Statistical Computing

Einführung in 

Günther Sawitzki
StatLab Heidelberg

8. Dezember 2005

noch in Vorbereitung

E-mail address: gs@statlab.uni-heidelberg.de
URL: http://www.statlab.uni-heidelberg.de/projects/r/

Key words and phrases. statistical computing, S programming language, R programming, data analysis, exploratory statistics, residual diagnostics

8. Dezember 2005.

Inhaltsverzeichnis

Einleitung

Diese Einführung in R ist als Arbeitsmaterial in einem Kompaktkurs oder zum Selbststudium gedacht. Der Kurs wendet sich an Studierende mit Grundkenntnissen in Stochastik. Begriffe wie Verteilungsfunktion, Quantil, Erwartungswert und Varianz und die damit verbundenen einfachen Eigenschaften werden vorausgesetzt. Ebenso sollten klassische Verteilungen wie Binomial-, Uniform- und Gaußverteilung sowie daraus abgeleitete Verteilungen und deren asymptotisches Verhalten bekannt sein. Kenntnisse in Statistik selbst werden nicht vorausgesetzt. Sie werden in diesem Kurs aber auch nicht vermittelt. Der Kurs konzentriert sich auf die "Computing"-Aspekte. Dabei werden statistische Betrachtungsweisen und Konzepte zwar eingeführt und diskutiert. Für eine eingehendere Diskussion wird aber auf die Statistik-Vorlesungen verwiesen.

Kenntnisse in der Rechnerbenutzung und zumindest oberflächliche Kenntnisse von Programmierkonzepten wie Variable, Schleifen und Funktionen werden vorausgesetzt. Weitergehende Kenntnisse werden nicht vorausgesetzt, aber auch nicht vermittelt. Der Kurs führt in die Benutzung von R als Anwender ein. Für eingehendere Diskussion der Computing-Aspekte wird auf die Arbeitsgemeinschaft "Computational Statistics" verwiesen.

<http://www.statlab.uni-heidelberg.de/studinfo/compstat/>

0.1. Was ist R?

R ist eine Programmiersprache, und auch der Name eines Software-Systems, das diese Sprache implementiert. Die Programmiersprache R ist eine für die Statistik und für stochastische Simulation entwickelte Programmiersprache, die mittlerweile zum Standard geworden ist. Genau genommen müsste man hier unterscheiden: Die Sprache heißt S, ihre Implementierung und das System heißen R. Die ursprünglichen Autoren von S sind John M. Chambers, R. A. Becker und A. R. Wilks, AT & T Bell Laboratories, Statistics Research Department. Die Sprache und ihre Entwicklung sind in einer Reihe von Büchern dokumentiert, nach ihrem Umschlag häufig als das weisse ([**CH92**]), blaue ([**BCW88**]) und grüne Buch ([**Cha98**]) bezeichnet.

Die AT & T-Implementierung von S war lange Zeit die "Referenz" für die Sprache S. Heute gibt es S als kommerzielles System S-Plus <http://www.insightful.com/> (basierend auf der AT & T-Implementierung) sowie als frei verfügbare Version R, auch "Gnu S" genannt[1] <http://www.r-project.org/>.

[1] R heißt nur zufällig so, wie auch zufälligerweise die Vornamen der Originalautoren (Ross Ihaka & Robert Gentleman) mit R beginnen.

Mittlerweile hat sich R zur Referenz-Implementierung entwickelt. Wesentliche Präzisierungen, und - falls notwendig - auch Modifikationen der Sprache werden durch R definiert. Der Einfachheit halber sprechen wir hier und in den folgenden Kapiteln von der Sprache R, auch wenn es genauer heißen müsste: die Sprache S in der R-Implementierung.

R ist eine interpretierte Programmiersprache. Anweisungen in R werden unmittelbar ausgeführt. R beinhaltet neben den ursprünglichen Elementen von S eine Reihe von Erweiterungen, zum Teil um Entwicklungen in der Statistik angemessen zu berücksichtigen, zum Teil um experimentelle Möglichkeiten zu eröffnen. Parallel dazu gibt es Weiterentwicklungen der S-Sprache.

Die (2005) aktuelle Version von R ist R 2.x. Diese Version ist weitgehend kompatibel mit den Vorläuferversionen R 1.x. Die wesentlichen Veränderungen sind im Inneren des Systems. Für den Anfang gibt es praktisch keinen Unterschied zu R 1.x. Für den fortgeschrittenen Nutzer gibt es drei wesentliche Neuerungen:

- *Grafik:* Das Basis-Grafiksystem von R implementiert ein Modell, dass an der Vorstellung von Stift und Papier orientiert ist. Ein Grafik-Port (Papier) wird eröffnet und darauf werden Linien, Punkte/Symbole gezeichnet. Mit R 2.x gibt es zusätzlich ein zweites Grafiksystem, dass an einem Kamera/Objekt-Modell orientiert ist. Grafische Objekte in unterschiedlicher Lage und Richtung werden in einem visuellen Raum abgebildet.
- *Packages:* Das ursrpüngliche System von R hat eine lineare Geschichte und einen einheitlichen Arbeitsraum. Mit R 2.x gibt es eine verbesserte Unterstützung von "Paketen", die in sich abgeschirmt werden können. Dazu dienen Sprachkonzepte wie "name spaces", aber auch unterstützende Werkzeuge.
- *Internationalisierung:* Die ursprüngliche Implementierung von R setzte Englisch als Sprache und ASCII als Zeichensatz voraus. Seit R 2.x gibt es umfassende Unterstützung für andere Sprachen und Zeichensätze. Dies ermöglicht es, "lokalisierte" Versionen zu erstellen. Derzeit ist man bei Kommandos, Ausgaben und Erklärungen jedoch noch auf Englisch angewiesen.

Zwei Aspekte sind in R nur unzureichend berücksichtigt: der interaktive Zugriff und die Einbettung in eine vernetzte Umgebung. Diese und weitere Aspekte sind Bestandteil von Omegahat - eines Versuchs, ein System der nächsten Generation zu entwickeln, das auf den Erfahrungen mit R aufbaut. Diese mehr experimentellen Arbeiten werden unter `<http://www.omegahat.org/>` bereitgestellt. Schon R bietet einfache Möglichkeiten, Prozeduren aus anderen Sprachen wie C und Fortran aufzurufen. Omegahat erweitert diese Möglichkeiten und bietet einen direkten Zugang zu Java, Perl .... Eine Java-basierte grafische Oberfläche ist als JGR unter `<http://stats.math.uni-augsburg.de/software/>` zugänglich. Dort findet sich als iplots auch eine Sammlung von interaktiven Displays für R.

Zahlreiche hilfreiche Erweiterungen sind auch unter `<http://www.bioconductor.org/>` zu finden.

0.2. Referenzen

R ist für die praktische Arbeit in der Statistik entworfen. Nützlichkeit hat oft Vorrang vor prinzipiellen Design-Überlegungen. Als Folge ist eine systematische Einführung in R nicht einfach. Stattdessen wird ein verschlungener Pfad gewählt: Fallstudien und Beispiele, an die sich systematische Übersichten anschliessen. Für die praktische Arbeit sollte auf das reichhaltige Online-Material zu R zugegriffen werden. Ein erster Zugriffspunkt sind dabei die "frequently asked questions" (FAQ) <http://www.cran.r-project.org/faqs.html>. "An Introduction to R" ([**R D04a**]) ist die "offizielle" Einführung. Diese Dokumentation und andere Manuale sind unter <http://www.cran.r-project.org/manuals.html> bereitgestellt.

R-Prozeduren sind zum Teil im Basis-System enthalten. Andere Prozeduren müssen aus Bibliotheken hinzugeladen werden. Für den Kurs wird ein vorkonfiguriertes R-System benutzt, das den Teilnehmern zu Beginn des Kurses zur Verfügung gestellt wird. Wird die Standard-Distribution von R benutzt, so müssen entsprechende Bibliotheken hinzu geladen werden.

Größere Unterschiede gibt es bei unterschiedlichen Versionen von S-Plus. S-Plus 4.x und S-Plus 2000 benutzen S Version 3 und sind weitestgehend mit R kompatibel. S-Plus 5 ist eine Implementierung von S Version 4 mit Änderungen, die eine Sonderbehandlung bei der Programmierung benötigen. Auf diese Besonderheiten wird hier nicht eingegangen. Informationen zu S-Plus findet man bei <http://www.insightful.com/>.

0.3. Umfang und Aufbau des Kurses

R beinhaltet in der Basis-Version mehr als 1000 Funktionen - zu viele, um sie in einem Kurs zu vermitteln, und zu viel, um sie sinnvollerweise zu lernen. Der Kurs kann nur dazu dienen, den Zugang zu R zu eröffnen.

Teilnehmerkreise können aus unterschiedlichem Hintergrund kommen und unterschiedliche Vorbedingungen mitbringen. Gerade für jüngere Schüler oder Studenten kann ein reiner Programmierkurs, der sich auf die technischen Grundlagen konzentriert, angemessen sein. Für diese Teilnehmer ist dieser Kurs nicht geeignet. In späteren Abschnitten stellt sich eher die Frage nach einer sinnvollen Einordnung und nach dem Hintergrund. Hierauf zielt der vorliegende Kurs. Das "technische" Material bildet das Skelett. Daneben wird versucht, den Blick auf statistische Fragestellungen zu richten und das Interesse am Hintergrund zu wecken. Der Kurs soll Appetit auf die Substanz wecken, die eine fundierte statistische Vorlesung bieten kann.

Das hier bereitgestellte Material besteht zunächst aus einer thematisch geordneten Sammlung, in der anhand von Beispiel-Fragestellungen illustriert wird, wie ein erster Zugang mit R erfolgen kann. Hinzu kommt eine Zusammenstellung von Sprachbestandteilen und Funktionen, die als Orientierungshilfe für das umfangreiche in R enthaltene Informationsmaterial dient. Für die praktische Arbeit sind die Online-Hilfen und Manuale die erste Informationsquelle.

Der Kurs kann bei einer Auswahl der Aufgaben in etwa vier Tagen durchgeführt werden. Konzeptuell ist er eine viertägige Einführung in die Statistik mit den Themenbereichen

- Ein-Stichprobenanalyse und Verteilungen
- Regresssion
- Zwei- oder Mehr-Ein-Stichprobenanalysen
- Multivariate Analysen

Eine großzügigere Zeit für die Übungsaufgaben wird empfohlen (ein Halbtag zusätzlich für einführende Aufgaben, ein Halbtag zusätzlich für eine der Projektaufgaben). Mit dieser Zeit kann der Kurs als Block in einer Woche durchgeführt werden, wenn im Anschluss die Möglichkeit geschaffen wird, die aufgetreten Fragen zu beantworten und das geweckte Interesse am statistischen Hintergrund zu vertiefen.

Für ein anschliessendes vertiefendes Selbststudium von R als Programmiersprache wird ([**VR00**]) empfohlen.

Beispiele und Eingaben im Text sind so formatiert, dass sie mit "Cut & Paste" übernommen und als Programmeingabe verwandt werden können. Deshalb sind bei Programmbeispielen im Text bisweilen Satzzeichen fortgelassen, und Eingabebeispiele werden ohne "Prompt" gezeigt. Einem Beispiel

Beispiel 0.1:

Eingabe

```
1 + 2
```

Ausgabe

```
3
```

enstpricht auf dem Bildschirm etwa

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

wobei anstelle des Prompt-Zeichens ">" je nach Konfiguration auch ein anderes Zeichen erscheinen kann.

0.4. Dank

Zu danken ist dem R core team für die Kommentare und Hinweise. Besonderen Dank an Friedrich Leisch vom R core team sowie an Antony Unwin, Univ. Augsburg.

0.5. Literatur und weitere Hinweise:

[**R D04a**] R Development Core Team (2000-2005): An Introduction to R.
Siehe: <http://www.r-project.org/manuals.html>.

[**R D04d**] R Development Core Team (2000-2005): R Reference Manual.
Siehe: <http://www.r-project.org/manuals.html>.

The Omega Development Group (2000): Omega.
Siehe: <http://www.omegahat.org/>.

[**BCW88**] Becker, R.A.; Chambers, J.M.; Wilks, A.R. (1988): The New S Language. NewYork: Chapman and Hall.

[**CH92**] Chambers, J.M.; Hastie, T.J. (eds) (1992): Statistical Models in S. NewYork: Chapman and Hall.

[**Cle93**] Cleveland, W.F. (1993): Visualizing Data. Summit: Hobart Press.

[**Gen95**] Gentleman, R. (1995): Statistical Computing Using R. Auckland: The University. Papers 528.188/T187.

[**Lee95**] Lee, A. (1995): Data Analysis - An Introduction Based on R. Auckland: The University. Papers 528.281/288.

[**VR02**] Venables, W.N.; Ripley, B.D. (2002): Modern Applied Statistics with S. Heidelberg:Springer.
Siehe: <http://www.stats.ox.ac.uk/pub/MASS4/>.

[**VR00**] Venables, W.N.; Ripley, B.D. (2000): Programming in S. Heidelberg:Springer.
Siehe: <http://www.stats.ox.ac.uk/pub/MASS3/Sprog>.

KAPITEL 1

Grundlagen

1.1. Programmierung: Konventionen

Wie jede Programmiersprache hat R bestimmte Konventionen. Hier die ersten Grundregeln.

R-*Konventionen*	
Objekte	Die Datenbausteine in R sind Objekte. Objekte können Klassen zugeordnet werden. *Beispiel:* Die Basis-Objekte in R sind Vektoren.

Namen	Auf R-Objekte wird anhand ihres Namens zugegriffen.
	Beispiele: `x`
	`y`
	Groß- und Kleinschreibung werden unterschieden.
	Beispiele: `Y87`
	`y87`

Zuweisungen	Zuweisungen haben die Form *Aufruf:* `Name <-Wert` oder alternativ `Name = Wert`. *Beispiel:* `x <- 10`
Funktionen	Funktionen in R werden aufgerufen in der Form *Aufruf:* `Name(Parameter ...`) *Beispiel:* `x <- exp(10)`. Diese Konvention gilt selbst, wenn keine Parameter vorhanden sind. *Beispiel:* Um R zu verlassen ruft man eine "Quit"-Prozedur auf `q()`. Parameter werden sehr flexibel gehandhabt. Sie können Default-Werte haben, die benutzt werden, wenn kein expliziter Parameter angegegeben ist. *Beispiele:* folgen später.

(Fortsetzung)→

R-*Konventionen* (Fortsetzung)	

	Funktionen können ***polymorph*** sein. Die aktuelle Funktion wird dann durch die Klasse der aktuellen Parameter bestimmt.
	Beispiele: `plot(x)`
	`plot(x,x^2)`
	`summary(x)`

Abfragen	Wird nur der Name eines Objekts eingegeben, so wird der Wert des Objekt ausgegeben. *Beispiel:* `x`

Hilfe	Dokumentation und Zusatzinformation für ein Objekt kann mit `help` angefordert werden.
	Aufruf: `help(Name)`
	Beispiele: `help(x)`
	`help(exp)`
	Alternative Form `?Name`
	Beispiele: `?x`
	`?exp`

Operatoren	Für Vektoren wirken Operatoren auf jede Komponente der Vektoren. *Beispiel:* Für Vektoren `y, z` ist `y*z` ein Vektor, der komponentenweise das Produkt enthält. Operatoren sind spezielle Funktionen. Sie können auch in Präfix-Form aufgerufen werden. *Beispiel:* `"+"(x,y)`
Indizes	Auf Vektorkomponenten wird über Indizes zugegriffen. *Beispiel:* `x[3]` Dabei können auch symbolische Namen verwandt werden. *Beispiele:* folgen später.

Wir beschäftigen uns im folgenden mit statistischen Methoden. Wir benutzen die Methoden zunächst in Simulationen, d.h. mit synthetischen Daten, deren Erzeugung wir weitgehend unter Kontrolle haben. Das erlaubt es uns, Erfahrung mit den Methoden zu gewinnen und sie kritisch zu beurteilen. Erst dann benutzen wir die Methoden zur Analyse von Daten.

1.2. Erzeugung von Zufallszahlen und Mustern

1.2.1. Zufallszahlen. Die Funktion ***runif*** erlaubt die Erzeugung von uniform verteilten Zufallsvariablen. Mit ***help(runif)*** oder ***?runif*** erhalten wir Informationen, wie die Funktion benutzt werden kann:

---

help(runif)

`Uniform` *The Uniform Distribution*

Description.

These functions provide information about the uniform distribution on the interval from `min` to `max`. `dunif` gives the density, `punif` gives the distribution function `qunif` gives the quantile function and `runif` generates random deviates.

Usage.

```
dunif(x, min=0, max=1, log = FALSE)
punif(q, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
qunif(p, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
runif(n, min=0, max=1)
```

Arguments.

`x,q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1`, the length is taken to be the number required.
`min,max`	lower and upper limits of the distribution.
`log, log.p`	logical; if TRUE, probabilities p are given as log(p).
`lower.tail`	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

Details.

If `min` or `max` are not specified they assume the default values of `0` and `1` respectively.

The uniform distribution has density

$$f(x) = \frac{1}{max - min}$$

for $min \leq x \leq max$.

For the case of $u := min == max$, the limit case of $X \equiv u$ is assumed.

References.

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

See Also.

`.Random.seed` about random number generation, `rnorm`, etc for other distributions.

Examples.

```
u <- runif(20)

## The following relations always hold :
punif(u) == u
dunif(u) == 1

var(runif(10000))#- ~ = 1/12 = .08333
```

Diese Hilfsinformation sagt uns: Als Parameter für ***runif*** muß die Anzahl ***n*** der zu generierenden Zufallswerte angegeben werden. Als weitere Parameter für ***runif*** können das Minimium und das Maximum des Wertebereichs angeben werden. Geben wir keine weiteren Parameter an, so werden die Default-Werte ***min=0*** und ***max=1*** genommen. Z. B. ***runif(100)*** erzeugt einen Vektor mit 100 uniform verteilten Zufallsvariablen im Bereich (0,1). Der Aufruf ***runif(100, -10, 10)*** erzeugt einen Vektor mit 100 uniform verteilten Zufallsvariablen im Bereich (-10,10). Die zusätzlichen Parameter können in der definierten Reihenfolge angegeben werden, oder mithilfe der Namen spezifiziert werden. Bei Angabe des Namens kann die Reihenfolge frei gewählt werden. Anstelle von ***runif(100, -10, 10)*** kann also ***runif(100, min=-10, max=10)*** oder ***runif(100, max=10, min=-10)*** benutzt werden. Dabei können auch ausgewählt einzelne Parameter gesetzt werden. Wird zum Beispiel das Minimum nicht angegeben, so wird für das Minimum der Default-Wert eingesetzt: die Angabe von ***runif(100, max=10)*** ist gleichwertig mit ***runif(100, min=0, max=10)***. Der besseren Lesbarkeit halber geben wir oft die Namen von Parametern an, auch falls es nicht nötig ist.

Jeder Aufruf von ***runif*** erzeugt neue Zufallszahlen. Wir können diese speichern.

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

erzeugt einen neuen Vektor von Zufallszahlen und weist ihn der Variablen ***x*** zu.

```
x
```

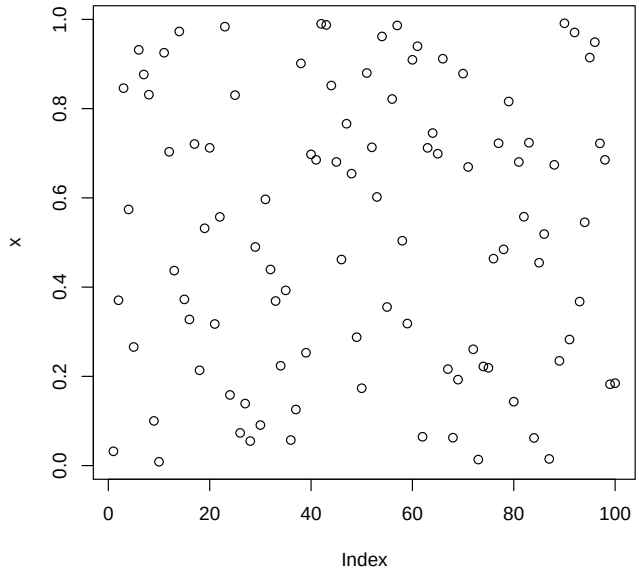
gibt jetzt dessen Werte aus; damit können wir die Resultate inspizieren. Eine grafische Darstellung, den Serienplot - einen Scatterplot der Einträge in ***x*** gegen den laufenden Index, erhalten wir mit

```
plot(x)
```

Beispiel 1.1:

Eingabe

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



Aufgabe 1.1	
	Experimentieren Sie mit den Plots und *runif()*. Sind die Plots Bilder von Zufallszahlen? Genauer: Akzeptieren Sie die Plots als Bilder von 100 unabhängigen Realisationen von uniform auf (0,1) verteilten Zufallszahlen? Wiederholen Sie das Experiment und versuchen Sie, die Argumente, die für oder gegen die (uniforme) Zufälligkeit sprechen, möglichst genau zu notieren. Ihr Resumée? Gehen Sie die Argumente noch einmal durch und versuchen Sie, eine Prüfstrategie zu entwerfen, mit der Sie eine Folge von Zahlen auf (uniforme) Zufälligkeit überprüfen könnten. Versuchen Sie, diese Strategie möglichst genau zu formulieren. (Fortsetzung)→

Aufgabe 1.1	(Fortsetzung)
	Hinweis: Sie können mehrere Abbildungen in einem Fenster halten. Mit `par(mfrow=c(2,2))` wird das Grafik-System so eingestellt, dass jeweils vier Abbildungen zeilenweise als 2×2-Matrix angeordnet gezeigt werden. Die Funktion `par` ist die zentrale Funktion, mit der die Grafik-Ausgabe parametrisiert wird. Weitere Information erhält man mit `help(par)`.

Wir lüften gleich das Geheimnis [1]: die Zahlen sind nicht zufällig, sondern ganz deterministisch. Genauer: im Hintergrund von `runif` wird eine deterministische Folge z_i generiert. Verschiedene Algorithmen stehen zur Verfügung. Informationen dazu erhält man mit `help(.Random.seed)`. Im einfachsten Fall, für lineare Kongruenzgeneratoren, werden aufeinanderfolgende Werte z_i, z_{i+1} sogar nur mit einer linearen Funktion generiert. Damit die Werte im kontrollierten Bereich bleiben, wird modulo einer oberen Grenze gerechnet, also

$$z_{i+1} = a\ z_i + b \mod M.$$

Die resultierenden Werte, die uns übergeben werden, sind umskaliert

$$\frac{z_i}{M} \cdot (max - min) + min.$$

Die Zahlenfolge ist also keine unabhängige Zufallsfolge, und die Verteilung ist auch nicht uniform auf (min, max).

Selbst wenn man das Geheimnis kennt, ist es nur mit viel weiterem Wissen möglich nachzuweisen, dass die erzeugte Folge nicht den Gesetzen folgt, die für eine unabhängige Folge von identisch uniform verteilten Zufallszahlen gelten.

Zahlenfolgen, die den Anspruch erheben, sich wie zufällige Zahlen zu verhalten, nennen wir ***Pseudo-Zufallszahlen***, wenn es wichtig ist, auf den Unterschied hinzuweisen. Wir benutzen diese Pseudo-Zufallszahlen, um uns geeignete Test-Datensätze zu generieren. Wir können damit untersuchen, wie sich statistische Verfahren unter nahezu bekannten Bedingungen verhalten. Dabei benutzen wir Pseudo-Zufallszahlen, als ob wir Zufallszahlen hätten.

Pseudo-Zufallszahlen sollten wir zum anderen als Herausforderung nehmen: Sind wir in der Lage, sie als nicht unabhängige Zufallszahlen zu erkennen? Wenn wir einen Unterschied erkennen, werden wir versuchen, den Pseudo-Zufallszahlengenerator gegen einen besseren auszutauschen. Aber zunächst geht die Herausforderung an uns. Sind wir überhaupt in der Lage, z.B. eine mit einem linearen Generator erzeugte deterministische Folge als nicht zufällig zu erkennen? Falls nicht: welche intellektuellen Konsequenzen ziehen wir daraus?

[1] ... nur teilweise. Die benutzten Zufallsgeneratoren in R sind konfigurierbar und können wesentlich komplexer sein, als hier vorgestellt. Für unsere Diskussion reicht jedoch hier die Familie der linearen Kongruenzgeneratoren. Sie können deren Verhalten in anderen Programmiersystemen nachvollziehen. Die übliche Referenz ist dabei der "minimal standard generator" mit $x_{i+1} = (x_i \times 7^5) \mod 2^{31} - 1$.

1.2.2. Muster. Außer Pseudo-Zufallszahlen gibt es in R eine ganze Reihe von Möglichkeiten, regelmäßige Sequenzen zu generieren. Die in anderen Sprachen notwendigen Schleifen werden damit weitgehend vermieden. Hier eine erste Übersicht:

R ***Sequenzen***	
`c()`	"combine". Kombiniert Argumente zu einem neuen Vektor. *Aufruf:* `c(..., recursive=FALSE)` *Beispiele:* `c(1,2,3)` `c(x,y)`
`:`	Erzeugt ganzzahlige Sequenz von `Anfang` bis `Ende`. *Aufruf:* `Anfang:Ende` *Beispiele:* `1:10` `10:1`
`seq()`	Erzeugt allgemeine Sequenzen. *Aufruf:* Siehe `help(seq)`
`rep()`	Wiederholt Argument. *Aufruf:* `rep(x, times, ...)` *Beispiele:* `rep(x,3)` `rep(1:3,c(2,3,1))`

Dabei steht "..." für eine variable Liste von Argumenten. Wir werden diese Notation noch häufiger benutzen.

