen verarbeitet wird, und der beteiligte R-Vektor wird so oft wiederholt, bis alle Elemente der Matrix verarbeitet sind.

Wollte man das gleiche mit den Spalten von **c** machen, müßte man zweimal transponieren (Zeilen mit Spalten vertauschen):

```
> t(t(c)*a)
     [,1] [,2] [,3]
[1,]   11   28   51
[2,]   12   30   54
[3,]   13   32   57
```

2.3.5 Feinheiten für Freaks

R bietet mehr Möglichkeiten, als wir bisher gezeigt haben.

Wenn man aus einer Matrix durch Indizierung eine Zeile oder Spalte auswählt, dann wird das Ergebnis automatisch in einen R-Vektor umgewandelt, dh. es geht jene Dimension des Ergebnisses verloren, die nur die Länge 1 hat. Diese Voreinstellung kann vermieden werden, wenn man die Option **drop** außer kraft setzt:

```
> x <- matrix(rnorm(9),nrow=3,ncol=3)
> x
           [,1]        [,2]       [,3]
[1,] -0.1705125  1.20435144 -0.1412469
[2,] -0.5342134 -0.05923893  0.4288122
[3,]  0.8236884  1.12844936 -0.5861073
> x[,1]
[1] -0.1705125 -0.5342134  0.8236884
> x[,1,drop=FALSE]
           [,1]
[1,] -0.1705125
[2,] -0.5342134
```

```
[3,]  0.8236884
```

Die Funktion **sweep** erlaubt es, die Schwierigkeiten zu vermeiden, die sich am Ende des vorhergehenden Abschnitts ergeben haben. Es geht darum, auf alle Zeilen oder alle Spalten einer Matrix eine bestimmte Rechenoperation anzuwenden.

Es sei z.B.

```
> x <- matrix(1:9,nrow=3,ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

und

```
> a <- c(2,5,10)
```

Um die Elemente von **x** durch die Komponenten von **a** zu dividieren, und zwar entlang dem ersten Index, verwendet man

```
> sweep(x,1,a,"/")
     [,1] [,2] [,3]
[1,]  0.5  2.0  3.5
[2,]  0.4  1.0  1.6
[3,]  0.3  0.6  0.9
```

Das ist das gleiche Ergebnis wie bei

```
> x/a
     [,1] [,2] [,3]
[1,]  0.5  2.0  3.5
[2,]  0.4  1.0  1.6
[3,]  0.3  0.6  0.9
```

Um die Elemente von **x** durch die Komponenten von **a** zu dividieren, und zwar entlang dem zweiten Index, verwendet man

```
> sweep(x,2,a,"/")
     [,1] [,2] [,3]
[1,]  0.5  0.8  0.7
[2,]  1.0  1.0  0.8
[3,]  1.5  1.2  0.9
```

Das ist das gleiche Ergebnis wie bei

```
> t(t(x)/a)
     [,1] [,2] [,3]
[1,]  0.5  0.8  0.7
[2,]  1.0  1.0  0.8
[3,]  1.5  1.2  0.9
```

Der Vorteil der Funktion **sweep** besteht darin, daß sie nicht nur auf Matrizen (zweidimensionale Felder) anwendbar ist, sondern auf Felder mit einer beliebigen Anzahl von Dimensionen.

cbind

2.4 Eingebaute Funktionen

*** IN ARBEIT ***

exp abs sign sqrt

uniroot

2.5 Ein Anwendungsbeispiel

Wir bearbeiten eine Beispiel aus der Input-Output Analyse (siehe Mathematik 1).

Gegeben ist eine Input-Output Tabelle:

```
> tabelle <- edit(matrix(0,nrow=3,ncol=4))
> tabelle
     col1 col2 col3 col4
[1,]  200   90  120   40
[2,]    7    6    3    5
[3,]   18   12   15   15
```

Wir isolieren die Matrix, die aus den ersten drei Spalten besteht:

```
> a <- tabelle[,1:3]
> a
     col1 col2 col3
[1,]  200   90  120
[2,]    7    6    3
[3,]   18   12   15
```

und den Spaltenvektor des Endbedarfs:

```
> b <- tabelle[,4,drop=FALSE]
> b
     [,1]
[1,]   40
[2,]    5
[3,]   15
```

Dann berechnen wir den aktuellen Outputvektor als Zeilensummen der Input-Output Tabelle:

```
> x <- apply(tabelle,1,sum)
> x
[1] 450  21  60
```

Wir warten noch einen Moment, bevor wir den Outputvektor **x**, der nur ein R-Vektor ist, in einen Spaltenvektor umwandeln. Wir brauchen nämlich den R-Vektor vorher noch zur Berechnung der Technologiematrix.

Um die Technologiematrix zu berechnen, müssen wir die Spalten von **a** durch die Komponenten von **x** dividieren:

```
> a <- t(t(a)/x)
> a
           col1      col2 col3
[1,] 0.44444444 4.2857143 2.00
[2,] 0.01555556 0.2857143 0.05
[3,] 0.04000000 0.5714286 0.25
```

Der Übersichtlichkeit halber stellen wir die Elemente der Technologiematrix **a** als Brüche dar:

```
> library(MASS)
> fractions(a)
     col1  col2  col3
[1,]   4/9  30/7     2
[2,] 7/450   2/7  1/20
[3,]  1/25   4/7   1/4
```

Nun machen wir auch den Outputvektor **x** zu einem Spaltenvektor:

```
> x <- matrix(x,nrow=3,ncol=1)
> x
     [,1]
[1,]  450
[2,]   21
[3,]   60
```

Die Input-Output Gleichung besagt, daß **x** mit

```
> a%*%x+b
       [,1]
  [1,]  450
  [2,]   21
  [3,]   60
```

identisch ist.

Nun sollen die Lieferungen des Sektors 2 an den Endverbrauch verdoppelt werden und die Lieferungen der anderen Sektoren halbiert. Wie ist der Output zu planen ?

Der neue Endverbauchsvektor lautet also:

```
> b.neu <- matrix(c(20,10,7.5),3,1)
> b.neu
     [,1]
[1,] 20.0
[2,] 10.0
[3,]  7.5
```

Wir definieren die Einheitsmatrix

```
> eye <- diag(c(1,1,1))
> eye
```

```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

und berechnen

```
> x.neu <- solve(eye-a)%*%b.neu
> x.neu
          [,1]
[1,] 445.42291
[2,]  27.53158
[3,]  54.73233
```

Die Löhne in den drei Wirtschaftssektoren betragen 10, 100 und 30 GE pro Produktion einer Mengeneinheit. Wie müssen die Preise festgelegt werden, damit die Wirtschaft im Gleichgewicht ist ?

Wir bilden den Zeilenvektor der Löhne

```
> w <- c(10,100,30)
> w <- matrix(w,nrow=1,ncol=3)
> w
     [,1] [,2] [,3]
[1,]   10  100   30
```

und berechnen

```
> w%*%solve(eye-a)
         [,1]     [,2]     [,3]
[1,] 51.00642 619.9143 217.3448
```

2.6 Diagramme

2.6.1 High Level Kommandos

Wir beginnen damit, einen einzelnen Punkt in einem Koordinatensystem zu zeichnen:

```
> plot(1,2)
```

Es entsteht ein Diagramm, das den Punkt $(1, 2)$ in einem Koordinatensystem zeichnet. Wir sehen uns das Diagramm genau an und besprechen Möglichkeiten, sein Erscheinungsbild zu ändern.

Die Größe des Diagrammfensters läßt sich interaktiv verändern. Dabei ist bemerkenswert, daß sich die Fontgröße der Beschriftung nicht ändert. Wenn wir das Diagramm als Grafikfile abspeichern, wird das Verhältnis zwischen Diagrammgröße und Fontgröße jedoch fixiert. Um in gedruckter Form eine große Beschriftung zu erhalten, muß das Diagrammfenster daher beim Abspeichern eher klein sein.

Die Größe des Diagrammfensters kann man vor dem Zeichnen des Diagramms festlegen:

```
> windows(width=4,height=4)
> plot(1,2)
```

Das Kommando **plot** ist ein sogenanntes High-Level-Kommando, das eine Anzahl von Operationen selbsttätig ausführt und dabei Voreinstellungen verwendet. Diese Voreinstellungen sind manchmal unerwünscht.

Als erstes sehen wir uns die Namensgebung der Achsen an. Um die voreingestellte Namensgebung zu unterdrücken, verwenden wir:

```
> plot(1,2,ann=FALSE)
```

Man kann auch eigene Beschriftungen hinzufügen:

```
> plot(1,2,xlab="x-Achse",ylab="y-Achse",main="Diagramm")
```

Als nächstes wenden wir uns den Koordinatenachsen zu. Der vom Diagramm dargestellte Bereich der Zeichenfläche wird so festgelegt:

```
> plot(1,2,ann=FALSE,xlim=c(-3,3),ylim=c(-3,3))
```

Allerdings wird die Achsenbeschriftung automatisch durchgeführt. Um die volle Kontrolle darüber zu bekommen, muß die Achsenbeschriftung im **plot**-Kommando unterdrückt und nachträglich hinzugefügt werden:

**** IN ARBEIT ******

boxplot ecdf

2.6.2 Modifikation der Voreinstellungen

2.6.3 Low Level Kommandos

***** IN ARBEIT *******

Kapitel 3

Listen und Dataframes

3.1 Listen

Der grundlegende Datentyp von R ist die Liste. Darunter versteht man eine Folge von Objekten, die von unterschiedlichem Typ sein können. Insofern ist eine Liste ein allgemeineres Konzept als ein Vektor, der nur aus gleichartigen Komponenten bestehen kann.

Die Herstellung einer Liste erfolgt entweder durch Aufzählung:

```
> x <- list(first=c(1,3),second="hallo",third=matrix(0,2,2))
> x
$first
[1] 1 3

$second
[1] "hallo"

$third
     [,1] [,2]
[1,]    0    0
[2,]    0    0
```

oder durch eine Schleife:

```
> y <- list(0)
> for (i in 1:3) y[i] <- list(matrix(i,2,2))
> names(y) <- c("first","second","third")
> y
$first
     [,1] [,2]
[1,]    1    1
[2,]    1    1

$second
     [,1] [,2]
[1,]    2    2
```

```
[2,]    2    2

$third
     [,1] [,2]
[1,]    3    3
[2,]    3    3
```

3.2 Dataframes

Dataframes sind spezielle Listen, die für die Darstellung von statistischen Datenmatrizen gedacht sind, bei denen die Spalten zwar gleich lang, aber von unterschiedlichem Typ sind.

Ein Dataframe ist eine Liste, deren Komponenten gleich lange Vektoren sind. Für Dataframes gibt es spezielle Methoden, von denen die Anzeige im Kommandofenster als erstes ins Auge fällt:

```
> x <- list(0)
> for (i in 1:3) x[i] <- list(rep(i,5))
> names(x) <- c("first","second","third")
> x
$first
[1] 1 1 1 1 1

$second
[1] 2 2 2 2 2

$third
[1] 3 3 3 3 3

> x <- as.data.frame(x)
> x
  first second third
1     1      2     3
2     1      2     3
3     1      2     3
4     1      2     3
5     1      2     3
```

Dataframes spielen sowohl für die statistischen Anwendungen als auch in Zusammenhang mit dem Datenimport nach R und Datenexport von R eine große Rolle.

Kapitel 4

Dateninput und Datenoutput

4.1 Export nach MS-Office

Für die Weiterverarbeitung von R-Ergebnissen ist es wünschenswert, die Ergebnisse in business-übliche Computerumgebungen zu bringen. Eine solche Computerumgebung sind beispielsweise die MS-Office Anwendungen Word und Excel.

Mit dem Datenaustausch zwischen R und Excel beschäftigt sich dieses Kapitel.

Zu Beginn sei klargestellt, daß es hier nicht darum geht, R als Rechen-Engine hinter Excel zu nutzen. Das kann man sehr effizient machen:

`http://www.ci.tuwien.ac.at/R/contrib/extra/dcom/`

`http://www.ci.tuwien.ac.at/R/contrib/extra/excel/`

Aber das ist nicht unser Thema.

Es geht vielmehr darum, bei gleichzeitiger Nutzung von Excel und R zwischen diesen beiden Anwendungen typische Daten auszutauschen. Wie kann man Ergebnisse, die mit R erzielt werden, in Präsentationsumgebungen, z.B. MS-Word oder MS-Excel, darstellen ?

Solange es sich nur um Text handelt, der aus dem R-Kommandofenster in eine andere Anwendung kopiert werden soll, ist das Problem einfach durch Copy und Paste (Markieren, Ctrl-C, Ctrl-V) zu lösen.

Interessanter und weniger trivial ist das Problem des Transports von Tabellenmaterial. Dieser Frage wollen wir uns hier zuwenden.

4.1.1 Windows Einstellungen

Grundsätzlich ist einmal festzuhalten, daß der Import von Tabellenmaterial nach MS-Anwendungen in der Regel möglich ist, wobei aber bestimmte Bedingungen erfüllt sein müssen. Diese Bedingungen betreffen vor allem die Trennung der Felder der Tabelle durch bestimmte Trennzeichen.

Wir gehen im folgenden davon aus, daß die Ländereinstellung des Windows-Betriebssystems den Punkt als Dezimaltrennzeichen und das Komma als Listentrennzeichen definiert. Bei einem englisch-

sprachigen MS-Windowssystems ist das die Voreinstellung, bei einem deutsch-sprachigen MS-Windowssystem muß dies nachträglich eingestellt werden (Start -> Einstellungen -> Systemsteuerung -> Ländereinstellungen -> Zahlen).

4.1.2 Kleine Tabellen exportieren

Für den eigentlichen Transportvorgang muß man in drei Schritten vorgehen:

1. Die Tabelle, die von R aus nach Word oder Excel kopiert werden soll, muß zunächst im passenden Format im Kommandofenster ausgegeben werden. (Leider ist es nicht möglich, von R aus direkt in das Windows-Clipboard zu schreiben.)

