Einführung in S-Plus

Martin Sieger

1. November 1995

Vorwort

Das vorliegende Skript ist ursprünglich als schriftliche Ergänzung zu einem Einführungskurs in S-Plus an der TU Berlin, Fachbereich Informatik, erstellt worden. Dieser Kurs richtet sich an Studenten mit Grundkenntnissen in Statistik und liefert eine elementare Einführung in die Arbeitsweise von S-Plus. Dieses Skript basiert auf der S-Plus Version 3.1 für SunSparc-Workstations. Es ist daher zu beachten, daß einige Unterschiede auftreten können, falls der Leser dieses Skripts eine andere Version bzw. Hardware benutzt. Gleiches gilt auch für einige im Skript auftretende Beispiele, insbesondere werden hier teilweise Institutseigene Datensätze benutzt.

Inhaltsverzeichnis

1 Einführung

1.1 Was ist S-Plus

Warum S-Plus hat viel mit der Frage 'was ist Statistik' zu tun. Vor der täglichen Datenschwemme kann sich heute niemand mehr verschließen. Daten bestimmen die Nachrichten, den Arbeitsplatz, die Fahrt in den Urlaub, usw. Wo immer wir uns befinden, Informationen -seien es Zahlen, Tabellen, Graphikensind allgegenwärtig. Nur müssen wir uns die Frage stellen: wie können wir diese Informationen verarbeiten, um sie zu verstehen, zu deuten. Vieles läßt sich mit dem uns eigenen gesunden Menschenverstand, der natürlichen Abstrahierung interpretieren. Werden die Datenmassen und Fragestellungen komplexer, stehen wir denoch vor einigen Problemen. Der Ausweg ist leicht gefunden: Statistik. Statistik versetzt uns in die Lage, Daten (hoffentlich) sinnvoll zu verarbeiten, wesentliche Informationen herauszuziehen und darzustellen. Es liegt auf der Hand, daß uns im Zeitalter der Technokratie der Computer eine auch hier notwendige Stütze sein kann. Weiter ist offensichtlich, daß wir ein geeignetes Programm benutzen, um nicht in die Verlegenheit zu geraten, jedes Verfahren selber programmieren zu müssen. In diesem Bereich gibt es mittlerweile eine ganze Menge von Programmpaketen: (das wohl bekannteste) SAS, SPSS, Pstat, APL, Lispstat, ISP, etc. um nur einige zu nennen. Wer sich in diesem Wust an Paketen nicht auskennt, dem fällt die Wahl sehr schwer und der kann sich auch durchaus vergreifen. Viele dieser Programme haben spezielle Zielgruppen, sei es ein soziologisch ausgerichtetes Paket oder ein fast ausschliesslich experimentell in der Forschung zu nutzendes. Die Popularität von SAS begründet sich zum Beispiel einerseits in der Komplexität des Inhalts; SAS läßt sich in sehr vielen Bereichen anwenden.
S-Plus ist eine sogenannte 'interaktive Programmierumgebung', d.h. bei S-Plus handelt es sich um ein System, welches um eine Sprache (C) geplant und gebaut wurde, und dabei eine Sammlung von 'low-level-facilities' liefert, auf denen dann höhere vom Benutzer zu spezifizierende Funktionen/Programme gebaut werden können. S-Plus liefert hierzu die wesentlichen Tools für Statistik und Datenanalyse (angefangen bei simplen Mittelwerten bis zu multivariaten Varianzanalysen) und leicht zu handhabende Methoden zur Spezifizierung statistischer Modelle und Werkzeuge zur linearen/nichtlinearen Datenanalyse. Eine der wesentlichen Stützen von S-Plus ist die Verwendung relativ mächtiger Graphikroutinen, die bereits zur Konzeption von S-Plus dazugehörten. Weiterhin wurde (insbesondere in den letzten Jahren) eine effiziente und leicht zu handhabende Objektorientiertheit bereitgestellt.

Bevor im folgenden eine Beispielsitzung angeführt wird, noch einige Bemerkungen zur weiterführenden Literatur. Als einführende Kurzskripte sind hier zu empfehlen:

Ripley, B.D. (1994) *Introductory Guide to S-Plus*

Venables, B. und D.Smith (1992) *Notes on S-Plus: a programming environment for data analysis und graphics*

Wolf, P. und A. Hebestreit (1995) *Gehversuche mit SPlus*

Diese drei Skripten sind auch über das WWW (world wide web) anzusehen, die zugehörige URL lautet: html://www.cs.tu-berlin.de/fachgebiete/stat/others.html
Eine sehr schöne Einführung, die über die reinen Anfänge auch hinausgeht, leicht zu lesen ist, und viele Beispiele verwendet, gibt das Buch:

Venables, W.N. und B.D. Ripley (1994) *Modern Applied Statistics with S-Plus*.Springer, New York.

Weiterhin liefert das Buch

Chambers, J.M. und T.J. Hastie (eds) (1992) *Statistical Models in S*.New York: Chapman and Hall

eine Sammlung von Beiträgen zu ausgewählten aktuellen Gebieten der Statistik.
Darüberhinaus stehen dem Benutzer noch diverse User-Manuals und Reference-Guides zur Verfügung.

1.2 Starten,Beenden,Hilfe

Starten. Die S-Plus Version 3.1 steht im Verzeichnis '/usr/splus/splus3.1'. Der Aufruf lautet dann

```
/usr/splus/splus3.1/Splus3.1
```

Es ist jedoch zu empfehlen, den oben angeführten Pfad bereits in der entsprechenden '.paths' oder '.cshrc' Datei zu setzen. Dann wird mit


```
Splus3.1
```

gestartet.

S-Plus-Prompt. Nach dem Aufruf des Programms erscheint der sogenannte 'Prompt', der defaultmäßig wie folgt aussieht:

```
>
```

Hinter dem Prompt werden später die einzelnen Befehle eingegeben.

Case-Sensitivity. Es sei bereits an dieser Stelle bemerkt, daß S-Plus wie auch fast alle anderen UNIX-Systeme 'case sensitive' ist, d.h. S-Plus unterscheidet Groß- und Kleinschreibung.

Hilfe. S-Plus beinhaltet eine sogenannte Online-Help, welche vergleichbar ist zu den man-pages in UNIX. Hier gibt es nun zwei Möglichkeiten:

1. die Hilfe wird direkt aufgerufen, z.B.

   ```
   help(mean)
   ```

   Die resultierende Hilfe für die Funktion `mean` wird dann im aktuellen Fenster erscheinen.

2. ein Help-Fenster wird geöffnet. Mit dem Kommando

   ```
   help.start(gui='motif')
   ```

   wird ein Help-Fenster gestartet, welches aus drei Teilen besteht. Der obere Bereich des Fensters hat im rechten Teil sämtliche Möglichkeiten von S-Plus in Kategorien aufgelistet, d.h. die Kategorie 'Regression' beinhaltet dann beispielsweise sämtliche Funktionen, die mit Regression zu tun haben. Wird mit der linken Maustaste eine Kategorie angeklickt, so erscheint im linken Teil des Fensters eine Auflistung der Funktionen dieser ausgewählten Kategorie. Durch Mausklick können nun die einzelnen Funktionen angeschaut werden. Der dritte Teil des Help-Fensters befindet sich als Kommandozeile unter diesen beiden Auflistungen und liefert ein simples Suchprogramm. Wird beispielsweise eine Funktion gesucht, welche Covarianzen berechnet, so läßt sich hier möglicherweise 'variance' angeben. Die Hilfe wird dann nach Funktionen suchen, welche mit dem angegebenen Stichwort in Verbindung stehen (könnten). Sämtliche so gefundenen Funktionen werden dann (wie schon unter Kategorien) aufgelistet.
   Das Hilfe-Fenster wird mit dem Befehl

```
help.off()
```

geschlossen.

Graphik/Hardcopy. Graphiken werden grundsätzlich in einem Extra-Fenster erstellt. Dieses wird mit dem Kommando

```
motif()
```

gestartet. Ein Ausdruck des aktuellen Fenster-Inhalts wird durch entsprechenden Mausklick im Print-Button der Fenster-Menüleiste erzeugt. Geschlossen wird das Graphik-Fenster mit

```
graphics.off()
```

Verlassen von S-Plus. Die S-Plus Sitzung wird mit dem Befehl

```
q()
```

verlassen.

1.3 Beispielsitzung

Im folgenden eine exemplarische Sitzung, gemäß dem Motto 'learning by doing'. Natürlich kann hier nicht jedes Detail auf Anhieb verstanden werden, dennoch wird jeder Nutzer schon hier ein mehr oder weniger großes Gefühl für S-Plus erlangen.

`Splus3.1`	Startet das Programm, die Graphik-
`motif()`	Oberfläche und das Help-Fenster.
`help.start(gui='motif')`	

Es empfiehlt sich, durch Klicken der linken Maustaste im Help-Fenster bzw. Eintippen des jeweiligen Namens unter der 'topic'-Abfrage die einzelnen nachfolgenden Befehle in der Help nachzulesen!

1.

`2*3+4`	Simple arithmetische Berechnung
`x<-2*3+4`	Variablenzuweisung
`x`	Anzeigen des Wertes der Variablen

2.

`y<-c(3,2,-5,6.8,0,0,-3.7,5,4,-1)`	Erzeugung eines Vektors `y` und Berech-
`summe<-sum(y)`	nung der Summe und des arithmeti-
`mittel<-mean(y)`	schen Mittels über alle Elemente.
`summe`	Anzeigenlassen der Ergebnisse.
`mittel`	
`z<-c(0,5,1,4,0.5,-0.1,0,2,8,3)`	Erzeugung eines Vektors `z` und einfa-
`plot(y,z)`	che Plot-Darstellung von `z` gegen `y` mit
`title('einfacher Scatter-Plot`	Überschrift
`von z gegen y')`	
`help(cor)`	
`cor(y,z)`	Online-Help für die Funktion `cor`
`objects()`	Correlation von `y` und `z`
`rm(x,y,z)`	Anschauen der existierenden Objekte
	und löschen derselben

3.

`geschw<-c(122.6,123.3,123.2,121.6,`	Einlesen eines hyp. Datensatzes
`121.7,123.6,121.5,122.1,123.7,123.1,124.2,`	
`124.0,125.3,124.4,124.2,125.2,126.8,127.3)`	
`hist(geschw)`	Histogramm der Daten
`stem(geschw)`	Stem- und Leaf-Diagramm
`boxplot(geschw)`	Boxplot
`t.test(geschw,mu=123)`	zweiseitiger t-Test für $H_0: \ \mu = 123$

4.

```
x<-seq(1,20,0.5)
w<-1+x/2
y<-x+w*rnorm(x)

dum<-data.frame(x,y,w)
rm(x,y,w)
fm<-lm(y~ x,data=dum)
summary(fm)
fm1<-lm(y~
x,data=dum,weight=1/w^ 2)
summary(fm1)

lrf<-loess(y~ x,dum)

attach(dum)

plot(x,y)
lines(spline(x,fitted(lrf)))
abline(0,1,lty=3)
abline(fm)
abline(fm1,lty=4)

plot(fitted(fm),resid(fm),xlab='Anpassung',ylab='Residuen')
qqnorm(resid(fm))
qqline(resid(fm))

detach()
rm(fm,fm1,lrf,dum)
```

Erzeugung eines Stützbereichs `x` und Gewichtsvektors `w`, sowie einer abhängigen Variablen `y`. Die Vektoren werden in einem Data-Frame abgelegt und anschließend gelöscht.

Durchgeführt werden nun drei Regressionsanpassungen, eine lineare Einfachregression von `y` auf `x`, eine gewichtete Regression, sowie eine 'glatte', sogenannte lokal-gewichtete Regression. Die Daten sowie die unterschiedlichen Anpassungen werden geplottet.

Im Anschluß werden ein diagnostischer Plot auf Heteroskedastizität (ungleiche Varianzen) durchgeführt sowie ein QQ-Plot zur Überprüfung der Schiefe, Wölbung und Ausreißern in den Residuen.

5.

```
x<-rnorm(50)
y<-rnorm(50)

h<-chull(x,y)

plot(x,y)
polygon(x[h],y[h],dens=15,angle=30)
objects()
rm(x,y,h)
```

Zunächst werden zwei Zufallsvektoren erzeugt gemäß einer Standardnormalverteilung und deren konvexe Hülle berechnet. Dann wird ein Plot der Daten erzeugt mit der konvexen Hülle. Anschließend werden die benannten Variablen gelöscht.


```
x<-rnorm(1000)
y<-rnorm(1000)
hist(c(x,y+2),25)

contour(hist2d(x,y,,8,8))
persp(hist2d(x,y,,8,8))
image(hist2d(x,y,,8,8))
```

Erneute Erzeugung von Zufallsvektoren, mit Histogrammdarstellung der gemischten Komponenten. Angeschlossen werden drei graphische Darstellungen, ein 2D-Histogramm, ein Perspektiven-Plot und ein Image-Plot.

6.

```
butterfly<-function()
{
theta<-seq(0,24*pi,len=2000)
radius<-exp(cos(theta)) -
2*cos(4*theta) + sin(theta/12)^
5
plot(radius*sin(theta),-radius*cos(theta),
type='l',axes=F,xlab='',ylab='')
}
butterfly()
```

Mit `butterfly` wird nun eine 'eigene', neue Funktion geschrieben, die auf der Basis bereits existierender Funktionen (`cos,sin,seq`) aufbaut

7.

```
q()
```

Beenden der S-Plus-Beispielsitzung.