Aufgabe 1.2	
	Generieren Sie mit `plot(sin(1:100))` einen Plot mit einer diskretisierten Sinusfunktion. (Falls Sie die Sinusfunktion nicht sofort erkennen, benutzen Sie `plot(sin(1:100), type="l")` um die Punkte zu verbinden. Benutzen Sie Ihre Strategie aus Aufgabe 1.3. Können Sie damit die Sinusfunktion als nicht zufällig erkennen?

Die Zahlenreihe eines Datensatzes, wie z.B. die Ausgabe eines Zufallszahlengenerators hilft selten, zugrunde liegende Strukturen zu erkennen. Nur wenig helfen einfache, unspezifische grafische Darstellungen wie der Serienplot. Selbst bei klaren Mustern sind diese Informationen selten aussagekräftig. Zielgerichtete Darstellungen sind nötig, um Verteilungseigenschaften zu untersuchen.

1.3. Fallstudie: Verteilungsdiagnostik

Wir brauchen genauere Strategien, um Strukturen zu erkennen oder deren Verletzung festzustellen. Wie diese Strategien aussehen können, skizzieren wir am Beispiel der Zufallszahlen. Wir konzentrieren uns hier auf die Verteilungseigenschaft. Angenommen, die Folge besteht aus unabhängigen Zufallszahlen mit einer gemeinsamen Verteilung. Wie überprüfen wir, ob dies die uniforme Verteilung ist? Wir ignorieren die mögliche Umskalierung auf $(\min, \max)$ - dies ist ein technisches Detail, das die Fragestellung nicht wesentlich tangiert. Wir betrachten $\min = 0; \max = 1$.

Aus Realisierungen von Zufallsvariablen können Verteilungen nicht direkt abgelesen werden. Dies ist unser kritisches Problem. Wir brauchen Kennzeichnungen der Verteilungen, die wir empirisch überprüfen können. Wir können zwar Beobachtungen als Maße betrachten: Für n Beobachtungen $X_1, \ldots, x_n$ können wir formal die empirische Verteilung P_n definieren als das Maß $P_n = \sum(1/n)\delta_{x_i}$, wobei δ_{X_i} das Dirac-Maß an der Stelle X_i ist. Also

$$P_n(A) = \#\{i : X_i \in A\}/n.$$

Aber leider ist das empirische Maß P_n einer Beobachtungsreihe von unabhängigen Beobachtungen mit gemeinsamem Maß P im allgemeinen sehr von P verschieden. Einige Eigenschaften gehen unwiederbringlich verloren. Dazu gehören inifinitesimale Eigenschaften: so ist z.B. P_n immer auf endlich viele Punkte konzentriert. Wir brauchen Konstrukte, die anhand von Realisierungen von Zufallsvariablen bestimmbar und mit den entsprechenden Konstrukten von theoretischen Verteilungen vergleichbar sind. Eine Strategie ist es, sich auf (empirisch handhabbare) Testmengen zu beschränken.

Beispiel 1.1. **Verteilungsfunktion**

Anstelle der Verteilung P betrachten wir ihre Verteilungsfunktion $F = F_P$ mit

$$F(x) = P(X \leq x).$$

Für eine empirische Verteilung P_n von n Beobachungen $X_1, \ldots, x_n$ ist entsprechend die empirische Verteilungsfunktion

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

Beispiel 1.2. **Histogramm**

Wir wählen disjunkte Testmengen $A_j, j = 1, \ldots, J$, die den Wertebereich von X überdecken. Für die uniforme Verteilung auf $(0, 1)$ können wir z.B. die Intervalle

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

als Testmengen wählen.

Anstelle der Verteilung P betrachten wir den Vektor $\big(P(A_j)\big)_{j=1,\ldots,J}$ bzw. den empirischen Vektor $\big(P_n(A_j)\big)_{j=1,\ldots,J}$.

Wir diskutieren diese Beispiele ausführlicher. Einige allgemeine Lehren können wir daraus ziehen. Wir machen mehrere Durchgänge, um von einem naiven Zugang zu einem entwickelten statistischen Ansatz zu kommen.

An dieser Stelle sei schon darauf hingewiesen, dass Histogramme kritisch von der Wahl der Testmengen abhängen. Inbesondere wenn Diskretisierungen in den Daten unglücklich mit der Wahl der Testmengen zusammentreffen, kann es zu sehr irreführenden Ergebnissen kommen. Eine Alternative zu Histogrammen ist es, die Daten zu glätten.

BEISPIEL 1.3. **Glättung** Wir ersetzen jeden Datenpunkt durch eine (lokale) Verteilung, d.h. wir verschmieren die Datenpunkte etwas. Wir benutzen dazu Gewichtsfunktionen. Diese Gewichtsfunktionen werden "Kerne" genannt und mit K bezeichnet. Wenn die Kerne integrierbar sind, normieren wir sie konventionell so, dass $\int K(x)dx = 1$. Wenn sie einen kompakten Träger haben, so wählen wir als Träger konventionell das Intervall $[-1, 1]$. Übliche Kerne sind

Kern	$K(x)$
Uniform	$1/2$
Epanechnikov (quadratisch)	$3/4(1-x^2)$
Biweight	$15/16(1-x^2)^2$
Triweight	$35/32(1-x^2)^3$
Gauß	$(2\pi)^{-1/2}\ exp(-x^2/2)$

TABELLE 1.6. Einige übliche Kerne

Durch Verschiebung und Umskalierung definiert jeder Kern eine ganze Familie

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

Der mit h skalierte Kern wird mit K_h bezeichnet

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

Der Skalenfaktor h wird Bandbreite genannt.

Die Funktion

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

ergibt anstelle des Histogramms ein geglättetes Bild.

Näheres dazu findet man unter dem Stichwort ***smoothing*** in der Literatur. Wir gehen hier nicht weiter darauf ein.

1.3.1. Erster Durchgang zu Beispiel 1.1: Verteilungsfunktion. Um zu prüfen, ob eine Zufallsfolge zu einer Verteilung mit Verteilungsfunktion F paßt, vergleiche man F mit F_n. Im Fall der uniformen Verteilung auf $(0, 1)$ ist $F(x) = F_{unif}(x) = x$ für $0 \leq x \leq 1$. Der ganz naive Zugang berechnet die Funktionen F_n und F. Eine erste Überlegung sagt: F_n ist eine stückweise konstante Funktion mit Sprungstellen an den Beobachtungspunkten. Wir bekommen also ein vollständiges Bild von F_n, wenn wir F_n an den Beobachtungspunkten $X_i, i = 1..n$ auswerten. Ist $X_{(i)}$ die i. Ordnungsstatistik, so ist - bis auf Bindungen - $F_n(X_{(i)}) = i/n$. Wir vergleichen $F_n(X_{(i)})$ mit dem "Sollwert" $F(X_{(i)}) = X_{(i)}$. Eine R-Implementierung, mit Hilfsvariablen notiert:

```
n <- 100
x <- runif(n)
xsort <- sort(x)
i <- (1:n)
y <- i/n
plot(xsort, y)
```

Eine zusätzliche Gerade für die “Sollwerte” kann mit

```
abline(0,1)
```

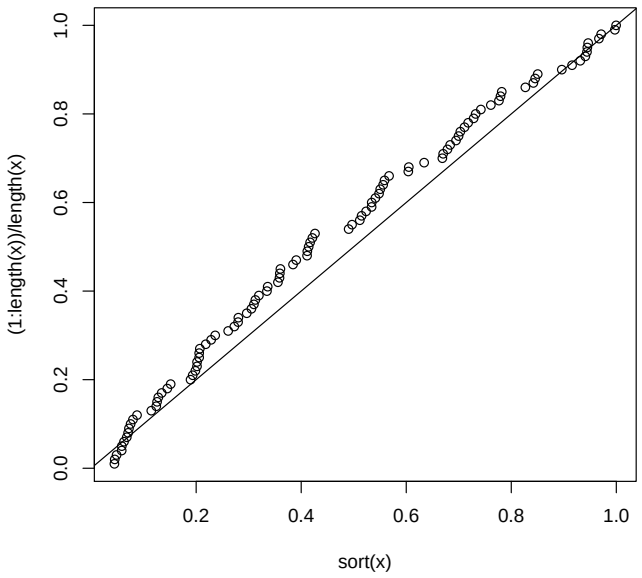
eingezeichnet werden.

Eine kompaktere Implementierung mit der Funktion `length()`:

Beispiel 1.2:

Eingabe

```
x <- runif(100)
plot(sort(x), (1:length(x))/length(x))
abline(0, 1)
```



R ***Funktionen***	
`sort()`	Sortiert Vektor *Beispiel:* `sort(runif(100))`

(Fortsetzung)→

R ***Funktionen*** (Fortsetzung)	
`length()`	Länge eines Vektors *Beispiel:* `length(x)`
`abline()`	Fügt Linie in Plot hinzu *Beispiel:* `abline(a=0, b=2)`

Die Funktion `plot()` fügt defaultmässig Beschriftungen hinzu. Damit die Grafik für sich aussagekräftig ist, wollen wir diese durch genauere Beschriftungen ersetzen. Dazu ersetzen wir die Default-Parameter von `plot()` durch unsere eigenen. Der Parameter `main` kontrolliert die Hauptüberschrift (Default: leer). Wir können diese zum Beispiel ersetzen wie in

```
plot(sort(x),(1:length(x))/length(x),
        main=" Empirische Verteilungsfunktion (X uniform)").
```

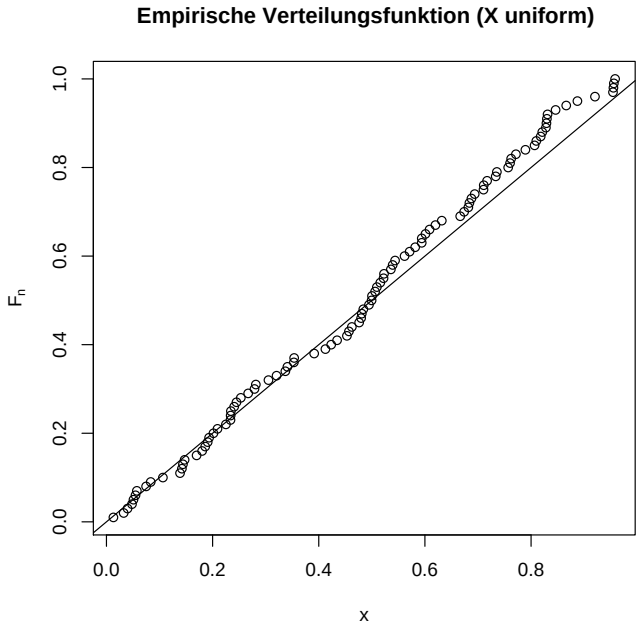
Mit `xlab` und `ylab` wird die Beschriftung der Achsen gesteuert. Über diese und weitere Parameter kann man Information mit `help(plot)` abfragen, werden dann aber weiter an `help(title)` verwiesen.

Die vertikale Achse gibt noch eine Herausforderung: mit `ylab="Fn(x)"` als Parameter würden wir eine Beschriftung mit $Fn(x)$ erhalten. Die übliche Bezeichnung setzt aber den Stichprobenumfang als Index, also $F_n(x)$. Hier hilft eine versteckte Eigenschaft der Beschriftungsfunktionen: Wird als Parameter eine Zeichenkette übergeben, so wird sie ohne Umwandlung angezeigt. Wird als Parameter ein R-Ausdruck übergeben, so wird versucht, die mathematisch übliche Darstellung zu geben. Details findet man mit `help(plotmath)` und Beispiele mit `demo(plotmath)`. Die Umwandlung einer Zeichenkette in einen (unausgewerteten) R-Ausdruck geschieht mit `expression()`.

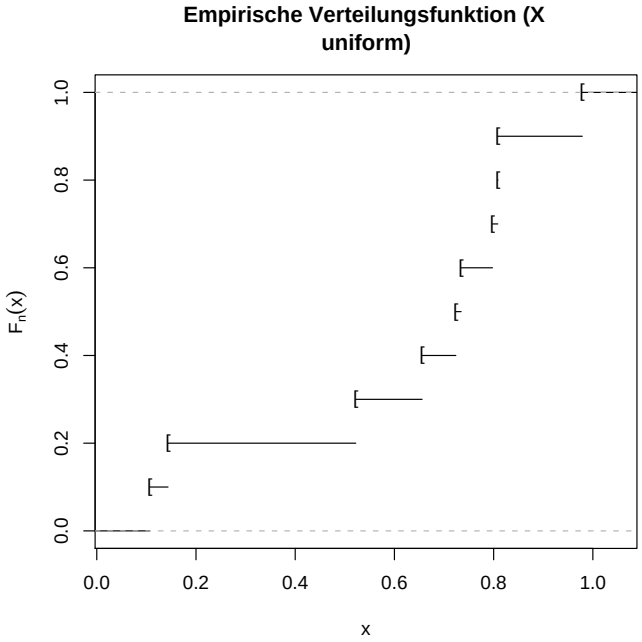
Beispiel 1.3:

Eingabe

```
x <- runif(100)
plot(sort(x), (1:length(x))/length(x), main = "Empirische Verteilungsfunktion (X unif
    xlab = "x", ylab = expression(F[n]))
abline(0, 1)
```



Dieses Beispiel dient hier nur zur Einführung. Es ist nicht notwendig, die Verteilungsfunktion selbst zu programmieren. In R gibt es z.B. die Klasse `ecdf` für die empirische Verteilungsfunktion. Wird die Funktion `plot()` auf ein Objekt der Klasse ecdf angewandt, so führt die "generische" Funktion `plot()` intern auf die spezielle Funktion `plot.ecdf`, und diese zeichnet in der für Verteilungsfunktionen speziellen Weise. Wir können das Beispiel also abkürzen durch den Aufruf `plot(ecdf( runif(100) ))`.

Aufgabe 1.3	
	Ergänzen Sie den Aufruf `plot(ecdf( runif(10) ))` durch weitere Parameter so, das die Ausgabe die folgende Form hat: 

Aufgabe 1.4	
	Mit `rnorm()` generieren Sie gaußverteilte Zufallsvariablen. Versuchen Sie, anhand der Serienplots gaußverteilte Zufallsvariablen von uniform verteilten zu unterscheiden. Benutzen Sie dann die empirischen Verteilungsfunktionen. Können Sie damit gaußverteilte von uniform verteilten unterscheiden? Die Sinus-Serie von uniform verteilten? von gaußverteilten?

1.3.2. Erster Durchgang zu Beispiel 1.2: Histogramm. Wir wählen Testmengen $A_j,\ j = 1, \ldots, J$ im Wertebereich von X. Strategie: Um zu prüfen, ob eine Zufallsfolge zu einer Verteilung P gehört, vergleiche man den Vektor $(P(A_j))_{j=1,\ldots,J}$ mit $(P_n(A_j))_{j=1,\ldots,J}$. Für die uniforme Verteilung auf $(0, 1)$ können wir z.B. die Intervalle

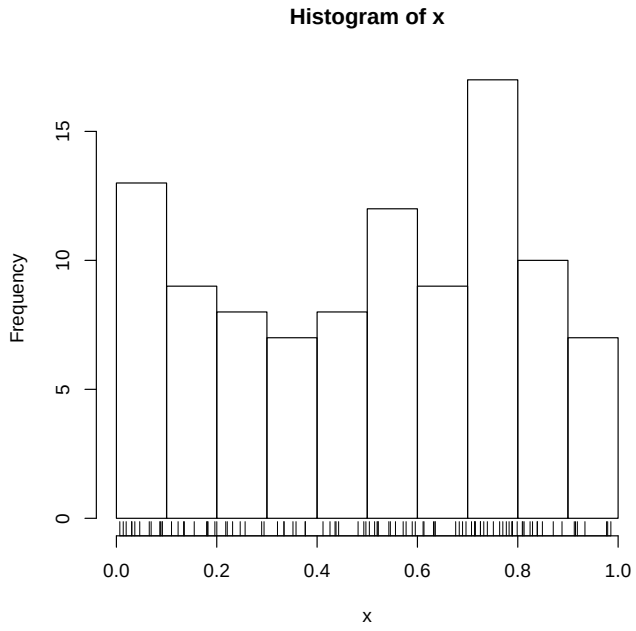
$$A_j = \Big((j-1)/J,\ \ j/J\Big]$$

als Testmengen wählen. Dann ist $(P(A_j))_{j=1,\ldots,J} = (1/J, \ldots, 1/J)$ unser Vergleichsvektor zu $(\#i : X_i \in A_j)_{j=1,\ldots,J/n}$. Vorläufige Implementierung: wir benutzen hier gleich eine vorgefertigte Funktion, die Histogramme zeichnet. Als Seiteneffekt liefert sie uns die gewünschten Werte. Mit der Funktion `rug()` können wir die Originaldaten zusätzlich einblenden.

Beispiel 1.4:

Eingabe

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

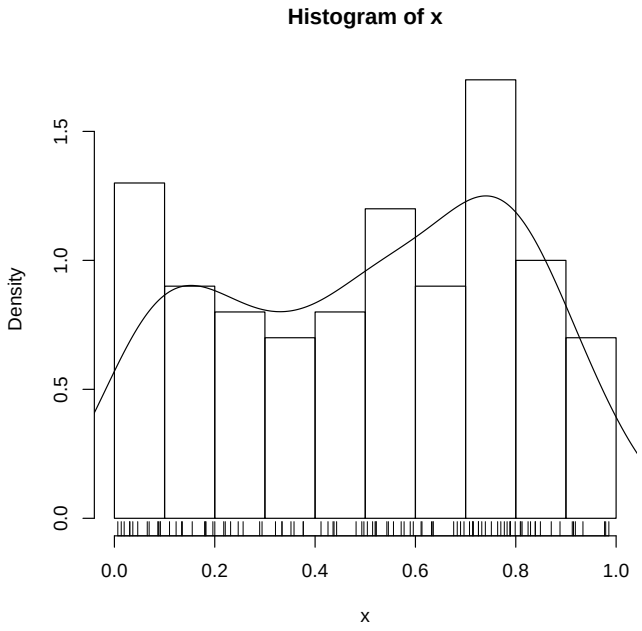


Zum Vergleich können wir einen Dichteschätzer überlagern. Da ***density()*** im Gegensatz zu ***hist()*** das Resultat nicht zeichnet, sondern ausdruckt, müssen wir die Grafik explizit anfordern. Damit die Skalen vergleichbar sind, fordern wir für das Histogramm eine Wahrscheinlichkeitsdarstellung an.

Beispiel 1.5:

Eingabe

```
hist(x, probability = TRUE)
rug(x)
lines(density(x))
```



Zurück zum Histgogramm: Benutzen wir eine Zuweisung

```
xhist<- hist(x),
```

so wird die interne Information des Histogramm unter ***xhist*** gespeichert und kann mit

```
xhist
```

abgerufen werden. Sie ergibt z.B.

Beispiel 1.6:

Eingabe

```
x <- runif(100)
xhist <- hist(x)
xhist
```

Ausgabe

```
$breaks
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

$counts
 [1] 13  2 17  7  7 10  6 11 15 12

$intensities
 [1] 1.300000 0.200000 1.700000 0.700000 0.700000 1.000000
 [7] 0.600000 1.100000 1.500000 1.200000

$density
 [1] 1.300000 0.200000 1.700000 0.700000 0.700000 1.000000
 [7] 0.600000 1.100000 1.500000 1.200000

$mids
 [1] 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95

$xname
[1] "x"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"
```

Counts gibt dabei die Besetzungszahlen der Histogrammzellen, d.h. die von uns gesuchte Anzahl. Die in ***xhist*** gespeicherte interne Information des Histogramms besteht aus fünf wesentlichen Komponenten - hier jeweils Vektoren. Diese Komponenten von ***xhist*** haben Namen und können mit Hilfe dieser Namen angesprochen werden. So gibt z.B.

```
xhist$counts
```

den Vektor der Besetzungszahlen.

R ***Datenstrukturen***	

(Fortsetzung)→

R ***Datenstrukturen*** (Fortsetzung)	
Vektoren	Komponenten eines Vektors werden über ihren Index angesprochen. Alle Elemente eines Vektors haben denselben Typ. *Beispiele:* `x` `x[10]`
Listen	Listen sind spezielle Vektoren. Die Komponenten einer Liste haben Namen, über die sie angesprochen werden können. Teilkomponenten einer Liste können von unterschiedlichem Typ sein. *Beispiele:* `xhist` `xhist$counts`

Weitere zusammengesetzte Datenstrukturen sind im Anhang (A.6) beschrieben.

Die Wahl der Histogrammgrenzen erfolgt automatisch. Um unsere Testmengen zu benutzen, müssen wir die Aufrufstruktur von `hist()` erfragen.

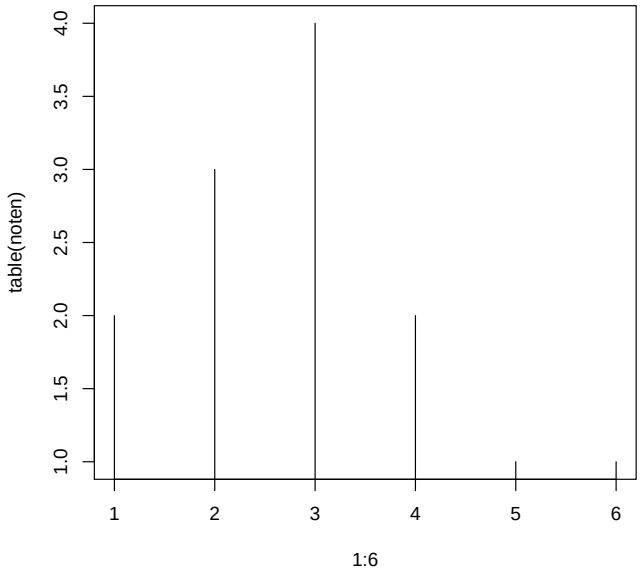
Aufgabe 1.5	
	Generieren Sie zu `runif(100)` Histogramme mit 5, 10, 20, 50 gleichgroßen Zellen und ziehen Sie wiederholt Stichproben. Entsprechen die Bilder dem, was Sie von unabhängig uniform verteilten Zufallsvariablen erwarten? Versuchen Sie, ihre Beobachtungen möglichst genau zu notieren. Wiederholen Sie das Experiment mit zwei Zellen (0,0.5], (0.5,1). `hist(runif(100), breaks =c(0,0.5,1))` Vergleichen Sie die Resultate von `runif(100)` und `rnorm(100)`.

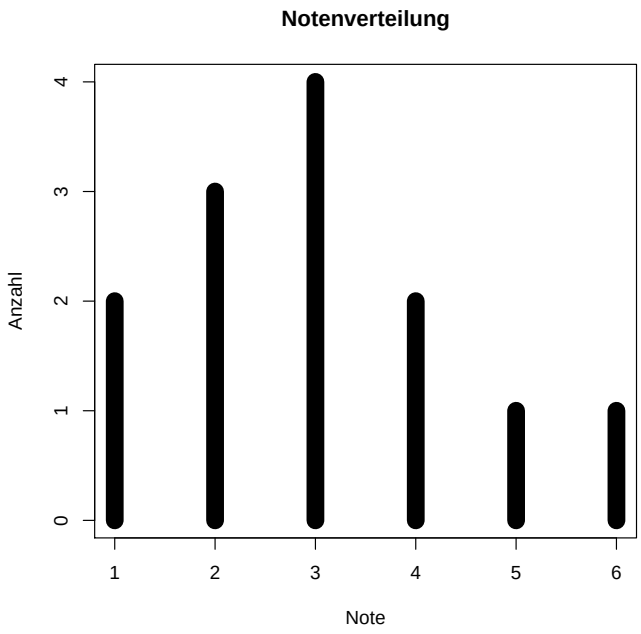
1.3.3. Balkendiagramme. Als Hinweis: wenn die Daten nicht quantitativ sind, sondern kategoriell (durch Kategorien-Label bezeichnet, wie z.B. "sehr gut, gut, befriedigend, ...", oder durch Kennziffern bezeichnet, wie z.B. "1, 2, 3, ..."), so ist ein Balkendiagramm eher geeignet. Einfache Balkendiagramme werden von `plot()` selbst durch den Parameter `type=h` unterstützt. Dazu müssen aus den Rohdaten die Häufigkeiten der einzelnen Stufen bestimmt werden. Dies kann mit der Funktion `table()` geschehen.

Beispiel 1.7:

Eingabe

```
noten <- c(2, 1, 3, 4, 2, 2, 3, 5, 1, 3, 4, 3, 6)
plot(1:6, table(noten), type = "h")
```



Aufgabe 1.6	
	Modifizieren Sie den Aufruf von `plot` im obigen Beispiel so, dass der Plot das folgende Aussehen hat: 

1.3.4. Zweiter Durchgang zum Beispiel Verteilungsfunktion. Wir machen jetzt einen Schritt von einem naiven Ansatz zu einer statistischen Betrachtung. Naiv haben wir für unabhängig identisch verteilte Variable $(X_1, \ldots, X_n)$ mit Verteilungsfunktion F angenommen, dass $i/n = F_n(X_{(i)}) \approx F(X_{(i)})$ und dies zur Überprüfung der Verteilungsannahme benutzt. Speziell für uniform (0,1) verteilte Variable ist diese naive Annahme: $i/n \approx X_{(i)} = F(X_{(i)})$.

Statistisch gesehen ist $X_{(i)}$ eine Zufallsvariable. Damit ist auch $F(X_{(i)})$ eine Zufallsvariable mit Werten in [0,1], und wir können die Verteilung dieser Zufallsvariablen untersuchen.