2. Anschließend wird die richtig formatierte Tabelle nach Word oder Excel kopiert.

3. Schließlich muß Word oder Excel wissen, daß es sich um eine Tabelle handelt.

Den ersten Schritt illustrieren wir an einem Beispiel. Zunächst erzeugen wir eine Matrix.

```
> a <- round(matrix(rnorm(16),nrow=4,ncol=4),3)
> a
      [,1]   [,2]   [,3]   [,4]
[1,] 1.891  0.381  0.951  2.500
[2,] 0.277  1.311 -0.247 -0.662
[3,] 0.179 -1.398 -0.695 -0.109
[4,] 0.601  0.489  1.174 -0.544
```

Um diese Matrix ins passende Format zu bringen, verwenden wir den Befehl:

```
>  write.table(a,quote=FALSE,sep=",",col.names=NA)
```

Dadurch erscheint im Kommandofenster von R die Matrix im Textformat, dh. als Textzeilen, deren Felder durch Kommas getrennt sind:

```
,X1,X2,X3,X4
1,1.891,0.381,0.951,2.5
2,0.277,1.311,-0.247,-0.662
3,0.179,-1.398,-0.695,-0.109
4,0.601,0.489,1.174,-0.544
```

In dieser Form kann die Tabelle mit Copy und Paste nach Word oder Excel kopiert werden. Das war dann der zweite Schritt des Vorgangs.

Nun muß noch Word oder Excel wissen, daß der kopierte Text als Tabelle zu formatieren ist. Hier reagieren Word und Excel unterschiedlich.

Excel macht es nach einem Neustart beim ersten Mal vermutlich falsch und schreibt jede Zeile des kopierten Materials in eine einzige Zelle. Es reicht aber aus, Excel bei diesem ersten Mal zu sagen, daß es sich dabei um ein Mißverständnis handelt. Wir markieren das kopierte Textmaterial, das sich in einer einzigen Spalte befindet. Unter dem Menupunkt

Daten -> Text in Spalten

erscheint ein Textkonvertierungs-Assistent, dem wir mitteilen, daß wir Kommas als Trennzeichen verwenden. Der Assistent formatiert dann unsere Tabelle richtig. Jeder weitere Kopiervorgang wird

dann sofort richtig durchgeführt.

Bei Word ist es ein wenig anders. Hier muß bei jedem Kopiervorgang das kopierte Textmaterial in eine Tabelle umgewandelt werden. Der entsprechende Menupunkt lautet

Tabelle -> Umwandeln -> Text in Tabelle.

4.1.3 Große Tabellen exportieren

Bei sehr großen Tabellen reicht das R-Kommandofenster nicht aus, um die passend formatierte Tabelle vollständig anzuzeigen. Man kann sie dann nicht markieren und daher schon gar nicht kopieren.

Ein Ausweg besteht darin, die passend formatierte Tabelle in eine Textdatei auszugeben, und von dort aus nach Excel einzulesen. (Nach Word wird man keine großen Tabellen transportieren.)

Nennen wir die Textdatei, die wir zur Auslagerung verwenden:

```
C:\Rtemp.txt
```

Dann erfolgt die Ausgabe der Matrix **a** dorthin durch:

```
> write.table(a,file="C:/Rtemp.txt",quote=FALSE,sep=",",col.names=NA)
```

Von Excel aus importiert man die Tabelle durch den Menupunkt:

Daten -> Externe Daten -> Textdatei importieren.

4.1.4 Diagramme exportieren

Diagramme lassen sich leicht nach Word oder Excel exportieren. Wenn man den rechten Mausknopf drückt, während sich die Maus im Grafikfenster befindet, erscheint ein kontextsensitives Menu, das das Kopieren der Grafik anbietet. Das Windows-spezifische Vektorformat ist **metafile**.

Das Kopieren als **metafile** speichert die Grafik ins Clipboard von Windows, von wo aus die Grafik durch Paste (Ctrl-V) in jede Office-Anwendung eingebettet werden kann.

Folgende Hinweise sind zu beachten, um brauchbare Ergebnisse zu erzielen.

Wenn man eine eingebettete Grafik in Windows vergrößert oder verkleinert, dann ändert sich die Beschriftungsgröße im gleichen Maßstab mit. Da R die Grafiken zunächst relativ groß anzeigt, würde eine in der ursprünglichen Größe kopierte Grafik nachträglich verkleinert werden müssen. Dadurch wird aber die Beschriftung der Grafik meist unleserlich.

R hat aber den großen Vorteil, daß beim Vergrößern und Verkleinern des Grafikfensters in R selbst die Beschriftungsgröße stabil bleibt. Daher sollte man bereits von R aus das Grafikfenster so verkleinern, daß es annähernd in der endgültigen Größe ist. Wenn man das tut, dann hat man nach dem Kopieren in eine Office-Anwendung eine Grafik mit einer leserlichen Beschriftung.

4.2 Textdateien importieren

Daten können aus Textdateien sehr leicht nach R importiert werden. Zu diesem Zweck steht der Befehl **read.table** mit seinen Varianten zu Verfügung (siehe Help-System).

Einige Hinweise sind aber zu beachten:

- Headers (Spaltenüberschriften) dürfen keine Umlaute enthalten. Wenn Umlaute auftreten, dann ist R nicht in der Lage, die Anzahl der Felder pro Zeile korrekt festzustellen.
- R versucht automatisch festzustellen, ob Headers und/oder Rownames existieren. Falls Headers, aber keine Rownames existieren, muß **headers=TRUE** gesetzt werden.
- Das verwendete Dezimaltrennzeichen ist in der Voreinstellung **dec=”.”**. Die Verwendung des Kommas muß angegeben werden. Zahlen mit 1000-Trennzeichen können nicht korrekt eingelesen werden.
- Das Missing-Value Symbol muß (z.B. durch **na.strings=”=NV()”**) angegeben werden. Wenn man das unterläßt, dann werden numerische Spalten mit Missing-Values als String-Spalten interpretiert.

Um Daten von Excel nach R zu transportieren, müssen sie in einer Textdatei zwischengelagert werden.

4.3 Tabellenspalten stacken

Ein häufiges Problem besteht darin, daß statistische Daten, die mit R untersucht werden sollen, in einer Textdatei als Kreuztabelle angeordnet sind. Solche Tabellen müssen in eine Datenmatrix umgewandelt werden. Dazu dient der Befehl **stack**.

Wir erklären das Problem und den Vorgang am besten an einem Beispiel.

Der Datensatz

```
> xx <- read.table("Bildung.txt",na.strings="#NV",header=TRUE)
> xx
    Schultyp Jahrgang Geschlecht Burgenland Karnten ...
1  Insgesamt     1960          z         98     687
2  Insgesamt     1960          w         NA      NA
3  Insgesamt     1970          z        445    1220
4  Insgesamt     1970          w        134     443
5  Insgesamt     1980          z        879    2346
6  Insgesamt     1980          w        423    1273
7  Insgesamt     1996          z       1189    2760
8  Insgesamt     1996          w        673    1482
...
```

enthält die Anzahl der Reifeprüfungen an Schulen, gegliedert unter anderem nach Bundesländern. Wir wollen die Variable **Bundesland** einführen. Der neue Datensatz soll daher fünf Variable enthalten, wobei die ersten drei bestehen bleiben. Die beiden letzten entstehen durch „stacken” der Bundesländervariablen:

```
p <- 3
n <- 48
m <- 13
zz <- NULL
for (i in 1:p) zz[i] <- list(rep(xx[,i],10))
names(zz) <- names(xx)[1:p]
zz[(p+1):(p+2)] <- stack(xx,select=(p+1):m)
names(zz)[p+1] <- "Anzahl"
names(zz)[p+2] <- "Bundesland"
zz <- as.data.frame(zz)
```

Wir erhalten so:

```
> zz
     Schultyp Jahrgang Geschlecht Anzahl       Bundesland
1   Insgesamt     1960          z     98       Burgenland
2   Insgesamt     1960          w     NA       Burgenland
3   Insgesamt     1970          z    445       Burgenland
4   Insgesamt     1970          w    134       Burgenland
5   Insgesamt     1980          z    879       Burgenland
6   Insgesamt     1980          w    423       Burgenland
7   Insgesamt     1996          z   1189       Burgenland
8   Insgesamt     1996          w    673       Burgenland
...
92        HAK     1996          w    322          Karnten
93        HAK     1997          z    520          Karnten
94        HAK     1997          w    340          Karnten
95        HAK     1998          z    585          Karnten
96        HAK     1998          w    364          Karnten
97  Insgesamt     1960          z   1552 Niederosterreich
98  Insgesamt     1960          w     NA Niederosterreich
99  Insgesamt     1970          z   2434 Niederosterreich
100 Insgesamt     1970          w    770 Niederosterreich
...
```

Kapitel 5

Elementare Statistik mit R

5.1 Technische Grundlagen

5.1.1 Das Arbeitsverzeichnis

Wir erstellen in Windows ein Verzeichnis mit dem Namen:

`C:\Workspace\R`

und ein Unterverzeichnis

`C:\Workspace\R\data`

In das Unterverzeichnis **data** kopieren wir die Datei:

`statlab.txt`

Die Datei **statlab.txt** enthält den Datensatz, den wir für unsere statistischen Beispiele verwenden werden. Nähere Informationen über diesen Datensatz finden sich unter

```
http://matrix.wu-wien.ac.at/homepage/fuerstudenten/unterlagen/
       datensaetze/statlab.html
```

R verwendet ein Arbeitsverzeichnis (Working Directory). Im Arbeitsverzeichnis sucht R nach Filenamen und legt Files beim Abspeichern ab. Bei einer Netzwerkinstallation wird das Arbeitsverzeichnis vom Netzwerkbetreuer festgelegt. Falls R auf dem PC lokal installiert ist, kann man das Arbeitsverzeichnis selbst festlegen. Um das Arbeitsverzeichnis festzulegen, geben wir ein:

```
> setwd("C:/Workspace/R")
```

Die Kontrolle ergibt:

```
> getwd()
[1] "C:\\Workspace\\R"
```

Soll das Arbeitsverzeichnis schon beim Start von R festgelegt werden, dann muß dieses Kommando im File **RProfile** enthalten sein. Dieses File befindet sich im R-Installationsverzeichnis unter **etc**.

Um den Inhalt des Arbeitsverzeichnisses anzuzeigen, verwenden wir:

```
> dir()
 ...
```

Mehr Information über diesen Befehl erhält man durch Befragen des Hilfesystems:

```
> ?dir
```

5.1.2 Der Suchpfad

Die Objekte, die für das R-System verfügbar, dh. unter ihrem Namen ansprechbar sind, sind in Listen gesammelt, die ihrerseits im sogenannten Suchpfad vereint sind:

```
> search()
[1] ".GlobalEnv"    "package:ctest" "Autoloads"     "package:base"
```

Der Suchpfad besteht also zu diesem Zeitpunkt aus insgesamt vier Objektlisten. Den Inhalt einer Liste kann man sich anschauen:

```
> objects("package:ctest")
 [1] "ansari.test"           "ansari.test.default"
 [3] "ansari.test.formula"   "bartlett.test"
 [5] "bartlett.test.default" "bartlett.test.formula"
 [7] "binom.test"            "chisq.test"
 [9] "cor.test"              "cor.test.default"
[11] "cor.test.formula"      "fisher.test"
[13] "fligner.test"          "fligner.test.default"
[15] "flig
...
```

Viele Funktionen und Befehle der R-Sprache sind in solchen Objektbibliotheken (packages) gesammelt. Nach dem Start von R sind nur **base** und **ctest** geladen. Benötigt man Funktionen, die sich in einer anderen Bibliothek befinden, muß man die betreffende Bibliothek laden:

```
> library(MASS)
> search()
[1] ".GlobalEnv"    "package:MASS"  "package:ctest"
[4] "Autoloads"     "package:base"
```

Die Biliothek **MASS** (benannt nach: W.N. Venables und B.D. Ripley, Modern Applied Statistics with S-Plus, Springer 1999), werden wir oft verwenden.

Um eine Bibliothek wieder loszuwerden, verwendet man:

```
> detach(package:MASS)
> search()
[1] ".GlobalEnv"    "package:ctest" "Autoloads"
[4] "package:base"
```

5.1.3 Benutzerdefinierte Objektbibliothek

Eine benutzerdefinierte Objektbibliothek wird im Arbeitsverzeichnis abgelegt. Solche Bibliotheken haben die Extension .r oder .R.

Wir verwenden im folgenden manchmal die Objektbibliothek **mylib.r**. Die Datei **mylib.r** ist eine Objektbibliothek, die die von uns benutzerdefinierten Makros enthält. Wir legen sie im Arbeitsverzeichnis ab. Sie wird durch

```
> source("mylib.r")
```

aktiviert. Nach der Aktivierung befinden sich die in der Bibliothek definierten Objekte im Workspace (.GlobalEnv) von R:

```
> objects(".GlobalEnv")
 [1] "cols"             "dynplot"          "edf"
 [4] "export.dataframe" "export.matrix"    "eye"
 [7] "frequencies"      "import.dataframe" "import.matrix"
[10] "mean"             "mylib"            "ones"
[13] "randn"            "randu"            "rows"
[16] "scatter1"         "statlab"          "svdd"
[19] "svdu"             "svdv"             "var"
[22] "weltraum"         "x"                "zeros"
[25] "zz"
```

(Dies ist der Stand der Objektbibliothek beim Verfassen dieser Zeilen.) Wenn die Objektbibliothek verändert worden ist, muß sie neu geladen werden.

Eine Objektbibliothek kann innerhalb der R-Benutzeroberfläche editiert werden:

```
> edit(file="mylib.r")
```

Dies ist aber nicht anzuraten, weil während der Öffnung des Editors das Kommandofenster nicht aktiviert werden kann. Es ist günstiger, den Editor außerhalb der R-Benutzeroberfläche zu starten. In diesem Fall kann zwischen dem Editorfenster und dem R-Kommandofenster beliebig gewechselt werden.

5.2 Statistische Daten

5.2.1 Datenframes

Statistische Daten werden in R meistens als Datenframes gespeichert. Dabei handelt es sich um Datenmatrizen mit Zeilennamen und Spaltennamen. Darüber hinaus können Spalten, die qualitative Variable enthalten, das Klassenattribut **factor** erhalten. Sie werden dann vom R-System anders behandelt als Spalten mit quantitativen Variablen.

5.2.2 Daten einlesen

Die einfachste Möglichkeit besteht darin, den Befehl **data(statlab)** zu verwenden. Um das zu tun, muß sich das File **statlab.txt** allerdings im Unterverzeichnis **data** des Arbeitsverzeichnisses befinden.

Es ist aber hilfreich zu wissen, wie man ein Textfile grundsätzlich einlesen kann. Dazu dienen die folgenden Hinweise.

Das File **statlab.txt** enthält Daten im Ascii-Format mit Spaltenüberschriften. In jeder Zeile sind die Daten durch Blanks getrennt. Dezimaltrennzeichen ist der Punkt.