2 Elemente von S-Plus - Grundlegende Funktionen, Syntax

Die Zielsetzungen, die zur Konzeption von S-Plus (ursprünglich S) geführt haben, waren einerseits die Notwendigkeit einer interaktiven Sprache (wie z.B. auch UNIX) sowie andererseits die Bereitstellung einer Programmiersprache mit geeigneten objekt-orientierten Ausrichtungen. S-Plus läßt sich somit als Sprache zur Manipulation von Objekten bezeichnen.

Die folgenden Ausführungen dienen nun einer kurzen Einführung in die grundlegenden Strukturen von S-Plus.

2.1 Objekte, Syntax

Sämtliche während einer S-Plus Sitzung anfallenden Daten, seien es komplette Datensätze, einfache Berechnungen, Resultate, werden als Objekte abgelegt

oder bekommen ein Objekt zugewiesen. Es ist zu beachten, daß in S-Plus durchaus verschiedene Objekt-Typen und Zuweisungen existieren.
Bevor nun Objektnamen näher erläutert werden, sei zunächst die Zuweisung betrachtet: mit dem Zuweisungsoperator `<-` weist man einen Ausdruck einem Objektnamen zu: <-, _

```
objektname <- ausdruck
```

Alternativ läßt sich auch der Underscore _ als Zuweisungsoperator verwenden.

Was läßt sich nun als Objektname sinnvoll benutzen? Grundsätzlich werden S-Plus Objekte aus den bekannten alphanumerischen Zeichen benannt, bzw. Kombinationen aus diesen Zeichen, d.h. die Ziffern 0,...,9 sowie Groß- und Kleinbuchstaben (beachte: wie auch in UNIX wird S-Plus beispielsweise 'a' von 'A' unterscheiden). Der Benutzer wird schnell feststellen, daß einige Namen bereits reserviert sind, z.B. `for, break, function`. Weiterhin sollte die Verwendung bereits existierender Systemnamen wie `c, s, t, C, S, T, diff, range` und `tree` vermieden werden.

Ein S-Plus-Kommando kann nun aus zwei Komponenten bestehen: einer Zuweisung oder einem Ausdruck. Mehrere Kommandos können dabei entweder durch Semikolons ; getrennt in einer Zeile stehen oder aber jeweils in einer eigenen Zeile. Wird ein Kommando unvollständig angegeben, beispielsweise wird ein Klammerpaar nicht geschlossen angegeben, so fordert S-Plus mit dem +-Zeichen das vervollständigte Ende des Kommandos nach.
Das Zuweisungskommando weist den Wert eines Ausdrucks einer zu benennenden Variablen zu, der Zuweisungsoperator lautet wie bereits erwähnt '< −' oder '_':

`a<-6`	`a` erhält den Wert 6
`b<-a+3`	`b` erhält den Wert des Ausdrucks `a+3`

2.2 Variablen: Mode, Class, Attribute

An dieser Stelle einige Bemerkungen zu Variablen als solche. Es gibt drei Möglichkeiten, S-Plus Opbjekte zu unterscheiden:

mode. Aus welcher Menge stammen die Elemente eines Objektes, bzw. welche Operationen sind durchführbar? Die folgende Tabelle gibt einen Überblick

der unterschiedlichen Modi:

`null`	das leere Objekt
`logical`	Wahrheitswerte
`numeric`	Zahlen
`complex`	komplexe Zahlen
`character`	Zeichenketten
`list`	Listen von Objekten

class. Der `class`-Typ gibt Auskunft über die Darstellungsart des Objektes, d.h. handelt es sich zum Beispiel um ein Objekt vom Typ 'Vektor' oder 'Matrix'. Nachfolgenden eine Liste der verschiedenen 'Klassen':

`vector`	Vektoren, einfache Objekte
`matrix`	Matrizen
`array`	Felder (mehrdimensional)
`category`	kategorielle Daten
`time-series`	Zeitreihen

attribute. Auch wenn `mode` und `class` bekannt sind, so kann ein Objekt noch weitere Informationen beinhalten; diese lassen sich zusammenfassen unter dem Begriff 'Attribut':

`length`	Länge eines Vektors
`mode`	mode der Objekt-Elemente
`names`	den Elementen zugehörige Namen
`dimnames`	Namensliste für die Levels der einzelnen Dimensionen
`dim`	Dimension einer Matrix oder eines Feldes
`tsp`	Zeitreihenparameter (Beginn, Ende, Frequenz)
`levels`	Vektor der Levels von kategoriellen Daten

Insbesondere werden die einzelnen `class`- und `mode`-Typen in den nachfolgenden Abschnitten näher erläutert sowie einige Attribute.

2.3 Vektoren, Matrizen, Arrays

S-Plus-Objekte sind entweder Vektoren, Funktionen oder Listen (beachte: statt einem Skalar wird in S-Plus ein Vektor der Länge 1 verwendet). Vektoren bestehen hier aus numerischen oder logischen Werten, oder auch aus Character-Strings.

2.3.1 Erzeugung

Die einfachste Weise , einen Vektor zu erzeugen, ist die Terminal-Eingabe über die Funktion `concatenate`, kurz `c`; die Elemente des Vektors werden dabei der Funktion als Argumente in der Reihenfolge ihres Auftretens übergeben. Sollen Elemente eines Vektors extrahiert werden, so wird die Indexnummer des Elements in eckigen Klammern übergeben. `c(...)`

```
x<-c(3,4,5,2,7.4,1.6)
y<-5:18
autos<-c('Ford','VW','Opel')

x
[1] 3.0 4.0 5.0 2.0 7.4 1.6

y[3]
[1] 7

autos[2:3]
[1] "VW"   "Opel"
```

Der :-Operator ermöglicht hier die Erstellung einer Folge ganzer Zahlen innerhalb der spezifizierten Intervallgrenzen. Die Eingabe der Characters erfolgt innerhalb von Anführungszeichen. Numerische Werte werden einfach nur durch Kommas getrennt. Die Darstellung von logischen Werten erfolgt über `T` oder `F`, die Eingabe über `TRUE` oder `FALSE` : `:`

```
x>3
[1] F T T F T F
```

`TRUE`
`FALSE`

Weiterhin lassen sich die Elemente eines Vektors auch benennen, dabei erfolgt der Zugriff auf einzelne Elemente über die Angabe eines Vektors mit den Namen oder den Indexnummern der zu extrahierenden Elemente bzw. über Angabe eines logischen Vektors:

```
names(x)<-c('a','b','d','c','f','e')
names(x)
[1] "a" "b" "d" "c" "f" "e"

x
 a b d c   f   e
 3 4 5 2 7.4 1.6

x['d']
 d
 5

letters[1:3]
[1] "a" "b" "c"

x[letters[1:3]]
 a b c
 3 4 2

x[x>3]
 b d   f
 4 5 7.4
```

Sollen einzelne Elemente eines Vektors nicht verwendet werden, was eine Schrumpfung des Vektors beinhaltet, so läßt sich dies über die Angabe negativer Indizes erreichen (die Namen können für diese Operation bei Character-Vektoren jedoch nicht benutzt werden):

```
x[-c(2:4)]
 a   f   e
 3 7.4 1.6
```

Wie läßt sich nun ein Vektor in eine Matrix umwandeln? Zwei einfache Möglichkeiten lauten wie folgt : `dim`

`dim(x)<-c(2,3)` — Direkte Änderung der Dimension von `c(6,1)` → `c(2,3)`

Die Namen der Elemente erscheinen hier nun als Attribute. Als zweite Möglichkeit bietet sich folgende Version (die gebräuchlichere) an: `matrix`

`matrix(x,2,3)` — Direktes Einlesen des Vektors in Matrixform.

Man beachte, daß der Vektor Defaultmäßig über die Spalten eingelesen wird, vergleiche dazu auch die Zeileneingabe gemäß

```
matrix(x,2,3,byrow=T)
```

In diesem Fall wird der Funktion `matrix` noch die zusätzliche Option `byrow=T` übergeben, welche die zeilenweise Eingabe des Vektors bewirkt.

`array`

Der Sprung von Matrizen zu Arrays ist nun nicht allzu schwer. Ein Array ist zunächst eine mutipel-indizierte Sammlung von Daten; der wesentliche Spezialfall eines Arrays ist wiederum eine Matrix, ein Array mit zwei Indizes. Ein Array läßt sich weiter charakterisieren über einen zugehörigen Dimensionsvektor, d.h. einen Vektor aus positiven ganzen Zahlen der Länge ≥ 2. Dabei gibt die Länge des Dimensionsvektors gerade die Dimension des Arrays an. Die Werte des Dimensionsvektors liefern die obere Grenze für jeden Index.
Ein Vektor kann nun durch Hinzufügen eines Dimensionsvektors als Array betrachtet werden. Beispielsweise wird ein Vektor `x` der Länge 150 durch

```
x<-array(x,dim=c(3,5,10))
```

ein $3 \times 5 \times 10$ Array generieren. Alternativ wird dasselbe Resultat durch

```
dim(x)<-c(3,5,10)
```

erzeugt. Der Zugriff auf ein Element erfolgt sodann durch Angabe in eckigen Klammern (wie schon zuvor), also z.B.

```
x[2,1,5]
```

Wie läßt sich nun wieder eine Matrix aus einem Array erhalten? Eine Matrix läßt sich entweder durch Arrays mit zugehörigem Dimensionsvektor der Länge 2 oder durch die bereits erwähnte Funktion `matrix` generieren:


```
matrix(0,nrow=10,ncol=10)
```

erzeugt so eine 10×10 Matrix mit den Einträgen 0.

(Manchmal ist es notwendig (oder zumindest hilfreich), den einzelnen Dimensionen des Arrays Namen zu geben. Dies läßt sich über das `dimnames`-Attribut erreichen.)

2.3.2 Indizierung, Extrahieren von Teilmengen

Im folgenden sei die Indizierung von Vektoren, Matrizen und Feldern etwas formaler betrachtet; die 'Index'-Vektoren können aus vier verschiedenen Typen bestehen:

logische Index-Vektoren. Logische Index-Vektoren haben die gleiche Länge wie die zugehörigen Vektoren, deren Elemente ausgewählt werden sollen. Dabei werden gerade die Elemente ausgewählt, an deren Position der Index-Vektor den Wert `T` liefert.

```
y<-x[!is.na(x)]
```

wird der Variablen `y` somit einen Vektor zuweisen, der aus den nichtfehlenden Werten von `x` besteht.

```
y<-x[x>3]
```

wählt für `y` sämtliche `x`-Elemente aus, die echt größer als 3 sind.

Index-Vektoren aus $I\!N_+$. Die Werte des Index-Vektors für `x` stammen aus der Menge $\{1, \ldots, $ `length(x)`$\}$. Die entsprechenden Werte des Vektors werden extrahiert. Das Resultat hat somit die gleiche Länge wie der Indexvektor:

```
x[3:10]
```

liefert somit den dritten bis zehnten Wert aus `x`.

Index-Vektoren aus den negativen ganzen Zahlen. Die Angabe eines solchen Indexvektors schließt die indizierten Elemente aus:

```
y<-x[-(1:6)]
```

wird einen Vektor `y` liefern, der die Elemente aus `x` von der siebten Position an enthält.

Character-Index-Vektoren. Besitzt ein Vektor-Objekt ein `names`-Attribut, so können durch Angabe der Namen die entsprechenden Elemente wie schon unter 2. extrahiert werden:

```
alter.kinder<-c(3,9,7,12)
namen.kinder<-c('Anna','Paul','Kurt','Rosi')
alter.kinder[c('Anna','Kurt')]
```

Die Flexibilität solcher Indizierungen erlaubt es, auch Zuweisungen auf einzelne Elemente durchzuführen. So wird durch

```
x[is.na(x)]<-0
```

sämtlichen fehlenden Werten der Wert 0 zugewiesen.

```
y[y<0]<- -y[y<0]
```

wirkt wie

```
abs(y)
```

Die Indizierung von Arrays verläuft analog. So läßt sich ein Teil-Array extrahieren, indem beispielsweise eine Index-Position im dem Array zugehörigen Index-Vektor frei bleibt, oder an einer Stelle ein Vektor angegeben wird. So erzeugt z.B. bei unserem $3 \times 5 \times 10$ Array das Kommando

```
x[,2:4,3:9]
```

ein $3 \times 3 \times 7$ Sub-Array von `x`.
Beispielsweise erhält man so aus dem Vektor der Zahlen 1, ... ,24 über

```
x<-array(1:24,dim=c(3,2,4))
```

ein $3 \times 2 \times 4$ Array:

```
x
```

```
, , 1
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2
     [,1] [,2]
[1,]    7   10
```



```
[2,]    8   11
[3,]    9   12

, , 3
     [,1] [,2]
[1,]   13   16
[2,]   14   17
[3,]   15   18

, , 4
     [,1] [,2]
[1,]   19   22
[2,]   20   23
[3,]   21   24
```

Ein $2 \times 2 \times 3$ Sub-Array wird nun beispielsweise extrahiert gemäß

```
x[2:3,,1:3]
```

```
, , 1
     [,1] [,2]
[1,]    2    5
[2,]    3    6

, , 2
     [,1] [,2]
[1,]    8   11
[2,]    9   12

, , 3
     [,1] [,2]
[1,]   14   17
[2,]   15   18
```