THEOREM 1.4. *Sind $(X_1, \ldots, X_n)$ unabhängig identisch verteilte Zufallsvariablen mit stetiger Verteilungsfunktion F, so ist $F(X_{(i)})$ verteilt nach der Beta-Verteilung $\beta(i, n - i + 1)$.*

BEWEIS. → Wahrscheinlichkeitstheorie. Hinweis: Benutze

$$X_{(i)} \leq x_\alpha \;\Leftrightarrow\; (\#j : X_j \leq x_\alpha) \geq i.$$

Für stetige Verteilungen ist $(\#j : X_j \leq x_\alpha)$ binomialverteilt mit Parametern (n, α). □

KOROLLAR 1.5.

$$E(F(X_{(i)})) = i/(n + 1).$$

Aufgabe 1.7	
	Mit `help(rbeta)` erhalten Sie Informationen über die Funktionen, die für die Beta-Verteilungen bereitstehen. Plotten Sie die entsprechenden Beta-Verteilungen für $n = 10, 50, 100$ und $i = n/4, n/2, 3n/4$. Benutzen Sie zum plotten die Funktion `curve()`. Zum Aufruf, siehe `help(curve)`.

Wir können also im statistischen Mittel für uniform auf (0,1) verteilte Variable nicht erwarten, dass $X_{(i)} \approx i/n$, sondern sollten $X_{(i)} \approx i/(n+1)$ erwarten, d.h. die "richtige Sollwertgerade" sollte mit `abline(a=0,b=n/n+1)` gezeichnet werden.

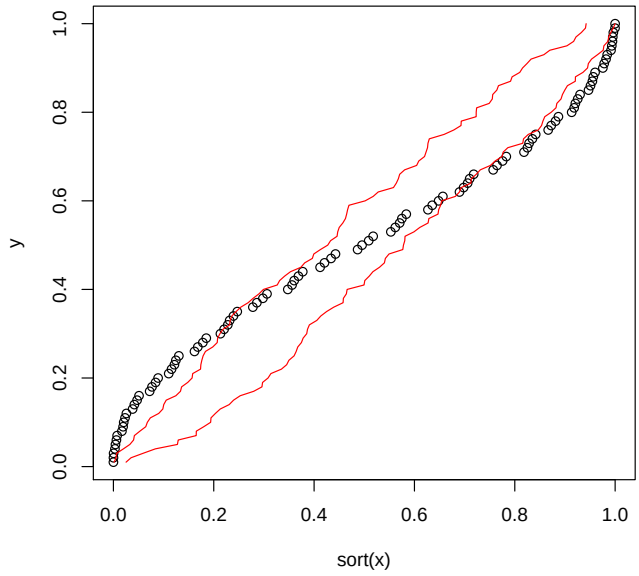
Aufgabe 1.8	
	Zeichnen Sie die Verteilungsfunktion mit der korrigierten Geraden.
*	Für die grafische Darstellung wird jeweils nur ein Plot benutzt. Ist der Erwartungswert hier der richtige Vergleichsmaßstab? Gibt es Alternativen? Falls Sie Alternativen sehen: implementieren Sie diese.

Mit einer Simulation können wir uns auch ein Bild von der typischen Fluktuation verschaffen. Wir benutzen Zufallszahlen, um eine (kleine) Anzahl von Stichproben bekannter Verteilung zu generieren, und vergleichen die in Frage stehende Stichprobe mit den Simulationen. Dazu bilden wir für die Simulationen die Einhüllende, und prüfen, ob die Stichprobe innerhalb dieses Bereichs liegt. Ist x der in Frage stehende Vektor mit Länge n, so benutzen wir z.B. die folgende Programmidee:

Beispiel 1.8:

Eingabe

```
x <- (sin(1:100) + 1)/2
y <- (1:length(x))/length(x)
plot(sort(x), y)
nrsamples <- 19
samples <- matrix(data = runif(length(x) * nrsamples),
    nrow = length(x), ncol = nrsamples)
samples <- apply(samples, 2, sort)
envelope <- t(apply(samples, 1, range))
lines(envelope[, 1], y, col = "red")
lines(envelope[, 2], y, col = "red")
```



Für die Programmierung wird hier eine für R typische Strategie erkennbar. R ist eine interpretierte vektororientierte Sprache. Einzelne Interpretationsschritte sind zeitintensiv. Deshalb sind Operationen mit weniger, dafür komplexeren Schritten effektiver als Operationen aus mehreren elementaren Schritten. Operationen auf Vektorebene sind effektiver als einzelne elementare Operationen. Iterationen und Schleifen werden vermieden zugunsten strukturierter Vektor-Operationen.

Aufgabe 1.9	
	Benutzen Sie die *help()*-Funktion und kommentieren Sie das obige Beispiel Schritt für Schritt. Notieren Sie insbesondere die neu hinzugekommenen Funktionen.

R ***Iteratoren***	
`apply()`	wendet eine Funktion auf die Zeilen oder Spalten einer Matrix an
`outer()`	erzeugt eine Matrix mit allen Paar-Kombinationen aus zwei Vektoren, und wendet eine Funktion auf jedes Paar an.

Wenn die Kurve für unsere Stichprobe die durch die Simulation gewonnenen Grenzen überschreitet, so widerspricht das der Hypothese, dass der Stichprobe und der Simulation das selbe Modell zugrunde liegt. Das hier skizzierte Verfahren heißt ***Monte-Carlo-Test***. Die Idee dahinter ist von sehr allgemeiner Bedeutung.

Aufgabe 1.10	
*	Wieso 19? *Hinweis:* versuchen Sie, das Problem zunächst abstrakt und vereinfacht zu betrachten: sei T eine messbare Funktion und $X_0, X_1, \ldots, X_{nrsamples}$ unabhängige Stichproben mit einer gemeinsamen stetigen Verteilungsfunktion. Berechnen Sie $P\left(T\left(X_0\right) > T\left(X_i\right)\right)$ für alle $i > 0$. Formulieren Sie dann das obige Beispiel abstrakt. Spezialisieren Sie dann für `nrsamples=19`.

Aufgabe 1.11	
*	Schätzen Sie die Überdeckungswahrscheinlichkeit des Monte-Carlo-Bands, in dem Sie wie folgt vorgehen: Generieren Sie zunächst analog zum obigen Beispiel ein Band. Wie können Sie das Band zeichnen, ohne zuvor für eine spezielle Stichprobe einen Plot zu machen?) Ziehen Sie für eine zu wählende Anzahl *sim* (100? 1000? 999?) jeweils eine Stichprobe von uniform verteilten Zufallszahlen vom Stichprobenumfang 100. Zählen Sie aus, wie oft die empirische Verteilungsfunktion der Stichprobe innerhalb des Bands verläuft. Schätzen Sie hieraus die Überdeckungswahrscheinlichkeit.

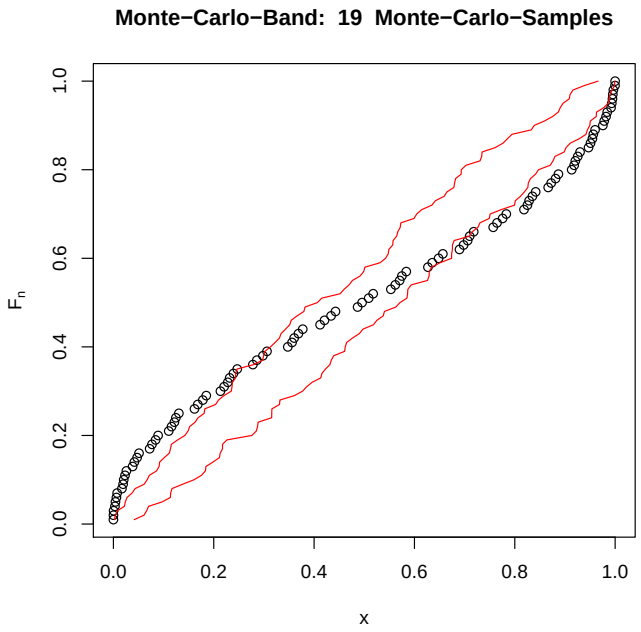
Wir wollen auch hier die Ausgabe noch überarbeiten, so dass der Plot genügend Information enthält. Bei der Beschriftung können wir zunächst analog zu Abschnitt 1.3.1 vorgehen. Die Anzahl `nrsamples` bedarf des Nachdenkens. Wenn wir nur eine feste Anzahl (z.B. 19) betrachten wollen, können wir diese wie gewohnt in die Beschriftung aufnehmen. Wenn das Programmfragment jedoch allgemeiner nutzbar sein soll, so müssten wir die jeweils gewählte Anzahl von Simulationen angeben. Dies von Hand zu tun ist eine Fehlerquelle, die vermieden werden kann. Die Funktion `bquote()` ermöglicht es, den jeweils aktuellen Wert zu erfragen. Damit die Anzahl

von Simulationen in die Überschrift übernommen werden kann, vertauschen wir die Anweisungen so dass sie vor dem Aufruf von `plot` festgelegt ist.

Beispiel 1.9:

Eingabe

```
x <- (sin(1:100) + 1)/2
y <- (1:length(x))/length(x)
nrsamples <- 19
plot(sort(x), y, ylab = expression(F[n]), xlab = "x",
    main = paste("Monte-Carlo-Band: ", bquote(.(nrsamples)),
        " Monte-Carlo-Samples"))
samples <- matrix(data = runif(length(x) * nrsamples),
    nrow = length(x), ncol = nrsamples)
samples <- apply(samples, 2, sort)
envelope <- t(apply(samples, 1, range))
lines(envelope[, 1], y, col = "red")
lines(envelope[, 2], y, col = "red")
```



Für die praktische Arbeit kann es notwendig sein, die Verteilungsdiagnostik auf ein einfaches Entscheidungsproblem zu reduzieren, etwa um anhand von Tabellen oder Kontrollkarten zu entscheiden, ob eine Verteilung in einem hypothetischen Bereich liegt, oder eine Kenngröße anzugeben, die die Abweichung von einem gegebenen Modell charakterisiert. Wenn wir auf Tabellen oder einfache Zahlen zurückgreifen wollen, müssen wir uns weiter einschränken. Wir müssen die Information, die in den Funktionen (F_n, F) steckt, weiter reduzieren, wenn wir die Unterschiede numerisch

zusammenfassen wollen. Eine Zusammenfassung ist etwa

$$\sup_x |F_n - F|(x).$$

Wenn wir diese Zusammenfassung als Kriterium benützen wollen, stehen wir wieder vor der Aufgabe, ihre Verteilung zu untersuchen.

THEOREM 1.6. *(Kolmogroff, Smirnoff) Für stetige Verteilungsfunktionen F ist die Verteilung von*

$$\sup_x |F_n - F|(x)$$

unabhängig von F (jedoch abhängig von n).

BEWEIS. → Wahrscheinlichkeitstheorie. Z.B. [Gänssler & Stute, Lemma 3.3.8]. □

THEOREM 1.7. *(Kolmogoroff): Für stetige Verteilungsfunktionen F und $n \to \infty$ hat*

$$\sqrt{n} \sup |F_n - F|$$

asymptotisch die Verteilungsfunktion

$$F_{Kolmogoroff-Smirnoff}(y) = \sum_{m \in \mathbb{Z}} (-1)^m e^{-2m^2y^2} \text{ für } y > 0.$$

BEWEIS. → Wahrscheinlichkeitstheorie. Z.B. [Gänssler & Stute, Formel (3.3.11)]. □

Für die praktische Arbeit bedeutet dies: Für stetige Verteilungsfunktionen können wir eine Entscheidungsstrategie formulieren: wir entscheiden, dass die Beobachtung $(X_1, \ldots, X_n)$ nicht mit der Hypothese von unabhängig, identisch nach F verteilten Zufallsvariablen vereinbar ist, falls $\sup |F_n - F|$ zu groß ist:

$$\sup |F_n - F| > F_{krit},$$

wobei F_{krit} aus der (von F unabhängigen) Verteilungsfunktion der Kolmogoroff-Smirnoff-Statistik zum Stichprobenumfang n entnommen wird. Wählen wir speziell das obere α-Quantil $F_{krit} = F_{Kolmogoroff-Smirnoff,1-\alpha}$, so wissen wir, dass bei Zutreffen der Hypothese der Wert F_{krit} oder ein höherer Wert höchstens mit Wahrscheinlichkeit α erreicht wird. Damit können wir unsere Irrtumswahrscheinlichkeit für eine ungerechtfertigte Ablehnung der Hypothese kontrollieren.

Asymptotisch, für große n, können wir anstelle der Verteilungsfunktion die Kolmogoroff-Approximation benutzen. Wenn die Modellverteilung F nicht stetig ist, sind weitere Überlegungen nötig.

Wir wollen uns hier auf die Programmierung konzentrieren und gehen nicht in die Details des Kolmogoroff-Smirnoff-Tests. Mit elementaren Mitteln können wir die Teststatistik $\sup_x |F_n - F|(x)$ für die uniforme Verteilung programmieren. Aus Monotoniegründen ist

$$\sup_x |F_n - F|(x) = \max_{X(i)} |F_n - F| X(i)$$

und für die uniforme Verteilung ist

$$\max_{X(i)} |F_n \ - \ F|\, X(i) = \max_i |i/n \ - \ X(i)|.$$

Damit gibt in R-Schreibweise

```
max( abs((1: length(x)) / length(x)) - sort(x)) )
```

für uns die gewünschte Statistik, wenn `x` unser Datenvektor ist.

Diese Statistik (und viele weitere allgemein benutzte Statistiken) sind in der Regel schon programmiert, ebenso wie die zugehörigen Verteilungsfunktionen. [2]

Aufgabe 1.12	

	Mit
	`help(ks.test)`
	erhalten Sie die Information, wie die Funktion `ks.test` angewandt wird.
	Welche Resultate erwarten Sie, wenn Sie die folgenden Vektoren auf uniforme Verteilung testen:
	`1:100`
	`runif(100)`
	`sin(1:100)`
	`rnorm(1:100)`?
	Führen Sie diese Tests durch und diskutieren Sie die Resultate.

1.3.5. Zweiter Durchgang zum Beispiel 1.2: Histogramm. Wie bei der Verteilungsfunktion machen wir einen Schritt in Richtung auf eine statistische Analyse. Der Einfachheit halber nehmen wir an, dass wir disjunkte Testmengen $A_j, j = 1, \ldots, J$ gewählt haben, die den Wertebereich von X überdecken Die Beobachtung $(x_1, \ldots, x_n)$ gibt dann Besetzungszahlen n_j

$$n_j = (\#i : X_i \in A_j).$$

Wenn $(X_i)_{i=1,\ldots,n}$ unabhängig sind mit identischer Verteilung P, so ist $(n_j)_{j=1,\ldots,J}$ ein Zufallsvektor mit Multinomialverteilung zu den Parametern $n, (p_j)_{j=1,\ldots,J}$ mit $p_j = P(A_j)$. Für den Spezialfall $J = 2$ haben wir die Binomialverteilung. Da wir freie Wahl über die Testmengen A_j haben, können wir damit eine ganze Reihe von oft hilfreichen Spezialfällen abdecken, z.B.

Mediantest auf Symmetrie:

$$A_1 = \{x < x_{0.5}\} \qquad A_2 = \{x \geq x_{0.5}\}$$

[2] Vor R Version 2.x gehörten diese jedoch nicht zum Basis-Umfang von R, sondern sind in speziellen Bibliotheken enthalten, die explizit hinzugeladen werden mussten. Die Bibliothek mit klassischen Tests in der R1.x-Implementierung heißt `ctest` und wird mit

```
library(ctest)
```

geladen. Unterschiedliche Implementierungen können hier andere Aufrufstrukturen vorsehen. Der Kolmogoroff-Smirnoff-Test findet sich in der Bibliothek `ctest` als `ks.test`.

Midrange-Test auf Konzentration:

$$A_1 = \{x_{0.25} \leq x < x_{0.75}\} \qquad A_2 = \{x < x_{0.25} \text{ oder} x \geq x_{0.75}\}.$$

Für den allgemeinen Fall müssen wir jedoch die empirischen Besetzungszahlen n_j anhand der Multinomialverteilung beurteilen, und diese ist sehr unangenehm zu berechnen. Deshalb greift man oft auf Approximationen zurück. Auf Pearson geht folgende Approximation zurück:

LEMMA 1.8. *(Pearson): Für* $(p_j)_{j=1,\ldots,J}$, $p_j > 0$ *gilt im Limes* $n \to \infty$ *die Approximation*

$$\begin{aligned} P_{mult}(n_1, \ldots, n_j) \approx \\ (2\pi n)^{-1/2} \Big(\prod_{j=1,\ldots,J} p_j \Big)^{-1/2} \cdot \\ \exp\Big(-1/2 \sum_{j=1,\ldots,J} \frac{(n_j - np_j)^2}{np_j} \\ -1/2 \sum_{j=1,\ldots,J} \frac{n_j - np_j}{np_j} \\ + 1/6 \sum_{j=1,\ldots,J} \frac{(n_j - np_j)^3}{(np_j)^2} + \ldots \Big). \end{aligned}$$

BEWEIS. → Wahrscheinlichkeitstheorie. Z.B. [**JK70**] p. 285. □

Der erste Term wird bestimmt von $\widehat{\chi^2} := \sum_{j=1\ldots J}(n_j - np_j)^2/np_j$. Dieser Term wird χ^2-Statistik genannt. Zumindest asymptotisch für $n \to \infty$ führen große Werte von $\widehat{\chi^2}$ zu kleinen Wahrscheinlichkeiten. Dies motiviert, die χ^2-Statistik approximativ als Anpassungsmaß zu benutzen. Der Name kommt aus der Verteilungsasymptotik:

THEOREM 1.9. *(Pearson): Für* $(p_j)_{j=1,\ldots,J}, p_j > 0$ *ist im Limes* $n \to \infty$ *die Statistik*

$$\widehat{\chi^2} := \sum_{j=1,\ldots,J} \frac{(n_j - np_j)^2}{np_j}$$

χ^2*-verteilt mit* $J - 1$ *Freiheitsgraden.*

Für eine formale Entscheidungsregel können wir wieder einen kritischen Wert χ^2_{krit} festlegen, und die Hypothese, dass die Beobachtungen $(X_1, \ldots, X_n)$ identisch uniform verteilte Zufallszahlen sind, wenn die χ^2-Statistik über diesem Wert liegt. Wählen wir als kritischen Wert das obere α-Quantil der χ^2-Verteilung, so wissen wir, dass bei Zutreffen der Hypothese der Wert χ^2_{krit} oder ein höherer Wert höchstens mit Wahrscheinlichkeit α erreicht wird. Damit können wir zumindest asymptotisch auch hier unsere Irrtumswahrscheinlichkeit für eine ungerechtfertigte Ablehnung der Hypothese kontrollieren.

Die χ^2-Tests gehören zu Basisumfang von R als Funktion `chisq.test()`. Sie sind so ausgelegt, dass sie für allgemeinere "Kontingenztafeln" genutzt werden können. Wir benötigen sie hier nur für einen Spezialfall: die Tafel ist in unserem Fall der

(eindimensionale) Vektor der Besetzungszahlen für vorgewählte Zellen. (Hinweis: in der R-Implementierung sind allgemeinere Varianten in der Bibliothek `loglin` zu finden.)

Aufgabe 1.13	
	Orientieren Sie sich mit `help(chisq.test)` über die Aufrufstruktur der χ^2-Tests. Wenden Sie ihn für die Hypothese $(p_j = 1/J), J = 5$ auf folgende Vektoren von Besetzungszahlen an: (3 3 3 3 3) (1 2 5 3 3) (0 0 9 0 6). Wenden Sie den Test auf den Vektor der Besetzungszahlen an, den Ihnen das Histogramm liefert. *Hinweis:* die Prozedur `table()` gibt die Möglichkeit, Besetzungstabellen direkt zu erstellen.
	Programmieren Sie zum Vergleich die Berechnung der χ^2-Statistik direkt in R. Dazu können Sie die Funktion `sum()` benutzen. Die Verteilungsfunktion der χ^2-Statistik können Sie mit `pchisq()` berechnen.

Die Approximationen für die χ^2-Statistik gelten zunächst nur, wenn die Zellen fest gewählt sind. Praktische Histogramm-Algorithmen bestimmen jedoch Zellenanzahl und Zellgrenzen aufgrund der Stichprobe. Dazu werden (implizit) Parameter der Verteilung geschätzt. Unter bestimmten Voraussetzungen gilt noch immer eine -Asymptotik, wie z.B nach dem folgenden Theorem aus [**Rao73**]:

THEOREM 1.10. *(i) Let the cell probabilities be the specified functions* $\pi_1(\boldsymbol{\theta}), \ldots, \pi_k(\boldsymbol{\theta})$ *involving* q *unknown parameters* $(\theta_1, \ldots, \theta_q) = \boldsymbol{\theta}'$. *Further let*

(a) $\widehat{\boldsymbol{\theta}}$ *be an efficient estimator of* $\boldsymbol{\theta}$ *in the sense of (5c.2.6),*
(b) each $\pi_i(\boldsymbol{\theta})$ *admit continuous partial derivatives of the first order (only) with respect to* θ_j, $j = 1, \ldots, q$ *or each* $\pi_i(\boldsymbol{\theta})$ *be a totally differentiable function of* $\theta_1, \ldots, \theta_q$, *and*
(c) the matrix $M = (\pi_r^{-1/2} \partial \pi_r / \partial \theta_s)$ *of order* $(k \times q)$ *computed at the true values of* $\boldsymbol{\theta}$ *is of rank* q. *Then the a..d of*

$$\chi^2 = \sum \frac{(n_i - n\widehat{\pi}_i)^2}{n\widehat{\pi}_i} = \sum \frac{(0 - E)^2}{E} \tag{1.1}$$

is $\chi^2(k - 1 - q)$, *where* $\widehat{\pi}_i = \pi_i(\widehat{\theta})$.

BEWEIS. Siehe [**Rao73**] Abschnitt 6b.2. □

Aufgabe 1.14	
*	Entwerfen Sie vergleichbare Testumgebungen für feste und adaptive Zellwahlen. Ziehen Sie für beide `s=1000` Stichproben aus `runif` vom Umfang 50; berechnen Sie formal die χ^2-Statistik und plotten Sie deren Verteilungsfunktion. Vergleichen Sie die Verteilungsfunktionen.

Wiederholte Stichproben

Wir haben uns bis jetzt darauf konzentriert, die Verteilung einer Zufallsvariablen zu untersuchen. Wir können das Verfahren fortsetzen. Wenn $(X_1, \ldots, X_n)$ identisch uniform verteilte Zufallszahlen sind, dann ist bei vorgewählten Zellen die χ^2-Statistik approximativ χ^2-verteilt, und $\kappa := \sqrt{n} \sup |F_n - F|$ hat asymptotisch die Kolmogoroff-Smirnoff-Verteilung.

Wir können wiederholt Stichproben $(X_{1j}, \ldots, X_{nj})_{j=1..m}$ ziehen und daraus Statistiken $\widehat{\chi^2}_j$ und κ_j berechnen. Bei unabhängig, identisch verteilten Ausgangsdaten müssen diese nach χ^2 bzw. Kolmogoroff-Smirnoff verteilt sein. Bei diesen wiederholten Stichproben wird nicht nur die Verteilung der einzelnen Beobachtungen, sondern die gemeinsame Verteilung der jeweils n Stichprobenelemente untersucht.

Aufgabe 1.15	
	Ziehen Sie für $n = 10, 50, 100$ wiederholt jeweils 300 Stichproben nach `runif()`. Berechnen Sie dafür jeweils die χ^2- und Kolmogoroff-Smirnoff-Statistik. Welchen χ^2-Test benutzen Sie? Plotten Sie die Verteilungsfunktionen dieser Statistiken und vergleichen Sie sie mit den theoretischen asymptotischen Verteilungen. Sprechen irgendwelche Befunde gegen die Annahme unabhängig uniform verteilter Zufallszahlen? *Hinweis:* die χ^2- und Kolmogoroff-Smirnoff-Funktionen speichern ihre Information intern als Liste. Um die Namen der Listenelemente zu bekommen, kann man sich ein Testobjekt generieren. Benutzen Sie z.B. `names(chisq.test(runif(100)))`.

Die uniforme Verteilung war in unserer Diskussion bislang die angezielte Modellverteilung, unsere "Hypothese". Wir haben diskutiert, wie die unterschiedlichen Verfahren sich verhalten müßten, wenn diese Hypothese gilt. Das daraus abgeleitete Verteilungsverhalten kann dazu dienen, kritische Grenzen für formale Tests festzulegen. Wir verwerfen die Hypothese, wenn die beobachteten Test-Statistiken zu extrem sind. Was "zu extrem" ist, wird anhand der abgeleiteten Verteilungen bestimmt. Dies führt zu Entscheidungsregeln wie:

verwerfe die Hypothese, wenn $F_{\chi^2}(\widehat{\chi^2}) \geq 1 - \alpha$
oder
verwerfe die Hypothese, wenn $F_{Kolmogoroff-Smirnoff}(\kappa) \geq 1 - \alpha$
für geeignet festzulegende (kleine) Werte von α.

Wenn wir ein Entscheidungsverfahren formal festgelegt haben, können wir im nächsten Schritt fragen, wie scharf das Verfahren ist, wenn die Hypothese tatsächlich abzulehnen ist. Eine genauere Analyse bleibt der Statistik-Vorlesung vorbehalten. Mit den bis jetzt diskutierten Möglichkeiten können wir jedoch schon das Verhalten mit einer Monte-Carlo-Strategie untersuchen.