Um den Datensatz **statlab** mit dem folgenden Befehl einlesen zu können, muß sich das File **statlab.txt** im Arbeitsverzeichnis befinden.

```
> statlab <- read.table(file="statlab.txt",header=TRUE,sep="",dec=".")
```

Wichtig: sep="" bedeutet, daß jede zusammenhängende Folge von Blanks als ein einziges Trennzeichen behandelt wird. Würden wir sep=" " eingeben, dann würde jedes einzelne Blank als Trennzeichen interpretiert.

Wir stellen nun die Anzahl der Zeilen und Spalten des Datensatzes fest:

```
> dim(statlab)
[1] 1296   34
```

sowie die Variablennamen:

```
> names(statlab)
 [1] "CODE"   "CBSEX"  "CBB"    "CBLGTH" "CBWGT"  "CBMO"
 [7] "CBD"    "CBHR"   "CTHGHT" "CTWGT"  "CTL"    "CTPEA"
[13] "CTRA"   "MBB"    "MBAG"   "MBWGT"  "MBO"    "MBSM"
[19] "MTHGHT" "MTWGT"  "MTE"    "MTO"    "MTSM"   "FBAG"
[25] "FBO"    "FBSM"   "FTHGHT" "FTWGT"  "FTE"    "FTO"
[31] "FTSM"   "FIB"    "FIT"    "FC"
```

Um den Datensatz zu Gänze anzuzeigen, verwenden wir

```
> data.entry(statlab)
```

Dadurch wird der Datensatz in einem Spreadsheet-ähnlichen Fenster angezeigt.

Wichtig: Wenn wir Daten bei dieser Anzeige ändern, hat das unmittelbar Auswirkung auf den gespeicherten Datensatz. Er wird dadurch verändert.

5.2.3 Attach

Nach dem Laden eines Datensatzes sind die einzelnen Variablen noch nicht unter ihrem Namen erreichbar:

```
> CBB
Error: Object "CBB" not found
```

Dies liegt daran, daß sich der Datensatz **statlab** noch nicht im Suchpfad befindet:

```
> search()
[1] ".GlobalEnv"    "package:ctest" "Autoloads"
[4] "package:base"
```

Um den Zugriff auf die einzelnen Variablen des Datensatzes zu vereinfachen, verwenden wir:

```
> attach(statlab)
```

Dieser Befehl fügt dem Suchpfad das Objekt *statlab* hinzu:

```
> search()
[1] ".GlobalEnv" "statlab" "package:ctest" "Autoloads"
[5] "package:base"
```

und nun sind alle Variablen des Datensatzes unter ihrem Spaltennamen dem System bekannt und erreichbar:

```
> CBB
   [1] 2 2 7 6 5 5 7 6 5 6 2 5 5 5 5 1 5 5 6 5 5 8 6 5 5 5 1 3 9 8 7 5
  [33] 5 1 5 5 7 5 5 9 5 5 5 5 5 6 5 6 5 9 5 5 2 3 2 5 2 5 5 8 1 3 6 5
  [65] 1 6 5 6 6 6 6 5 9 5 6 2 5 5 7 5 5 7 5 7 2 6 6 6 8 9 5 2 5 6 7 6
  ...
```

Anzeigen und Filtern

Man kann den gesamten Datenframe durch das Kommando **statlab** anzeigen lassen. Bei großen Datensätzen ist das aber unübersichtlich.

Einzelne Variablenspalten lassen sich durch ihren Namen aufrufen:

```
> MBWGT
   [1] 119 130 134 135 130 104 145 102 128 145  99 100 145 112 120 115
  [17] 117 125 100 122 128 106 135 129 105  97 113 121 140 138 140 104
  [33]  97 122 123 110 100 110 125  95 130 109 145 102 102 130 110 167
  ...
```

Das gleiche gilt für einzelne Datenzeilen:

```
> statlab["2",]
  CODE CBSEX CBB CBLGTH CBWGT CBMO CBD CBHR CTHGHT CTWGT CTL
2 1112     0   2     20   6.4    5   7   13   48.9    59   3
  CTPEA CTRA MBB MBAG MBWGT MBO MBSM MTHGHT MTWGT MTE MTO
2    74   34   6   17   130   0   20   62.8   159   3   1
  MTSM FBAG FBO FBSM FTHGHT FTWGT FTE FTO FTSM FIB FIT FC
2   10   23   6   11     65   130   1   6   20  40 175  6
```

Man kann auch mehrere Zeilen und Spalten auswählen:

```
> statlab[1:10,1:4]
   CODE CBSEX CBB CBLGTH
1  1111     0   2   20.0
2  1112     0   2   20.0
3  1113     0   7   19.8
```

```
4  1114     0   6   19.5
5  1115     0   5   19.5
6  1116     0   5   22.0
7  1121     0   7   21.0
8  1122     0   6   20.5
9  1123     0   5   21.5
10 1124     0   6   20.5
```

```
> statlab[c(4,12,44,110),c(2,5,12)]
    CBSEX CBWGT CTPEA
4       0   7.0    87
12      0   8.1    87
44      0   6.3    86
110     0   7.8    82
```

```
> statlab[1:10,c("CBSEX","CBWGT","CTPEA")]
   CBSEX CBWGT CTPEA
1      0   6.6    85
2      0   6.4    74
3      0   6.1    64
4      0   7.0    87
5      0   7.9    87
6      0   9.5    83
7      0   7.1    81
8      0   6.4    74
9      0   8.2    72
10     0   7.6    64
```

Mit logischen Ausdrücken kann man Zeilen mit bestimmten Eigenschaften gezielt auswählen:

```
> sum((CBB==3)&(MBB==2))
[1] 2
> statlab[(CBB==3)&(MBB==2),]
    CODE CBSEX CBB CBLGTH CBWGT CBMO CBD CBHR CTHGHT CTWGT CTL CTPEA
28  1154     0   3   20.3   7.3    5   4   14   51.3    58   1    80
542 3412     0   3   21.0   8.1    6   2   13   52.9    67   3    70
    CTRA MBB MBAG MBWGT MBO MBSM MTHGHT MTWGT MTE MTO MTSM FBAG FBO
28    45   2   20   121   1    0   62.0   132   2   0   -1   21   4
542   33   2   35   137   0    1   62.9   144   4   1    0   36   1
    FBSM FTHGHT FTWGT FTE FTO FTSM FIB FIT FC
28    24   71.0   165   2   4    0  54 114  6
542   25   70.5   158   4   0    0  31 122  4
```

Eine andere, sehr komfortable Möglichkeit, um Teile von Datenframes auszuwählen, bietet **subset**. Mehr Information darüber im Hilfesystem.

5.2.4 Demo-Datensätze in R

In R wird eine große Anzahl von Datensätzen zu Verfügung gestellt. Eine Liste aller Datensätze im Package **base** erhält man

```
> data()
```

und die Liste aller Datensätze:

```
> data(package = .packages(all.available = TRUE))
```

In einem neuen Fenster erscheint dann die Liste aller Datensätze. Will man die Story zu einem bestimmten Datensatz, z.B. „USArrests", lesen, dann gibt man ein:

```
> help("USArrests")
```

In einem neuen Fenster erscheint die Beschreibung des Datensatzes. Das Einlesen eines Demo-Datensatzes erfolgt besonders einfach durch:

```
> data(USArrests)
> USArrests
               Murder Assault UrbanPop Rape
Alabama          13.2     236       58 21.2
Alaska           10.0     263       48 44.5
Arizona           8.1     294       80 31.0
...
```

Wichtig: Manche der Demo-Datensätze enthalten nicht Rohdaten, sondern Tabellen.

Es ist günstig, die eigenen Datensätze, mit denen man arbeitet, in einem Unterverzeichnis **data** des Arbeitsverzeichnisses abzulegen. Die Datensätze sind dann genauso wie die vorgefertigten Datensätze durch

```
> data(statlab)
> attach(statlab)
> objects("statlab")
 [1] "CBB"    "CBD"    "CBHR"   "CBLGTH" "CBMO"   "CBSEX"
 [7] "CBWGT"  "CODE"   "CTHGHT" "CTL"    "CTPEA"  "CTRA"
[13] "CTWGT"  "FBAG"   "FBO"    "FBSM"   "FC"     "FIB"
[19] "FIT"    "FTE"    "FTHGHT" "FTO"    "FTSM"   "FTWGT"
[25] "MBAG"   "MBB"    "MBO"    "MBSM"   "MBWGT"  "MTE"
[31] "MTHGHT" "MTO"    "MTSM"   "MTWGT"
```

verfügbar zu machen.

5.2.5 Anhang: Objekte und Attribute

Der Bezeichner **statlab** bezeichnet jetzt ein R-Objekt. Jedes R-Objekt hat mehrere sogenannte Attribute. Wichtige Attribute, die jedes R-Objekt besitzt, sind Typ und Länge, die man durch **typeof** und **length** anzeigen kann:

```
> typeof(statlab)
[1] "list"
```

Es handelt sich um eine sogenannte Liste. Ein Liste hat eine Länge:

```
> length(statlab)
[1] 34
```

Im vorliegenden Fall handelt es sich um eine Liste, die aus Datenspalten besteht. Die Länge der Liste ist also hier die Anzahl der Spalten. Darüber hinaus kann es noch andere Attribute geben. Das ist dann von Objekt zu Objekt verschieden. Ein Datenframe z.B. hat noch die Attribute **class**, **names** und **row.names**. Man kann sich alle zusätzlichen Attribute eines Objekts verschaffen durch:

```
> attributes(statlab)
$names
 [1] "CODE"   "CBSEX"  "CBB"    "CBLGTH" "CBWGT"  "CBMO"   "CBD"
 [8] "CBHR"   "CTHGHT" "CTWGT"  "CTL"    "CTPEA"  "CTRA"   "MBB"
[15] "MBAG"   "MBWGT"  "MBO"    "MBSM"   "MTHGHT" "MTWGT"  "MTE"
[22] "MTO"    "MTSM"   "FBAG"   "FBO"    "FBSM"   "FTHGHT" "FTWGT"
[29] "FTE"    "FTO"    "FTSM"   "FIB"    "FIT"    "FC"

$class
[1] "data.frame"

$row.names
   [1] "1"    "2"    "3"    "4"    "5"    "6"    "7"    "8"    "9"
  [10] "10"   "11"   "12"   "13"   "14"   "15"   "16"   "17"   "18"
  [19] "19"   "20"   "21"   "22"   "23"   "24"   "25"   "26"   "27"
  ...
```

Attribute kann man auch direkt abfragen:

```
> attr(statlab,"names")
 [1] "CODE"   "CBSEX"  "CBB"    "CBLGTH" "CBWGT"  "CBMO"   "CBD"
 [8] "CBHR"   "CTHGHT" "CTWGT"  "CTL"    "CTPEA"  "CTRA"   "MBB"
[15] "MBAG"   "MBWGT"  "MBO"    "MBSM"   "MTHGHT" "MTWGT"  "MTE"
[22] "MTO"    "MTSM"   "FBAG"   "FBO"    "FBSM"   "FTHGHT" "FTWGT"
[29] "FTE"    "FTO"    "FTSM"   "FIB"    "FIT"    "FC"
```

5.3 Analyse von univariaten Datensätzen

5.3.1 Qualitative Variable - Faktoren

Für statistische Analysen ist grundsätzlich der Variablentyp zu beachten. Die allereinfachste Unterscheidung erfolgt zwischen qualitativen und quantitativen Variablen. In R werden qualitative Variablen als Faktoren bezeichnet.

Beim Einlesen eines Datenframes betrachtet R (als Voreinstellung) jede numerische Datenspalte als quantitativ, und jede nichtnumerische Datenspalte als Faktor.

Wir wollen die Variable CBB untersuchen. Das ist eine numerisch kodierte qualitative Variable. R glaubt zunächst (auf Grund der numerischen Kodierung), daß es sich um eine quantitative Variable handelt. Das kann man unter anderem daraus sehen, daß mit dem Befehl

```
> summary(CBB)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   5.531   6.000   9.000
```

eine Zusammenfassung der Datenliste erstellt wird, die sich für qualitative Variable nicht eignet.

Wenn numerisch kodierte Datenspalten als Faktoren behandelt werden sollen, dann muß das dem R-System mitgeteilt werden:

```
> CBB <- factor(CBB)
> summary(CBB)
  1   2   3   4   5   6   7   8   9
 61  64  22   6 481 376 145  56  85
```

Diese Häufigkeitstabelle ist nun eine angemessene Zusammenfassung der Datenliste einer qualitativen Variablen.

Für Datensätze, die oft gebraucht werden, ist es zweckmäßig, den Einleseprozeß so zu automatisieren, daß qualitative Variable sofort in Faktoren umgewandelt werden. Wenn man im Unterverzeichnis **data** die Datei folgende Datei **statlab.R**

```
statlab <- read.table("statlab.txt",header=TRUE)
statlab$CODE <- as.factor(statlab$CODE)
statlab$CBSEX <- as.factor(statlab$CBSEX)
statlab$CBB <- as.factor(statlab$CBB)
statlab$CBMO <- as.factor(statlab$CBMO)
statlab$CBD <- as.factor(statlab$CBD)
statlab$CBHR <- as.factor(statlab$CBHR)
statlab$CTL <- as.factor(statlab$CTL)
statlab$MBB <- as.factor(statlab$MBB)
statlab$MBO <- as.factor(statlab$MBO)
statlab$MBSM <- as.factor(statlab$MBSM)
statlab$MTE <- as.factor(statlab$MTE)
statlab$MTO <- as.factor(statlab$MTO)
statlab$MTSM <- as.factor(statlab$MTSM)
statlab$FBO <- as.factor(statlab$FBO)
statlab$FBSM <- as.factor(statlab$FBSM)
statlab$FTE <- as.factor(statlab$FTE)
statlab$FTO <- as.factor(statlab$FTO)
statlab$FTSM <- as.factor(statlab$FTSM)
statlab$FC <- as.factor(statlab$FC)
attach(statlab)
print(objects("statlab"))
```

ablegt, werden durch das Kommando

```
> data(statlab)
```

die Daten mit den korrekten Datentypen eingelesen und attached.