2.4 Listen

Im Gegensatz zu Vektoren ermöglichen Listen auch die Zusammenstellung von Komponenten unterschiedlichen Typs, z.B. `list`

```
Fam.Meier<-list(im.beruf=c('anna meier','paul meier'), kinder=2,
                alter.kinder=c(15,12))
Fam.Meier
 $im.beruf:
[1] "anna meier" "paul meier"

$kinder:
[1] 2

$alter.kinder:
[1] 15 12
```

Der Zugriff auf einzelne Listenelemente erfolgt auf unterschiedliche Weise, entweder durch Angabe des Komponenten-Index

```
Fam.Meier[[2]]
[1] 2

Fam.Meier[[3]][1]
[1] 15
```

oder aber man nutzt den `$`-Operator zur Angabe des Komponentennamens `$` (mit eventuell anschließender Namen-Indizierung des entsprechenden Komponenten-Elements)

```
Fam.Meier$kinder
Fam.Meier$alter.kinder[1]
```

bzw. auch in Kurzform (sofern sich diese eindeutig unterscheiden läßt)

```
Fam.Meier$k
Fam.Meier$a[1]
```

2.5 Faktoren

Die übliche Weise, kategoriale Variablen zu repräsentieren, geschieht durch `factor` Verwendung von Faktoren. Dabei ist zu beachten, daß die Eingabe der Faktoren wie die Eingabe eines Charakter-Strings funktioniert, die Ausgabe jedoch ohne Anführungszeichen erscheint:

```
geb.land<-factor(c('F','D','A','D','D','A','F','D'))
geb.land
[1] F D A D D A F D
```


Intern wird ein Faktor als Code gespeichert, was über (bspw.)

```
print.default(geb.land)
[1] 3 2 1 2 2 1 3 2
attr(, "levels"):
[1] "A" "D" "F"
attr(, "class"):
[1] "factor"
```

abgefragt werden kann. Die Verwendung von Faktoren spielt insbesondere in der Analyse kategorialer Daten eine große Rolle, hier wird durch die `factor`-Funktion angezeigt, daß es sich bei der jeweiligen Variablen um eine kategoriale Variable handelt.

2.6 Data-Frames

Data-Frames geben dem Benutzer die Möglichkeit, auf relativ einfache Weise ein Datenobjekt in Form einer Matrix darzustellen und zu manipulieren. Ein Data-Frame hat die Form einer Matrix, welche als Liste mit gleich langen Variablen, durchaus unterschiedlichen Typs, aufgefasst werden kann.
Beispielsweise liefert der folgende Data-Frame die umsatzstärksten (in Ecu) Unternehmen der Europäischen Union mit zugehöriger Nationenzugehörigkeit:

```
nation umsatz angest        firma
1      D  37042 368226 Daimler-Benz
2      D  31689 257561   Volkswagen
3      D  29640 365000      Siemens
4      D  23860  92091         Veba
5      D  23089 136990         BASF
.
.
.
```

Die Komponenten des Data-Frames werden als Spalten dargestellt (vgl. auch mit der Zeilendarstellung von Vektorkomponenten bei Listen). Insbesondere lassen sich Data-Frames ähnlich wie Matrizen indizieren, durch Zeilen- und Spaltenindizes:

```
unternehmen[1:3,c(2,3)]
  umsatz angest
1  37042 368226
```

```
2  31689 257561
3  29640 365000
```

Weiterhin ist es möglich, auf einzelne Komponenten wie bei Listen zuzugreifen, d.h. durch Angabe des Komponentennamens unter Verwendung des $-Operators; z.B.

```
unternehmen$angest
```

Erleichtert wird die Arbeit mit den Komponenten durch die Funktionen `attach` und `detach`. Als Argument erhalten die Funktionen den Namen eines oder mehrerer Data-Frames. Mit `attach` werden dann die Komponenten des Data-Frames als Variablen unter dem entsprechenden Komponentennamen temporär 'sichtbar' gemacht, mit `detach` wieder 'unsichtbar':

attach
detach

```
attach(unternehmen)
umsatz
angest
detach(unternehmen)
```

Die Erzeugung von Data-Frames aus Vektoren erfolgt über die Funktion `data.frame` :

data.frame

```
data.frame(name1=vec1,name2=vec2,...)
```

Dabei bezeichnen `name1,name2,...` die entsprechenden Namen für die Komponenten und `vec1,vec2,...` die zu übergebenden Vektoren für die Komponenten. Man beachte, daß Character-Vektoren automatisch wie Faktoren behandelt werden.

2.7 Operatoren und Funktionen

2.7.1 Arithmetische Ausdrücke

Arithmetik

Grundsätzlich werden arithmetische Ausdrücke elementweise auf Vektoren ausgeführt. Die Grundrechenarten werden über die Operatoren `+,-,*,/,^` geregelt. Komplexe Vektoren werden dargestellt in der Form `a+bi`, wobei `a` und `b` reelle Zahlen darstellen und `i` ohne Leerzeichen angehängt wird; durch die Funktionen `Re` und `Im` lassen sich dann Real- und Imaginärteil extrahieren.

```
a<-3+4i
```



```
a
[1] 3+4i
Re(a)
[1] 3

Im(a)
[1] 4
```

Bei standardarithmetischen Operationen geht S-Plus grundsätzlich wie folgt vor: das Ergebnis einer Operation wird ein Vektor sein, dessen Länge der Länge des längsten Operanden des Ausdrucks entspricht. Die kürzeren Vektoren werden bei der Berechnung auf gerade diese Länge erweitert, indem die fehlenden Stellen mit den Werten der bereits besetzten Stellen aufgefüllt werden. Zum Beispiel

```
x<-c(1,3,4,2,3.5)
y<-c(x,0,x)
v<-2*x+y+1
```

Erzeugung von Vektoren `x` und `y`, wobei `x` der Längen 5 bzw. 11. Die Berechnung von `v` folgt dann der angegebenen Regel; der Vektor 2 der Länge 1 wird zunächst auf einen Vektor der Länge 5 erweitert $((2,2,2,2,2))$ und elementweise mit den Elementen von `x` multipliziert $((2,6,8,4,7))$. Zur Addition mit `y` wird der neue Vektor 2`x` auf die Länge 11 erweitert durch Auffüllen der freien Plätze mit den bekannten Werten in exakt der Reihenfolge ihres Auftretens, d.h. 2`x` wird zu $(2,6,8,4,7,2,6,8,4,7,2)$.

Trotzdem erhält man eine Warnung:

```
Warning messages:
  Length of longer object is not a multiple of the length of the
  shorter object in: 2 * x + y
```

Es empfiehlt sich, immer zu prüfen, inwieweit die Dimensionen der Operatoren auch zusammenpassen.

2.7.2 Wertigkeit der Operatoren

Die Wertigkeit der Operatoren ist in der nachstehenden Tabelle aufgeführt. Je tiefer ein Operator aufgelistet ist, desto geringer ist seine Wertigkeit.

`$`	Extrahieren von Listen
`[ [[`	Extrahieren von Vektor- und Listenelementen
`^`	Potenzierung
`:`	Generierung von Folgen
`%%, %/%, %*%`	spezielle Matrixoperatoren
`* /`	Multiplikation, Division
`+ -`	Addition, Subtraktion
`< > <= >= == !=`	logische Vergleichsoperatoren
`!`	logische Negation
`& | && ||`	logischer Schnitt, Vereinigung
`~`	Formel-Operator
`<<-`	globale Zuweisung (innerhalb von Funktionen)
`<- _ ->`	Zuweisung

So wird beispielsweise

```
x$abc*3:7^2
```

entwickelt gemäß

```
x$abc * ( 3:(7^2) )
```

2.7.3 Logische Ausdrücke

Logische Vektoren entstehen in der Regel durch Abfragen, häufig eingebettet in Bedingungen. Die zugehörigen Operatoren lauten (es seien dabei `c1` und `c2` logische Ausdrücke):

`TRUE`
`FALSE`

`==`	exakte Gleichheit
`!=`	nicht exakte Gleichheit
`c1& c2`	logisches 'und'
`c1|c2`	logisches 'oder'
`!c1`	logische Negation

Werden logische Vektoren in arithmetische Operationen eingebettet, so werden `F` als 0 und `T` als 1 interpretiert.
Weitere nützliche Funktionen sind hier `any, all` und `all.equal`.

2.7.4 Elementare Matrix-Operationen

Die grundlegenden Matrix-Operationen lauten:

`t(X)`	Transponierte von `X`
`nrow(X)`	Zeilenanzahl von `X`
`ncol(X)`	Spaltenanzahl von `X`
`X*Y`	Elemtweises Produkt von `X` und `Y` (beide quadratisch!)
`X%*%Y`	Matrixprodukt

Ist beispielsweise `x` eine 4×4 Matrix mit den Einträgen 1, ... ,16 und `y` eine 4×4 Matrix mit den Einträgen 36, ... ,21:

```
x<-matrix(1:16,4,4)
y<-matrix(36:11,4,4)

x
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16

y
     [,1] [,2] [,3] [,4]
```

```
[1,]   36   32   28   24
[2,]   35   31   27   23
[3,]   34   30   26   22
[4,]   33   29   25   21
```

so ergeben sich die angeführten Operationen zu

```
t(x)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16

nrow(x)
[1] 4
ncol(x)
[1] 4

x*y
     [,1] [,2] [,3] [,4]
[1,]   36  160  252  312
[2,]   70  186  270  322
[3,]  102  210  286  330
[4,]  132  232  300  336

x%*%y
     [,1] [,2] [,3] [,4]
[1,]  946  834  722  610
[2,] 1084  956  828  700
[3,] 1222 1078  934  790
[4,] 1360 1200 1040  880
```

2.8 Funktionsaufrufe

Bereits in der Beispielsitzung wurden S-Plus-Funktionen implizit verwendet. Eine S-Plus-Funktion wird dabei gestartet durch die Angabe eines Funktionsnamens sowie der Übergabe der hinreichenden, (notwendigen) Argumente innerhalb eines runden Klammerpaares:

```
funktionsname ( argumentliste )
```


Zusätzlich können Optionen bzw. weitere Argumente angegeben werden, die intern bereits defaultmäßig gesetzt wurden und vom Benutzer individuell zu gestalten sind. Eine detaillierte Beschreibung liefert jeweils die Online-Help. Wird eine Funktion nur mit dem Funktionsnamen aufgerufen, d.h. ohne Klammerpaar und Argumente, so erhält man die Definition der Funktion. Die internen S-Plus Funktionen sind in der Regel ebenfalls aus S-Plus Sprachkonstrukten gebaut, sodaß hier auch Hinweise erhalten werden über elegantes Programmieren (siehe dazu insbesondere Abschnitt 7).

Bei der Angabe der Argumente ist folgendes zu beachten: verlangt die Funktions-Definition eine bestimmte Reihenfolge für die Angabe, so muß dieses Reihenfolge eingehalten werden; weiterhin lassen sich viele Argumente auch über `name=wert` spezifizieren, falls die Reihenfolge irrelevant ist. So beinhaltet die Funktion `mean` die folgende Definition:

```
mean(x,trim=0,na.rm=F)
```

d.h. mit `x` wird ein numerisches Objekt (in der Regel ein Vektor) übergeben, mit `trim=0` und `na.rm=F` werden der Funktion defaultmäßig Optionen gesetzt. Damit erhält man z.B.

`mean(x)`	gewöhnliches arith. Mittel
`mean(x,0.2)`	20-Prozent-getrimmtes Mittel
`mean(x,0.2,T)`	gleiches Resultat wie zuvor
`mean(x,na.rm=T,trim=0.2)`	mit Enfernung von `NA`s

2.8.1 Standardfunktionen

S-Plus beinhaltet neben den Grundrechenarten (natürlich) eine Reihe mathematischer Standardfunktionen:

Rundungsfunktionen. `round, floor, ceiling, trunc`

logarithmische/trigonometrische Funktionen. `log, log10, exp, sin, cos, tan, acos, asin, atan, cosh, sinh, tanh`

Summations-/Produktfunktionen. `sum, cumsum, prod, cumprod`

Rangefunktionen. `max, min, range, cummax, cummin`

Sortier-/Ordnungsfunktionen. `sort , order`

2.8.2 Erzeugung von Zahlenfolgen

Der einfachste Weg, eine Folge ganzer Zahlen in einem vorbestimmten Intervall zu generieren führt über den :-Operator. :

```
2:15
[1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15
(-6):1
[1] -6 -5 -4 -3 -2 -1  0  1

12:7
[1] 12 11 10  9  8  7
```

Eine allgemeinere und flexiblere Handhabung liefert die Funktion `seq`. Der Funktion können bis zu fünf Argumente übergeben werden:

`seq(from, to, by, length, along)`

Allerdings werden niemals alle Argumente für einen Aufruf benötigt. So erzeugen

```
seq(2,15)
seq(from=2, to=15)
```

dieselbe Zahlenfolge wie schon

```
2:15
```

Die Argumente `by` und `length` spezifizieren hier die Schrittgröße bzw. Länge der Folge:

```
seq(from=2, length=14)
seq(from=2, to=15, by=1)
```

erzeugen hier ebenfalls die Folge ganzer Zahlen von 2 bis 15.
Das Argument `along` erhält einen Vektor als Wert; wird z.B. mit `x` ein beliebiger Vektor bezeichnet und gibt man lediglich

```
seq(along=x)
```

an, so wird man als Ergebnis eine Zahlenfolge erhalten wie bei

```
1:length(x)
```

Eine weitere Möglichkeit, Folgen zu erzeugen, bietet die `rep`-Funktion. Mit

```
rep(x,times=n)
```


wird der Vektor `x` gerade n-mal wiederholgt, d.h. hintereinandergesetzt; z.B.

```
x<-1:4
i<-rep(2,4)
y<-rep(x,2)
z<-rep(x,i)
w<-rep(x,x)

x
[1] 1 2 3 4
i
[1] 2 2 2 2
y
[1] 1 2 3 4 1 2 3 4
z
[1] 1 1 2 2 3 3 4 4
w
[1] 1 2 2 3 3 3 4 4 4 4
```

2.8.3 Markierung fehlender Werte

Fehlende Werte in Vektoren werden durch den Wert `NA` ersetzt bzw. dargestellt. `NA`
Die Funktion `is.na(x)` liefert dann `T` an der Stelle des fehlenden Wertes, ansonsten `F`. Generell werden sämtliche Operationen auf `NA`s ein `NA` als Resultat haben; Auswege liefern in bestimmten Fällen die Funktionen `na.fail` und `na.omit`.