Als Simulations-Szenario wählen wir eine Familie von Alternativen. Die uniforme Verteilung fügt sich in die Beta-Verteilungen mit den Dichten

$$p_{a,b}(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1} \quad \text{für } a > 0, b > 0 \text{ und } 0 < x < 1$$

ein. Wir wählen als Alternativen Verteilungen aus dieser Familie. Daraus ziehen wir wiederholt Stichproben, und wenden jeweils formal unsere Entscheidungsverfahren an. Wir registrieren, ob das Verfahren zu einer Ablehnung der Hypothese führt oder nicht. Zu gegebener Wahl eines Stichprobenumfangs n und einer Wiederholungsanzahl m und bei Wahl einer Grenzwahrscheinlichkeit a erhalten wir eine Tabelle

$$(a, b) \;\mapsto\; \#\text{ Simulationen, bei denen die Hypothese verworfen wird.}$$

Speziell für die uniforme Verteilung $(a, b) = (1, 1)$ erwarten wir annähernd $m \cdot \alpha$ Verwerfungen. Für andere Verteilungen ist ein Verfahren um so entscheidungsschärfer, je größer der Anteil der Verwerfungen ist.

Aufgabe 1.16	
**	Untersuchen Sie die Trennschärfe des Kolmogoroff-Smirnoff-Test und des χ^2-Tests. Wählen Sie jeweils einen Wert für n, m und α, und wählen Sie 9 Paare für (a, b). Notieren Sie die Überlegungen hinter Ihrer Wahl. Ziehen Sie zu diesen Paramtern mit `rbeta()` Zufallsstichproben. Führen Sie jeweils den Kolmogoroff-Smirnoff-Test und einen χ^2-Test mit 10 gleichgroßen Zellen auf $(0, 1)$ durch.
	Wählen Sie Alternativparameter (a, b) so, dass Sie entlang der folgenden Geraden die Entscheidungsverfahren vergleichen können: i) a=b ii) b=1 iii) a=1 und führen Sie eine entsprechende Simulation durch.
	Wählen Sie Alternativparameter (a, b) so, dass Sie für den Bereich $0 < a, b < 5$ die Entscheidungsverfahren vergleichen können. Ihre Schlüsse? (Fortsetzung)→

Aufgabe 1.16	(Fortsetzung)
	Hinweis: Mit `outer(x,y, fun)` wird eine Funktion fun auf alle Paare aus den Werten von x, y angewandt und das Ergebnis als Resultat zurückgeliefert. Mit `contour()` können Sie einen Contour-Plot erzeugen. Siehe `demo("graphic")`.

Aufgabe 1.17	
⋆⋆	Entwerfen Sie eine Prüfstrategie, um "Pseudozufallszahlen" zu entlarven. Testen Sie diese Strategie an einfachen Beispielen i) $x \quad x = 1..100 \mod m$ für geeignete m ii) $\sin(x)$ $x = 1..100$ iii) ... Werden diese als "nicht zufällig" erkannt? Versuchen Sie dann, die bereitgestellten Zufallszahlengeneratoren zu entlarven.

1.4. Momente und Quantile

Verteilungsfunktionen oder Dichten sind mathematisch nicht einfach zu handhaben: der Raum der Funktionen ist im allgemeinen unendlich-dimensional und endliche geometrische Argumente oder endliche Optimierungsargumente sind nicht direkt anwendbar. Um die Analyse zu vereinfachen, greift man bisweilen auf endliche Beschreibungen zurück.

Historisch haben die Momente eine wichtige Rolle gespielt: Wahrscheinlichkeiten werden als Masse-Verteilungen interpretiert, und die Momente analog zu den Momenten der Mechanik eingeführt. Das erste Moment, entsprechend dem Schwerpunkt, heißt in der Statistik ***Erwartungswert***.

Definition 1.11. Ist X eine reellwertige Zufallsvariable mit Verteilung P, so ist der Erwartungswert von X definiert als

$$E_P(X) := E(X) := \int X dP.$$

Das zweite Moment und höhere Momente werden konventionell zentriert. Für das zweite (zentrale) Moment, die ***Varianz***, haben wir die folgende Definition:

Definition 1.12. Ist X eine reellwertige Zufallsvariable mit Verteilung P, so ist die Varianz von X definiert als

$$Var_P(X) := Var(X) := \int (X - E(X))^2 dP.$$

Die Integralausdrücke müssen nicht immer definiert sein, d.h. die Momente müssen nicht immer existieren. Existieren sie jedoch, so geben sie eine erste Information über die Verteilung. Der Erwartungswert wird oft als das "statistische Mittel" interpretiert; die Wurzel aus der Varianz, die ***Standardabweichung***, als "Streuung".

Die Definitionen können selbstverständlich auch auf empirische Verteilungen angewandt werden. Dies ermöglicht es, die Momente einer unbekannten theoretischen Verteilung aus den Daten zu schätzen. Für den Mittelwert gilt Konsistenz:

$$E_P(E_{P_n}(X)) = E_P(X),$$

d.h. im statistischen Mittel stimmen empirischer Erwartungswert und Erwartungswert der zu Grunde liegenden Verteilung überein (falls definiert).

Für die Varianz gilt diese Konsistenz nicht, sondern es gilt

$$\frac{n}{n-1} E_P(Var_{P_n}(X)) = Var_P(X),$$

falls $n > 1$. Der mathematische Hintergrund ist dass der Erwartungswert ein linearer Operator ist. Er kommutiert mit linearen Operatoren. Aber die Varianz ist ein quadratischer Operator, und dass macht eine Korrektur nötig, wenn man Konsistenz will. Die entsprechend korrigierte Varianz wird oft als ***Stichprobenvarianz*** bezeichnet.

Für die Schätzung der ersten beiden Momente eines Vektor von Zufallszahlen stehen in R Funktionen bereit: `mean()` schätzt den Mittelwert und `var()` die (Stichproben-)Varianz. Die Funktion `sd()` schätzt die Standardabweichung eines Vektors.

Aufgabe 1.18	
	Generieren Sie jeweils eine Stichprobe von 100 Zufallsvariablen aus den Verteilungen mit den folgenden Dichten: $$p(x) = \begin{cases} 0 & x < 0 \\ 1 & 0 \le x \le 1 \\ 0 & x > 1 \end{cases}$$ sowie $$p(x) = \begin{cases} 0 & x \le 0 \\ 2 & 0 < x \le 1/4 \\ 0 & 1/4 < x \le 3/4 \\ 2 & 3/4 < x \le 1 \\ 0 & x > 1 \end{cases}$$ Schätzen Sie dazu Mittelwert, Varianz und Standardabweichung. (Fortsetzung)→

Aufgabe 1.18	(Fortsetzung)
	Wiederholen Sie die Schätzung für 1000 Stichproben. Analysieren Sie die Verteilung von geschätztem Mittelwert, Varianz und Standardabweichung bei wiederholten Stichproben.

Momente sind durch einfache arithmetische Operationen zu berechnen und ihre Kombination folgt (exakt oder approximierbar) einfachen Gesetzen. Sie sind jedoch sehr sensitiv. Die Verschiebung einer beliebig kleinen Wahrscheinlichkeitsmasse kann sie zum Zusammenbruch bringen. Für die empirische Verteilung bedeutet dies: stammen die beobachteten Daten zu einem Anteil $1-\varepsilon$ aus einer Modellverteilung und zu einem Anteil ε aus einer anderen Verteilung, so können die Momente jeden beliebigen Wert annehmen, für jeden beliebig kleinen Wert von ε.

Mit der Verfügbarkeit von programmierbaren Rechnern haben Quantile als beschreibende Größe an Bedeutung gewonnen. Ihre Berechnung setzt implizit eine Sortier-Operation voraus, ist also komplexer als die Berechnung von Momenten. Auch die Regeln zur Kombination sind nicht so einfach wie bei Momenten und setzt oft eine explizite Rechnung voraus. Aber mit den verfügbaren technischen Mitteln ist dies keine wesentliche Einschränkung.

Quantile sind gegenüber einem Zusammenbruch robuster als Momente. So müssen 50% der Daten "Ausreißer" sein, bis der der Median beinflusst wird.

R bietet eine Reihe von Funktionen, um mit Quantilen zu arbeiten. *`quantile()`* ist eine elementare Funktion, um Quantile zu bestimmen. Die Funktion *`summary()`* bestimmt Zusammenfassung der Verteilungsinformation, die auf Quantilen basiert ist.

Mit *`boxplot()`* erhält man eine grafische Repräsentation dieser Zusammenfassung. Der hier benutzte "Box&Whisker-Plot" hat eine Reihe von Variationen. Deshalb ist es bei der Interpretation notwendig, sich jeweils über die benutzten Details zu informieren. Üblich ist eine Kennzeichnung durch eine "Box", die den zentralen Teil der Verteilung beschreibt. In der Standardversion kennzeichnet eine Linie den Median, und eine "Box" darum reicht vom Median der oberen Hälfte bis zum Median der unteren Hälfte. Grob entspricht dies dem oberen und dem unteren Quartil. Die feinere Definition sorgt dafür, dass die Information auch noch verlässlich wieder gegeben wird, wenn Bindungen, d.h. vielfache Beobachtungen des selben Wertes auftreten. Die "Whisker" beschreiben die angrenzenden Bereiche. Ausreißer sind besonders gekennzeichnet.

Aufgabe 1.19	
	Generieren Sie jeweils eine Stichprobe von 100 Zufallsvariablen aus den Verteilungen von Aufgabe 1.33. Schätzen Sie dazu Median, oberes und unteres Quartil. (Fortsetzung)→

Aufgabe 1.19	(Fortsetzung)
	Wiederholen Sie die Schätzung für 1000 Stichproben. Analysieren Sie die Verteilung von geschätztem Median, oberen und unterem Quartil bei wiederholten Stichproben.

Theorem 1.4 gibt eine Möglichkeit, Konfidenzintervalle für Quantile zu bestimmen, die allgemein gültig sind, unabhängig von der Form der zu Grunde liegenden Verteilung.

Um das $p-$Quantil x_p einer stetigen Verteilungsfunktion durch eine Ordnungsstatistik $X_{(k:n)}$ zum Konfidenzniveau $1-\alpha$ nach oben abzuschätzen, suchen wir

$$\min_k : P(X_{(k:n)} \geq x_p) \geq 1-\alpha.$$

Aber $X_{(k:n)} \geq x_p \iff F(X_{(k:n)} \geq p$ und wegen Theorem 1.4 ist damit

$$P(X_{(k:n)} \geq x_p) = 1 - F_{beta}(p; k, n-k+1).$$

Wir können also $\min_k$ direkt aus der Beta-Verteilung ermitteln, oder wir benutzen die Beziehung zur Binomialverteilung und bestimmen k als

$$\min_k : P_{bin}(X \leq k-1; n, p) \geq 1-\alpha.$$

Aufgabe 1.20	
	Für stetige Verteilungen und den Verteilungsmedian X_{med} ist $P(X_i \geq X_{med}) = 0.5$. Deshalb kann ein k so bestimmt werden, dass $$k = min\{k : P(X_{(k)} \leq X_{med}) < \alpha\}$$ und $X_{(k)}$ als obere Abschätzung für den Median zum Konfindenzniveau $1-\alpha$ gewählt werden. Konstruieren Sie mit dieser Idee ein Konfidenzintervall für den Median und modifizieren Sie den Box & Whiskerplot so, dass er dieses Intervall einzeichnet. *Hinweis:* Sie benötigen dazu die Verteilungsfunktion F_X, ausgewertet an der durch die Ordnungsstatistik $X_{(k)}$ definierten Stelle. Die Verteilungen von $F_X(X_{(k)})$ wird im ersten Kapitel diskutiert.
	Der Boxplot bietet eine Option `notch=TRUE`, um Konfidenzintervalle zu generieren. Versuchen Sie, mithilfe der Dokumentation herauszufinden, wie ein `notch` bestimmt wird. Vergleichen Sie Ihre Konfidenzintervalle mit den durch `notch` gekennzeichneten Intervallen.
*	Bestimmen Sie analog ein verteilungsunabhängiges Konfidenzintervall für den Interquartilsabstand.
* * *	Ergänzen Sie den Box & Whiskerplot so, dass er die Skaleninformation statistisch verlässlich darstellt. *Hinweis:* Wieso reicht es nicht, Konfidenzintervalle für die Quartile einzuzeichnen?

95%-Konfidenzintervalle für den Median.
`notch`-Option von `boxplot()`.

1.5. Ergänzungen

1.5.1. Ergänzung: Zufallszahlen. Wenn wir identisch uniform verteilte Zufallszahlen hätten, könnten wir auch Zufallszahlen mit vielen anderen Verteilungen generieren. Z.B.

LEMMA 1.13. *(Inversionsmethode): Ist (U_i) eine Folge unabhängiger $U[0,1]$ verteilter Zufallsvariablen und F eine Verteilungsfunktion, so ist $(X_i) := (F^{-1}U_i)$ eine Folge unabhängiger Zufallsvariablen mit Verteilung F.*

Analytisch ist dieses Lemma nur brauchbar, wenn F^{-1} bekannt ist. Numerisch hilft es jedoch viel weiter: anstelle von F^{-1} werden Approximationen benutzt, oft sogar nur eine Inversionstabelle.

Die Inversionsmethode ist eine Methode, aus gleichverteilten Zufallszahlen andere Zielverteilungen abzuleiten. Weitere (evtl. effektivere) Methoden, aus gleichverteilten Zufallszahlen andere Zielverteilungen abzuleiten, werden in der Literatur zur statistischen Simulation diskutiert.

Für eine Reihe von Verteilungen werden transformierte Zufallsgeneratoren bereitgestellt. Eine Liste ist im Anhang (Seite A-39) angegeben. Zu jeder Verteilungsfamilie gibt es dabei eine Reihe von Funktionen, deren Namen aus einem Kurznamen für die Verteilung abgeleitet sind. Für die Familie *xyz* ist `rxyz` eine Funktion, die Zufallszahlen erzeugt. `dxyz` berechnet die Dichte bzw. das Zählmaß für diese Familie, `pxyz` die Verteilungsfunktion, und `qxyz` die Quantile[3].

Übersicht: einige ausgewählte Verteilungen

Verteilung	***Zufalls-zahlen***	***Dichte***	***Verteilungs-funktion***	***Quantile***
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Hypergeometrisch	`rhyper`	`dhyper`	`phyper`	`qhyper`
Poisson	`rpois`	`dpois`	`ppois`	`qpois`
Gauß	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Exponential	`rexp`	`dexp`	`pexp`	`qexp`

1.5.2. Ergänzung: Grafische Vergleiche. Abweichungen von einfachen geometrischen Formen werden besser wahrgenommen als Abweichungen zwischen allgemeinen Grafen ähnlicher Form. Deshalb kann es hilfreich sein, Darstellungen zu wählen, die auf einfache Formen wie z.B. Geraden führen. So wählt man um zwei Verteilungsfunktionen F, G zu vergleichen anstelle der Funktionsgraphen den Grafen

[3] d.h. mit den in der Statistik üblichen Bezeichnungen ist verwirrender Weise $p_{xyz} \equiv$ `dxyz` und $F_{xyz} \equiv$ `pxyz`.

von

$$x \mapsto (F(x), G(x)).$$

Dieser Graph heißt PP***-Plot*** oder ***probability plot***. Stimmen die Verteilungen überein, so ist der Plot eine diagonale Gerade. Abweichungen von der Diagonalgestalt sind leicht zu erkennen.

Aufgabe 1.21	
	Benutzen Sie PP-Plots anstelle von Verteilungsfunktionen, um die χ^2- und Kolmogoroff-Smirnoff-Approximationen darzustellen.

Alternativ kann die Merkmalsskala als Bezug genommen werden und der Graph von

$$p \mapsto (F^{-1}(p), G^{-1}(p))$$

betrachtet werden. Dieser Graph heißt QQ***-Plot*** oder ***Quantilplot***. Der Quantilplot ist als Funktion `qqplot()` bereitgestellt. Stimmen die Verteilungen überein, so zeigt auch dieser Plot eine diagonale Gerade. Können die Verteilungen durch eine affine Transformation im Merkmalsraum ineinander überführt werden, so zeigt der Plot immer noch eine Gerade; Steigung und Achsenabschnitt repräsentieren die affine Transformation.

Aufgabe 1.22	
	Benutzen Sie QQ-Plots anstelle von Verteilungsfunktionen. Können Sie in diesem Plot mit Hilfe der χ^2- bzw. Kolmogoroff-Smirnoff-Stastistik Konfidenzbereiche darstellen?

Um einen Eindruck über die Fluktuation zu bekommen, können wieder Monte-Carlo-Bänder konstruiert werden.

Aufgabe 1.23	
	Erzeugen Sie sich mit `rnorm()` Pseudozufallszahlen für die Gaußverteilung zum Stichprobenumfang $n = 10, 20, 50, 100$. Erzeugen Sie jeweils einen PP-Plot und einen QQ-Plot, wobei die theoretische Gaußverteilung als Bezug dient. Der QQ-Plot ist für diese Situation bereits als Funktion `qqnorm()` vorbereitet.
	Fügen Sie Monte-Carlo-Bänder aus der Einhüllenden von 19 Simulationen hinzu.

1.5.3. Ergänzung: Grafik-Aufbereitung. Bislang wurde die R-Grafik in rudimentärer Form benutzt. Für ernsthafte Arbeit muss die Grafik so aufbereitet werden, dass ihre Bestandteile identifiziert und wiedererkennbar sind. Dazu gehören Überschriftungen, Achsenkennzeichnungen etc. R unterscheidet zwischen "high level"-Grafik und "low level". "High level"-Funktionen erzeugen eine neue Grafik. Sie bieten darüber hinaus Möglichkeiten, allgemeine Grafikparameter zu steuern.

Aufgabe 1.24	
	Inspizieren Sie mit *help(plot)* die Steuerungsmöglichkeiten der plot-Funktion. Einige Detail-Information zu den Parametern erhalten Sie erst in *help(plot.default)*. Korrigieren Sie Ihren letzten Plot so, dass er eine korrekte Überschrift trägt.

Die "low level"-Funktionen fügen Elemente zu vorhandenen Grafiken hinzu oder modifizieren die Grafik im Detail. Eine sehr hilfreiche Funktion für Kennzeichnungen ist *legend()*.

Weitere Hinweise:[**R D04a**] Ch. 12.

1.5.4. Ergänzung: Funktionen. R-Kommandos können zu Funktionen zusammengefasst werden. Funktionen können parametrisiert sein. Funktionen erlauben eine flexible Wiederverwendbarkeit.

Beispiel für eine Funktion

Beispiel 1.10:

Eingabe

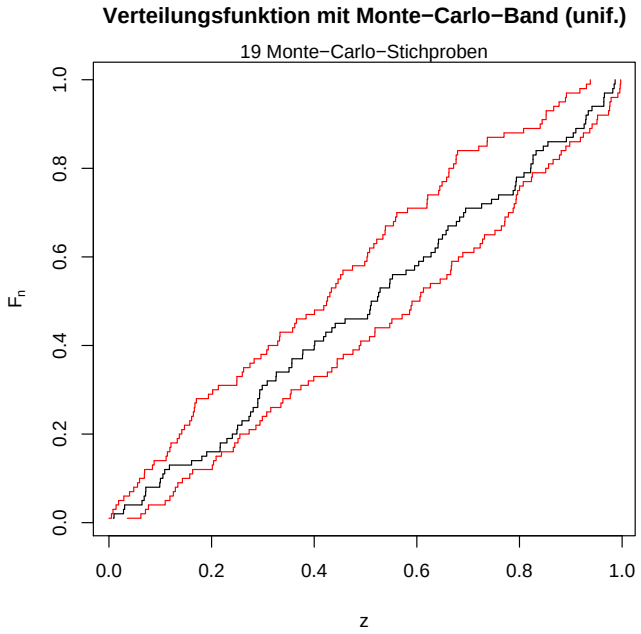
```
ppdemo <- function(x) {
    y <- (1:length(x))/length(x)
    plot(sort(x), y, main = "Verteilungsfunktion mit Monte-Carlo-Band (unif.)",
        xlab = substitute(x), ylab = expression(F[n]),
        type = "s")
    samps <- 19
    mtext(paste(samps, "Monte-Carlo-Stichproben"),
        side = 3)
    samples <- matrix(runif(length(x) * samps), nrow = length(x),
        ncol = samps)
    samples <- apply(samples, 2, sort)
    envelope <- t(apply(samples, 1, range))
    lines(envelope[, 1], y, type = "s", col = "red")
    lines(envelope[, 2], y, type = "s", col = "red")
}
```

Funktionen werden in der Form ⟨`Name`⟩`(`⟨`Aktuelle Parameterliste`⟩`)` aufgerufen. Beispiel:

Beispiel 1.11:

Eingabe

```
z <- runif(100)
ppdemo(z)
```



Wird nur der Name der Funktion eingegeben, so die Definition der Funktion zurück gegeben, d.h. die Funktion wird aufgelistet. Beispiel:

Eingabe

```
ppdemo
```

Ausgabe

```
function (x) {
y<- (1:length(x))/length(x)
plot(sort(x),y, main="Verteilungsfunktion mit Monte-Carlo-Band (unif.)",
xlab=substitute(x), ylab=expression(F[n]), type="s")
samps <- 19        # Anzahl der Simulationen
mtext(paste(samps, "Monte-Carlo-Stichproben"),side=3)
samples <- matrix(runif(length(x)* samps), nrow=length(x), ncol=samps)
samples <- apply(samples,2, sort)
envelope <- t(apply(samples,1, range))
lines(envelope[,1],y,type="s", col="red");
lines(envelope[,2],y,type="s", col="red")
}
```

Aufgabe 1.25	
	Inspizieren Sie *runif()* mit *qqplot()* und *plot()*. Überarbeiten Sie Ihre bisherigen Programmieraufgaben und schreiben Sie die wiederverwendbaren Teile als Funktionen.

Wir haben bei *ppdemo()* die Funktion *mtext()* benutzt, die eine feinere Positionierung von Randbeschriftungen erlaubt.

Parameter bei Funktionen werden dem Wert nach übergeben. Jede Funktion erhält eine Kopie der aktuellen Parameter-Werte. Dies sorgt für eine sichere Programmierumgebung. Auf der anderen Seite führt dies zu eine Speicherbelastung und bringt einen Zeitverlust mit sich. In Situationen, wo der Parameterumfang groß ist oder die Zeit eine kritische Grösse ist, kann dieser Aufwand vermieden werden, indem direkt auf Variable zugegriffen wird, die in der Umgebung der Funktion definiert sind. Entsprechende Techniken sind in [**GI00**] beschrieben.

Während Anweisungen in R schrittweise ausgeführt werden und so die Resultate bei jedem Schritt inspiziert werden können, werden beim Aufruf einer Funktion alle Anweisungen in der Funktion als Einheit ausgeführt. Dies kann eine Fehlersuche schwierig machen. R bietet Möglichkeiten, die Inspektion gezielt auf Funktionen zu ermöglichen. Details dazu finden sich im Anhang A.10 Seite A-19.

1.5.5. Ergänzung: Das Innere von R. Ein typischer Arbeitsabschnitt von R verarbeitet Kommandos in drei Teilschritten:

- *parse()* analysiert einen Eingabetext und wandelt ihn in ein interne Darstellung als R-Ausdruck. R-Ausdrücke sind spezielle R-Objekte.
- *eval()* interpretiert diesen Ausdruck und wertet ihn aus. Das Resultat ist wieder ein R-Objekt.
- *print()* zeigt das resultierende Objekt.

Details sind zu ergänzen:

1.5.5.1. *Parse.* Der erste Schritt besteht aus zwei Teilen: einem Leseprozess, der die Eingabe einscannt und in Bausteine (Tokens) zerlegt, und dem eigentlichen Parsing, das die Bausteine falls möglich zu einem syntaktisch korrekten Ausdruck zusammenfasst. Die Funktion *parse()* fasst beide Schritte zusammen. Dabei kann *parse()* sowohl auf lokalen Dateien arbeiten, als auch auf externen, durch eine URL-Referenz bezeichneten Dateien.

1.5.5.2. *Eval.* Die Funktion *eval()* wertet einen R-Ausdruck aus. Dazu müssen die Referenzen im Ausdruck je nach den aktuellen Umgebungsbedingungen in entsprechende Werte übersetzt werden. Da R ein interpretiertes System ist, können die Umgebungsbedingungen variieren; je nach Umgebung kann derselbe Ausdruck zu unterschiedlichen Resultaten führen.

Jede Funktion definiert eine eigene lokale Umgebung. Funktionen können geschachtelt sein und somit auch die Umgebungen. Die Umgebung kann auch dadurch verändert werden, dass Zusatzpakete für R geladen werden. Die aktuelle unmittelbare Umgebung kann mit *environment()* erfragt werden. Mit *search()* erhält man eine Liste der Umgebungen, die sukzessive durchsucht werden, um Referenzen aufzulösen.

Die Erweiterbarkeit von R bringt die Möglichkeit mit sich, dass Bezeichnungen kollidieren und damit die Übersetzung von Referenzen in aktuelle Werte fraglich wird. Als

Schutz dagegen bietet R 2.x die Möglichkeit, Bezeichnungen in (geschützten) Namensräumen zusammen zu fassen. In den meisten Fällen ist dies transparent für den Benutzer; die Auflösung von Namen folgt der Suchreihenfolge, die durch die Kette der Umgebungen bestimmt ist. Um explizit auf Objekte eines bestimmten Namensraums zuzugreifen, kann dieser mit angegeben werden (z. B. `base::pi` als expliziter Name für die `pi` im Namensraum `base`).

1.5.5.3. *Print.* Die Funktion `print()` ist als ***polymorphe*** Funktion implementiert. Um `print()` auszuführen bestimmt R anhand der Klasse des zu druckenden Objekts eine geeignete Methode. Details folgen später in Abschnitt 2.6.4 Seite 2-23.