Tabellen

Die Datenliste eines Faktors wird beschrieben durch die Wertemenge

```
> a <- levels(CBB)
> a
```

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

und die Häufigkeiten:

```
> freq <- table(CBB)
> freq
CBB
  1   2   3   4   5   6   7   8   9
 61  64  22   6 481 376 145  56  85
```

Das Objekt **freq** ist eine einfache Liste und wird deshalb von R als Zeile angezeigt. Um die Tabelle als Spalte anzuzeigen, müssen wir sie in einen Spaltenvektor (Matrix) umwandeln:

```
> cbind(freq)
    freq
1  61
2  64
3  22
4   6
5 481
6 376
7 145
8  56
9  85
```

Die folgende Kommandosequenz stellt eine ausführliche Häufigkeitstabelle her:

```
> n <- length(MBAG)
> freq <- table(MBAG)
> cumfreq <- cumsum(freq)
> perc <- freq/n
> cumperc <- cumsum(perc)
> cbind(freq,cumfreq,perc,cumperc)
   freq cumfreq        perc     cumperc
13    1       1 0.000771605 0.000771605
15    1       2 0.000771605 0.001543210
17    6       8 0.004629630 0.006172840
18   13      21 0.010030864 0.016203704
19   25      46 0.019290123 0.035493827
20   50      96 0.038580247 0.074074074
...
```

oder:

```
> round(cbind(freq,cumfreq,perc,cumperc),3)
   freq cumfreq  perc cumperc
13    1       1 0.001   0.001
15    1       2 0.001   0.002
17    6       8 0.005   0.006
18   13      21 0.010   0.016
19   25      46 0.019   0.035
20   50      96 0.039   0.074
```

```
21   65     161 0.050   0.124
...
```

Diagramme

Die graphische Darstellung von Datenlisten einer qualitativen Variablen erfolgt als Säulendiagramm

```
> barplot(table(CBB))
```

oder als Kreisdiagramm

```
> piechart(x,col=rainbow(length(x)),radius=0.9)
```

Kodierung

Eine Kodierung eines Faktors kann sich darauf beschränken, die ursprünglichen Werte nur unzubenennen, oder aber eine neue Gruppierung (Partition) der Werte vorzunehmen.

Wenn nur umbenannt werden soll, dann ist die Sache sehr einfach. Entweder man gibt die neuen Labels an:

```
> z <- factor(CBSEX,labels=c("m","w"))
> z[1:5]
[1] m m m m m
Levels:  m w
> z[801:805]
[1] w w w w w
Levels:  m w
```

oder man definiert einen Namen:

```
> z <- factor(CBB,label="blood")
> z[1:10]
 [1] blood2 blood2 blood7 blood6 blood5 blood5 blood7 blood6 blood5
[10] blood6
Levels:  blood1 blood2 blood3 blood4 blood5 blood6 blood7 blood8 blood9
```

Manchmal will man aber die Werte eines Faktors neu gruppieren. Betrachten wir die Variable MTSM. Hier ist es sinnvoll, die Werte -1 und 0 zu belassen, aber die Anzahl der Zigaretten, die im Datensatz von 1 bis 60 schwankt, kompakter zu kodieren.

Wir definieren Teilungspunkte:

```
> br <- c(-2,-1,seq(0,60,10))
> br
[1] -2 -1  0 10 20 30 40 50 60
```

Um einen Faktor neu zu kodieren, muß er zuerst in eine numerische Datenliste umgewandelt werden:

```
> temp <- as.numeric(as.character(MTSM))
> mode(temp)
[1] "numeric"
```

Nun kodieren wir um:

```
> x <- cut(temp,br)
> cbind(table(x))
        [,1]
(-2,-1]  678
(-1,0]   206
(0,10]   138
(10,20]  191
(20,30]   45
(30,40]   35
(40,50]    2
(50,60]    1
```

Um mit Zahlen zu kodieren, müssen wir Labels einführen:

```
> x <- cut(temp,br,labels=-1:6)
> cbind(table(x))
   [,1]
-1  678
0   206
1   138
2   191
3    45
4    35
5     2
6     1
```

Die Kodierung mit dem Befehl **cut** liefert als Default einen Faktor:

```
> class(x)
[1] "factor"
```

5.3.2 Quantitative Variable

Stem and Leaf Plot

Einen ersten Überblick über die Werte einer quantitativen Datenliste erhält man angeblich durch den Stem and Leaf-Plot. Diese Darstellungsmethode eignet sich aber nur für kleinere Datensätze. Wir illustrieren sie daher anhand eines einfachen künstlich generierten Datensatzes:

```
> x <- rnorm(50)
> x
 [1] -0.17051255 -0.53421335  0.82368839  1.20435144 -0.05923893
 [6]  1.12844936 -0.14124691  0.42881220 -0.58610728  0.94565105
 ...
> stem(x)

  The decimal point is at the |
```

```
  -2 | 4
  -1 |
  -1 | 444
  -0 | 877666655
  -0 | 3322221110
   0 | 12233444
   0 | 5678889
   1 | 00122234
   1 | 9
   2 | 03
   2 | 5
```

Tabellen

Häufigkeitstabellen sind bei quantitativen Daten nur dann brauchbar, wenn der Umfang der Wertemenge deutlich kleiner ist als der Umfang der Datenliste. Das ist z.B. der Fall bei MBAG:

```
> table(MBAG)
MBAG
13 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
 1  1  6 13 25 50 65 68 75 88 96 93 84 75 66 67 50 56 40 47 40 44 34
38 39 40 41 42 43 44 45 46
19 36 16 15 11  8  4  2  1
```

Man kann Häufigkeitstabellen wie gewohnt herstellen:

```
> n <- length(MBAG)
> x <- table(MBAG)
> cbind(x,cumsum(x),x/n,cumsum(x)/n)
    x
13  1    1 0.000771605 0.000771605
15  1    2 0.000771605 0.001543210
17  6    8 0.004629630 0.006172840
18 13   21 0.010030864 0.016203704
19 25   46 0.019290123 0.035493827
20 50   96 0.038580247 0.074074074
21 65  161 0.050154321 0.124228395
22 68  229 0.052469136 0.176697531
...
```

Diagramme

Trotz eines erheblichen Komprimierungseffekts, der durch die Häufigkeitstabelle erreicht wird, ist die Übersichtlichkeit nicht berauschend. Eine bessere Darstellung erhält man durch Diagramme.

Das einfachste Diagramm einer quantitativen Datenliste ist das eindimensionale Streudiagramm. Man berechnet zunächst die Wertemenge der Datenliste

```
> a <- sort(unique(MBAG))
```

und die Häufigkeiten

```
> b <- table(MBAG)
```

und zeichnet anschließend den Plot

```
> plot(a,b,type="h")
```

Das Bild, das dieser Diagrammtyp liefert, hängt stark davon ab, wie häufig die Bindungen (mehrfachen Daten) in der Datenliste sind, z.B.:

```
> x <- rnorm(100)
> plot(sort(unique(x)),table(x),type="h")
```

Der zweite wichtige Diagrammtyp für quantitative Datenlisten ist die EDF (empirische Verteilungsfunktion):

```
> plot(sort(unique(x)),cumsum(table(x)),type="s")
```

Dabei handelt es sich um eine graphische Darstellung der Summenhäufigkeiten.

R bietet die EDF auch als eingebaute Funktion an:

```
> library(stepfun)
> f <- ecdf(MBAG)
> plot(f)
```

Die bisher behandelten Methoden stellen die gesamte Information dar, die im Datensatz enthalten ist, mit Ausnahme der Reihenfolge der Daten. Alle weiteren Methoden reduzieren die Information.

Maßzahlen

Wir wollen nun Mittelwert, Varianz, Standardabweichung und Minimum bzw. Maximum von Datenlisten berechnen.

Für eine einzelne (univariate) Datenliste ist das ganz einfach:

```
> mean(MTWGT)
[1] 143.1026
> var(MTWGT)
[1] 785.9362
> sd(MTWGT)
[1] 28.03455
> max(MTWGT)
[1] 284
> min(MTWGT)
[1] 75
```

Oft ist es aber auch wichtig, für eine Datenmatrix die Maßzahlen für alle Datenspalten gleichzeitig zu erhalten. Dann leistet der Befehl **apply**:

```
> yy <- data.frame(MBAG,FBAG,FIB,FIT)
> apply(yy,2,mean)
     MBAG      FBAG       FIB       FIT
 28.27778  31.51080  73.83565 155.19444
```

```
> apply(yy,2,sd)
     MBAG      FBAG       FIB       FIT
 6.002573  6.660890 29.359731 68.243664
```

Standardscores berechnet man gemäß ihrer Definition

```
> s <- (FTHGHT-mean(FTHGHT))/sd(FTHGHT)
> mean(s)
[1] -4.772725e-14
> sd(s)
[1] 1
```

oder noch einfacher durch

```
> s <- scale(FTHGHT)
```

Extreme Daten lassen sich nun leicht aufspüren:

```
> which(abs(s)>3)
[1] 1039 1096 1126
> FTHGHT[abs(s)>3]
[1] 78.8 60.8 60.8
```

Wie standardisiert man eine Datenmatrix ?

```
> s <- scale(FTHGHT)
> yy <- scale(cbind(MBAG,FBAG,FIB,FIT))
> apply(yy,2,mean)
         MBAG          FBAG           FIB           FIT
-1.121874e-15  5.996575e-16 -2.111651e-16 -1.519703e-16
> apply(yy,2,sd)
MBAG FBAG  FIB  FIT
   1    1    1    1
```

Quantile

Quantile eines Datensatzes erhält man in zwei Schritten. Zunächst gibt man die Anteile an, für die man die Quantile berechnen will:

```
> p <- seq(from=0,to=1,by=1/10)
> p
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Anschließend berechnet man die Quantile:

```
> quantile(MBAG,p)
  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
  13   21   23   24   26   27   29   31   34   37   46
```

Ohne Angabe der Anteile erhält man die Quartile, dh. die 5-Point Summary der Daten:

```
> quantile(MBAG)
  0%  25%  50%  75% 100%
  13   24   27   32   46
```

Dies entspricht auch der voreingestellten Zusammenfassung von quantitativen Datenlisten:

```
> summary(MBAG)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  13.00   24.00   27.00   28.28   32.00   46.00
```

Wenn man die Quartile zeichnet, entsteht ein Boxplot:

```
> boxplot(MBAG,range=0)
```

Gruppierung

Wenn man eindimensionale quantitative Daten kodiert, verwendet man eine Intervalleinteilung. Man spricht dann von Gruppierung.

Für eine Gruppierung braucht man zunächst Teilungspunkte der Intervalle. Neun Teilungspunkte

```
> br <- seq(from=min(MBAG),to=max(MBAG),length=9)
> br
[1] 13.000 17.125 21.250 25.375 29.500 33.625 37.750 41.875 46.000
```

ergeben eine Gruppierung mit acht Intervallen:

```
> x <- cut(MBAG,br,include.lowest=TRUE)
> cbind(table(x))
            [,1]
(13,17.1]      8
(17.1,21.3]  153
(21.3,25.4]  327
(25.4,29.5]  318
(29.5,33.6]  213
(33.6,37.8]  165
(37.8,41.9]   86
(41.9,46]     26
```

Die neu kodierte Variable hat als Werte Strings:

```
> attr(x,"levels")
[1] "(13,17.1]"   "(17.1,21.3]" "(21.3,25.4]" "(25.4,29.5]" "(29.5,33.6]"
[6] "(33.6,37.8]" "(37.8,41.9]" "(41.9,46]"
```

Will man andere Labels, dann muß man sie angeben:

```
> x <- cut(MBAG,br,include.lowest=TRUE,labels=1:8)
> cbind(table(x))
  [,1]
1    8
2  153
3  327
4  318
5  213
6  165
7   86
```

```
8   26
```

Eine graphische Darstellung der gruppierten Daten erhält man durch:

```
> h <- as.numeric(as.character((factor(MBAG,labels=table(MBAG)))))
> plot(MBAG,h,type="h",col=num(x))
```

Man kann auch an Gruppierungen mit unterschiedlichen Intervallbreiten interessiert sein. Sollen die Häufigkeiten etwa gleich groß sein, dann kann man Quantile als Teilungspunkte verwenden:

```
> br <- quantile(FTHGHT,(0:10)/10)
> br
   0%   10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
60.80 66.40 68.00 68.65 69.50 70.15 71.00 71.85 72.00 74.00 78.80
```

Allerdings können die Häufigkeiten dann trotzdem sehr unterschiedlich sein, wenn der Datensatz zahlreiche Bindungen aufweist:

```
> x <- cut(FTHGHT,br,include.lowest=TRUE)
> cbind(table(x))
            [,1]
(60.8,66.4]  131
(66.4,68]    219
(68,68.7]     39
(68.7,69.5]  133
(69.5,70.2]  126
(70.2,71]    209
(71,71.8]     50
(71.8,72]    136
(72,74]      174
(74,78.8]     79
```

Um eine Kodierung mit fast gleichhäufigen Werten zu erzwingen, kann man die Funktion **my.codes** aus der Objektbibliothek verwenden. Die Gleichmäßigkeit der Häufigkeiten wird hier durch zufällige Zuordnung an den Intervallgrenzen erreicht:

```
> a <- my.codes(FTHGHT,10)
> summary(a)
  1   2   3   4   5   6   7   8   9  10
129 130 130 129 130 130 129 130 130 129
> boxplot(FTHGHT~a,range=0)
```

Histogramme

Ein Histogramm ist ein Diagramm, bei dem die Dichte einer gruppierten Häufigkeitsverteilung dargestellt wird. In R wird das durch die Funktion **hist** mit der Option **freq**=FALSE erreicht:

```
> br <- seq(from=min(MBAG),to=max(MBAG),length=9)
> hist(MBAG,br,freq=FALSE)
```

5.3.3 Die Normalverteilung

Die Dichte und die Verteilungsfunktion der Standardnormalverteilung erhält man durch:

```
> x <- seq(from=-4,to=4,by=0.05)
> plot(x,dnorm(x),type="l")
> plot(x,pnorm(x),type="l")
```

Versuchen Sie auch einmal:

```
> plot(dnorm,from=-4,to=4)
> plot(pnorm,from=-4,to=4)
```

Quantile der Standardnormalverteilung berechnet man mit

```
> qnorm(0.01)
[1] -2.326348
```

Man kann auch Mittelwert und Standardabweichung festlegen.

5.3.4 Verteilungsanalyse

Verteilungsnalyse erfolgt entweder durch Diagramme oder durch Maßzahlen.

Die einfachsten Diagramme haben wir schon kennengelernt. Histogramme stellen die Dichte der Daten graphisch dar. Das kann man aber besser machen durch sogenannte Dichteschätzer:

```
> plot(density(MBAG))
```

Um zu sehen, wie gut oder wie schlecht ein Histogramm die Dichte schätzt, überlagert man ein Histogramm mit einem Dichteschätzer:

```
> hist(MBAG,freq=FALSE)
> lines(density(MBAG))
```

Die erste Frage, die man sich bei einer Verteilungsanalyse stellt, ist die nach den Unterschieden zu einer Normalverteilung. Zu diesem Zweck könnte man einen Dichteschätzer der Daten mit der Dichtefunktion der Normalverteilung vergleichen. Das hat aber nur Sinn, wenn man den gleichen Mittelwert und die gleiche Varianz verwendet:

```
> m <- mean(MBAG)
> s <- sqrt(var(MBAG))
> x <- seq(from=m-3*s,to=m+3*s,length=300)
> y <- dnorm(x,mean=m,sd=s)
> plot(x,y,type='l')
> lines(density(MBAG))
```

Eine genauere Methode zur graphischen Beurteilung der Übereinstimmung mit einer Normalverteilung ist ein Normalplot:

```
> qqnorm(CBWGT)
> qqline(CBWGT)
```

Man kann die Abweichung von der Normalverteilung auch auf Signifikanz abtesten. Der auf dem Normalplot beruhenden Test ist der Shapiro-Wilk Test:

```
> shapiro.test(CBWGT)

        Shapiro-Wilk normality test

data:  CBWGT
W = 0.9868, p-value = 1.935e-09
```

5.4 Univariate Inferenzmethoden

5.4.1 Wahrscheinlichkeiten

R bietet für das Testen und Schätzen von Wahrscheinlichkeiten und Anteilen zwei Methoden an: **prop.test** und **binom.test**.

prop.test verwendet die Näherungsmethoden, die auf der NV-Approximation beruhen. **binom.test** ist ein exaktes Verfahren, das auf dem Modell der Binomialverteilung beruht, also bei Ziehungsexperimenten auf dem Modell des Ziehens mit Zurücklegen. Direkte Methoden für das Ziehen ohne Zurücklegen werden zur Zeit nicht angeboten.

Wir besprechen die Funktion **binom.test**.

Bei der Variablen CBSEX beträgt der Anteil der männlichen Geburten:

```
> p.hyp <- mean(CBSEX==1)
> p.hyp
[1] 0.5
```

Wir ziehen eine Stichprobe vom Umfang 20:

```
> x <- sample(CBSEX,20)
> x
 [1] 1 1 1 0 1 1 0 0 0 0 1 0 0 1 1 0 1 0 1 1
Levels:  0 1
```

Diese Stichprobe besitzt die Stichprobenhäufigkeit

```
> h <- sum(x==1)
> h
[1] 11
```

und den Stichprobenumfang

```
> n <- length(x)
> n
[1] 20
```

Wir wenden nun das R-Kommando **binom.test** an, um die Nullhypothese p=**p.hyp** zu testen:

```
> binom.test(h,n,p=p.hyp)
```

```
        Exact binomial test

data:  h and n
number of successes = 11, number of trials = 20, p-value = 0.8238
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.3152610 0.7694215
sample estimates:
probability of success
                  0.55
```

Der Output enthält sowohl die Informationen über das Ergebnis des Tests als auch ein Konfidenzintervall.

Man kann die voreingestellten Parameterwerte ändern. Mit dem Argument **alternative** (Werte: ”two.sided”, ”less”, ”greater”) legt man fest, ob die Fragestellung zweiseitig, oder einseitig ist. Mit **conf.level** legt man die Überdeckungswahrscheinlichkeit des Konfidenzintervalls fest.

```
> binom.test(h,n,0.4,alternative="greater",conf.level=0.99)

        Exact binomial test

data:  h and n
number of successes = 11, number of trials = 20, p-value = 0.1275
alternative hypothesis: true probability of success is greater than 0.4
99 percent confidence interval:
 0.2801000 1.0000000
sample estimates:
probability of success
                  0.55
```

Die einzelnen Komponenten des Textoutputs sind individuell ansprechbar. Hierzu muß zunächst das Ergbnis des R-Kommandos als Objekt abgespeichert werden:

```
> out <-  binom.test(h,n,0.4,alternative="greater",conf.level=0.99)
```

Das Objekt **out** hat dann die Attribute:

```
> attributes(out)
$names
[1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
[6] "null.value"  "alternative" "method"      "data.name"

$class
[1] "htest"
```

Das Objekt **out** ist eine Liste vom Klassentyp **htest**, und ihre Kompnenten haben die Namen, die in **names** angegeben sind. Diese Komponenten sind individuell ansprechbar:

```
> out$p.value
[1] 0.1275212
```

```
> out$conf.int[1:2]
[1] 0.2801000 1.0000000
> attr(out$conf.int,"conf.level")
[1] 0.99
```

5.4.2 Erwartungswerte

Der t-Test

Für Tests und Konfidenzintervalle betreffend einen Erwartungswert verwendet man das R-Kommando **t.test**.

Die Variable MBAG hat den Mittelwert

```
> m.hyp <- mean(MBAG)
> m.hyp
[1] 28.27778
```

Wir betrachten diesen Mittelwert als Mittelwert der Grundgesamtheit. Nun ziehen wir eine Stichprobe:

```
> x <- sample(MBAG,20)
> x
 [1] 22 30 28 28 26 29 36 32 27 28 27 22 28 27 36 31 38 19 29 18
```

Wir wenden nun das R-Kommando t.test an.

```
> t.test(x,mu=m.hyp)

         One Sample t-test

data:  x
t = -0.1949, df = 19, p-value = 0.8475
alternative hypothesis: true mean is not equal to 28.27778
95 percent confidence interval:
 25.60406 30.49594
sample estimates:
mean of x
    28.05
```

Die Voreinstellungen der Eingabeparameter können abgeändert werden. Um das zu tun und um die einzelnen Komponenten des Textoutputs einzeln anzusprechen, gehen wir so vor:

```
> out <- t.test(x,mu=25,conf.level=0.99,alternative="greater")
> attributes(out)
$names
[1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
[6] "null.value"  "alternative" "method"      "data.name"

$class
[1] "htest"
```

```
> out$p.value
[1] 0.008608161
> out$conf.int[1:2]
[1] 25.08233      Inf
> attr(out$conf.int,"conf.level")
[1] 0.99
```

Die ANOVA-Tabelle

Man kann die Überstimmung einer Datenliste mit einem vorgegebenen Wert (Nullhypothese) auch durch eine Streuungszerlegung analysieren.

Zunächst sei der Vergleichswert **m.hyp**=0.

Die in R anzuwendende Methode beruht auf linearen Modellen. Eine ausführliche Erklärung der nachfolgenden Kommandos würde hier aber zu weit führen.

Wir benötigen einen Datenvektor, der nur aus 1-en besteht:

```
> const <- rep(1,times=length(MBAG))
```

Anschließend fertigen wir ein lineares Modell für die zuerklärende Variable MBAG an, welches ausschließlich aus der Variablen e besteht und werten es aus:

```
> anova(lm(MBAG~0+const))
            Df  Sum Sq Mean Sq F value    Pr(>F)
const       1 1036324 1036324   28762 < 2.2e-16 ***
Residuals 1295   46660      36
---
Signif. codes:  0  ‘***’  0.001  ‘**’  0.01  ‘*’  0.05  ‘.’  0.1  ‘ ’  1
```

Natürlich ist die Anpassung dieses Modells sehr schlecht, denn m.hyp=0 liegt nicht im entferntesten in der Nähe des Mittelwertes von MBAG. Eine plausiblere Hypothese wäre m.hyp=28. Um die Anpassung dieses Wertes an die Datenliste zu analysieren, analysieren wir einfach die Differenzen MBAG-m.hyp:

```
> anova(lm(MBAG-28~0+e))
            Df Sum Sq Mean Sq F value  Pr(>F)
const       1    100     100  2.7754 0.09596 .
Residuals 1295  46660      36
---
Signif. codes:  0  ‘***’  0.001  ‘**’  0.01  ‘*’  0.05  ‘.’  0.1  ‘ ’  1
```

5.5 Analyse von bivariaten Datensätzen

5.5.1 Zwei quantitative Variable

Korrelation

Die Verteilung der Daten von zwei quantitativen Variablen wird in einem Streudiagramm graphisch dargestellt:

```
> plot(FTHGHT,FTWGT)
```

Das gleiche Ergebnis liefert der Befehl

```
> plot(~FTHGHT+FTWGT)
```

der die modellsprachliche Syntax von R verwendet. Über diese modellsprachlichen Möglichkeiten werden wir im folgenden immer mehr erfahren.

Den Korrelationskoeffizienten berechnet man durch:

```
> cor(FTHGHT,FTWGT)
[1] 0.5561648
```

Allerdings erhält man mehr Information, wenn man aus dem ctest-Package die Funktion cor.test verwendet:

```
> library(ctest)
> cor.test(FTHGHT,FTWGT,method="pearson")

        Pearson's product-moment correlation

data:  FTHGHT and FTWGT
t = 24.0731, df = 1294, p-value = < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
sample estimates:
      cor
0.5561648
```

Neben dem gewöhnlichen (Pearson) Korrelationskoeffizienten ist hier noch der Spearman'sche und der Kendall'sche Korrelationskoeffizient verfügbar:

```
> cor.test(FTHGHT,FTWGT,method="kendall")

        Kendall's rank correlation tau

data:  FTHGHT and FTWGT
z.tau = 22.3891, p-value = < 2.2e-16
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.4151721
```

Manchmal ist es interessant, sich Datensätze anzusehen, die für einen bestimmten Wert des Korrelationskoeffizienten typisch sind. Um einen normalverteilten Datensatz mit der Korrelation r=0.99 zu erzeugen, gehen wir so vor:

```
> r <- 0.99
> c <- matrix(c(1,r,r,1),2,2)
```

```
> c
     [,1] [,2]
[1,] 1.00 0.99
[2,] 0.99 1.00
> library(MASS)
> x <- mvrnorm(100,mu=c(0,0),Sigma=c)
> plot(x)
> cov(x)
          [,1]      [,2]
[1,] 0.7524042 0.7536942
[2,] 0.7536942 0.7755578
```

Einfache lineare Regression

Beim Regressionsproblem wird den beiden Variablen eine unterschiedliche Rolle zugewiesen. Eine der Variablen spielt die Rolle des Prädiktors, die andere ist die Responsevariable. Das Ziel besteht darin, die Werte der Responsevariablen mit Hilfe der Prädiktorvariablen vorherzusagen, und die Qualität dieser Vorhersage zu beurteilen.

Beim linearen Regressionsproblem geht es darum, ein lineares Erklärungsmodell zu schätzen, mit dem die Prognose erfolgen kann.

Es sei MBAG die Prädiktorvariable und CTPEA die Responsevariable. Ein lineares Modell wird angepaßt durch:

```
> fm<-lm(CTPEA~1+MBAG)
> summary(fm)

Call:
lm(formula = CTPEA ~ 1 + MBAG)

Residuals:
     Min       1Q   Median       3Q      Max
-26.3626  -6.8508   0.6274   6.2286  47.7010

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  72.4661     1.4021  51.685  < 2e-16 ***
MBAG          0.2341     0.0485   4.827 1.55e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 10.48 on 1294 degrees of freedom
Multiple R-Squared: 0.01769,    Adjusted R-squared: 0.01693
F-statistic:  23.3 on 1 and 1294 DF,  p-value: 1.552e-006
```

Die Koeffizienten der Regressionsgeraden kann man dann isolieren

```
> coefficients(fm)
(Intercept)        MBAG
```

```
 72.4661175   0.2341168
```

und die Regressionsgerade zeichnen:

```
> plot(MBAG,CTPEA)
> abline(coefficients(fm))
```

Eine Zusammenfassung der Prognosegenauigkeit in einer ANOVA-Tabelle erhält man durch:

```
> anova(fm)
Analysis of Variance Table

Response: CTPEA
            Df Sum Sq Mean Sq F value    Pr(>F)
MBAG         1   2557    2557  23.299 1.552e-06 ***
Residuals 1294 142039     110
---
Signif. codes:  0  ‘***’  0.001  ‘**’  0.01  ‘*’  0.05  ‘.’  0.1  ‘ ’  1
```

Man kann die einzelnen Teilergebnisse direkt ansprechen, z.B.:

```
> anova(fm)[1,2]
2557.466
> anova(fm)[2,2]
142038.9
```

Das in diesem Beispiel sehr niedrige Bestimmtheitsmaß

```
> summary(fm)$r.squared
[1] 0.01768693
```

zeigt, daß die Regressionsprognose eine kaum kleinere Varianz besitzt als die die Varianz der zu prognostizierenden Variablen:

```
> anova(fm)[2,3]
109.7673
> var(CTPEA)
[1] 111.6574
```

5.5.2 Eine quantitative und eine qualitative Variable

Grundsätzlich sind in diesem Fall zwei Fragestellungen denkbar.

Fall 1: Prädiktor qualitativ, Response quantitativ.

Fall 2: Prädiktor quantitativ, Response qualitativ.

Wir beschränken uns hier auf den Fall 1.

Deskriptiver Vergleich der Stichproben

Der qualitative Prädiktor definiert eine Gruppeneinteilung (Partition). Die quantitative Datenliste der Responsevariablen zerfällt so in getrennte Stichproben:

```
> x <- split(CTRA,MTE)
> x
$"0"
 [1] 32 25 17 21 22 17 21 26 39 24 17 14 31 15 35 13 14 28 14 22
[21] 23 24 32  9 20 20

$"1"
  [1] 34 29 19 37 34 34 32 24 28 12 31 26 25 34 27 28 37 46 28
 [20] 41 13 39 29 10 22 20 16 24 11 21 27 19 25 23 28 33 24 27
...
```

Statistische Kennzahlen der einzelnen Stichproben verschafft man sich durch:

```
> tapply(CTRA,MTE,mean)
       0        1        2        3        4
22.11538 26.00990 28.42915 31.95052 36.41581

> tapply(CTRA,MTE,var)
       0        1        2        3        4
55.38615 74.00990 87.64101 96.27170 80.20927

> cbind(mean=tapply(CTRA,MTE,mean),var=tapply(CTRA,MTE,var))
      mean      var
0 22.11538 55.38615
1 26.00990 74.00990
2 28.42915 87.64101
3 31.95052 96.27170
4 36.41581 80.20927
```

Ein graphischer Vergleich der Stichproben ist möglich durch:

```
> boxplot(CTRA~MTE,range=0)
```

ANOVA-Tabelle

Ein Vergleich der Mittelwerte und Erwartungswerte erfolgt durch eine Varianzanalyse:

```
> anova(lm(CBWGT~CBB))
Analysis of Variance Table

Response: CBWGT
            Df  Sum Sq Mean Sq F value Pr(>F)
CBB          8    6.47    0.81  0.6043 0.7749
Residuals 1287 1722.73    1.34
```

Der t-Test

Falls die qualitative Variable nur zwei Gruppen definiert, kann man den Mittelwertvergleich auch mit dem t-Test durchführen:

Die Variable CTPEA hat für die beiden Geschlechter die Mittelwerte

```
> m.hyp <- tapply(CTPEA,CBSEX,mean)
> m.hyp
       0        1
77.21142 80.96142
```

Wir ziehen nun Stichproben:

```
> s <- split(CTPEA,CBSEX)
> x <- sample(s[[1]],20)
> x
 [1] 85 92 68 69 82 81 83 96 77 80 82 68 98 73 73 89 90 70 92 75
> y <- sample(s[[2]],30)
> y
 [1]  85  82  73  72  62  82  84  68 117  88  82 102  82  88  70  88  89  7
[20]  64  69  74  80  76  63  87  82  69  87  66
```

und überprüfen den Mittelwertunterschied in den Stichproben:

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

         Welch Two Sample t-test

data:  x and y
t = 0.446, df = 46.28, p-value = 0.6577
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.742165  7.442165
sample estimates:
mean of x mean of y
    81.15     79.80
```

Die Voreinstellungen der Eingabeparameter können wieder abgeändert werden.