2.8.4 Operationen auf Character-Strings

Der Benutzer sollte sich bewußt sein, daß S-Plus auf Character-'Strings' operiert und nicht auf den einzelnen Charactern! So ist `""`
ein regulärer character string ohne Characters als Inhalt. Andererseits ist `character(0)` ein leerer Vektor, der allerdings String-Elemente besäße, falls er Elemente hätte. Somit hat die Länge 1 und `character(0)` die Länge 0.
Beispiele für Funktionen, die auf character-Vektoren arbeiten sind z.B. die Funktion `paste` zum Zusammenfügen von einzelnen Character-Strings, die 'substring'-Funktion zum Extrahieren von Teilstrings oder 'abbreviate' zum allgemeinen Abkürzen von character-strings.

2.8.5 Sortieren

Sowohl numerische als auch character-Vektoren lassen sich mit der Funktion `sort` sortieren. Existiert ein `names`-Attribut, so werden die zugehörigen Namen beibehalten. Soll nur ein bestimmter Teil der Daten sortiert werden, so läßt sich die Option `partial` verwenden. Um beispielsweise den Median von 100001 Daten zu finden, so reicht es aus, lediglich 50001 Werte zu sortieren und den letzten Wert dieser sortierten Folge zu extrahieren: `sort`

```
median<-sort(x,partial=50001)[50001]
```

Eine weitere Funktion ist hier `order`, welche einen Vektor der Indizes liefert, die eine aus `sort` rusultierende Ordnung erstellen.

2.8.6 Matrix-Operationen und Lineare Algebra

Grundlegende Funktionen Im folgenden einige Matrix-Operationen. Es ist anzumerken, daß viele der Operationen auch mit numerischen Data-Frames funktionieren (s. auch später).

Kreuzprodukt. Die Funktion `crossprod` liefert Kreuzprodukte, d.h. `crossprod`

```
crossprod(X,Y)
```

liefert das Resultat für $X^T y$. Wird nur ein Argument übergeben, so wird das Kreuzprodukt bezüglich des ersten Arguments gebildet:

```
crossprod(X)
```

liefert somit $X^T X$.

Äußeres Produkt. Das äußere Produkt zweier numerischer Arrays `a` und `b` liefert ein Array, dessen Dimensionsvektor durch Zusammenfügen der beiden Dimensionsvektoren entsteht. Die Elemente es äußeren Produkts berechnen sich aus sämtlichen elementweisen Produkten der Elemente aus `a` und `b`. `outer %o%`
Es gibt zwei Möglichkeiten in S-Plus zur Berechnung solcher äußerer Produkte:

```
a%o%b
outer(a,b,*)
```

Dabei ist die `outer`-Funktion weitaus flexibler. Der (Default) `*` für die Produktoperation kann durch beliebige Funktionen ersetzt werden.

Diagonalmatrix. Die `diag`-Funktion kann einerseits Diagonalmatrizen erstellen oder aber Diagonalelemente einer Matrix extrahieren `diag`

Standards der Linearen Algebra

Invertierung. Invertierung von Matrizen und das Lösen linearer Gleichungssysteme erfolgt mit der Funktion `solve`. Dabei wird mit `solve`

```
solve(A)
```

die Inverse der Matrix `A` berechnet und

```
solve(A,b)
```

löst das Gleichungssystem $Ax = b$.

Cholesky-Zerlegung. Die Cholesky-Zerlegung $A = U^T U$ einer nicht negativ- `chol`
definiten Matrix A wird mit der Funktion `chol` durchgeführt.

Eigenwerte. Eigenwerte und Eigenvektoren einer symmetrischen Matrix werden mit der Funktion `eigen` berechnet. Das Resultat der Funktion ist gerade eine Liste mit den Komponenten `values` und `vectors`. `eigen`

Singulärwertzerlegung. Die Singulärwertzerlegung einer $n \times p$ Matrix X gemäß $X = U \Lambda V^T$, wobei U und V gerade $n \times \min(n,p)$ und $p \times \min(n,p)$ Matrizen mit orthonormalen Spalten beschreiben und Λ eine Diagonalmatrix ist, wird über die Funktion `svd` erhalten. `svd`

Qr-Zerlegung. Die QR-Zerlegung $M = QR$, wobei Q eine Orthonormal-Matrix und R eine obere Dreiecksmatrix bezeichnet, kann mit der Funktion `qr` durchgeführt werden. `qr`

Matrix → Data-Frame. Bereits angesprochen wurde die Konvertierung von Matrizen in Data-Frames. Diese Operation erfolgt mit der Funktion `as.data.frame`. Als Resultat erhält man einen Data-Frame, mit den Spalten der ursprünglichen Matraix als Variablen. Besitzt die Matrix ein `dimnames`-Attribut, so werden die dimnames als Zeilennamen übernommen. `as.data.frame`

Data-Frame → Matrix. Die Konvertierung von Data-Frames in Matrizen erfolgt über die Funktionen `as.matrix` bzw. `data.matrix`. Welche Funktion benutzt wird, hängt von der Gestalt des Data-Frames ab. `as.matrix` `data.matrix`

2.8.7 Die Funktion 'apply'

Die Funktion `apply` erlaubt es, Funktionen über Array-Indizes laufen zu lassen. `apply`
Der (S-Plus-interne) Datensatz `iris` beinhaltet beispielsweise als $50 \times 4 \times 3$

Array die Beobachtungen von vier Merkmalen an je 50 Exemplaren von 3 Arten der Pflanze:

```
iris[1,,]
         Setosa Versicolor Virginica
Sepal L.    5.1        7.0       6.3
Sepal W.    3.5        3.2       3.3
Petal L.    1.4        4.7       6.0
Petal W.    0.2        1.4       2.5
```

stellt die entsprechenden Messungen für das erste Untersuchungsobjekt dar. Um nun z.B. für jedes der vier Merkmale artspezifische Mittelwerte zu berechnen, kann die `apply`-Funktion benutzt werden. Die Argumente von `apply` sind

```
apply(x,margin=,fun=,...)
```

wobei `x` das Array beschreibt, `margin` den Indexvektor beschreibt, bezüglich dem die Funktion operieren soll, `fun` erhält den Funktionsnamen, in ... können weitere, für die Funktion zusätzliche Argumente übergeben werden.
Für das beschriebene Beispiel somit

```
apply(iris, c(2,3), mean)
```

```
         Setosa Versicolor Virginica
Sepal L.  5.006      5.936     6.588
Sepal W.  3.428      2.770     2.974
Petal L.  1.462      4.260     5.552
Petal W.  0.246      1.326     2.026
```

2.9 Faktor- und Listenoperationen

Betrachte als Beispiel einen Data-Frame mit den Umsätzen (in Ecu) der größten europäischen Unternehmen:

```
nation umsatz angest        firma
1      D  37042 368226 Daimler-Benz
2      D  31689 257561   Volkswagen
3      D  29640 365000      Siemens
4      D  23860  92091         Veba
5      D  23089 136990         BASF
```


.
.
.

Man beachte, daß die Character-Vektoren (`firma, umsatz`) hier Faktoren sind (auch logische Vektoren wären hier Faktoren).
Die Benutzung der Funktion `table` liefert nun eine bezüglich der Faktoren (eines oder mehrerer) erstellte Häufigkeits- (Kontingenz-) Tabellen: `table`

```
table(nation)
  A  B CH  D DK  E  F GB  I IR  L  N  P
 20 20 20 48 20 20 20 20 20 20 10 20 20
```

Werden der elementaren `plot`-Funktion als erstes Argument ein Faktor angegeben, so liefert die Funktion faktorspezifische Boxplots (s. auch Abb. 1): `plot`

```
plot(nation,umsatz)
```

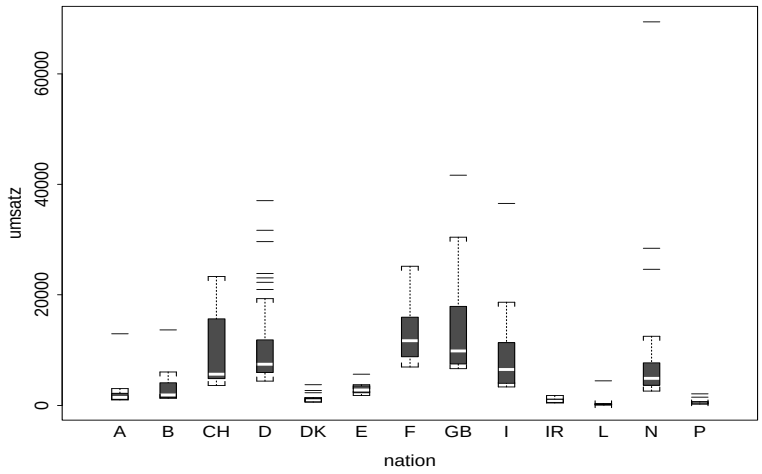

Abbildung 1: Resultat von 'plot(nation,umsatz)'

Mit der Funktion `tapply` lassen sich nun ähnlich der `apply`-Funktion beliebige Funktionen bezüglich der einzelnen, zu spezifizierenden Faktoren erstellen. So wird durch

```
tapply(umsatz,nation,mean)
       A       B     CH        D      DK       E        F       GB
 2157.15 3118.65 9045.3 10673.88 1147.45 2908.75 12883.75 14147.65
       I  IR     L       N     P
 8949.25 852 595.6 10537.6 529.2
```

beispielsweise der durchschnittliche Umsatz pro Nation berechnet.

3 Daten in S-Plus

3.1 Einlesen von Daten

Gewöhnlich werden Datensätze nicht direkt am Terminal eingegeben, sondern eingelesen.

3.1.1 Die `read.table` Funktion

Der einzulesende Datensatz liegt als Data-Frame vor, und zwar in folgendem Format:

- die erste Zeile enthält die Variablennamen
- die nachfolgenden Zeilen beginnen mit einem Zeilenlabel

`read.table`

Der Datensatz kann dann eingelesen werden gemäß

```
read.table('dateiname')
```

Liegen keine Zeilenlabels vor, so muß zusätzlich die Option `header=T` angegeben werden, um die Variablennamen korrekt einzulesen (man betrachte auch den help-file zur `read.table`).

3.1.2 Die `scan` Funktion

`scan`

Die `scan`-Funktion ermöglicht dem Benutzer ein flexibleres Einlesen von Daten:

Keyboard-Eingaben.

```
x<-scan()
.
. <Eingabe>
.
<Leerzeile>
```

d.h. nach der (durch einzelne Leerzeichen getrennten) Eingabe der Daten wird zum Abschluß zunächst die Return-Taste gedrückt, und dann noch einmal zum Beenden der Leerzeile.

Einlesen von Character-Vektoren.

```
x<-scan(," ")
```

Hier signalisiert die Option "" , daß die nachfolgende Eingabe aus Charakters besteht.

Einlesen von Dateien. Hier wird als erstes Argument der Dateiname angegeben:

```
x<-scan("dateiname")
```

Sei jetzt angenommen, daß die Datei aus mehreren Vektoren bestehe, wobei der erste vom Typ Character ist, die nachfolgenden numerisch. Dann wird der Funktion als zweites Argument eine sogenannte 'Dummy'-Liste übergeben mit entsprechenden Komponentennamen `name1,...`:

```
x<-scan("dateiname",list(name1="",name2=0,name3=0,...)
```

Einlesen von Daten-Matrizen.

```
x<-matrix(scan('dateiname'),ncol=n,byrow=T)
```

3.2 Input/Output

`write`

Eine Character-Darstellung von einem beliebigen S-Plus-Vektors `x` kann mit der `write`-Funktion erfolgen:

```
write(x,file="dateiname")
```

`write.table`

Es ist jedoch zu beachten, daß die Funktion nicht viele Möglichkeiten bezüglich der Formatierung zuläßt.

Ein Data-Frame läßt sich sinnvollerweise am besten mit `write.table` in eine Datei schreiben:

```
write.table(dataframe,file="",sep="")
```

Zusätzlich läßt `write.table` eine Option `dimnames.col=` zu, über welche die Kontrolle der Zeilen- und Spaltenlabels ermöglicht wird.