1.5.5.4. *Ausführung von Dateien.* Die Funktion `source()` steht bereit, um eine Datei als Eingabe für R zu benutzen. Dabei kann die Datei lokal sein, oder über eine URL-Referenz bezeichnet sein. Konventionell wird für die Namen von R-Kommandodateien die Endung `.R` benutzt.

Die Funktion `Sweave()` erlaubt es, Dokumentation und Kommandos miteinander zu verweben. Konventionell wird für die Namen von `Sweave()`-Eingabedateien die Endung `.Rnw` benutzt.

1.5.6. Ergänzung: Pakete. Funktionen, Beispiele, Datensätze etc. können in R zu Paketen zusammengefasst, die bestimmten Konventionen entsprechen. Die Konventionen unterscheiden sich bei verschiedenen Implementierungen. Als aktuelle Referenz sollten die Konventionen von R 2.x benutzt werden, denen wir auch hier folgen.

Pakete müssen zunächst im R-System installiert werden. In der Regel gibt es dazu betriessystem-spezifische Kommendos. Komfortabler ist jedoch die Installation aus R mit den Funktionen `install.packages()` bzw. `update.packages()`, oder, falls implementiert mit `package.manager()`.

Pakete werden mit `library(pkgname)` geladen. Sie werden wieder frei gegeben mit

```
detach("package:pkgname").
```

Auf die Pakete wird mit Hilfe eines Suchpfades zugegriffen. Details sind implementationsabhängig. Sie sollten sich für Ihre Implementierung über Suchpfade und Konfigurationsparameter informieren.

Pakete sind Verzeichnisse, die den R-Konventionen folgen. In der Regel wird man zunächst als Benutzer vorbereitete Binärpakete installieren. Nur selten muss man auf die Quellpakete anderer Entwickler zurückgreifen. Es lohnt sich jedoch, auch bei der Organisation der eigenen Arbeit den R-Konventionen zu folgen und zusammen gehörende Teile als R-Pakete organisieren. Dann stellt R eine ganze Reihe von Werkzeugen zur Unterstützung bereit. Die Konventionen und die bereitgestellten Werkzeuge sind in [**R D04e**] dokumentiert.

Als Einstieg: Die Funktion
`package.skeleton()` hilft bei der Konstruktion neuer Pakete.

Pakete sollten mindestens die drei Dateien TITLE, INDEX und DESCRIPTION mit bestimmter Information enthalten. Weiteres ist optional.

Name	***Art***	***Inhalt***
TITLE	Datei	Name des Pakets und eine kurze Beschreibung.
INDEX	Datei	Namen und sehr kurze (einzeilige) Beschreibung für jedes wesentliche Objekt des Pakets.
DESCRIPTION	Datei	eine Herkunftsbeschreibung nach Formatkonventionen.
R	Verzeichnis	R code. Dateien in diesem Verzeichnis sollten mit `source()` gelesen werden können. Empfohlene Namensendung: .R .
data	Verzeichnis	Zusätzliche Daten. Dateien in diesem Verzeichnis sollten mit `data()` gelesen werden können. Empfohlene Namensendungen und Formate: .R für R-Code. Alternativ: .r .tab für Tabellen. Alternativ: .txt, .csv .RData für Ausgaben von `save()`.`()` Alternativ: .rda. Das Verzeichnis sollte eine Datei ***00Index*** mit einer Übersicht über die Datensätze enthalten.
exec	Verzeichnis	Zusätzliche ausführbare Dateien, z.B. Perl- oder Shell-Skripte.
inst	Verzeichnis	Wird (rekursiv) in das Zielverzeichnis kopiert.
man	Verzeichnis	Dokumentation im R-Dokumentationsformat (siehe: [**R D04e**] "Writing R extensions", zugänglich über <http://www.cran.r-project.org/>). Empfohlene Namensendung: .Rd
src	Verzeichnis	Fortran, C und andere Quellen.

Aufgabe 1.26	
	Installieren Sie die Funktionen aus der letzten Aufgabe als Bibliothek. Sie können das Paket mit `package.skeleton()` generieren. Überprüfen Sie, ob Sie diese Bibliothek auch nach Neustart wieder mit `library()` laden können. *Hinweis:* ist x ein Objekt, so erzeugt die Funktion `prompt(x)` ein Gerüst, aus dem eine Dokumentation für `x` entwickelt werden kann.

1.6. Statistische Zusammenfassung

Als Leitbeispiel diente in diesem Kapitel die statistische Analyse einer (univariaten) Stichprobe. Dabei haben wir eine in der Statistik zentrale Modellvorstellung benutzt: die Werte der Stichprobe werden als Zufallsvariable aufgefasst, die aus einer zugrundeliegenden theoretischen Verteilung entstammen. Ziel der statistischen Analyse ist der Schluss aus der empirischen Verteilung der Stichprobe auf die unbekannte zu Grunde liegende theoretische Verteilung. Dieser Schluss kann zwei Formen annehmen: wir können die empirische Verteilung mit einer hypothetischen Verteilung vergleichen. Dies ist das Vorgehen der klassischen Statistik. Oder wir können versuchen, aus der empirischen Verteilung Merkmale der zu Grunde liegenden Verteilung zu extrahieren. Dies ist das Vorgehen der Datenanalyse.

Beide Wege sind eng miteinander verwandt. Das wesentliche Werkzeug für beide war hier die Untersuchung der emprischen Verteilungsfunktion.

Literatur

[**R D04e**]R Development Core Team (2000-2005): Writing R extensions.
Siehe: <http://www.r-project.org/manuals.html>.

[**GS77**] Gänssler, P; Stute, W.: Wahrscheinlichkeitstheorie. Heidelberg: Springer 1977.

[**GI00**] Gentleman, R.; Ihaka, R.: Lexical Scope and Statistical Computing. Journal of Computational and Graphical Statistics 9 (2000) 491–508.

KAPITEL 2

Regression

2.1. Allgemeines Regressionsmodell

Aus der Tradition experimenteller Wissenschaften stammt das Paradigma des (kontrollierten) Versuchs. Unter Versuchsbedingungen x wird ein Resultat y gemessen, zusammengesetzt aus einem systematischen Effekt $m(x)$ und einem Messfehler ε.

$$y \;=\; m(x) \;+\; \varepsilon.$$

Dies ist eine ganz spezielle Betrachtungsweise; es wird nicht unvoreingenommen das gemeinsame Verhalten von x und y untersucht, sondern eine Unsymmetrie hineingesteckt: x ist die "Ursache", y (oder eine Veränderung von y) der Effekt. Die "Ursache" x ist in dieser Vorstellung vorgegeben oder direkt kontrollierbar; y ist mittelbar (über die Versuchsbedingungen) beeinflusst. In dieser Vorstellung ist ε ein Messfehler, der durch geeignete Rahmenbedingungen des Versuchs möglichst klein gehalten wird und im Mittel verschwinden sollte: es sollte keinen systematischen Fehler geben.

Vom statistischen Standpunkt ist der wesentliche Unterschied der Rollen von x und y, dass für y das stochastische Verhalten mithilfe von ε modelliert wird, während x als "gegeben" angenommen wird und dafür keine Stochastik im Modell vorgesehen ist.

Um einen formal überschaubaren Rahmen zu bekommen, betrachten wir den Fall, dass x als Vektor von reellen Variablen repräsentiert werden kann, $x \in \mathbb{R}^p$, und dass die Messwerte eindimensionale reelle Werte sind, $y \in \mathbb{R}$. In einem stochastischen Modell kann die oben skizzierte Idee formal gefaßt werden. Eine mögliche Formalisierung ist es, den Messfehler ε als Zufallsvariable zu modellieren. Nehmen wir ferner an, dass der Erwartungswert von ε existiert, so können wir die Annahme, dass der Messfehler im Mittel verschwindet, formalisieren als $E(\varepsilon) = 0$.

Um den systematischen Effekt m zu untersuchen, betrachten wir Messreihen. Der Index $i, i = 1, \ldots, n$, kennzeichnet den Messpunkt, und das Modell ist damit

$$\begin{aligned} & y_i \;=\; m(x_i) \;+\; \varepsilon_i \quad i = 1, \ldots, n \\ \text{mit} \quad & x_i \in \mathbb{R}^p \\ & E(\varepsilon_i) = 0. \end{aligned}$$

Das statistische Problem ist:
schätze die Funktion m aus den Messwerten y_i bei Messbedingung x_i.

Zu diesem Problem der Kurvenschätzung oder "Regression" gibt es eine umfangreiche Literatur in der Statistik. Wir wollen uns hier auf das "computing" konzentrieren. Dazu betrachten wir zunächst eine sehr vereinfachte Version des Regressionsproblems, die lineare Regression. Wesentliche Aspekte lassen sich bereits an diesem Problem illustrieren.

Einer einheitlichen Sprechweise zuliebe nennen wir y_i die ***Respons*** und die Komponenten von x_{ij} mit $j = 1, \ldots, p$ die ***Regressoren***. Die Funktion m heißt die ***Modellfunktion***.

2.2. Lineare Regression

Wir beginnen mit dem Regressionsmodell - jetzt in Vektorschreibweise[1] -

$$\begin{aligned} Y &= m(X) + \varepsilon \quad Y \text{ mit Werten in } \in \mathbb{R}^n \\ \text{mit} \quad & X \in \mathbb{R}^{n\times p} \\ & E(\varepsilon) = 0 \end{aligned}$$

und setzen zusätzlich voraus, dass m linear ist. Dann gibt es (mindestens) einen Vektor $\beta \in \mathbb{R}^p$, so dass

$$m(X) = X\beta$$

und das Regressionsproblem ist jetzt reduziert auf die Aufgabe, β aus der Information (Y, X) zu schätzen.

Das so modifizierte Regressionsmodell heißt ***lineare Regression***.

2.2.1. Kleinste-Quadrate-Schätzung. Eine erste Idee zur Schätzung im linearen Regressionsmodell kann so gewonnen werden: Bei gegebenem X ist $E(Y) = X\beta$, also $X^\top E(Y) = X^\top X\beta$ und damit $(X^\top X)^- X^\top E(Y) = \beta$. Dabei bedeutet $X^\top$ die transponierte Matrix zu X und $(X^\top X)^-$ die (generalisierte) Inverse von $(X^\top X)$. Die Gleichung motiviert das folgende Schätzverfahren:

$$\widehat{\beta} = (X^\top X)^- X^\top Y.$$

Triviale Umkehrung: Bei gegebenem X ist $E(\widehat{\beta}) = \beta$, d.h. $\widehat{\beta}$ ist ein erwartungstreuer Schätzer für β. Ob und wieweit dieser Schätzer neben dieser Konsistenz auch noch statistische Qualitäten hat, wird in Statistik-Vorlesungen diskutiert. Ein Satz zur Charakterisierung dieses Schätzers ist dort als Gauß-Markoff-Theorem bekannt. Wir werden auf diesen Schätzer häufig zurückkommen und geben ihm deshalb einen Namen: ***Gauß-Markoff-Schätzer***. Im Fall eines linearen Modells, wie dem Regressionsmodell, hat dieser Schätzer eine Reihe von Optimalitätseigenschaften. So minimiert dieser Schätzer die mittlere quadratische Abweichung, ist also in diesem Modell ein ***Kleinster-Quadrate-Schätzer***.

Der Schätzer führt unmittelbar zu einer Schätzung $\widehat{m}$ für die Funktion m in unserem ursprünglichen Modell:

$$\widehat{m}(x) = x^\top \cdot \widehat{\beta}.$$

Die Auswertung an unseren Messpunkte x_i ergibt Werte $\widehat{y}_i := \widehat{m}(x_i)$, den ***Fit*** an der Stelle x_i. In Matrixschreibweise, mit aufgelöster Definition

$$\widehat{Y} = X(X^\top X)^- X^\top \cdot Y.$$

Die Matrix $H := X(X^\top X)^- X^\top$ nennt man ***Hut-Matrix***. Sie ist das wesentliche Werkzeug, um den Gauß-Markoff-Schätzer für eine bestimmte Design-Matrix X zu untersuchen (unabhängig von den beobachteten Stichprobenwerten Y). Der Fit hingegen bezieht sich auf eine bestimmte Stichprobe. Die Werte der Zufallsbeobachtung Y unterscheidet sich in der Regel vom Fit $\widehat{Y}$. Die Differenz

$$R_X(Y) := Y - \widehat{Y}$$

heißt ***Residuum***.

[1] Wir wechseln Konventionen und Schreibweisen, wenn es hilfreich ist. Die Verwirrung gehört zu den Konventionen: in einigen Konventionen kennzeichnen Großbuchstaben Zufallsvariable, in anderen Funktionen, in wieder anderen Vektoren. Die Auflösung bleibt jeweils dem Leser überlassen.

Beispiel 2.1:

Eingabe

```
x <- 1:100
err <- rnorm(100)
y <- 2.5 * x + err
lm(y ~ x)
```

Ausgabe

```
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     0.1979       2.4969
```

Beispiel 2.2:

Eingabe

```
summary(lm(y ~ x))
```

Ausgabe

```
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4810 -0.7037 -0.1070  0.7169  2.8859

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.197923   0.216418   0.915    0.363
x           2.496911   0.003721 671.109   <2e-16 ***
---
Signif. codes: 0 *** 0.001 ** 0.01 * 0.05 . 0.1    1

Residual standard error: 1.074 on 98 degrees of freedom
Multiple R-Squared: 0.9998,        Adjusted R-squared: 0.9998
F-statistic: 4.504e+05 on 1 and 98 DF,  p-value: < 2.2e-16
```

Aufgabe 2.1	
	Analysieren Sie die in Beispiel 2.2 (Seite 2-3) gezeigten Ausgaben von *lm()*. Welche Terme können Sie interpretieren? Stellen Sie diese Interpretationen schriftlich zusammen. Für welche Terme fehlt Ihnen noch Information? (Fortsetzung)→

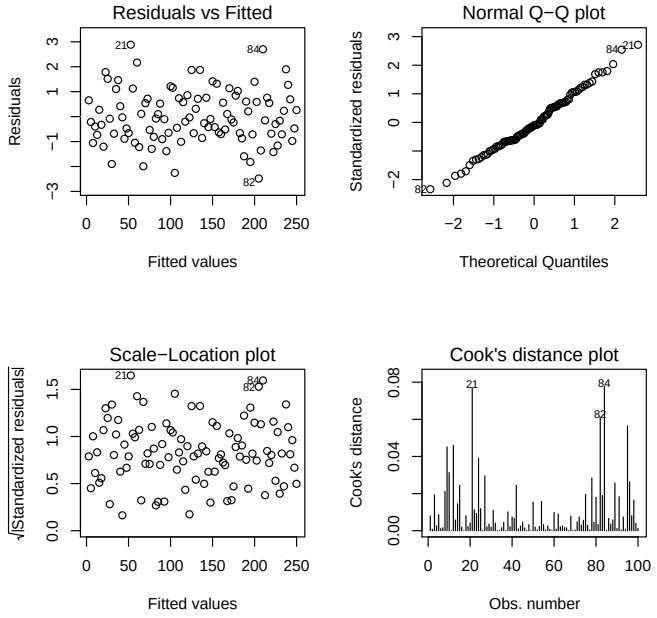
Aufgabe 2.1	(Fortsetzung)
	Erstellen Sie eine kommentierte Version der Ausgabe.
	Konstante Terme müssen explizit ausgeschlossen werden. Die Formel `y~x` wird also interpretiert als $y_i = a + b\ x_i + \varepsilon_i$. Schätzen Sie zum Vergleich mit `lm(y ~ 0 + x)` im Modell $y_i = b\ x_i + \varepsilon_i$.

Die Funktion `lm()` führt nicht nur die Schätzung im linearen Modell durch, sondern liefert eine ganze Reihe von Diagnostiken, die helfen können zu beurteilen, ob die Modellvoraussetzungen vertretbar erscheinen. Eine Darstellung mit `plot()` zeigt vier Aspekte davon. Der obere linke Plot sollte annähernd wie ein Scatterplot von unabhängig normalverteilten Variablen aussehen, die gegen den Fit geplottet sind. Der obere rechte Plot sollte annähernd wie der "normal probabiliy plot" von normalverteilen Variablen aussehen. Die beiden übrigen Plots sind spezielle Diagnostiken für lineare Modelle.

Beispiel 2.3:

Eingabe

```
plot(lm(y ~ x))
```



help(lm)

`lm` *Fitting Linear Models*

Description.

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage.

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments.

`formula`	a symbolic description of the model to be fit. The details of model specification are given below.
`data`	an optional data frame containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. If specified, weighted least squares is used with weights `weights` (that is, minimizing `sum(w*e^2)`); otherwise ordinary least squares is used.
`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The "factory-fresh" default is `na.omit`. Another possible value is `NULL`, no action. Value `na.exclude` can be useful.
`method`	the method to be used; for fitting, currently only `method = "qr"` is supported; `method = "model.frame"` returns the model frame (the same as with `model = TRUE`, see below).
`model, x, y, qr`	logicals. If `TRUE` the corresponding components of the fit (the model frame, the model matrix, the response, the QR decomposition) are returned.
`singular.ok`	logical. If `FALSE` (the default in S but not in R) a singular fit is an error.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`offset`	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. An `offset` term can be included in the formula instead or as well, and if both are specified their sum is used.
`...`	additional arguments to be passed to the low level regression fitting functions (see below).

Details.

Models for `lm` are specified symbolically. A typical model has the form `response ~ terms` where `response` is the (numeric) response vector and `terms` is a series of terms which specifies a linear predictor for `response`. A terms specification of the form `first + second` indicates all the terms in `first` together with all the terms in `second` with duplicates removed. A specification of the form `first:second` indicates the set of terms obtained by taking the interactions of all terms in `first` with all terms in `second`. The

specification `first*second` indicates the *cross* of `first` and `second`. This is the same as `first + second + first:second`.

If `response` is a matrix a linear model is fitted separately by least-squares to each column of the matrix.

See `model.matrix` for some further details. The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a `terms` object as the formula.

A formula has an implied intercept term. To remove this use either `y ~ x - 1` or `y ~ 0 + x`. See `formula` for more details of allowed formulae.

`lm` calls the lower level functions `lm.fit`, etc, see below, for the actual numerical computations. For programming only, you may consider doing likewise.

All of `weights`, `subset` and `offset` are evaluated in the same way as variables in `formula`, that is first in `data` and then in the environment of `formula`.

Value.

`lm` returns an object of `class "lm"` or for multiple responses of class `c("mlm", "lm")`.

The functions `summary` and `anova` are used to obtain and print a summary and analysis of variance table of the results. The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` extract various useful features of the value returned by `lm`.

An object of class `"lm"` is a list containing at least the following components:

`coefficients`	a named vector of coefficients
`residuals`	the residuals, that is response minus fitted values.
`fitted.values`	the fitted mean values.
`rank`	the numeric rank of the fitted linear model.
`weights`	(only for weighted fits) the specified weights.
`df.residual`	the residual degrees of freedom.
`call`	the matched call.
`terms`	the `terms` object used.
`contrasts`	(only where relevant) the contrasts used.
`xlevels`	(only where relevant) a record of the levels of the factors used in fitting.
`offset`	the offset used (missing if none were used).
`y`	if requested, the response used.
`x`	if requested, the model matrix used.
`model`	if requested (the default), the model frame used.

In addition, non-null fits will have components `assign`, `effects` and (unless not requested) `qr` relating to the linear fit, for use by extractor functions such as `summary` and `effects`.

Using time series.

Considerable care is needed when using `lm` with time series.

Unless `na.action = NULL`, the time series attributes are stripped from the variables before the regression is done. (This is necessary as omitting `NA`s would invalidate the time series attributes, and if `NA`s are omitted in the middle of the series the result would no longer be a regular time series.)

Even if the time series attributes are retained, they are not used to line up series, so that the time shift of a lagged or differenced regressor would be ignored. It is good practice to prepare a `data` argument by `ts.intersect(..., dframe = TRUE)`, then apply a suitable `na.action` to that data frame and call `lm` with `na.action = NULL` so that residuals and fitted values are time series.

Note.

Offsets specified by `offset` will not be included in predictions by `predict.lm`, whereas those specified by an offset term in the formula will be.

Author(s).

The design was inspired by the S function of the same name described in Chambers (1992). The implementation of model formula by Ross Ihaka was based on Wilkinson & Rogers (1973).

References.

Chambers, J. M. (1992) *Linear models.* Chapter 4 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Wilkinson, G. N. and Rogers, C. E. (1973) Symbolic descriptions of factorial models for analysis of variance. *Applied Statistics*, **22**, 392–9.

See Also.

`summary.lm` for summaries and `anova.lm` for the ANOVA table; `aov` for a different interface.

The generic functions `coef`, `effects`, `residuals`, `fitted`, `vcov`.

`predict.lm` (via `predict`) for prediction, including confidence and prediction intervals; `confint` for confidence intervals of *parameters*.

`lm.influence` for regression diagnostics, and `glm` for **generalized** linear models.

The underlying low level functions, `lm.fit` for plain, and `lm.wfit` for weighted regression fitting.

Examples.

```
## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2,10,20, labels=c("Ctl","Trt"))
weight <- c(ctl, trt)
anova(lm.D9 <- lm(weight ~ group))
summary(lm.D90 <- lm(weight ~ group - 1))# omitting intercept
summary(resid(lm.D9) - resid(lm.D90)) #- residuals almost identical

opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
plot(lm.D9, las = 1)      # Residuals, Fitted, ...
par(opar)

## model frame :
stopifnot(identical(lm(weight ~ group, method = "model.frame"),
                    model.frame(lm.D9)))
```

Nicht erwähnt in der Help-Information: mit *`first-second`* werden Terme der ersten Gruppe ins Modell aufgenommen, die der zweiten Gruppe aber ausgeschlossen.

Die Hut-Matrix ist eine Besonderheit linearer Modelle. Fit und Residuum jedoch sind allgemeine Konzepte, die bei allen Arten der Schätzung angewandt werden können. Die

Anwender sind oft mit dem Fit (oder der Schätzung) zufrieden. Für den ernsthaften Anwender und für den Statistiker sind die Residuen oft wichtiger: sie weisen darauf hin, was vom Modell oder der Schätzung noch nicht erfaßt ist.

Die Matrix X heißt die ***Design-Matrix*** des Modells. Sie kann die Matrix sein, die mit den ursprünglichen Messbedingungen x_i als Zeilenvektoren gebildet wird. Aber sie ist nicht auf diesen Spezialfall beschränkt. Unter diese scheinbar so einfache Modellklasse lassen sich so viele wichtige Spezialfälle einordnen. Ein paar davon sind im folgenden Abschnitt zusammengestellt.

2.2.2. Beispiele. *Einfache lineare Regression:*

$$y_i \;=\; a + b\; x_i \;+\; \varepsilon_i \qquad \text{mit } x_i \in \mathbb{R}, a, b \in \mathbb{R}$$

kann als lineares Modell mit

$$X \;=\; (1\, x)$$

geschrieben werden, wobei $1 = (1, \ldots, 1)^\top \in \mathbb{R}^n$.

Polynomiale Regression:

$$y_i \quad = \quad a + b_1 x_i \;+\; b_2 x_i^2 \;+ \ldots +\; b_k x_i^k \;+\; \varepsilon_i \qquad \text{mit } x_i \in \mathbb{R}, a, bj \in \mathbb{R}$$

kann als lineares Modell mit

$$X \;=\; (1\, x\, x^2 \ldots x^k)$$

geschrieben werden, wobei $x^j = (x_1^j \ldots x_n^j)^\top$.

Analog für eine Vielzahl von Modellen, die durch andere Transformationen erreicht werden können.

Varianz-Analyse: Einweg-Layout

Gemessen wird unter m Versuchsbedingungen, dabei n_j Messungen unter Versuchsbedingung $j, j = 1, \ldots, m$. Die Messung setze sich additiv zusammen aus einem Grundeffekt μ, einem für die Bedingung j spezifischen Beitrag α_j, und einem Messfehler nach

$$y_{ij} \;=\; \mu + \alpha_j \;+\; \varepsilon_{ij} \qquad \text{mit } \mu, \alpha_j \in \mathbb{R}.$$

Mit $n = \sum n_j$ und

$$X \;=\; (1\, I_1 \ldots I_m),$$

wobei I_j die Indikatorvariablen für die Zugehörigkeit zur Versuchsgruppe j sind, läßt sich dies als kanonisches lineares Modell schreiben.[2].

Covarianz-Analyse

Analog zur Varianz-Analye werden Unterschiede zwischen Gruppen untersucht, aber zusätzliche (linear eingehende) Einflussfaktoren werden korrigierend berücksichtigt. Unter Versuchsbedingung j bei Beobachtung i hängt die Messung zusätzlich von Einflussfaktoren x_{ij} der Versuchseinheit ij ab.

$$y_{ij} \;=\; \mu + \alpha_j + \; bx_{ij} + \varepsilon_{ij} \quad \text{mit } \mu, \alpha_j \in \mathbb{R}.$$

[2]Es ist Konvention, dass bei Varianzanalysen der letzte Index die Beobachtung zählt, und Indizes in alphabetischer Folge vergeben werden. Konventionell werden also im Vergleich zu unserer Notation die Rollen von i und j vertauscht

2.2.3. Modellformeln. R erlaubt es, Modelle auch dadurch zu spezifizieren, dass die Regeln angegeben werden, nach denen die Design-Matrix gebildet wird. Die Syntax, nach denen die Regeln notiert werden, ist sehr kurz in der Beschreibung von `lm()` angegeben. Wir diskutieren sie jetzt etwas ausführlicher. Diese Modell-Spezifikation ist auch für allgemeinere, nicht lineare Modelle möglich. Die Modell-Spezifikationen werden als Attribut mit dem Namen "formula" gespeichert. Sie können mit `formula()` manipuliert werden.