5.5.3 Zwei qualitative Variable

Wir stellen die Kontingenztafel der Variablen MTE und FTE auf. Dabei verwenden wir die relativen Häufigkeiten.

```
> n <- dim(statlab)[1]
> observed <- table(MTE,FTE)/n
```

Die bei Unabhängigkeit erwarteten Häufigkeiten erhalten wir durch Bildung der Indifferenztafel:

```
> expected <- apply(observed,1,sum)%o%apply(observed,2,sum)
```

Für die Analyse der Abhängigkeit berechnen wir nun die standardisierten Differenzen zwischen der Kontingenztafel und der Indifferenztafel:

```
> sd <- (observed-expected)/sqrt(expected/n)
> sd
```

```
   FTE
MTE          0         1         2         3          4
  0 11.8328181  1.711114 -0.864514 -1.440800 -2.6042372
  1  3.5485108  5.871089  3.272386 -2.185825 -5.2852619
  2  0.2999513  3.134468  5.121385  1.583985 -7.8448174
  3 -2.5028603 -3.037903 -1.291039  3.452124  0.5303301
  4 -3.1431878 -4.564543 -6.859146 -4.310958 13.5041040
attr(,"class")
[1] "table"
```

Diese Tabelle kann man auch graphisch veranschaulichen:

```
> assocplot(obs)
```

Unter der Chiquadratgröße versteht man die Quadratsumme dieser standarisierten Differenzen:

```
> sum(sd*sd)
[1] 651.2905
```

Der Chiquadrattest überprüft die Größe auf Signifikanz:

```
> chisq.test(observed)

         Pearson’s Chi-squared test

data:  observed
X-squared = 651.2905, df = 16, p-value = < 2.2e-16

Warning message:
Chi-squared approximation may be incorrect in: chisq.test(observed)
```

Kapitel 6

Stochastik

rnorm

pchisq

Kapitel 7

Lineare Erklärungsmodelle

7.1 Multiple Regression

7.1.1 Modellanpassung

Multiple Regression untersucht die lineare Erklärbarkeit einer quantitativen Responsevariablen durch mehrere quantitative Prädiktoren.

Wir verwenden als Responsevariable CTHGHT und als Prädiktoren MTHGHT und FTHGHT. Zunächst passen wir ein lineares Modell an:

```
> fm <- lm(CTHGHT~1+MTHGHT+FTHGHT)
> coefficients(fm)
(Intercept)      MTHGHT      FTHGHT
 16.6082082   0.2936115   0.2558320
```

Mehr Information erhalten wir durch:

```
> summary(fm)

Call:
lm(formula = CTHGHT ~ MTHGHT + FTHGHT)

Residuals:
    Min      1Q  Median      3Q     Max
-5.9584 -1.5530 -0.1323  1.4859  7.9270

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  16.6082     1.9422   8.551   <2e-16 ***
MTHGHT        0.2936     0.0268  10.955   <2e-16 ***
FTHGHT        0.2558     0.0239  10.703   <2e-16 ***
---
Signif. codes:  0  ‘***’  0.001  ‘**’  0.01  ‘*’  0.05  ‘.’  0.1  ‘ ’  1

Residual standard error: 2.268 on 1293 degrees of freedom
```

```
Multiple R-Squared: 0.218,      Adjusted R-squared: 0.2168
F-statistic: 180.2 on 2 and 1293 degrees of freedom,    p-value:     0
```

Oft wird vorgeschlagen, man sollte die Residuen daraufhin überprüfen, ob sie wirklich normalverteilt sind. Wir machen das durch Diagramme:

```
> plot(density(residuals(fm)))
> qqnorm(residuals(fm))
```

Wenn das lineare Modell gut paßt, dann sollten die Residuen mit den Vorhersagewerten unkorreliert sein. Insbesondere sollte die Varianz der Residuen nicht von der Größe der Vorhersagewerte abhängen:

```
> plot(fitted(fm),residuals(fm))
```

7.1.2 Modellwahl

Bei einem multiplen Regressionsproblem stellt sich oft die Frage, wie man das Modell vereinfachen kann, ohne daß signifikante Teile des Erklärungswertes verloren gehen.

Einen automatischen Modellwahlprozeß bietet R an:

```
> step(lm(CTPEA~FIT+MBAG+FBAG))
Start:  AIC= 6034.24
 CTPEA ~ FIT + MBAG + FBAG

       Df Sum of Sq    RSS    AIC
- FBAG  1         1 135526   6032
<none>              135526   6034
- MBAG  1       640 136166   6038
- FIT   1      6415 141941   6092

Step:  AIC= 6032.25
 CTPEA ~ FIT + MBAG

       Df Sum of Sq    RSS    AIC
<none>              135526   6032
- MBAG  1      2158 137684   6051
- FIT   1      6513 142039   6091

Call:
lm(formula = CTPEA ~ FIT + MBAG)

Coefficients:
(Intercept)          FIT         MBAG
    67.8915       0.0329       0.2153
```

Wer selbst denken will, verwendet aus der Objektbibliothek **mylib.r** die Funktion:

```
> modelselection(cbind(CTPEA,FIT,MBAG,FBAG))
  ExplVar   RSquare     PValue FIT MBAG FBAG
```

```
1       3  0.062731             1    1    1
2       2  0.062728    0.94318   1    1    0
3       2  0.058304    0.01362   1    0    1
4       1  0.047806 3.6936e-05   1    0    0
5       2  0.018364 1.0880e-14   0    1    1
6       1  0.017687 6.7724e-14   0    1    0
7       1 0.0090892 2.2204e-16   0    0    1
```

Die angegebenen P-Werte beziehen sich auf jene Nullhypothesen, die Erklärungsbeitrag der mit 0 bezeichneten Variablen negieren. Ein großer P-Wert bedeutet also die Akzeptanz des angegebenen Modells, ein kleiner P-Wert seine Verwerfung. Das einfachste Modell, das gerade noch nicht verworfen wird, ist hier das zweite Modell.

7.1.3 Partielle Korrelation

Die Residuen eines linearen Modells nennt man auch Partialvariable, wenn man herausstreichen will, daß in den Residuen noch Information über die ursprüngliche Responsevariable enthalten ist, lediglich vermindert um jene Information, die durch die Erklärungsvariablen erklärt wird.

Die Korrelation zwischen CTPEA und MBAG beträgt

```
> cor(CTPEA,MBAG)
[1] 0.1329922
```

Diese Korrelation muß nicht (zur Gänze) auf eine kausale Beziehung zwischen diesen beiden Variablen zurückgehen, sondern kann durch andere Variable (versteckte Faktoren) verursacht sein. Um dies zu untersuchen, eliminiert man durch multiple Regression den Einfluß jener Variablen, die man verdächtigt, als versteckte Faktoren für die bestehende Korrelation verantwortlich zu sein. Im vorliegenden Beispiel könnten das die Variablen FIB und FIT sein:

```
> res.CTPEA <- residuals(lm(CTPEA~1+FIB+FIT))
> res.MBAG <- residuals(lm(MBAG~1+FIB+FIT))
```

Wenn wir nun die Korrelation dieser Partialvariablen bilden, dann erhalten wir:

```
> cor(res.CTPEA,res.MBAG)
[1] 0.09092254
```

Diese Korrelation nennt man die partielle Korrelation zwischen CTPEA und MBAG (partialisiert unter FIB und FIT), und sie ist deutlich geringer als die gewöhnliche Korrelation.

Diese Erscheinung hängt mit dem Phänomen der sogenannten Scheinkorrelation zusammen. Unter einer Scheinkorrelation versteht man eine Korrelation zwischen Variablen, die nicht auf eine kausale Beziehung zwischen den Variablen zurückgeht. Eine Scheinkorrelation verschwindet, wenn man die Variablen herauspartialisiert, die für die Korrelation verantwortlich sind.

Es wäre aber falsch zu meinen, daß partielle Korrelationen immer geringer sind als ihre entsprechenden nichtpartiellen Gegenstücke. Sehen wir uns ein Beispiel an.

Die Korrelation zwischen CTHGHT und MBAG beträgt

```
> cor(CTHGHT,MBAG)
[1] 0.08800654
```

Wenn wir jedoch den Einfluß von FTHGHT herauspartialisieren, erhalten wir

```
> res.CTHGHT <- residuals(lm(CTHGHT~1+FTHGHT))
> res.MBAG <- residuals(lm(MBAG~1+FTHGHT))
```

und als partielle Korrelation

```
> cor(res.CTHGHT,res.MBAG)
[1] 0.1432389
```

Hier ist nun die partielle Korrelation fast doppelt so groß wie die gewöhnliche Korrelation. Eine Erklärung für dieses Phänomen kann man leicht geben, würde aber hier zu weit führen.

Die partiellen Korrelationskoeffizienten lassen sich mit dem Befehl **pcor** aus der Objektbibliothek **mylib.r** rasch berechnen:

```
> pcor(cbind(CTPEA,MBAG,FIB,FIT))
           CTPEA        MBAG       FIB          FIT
CTPEA 1.00000000  0.09092254 0.1072736  0.17128906
MBAG  0.09092254  1.00000000 0.2696731 -0.06016844
FIB   0.10727360  0.26967309 1.0000000  0.29780135
FIT   0.17128906 -0.06016844 0.2978014  1.00000000
```

```
> pcor(cbind(CTHGHT,MBAG,FTHGHT))
          CTHGHT       MBAG     FTHGHT
CTHGHT 1.0000000  0.1432389  0.3955040
MBAG   0.1432389  1.0000000 -0.1604414
FTHGHT 0.3955040 -0.1604414  1.0000000
```

7.1.4 Die Rolle der Erklärungsvariablen

In Zusammenhang mit multipler Regression gibt es auch eine Varianzanalyse. Die Ergebnisse der Varianzanalyse hängen allerdings davon ab, in welcher Reihenfolge man die Erklärungsvariablen in das Modell einführt:

```
> anova(lm(CTHGHT~MTHGHT+FTHGHT))
Analysis of Variance Table

Response: CTHGHT
            Df Sum Sq Mean Sq F value    Pr(>F)
MTHGHT       1 1265.0  1265.0  245.91 < 2.2e-16 ***
FTHGHT       1  589.3   589.3  114.55 < 2.2e-16 ***
Residuals 1293 6651.5     5.1
---
Signif. codes:  0  ‘***’  0.001  ‘**’  0.01  ‘*’  0.05  ‘.’  0.1  ‘ ’  1

> anova(lm(CTHGHT~FTHGHT+MTHGHT))
Analysis of Variance Table

Response: CTHGHT
            Df Sum Sq Mean Sq F value    Pr(>F)
```

```
FTHGHT       1 1236.9  1236.9  240.45 < 2.2e-16 ***
MTHGHT       1  617.4   617.4  120.01 < 2.2e-16 ***
Residuals 1293 6651.5     5.1
---
Signif. codes:  0  '***'  0.001  '**'  0.01  '*'  0.05  '.'  0.1  ' '  1
```

Die Quadratsummen hängen mit den Korrelationskoeffizienten zusammen. Beispielsweise gilt für die letzte Tabelle

$$\frac{1236.9}{1236.9 + 617.4 + 6651.5} = r^2_{CTHGHT,FTHGHT},$$

aber

$$\frac{617.4}{617.4 + 6651.5} = r^2_{(CTHGHT,MTHGHT)-FTHGHT}.$$

7.2 Ein qualitativer Prädiktor - „Einfache Varianzanalyse"

Man kann den Vergleich der Erwartungswerte von unabhängigen Stichproben als multiples Regressionsmodell behandeln. Um die inhaltliche Bedeutung (Interpretation) der Regressionsparameter festzulegen, definiert man eine sogenannte Kontrastmatrix. Jede Zeile der Kontrastmatrix definiert einen Regressionsparameter.

Da die Variable MTE, die unsere Gruppierung definiert, ordinal ist und fünf Werte hat, definieren wir:

```
> contr <- rbind(c(1/5,1/5,1/5,1/5,1/5),
+ c(-1,1,0,0,0),
+ c(0,-1,1,0,0),
+ c(0,0,-1,1,0),
+ c(0,0,0,0-1,1))
> contr
     [,1] [,2] [,3] [,4] [,5]
[1,]  0.2  0.2  0.2  0.2  0.2
[2,] -1.0  1.0  0.0  0.0  0.0
[3,]  0.0 -1.0  1.0  0.0  0.0
[4,]  0.0  0.0 -1.0  1.0  0.0
[5,]  0.0  0.0  0.0 -1.0  1.0
```

Diese Kontrastmatrix definiert als ersten Parameter den Durchschnitt der Gruppenmittelwerte, und die restlichen Parameter sind die Mittelwertsdifferenzen zwischen aufeinanderfolgenden Gruppen.