3.2.1 Steuerung zwischen Datei und S-Plus-Session

Befindet sich ein S-Plus-Kommando oder eine Kommandofolge in einer Datei, so läßt sich die Kommandofolge mit der `source`-Funktion ausführen: `source`

```
source('dateiname')
```

Umgekehrt läßt sich der Output eines S-Plus-Kommandos über die `sink`-Funktion in eine Datei überführen: `sink`

```
sink('dateiname')
```

übergibt alle nachfolgenden Resultate der Datei 'dateiname'.

Die Überführung von S-Plus-Objekten erfolgt mit den Funktionen `dump` bzw. `data.dump`. Die Funktionen erhalten die Objekte in Form von Character-Vektoren, welche in die angegebene Datei in Form von Zuweisungen geschrieben werden. Dadurch wird ermöglicht, die Objekte mit der bereits erwähnten `source`-Funktion erneut zu laden und damit zu-rekonstruieren. Wird `data.dump` benutzt, so erfolgt das erneute Einlesen mit `data.restore`. `dump` `data.dump`

4 Graphik

Die Erstellung von Graphiken und die graphische Darstellung von S-Plus-Objekten war eine der wesentlichen Anforderungen, die bei der Konzeption von S-Plus gestellt wurden. Dabei reichen die Darstellungsmöglichkeiten von einfachen Plots bis zu qualitativ hoch anzusiedelnden Graphiken. Der Benutzer wird schnell feststellen, wie komplex die Menge der Darstellungsmöglichkeiten ist und wie groß die Zahl graphischer Steuerungsparameter.

4.1 Graphik-Device

Vor der expliziten Erstellung von Graphiken muß zunächst der sogenannte Graphik-Device geöffnet werden. Dabei handelt es sich in der Regel um ein Fenster, welches geöffnet wird. Über die Online-Help läßt sich abfragen, welche Graphik-Devices zur Verfügung stehen. Eine Liste der für S-Plus verfügbaren Graphik-Devices liefert folgende Tabelle: `devices`

`motif`	X11-Window Systeme
`openlook`	X11-Window Systeme
`iris4d`	Silicon Graphics Iris-Workstations
`suntools`	Sun unter SunView
`win.graph`	Windows-Version von S-Plus
`postscript`	Postscript Drucker
`hplj`	Hewlett-Packard LaserJet Drucker
`hpgl`	Hewlett-Packard HP-GL Drucker
`win.printer`	Windows Drucker
`pic`	troff Bildverarbeitungssprache
`fig`	mit xfig zu benutzen

Die einfachste Möglichkeit, einen Graphnik-Device zu starten, ist beispielsweise

```
motif()
```

Das Schließen des Devices erfolgt mit `graphics.off`

```
graphics.off()
```

4.2 Hardcopies

Für graphische Ausdrucke wird der entsprechende Item im Menü des Graphik-Device angeklickt. Der entsprechende Drucker wird unter dem `Properties`-Button eingestellt. Soll direkt eine Postscript-Datei erstellt werden, empfiehlt es sich, unter `Properties` direkt das Kommando

```
cat > plot_file_name <
```

im `Print`-Item einzugeben.

4.3 Grundlegende Plot-Funktionen

Ähnlich wie bei sämtlichen anderen Gebieten, die von S-Plus unterstützt werden, so werden auch hier einerseits sogenannte 'low level graphical functions' sowie andererseits 'high level plotting functions' bereit gestellt. Dabei dienen letztere der Erstellung kompletter Graphiken, während die erstgenannten bereits existierende Graphiken um weitere Elemente bereichern bzw. in Kombination eigene Graphiken ermöglichen.

Beispiele für die high level plotting functions sind:

`boxplot`	Boxplots
`hist`	Histogramme
`plot`	Scatterplots
`tsplot`	Zeitreihenplots

Die einfachste Möglichkeit , ein S-Plus-Objekt graphisch darzustellen, ist die Benutzung der `plot`-Funktion (s. auch Beispielsitzung). Häufig ist das Resultat jedoch nicht hinreichend geeignet für eine weiter Verwendung. `plot`

Barcharts. Um Barcharts (Säulendiagramme) darzustellen, läßt sich die Funktion `barplot` verwenden. `barplot`

Linien-/Scatterplots. Ein einfacher Linien- oder Scatterplot wird über die bereits erwähnte `plot`-Funktion erstellt. Notwendig ist hier die Übergabe von Argumenten `x` und `y`, die beide Vektoren gleicher Länge, eine Matrix mit zwei Spalten oder ein Liste mit Komponenten `x` und `y` sein können. Unter den zahlreichen, individuell einstellbaren Optionen, den 'graphischen Parametern' sind die gebräuchlichsten in der folgenden Tabelle aufgeführt:

`type='c'`	Plot-Typ, typische Werte für `c` sind `p` für Punkte `l` für Linien `b` für Punkte und Linien `s` für Treppendarstellungen `o` für überlagerte Punkte und Linien `n` für no plotting (Axen werden dennoch dargestellt)
`axes=L`	`T` oder `F` für Axenerstellung
`xlab='string'`	Axen-Labels für x-Achse
`ylab='string'`	Axen-Labels für y-Achse
`sub='string'`	Untertitel
`main='string'`	Haupttitel (Überschrift des Plots)
`xlim=c(lo,hi)`	Achsenrange für x-Achse
`ylim=c(lo,hi)`	Achsenrange für y-Achse

Um dem Plot zusätzlich Linien, Texte etc. anzugeben, lassen sich unter anderem folgende Funktionen verwenden:

`points(x,y,...)`	Hinzufügen von zusätzlichen Punkten
`lines(x,y,...)`	Hinzufügen von zusätzlichen Linien
`text(x,y,labels,...)`	Hinzufügen von Text an vorgegebenen Punkten im Plot
`abline(a,b,...)`	Linie mit Intercept `a` und Steigung `b`
`abline(h=c,...)`	Horizontale Linie durch `c`
`abline(v=c,...)`	Vertikale Linie durch `c`
`abline(lmobjekt,...)`	durch ein `lmobjekt` definierte Linie

Multivariate Plots. Eine sukzessiv paarweise Darstellung von Variablen eines multivariaten Datensatzes erfolgt mit der Funktion `pairs`. Die Darstellung der Plots erfolgt in Gestalt einer Matrix. `pairs`

Dynamische Graphik. Eine simple Interaktion mit (der unteren Hälfte von) Scatterplots wird über die Funktion `brush` ermöglicht (brush=bürsten). `brush`

Der Benutzer kann hier Beobachtungspunkte hervorheben, ausradieren, mit Labeln versehen usw. Zusätzlich können drei Variablen ausgewählt werden, welche über eine simple 3D-Rotation weitere Informationen liefern können.

Darstellung von Oberflächen. Auf einem zweidimensionalen Gitter definierte Funktionen können über die Funktionen `contour`, `persp` und `image` dargestellt werden. `contour`

5 Verteilungen und Zusammenfassung von Daten (klassische univariate Statistik)

5.1 Zusammenfassung von Daten

Bereits erwähnt wurden einige Standard-Kenngrößen, bereitgestellt über die Funktionen `mean`, `var`, `median`. Die Funktion `summary` liefert direkt eine Auflistung der deskriptiven Größen Mittelwert, Quantile, Anzahl der fehlenden Werte. `Kenngroessen`
Wird den Funktionen `var` und `cor` eine Datenmatrix als Argument übergeben, so werden Covarianz- bzw. Korrelationsmatrix berechnet.

Mit der Funktion `quantile` können die Quantile eines Datenvektors berechnet werden. Die klassische '5 Zahlen'-Zusammenfassung $(q_0, q_{0.25}, q_{0.5}, q_{0.75}, q_1)$ liefert die Funktion defaultmäßig:

`quantile(x,c(0.3,0.6))`	30 und 60 Prozent Quantil von `x`
`quantile(x)`	5-Zahlen-Zusammenfassung von `x`

5.1.1 Histogramme/Stem-und-Leaf Darstellungen

Ein Standard-Histogramm wird über die Funktion `hist` erstellt. Der Funktion können eine ganze Reihe zusätzlicher Argumente übergeben werden, die wesentlich liefert die folgende Tabelle: `hist`

`probability=T`	für relative Häufigkeiten
`style='old'`	herkömmliche Darstellung ohne Grauschattierung
`nclass=n`	Klassenanzahl
`breaks=c(...)`	Klassengrenzen

Der Datensatz `buffalo` beinhaltet beispielsweise die in Buffalo, NY, in den Wintern der Jahre 1910/11 bis 1972/73 gefallene Schneemenge. Das Histogramm mit den Defaulteinstellungen gemäß dem Aufruf

```
hist(buffalo)
```

liefert die nachstehende Abbildung 2. Ein Stem-und-Leaf Plot ist eine Art modifiziertes Histogramm in horizontaler Darstellung. Dazu werden die Daten `stem`

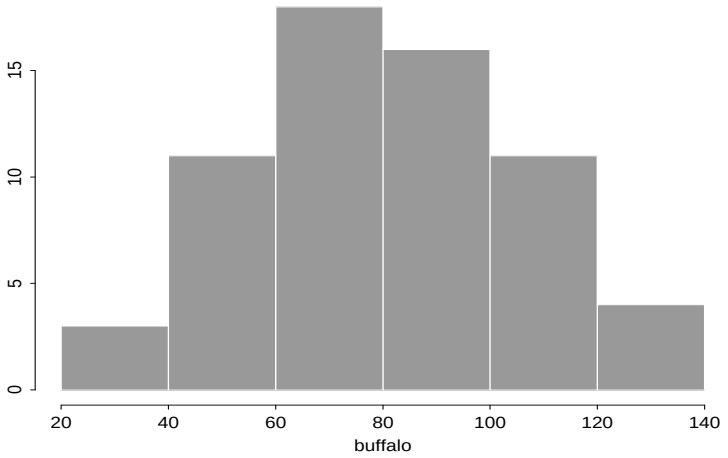

Abbildung 2: Resultat von 'hist(buffalo)'

ebenfalls in Klassen eingeteilt, wobei die 'Höhe' gerade durch die nachfolgenden Stellen in geordneter Reihenfolge erscheinen.
Für den bereits erwähnten `buffalo`-Datensatz erhält man die folgende Darstellung:

```
stem(buffalo)

N = 63   Median = 79.6
Quartiles = 63.6, 98.3

Decimal point is 1 place to the right of the colon

   2 : 5
   3 :
   4 : 00079
   5 : 01235568
   6 : 04569
   7 : 111234688899
   8 : 01223445579
   9 : 000178
  10 : 12445
  11 : 000446
  12 : 0156
```

5.1.2 Boxplots

`boxplot`

Boxplots dienen als (äußerst) nützliche Darstellung der Form einer Stichprobenverteilung. Die zentrale Box zeigt dabei das Datenzentrum (grob gesprochen der Interquartilsrange), mit dem als Linie eingetragenen Median. Gestrichelte Linien gehen zu den Extremwerten und äußerst extreme Punkte werden einzeln dargestellt. Abbildung 3 liefert einen Default-Boxplot gemäß

```
boxplot(buffalo)
```

für den `buffalo`-Datensatz.

5.2 Wahrscheinlichkeitsverteilungen

Betrachtet werden univariate Verteilungen, d.h. Verteilungen einer Zufallsvariablen X mit Werten im Raum der reellen Zahlen $\mathbb{R}$. Definiert sind solche

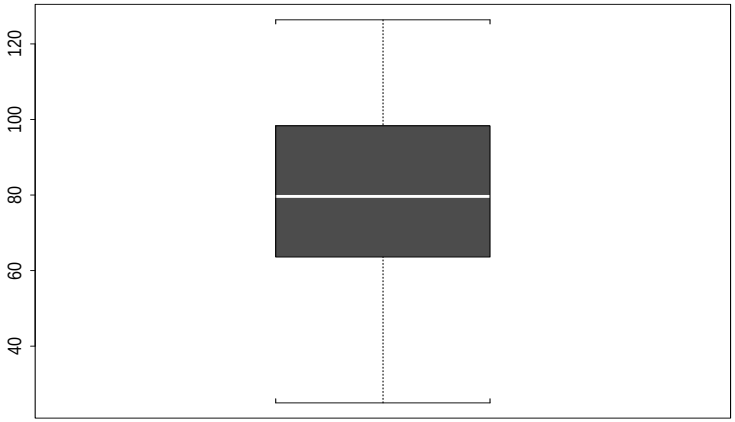

Abbildung 3: Resultat von 'boxplot(buffalo)'

(Standard-) Verteilungen über die zugehörigen Wahrscheinlichkeitsdichtefunktion (bei stetigen Variablen) bzw. die Wahrscheinlichkeitsfunktion (für diskrete Zufallsgrößen). Um eine nicht allzu verwirrende Bezeichnungsvielfalt einzuführen, wird im folgenden allgemein von Dichte gesprochen (in der tat lassen sich mathematisch die Wahrscheinlichkeitsfunktionen auch als Dichten auffassen).
Mit $F(x) := \Pr(X \leq x)$ wird die kumulierte Verteilungsfunktion (kurz CDF = cumulative distribution function) bezeichnet. Die Inverse der CDF, falls existent, wird als Quantilsfunktion $Q(u) := F^{-1}(u)$ bezeichnet. Die Quantilsfunktion liefert somit die 'Prozentpunkte' der Verteilung.