Beispiele

`y~1 + x`	entspricht $y_i = (1\ x_i)(\beta_1\ \beta_2)^\top + \varepsilon$
`y~x`	Kurzschreibweise für $y \sim 1 + x$ (ein konstanter Term wird implizit angenommen)
`y~0 + x`	entspricht $y_i = x_i\beta + \varepsilon$
`log (y) ~ x1 + x2`	entspricht $\log(y_i) = (1\ \ x_{i1}\ \ x_{i2})(\beta_1\ \ \beta_2\ \ \beta_3)^\top + \varepsilon$ (ein konstanter Term wird implizit angenommen)
`lm(y ~poly(x,4), data = experiment)`	analysiert den Datensatz "experiment" mit einem linearen Modell für polynomiale Regression vom Grade 4 in x.

Eine Übersicht über alle Operatoren zur Modellspezifikation ist im Anhang A.44 (Seite A-31) zu finden.

Aufgabe 2.2	
	Schreiben Sie die vier oben genannten Modelle als R-Modellformeln.
	Erzeugen Sie sich für jedes dieser Modelle ein Beispiel durch Simulation und wenden Sie `lm()` auf diese Beispiele an. Vergleichen Sie die durch `lm()` geschätzten Parameter mit den Parametern, die Sie in der Simulation benutzt haben.

Die Modellformel wird in einem Eintrag im Resultat von `lm()` gespeichert. Sie kann also aus dem Resultat zurück gewonen werden. Anhand der Formel-Notation generiert R implizit eine Design-Matrix. Mit `model.matrix()` kann diese Design-Matrix inspiziert werden.

2.2.4. Faktoren. Mit Hilfe der Notation zur Design- und Modellbeschreibung kann die Übersetzung zwischen einer fallorientierten Beschreibung eines Designs in eine Design-Matrix für ein lineares Modell in kanonischer Form automatisch geschehen. Bisweilen braucht die Übersetzung etwas Nachhilfe. Betrachten Sie z.B. einen Datensatz

```
y <- c( 1.1, 1.2, 2.4, 2.3, 1.8, 1.9)
x <- c( 1, 1, 2, 2, 3, 3).
```

Der Vektor x kann als quantitativer Vektor für das Regressions-Modell

$$y_i \ = \ a + b\, x_i \ + \ \varepsilon_i$$

als Regressor gemeint sein, oder es kann im Modell der Einweg-Varianzanalyse

$$y_{i\,x} \ = \ \mu + \alpha_x \ + \ \varepsilon_{i\,x}$$

die Kennzeichnung einer Behandlungsgruppe sein. Um beide Möglichkeiten zu unterscheiden, können Vektoren in R als ***Faktoren*** definiert werden. Vektoren, die keine Faktoren

sind, werden als quantitative Variable behandelt wie im ersten Beispiel. Faktoren werden als Kennzeichner behandelt und in der Design-Matrix in entsprechende Indikatorvariable übersetzt. So ergibt

```
lm(y ~ x)
```

das Regressionsmodell, jedoch

```
lm(y ~ factor(x))
```

das Varianz-Modell für das Einweg-Layout.

Durch einen Parameter *ordered=TRUE* kann beim Aufruf der Funktion *factor()* die erzeugte Variable als geordnet gekennzeichnet werden. Die erzeugte Variable wird dann bei den Auswertungen als ordinal skaliert behandelt.

```
lm(y ~ factor(x, ordered = TRUE))
```

Ohne diese Kennzeichnung werden Faktoren als kategoriell skaliert betrachtet.

Die Zahlenwerte von Faktoren brauchen keine aufsteigende Folge zu sein. Sie werden (auch für ordinale Faktoren) als bloße Namen benutzt und durch eine laufende Nummer ersetzt. So ergibt

```
factor(c( 2, 2, 5, 5, 4, 4))
```

einen Vektor mit drei Faktorwerten 1, 2, 3, die die Namen "2", "5" und "4" haben. Faktoren können auch durch Namen bezeichnet werden, z.B.

```
lm(y ~ factor ( c("Beh1", "Beh1", "Beh2", "Beh2", "Beh3", "Beh3") )
```

Die unterschiedlichen Werte eines Faktors nennt man ***Stufen*** des Faktors. Sie können mit *levels()* erfragt werden, z.B.

```
levels(factor( c( 2, 2, 5, 5, 4, 4) ))
levels(factor( c("Beh1", "Beh1", "Beh2", "Beh2", "Beh3", "Beh3") ))
```

Aufgabe 2.3	
	Generieren Sie drei Vektoren mit je 10 $N(\mu_j, 1)$-verteilten Zufallsvariablen $\mu_j = j,\ j = 1, 3, 9$. Verketten Sie diese zu einem Vektor y. Generieren Sie sich einen Vektor x aus je 10 wiederholten Werten $j,\ j = 1, 3, 9$. Berechnen Sie die Gauß-Markoff-Schätzer in den linearen Modellen `y~x` und `y~factor(x)`. Lassen Sie sich das Resultat jeweils als Tabelle mit *summary()* und als Grafik mit *plot()* anzeigen und vergleichen Sie die Resultate.

2.2.5. Gauß-Markoff-Schätzer und Residuen. Wir werfen nun einen genaueren Blick auf den Gauß-Markoff-Schätzer. Kenntnisse aus der linearen Algebra, langes Nachdenken oder andere Quellen sagen uns:

Bemerkung 2.1.

(1) Die Design-Matrix X definiert eine Abbildung $\mathbb{R}^p \to \mathbb{R}^n$
$\beta \mapsto X\beta$.
Der Bild-Raum dieser Abbildung sei $\mathscr{M}_X$, $\mathscr{M}_X \subset \mathbb{R}^n$.
(2) Sind die Modell-Annahmen erfüllt, so ist $E(Y) \in \mathscr{M}_X$.
(3) $\widehat{Y} = \pi_{\mathscr{M}_X}(Y)$, wobei $\pi_{\mathscr{M}_X} : \mathbb{R}^n \to \mathscr{M}_X$
die (euklidische) Orthogonalprojektion ist.
(4) $\widehat{\beta} = argmin_\beta |Y - \widehat{Y}_\beta|^2$ wobei $\widehat{Y}_\beta = X\beta$.

Die Charakterisierung (3) des Gauß-Markoffschätzers als Orthogonalprojektion hilft für das Verständnis oft weiter: der Fit ist die Orthogonalprojektion des Beobachtungsvektors auf den Erwartungswertraum des Modells (und minimiert damit den quadratischen Abstand). Das Residuum ist das orthogonale Komplement. In der Statistik ist die Charakterisierung als Orthogonalprojektion auch ein Ausgangspunkt, um den Schätzer systematisch zu analyisieren.

In einfachen Fällen helfen Kenntnisse aus der Wahrscheinlichkeitstheorie schon weiter, etwa zusammengefasst im folgenden Satz:

THEOREM 2.2. *Sei Z eine Zufallsvariable mit Werten in $\mathbb{R}^n$, die nach $N(0, \sigma^2 I)$ verteilt ist und sei $\mathbb{R}^n = L_0 \oplus \ldots \oplus L_r$ eine Orthogonalzerlegung. Sei $\pi_i = \pi L_i$ die Orthogonalprojektion auf L_i, $i = 0, \ldots, r$.*
Dann gilt

(i) $\pi_0(Z), \ldots, \pi_r(Z)$ sind unabhängige Zufallsvariablen.
(ii) $|\pi_i(Z)|^2 \sim \sigma^2 \chi^2(\dim L_i)$ für $i = 0, \ldots, r$.

BEWEIS. → Wahrscheinlichkeitstheorie. Siehe z.B. [Jørgensen 1993, 2.5 Theorem 3]. □

Mit $\varepsilon = Y - X\beta$ können daraus theoretische Verteilungsaussagen für Schätzer $\widehat{\beta}$ und Residuen $Y - \widehat{Y}$ abgeleitet werden.

Aufgabe 2.4	
	Welche Verteilung hat $\lvert R_X(Y)\rvert^2$, wenn ε nach $N(0, \sigma^2 I)$ verteilt ist?

Auf den ersten Blick ist $|R_X(Y)|^2 = |Y - \widehat{Y}|^2$ ein geeignetes Maß, um die Qualität eines Modells zu beurteilen: kleine Werte sprechen für den Fit, große Werte zeigen, dass der Fit schlecht ist. Dies ist jedoch mit Sorgfalt zu betrachten. Zum einen hängt diese Größe von linearen Skalenfaktoren ab. Zum anderen muss die Dimensionen der jeweiligen Räume mit in Betracht gezogen werden. Was passiert, wenn weitere Regressoren ins Modell aufgenommen werden? Wir haben z.B. gesehen, dass "linear" auch die Möglichkeit gibt, nichtlineare Beziehungen zu modellieren, zum Beispiel dadurch, dass geeignet transformierte Variable in die Design-Matrix mit aufgenommen werden. Die Charakterisierung (3) aus Bemerkung 2.1 sagt uns, dass effektiv nur der von der Design-Matrix aufgespannte Raum relevant ist. Hier sind die Grenzen des Gauß-Markoff-Schätzers im linearen Modell erkennbar: wenn viele transformierte Variablen aufgenommen werden, oder generell wenn der durch die Design-Matrix bestimmte Bildraum zu groß wird, gibt es eine Überanpassung. Im Extrem ist $\widehat{Y} = Y$. Damit werden alle Residuen zu null, aber die Schätzung ist nicht brauchbar.

Wir benutzen $|R_X(Y)|^2 / \dim(L_X)$, wobei L_X das orthogonale Komplement von $\mathscr{M}_X$ in $\mathbb{R}^n$ ist (also $\dim(L_X) = n - \dim(\mathscr{M}_X)$), um die Dimensionsabhängigkeit zu kompensieren.

Aufgabe 2.5	
	Modifizieren Sie die Plot-Ausgabe `plot.lm()` für die linearen Modelle so, dass anstelle des Tukey-Anscombe-Plots die studentisierten Residuen gegen den Fit aufgetragen werden.
$*$	Ergänzen Sie den *QQ*-Plot durch Monte-Carlo-Bänder für unabhängige Gauß'sche Fehler. *Hinweis:* Sie können die Bänder nicht direkt aus der Gaußverteilung generieren - Sie brauchen die Residuenverteilung, nicht die Fehlerverteilung.

Aufgabe 2.6	
	Schreiben Sie eine Prozedur, die für die einfache lineare Regression $y_i = a + bx_i + \varepsilon_i \quad$ mit $x_i \in \mathbb{R}, a, b \in \mathbb{R}$ den Gauß-Markoff-Schätzer berechnet und vier Plots darstellt: • Respons gegen Regressor, mit geschätzter Geraden • studentisierte Residuen gegen Fit • Verteilungsfunktion der studentisierten Residuen im QQ-Plot mit Bändern • Histogramm der studentisierten Residuen

2.3. Streuungszerlegung und Varianzanalyse

Die Interpretation des Gauß-Markoff-Schätzers als Orthogonalprojektion (Bem. 2.1 3) zeigt eine Möglichkeit, Modelle zu vergleichen: Für X, X' Design-Matrizen mit $\mathscr{M}_{X'} \subset \mathscr{M}_X$, betrachten wir die Zerlegung $\mathbb{R}^n = L_0 \oplus \ldots \oplus L_r$ mit $L_0 := \mathscr{M}_{X'}$, und die orthogonale Komplemente $L_1 := \mathscr{M}_X \ominus \mathscr{M}_{X'}, L_2 := \mathbb{R}^n \ominus \mathscr{M}_X$. Wieder bezeichnet π_i jeweils die entsprechende Projektion auf L_i

$$F := \frac{\frac{1}{\dim(L_1)}|\pi_1|^2}{\frac{1}{\dim(L_2)}|\pi_2|^2}.$$

Diese Statistik, die F-Statistik (nach R.A. Fisher) ist die Basis für die ***Varianzanalyse***, einer klassischen Strategie, Modelle zu vergleichen. ***Streuungszerlegung*** ist ein anderer Name für diesen Ansatz.

Die Idee wird auf Ketten von Modellen verallgemeinert. Ist $\mathscr{M}_0 \subset \ldots \subset \mathscr{M}_r = \mathbb{R}^n$, so liefert $L_0 := \mathscr{M}_0, L_i := \mathscr{M}_i \ominus \mathscr{M}_{i-1}$ für $i = 1..r$ eine Orthogonalzerlegung. Mit den Bezeichnungen von oben ist dann

$$\frac{\frac{1}{\dim L_{i-1}}|\pi_{i-1}|^2}{\frac{1}{\dim L_i}|\pi_i|^2}$$

eine Teststatistik, die zum Test für das Modell $\mathscr{M}_{i-1}$ im Vergleich zum Obermodell $\mathscr{M}_i$ herangezogen wird.

Aufgabe 2.7	
	Welche Verteilung hat F, wenn $E(Y) \in \mathscr{M}_{X'}$ gilt und ε nach $N(0, \sigma^2 I)$ verteilt ist?

Aufgabe 2.8	
	Geben Sie eine explizite Formel für die F-Statistik zur Varianzanalyse im Einweg-Layout $y_{ij} = \mu + \alpha_j + \varepsilon_{ij}$ im Vergleich zum homogenen Modell $y_{ij} = \mu + \varepsilon_{ij}$.

Aufgabe 2.9	
	Analysieren Sie die in Beispiel 2.2 (Seite 2-3) gezeigten Ausgaben von `lm()`. Welche Terme können Sie jetzt interpretieren? Stellen Sie diese Interpretationen schriftlich zusammen. Für welche Terme fehlt Ihnen noch Information?

help(anova)

`anova` *Anova Tables*

Description.

Compute analysis of variance (or deviance) tables for one or more fitted model objects.

Usage.

```
anova(object, ...)
```

Arguments.

`object`	an object containing the results returned by a model fitting function (e.g., `lm` or `glm`).
`...`	additional objects of the same type.

Value.

This (generic) function returns an object of class `anova`. These objects represent analysis-of-variance and analysis-of-deviance tables. When given a single argument it produces a table which tests whether the model terms are significant.

When given a sequence of objects, `anova` tests the models against one another in the order specified.

The print method for `anova` objects prints tables in a "pretty" form.

Warning.

The comparison between two or more models will only be valid if they are fitted to the same dataset. This may be a problem if there are missing values and R's default of `na.action = na.omit` is used.

References.

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*, Wadsworth & Brooks/Cole.

See Also.

`coefficients`, `effects`, `fitted.values`, `residuals`, `summary`, `drop1`, `add1`.

Modelle für die Varianzanalyse können als Regeln angegeben werden. Dieselbe Syntax zur Modelbeschreibung wird benutzt wie schon bei der Regression. Wenn Terme auf der rechten Seite der Modelbeschreibung Faktoren sind, wird automatisch ein Varianzanalyse-Modell anstelle eines Regressionsmodells generiert. Wichtige Spezialfälle sind

`y~A`	Einweg-Varianzanalyse mit Faktor A,
`y~A + x`	Covarianzanalyse mit Faktor A und Covariable x,
`y~A * B`	Zwei-Faktor-Kreuz-Layout mit Faktoren A und B,
`y~A/B`	Zwei-Faktor hierarchisches Layout mit Faktor A und Subfaktor B.

Die Modellbeschreibung bestimmt die linearen Räume, in denen die Erwartungswerte liegen. Die Streuungszerlegungen sind dadurch jedoch nicht eindeutig bestimmt: die Angabe der Räume lässt evtl. noch verschiedene Orthogonalzerlegungen zu (z.B. abhängig von der Reihenfolge). Mehr noch: die Angabe der Faktoren bestimmt ein Erzeugendensystem der Räume. Die Fakoren brauchen nicht orthogonal zu sein, noch nicht einmal unabhängig. Die Einweg-Varianzanalyse in Koordinatendarstellung illustriert dieses Problem: mit

$$y_{ij} \;=\; \mu + \alpha_j \;+\; \varepsilon_{ij} \quad \text{mit } \mu, \alpha_j \in \mathbb{R}$$

ist für $n_j > 0$ die Zerlegung in μ und α_j nicht eindeutig. Der tieferliegende Grund ist: der globale Faktor μ definiert den vom Einheitsvektor 1 aufgespannten Raum, und dieser liegt in dem von den Gruppenindikatoren aufgespannten Raum.

Die Modellformel definiert eine Designmatrix X und damit einen Modellraum. Eine zusätzliche Matrix C wird benutzt, um die Matrix zu reduzieren und damit eine eindeutige Streuungszerlegung zu spezifizieren. Die effektive Designmatrix ist dann $[1 \;\; X \; C]; C$ heißt ***Kontrastmatrix***. Die Funktionen zur Varianzanalyse erlauben es, die Kontraste zu spezifizieren.

Die Funktion `anova()` operiert wie eine spezielle Formatierung der Ausgabe und wird analog `summary()` benutzt, also z.B. in der Form `anova(lm())`.

Aufgabe 2.10	
	Die Datei “micronuclei” enthält einen Datensatz aus einem Mutagenitätstest. Zellkulturen (je 50 Einheiten) wurden in einer Kontrollgruppe und unter 5 chemischen Behandlungen beobachtet. Der Effekt der Substanzen ist, die Chromosomen aufzubrechen und Mikronuklei zu bilden. Registriert wurde die Größe der Mikronuklei (relativ zum Eltern-Nukleus). Lesen Sie die Datei “micronuclei” und berechnen Sie für jede Gruppe Mittelwert und Varianz. Hinweis: Sie können die Datei mit `data()` einlesen. Für Dateien mit Tabellenformat gibt es die spezielle Anweisung `read.table()`. Informieren Sie sich mit `help()` über beide Funktionen. Einige ausgewählte statistische Funktionen (z.B. Mittelwert) finden Sie in Tabelle A.18 im Anhang.
	Vergleichen Sie die Resultate. Sind Behandlungseffekte nachweisbar? Hinweise: Versuchen Sie zunächst, die Aufgabe als Einweg-Varianzanalyse zu formulieren. Den Datensatz müssen Sie zunächst z.B. mit Hilfe von `c()` auf eine geeignete Form bringen.

Aufgabe 2.11	
*	Schreiben Sie eine Funktion `oneway()`, die als Argument eine Datentabelle nimmt und eine Einweg-Varianzanalyse als Test auf die Differenz zwischen den Spalten durchführt.
*	Ergänzen Sie `oneway()` durch die notwendigen diagnostischen Plots. Welche Diagnostiken sind notwendig?

Aufgabe 2.12	
	Das Industrieunternehmen Kiwi-Hopp [3] möchte einen neuen Hubschrauber auf den Markt bringen. Die Hubschrauber müssen also danach beurteit werden, wie lange sie sich in der Luft halten, bis sie aus einer gegebenen Höhe (ca. 2m) den Boden erreichen[4]. Eine Konstruktionszeichnung ist unten (Abbildung 2.1, Seite 2-26) angegeben. Welche Faktoren könnten die Variabilität der Flug(Sink)zeiten beeinflussen? Welche Faktoren könnten die mittlere Flugzeit beeinflussen?
	Führen Sie 30 Versuchsflüge mit einem Prototyp durch und messen Sie die Zeit in 1/100s. (Sie müssen vielleicht zusammenarbeiten, um die Messungen durchzuführen.) Würden Sie die gemessene Zeit als normalverteilt ansehen? Die Anforderung ist, dass die mittlere Flugdauer mindestens 2.4s erreicht. Erfüllt der Prototyp diese Anforderung?
	(Fortsetzung)→

[3] Nach einer Idee von Alan Lee, Univ. Auckland, Neuseeland

[4] Kiwis können nicht fliegen.

Aufgabe 2.12	(Fortsetzung)
	Sie haben die Aufgabe, einen Entwurf für die Produktion auszusuchen. Folgende Varianten stehen zur Diskussion: Rotorbreite 45mm Rotorbreite 35mm Rotorbreite 45mm mit Zusatzfalte als Stabilisierung Rotorbreite 35mm mit Zusatzfalte als Stabilisierung. Ihr Haushalt erlaubt ca. 40 Testflüge. Bauen Sie 4 Prototypen und führen sie je 10 Testflüge durch, bei denen Sie die Zeit messen. Finden Sie diejenige Konstruktion, die die längste Flugdauer ergibt. Erstellen Sie einen Bericht. Der Bericht sollte folgende Details enthalten: • eine Liste der erhobenen Daten und eine Beschreibung des experimentellen Vorgehens. • geeignete Plots für jede Konstruktion • eine Varianz-Analyse • eine klare Zusammenfassung Ihrer Schlüsse. *Weitere Hinweise:* Randomisieren Sie die Reihenfolge Ihrer Experimente. Reduzieren Sie die Variation, indem Sie gleichmässige Bedingungen für das Experiment schaffen (gleiche Höhe, gleiche Abwurftechnik etc.).
	Die Zusatzfaltung verursacht zusätzliche Arbeitskosten. Schätzen sie den Effekt ab, den diese Zusatzinvestition bringt.

Aufgabe 2.13	
	Benutzen Sie den Quantil-Quantil-Plot, um paarweise die Resultate des Helikopter-Experiments aus dem letzten Kapitel zu vergleichen. Formulieren Sie die Resultate.

Aufgabe 2.14	
	Inspizieren Sie die Implementierung von `qqnorm()`. Programmieren Sie eine analoge Funktion für den PP-Plot und wenden Sie diese auf die Helikopter-Daten an.

2.4. Simultane Schätzung und Kontraste

Der Kleinste-Quadrate-Schätzer schätzt im Prinzip alle Komponenten des Parameter-Vektors simultan. Die Optimalitäts-Aussagen des Gauß-Markoff-Theorems beziehen sich nur auf eindimensionale lineare Statistiken. Die Konfindenzaussagen gelten jedoch multivariat. Es gilt: Der mithilfe der F-Verteilung gewonnene Konfidenzbereich zum Konfidenzniveau $1-\alpha$ hat die Form

$$\{\widehat{\overline{\beta}} \in \mathbb{R}^k : (\sum_{j=1}^{k} (\widehat{\overline{\beta}}_j - \beta_j)^2 \|x_j\|^2/k)/\widehat{\sigma^2} \leq F_{1-\alpha}(k, n-k)\},$$

d.h. der Konfidenzbereich ist eine Elllipse. Wir können die Ellipse auch als den Bereich definieren, der durch alle Tangenten der Ellipse begrenzt wird. Dies übersetzt die (eine) quadratischen Bedingung an die Punkte im Konfidenzbereich durch (unendlich viele) lineare Bedingungen. Diese geometrische Beziehung ist der Kern für den folgenden Satz:

THEOREM 2.3. *Sei $\mathscr{L} \subset \mathbb{R}^k$ ein linearer Unterraum der Dimension d; $EY = Xb$ mit $rk(X) = p < n$. Dann ist*

$$P\{\ell^t\beta \in \ell^t\widehat{\overline{\beta}} \pm (dF^{\alpha}_{d,n-\alpha})^{1/2} s(\ell^t(X^tX)^{-1}\ell)^{1/2} \; \forall \ell \in \mathscr{L}\} = (1-\alpha).$$

BEWEIS. [**Mil81**, 2.2, p. 48] □

Dies ist ein simultaner Konfidenzbereich für alle Linearkombinationen aus $\mathscr{L}$. Als Test übersetzt ergibt dies einen simultanen Test für alle linearen Hypothesen aus $\mathscr{L}$. Im Falle $d = 1$ reduziert sich dieser Scheffé-Test auf den üblichen F-Test.

Geometrisch ist das Konfidenz-Ellipsoid also durch seine (unendlich vielen) Tangentialebenen gekennzeichnet. Übersetzt als Test werden hier also unendlich viele lineare Tests simultan durchgeführt. In vielen Anwendungen ist es jedoch möglich, gezieltere Fragestellungen anzugehen, etwa im Zwei-Stichprobenfall nur die Hypothese $\beta_1 - \beta_2 = 0$. Diese reduzierten Fragestellungen können in linearen Modellen formuliert werden und zu schärferen Tests führen. Dies geschieht durch die Spezifizierung von ***Kontrasten*** und wird in R auch für die Varianzanalyse unterstützt.

2.5. Nichtparametrische Regression

2.5.1. Zwischenspiel: verallgemeinerte lineare Modelle. Wir wollen schnell zur praktischen Arbeit kommen. An dieser Stelle sollte jedoch eine Ausblick nicht fehlen, wie wir über die einschränkenden Annahmen des linearen Modells hinauskommen. Die linearen Modelle gehören zu den am besten untersuchten Modellen. Theorie und Algorithmen hierfür sind weit entwickelt. Von daher ist es naheliegend, zu probieren, wieweit sich die Modellklasse so erweitern läßt, dass theoretische und algorihmische Erfahrungen noch nutzbar sind.

Wir notierten das lineare Modell als

$$\begin{aligned} Y &= m(X) + \varepsilon \quad Y \in \mathbb{R}^n \\ & X \in \mathbb{R}^{n\times p} \\ & E(\varepsilon) = 0 \\ & m(X) = X\beta \quad \beta \in \mathbb{R}^p. \end{aligned}$$

Eine wichtige Erweiterung ist, die Bedingung der Linearität aufzuheben. Sie wird abgemildert mit einer Zwischenstufe. Wir setzen also nicht mehr voraus, dass m linear ist, sondern nur, dass es sich über eine lineare Funktion faktorisieren läßt. Dies ergibt ein verallgemeinertes lineares Modell

$$\begin{aligned} Y &= m(X) + \varepsilon \quad Y \in \mathbb{R}^n \\ & X \in \mathbb{R}^{n\times p} \\ & E(\varepsilon) = 0 \\ & m(X) = \overline{m}(\eta) \text{ mit } \eta = X\beta,\ \beta \in \mathbb{R}^p. \end{aligned}$$

Die nächste naheliegende Verallgemeinerung ist, eine Transformation für Y zu berücksichtigen. Zahlreiche weitere Abschwächungen sind diskutiert worden; eine kleine Anzahl

hat sich als handhabbar erwiesen. Diese verbliebene Modellgruppe wird als generalisierte lineare Modelle bezeichnet. Generalisierte lineare Modelle haben in R eine weitgehende Unterstützung. In der Regel findet sich zu den hier diskutierten R-Funktionen für lineare Modelle eine Entsprechung für generalisierte lineare Modelle. Weitere Information mit `help(glm)`.