Einfacher gehts mit

```
> contr <- matrix(0,5,5)
> data.entry(contr)
```

Um die Spalten des Regressionsmodells zu berechnen, muß diese Kontrastmatrix invertiert werden. Das Ergebnis ist die sogenannte Designmatrix:

```
> desgn <- solve(contr)
> desgn
     [,1] [,2] [,3] [,4] [,5]
[1,]    1 -0.8 -0.6 -0.4 -0.2
```

```
[2,]    1  0.2 -0.6 -0.4 -0.2
[3,]    1  0.2  0.4 -0.4 -0.2
[4,]    1  0.2  0.4  0.6 -0.2
[5,]    1  0.2  0.4  0.6  0.8
```

R benötigt die letzten vier Spalten der Designmatrix für die Bildung des Regressionsmodells:

```
> contrasts(MTE) <- desgn[,2:5]
```

Diese Designmatrix ist in R in der Library MASS mit dem Kommando

```
> contrasts(MTE) <- contr.sdif(5)
```

direkt erhältlich.

Nun können wir das lineare Modell analysieren:

```
> x <- lm(CTRA~MTE)
> summary(x)

Call:
lm(formula = CTRA ~ MTE)

Residuals:
     Min       1Q   Median       3Q      Max
-27.4158  -6.4291   0.5842   6.9901  27.9901

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  28.9842     0.4428  65.463  < 2e-16 ***
MTE1          3.8945     2.0495   1.900   0.0576 .
MTE2          2.4192     1.0177   2.377   0.0176 *
MTE3          3.5214     0.6340   5.554 3.39e-08 ***
MTE4          4.4653     0.7243   6.165 9.42e-10 ***
---
Signif. codes:  0  ‘***’  0.001  ‘**’  0.01  ‘*’  0.05  ‘.’  0.1  ‘ ’  1

Residual standard error: 9.319 on 1291 degrees of freedom
Multiple R-Squared: 0.1297,     Adjusted R-squared: 0.127
```

Wir überzeugen uns leicht, daß die Regressionsparameter genau jene Werte haben, die wir auf Grund der von uns vorgegebenen Interpretation erwarten:

```
> tapply(CTRA,MTE,mean)
       0        1        2        3        4
22.11538 26.00990 28.42915 31.95052 36.41581
> mean(tapply(CTRA,MTE,mean))
[1] 28.98415
```

Wir haben die Kontraste, die R als Defaults verwendet, bewußt verändert. An der Variablen FTE sehen wir uns an, welche Kontraste R in der Voreinstellung einsetzt:

```
> contrasts(FTE)
```

```
  1 2 3 4
0 0 0 0 0
1 1 0 0 0
2 0 1 0 0
3 0 0 1 0
4 0 0 0 1
```

Wir untersuchen die inhaltliche Bedeutung dieser Kontraste:

```
> const <- rep(1,times=5)
> desgn <- cbind(const,contrasts(FTE))
> round(solve(desgn))
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]   -1    1    0    0    0
[3,]   -1    0    1    0    0
[4,]   -1    0    0    1    0
[5,]   -1    0    0    0    1
```

Die Zeilen zeigen die Definition der Regressionsparameter. Dies können wir auch nachprüfen:

```
> lm(CTRA~FTE)

Call:
lm(formula = CTRA ~ FTE)

Coefficients:
(Intercept)         FTE1         FTE2         FTE3         FTE4
     23.932        2.017        4.658        6.205       11.733

> tapply(CTRA,FTE,mean)
       0        1        2        3        4
23.93182 25.94915 28.59016 30.13690 35.66435
```

7.3 Zwei qualitative Prädiktoren - „Zweifache Varianzanalyse”

Um den Einfluß der beiden qualitativen Variablen MTE und FTE auf CTPEA zu untersuchen, bilden wir Kreuztabelle der Mittelwerte

```
> mt <- round(tapply(CTPEA,list(MTE,FTE),mean),2)
> mt
      0     1     2     3     4
0 66.17 76.20 65.80 75.33 67.00
1 71.70 72.63 73.28 76.40 75.33
2 69.78 73.38 76.32 78.31 79.97
3 68.75 74.88 77.26 79.75 82.64
4    NA 72.67 80.60 85.63 85.35
```

und zur Kontrolle der Stichprobengrößen in den einzelnen Zellen die Kontingenztafel

```
> table(MTE,FTE)
   FTE
MTE  0  1   2   3   4
  0 12  5   5   3   1
  1 10 27  46  15   3
  2 18 66 200 146  64
  3  4 17  95 134 134
  4  0  3  20  38 230
```

Die Haupteffekte von MTE und FTE ergeben sich durch die Unterschiede der Gruppenmittelwerte zum Gesamtmittelwert:

```
> eff.MTE <- tapply(CTPEA,MTE,mean)-mean(CTPEA)
> eff.MTE
         0          1          2          3          4
-9.9710351 -5.6111722 -2.3374319  0.7286844  5.8448517
> eff.FTE
         0          1          2          3          4
-9.9500561 -5.5609960 -2.8131957  0.5147707  4.5154321
```

Würden die beiden Haupteffekte den gesamten Einfluß der Variablen MTE und FTE beschreiben, dann würde man als Zellmittelwerte in der Kreuztabellierung erwarten:

```
> mt.exp<- round(mean(CTPEA)+(eff.MTE%+%eff.FTE),2)
> mt.exp
      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] 59.17 63.55 66.30 69.63 73.63
[2,] 63.53 67.91 70.66 73.99 77.99
[3,] 66.80 71.19 73.94 77.26 81.26
[4,] 69.87 74.25 77.00 80.33 84.33
[5,] 74.98 79.37 82.12 85.45 89.45
```

Tatsächlich gibt es aber Abweichungen zwischen den erwarteten und den beobachteten Mittelwerten:

```
> mt-mt.exp
      0     1     2     3     4
0  7.00 12.65 -0.50  5.70 -6.63
1  8.17  4.72  2.62  2.41 -2.66
2  2.98  2.19  2.38  1.05 -1.29
3 -1.12  0.63  0.26 -0.58 -1.69
4    NA -6.70 -1.52  0.18 -4.10
```

Diese Abweichungen nennt man den Wechselwirkungseffekt zwischen den Variablen MTE und FTE.

Eine Signifikanzprüfung der Haupteffekte und des Wechselwirkungseffekts erhält man durch:

```
> fm <- lm(CTPEA~1+MTE+FTE+MTE:FTE)
> anova(fm)
Analysis of Variance Table

Response: CTPEA
           Df Sum Sq Mean Sq F value    Pr(>F)
```

```
MTE          4  18609    4652 49.6841 < 2.2e-16 ***
FTE          4   5758    1439 15.3725 2.768e-12 ***
MTE:FTE     15   1123      75  0.7995    0.6793
Residuals 1272 119107      94
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

7.4 Gemischte Prädiktoren - „Kovarianzanalyse”

Wir betrachten nun den Fall, bei dem eine quantitative Responsevariable durch eine qualitative und eine quantitative Variable erklärt werden soll.

Wir wollen im Datensatz STATLAB die Intelligenz des Kindes erklären durch das Alter der Mutter MBAG und die Schulbildung der Mutter MTE.

Die Intelligenz des Kindes beschreiben wir durch eine neue Variable:

```
> y <- apply(scale(cbind(CTRA,CTPEA)),1,mean)
```

Eine Abhängigkeit der Intelligenz des Kindes von der Schulbildung der Mutter läßt sich deskriptiv ganz einfach feststellen:

```
> tapply(y,MTE,mean)
          0           1           2           3           4
-0.91470643 -0.51318016 -0.23700000  0.08460356  0.55052851
```

Die Frage ist, ob diese Abhängigkeit durch das unterschiedliche Alter der Mutter erklärt werden kann.

Die Einflüsse der Variablen MBAG und MTE lassen sich durch das anpassen des linearen Modells

```
> fm <- lm(y~MBAG+MTE)
> anova(fm)
Analysis of Variance Table

Response: y
           Df Sum Sq Mean Sq F value    Pr(>F)
MBAG        1   8.46    8.46  13.670 0.0002271 ***
MTE         4 163.05   40.76  65.898 < 2.2e-16 ***
Residuals 1290 797.94    0.62
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

beurteilen. Das Modell zerlegt die Varianz sequentiell, dh. zunächst wird MBAG zur Erklärung von y verwendet, und anschließend werden die dabei entstehenden Residuen daraufhin untersucht, ob sie durch MTE beeinflusst werden. Um die Gruppenunterschiede zu analysieren, sollte man vernünftige Kontraste verwenden:

```
> contr <- rbind(c(1/5,1/5,1/5,1/5,1/5),
+c(-1,1,0,0,0),
+c(0,-1,1,0,0),
+c(0,0,-1,1,0),
```

```
+c(0,0,0,0-1,1))
> desgn <- solve(contr)
> contrasts(MTE,4) <- desgn[,2:5]
> fm<-lm(y~MBAG+MTE)
> anova(fm)
Analysis of Variance Table

Response: y
            Df Sum Sq Mean Sq F value    Pr(>F)
MBAG         1   8.46    8.46  13.670 0.0002271 ***
MTE          4 163.05   40.76  65.898 < 2.2e-16 ***
Residuals 1290 797.94    0.62
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> summary(fm)

Call:
lm(formula = y ~ MBAG + MTE)

Residuals:
     Min       1Q   Median       3Q      Max
-2.23508 -0.51510  0.02491  0.52121  3.43323

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.503605   0.116995  -4.304 1.80e-05 ***
MBAG         0.010054   0.003745   2.685  0.00735 **
MTE1         0.453939   0.174058   2.608  0.00921 **
MTE2         0.291091   0.086066   3.382  0.00074 ***
MTE3         0.322381   0.053508   6.025 2.20e-09 ***
MTE4         0.438584   0.061969   7.077 2.40e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7865 on 1290 degrees of freedom
Multiple R-Squared: 0.1769,     Adjusted R-squared: 0.1737
F-statistic: 55.45 on 5 and 1290 DF,  p-value:     0
```

Will man wissen, ob der Einfluß des Alters für unterschiedliche Schulbildungen verschieden stark ist, muß man die Wechselwirkung zwischen Alter und Schulbildung ins Erklärungsmodell einfügen:

```
> fm <- lm(y~MBAG+MTE+MTE:MBAG)
> anova(fm)
Analysis of Variance Table

Response: y
            Df Sum Sq Mean Sq F value    Pr(>F)
MBAG         1   8.46    8.46 13.6998 0.0002235 ***
MTE          4 163.05   40.76 66.0435 < 2.2e-16 ***
```

```
MBAG:MTE     4   4.22    1.06  1.7104 0.1452129
Residuals 1286 793.72    0.62
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Die Regressionsgeraden für die einzelnen Gruppen erhält man durch

```
> lm(y~MBAG,subset=(MTE==0))$coefficients
(Intercept)        MBAG
-1.29653414  0.01117964
> lm(y~MBAG,subset=(MTE==1))$coefficients
 (Intercept)         MBAG
-0.442749905 -0.002433615
...
```

oder alle auf einmal durch

```
> fm <- lm(y~0+MTE+MTE:MBAG)
> summary(fm)

Call:
lm(formula = y ~ 0 + MTE + MTE:MBAG)

Residuals:
     Min       1Q   Median       3Q      Max
-2.20111 -0.52043  0.01397  0.52360  3.22403

Coefficients:
           Estimate Std. Error t value Pr(>|t|)
MTE0      -1.296534   0.790577  -1.640 0.101253
MTE1      -0.442750   0.345665  -1.281 0.200472
MTE2      -0.458221   0.159133  -2.879 0.004049 **
MTE3      -0.582465   0.198968  -2.927 0.003478 **
MTE4       0.600457   0.289563   2.074 0.038309 *
MTE0:MBAG  0.011180   0.022704   0.492 0.622510
MTE1:MBAG -0.002434   0.011635  -0.209 0.834347
MTE2:MBAG  0.008057   0.005651   1.426 0.154175
MTE3:MBAG  0.024363   0.007118   3.423 0.000639 ***
MTE4:MBAG -0.001659   0.009498  -0.175 0.861383
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7856 on 1286 degrees of freedom
Multiple R-Squared: 0.1813,     Adjusted R-squared: 0.1749
F-statistic: 28.47 on 10 and 1286 DF,  p-value:     0
```

Diagramme liefert:

```
> coplot(y~MBAG|MTE,show.given=F,panel=panel.scatter,
          + xlim=range(MBAG),
          + ylim=range(y))
```

wobei die Reihenfolge der Diagramme von unten nach oben zu lesen ist.

Kapitel 8

Weitere multivariate Methoden

8.1 Mehrdimensionale Skalierung

Skalierung ist ein anderer Ausdruck für Kodierung. Über die Kodierung von eindimensionalen quantitativen Variablen haben wir schon im Abschnitt 5.3.2 gesprochen.

Segmentierung

Mit Segmentierung meinen wir hier die Transformation einer oder mehrerer quantitativen Variablen

Wenn man mehrere quantitative Variable in einzige qualitative Variable umkodieren will, sollte man die Daten zunächst auf eine gemeinsame Skala transformieren, dh. standardisieren:

```
> yy <- cbind(CTPEA,CTRA)
> zz <- stsc(yy)
```

Anschließend gilden wird einen Faktor, also eine qualitative Variable. Die übliche Methode, die dafür verwendet wird, ist der K-Means-Algorithmus:

```
> b <- kmeans(zz,5)
> attributes(b)
$names
[1] "cluster"  "centers"  "withinss" "size"
```

Die neue Variable

```
> b$cluster
   [1] 5 2 1 5 5 5 5 2 1 2 2 5 1 3 5 2 2 1 5 2 5 2 2 1 1 1 2 5 2 4 3 5
  [33] 5 2 5 2 2 2 3 3 2 5 3 5 5 1 1 1 5 3 2 2 2 1 2 2 4 1 1 1 2 3 5 1
  [65] 2 3 5 3 3 5 1 3 5 1 2 2 2 1 2 2 2 2 3 5 5 1 1 1 1 5 1 5 3 5 3 1
  [97] 2 1 1 2 1 1 3 5 3 1 1 3 1 5 1 2 2 1 1 5 5 3 5 1 1 1 1 1 1 4 3 5
  ...
```

enthält die Kodierung. Die Mittelwerte der gebildeten Gruppen verschafft man sich durch:

```
> b$centers
      CTRA    CTPEA
```

```
1 17.17568 66.70721
2 31.66372 69.38938
3 25.03333 82.47879
4 41.44203 97.41304
5 39.90263 82.48421
```

oder als Diagramm:

```
> barplot(t(b$centers),beside=T)
```

Um zu sehen, wie die Segmente im ursprünglichen Datensatz liegen, zeichnen wir ein Streudiagramm, bei dem wir die Gruppen einfärben:

```
> plot(CTRA,CTPEA,col=b$cluster)
```

Wir illustrieren den Vorgang nochmals anhand von höherdimensionalen Daten:

```
> yy <- cbind(CTRA,CTPEA,MBAG,FBAG)
> zz <- stsc(yy)
> b <- kmeans(zz,10)
> barplot(t(b$centers),beside=T)
> pairs(zz,col=b$cluster)
> pairs(prcomp(zz)$x,col=b$cluster)
```

8.2 Graphische Modelle

Um die Wechselbeziehungen zwischen drei qualitativen Variablen zu analysieren, verwenden wir sogenannte graphische Modelle. Das sind spezielle loglineare Modelle.

Wir beginnen mit einer dreidimensionalen Kontigenztafel

```
> x <- kmeans(stsc(cbind(CTPEA,CTRA)),5)$cluster
> y <- kmeans(MBAG,3)$cluster
> z <- CBSEX
> ftable<-table(x,y,z)
> ftable
, , z = 0

   y
x    1  2  3
  1 13 21 16
  2 53 30 38
  3 48 62 39
  4 62 96 51
  5 30 39 50

, , z = 1

   y
x    1  2  3
```

```
  1 19 40 31
  2 33 38 30
  3 30 35 13
  4 45 75 50
  5 59 95 55
```

Um die bedingte Unabhängigkeit der Variablen 1 und 2 zu prüfen, paßt man das folgende Modell an:

```
> library(MASS)
> fm<-loglm(~1*3+2*3,ftable)
> fm
Call:
loglm(formula = ~1 * 3 + 2 * 3, data = ftable)

Statistics:
                      X^2 df    P(> X^2)
Likelihood Ratio 39.11778 16 0.001045839
Pearson          38.94611 16 0.001107263
```

Alle graphischen Modelle liefert:

```
> graphical.model.3(ftable)
     edges  noedges  deviance df           pv
1 (13)(23)     (12)  39.11778 16 1.045839e-03
2 (12)(23)     (13)  79.38778 12 5.399792e-12
3 (12)(13)     (23)  18.30791 10 4.998653e-02
4     (12) (13)(23)  83.32065 14 6.819767e-12
5     (13) (12)(23)  43.05065 18 7.871463e-04
6     (23) (12)(13) 104.13053 20 2.280398e-13
```

Kapitel 9

Zeitreihen

9.1 Rekursive Folgen

Rekursive Folgen mit beliebigen Funktionstermen können mit der Funktion **iterate** aus der Objektbibliothek **mylib.r** berechnet werden:

```
> heron <- function(x){(x+2/x)/2}
> x <- iterate(fun=heron,start=3,len=20)
> x
          [,1]
 [1,] 3.000000
 [2,] 1.833333
 [3,] 1.462121
...
[19,] 1.414214
[20,] 1.414214
[21,] 1.414214
```

Bei linearen Funktionstermen kann man **filter** aus der Library **ts** verwenden:

```
> x <- filter(rep(2,20),filter=0.8,init=1,method="recursive")
> x
Time Series:
Start = 1
End = 20
Frequency = 1
 [1] 2.800000 4.240000 5.392000 6.313600 7.050880 7.640704 8.112563
 [8] 8.490051 8.792040 9.033632 9.226906 9.381525 9.505220 9.604176
[15] 9.683341 9.746673 9.797338 9.837870 9.870296 9.896237
```

liefert das gleiche wie

```
> lindiff <- function(x){0.8*x+2}
> x <- iterate(lindiff,start=1,len=20)
> as.vector(x)
```

```
 [1] 1.000000 2.800000 4.240000 5.392000 6.313600 7.050880 7.640704
 [8] 8.112563 8.490051 8.792040 9.033632 9.226906 9.381525 9.505220
[15] 9.604176 9.683341 9.746673 9.797338 9.837870 9.870296 9.896237
```

Die Verwendung des Befehls **filter(...,method="recursive",...)** für ist aber eher für lineare Iterationsformeln mit veränderlichem Additionsterm gedacht. Mit dem Filterkoeffizienten 1 entstehen Partialsummen, z.B. einer arithmetischen Folge:

```
> filter(1:10,filter=1,init=0,method="recursive")
Time Series:
Start = 1
End = 10
Frequency = 1
 [1]  1  3  6 10 15 21 28 36 45 55
```

oder:

```
> filter(2*(1:10)-1,filter=1,init=0,method="recursive")
Time Series:
Start = 1
End = 10
Frequency = 1
 [1]   1   4   9  16  25  36  49  64  81 100
```

oder einer geometrischen Folge:

```
> filter(2^(0:9),filter=1,init=0,method="recursive")
Time Series:
Start = 1
End = 10
Frequency = 1
 [1]    1    3    7   15   31   63  127  255  511 1023
```

Die Partialsummen von Zufallszahlen nennt man Random Walk:

```
> filter(rnorm(20),filter=1,init=0,method="recursive")
Time Series:
Start = 1
End = 20
Frequency = 1
 [1] -0.6080907 -1.0648259 -1.5880553 -2.2077415 -3.2708912
 [6] -2.6520020 -3.9090656 -5.2416086 -4.2438530 -4.3638421
[11] -4.4859727 -4.3289431 -4.1775086 -4.8586667 -4.6914896
[16] -5.1702869 -4.2839028 -4.9682436 -3.8400520 -2.5570815
```

Bei zahlreichen kleinen Perioden entsteht so der Wiener-Prozeß:

```
plot(filter(rnorm(1000),filter=1,init=0,method="recursive"))
```

Senkt man die Filterkonstante auf einen Wert zwischen 0 und 1, dann dann entsteht bei konstanten Inkrementen eine konvergente Folge:

```
> plot(filter(rep(2,30),filter=0.8,init=0,method="recursive"))
> plot(filter(rep(2,30),filter=-0.8,init=0,method="recursive"))
```

und bei zufälligen Inkrementen eine stationäre Zeitreihe:

```
plot(filter(rnorm(100),filter=0.5,init=0,method="recursive"))
```

9.2 Glättung von Zeitreihen

9.2.1 Exponentielle Glättung

Im folgenden verwenden wir eine feste Zeitreihe mit einem sinusförmigen Trend und einer autoregressiven Zufallskomponente:

```
> x <- sin(2*pi*(1:100)/100)+
       0.3*filter(rnorm(100),0.7,method="recursive")
> plot(x)
```

Eine exponentielle Glättung mit $\alpha = 0.3$ nach der Glättungsformel („Glättungsparameter"=$\alpha = 0.3$)

$$\hat{y}_t = 0.3x_t + 0.7\hat{y}_{t-1}$$

wird auf folgende Weise erreicht:

```
> y <- filter(0.3*x,0.7,method="recursive")
> plot(x,col=4)
> lines(y,col=2)
```

oder für $\alpha = 0.1$:

```
> y <- filter(0.1*x,0.9,method="recursive")
> plot(x,col=4)
> lines(y,col=2)
```

9.2.2 Gleitende Mittelwerte

Gleitende Mittelwerte lassen sich auch mit dem Befehl **filter** herstellen, und zwar einseitig:

```
> y <- filter(x,rep(1,10)/10,method="convolution",sides=1)
> plot(x,col=4)
> lines(y,col=2)
```

und zweiseitig:

```
> y <- filter(x,rep(1,10)/10,method="convolution",sides=2)
> plot(x,col=4)
> lines(y,col=2)
```

9.3 Saisonbereinigung

Saisonbereinigung erfolgt für periodische Zeitreihen. Wenn eine Folge den Klassentyp **ts** (Zeitreihe) erhält, kann man mit der Option **frequency** die Periodenlänge angeben:

```
> x <- rep(c(1,3,6,5,2,8),8)
> x <- ts(x, frequency=6)
> x
Time Series:
Start = c(1, 1)
End = c(8, 6)
Frequency = 6
 [1] 1 3 6 5 2 8 1 3 6 5 2 8 1 3 6 5 2 8 1 3 6 5 2 8 1 3 6 5 2 8 1 3
[33] 6 5 2 8 1 3 6 5 2 8 1 3 6 5 2 8
```

Der Befehl **stl** führt eine klassische Saisonbereinigung durch:

```
> stl(x,s.window="periodic")
 Call:
 stl(x = x, s.window = "periodic")

Components
Time Series:
Start = c(1, 1)
End = c(8, 6)
Frequency = 6
           seasonal    trend     remainder
1.000000 -3.1666667 4.166667  8.881784e-16
1.166667 -1.1666667 4.166667  1.776357e-15
1.333333  1.8333333 4.166667  0.000000e+00
...
```

Wir laden eine empirische Zeitreihe mit Saisonkomponente:

```
> library(MASS)
> data(accdeaths)
> plot(accdeaths)
> frequency(accdeaths)
[1] 12
```

Die Saisonbereinigung ergibt:

```
> x.stl <- stl(accdeaths,s.window="periodic")
> plot(x.stl)

> x.stl$time.series
            seasonal    trend   remainder
Jan 1973  -819.87198 9934.537 -107.665474
Feb 1973 -1559.05452 9880.772 -215.717176
Mar 1973  -759.57068 9827.006 -139.435258
Apr 1973  -530.48523 9765.695  -98.209997
May 1973   334.60040 9704.385  -21.984925
Jun 1973   814.91223 9636.872  374.215669
...
```

Kapitel 10

Die Bibliothek MYLIB.R

```
# Kodierung einer quantitativen Datenliste in gleich große
# Gruppen.

my.codes <- function(x,br)
   {
    if (length(br)>1)
       {
        y <- (x>br[1])
        for (i in 2:(length(br)-1)) y <- y+(x>br[i])
        }
    else
       {
        i <- sort.list(x+runif(length(x))*0.000001)
        y <- numeric(0)
        y[i] <- ceiling((1:length(x))/(length(x)+1)*br)
       }
   return(as.factor(y))
   }

# Eckpunkte des d-dimensionalen Einheitswürfels

cube <- function(d){
   x <- 0:(2^d-1)
   cube <- numeric()
   while(any(x>0)){
      temp <- floor(x/2)
      cube <- rbind(cube,x-temp*2)
      x <- temp
      }
   return(t(cube))
   }

# Modellgraph bei multipler Regression
```

```
modelselection <-  function(xx){
   n <- dim(xx)[1]
   k <- dim(xx)[2]
   y <- xx[,1]
   x <- xx[,2:k]
   k <- k-1
   c <- cube(k)
   c <- c[2:dim(c)[1],]
   sst <- var(y)
   m <- dim(c)[1]
   c <- c[m:1,]
   out <- numeric()
   for (i in 1:m){
      z <- x[,c[i,]==1]
      z <- cbind(rep(1,n),z)
      res <- y-z%*%(pinv(z)%*%y)
      ssr <- sum(res*res)
      if (i==1) {ssr0 <- ssr}
      bm <- 1-ssr/(sst*(n-1))
      kk <- sum(c[i,])
      if (i>1)
         {fv <- (ssr-ssr0)/ssr0*(n-k-1)/(k-kk)
          pv <- format.pval(1-pf(fv,k-kk,n-k-1))}
         else
         {pv <- ""}
      out <- rbind(out,c(kk,format.pval(bm),pv,c[i,]))
      }
    i <- order(out[,2])
    out <- out[i,]
    out <- out[m:1,]
    out <- data.frame(out)
    temp <- attr(xx,"dimnames")[[2]][2:(k+1)]
    attr(out,"names") <- c("ExplVar","RSquare","PValue",temp)
    return(out)
    }

# Analyse von dreidimensionalen Kontingenztafeln

graphical.model.3 <- function(ftable){
   edges <- "(13)(23)"
   noedges <- "(12)"
   fm <- loglm(~1*3+2*3,ftable)
   deviance <- fm$deviance
   df <- fm$df
   pv <- 1-pchisq(fm$deviance,fm$df)

   edges <- c(edges,"(12)(23)")
   noedges <- c(noedges,"(13)")
```

```
   fm <- loglm(~1*2+2*3,ftable)
   deviance <- c(deviance,fm$deviance)
   df <- c(df,fm$df)
   pv <- c(pv,1-pchisq(fm$deviance,fm$df))

   edges <- c(edges,"(12)(13)")
   noedges <- c(noedges,"(23)")
   fm <- loglm(~1*2+1*3,ftable)
   deviance <- c(deviance,fm$deviance)
   df <- c(df,fm$df)
   pv <- c(pv,1-pchisq(fm$deviance,fm$df))

   edges <- c(edges,"(12)")
   noedges <- c(noedges,"(13)(23)")
   fm <- loglm(~3+1*2,ftable)
   deviance <- c(deviance,fm$deviance)
   df <- c(df,fm$df)
   pv <- c(pv,1-pchisq(fm$deviance,fm$df))

   edges <- c(edges,"(13)")
   noedges <- c(noedges,"(12)(23)")
   fm <- loglm(~2+1*3,ftable)
   deviance <- c(deviance,fm$deviance)
   df <- c(df,fm$df)
   pv <- c(pv,1-pchisq(fm$deviance,fm$df))

   edges <- c(edges,"(23)")
   noedges <- c(noedges,"(12)(13)")
   fm <- loglm(~1+2*3,ftable)
   deviance <- c(deviance,fm$deviance)
   df <- c(df,fm$df)
   pv <- c(pv,1-pchisq(fm$deviance,fm$df))

   out <- data.frame(edges,noedges,deviance,df,pv)
   return(out)
   }

# Zweidimensionaler Scatterplot mit Regressionsgerade

panel.scatter <- function(x,y,...){
   fm <- lm(y~x)
   points(x,y,...)
   abline(fm$coefficients)
   }

# Partielle Korrelationen

pcor  <-  function(x){
```

```
   y  <-  solve(cov(x))
   d  <-  sqrt(diag(y))
   y  <-  -t(t(y/d)/d)
   diag(y)  <-  1
   dimnames(y)  <-  dimnames(x)
   dimnames(y)[[1]]  <-  dimnames(y)[[2]]
   return(y)
   }

# Iterationen

iterate <- function(fun,start,len){
   x <- rbind(as.vector(start))
   prev <- start
   for (i in 1:len){
      prev <- fun(prev)
      x <- rbind(x,as.vector(prev))
      }
   return(x)
   }
```