`Verteilungen`

S-Plus beinhaltet eine Reihe von Standardverteilungen, für welche die Dichte, die CDF und die Quantilsfunktion berechnet werden können. Die zugehörigen Funktionsnamen bestehen dann aus zwei Teilen. Der erste Buchstabe im Namen beschreibt die jeweilige Funktion, d.h. `p` für CDF, `q` für die Quantilsfunktion, `d` für die Dichte; angeschlossen ist der Verteilungstyp, d.h. beispielsweise

`norm` für die Normalverteilung, `binom` für die Binomialverteilung etc. Die Argumente sind hier einerseits funktionsspezifisch (so steht `q` für die Dichte und die CDF, `p` für die Quantilsfunktion) sowie verteilungsspezifisch (Verteilungsparameter, z.B. μ, σ für die Normalverteilung). Die nachstehende Tabelle gibt einen Überblick der zur Verfügung stehenden Verteilungen:

Verteilung	S-Plus-Name	Parameter
Beta	`beta`	`shape1, shape2`
Binomial	`binom`	`size, prob`
Cauchy	`cauchy`	`location, scale`
Chi-Quadrat	`chisq`	`df`
Exponential	`exp`	`rate`
F	`f`	`df1, df2`
Gamma	`gamma`	`shape`
Geometrische	`geom`	`prob`
Hypergeometrische	`hyper`	`m, n, k`
Log-Normal	`lnorm`	`meanlog, sdlog`
Logistische	`logis`	`location, scale`
Negativ-Binomial	`nbinom`	`size, prob`
Normal	`norm`	`mean, sd`
Poisson	`pois`	`size`
Stabile	`stab`	`lambda`
T	`t`	`df`
Gleich	`unif`	`min, max`
Weibull	`weibull`	`shape`
Wilcoxon	`wilcox`	`m, n`

5.2.1 Quantile-Quantile Plots

`qqplot`

Um eine Stichprobenverteilung mit einer theoretischen Verteilung zu vergleichen, eignen sich die sogenannten Quantile-Quantile Plots, kurz QQ-Plot. Wie bereits erwähnt, ist die Quantilsfunktion einer Stichprobe gerade die Inverse der empirischen CDF:

$$Quantil(p) = \min\{z|\ \text{Anteil } p \text{ derDaten } \leq z\}$$

Sollen nun zwei Stichprobenverteilungen (seien `x` und `y` die entsprechenden Stichproben) miteinander verglichen werden, so kann die Funktion

```
qqplot(x,y,...)
```

herangezogen werden. Die Funktion liefert einen Plot der jeweiligen Quantilsfunktionen gegeneinander.
Wird die Stichprobenverteilung `x` mit der (theoretischen) Standardnormalverteilung verglichen, so wird die der Stichprobe `x` zugehörige Quantilsfunktion gegen die theoretische Quantilsfunktion der Standardnormalverteilung dargestellt; hierzu läßt sich die Funktion `qqnorm`

```
qqnorm(x)
```

benutzen.

5.2.2 Erzeugung zufälliger Daten

Für sämtliche in der Tabelle des letzten Abschnitts dargestellten Verteilungen lassen sich unabhängige Zufallsstichproben generieren. Die Kennung erfolgt hier über den ersten Buchstaben `r` (für random=Zufall) und das erste Argument `n`, die Länge des zu erzeugenden Vektors: `random`

```
rnorm(100,1,2)
```

erzeugt einen Vektor der Länge 100 mit Elementen, die eine Zufallsstichprobe einer $N(\mu = 1, \sigma = 2)$-Verteilung darstellen.

5.2.3 Dichteschätzung

Im folgenden einige Anmerkungen zur nichtparametrischen Dichteschätzung.
Bereits in Abschnitt 5.1 wurden Histogramme vorgestellt. Histogramme liefern bereits einfache Dichteschätzungen. Um die Schätzungen transparenter zu machen, bietet es sich an, sogenannte 'Häufigkeitspolygone' zu benutzen; diese Darstellung ergibt sich durch die Verbindung der Mittelpunkte der einzelnen Histogrammblöcke.
Betrachtet man Häufigkeitspolygone für den Datensatz der Regenfälle in Buffalo, so erhält man Schätzungen wie beispielsweise in Abbildung 4. Offensichtlich liefert die Darstellung zwar eine alternative Darstellungsmöglichkeit zu Histogrammen, dennoch ist die Aussagekraft wiederum begrenzt, insbesondere durch die starre Verbindung der Mittelpunkte: Die einzige in S-Plus verfügbare Funktion zur direkten Dichteschätzung ist `density`, welche sogenannte 'Kern- `density`

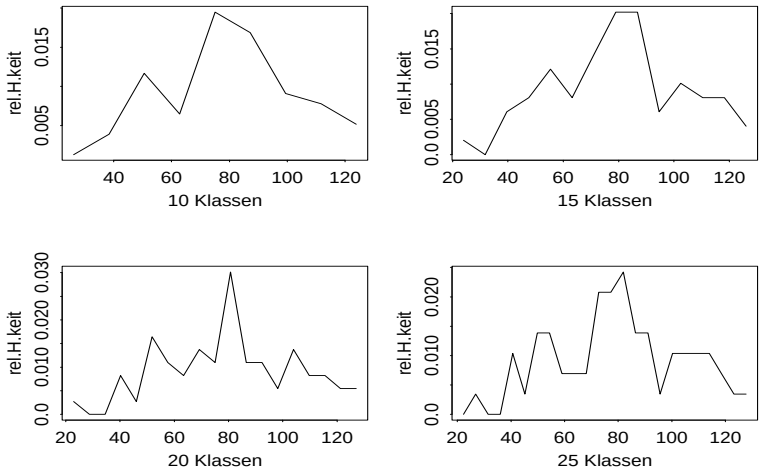

Abbildung 4: Resultat von verschiedenen Dichteschätzungen

dichteschätzungen' durchführt:

$$\hat{f}(x) = \frac{1}{b}\sum_{j=1}^{n} K\left(\frac{x - x_j}{b}\right)$$

dabei ist $x_1, \ldots, x_n$ eine Stichprobe und $K(\cdot)$ die sogenannte Kernfunktion bezüglich einer Bandbreite b. Als Kernfunktion wird in der Regel eine Wahrscheinlichkeitsdichtefunktion verwendet. Die Wahl der Bandbreite spiegelt den Trade-Off zwischen Glattheit und Erwartungstreue wieder.
Man betrachte erneut den Datensatz für die Regenfälle in Buffalo. Die Abbildung 5 zeigt hier vier verschiedene Dichteschätzungen für den Datensatz. Dabei zeigt das erste Bild den Aufruf gemäß

```
plot(density(buffalo),type='l',xlab='default breite',ylab='')
```

und die nachfolgenden Bilder Dichteschätzungen gemäß

```
plot(density(buffalo,width=10),type='l',xlab='breite=10',ylab='')
plot(density(buffalo,width=15),type='l',xlab='breite=15',ylab='')
plot(density(buffalo,width=20),type='l',xlab='breite=20',ylab='')
```


d.h. mit immer größer werdender Bandbreite. Es ist deutlich zu erkennen, wie die Schätzungen immer glatter werden, wobei eventuell lokale Schwankungen allerdings versteckt werden.

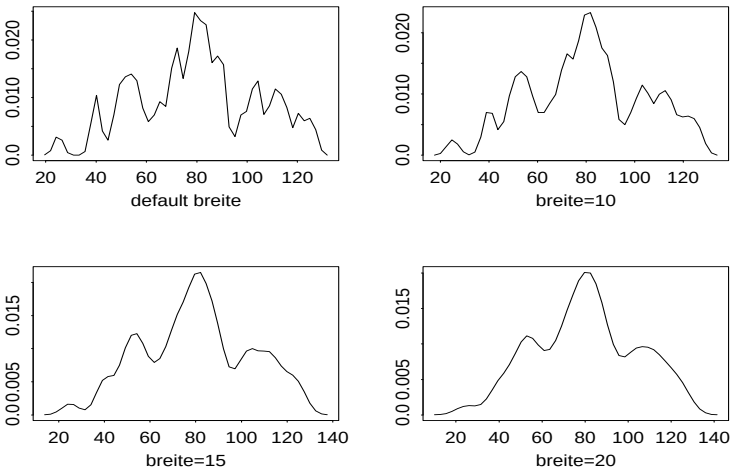

Abbildung 5: Resultat von verschiedenen Dichteschätzungen

5.3 Klassische univariate Statistik

Klassische Prozeduren der Testtheorie und die Berechnung von Konfidenzintervallen bilden eine weitere Sektion in S-Plus. Einige dieser Prozeduren werden anhand eines einfachen Beispiels vorgestellt:

Beispiel 5.1 *(aus Box, G.E.P., Hunter, W.G. und S. Hunter (1978): 'Statistics for Experimenters', New York: John Wiley and Sons) Betrachtet werden zwei Materialien A und B für Schuhe, welche zufällig dem linken oder rechten Fuß der Testpersonen zugewiesen wurden. Die beobachteten Werte des Materialabriebs liefert folgende Tabelle:*

Personnen-Nummer	*Material A*	*Material B*
1	*13.2 (L)*	*14.0 (R)*
2	*8.2 (L)*	*8.8 (R)*
3	*10.9 (R)*	*11.2 (L)*
4	*14.3 (L)*	*14.2 (R)*
5	*10.7 (R)*	*11.8 (L)*
6	*6.6 (L)*	*6.4 (R)*
7	*9.5 (L)*	*9.8 (R)*
8	*10.8 (L)*	*11.3 (R)*
9	*8.8 (R)*	*9.3 (L)*
10	*13.3 (L)*	*13.6 (R)*

Zunächst müssen die Daten eingelesen werden:

```
schuhe<-scan(,list(A=0,B=0))
.
. <Terminal-Eingabe paarweise)
.
attach(schuhe)
boxplot(schuhe)
```

Mit `attach` *werden die Variablennamen* `A` *und* `B` *sichtbar gemacht, und im Anschluß wird ein Boxplot (s. auch Abb. 6) erzeugt.* `attach`

Die nächsten Aufrufe liefern einen Ein-Stichprobentest für die Hypothese H_0*:* $\mu = 10$ *sowie ein Schintervall für* μ*:* `t.test`

```
t.test(A,mu=10)

 One-sample t-Test

data:  A
t = 0.8127, df = 9, p-value = 0.4373
alternative hypothesis: true mean is not equal to 10
95 percent confidence interval:
  8.876427 12.383573
sample estimates:
```


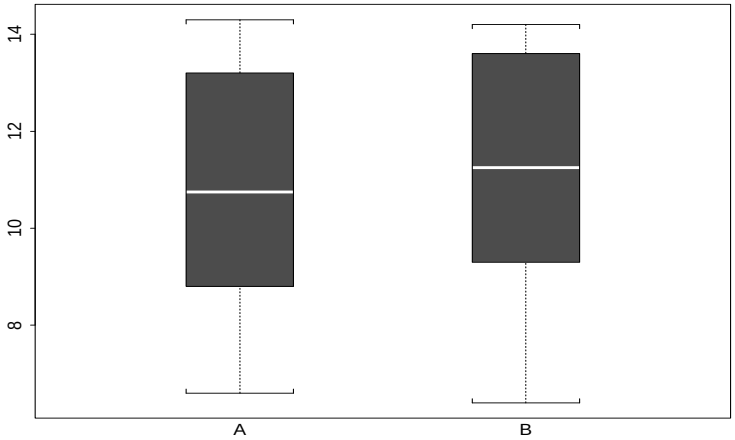

Abbildung 6: Resultat von 'boxplot(schuhe)'

```
mean of x
    10.63

t.test(A)$conf.int
[1]  8.876427 12.383573
attr(, "conf.level"):
[1] 0.95
```

Darüberhinaus läßt sich hier noch ein Zwei-Stichprobentest für verbundene Stichproben durchführen (es ist zu beachten, daß hierfür die Gleichheit der Varianzen vorausgesetzt wird): `wilcox.test`

```
t.test(A,B,paired=T)

 Paired t-Test

data:  A and B
```

```
t = -3.3489, df = 9, p-value = 0.0085
alternative hypothesis: true mean of differences is not equal to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
 mean of x - y
        -0.41

wilcox.test(A,B,paired=T)

 Wilcoxon signed-rank test

data:  A and B
signed-rank normal statistic with correction Z = -2.4495, p-value = 0.0143
alternative hypothesis: true mu is not equal to 0
```

Eine Auflistung der klassischen Tests liefert die folgende Liste:

```
binom.test     chisq.test    cor.test         fisher.test
friedman.test  kruskal.test  mantelhaen.test  mcnemar.test
prop.test      t.test        var.test         wilcox.test
```

6 Lineare Statistische Modelle

6.1 Regressionsmodelle

Das Herz der klassischen Statistik bilden lineare (Regressions-) Modelle. Zunächst soll ein Beispiel in die Thematik einführen, bevor einige wesentliche Funktionen vorgestellt werden:
Erhoben werden drei Merkmale bei 144 Katzen: Geschlecht ('Sex', nominal: männlich/weiblich), Herzgewicht ('Hwt', kardinal: in Gramm), Körpergewicht ('Bwt', kardinal: in Kilogramm). Mögliche Fragestellungen sind hier unter anderem:

- Existiert ein Zusammenhang zwischen Körper- und Herzgewicht, aus dem auf das Geschlecht der Katze geschlossen werden kann?