2.5.2. Lokale Regression. Wir machen nun einen großen Sprung. Wir haben lineare Modelle diskutiert. Wir wissen, dass damit auch nichtlineare Funktionen modelliert werden können. Aber die Terme, die in die Funktion eingehen, müssen vorab spezifiziert werden. Zu viele Terme führen zu einer Überanpassung. Die statistische Behandlung von Regressionsproblemen mit geringen Einschränkungen an die Modellfunktion bleibt ein Problem.

Ein partieller Lösungsansatz kommt aus der Analysis. Dort ist es eine Standard-Technik, Funktionen lokal zu approximieren. Das analoge Vorgehen in der Statistik ist, anstelle eines globalen Schätzverfahrens eine lokalisierte Variante zu wählen. Wir nehmen immer noch an, dass

$$\begin{aligned} Y \quad = \quad & m(X) \; + \; \varepsilon \quad Y \; \in \mathbb{R}^n \\ & X \; \in \; \mathbb{R}^{n \times p} \\ & E(\varepsilon) = 0, \end{aligned}$$

aber wir nehmen Linearität nur lokal an:

$$m(x) \approx x' \beta_{x_0} \quad \beta_{x_0} \; \in \; \mathbb{R}^p \text{ und } x \approx x_0.$$

Wenn wir praktisch arbeiten wollen, reicht abstrakte Asymptotik nicht. Wir müssen das $\approx$ spezifizieren. Dies kann skalenspezifisch geschehen (z.B. $x \approx x_0$ wenn $|x - x_0| < 3$) oder designabhängig (z.B. $x \approx x_0$ wenn $\#i : |x - x_i| \leq |x - x_0| < n/3$). Die heute üblichen Implementierungen haben feinere Varianten, die hier noch nicht diskutiert werden können. Der Illustration halber kann die folgende Vergröberung reichen:

Lokalisierter Gauß-Markoff-Schätzer:

Für $x \in \mathbb{R}^p$, bestimme

$$\delta = \min_d : (\#i : |x - x_i| \leq d) \geq n \cdot f$$

wobei f ein gewählter Anteil (z.B. 0.5) ist.

Bestimme den Gauß-Markoff-Schätzer $\widehat{\beta}_x$, wobei nur diejenigen Beobachtungen einbezogen werden, für die $|x - x_i| \leq \delta$.
Schätze

$$\widehat{m}(x) = x' \widehat{\beta}_x.$$

Diese Vergröberung ignoriert alle Messpunkte, die einen Abstand über δ haben. Feinere Methoden benutzen eine Gewichtung, um den Einfluss entfernter Messpunkte zunehmend zu reduzieren.

---

help(loess)

loess *Local Polynomial Regression Fitting*

Description.

Fit a polynomial surface determined by one or more numerical predictors, using local fitting.

Usage.

```
loess(formula, data, weights, subset, na.action, model = FALSE,
      span = 0.75, enp.target, degree = 2,
      parametric = FALSE, drop.square = FALSE, normalize = TRUE,
      family = c("gaussian", "symmetric"),
      method = c("loess", "model.frame"),
      control = loess.control(...), ...)
```

Arguments.

`formula`	a formula specifying the numeric response and one to four numeric predictors (best specified via an interaction, but can also be specified additively).
`data`	an optional data frame within which to look first for the response, predictors and weights.
`weights`	optional weights for each case.
`subset`	an optional specification of a subset of the data to be used.
`na.action`	the action to be taken with missing values in the response or predictors. The default is given by `getOption("na.action")`.
`model`	should the model frame be returned?
`span`	the parameter α which controls the degree of smoothing.
`enp.target`	an alternative way to specify `span`, as the approximate equivalent number of parameters to be used.
`degree`	the degree of the polynomials to be used, up to 2.
`parametric`	should any terms be fitted globally rather than locally? Terms can be specified by name, number or as a logical vector of the same length as the number of predictors.
`drop.square`	for fits with more than one predictor and `degree=2`, should the quadratic term (and cross-terms) be dropped for particular predictors? Terms are specified in the same way as for `parametric`.
`normalize`	should the predictors be normalized to a common scale if there is more than one? The normalization used is to set the 10% trimmed standard deviation to one. Set to false for spatial coordinate predictors and others know to be a common scale.
`family`	if `"gaussian"` fitting is by least-squares, and if `ßymmetric"` a redescending M estimator is used with Tukey's biweight function.
`method`	fit the model or just extract the model frame.
`control`	control parameters: see `loess.control`.
`...`	control parameters can also be supplied directly.

Details.

Fitting is done locally. That is, for the fit at point x, the fit is made using points in a neighbourhood of x, weighted by their distance from x (with differences in 'parametric'

variables being ignored when computing the distance). The size of the neighbourhood is controlled by α (set by `span` or `enp.target`). For $\alpha < 1$, the neighbourhood includes proportion α of the points, and these have tricubic weighting (proportional to $(1 - (\text{dist}/\text{maxdist})^3)^3$. For $\alpha > 1$, all points are used, with the 'maximum distance' assumed to be $\alpha^{1/p}$ times the actual maximum distance for p explanatory variables.

For the default family, fitting is by (weighted) least squares. For `family=ßymmetric"` a few iterations of an M-estimation procedure with Tukey's biweight are used. Be aware that as the initial value is the least-squares fit, this need not be a very resistant fit.

It can be important to tune the control list to achieve acceptable speed. See `loess.control` for details.

Value.

An object of class `"loess"`.

Note.

As this is based on the `cloess` package available at `netlib`, it is similar to but not identical to the `loess` function of S. In particular, conditioning is not implemented.

The memory usage of this implementation of `loess` is roughly quadratic in the number of points, with 1000 points taking about 10Mb.

Author(s).

B.D. Ripley, based on the `cloess` package of Cleveland, Grosse and Shyu avaliable at `http://www.netlib.org/a/`.

References.

W.S. Cleveland, E. Grosse and W.M. Shyu (1992) Local regression models. Chapter 8 of *Statistical Models in S* eds J.M. Chambers and T.J. Hastie, Wadsworth & Brooks/Cole.

See Also.

`loess.control`, `predict.loess`.

`lowess`, the ancestor of `loess` (with different defaults!).

Examples.

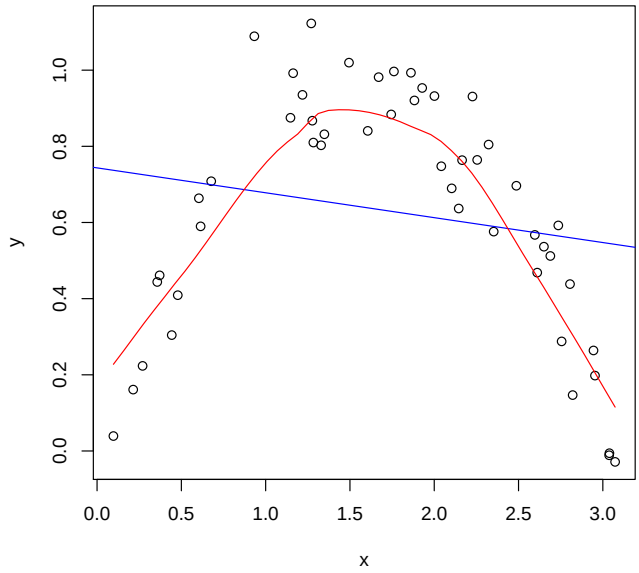
```
cars.lo <- loess(dist ~ speed, cars)
predict(cars.lo, data.frame(speed = seq(5, 30, 1)), se = TRUE)
# to allow extrapolation
cars.lo2 <- loess(dist ~ speed, cars,
  control = loess.control(surface = "direct"))
predict(cars.lo2, data.frame(speed = seq(5, 30, 1)), se = TRUE)
```

Während die lineare Regression durch die Modellannahmen verpflichtet ist, immer ein lineares (oder linear parametrisiertes) Bild zu geben, kann bei einer lokalisierten Variante auch Nichtlinearitäten gefolgt werden. Die Untersuchung dieser Familie von Verfahren bildet ein eigenes Teilgebiet der Statistik, die nichtparametrische Regression.

Beispiel 2.4:

Eingabe

```
x <- runif(50) * pi
y <- sin(x) + rnorm(50)/10
plot(x, y)
abline(lm(y ~ x), col = "blue")
lines(loess.smooth(x, y), col = "red")
```



2.6. Ergänzungen

2.6.1. Ergänzung: Diskretisierungen. Analog zum Vorgehen bei den Histogrammen können wir wieder diskretisieren. Im Hinblick auf die Regressoren haben wir dies beim Helikopter-Beispiel getan. Die Diskretisierung können wir auch bei der Respons vornehmen. Damit wird aus dem Regressionsproblem ein Kontingenztafel-Problem. Wir gehen hier nicht weiter auf diese Möglichkeit ein.

2.6.2. Ergänzung: Software-Test. Alle vorbereiteten Algorithmen, wie hier die Algorithmen zu den linearen Modellen und deren Varianten, sollten mit derselben Vorsicht behandelt werden wie mathematische Veröffentlichungen und Zitate. Selbst einfache Programme jedoch haben schnell eine semantische Komplexität, die weit über die mathematischer Beweise hinaus geht. Die übliche Strategie des "Nachrechnens" oder des schrittweisen Nachvollziehens verbietet sich dadurch. Anstelle einer vollständigen Überprüfung muss ein selektives Testen treten. Eine Teststrategie ist z.B. in [Sawitzki, 1994] beschrieben.

Die Überprüfung ist sowohl für die Implementierung als auch für den zu Grunde liegenden abstrakten Algorithmus nötig.

Aufgabe 2.15	
	Für diese Aufgabenserie sei $y_i = a + bx_i + \varepsilon_i$ mit ε_i iid $\sim N(0, \sigma^2)$ und $x_i = i, i = 1, \ldots, 10$.
	Wählen Sie eine Strategie, um `lm()` im Hinblick auf den Parameterraum (a, b, σ^2) zu überprüfen. Gibt es eine naheliegende Zellzerlegung für die einzelnen Parameter a, b, σ^2? Welche trivialen Fälle gibt es? Welche (uniforme) Asymptotik? Wählen Sie Testpunkte jeweils in der Mitte jeder Zelle und an den Rändern. Führen Sie diese Test durch und fassen Sie die Resultate zusammen.
	Wählen Sie eine Strategie, um `lm()` im Hinblick auf den gemeinsamen Parameterraum (a, b, σ^2) zu überprüfen. Gibt es eine naheliegende Zellzerlegung? Wählen Sie Testpunkte jeweils in der Mitte jeder Zelle und an den Rändern. Führen Sie diese Test durch und fassen Sie die Resultate zusammen.
	Welche Symmetrien/Antisymmetrien gibt es? Überprüfen Sie diese Symmetrien.
	Welche Invarianten/welches Covariante Verhalten gibt es? Überprüfen Sie diese Invarianten/Covarianten.

Aufgabe 2.16	
	Für diese Aufgabenserie sei $y_i = a + bx_i + \varepsilon_i$ mit ε_i iid $\sim N(0, \sigma^2)$.
	Welche extremen Designs (x_i) gibt es? Überprüfen Sie das Verhalten von `lm()` bei vier extremalen Designs.
	Führen Sie die Aufgaben aus der letzten Gruppe aus, jetzt mit variablem Design. Fassen Sie Ihren Bericht zusammen.

Aufgabe 2.17	
	Für diese Aufgabenserie sei $y_i = a + bx_i + \varepsilon_i$ mit ε_i iid $\sim N(0, \sigma^2)$.
	Modifizieren Sie `lm()` so, dass eine gesicherte Funktion für das einfache lineare Modell entsteht.

2.6.3. R-Datentypen. R ist eine interpretierte Programmiersprache. Sie will es dem Anwender erlauben, Definitionen und Konkretisierungen flexibel zu handhaben. Aus Geschwindigkeitsgründen versucht R, Auswertungen so spät wie möglich durchzuführen. Dies erfordert einige Einschränkungen an die Sprache, die R von anderen Programiersprachen unterscheidet.

R kennt keine abstrakten Datantypen. Ein Datentyp ist durch seine Instanzen, die Variablen, definiert.

Der Datentyp einer Variablen ist dynamisch: derselbe Name in denselben Kontext kann zu unterschiedlichen Zeiten unterschiedliche Variablenwerte und Variablentypen kennzeichnen.

Dennoch hat zu jeder Zeit eine Variable einen bestimmten Typ. Das R-Typensystem versteht man jedoch am besten in seiner historischen Entwicklung und die entsprechenden Funktionen. In der ersten Stufe war der Typ beschrieben durch *mode()* (z.B. "numeric") und *storage.mode()* (z.B. "integer" oder "real").

Beide Funktionen sind weitgehend durch *typeof()* abgelöst. Eine Zusammenfassung der Typen, die durch *typeof()* derzeit berichtet werden, ist in [**R D04c**] zu finden.

Komplexere Datentypen werden auf die in [**R D04c**] definierten zurückgeführt, indem die Variablen mit Attributen versehen werden. Dies geschieht mit der Funktion *attr()*, die auch benutzt werden kann, um Attribute zu setzen. So sind eine Matrix oder ein Array nur spezielle Vektoren, die sich dadurch auszeichnen, dass sie ein *dim*-Attribut haben.

Für die wesentlichen Typen sind Inspektionsprozeduren und Umwandlungsprozduren: *is.*⟨typ⟩*()* prüft auf Typenzugehörigkeit, *as.*⟨typ⟩*()* wandelt den Typ.

2.6.4. Klassen und Polymorphe Funktionen. Im Zuge der Weiterentwicklung wurde eine Anleihe an objekt-orientierte Programmierung gemacht. Dafür wurde ein spezielles Attribut mit dem Namen *class* genutzt: der Name des Typs (oder der "Klasse") wird hier gespeichert. Multiple Klassenzugehörigkeit in einer Hierarchie von Klassen ist auch möglich. In diesem Fall enthält *class* einen Vektor von Klassennamen. So hat zum Beispiel ein geordneter Faktor die Kennzeichnung *class=c("ordered", "factor")*. Zur Verwaltung der Klassen stehen Funktionen *class()*, *unclass()*, *inherits()* zur Verfügung.

Die Klassenzuordnung basiert dabei auf Vertrauen. R überprüft nicht, ob die Datenstruktur mit der angegebenen Klasse konsistent ist.

Funktionen wie *plot()*, *print()* und viele weitere überprüfen die Typen- und Klassenzugehörigkeit ihrer Argument und verzweigen dann zu entsprechenden spezialisierten Funktionen. Dieses nennt man Polymorphismus. Wenn man eine polymorphe Funktion auflistet, erhält man zunächst nur den Hinweise, das eine Dispatch-Funktion *UseMethod()* aufgerufen wird. Beispiel:

Eingabe

```
plot
```

Ausgabe

```
function (x, y, ...)
{
    if (is.null(attr(x, "class")) && is.function(x)) {
        nms <- names(list(...))
        if (missing(y))
            y <- {
                if (!"from" %in% nms)
                  0
                else if (!"to" %in% nms)
                  1
                else if (!"xlim" %in% nms)
                  NULL
            }
```

```
        if ("ylab" %in% nms)
            plot.function(x, y, ...)
        else plot.function(x, y, ylab = paste(deparse(substitute(x)),
            "(x)"), ...)
    }
    else UseMethod("plot")
}
<environment: namespace:graphics>
```

UseMethod() bestimmt die Klasse des ersten Argument, mit dem die Funktion aufgerufen wurde, sucht dann ein Spezialisierung für diese Klasse und ruft schliesslich die gefundene Funktion auf. Für ***polymorphe*** Funktionen findet man die entsprechenden Spezialisierungen mit Hilfe von *methods()*, z.B. *methods(plot)*.

2.7. Statistische Zusammenfassung

Als Leitbeispiel diente in diesem Kapitel die statistische Analyse eines funktionellen Zusammenhangs. Die betrachteten Modelle sind finit in dem Sinne, dass ein endlichdimensionaler Funktionenraum den in Betracht gezogenen Zusammenhang zwischen Regressoren und Respons beschreibt. Die stochastische Komponente in diesen Modellen ist noch auf eine (eindimensionale) Zufallsverteilung beschränkt. Die Dimensionsbegrifffe verdienen hier eine genauere Betrachtung. Wir haben zum einen die Regressor-Dimension. Dies ist die Dimension des Raumes der beobachteten oder abgeleiteten Parameter. Nicht alle Parameter sind identifizierbar oder schätzbar. Genauer gefasst ist die Dimension die Vektorraum-Dimension des gewählten Modell-Raums. Die Modelle werden durch Parameter in diesem Raum beschrieben. Diese Parameter können unbekannt oder hypothetisch sein. In jedem Fall aber haben wir sie als deterministisch betrachtet. Zum anderen haben wir die stochastische Komponente, repräsentiert durch den Fehler-Term. In diesem Kapitel sind wir von homogenen Fehlern ausgegangen. Damit bestimmt der Fehler-Term im Prinzip eine Dimension, die allerdings aus einem Raum von Verteilungen stammt. Für den Spezialfall der einfachen Gauß-linearen Modell sind die Verteilungen mit zwei Parametern präzisiert, dem Erwartungswert und der Varianz. Von dem Erwartungswert haben wir uns durch die Annahme befreit, dass das Modell im Mittel alle systematischen Effekte erfasst, also der Erwartungswert null ist. Die Varianz ist in unseren Problemen noch ein unbekannter Störparameter. Wir haben die dadurch entstehenden Problemen vermieden, indem wir uns auf Probleme beschränkt haben, in denen diese Störparameter durch einen geschätzten Wert ersetzt und so eliminiert wird.

2.8. Literatur und weitere Hinweise:

[**CH92**] Chambers, J.M.; Hastie, T.J. (eds.) (1992): Statistical Models in S. NewYork: Chapman & Hall.

[**Jør93**] Jørgensen, B. (1993): The Theory of Linear Models. NewYork: Chapman & Hall.

[**R D04c**] R Development Core Team (2004): The R language definition.

[**Saw94a**] Sawitzki, G. (1994): Numerical Reliability of Data Analysis Systems. Computational Statistics & Data Analysis 18.2 (1994) 269-286. Siehe <http://www.statlab.uni-heidelberg.de/reports/>.

[**Saw94b**] Sawitzki, G. (1994):Report on the Numerical Reliability of Data Analysis Systems. Computational Statistics & Data Analysis/SSN 18.2 (1994) 289-301. <http://www.statlab.uni-heidelberg.de/reports/>.

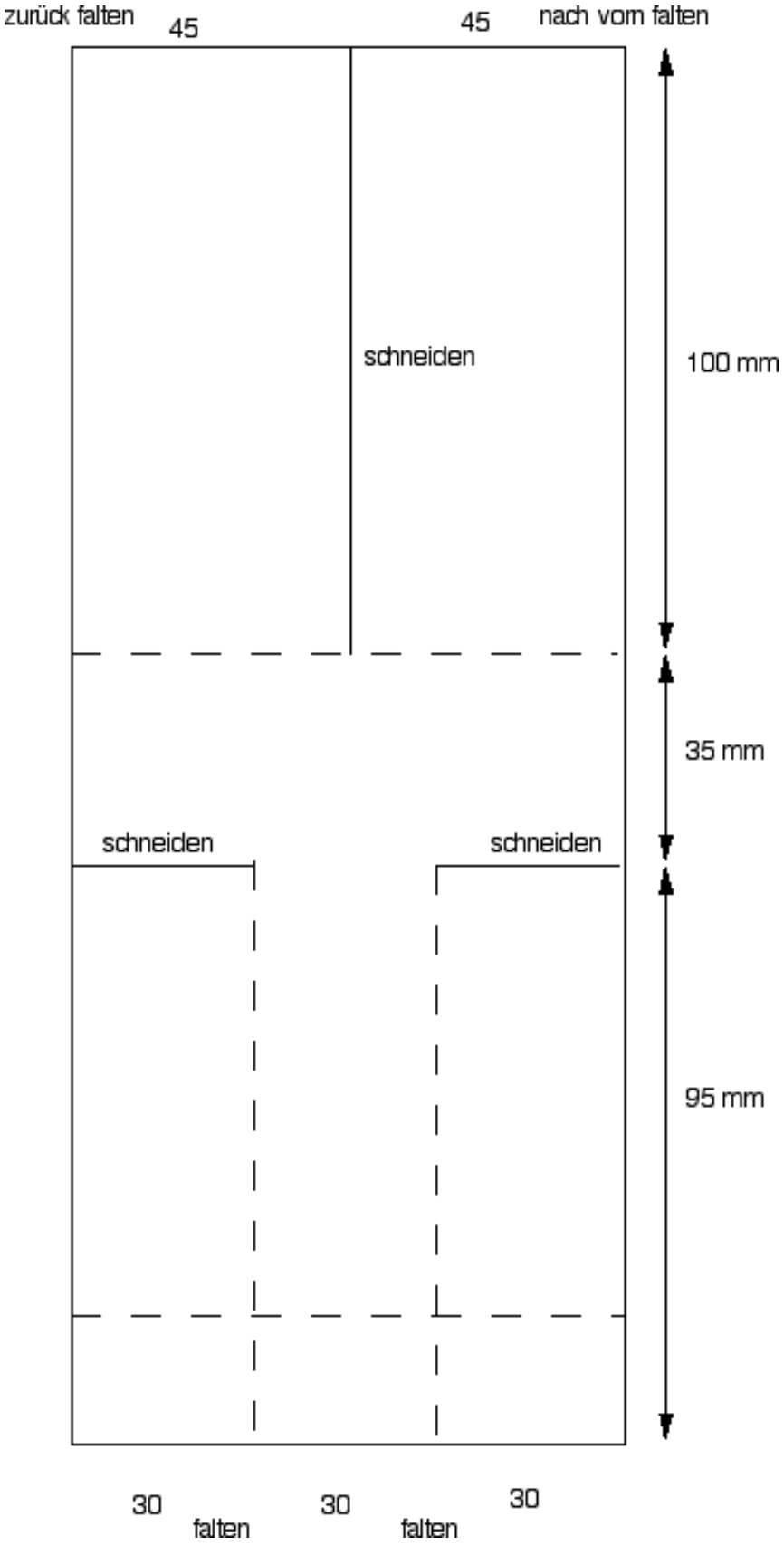


Abbildung 2.1. KiwiHopp

KAPITEL 3

Vergleich von Verteilungen

Wir beginnen mit der Konstruktion eines kleinen Werkzeugs, das uns Beispieldaten liefern wird. Basis ist ein kleiner Reaktionstester. Wir zeichen einen "zufälligen" Punkt, warten auf einen Maus-Klick, und registrieren die Position des Mauszeigers. Damit bei wiederholten Aufrufen das Bild stabil bleibt, fixieren wir das Koordinatensystem.

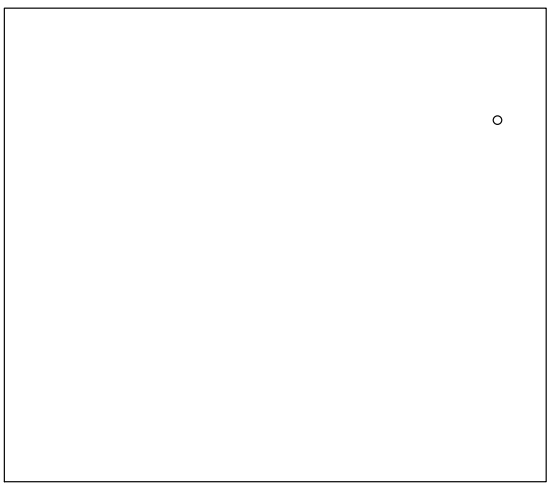
Beispiel 3.1:

Eingabe

```
plot(x = runif(1), y = runif(1), xlim = c(0, 1),
    ylim = c(0, 1), main = "Bitte auf den Punkt klicken",
    xlab = "", ylab = "", axes = FALSE, frame.plot = TRUE)
locator(1)
```

Ausgabe

```
$x
[1] 0.8214286

$y
[1] 0.89375
```



Wir verpacken nun diesen Basistester. Wir merken uns die Koordinaten, versuchen, die Reaktionszeit des Benutzers zu messen, und liefern alle Resultate als Liste zurück.

Beispiel 3.2:

Eingabe

```
click1 <- function() {
    x <- runif(1)
    y <- runif(1)
    plot(x = x, y = y, xlim = c(0, 1), ylim = c(0,
        1), main = "Bitte auf den Punkt klicken",
        xlab = "", ylab = "", axes = FALSE, frame.plot = TRUE)
    clicktime <- system.time(xyclick <- locator(1))
    list(timestamp = Sys.time(), x = x, y = y, xclick = xyclick$x,
        yclick = xyclick$y, tclick = clicktime[3])
}
```

Zur weiteren Verarbeitung können wir die Liste in einen `data.frame` integrieren und diesen `data.frame` schrittweise mit Hilfe von `rbind` erweitern.