- Sind bei geschlechtsspezifisch angepassten linearen Einfachregressionen die Koeffizienten signifikant?

Der angeführte Datensatz wurde bereits von Fisher untersucht. Mögliche Vorgehensweisen werden im folgenden exemplarisch vorgestellt. In den nachfolgenden Abschnitten werden die verwendeten Funktionen näher erläutert.

```
tapply(Hwt,Sex,mean)/tapply(Bwt,Sex,mean)
F M
3.89991 3.904373
```

Betrachtung des geschlechtsspezifischen Verhältnisses von Herz- zu Körpergewicht. Offenbar sind hier keine auffallenden Unterschiede zu erkennen.

```
tapply(Hwt/Bwt,Sex,mean)
F M
3.915119 3.894547
```

```
catsF<-lm(Hwt ~
Bwt,data=cats,subset =
Sex=='F')
catsM<-update(catsF,subset
= Sex=='M')

summary(catsF)
summary(catsM)
```

Anpassung von geschlechtsspezifischen linearen Einfachregressionen und Anschauen der Resultate. Es zeigt sich, dass das Absolutglied bei der männlichen Anpassung nicht signifikant ist, bei der weiblichen gerade an der Grenze zur Signifikanz.

```
plot(Bwt,Hwt,xlab='Koerpergewicht
(kg)',ylab='Herzgewicht
(g)', type='n')
text(Bwt,Hwt,c('+','-')[Sex])
legend(2,20,
c('weibl.','maennl.'),pch="+
-",lty=c(2,-1,3),bty='n')

lines(Bwt[Sex=='F'],fitted(catsF),lty=1)
lines(Bwt[Sex=='M'],fitted(catsM),lty=2)
```

Erzeugung eines geeigneten Regressionsplots

```
catsMF<-lm(Hwt
~ Sex/Bwt - 1,
data=cats)
catsMF1<-lm(Hwt ~
Sex/factor(Bwt),
data=cats,
singular.ok=T)

anova(catsMF,catsMF1)
```

Erneute Regressionsanpassungen, dabei wird das Geschlecht als Faktor hinzugefügt

Die wesentliche Funktion für lineare Modelle ist `lm`: `lm`

```
lm(formula,data,weights,subsets,na.action)
```

Die einzelnen Argumente haben folgende Funktion:

`formula`	Formel für das zu nutzende Modell
`data`	Data-Frame (optional)
`weights`	positiver Gewichtsvektor (default: gleiche Gewichte)
`subsets`	Auswählen eines Teil-Data-Frames (default: sämtliche Zeilen)
`na.action`	Strategie zur Handhabung von missing values (default: m.v. sind nicht erlaubt)

Das `formula`-Argument (s. Abschnitt 6.2) ist hier von der Form

unabhängige Variable $\sim$ regressor1 + regressor2 + ...

Der Output eines `lm`-Aufrufs ist ein Objekt vom Typ Liste. Auf die Listenelemente kann wie bereits erläutert zugegriffen werden, oder aber es wird das gesamte Objekt einer sogenannten 'generic function' übergeben. Hier gibt es unter anderem folgende Funktionen:

`print`	einfache Darstellung, Übersicht
`summary`	gewöhnlicher Regressions-Output (Standardzusammenfassung)
`coef`	Koeffizienten
`resid`	Residuen
`fitted`	Schätungen, angepasste Werte
`deviance`	Residuenquadratsumme
`plot`	diagnostische Plots

6.2 Die S-Plus-Formel-Darstellung für statistische Modelle

Ein gewöhnliches multiples Regressionsmodell mit drei Regressoren hat die folgende (algebraische) Gestalt:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon$$

wobei ε einen Fehlerterm beschreibt mit Erwartungswert 0 und konstanter Varianz. Weiter beschreibt y die (unabhängige) Response-Variable, den Regressand, und x_1, x_2, x_3 die (abhängigen) Einflußgrößen, die Regressoren. y hat hier Vektor- oder Matrixgestalt, die Regressoren ebenfalls (oder aber auch als Faktor, was hier zunächst nicht betrachtet wird).
Als Modell-Formel in S-Plus wird der angeführte Regressionsansatz wie folgt angegeben: `formula`

```
y ~ 1 + x1 + x2 + x3
```

Dabei steht die 1 für die Einbeziehung des Absolutglieds β_0. Die einzubeziehenden Regressoren werden über `+` Zeichen verbunden. Der Ausschluß einer Größe erfolgt über ein `-` Zeichen; z.B.

```
y ~ -1 + x1 + x2 + x3
```

liefert den Regressionsansatz ohne Absolutglied, d.h. einen Ansatz der Form

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon$$

Die folgende Tabelle gibt ein paar Beispiele für die Verwendung solcher Formel-Darstellungen:

`y ~ x` `y ~ 1 + x`	Beide Darstellungen vollziehen eine lineare Einfachregression von `y` auf `x`; dabei wird in der ersten Form das Absolutglied implizit, in der zweiten explizit angegeben
`y ~ x - 1`	Lineare Einfachregression von `y` auf `x` ohne Absolutglied (d.h. durch den Ursprung)
`log(y) ~ x1 + x2`	Lineare Einfachregression einer transformierten Variablen auf zwei Regressoren (Absolutglied implizit)
`y ~ poly(x,2)`	Polynomiale Regression zweiten Grades von `y` auf `x`

6.3 Varianzanalyse - Vergleich von Modellen

Auf eine ähnliche Weise wie die `lm`-Funktion arbeitet `aov` (analysis of variance). Es ist anzumerken, daß die Benutzung von 'aov' weit über die gewöhnliche Anpassung von Regressionsfunktionen (wie bei `lm`) hinausgeht; insbesondere ermöglicht die Funktion die Anpassung von Multistratum-Modellen (Blockdesigns, etc). `aov`

Den Vergleich mehrerer Modelle liefert die Funktion `anova`. Dabei liefert der Aufruf `anova`

```
anova(modell1, modell2, ...)
```

die Unterschiede zwischen den angepassten Modellen.

6.4 Updating von Modellen

Werden Modelle betrachtet, die aus einem bereits existierenden (angepassten) Modell entstehen sollen, entweder durch Hinzunahme eines Regressors oder durch Zurücknahme, so läßt sich die `update`-Funktion benutzen: `update`

```
modell.neu<-update(modell.alt, formel.neu)
```

Innerhalb der neuen Formel läßt sich nun der Name . benutzen. Sei beispielsweise durch `.`

```
modell1<-lm(y ~ x1 + x2 + x3, data=data.frame1)
```


eine Regression von `y` auf drei Regressoren des Data-Frames `data.frame1` durchgeführt und das Ergebnis in der Variablen, dem Objekt `modell1` abgelegt. Soll nun das Modell lediglich um einen weiteren Regressor, z.B. `x5` erweitert werden, so kann dies unter Verwendung der `update`-Funktion und dem .-Namen geschehen:

```
modell.neu<-update(modell1, . ~ . + x5)
```

Hier wird als Basis das alte `modell1` genommen, unter Verwendung der alten Modell-Formel und Hinzufügen des fünften Regressors. Durch

```
modell.neu<-update(modell1, sqrt(.) ~ .)
```

wird beispielsweise der (durch Wurzel) transformierte Regressand durch die bisherigen Regressoren angepasst.
Eine weitere Möglichkeit zur Verwendung des .-Namens liefert

```
anpassung<-lm(y ~ .,data=data.frame)
```

Dabei wird eine Regression von `y` auf sämtliche im Data-Frame vorhandenen Variablen durchgeführt (praktisch, wenn sehr viele Variablen im Data-Frame auftauchen).
Weitere Funktionen, die hilfreich sein können zur Untersuchung wachsender Modell-Folgen sind `add`, `drop1`, `step` und `stepwise`.

7 Programmierung in S-Plus

S-Plus liefert neben einer interaktiven Sprache auch die Möglichkeiten, eigene, neue Funktionen in S-Plus zu integrieren. Damit ist S-Plus eine vollständige Programmiersprache mit Kontrollstrukturen, Rekursion und verschiedenen Daten-Typen.

7.1 Kontroll-Strukturen

Über Kontroll-Strukturen werden Entscheidungen getroffen oder aber Schleifen ausgeführt. In der Regel wird man Kontroll-Strukturen in Programmen finden und nicht in der interaktiven Benutzung von S-Plus.

7.1.1 Bedingungen

Bedingungen können mit der `if`-Funktion durchgeführt werden oder mit `switch`.
Der `if`-Ausdruck ist von der Form `if`

```
if (bedingung) ausdruck1 else ausdruck2
```

Innerhalb der `bedingung` findet ein logischer Ausdruck seine Entwicklung zu `T` oder `F`. Falls das Resultat `T` ist, so wird `ausdruck1` ausgeführt, im Falle von `F` `ausdruck2`. Besteht ein `if`-Ausdruck aus mehreren geschachtelten `if`-Ausdrücken, so werden diese in geschweifte Klammern gesetzt.
Soll beispielsweise die Wurzel aus einer Zahl oder den Elementen eines Vektors `x` berechnet werden, so muß `x` zunächst zwei Anforderungen erfüllen:

- `x` muß numerisch sein
- und `x` muß positiv sein (bzw. das kleinste Element größer 0, falls `x` ein Vektor ist)

Um beide Bedingungen zu überprüfen, läßt sich die folgende `if`-Anweisung anwenden:

```
if (is.numeric(x) && min(x)>0) sx<-sqrt(x)
 else stop("x muss numerisch sein und alle Komponenten positiv")
```

Hier werden in runden Klammern die Bedingungen durch ein logisches 'und' verknüpft; dabei überprüft der erste Ausdruck die erste Bedingung, und der zweite Ausdruck prüft, ob `x` ausschließlich positive Elemente besitzt. Falls die Bedingung sich zu `True` entwickelt, so wird der Variablen `sx` die Wurzel aus `x` zugewiesen, ansonsten wird eine Fehlermeldung (zur `stop`-Funktion siehe nächsten Abschnitt) ausgewiesen.

Eine weitere Möglichkeit liefert die `ifelse`-Funktion, das Vektor-Gegenstück zum `if`-Ausdruck: `ifelse`

```
ifelse(test, true.value, false.value)
```

Hier wird nun das `test`-Argument entwickelt, und notfalls in einen logischen Wert überführt. Gleichzeitig werden beide Ausdrücke entwickelt. Wurde `test` zu `T` entwickelt, so ist die `true.value`-Komponente das Resultat, ansonsten `false.value`. Die `ifelse`-Funktion ist relativ schnell und sollte so oft wie möglich verwendet werden.
Die Verwendung von verzweigten `if`-Ausdrücken läßt sich elegant umgehen durch die `switch`-Funktion: `switch`

```
switch(test, name1=ausdruck1, name2=ausdruck2, name3=ausdruck3,...)
```

Es gibt nun zwei Möglichkeiten: das `test`-Argument kann sich zu einem Character-String entwickeln oder zu einem numerischen Wert. Wird `test` zu einem Character-String entwickelt, so werden die nachfolgenden Ausdrücke daraufhin überprüft,

ob einer der Namen dem Namen des Strings entspricht, was dann zur Entwicklung des korrespondierenden Ausdrucks führt. Liefert die Entwicklung eine (positive ganze) Zahl, so werden die Namen ignoriert, und der entsprechend numerierte Ausdruck ausgeführt.
Bezeichnet `x` beispielsweise einen Datenvektor, für den eine der folgenden Kenngrößen berechnet werden soll:

- arithmetisches Mittel (mit der Funktion `mean`)
- Median (mit `median`)
- Varianz (mit `var`)
- oder defaultmäßig die fünf-Zahlen-Zusammenfassung (mit `quantile`)

und soll die Berechnung der jeweiligen Größe über eines der Schlüsselworte 'mittelwert', 'median' oder 'varianz' geschehen, so läßt sich hier zunächst eine Schachtelung von 'if'-Anweisungen verwenden:

```
result<-if (test=="mittelwert") mean(x)
       else if (test=="median") median(x)
            else if (test=="varianz") var(x)
                 else quantile(x)
```

Einfacher läßt sich dies über die `switch`-Funktion gestalten:

```
result<-switch(test,
               mittelwert=mean(x),
               median=median(x),
               varianz=var(x),
               quantile(x)
              )
```

7.1.2 Schleifen

In S-Plus gibt es drei Schleifen-Konstrukte: `for`, `while` und `repeat`.
Eine `for`-Schleife erlaubt die iterative Durchführung eines Ausdrucks, wenn sich der Wert einer Variablen innerhalb einer vorgegebenen Folge bewegt: `for`

```
for (variable in sequence) ausdruck
```

Dabei steht `in` für ein Schlüsselwort, `variable` bezeichnet die Schleifenvariable und `sequence` beinhaltet dann den Vektor der Werte, innerhalb derer sich die Schleife iteriert.
Die `while`- und `repeat`-Schleifen kommen ohne eine Schleifenvariable aus : `while` `repeat`

```
while (bedingung) ausdruck
repeat ausdruck
```

In beiden Fällen werden die im `ausdruck` stehenden Kommandos wiederholt. Dabei wird die `while`-Schleife abgebrochen, sobald sich die `bedingung` zu False entwickelt. Die `repeat`-Schleife kann lediglich durch ein innerhalb des `ausdruck` stehendes `break`-Kommando abgebrochen werden.
Mit dem `next`-Ausdruck lassen sich weiterhin innerhalb von den Schleifenkonstrukten Sprünge in die nächste Iteration bewirken.

7.2 Erzeugung eigener Funktionen

Funktionen werden in S-Plus durch Zuweisung über das Schlüsselwort `function` erzeugt: `function`

```
fkt.name<-function(arg1,arg2,...) ausdruck
```

Dabei bezeichnen `arg1, arg2,...` gerade die Argumente (notwendige und optionale) der Funktion und `ausdruck` steht hier für den Funktions-Koerper, ein beliebiger S-Plus-Ausdruck (in der Regel ein gruppierter Ausdruck in geschweiften Klammern).
Der Aufruf erfolgt über die Angabe des Funktionsnamens und der Argumente:

```
fkt.name(val1,val2,...)
```

Normalerweise wird das Resultat eines Funktionsaufrufs der Wert des Ausdrucks sein, der am Ende des (gruppierten) Funktions-Körpers (in der Regel ohne Zuweisung) steht. Explizit kann eine Funktion jederzeit durch die Funktion `return` beendet werden. Die `return`-Argumente liefern dann das Resultat des Funktionsaufrufs.
Eine eigene Funktion zur Berechnung des arithmetischen Mittels (S-Plus liefert hier ja auch die Funktion `mean`) läßt sich wie folgt realisieren:

```
mw<-function(daten)
    { zaehler<-sum(daten)
      nenner<-length(daten)

      zaehler/nenner
    }

mw(1:10)
[1] 5.5
```


Die Funktion erhält den Namen `mw` und als Funktionsargument `daten`. Im Körper der Funktion werden zunächst mit der Summationsfunktion `sum` sämtliche Elemente des Datenvektors aufsummiert ($\rightarrow \sum x_i$), mit `length` wird die Anzahl Elemente ($\rightarrow n$) überprüft. Die Division beider Komponenten liefert dann das gewünschte Ergebnis.

7.2.1 Warnungen und Stops

Die Funktionen `warning` und `stop` helfen bei der Handhabung von ungewöhnlichen Situationen. Dabei liefert `warning` eine Warnung, falls falsche Argumente übergeben wurden, etc. Mit `stop` läßt sich eine Funktion direkt beenden.
Zusätzlich existiert die Funktion `missing`, mit der das eventuelle nicht-Vorhandensein eines Funktionsargumentes überprüft wird.

7.3 Frames - Objektsuche

Ein 'Evaluation Frame' ist eine Liste, welche Namen mit Werten assoziiert. Sämtliche Werte eines Ausdrucks werden zunächst im aktuellen Frame gesucht, dem 'lokalen' Frame. Ansonsten wird in den nächst höher gelegenen Frames nachgeschaut. Ein 'Frame' ist hier gerade als ein Objektspeicher anzusehen, der bei Bedarf eingerichtet und wieder freigegeben wird, wenn er nicht mehr benötigt wird. So sind alle Umgebungsparameter im 'frame0' untergebracht. Werden nun Funktionsaufrufe gestartet, so werden die benötigten Datenobjekte (Kopien derselben) im 'frame1' abgelegt, für die Funktionsargumente wird ein zweiter Frame, 'frame2', geöffnet. Beinhaltet ein Operand weitere Funktionsaufrufe, so werden zusätzliche Frames geöffnet. Abgebaut werden die Frames durch die Durchführung der Operation und Zurücksendung des Ergebnisses.

Damit gilt nun folgendes:

1. Werden innerhalb einer Funktion Objekte definiert, die bereits existieren, so ist dies kein Problem, da die Zuweisung im entsprechenden Frame, der unterhalb des Frames des Arbeitsverzeichnisses liegt, stattfindet.
2. Aus einer Funktion heraus kann man auf Objekte, Daten des Arbeitsverzeichnisses zugreifen.
3. Beim Absturz einer Funktion werden die entsprechenden Frames aufgelöst, d.h. innerhalb der Funktion definierte Objekte verschwinden wieder, sie werden gelöscht.

8 Aufgaben zur Übung

8.1 Die ersten Schritte, Vektoren/Matrizen/Arrays/Listen

Die ersten Schritte

1. Logge Dich ein und starte die Oberfläche
2. Starte S-Plus , öffne das Help-Fenster und den Graphik-Device
3. Gehe die Beispielsitzung (siehe Seite 7 ff) durch (benutze dabei fleissig die Hilfe, um die einzelnen Funktionen und Befehle schon mal anzuschauen)

Vektoren/Matrizen/Arrays

A

1. Erzeuge einen Vektor mit den nachfolgenden Einträgen und weise dem Vektor einen Namen zu (Quelle: Daten zur Umwelt, 1988/89, S.68, Umweltbundesamt; angegeben sind die mittleren loklalen Geschwindigkeiten von PKWs auf Bundesautobahnen):

1978	1979	1980	1981	1982	1983	1984	1985	1986
122.6	123.2	121.7	121.5	123.7	124.2	125.3	124.2	126.8
123.3	121.6	123.6	122.1	123.1	124.0	124.4	125.2	127.3

2. Berechne arithmetisches Mittel und Median des Datenvektors (→ `mean, median`)
3. Übergebe den Elementen des Vektors die ersten 18 Buchstaben des Alphabets als Namen (→ `letters, names`)

B

1. Erzeuge einen Vektor der Länge 36 mit den Zahlen 1, ... ,36
2. Modifiziere den Vektor zu einer 4×9 Matrix, (je einmal zeilen- und spaltenweise einlesen)

3. Extrahiere diverse Teilmatrizen und Elemente
4. Modifiziere den Vektor zu einem 3×2×6 Array und extrahiere verschiedene Teilarrays

C

Betrachte den S-Plus internen Datensatz `iris`.

1. Was sagen die Attribute aus (→ `attributes`)?
2. Berechne für den Datensatz das arithmetische Mittel, das 20-Prozent getrimmte Mittel und den Median, bezüglich der drei Spezies.

Listen/Data-Frames

1. Erzeuge eine Liste mit folgenden Komponenenten:

Fam.name	Eltern.name	berufstätig (Vater/Mutter)	Anzahl.kinder
Meier	Paul Anna	T T	1
Mueller	Markus Klara	F T	0
Schmidt	Rainer Maria	T F	3

2. Übe den Zugriff auf einzelne Komponenten und Elemente
3. Erzeuge für den Datensatz einen Data-Frame

Folgen

Erzeuge folgende Zahlenfolgen

1. -3,-2, ... ,16,17,18
2. 44 äquidistante Punkte im Intervall $[-3, 18]$
3. 44 äquidistante Punkte mit Abstand 0.5 beginnend bei -3

Was bewirken die folgenden Befehle:

```
x<-1:4
i<-rep(2,4)
y<-rep(x,2)
z<-rep(x,i)
w<-rep(x,x)
```

Zur weiteren Übung Betrachte die Hilfe für die folgenden Funktionen:

1. `round, ceiling, floor`
2. `sort, order`
3. `t, solve, diag`

Probiere die Operationen aus; dabei erzeuge je nach Notwendigkeit einen Vektor oder eine kleine Matrix.

8.2 Einlesen von Daten, Input/Output, klassische Statistik

Aufgabe 1
Betrachte den Datensatz 'unternehmen.data' im Datenverzeichnis '/home/stat/datasets/skurs/data'. Aufgeführt sind die umsatzstärksten (in ECU) europäischen Unternehmen mit den folgenden Variablen: `nation, umsatz, angest, firma`

a) Lese den Datensatz ein; benutze dabei sowohl die Funktion `scan` als auch `read.table`

b) Erstelle eine Häufigkeitstabelle für die Variable Umsatz, sowie die durschnittlichen Umsätze pro Nation

c) Erstelle für die Variable 'Umsatz' landesspezifische Boxplots und interpretiere diese.

d) Betrachte die Umsätze von Italien, Großbritannien und Niederlande etwas genauer, berechne Lageparameter ($\bar{x}$, x_{med}) und interpretiere anhand von entsprechenden Graphiken (was fällt auf? Ursachen? etc.)

Aufgabe 2
Erstelle Objekte unterschiedlichen Typs (Listen, Vektoren,...) und probiere die folgenden Funktionen aus: `write, dump, data.dump`.

Aufgabe 3
Betrachte den S-Plus-internen Datensatz `telsam.response`. Erstelle ein Säulendiagramm für die ersten fünf Interviewer. Versuche dabei, die Plot-Darstellung durch geeignete Einstellung der Argumente `ylim, angle, density, legend, names` zu optimieren.

Aufgabe 4
Betrachte im '/home/stat/datasets/skurs/data' Verzeichnis den Datensatz 'buffalo.data'; angegeben sind hier die Mengen an gefallenem Winter-Schnee (in Inches) in Buffalo, NY, von 1910/11 bis 1972/73.

a) Berechne die 5-Zahlen-Quantils-Zusammenfassung des Datensatzes

b) Erstelle verschiedene Histogramme für den Datensatz; variiere dabei insbesondere mit den Gruppenbreiten und der Anzahl der Gruppen

c) Erstelle verschiedene Boxplots für den Datensatz; variiere auch hier mit den verschiedenen Stilparametern

Aufgabe 5
Konstruiere eine Zufallsstichprobe von 1000 Realisationen einer t-verteilten Zufallsvariablen mit 2 Freiheitsgraden.

a) Erstelle ein Histogramm für die Daten; variiere dabei mit den Klassenbreiten und der Klassenanzahl

b) Führe eine Dichteschätzung durch (`density`) und stelle das Ergebnis graphisch dar. Spiele dabei mit dem `width`-Argument

c) Erstelle einen QQ-Plot gegen die Standardnormalverteilung. Ergänze den Plot durch die Anpassungsgerade und eine Titelüberschrift

Aufgabe 6
Gegeben seien die folgenden Datenreihen aus einem Experiment:

Personnen-Nummer	Serie A	Serie B
1	13.2	14.0
2	8.2	8.8
3	10.9	11.2
4	14.3	14.2
5	10.7	11.8
6	6.6	6.4
7	9.5	9.8
8	10.8	11.3
9	8.8	9.3
10	13.3	13.6

a) Führe einen Ein-Stichprobentest durch für die Hypothese $H_o: \ \mu = 10$ für die Reihe A

b) Führe einen Zwei-Stichprobentest durch für die verbundenen Stichproben

8.3 Regression, Bedingungen/Schleifen, eigene Funktionen

Aufgabe 1

Betrachte den Datensatz 'geburt.data' im Verzeichnis '/home/stat/datasets/skurs/data'. Die Variablen sind im Vorspann der Datei erklärt.

a) Lese den Datensatz ein und stelle ihn als Data-Frame dar.

b) Führe eine lineare Einfach-Regression durch; dabei sei das Geburtsgewicht der Regressor und die Größe der Regressand. Interpretiere den Regressionsoutput.

c) Passe nun separate Einfach-Regressionen an; klassifiziere dabei gemäß der 'Reife'-Variablen. Stelle die jeweiligen Outputs geeignet graphisch dar.

d) Stelle in einer Graphik die unter b) und c) erhaltenen Anpassungsgeraden dar, sowie die Ursprungswerte in verschiedenen Symbolen. Achsenbeschriftung und Überschrift sollten selbstverständlich sein.

e) Führe die unter b), c) und d) geforderten Anpassungen erneut durch; verzichte dabei auf das Absolutglied. Welche Anpassungen sind wohl angebrachter?

Aufgabe 2

Erstelle zwei kleine Funktionen für die folgenden Aufgaben:

a) Fakultäts-Berechnung

b) Sortieren in natürlicher Reihenfolge

Aufgabe 3

Schreibe eine Funktion, welche die folgende Problemstellung löst: betrachtet wird ein Datenvektor $x = (x_1, \ldots, x_n)$. Dieser Datenvektor soll geglättet werden, d.h. man ist bestrebt, den allgemeinen Verlauf darzustellen. Diese Glättung soll dem nachstehenden Prinzip folgen:

1. zur Glättung am Punkte x_k werden m (ungerade) Beobachtungspunkte benötigt: $x_{k-[m/2]}, \ldots, x_{k+[m/2]}$ ($[z]$ bedeutet die größte ganze Zahl kleiner oder gleich z)
2. die betrachteten Punkte werden nun mit einer Funktion bearbeitet, das Ergebnis liefert den geglätteten Wert an dieser Stelle x_k. Als Funktionen kommen in betracht: das arithmetische Mittel, das (beliebig) getrimmte Mittel, sowie der Median

Die Funktion soll nun folgendes liefern:

Input: den Datenvektor, die Anzahl m heranzuziehender Punkte, die Berechnungsvorschrift, eine Plot-Option

Output: je nach Einstellung der Plot-Option erhält man entweder

a) eine Liste mit den folgenden Komponenten: Input-Variablen, gelättete Werte;

b) eine Plot-Darstellung des Datenvektors mit eingezeichneter Glättung in geeigneter Darstellung und mit Legende. Dabei soll in der Über -oder Unterschrift auch der Wert m sowie der Name der Berechnungsfunktion auftauchen (Hinweis: Funktionen `paste, eval` helfen dabei)

Als Testdatensätze sind hier der S-Plus-interne Datensatz `hstart` sowie der Datensatz 'arbeit.data' im 'home/stat/datasets/skurs/daten'-Verzeichnis heranzuziehen; letzterer liefert die monatlichen Zahlen der Arbeitslosen der BRD vom Januar 1950 bis zum August 1990.