Beispiel 3.3:

Eingabe

```
dx <- as.data.frame(click1())
dx <- rbind(dx, data.frame(click1()))
dx
```

Ausgabe

```
             timestamp       x       y  xclick  yclick tclick
1  2005-12-08 11:23:26 0.30744 0.44278 0.81765 0.88077   0.18
11 2005-12-08 11:23:26 0.34552 0.28214 0.45462 0.43077   0.49
```

Aufgabe 3.1	
	Definieren Sie eine Funktion `click(runs)`, die zu vorgegebener Anzahl *runs* die Aufgabe von `click1()` wiederholt und das Resultat als `data.frame` übergibt. Eine erste (zusätzliche) Messung sollte als Warmlaufen betrachtet werden und nicht in die Auswertung mit einbezogen werden. Wählen Sie eine Anzahl `runs`. Begründen Sie Ihre Wahl von `runs`. Führen Sie `click(runs)` und speichern Sie das Resultat mit Hilfe von `write.table()` in einer Datei. Stellen Sie die Verteilung der Komponente `tclick()` mit den Methoden aus Kapitel 1 (Verteilungsfunktion, Histogramm, Boxplot) dar.

3.1. Shift/Skalenfamilien

Ein Vergleich von Verteilungen kann eine sehr anspruchsvolle Aufgabe sein. Der mathematische Raum, in dem Verteilungen angesiedelt sind, ist nicht mehr ein Zahlenraum oder ein (endlichdimensionaler) Vektorraum. Der eigentliche Raum, in dem Verteilungen beheimatet sind, ist ein Raum von Maßen. In einfachen Fällen, etwa bei Verteilungen auf $\mathbb{R}$, können wir alles auf Verteilungsfunktionen reduzieren und sind damit immerhin bei einem Funktionenraum. Selbst hier kann ein Vergleich noch große Schwierigkeiten machen. Wir haben keine einfache Ordnungsrelation.

Aufgabe 3.2	
	Führen Sie Aufgabe 3.4 einmal mit der rechten und einmal mit der linken Hand durch. Vergleichen Sie die empirischen Verteilungen von `tclick()`. Die erhobenen Daten enthalten auch Information über die Positionen. Definieren Sie ein Maß `dist` für die Abweichung. Begründen Sie Ihre Definition. Führen Sie auch für `dist` einen rechts/links Vergleich durch.

Wir konzentrieren uns hier auf den Vergleich von nur zwei Verteilungen, etwa der von Messungen in zwei Behandlungsgruppen. Wie nehmen wieder einen einfachen Fall: die Beobachtungen seien jeweils unabhängig identisch verteilt (jetzt mit der für den Vergleich von Behandlungen üblichen Index-Notation).

Y_{ij} unabhängig identisch verteilt mit Verteilungsfunktion F_i

$i = 1, 2$ Behandlungen

$j = 1, \ldots, n_i$ Beobachtungen in Behandlungsgruppe i.

Wie vergleichen wir die Beobachtungen in den Gruppen $i = 1, 2$? Die (einfachen) linearen Modelle

$$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$$

betrachten als Unterschied häufig nur eine Verschiebung $\Delta = \alpha_1 - \alpha_2$.

Bezeichnungen: Zu einer Verteilung mit Verteilungsfunktion F heißt die Familie mit

$$F_a(x) \;=\; F(x - a)$$

die ***Shift-Familie*** zu F. Die Verschiebung a heißt Shift- oder Lage-Parameter.

Die Behandlung kann aber, in Wahrscheinlichkeiten gesprochen, die Wahrscheinlichkeitsmassen auch in anderer Weise verschieben, als es ein additiver Term im Modell bewirken kann. Wir brauchen allgemeinere Vergleichsmöglichkeiten als die durch einen Shift definierten.

Bezeichnung: Eine Verteilung mit Verteilungsfunktion F_1 heißt ***stochastisch kleiner*** als eine mit Verteilungsfunktion $F_2 (F_1 \prec F_2)$, wenn F_1 eher bei kleineren Werten liegt als F_2:

$$F_1(x) \geq F_2(x) \; \forall x$$

und

$$F_1(x) > F_2(x) \text{ für mindestens ein } x.$$

Aufgabe 3.3	
	Wie sieht ein PP-Plot für F_1 gegen F_2 aus, wenn $F_1 \prec F_2$? Wie sieht ein QQ-Plot für F_1 gegen F_2 aus, wenn $F_1 \prec F_2$?

Leider ist die dadurch definierte stochastische Ordnung nur von beschränktem Wert. Sie definiert keine vollständige Ordnung. Für Shift-Familien ist sie ausreichend. Aber Gegenbeispiele kann man sich konstruieren, wenn man die Shift-Familien nur geringfügig erweitert.

Bezeichnungen: Zu einer Verteilung mit Verteilungsfunktion F heißt die Familie mit

$$F_{a,b}(x) \;=\; F(\frac{x-a}{b})$$

die ***Skalen-Shiftfamile*** zu F.

Aufgabe 3.4	
	Die Skalen-Shiftfamilien zur $N(0,1)$-Verteilung sind die $N(\mu,\sigma^2)$-Verteilungen. Welche $N(\mu,\sigma^2)$-Verteilungen sind stochastisch kleiner als die $N(0,1)$-Verteilung? Welche sind stochastisch größer? Für welche ist die Ordnungsrelation undefiniert?

Die aus der linearen Theorie kommende Einordnung nach Lage/Skalen und die stochastische Ordnung klaffen auseinander, und beide Aspekte müssen oft getrennt betrachtet werden. Viele statistische Methoden konzentrieren sich auf Aspekte, die durch Skalen-Shiftfamilien motiviert sind. Unterschiede jenseits dessen, was durch Skala und Shift beschrieben werden kann, bedürfen oft besonderer Aufmerksamkeit.

In Kapitel 2 haben wir eine typische Situation für lineare Modelle betrachtet. Im Prinzip haben wir es mit Skalen-Shiftfamilien zu tun. Der (stochastische) Skalenparameter in diesen Modellen ist jedoch nur ein Störparameter, der eliminiert werden kann. Dazu benutzten wir einen Schätzer für diesen Skalenparameter, die residuelle Varianz, die wir dann heraus gekürzt haben. Als eine Besonderheit bei Gauß-linearen Modellen erhalten wir hier unabhängige Schätzer für Erwartungswert und Varianz. Dadurch können wir im Falle der einfachen Gauß-linearen Modelle Statistiken gewinnen, die nicht mehr vom Skalenparameter abhängen.

Im allgemeinen Fall haben wir jedoch eine aufsteigende Leiter von Problemen:

- Shift-Alternativen
- Shift/Skalen-Alternativen
- stochastische Ordnung
- allgemeinere Alternativen

3.2. QQ-Plot, PP-Plot

Als Vergleichsdarstellung für Verteilungsfunktionen haben wir den PP-Plot und den QQ-Plot kennengelernt. So lange man innerhalb einer Skalen-Shiftfamilie bleibt, hat der QQ-Plot zumindest in einer Hinsicht einen Vorteil gegenüber dem PP-Plot:

Bemerkung 3.1. Sind F_1, F_2 Verteilungsfunktionen aus einer gemeinsamen Skalen-Shiftfamilie, so ist der QQ-Plot von F_1 gegen F_2 eine Gerade.

Insbesondere für die Gaußverteilungen ist der QQ-Plot gegen $N(0,1)$ ein wichtiges Hilfsmittel. Jede Gaußverteilung gibt in diesem Plot eine Gerade.

Der QQ-Plot ist für diese Situation bereits als Funktion *qqnorm()* vorbereitet.

Für den Vergleich von zwei Stichproben mit gleichem Stichprobenumfang kann die entsprechende Funktion *qqplot()* genutzt werden: bezeichnen wir die empirischen Quantile mit $Y_{1(i:n)}$ bzw. $Y_{2(i:n)}$, so ist dieser Plot der Graph $\left(Y_{1(i:n)}, Y_{2(i:n)}\right)_{i=1\ldots n}$. Sind die Stichprobenumfänge verschieden, so behilft sich R und generiert die Markierungspunkte durch lineare Interpolation, wobei der kleinere der beiden Stichprobenumfänge gewählt wird.

Der PP-Plot hat keine dem QQ-Plot vergleichbare Äquivarianzeigenschaften. Wenn wir Skalen-Shiftparameter eliminieren wollen, müssen wir die Daten zunächst entsprechend transformieren.

Der Äquivarianz des QQ-Plots als Vorteil stehen auf der anderen Seite strukturelle Nachteile entgegen. In Bereichen niedriger Dichte bestimmen empirisch wenige Datenpunkte den Plot. Entsprechend hat er hier eine große Varianz. Gleichzeitig sind hier der Wahrscheinlichkeit nach benachbarte Quantile im Wertebereich weit entfernt: die große Varianz kombiniert sich ungünstig mit einer großen Variabilität, und der QQ-Plot zeigt entsprechend große Fluktuation. Für die meisten Lehrbuch-Verteilungen bedeutet dies, dass der QQ-Plot in den Randbereichen kaum zu interpretieren ist. Der PP-Plot hat keine entsprechenden Skalendefizite.

---

help(qqplot)

`qqnorm` *Quantile-Quantile Plots*

Description.

`qqnorm` is a generic function the default method of which produces a normal QQ plot of the values in `y`. `qqline` adds a line to a normal quantile-quantile plot which passes through the first and third quartiles.

`qqplot` produces a QQ plot of two datasets.

Graphical parameters may be given as arguments to `qqnorm`, `qqplot` and `qqline`.

Usage.

```
qqnorm(y, ...)
## Default S3 method:
qqnorm(y, ylim, main = "Normal Q-Q Plot",
       xlab = "Theoretical Quantiles",
       ylab = "Sample Quantiles", plot.it = TRUE, datax = FALSE,
       ...)
qqline(y, datax = FALSE, ...)
qqplot(x, y, plot.it = TRUE, xlab = deparse(substitute(x)),
       ylab = deparse(substitute(y)), ...)
```

Arguments.

`x`	The first sample for `qqplot`.
`y`	The second or only data sample.
`xlab, ylab, main`	plot labels.

`plot.it` logical. Should the result be plotted?
`datax` logical. Should data values be on the x-axis?
`ylim, ...` graphical parameters.

Value.

For `qqnorm` and `qqplot`, a list with components
`x` The x coordinates of the points that were/would be plotted
`y` The original `y` vector, i.e., the corresponding y coordinates *including NAs*.

References.

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

See Also.

`ppoints`.

Examples.

```
y <- rt(200, df = 5)
qqnorm(y); qqline(y, col = 2)
qqplot(y, rt(300, df = 5))

qqnorm(precip, ylab = "Precipitation [in/yr] for 70 US cities")
```

Aufgabe 3.5	
	Benutzen Sie den Quantil-Quantil-Plot, um die Resultate des rechts/-links *click*-Experiments zu vergleichen. Formulieren Sie die Resultate.
	Fassen Sie die rechts/links *tclick*-Daten zu einem Vektor zusammen. Vergleichen Sie den Quantil-Quantil-Plot mit dem von Monte-Carlo-Stichproben aus dem zusammengefassten Vektor. Erinnerung: Zufallsstichproben können Sie mit *sample()* ziehen. Mit *par(mfrow=c(2,2))* teilen Sie den Zeichenbereich so ein, dass Sie vier Plots gleichzeitig sehen können.
**	Benutzen Sie bei *sample()* den Parameterwert *replace = FALSE*. Wie müssen Sie jetzt *sample()* anwenden, um den zusammengefassten Vektor in zwei Vektoren mit Monte-Carlo-Stichproben aufzuteilen? Welche Unterschiede zu *replace = TRUE* sind zu erwarten?

(Fortsetzung)→

Aufgabe 3.6	(Fortsetzung)

Aufgabe 3.6	
	Bestimmen Sie für die `tclick`-Daten des rechts/links `click`-Experiments Skalen- und Shiftparameter so, dass die Verteilungen in den Gruppen nach Skalen-Shift-Transformation möglichst gut übereinstimmen. Beschreiben Sie die Unterschiede anhand der Skalen-Shiftparameter. Verwenden Sie dazu eine Modellierung mit einem linearen Modell.
	Benutzen Sie die Funktion `boxplot()`, um Quartile und Flankenverhalten darzustellen. Vergleichen Sie die Information mit den Skalen-Shiftparametern. *Hinweis:* was entspricht dem Shift(Lage)parameter? Was entspricht dem Skalenparameter?

Wenn Darstellungen affin invariant sind, könnnen Skalen-Shiftparameter ignoriert werden. Wenn Darstellungen nicht affin invariant sind, ist es häufig hilfreich, zunächst Skalen-Shiftparameter geeignet zu schätzen, die Verteilungen zu standardisieren, und dann die standardisierten Verteilungen zu untersuchen.

Das Problem, das wir uns damit potentiell einhandeln, ist, dass dann das stochastische Verhalten der Schätzung für die Skalen-Shiftparameter berücksichtigt werden muss. Der übliche Ausweg ist es, vorsichtigerweise "konservative" Tests und robuste Schätzer zu benutzen.

Um Verteilungen direkt miteinander vergleichen zu können, greifen wir auf Techniken aus dem ersten Kapitel zurück. Was dort über den Vergleich zu einer theoretischen Verteilung gesagt worden ist, kann analog auf den Vergleich von zwei Verteilungen, z.B. aus zwei Behandlungsgruppen, übertragen werden. Die statistischen Aussagen müssen jedoch revidiert werden. Nun betrachten wir nicht mehr eine feste und eine zufällige Verteilung, sondern wir vergleichen zwei zufällige (empirische) Verteilungen.

Die für den Einstichproben-Fall (eine Stichprobe im Vergleich zu einer hypothetischen Verteilung) benutzte Idee von Monte-Carlo-Bändern kann nicht unmittelbar übertragen werden: wir wollen zwei Verteilungen miteinander vergleichen, aber wir haben keine ausgezeichnete Modellverteilung, aus der wir Referenzstichproben ziehen können.

Wir können jedoch die Idee modifizieren und bedingte Monte-Carlo-Bänder konstruieren. Bedingt bedeutet hier: die Konstruktion hängt von beobachteten Stichprobenwerten ab. Wir nehmen an, dass wir zwei Stichproben $Y_{11}, \ldots, Y_{1n_1}$ und $Y_{21}, \ldots, Y_{2n_2}$ von insgesamt unabhängigen und innerhalb der Gruppen identisch nach F_1 bzw. F_2 verteilten Beobachtungen haben. Falls kein Unterschied zwischen den Verteilungen besteht, so ist $(Y_{11}, \ldots, Y_{1n_1}, Y_{21}, \ldots, Y_{2n_2})$ eine iid-Stichprobe aus einer gemeinsamen Verteilung $F = F_1 = F_2$ mit Stichprobenumfang $n = n_1 + n_2$. Bei einer iid-Stichprobe hätte jede Permutation der Indizes die gleiche Wahrscheinlichkeit.

Die motiviert das folgende Verfahren: wir permutieren das Tupel $(Y_{11}, \ldots, Y_{1n_1}, Y_{21}, \ldots, Y_{2n_2})$ und ordnen die ersten n_1 Werte (nach Permutation) der ersten Gruppe zu, die anderen der zweiten. Wir benutzen die so generierten Werte, um Monte-Carlo-Bänder zu generieren.

Aufgabe 3.7	
	Modifizieren Sie die Funktionen für PP-Plot und QQ-Plot so, dass Monte-Carlo-Bänder für den Vergleich von zwei Stichproben hinzugefügt werden. *Hinweis:* mit der Funktione `sample()` können Sie zufällige Permutationen generieren.

Bei größerem Stichprobenumfang kann der Aufwand Permutationen zu generieren zu zeitaufwendig sein. Um Verwaltungsaufwand zu sparen, können wir die Permutation durch ein Ziehen aus den n Werten $(Y_{11}, \ldots, Y_{1n_1}, Y_{12}, \ldots, Y_{1n_2})$ ***mit Zurücklegen*** ersetzen. Diese approximative Lösung wird als ***Bootstrap-Approximation***[1] bezeichnet.

Da es nur endlich viele Permutationen gibt, können wir bei kleinem Stichprobenumfang auch alle Permutationen durchgehen. Wir wählen die Bänder dann so, dass ein hinreichend großer Anteil (etwa mehr als 95 %) aller Kurven innerhalb der Bänder liegt. Permutationen, die sich nur innerhalb der Gruppen unterscheiden, ergeben dieselben Kurven. Diese Zusatzüberlegung zeigt, dass wir nicht alle n! Permutationen überprüfen müssen, sondern nur die $\binom{n}{n_1}$ Auswahlen für die Zuteilung zu den Gruppen.

Aufgabe 3.8	
**	Ergänzen Sie PP-Plot und QQ-Plot für die `click`-Experimente duch Permutations-Bänder, die 95 % der Permutationen abdecken.
*	Erzeugen Sie neue Plots, in denen Sie die PP-Plots und QQ-Plots durch Monte-Carlo-Bänder aus den Permutationen ergänzen. Benutzen Sie die Einhüllende von 19 Monte-Carlo-Stichproben. *Hinweis:* benutzen Sie die Funktion `sample()` um eine Stichprobe vom Umfang n_1 aus $x = (Y_{11}, \ldots, Y_{1n_1}, Y_{12}, \ldots, Y_{1n_2})$ zu ziehen.
	Erzeugen Sie neue Plots, in denen Sie die PP-Plots und QQ-Plots durch Monte-Carlo-Bänder aus den Permutationen ergänzen. Benutzen Sie die Einhüllende von 19 Bootstrap-Stichproben. *Hinweis:* Siehe `help(sample)`.

Aufgabe 3.9	
*	Versuchen Sie, die Eigenschaften der Permutationsbänder, Monte-Carlo-Bänder und Bootstrap-Bänder zu vergleichen, wenn $F_1 = F_2$ gilt.

Wenn nicht die Verteilungen verglichen werden sollen, sondern nur einzelne festgelegte Kenngrößen, so können diese Strategien analog eingesetzt werden. Wenn wir uns z.B. auf die Lage-Alternative beschränken (d.h. F_1 und F_2 sind aus eine Lagefamilie, d.h. $F_1(x) = F_2(x-a)$ für ein a), so können wir etwa den Mittelwert (oder den Median) als Kenngröße nehmen. Auf diese Kenngröße kann das obige Vorgehen analog angewandt werden, um zu

[1] Vorsicht: es gibt beliebig wilde Definitionen von Bootstrap. Versuchen Sie stets, das Vorgehen mathematisch genau zu formulieren, wenn von Bootstrap die Rede ist.

entscheiden, ob die Hypothese, dass die Verteilungen sich nicht unterscheiden ($a = 0$), angesichts der Daten haltbar ist.

Aufgabe 3.10	
*	Formulieren Sie die obigen Strategien für Intervalle für einzelne Teststatistiken (Beispiel: Mittelwert) anstelle für Bänder. *Hinweis:* Können Sie anstelle der zwei Mittelwerte für beide Gruppen eine eindimensionale zusammenfassende Statistik benutzen?

3.3. Tests auf Shift

Wenn wir zusätzliche Verteilungsannahmen machen, können wir andere Entscheidungsverfahren wählen. Für diese Verfahren sind aber die Verteilungsannahmen kritisch. Diese Abhängigkeit von den Verteilungsannahmen kann gemildert oder vermieden werden, wenn wir geeignete Verteilungsannahmen sicherstellen könnnen. Der F-Test, den wir im letzten Kapitel kennengelernt haben, ist ein Beispiel für ein verteilungsabhängiges Verfahren. Für den Zwei-Stichprobenfall kann dieser Test modifiziert werden zum t-Test.

---

help(**t**.test)

`t.test` *Student's t-Test*

Description.

Performs one and two sample t-tests on vectors of data.

Usage.

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

## Default S3 method:
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)

## S3 method for class 'formula':
t.test(formula, data, subset, na.action, ...)
```

Arguments.

`x`	a numeric vector of data values.
`y`	an optional numeric vector data values.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`. You can specify just the initial letter.
`mu`	a number indicating the true value of the mean (or difference in means if you are performing a two sample test).
`paired`	a logical indicating whether you want a paired t-test.

`var.equal`	a logical variable indicating whether to treat the two variances as being equal. If `TRUE` then the pooled variance is used to estimate the variance otherwise the Welch (or Satterthwaite) approximation to the degrees of freedom is used.
`conf.level`	confidence level of the interval.
`formula`	a formula of the form `lhs ~ rhs` where `lhs` is a numeric variable giving the data values and `rhs` a factor with two levels giving the corresponding groups.
`data`	an optional data frame containing the variables in the model formula.
`subset`	an optional vector specifying a subset of observations to be used.
`na.action`	a function which indicates what should happen when the data contain `NA`s. Defaults to `getOption("na.action")`.
`...`	further arguments to be passed to or from methods.

Details.

The formula interface is only applicable for the 2-sample tests.

If `paired` is `TRUE` then both `x` and `y` must be specified and they must be the same length. Missing values are removed (in pairs if `paired` is `TRUE`). If `var.equal` is `TRUE` then the pooled estimate of the variance is used. By default, if `var.equal` is `FALSE` then the variance is estimated separately for both groups and the Welch modification to the degrees of freedom is used.

If the input data are effectively constant (compared to the larger of the two means) an error is generated.

Value.

A list with class `"htest"` containing the following components:

`statistic`	the value of the t-statistic.
`parameter`	the degrees of freedom for the t-statistic.
`p.value`	the p-value for the test.
`conf.int`	a confidence interval for the mean appropriate to the specified alternative hypothesis.
`estimate`	the estimated mean or difference in means depending on whether it was a one-sample test or a two-sample test.
`null.value`	the specified hypothesized value of the mean or mean difference depending on whether it was a one-sample test or a two-sample test.
`alternative`	a character string describing the alternative hypothesis.
`method`	a character string indicating what type of t-test was performed.
`data.name`	a character string giving the name(s) of the data.

See Also.

`prop.test`

Examples.

```
t.test(1:10,y=c(7:20))      # P = .00001855
t.test(1:10,y=c(7:20, 200)) # P = .1245    -- NOT significant anymore

## Classical example: Student's sleep data
plot(extra ~ group, data = sleep)
## Traditional interface
with(sleep, t.test(extra[group == 1], extra[group == 2]))
```

```
## Formula interface
t.test(extra ~ group, data = sleep)
```

In seiner einfachsten Form setzt der t-Test voraus, dass wir unabhängig identisch verteilte Stichproben aus Normalverteilungen haben. Tatsächlich reichen schwächere Voraussetzungen. Wenn wir die t-Test-Statistik als

$$t = \frac{\widehat{\mu_1} - \widehat{\mu_2}}{\sqrt{(\widehat{Var(\widehat{\mu_1} - \widehat{\mu_2})})}}$$

schreiben, so sehen wir, dass t t-verteilt ist, wenn $\widehat{\mu_1} - \widehat{\mu_2}$ normalverteilt und $(\widehat{Var(\widehat{\mu_1} - \widehat{\mu_2})})$ χ^2 verteilt ist, und beide Term unabhängig sind. Der zentrale Grenzwertsatz garantiert, dass $\widehat{\mu_1} - \widehat{\mu_2}$ unter milden Bedingungen zumindest asymptotisch normalverteilt ist. Analoges gilt oft für $(\widehat{Var(\widehat{\mu_1} - \widehat{\mu_2})})$. Gilt die Unabhängigkeit beider Terme, so ist t approximativ t-verteilt.

Aufgabe 3.11	
*	Bestimmen Sie in einer Simulation die Verteilung von $\overline{Y}$, $\widehat{Var(Y)}$ und der t-Statistik für Y aus der uniformen Verteilung $U[0,1]$ mit Stichprobenumfang $n = 1, \ldots, 10$. Vergleichen Sie die Verteilungen aus der Simulation mit der entsprechenden Normal-, χ^2- bzw. t-Verteilung. Bestimmen Sie in einer Simulation die Verteilung von $\overline{Y}$, $\widehat{Var(Y)}$ und der t-Statistik für Y aus einer Mischung, die zu 90% aus einer $N(0,1)$- und zu 10% aus einer $N(0,10)$-Verteilung stammt, mit Stichprobenumfang $n = 1, \ldots, 10$. Vergleichen Sie die Verteilungen aus der Simulation mit der entsprechenden Normal-, χ^2- bzw. t-Verteilung.

Der t-Test hat eine gewisse Robustheit, die ihm eine approximative Gültigkeit geben kann. Man kann sich jedoch ganz von der Normalverteilungs-Voraussetzung befreien. Wenn wir analog zum F-Test bzw. t-Test vorgehen, aber anstelle der Urdaten die Ränge benutzen, gewinnen wir Testverfahren, die verteilungsunabhängig sind (zumindest, solange keine Bindungen auftreten können). Der Wilcoxon-Test ist eine verteilungsunabhängige Variante des t-Tests. Theoretisch entspricht er genau dem t-Test, angewandt auf die (gemeinsam) rangtransformierten Daten. Wie der t-Test ist dieser Test nur darauf ausgelegt, die Nullhypothese (kein Unterschied) gegen eine Shift-Alternative zu testen. Für die praktische Anwendung können arithmetische Vereinfachungen ausgenutzt werden. Deshalb ist die Beziehung zwischen den üblichen Formeln für den t-Test und für den Wilcoxon-Test nicht einfach zu erkennen.

Um den Wilcoxon-Test anzuwenden, muss zum einen die Teststatistik berechnet werden. Zur Bestimmung kritischer Werte, mit denen die Teststatistik zu vergleichen ist, muss zum anderen die Verteilungsfunktion ausgewertet werden. Sind alle Beobachtungen paarweise verschieden, so hängt diese Funktion nur von n_1 und n_2 ab, und relativ einfache Algorithmen stehen zur Verfügung. Diese sind in der Funktion R standardmäßig verfügbar und werden von *`wilcox.test()`* benutzt. Gibt es Bindungen in den Daten, d.h. gibt es übereinstimmende Werte, so hängt die Verteilung vom speziellen Muster dieser Bindungen ab und die Berechnung ist aufwendiger. *`wilcox.test()`* greift in diesem Fall auf Approximationen zurück. Zur exakten (im Gegensatz zur approximativen) Auswertung stehen jedoch die entsprechenden Algorithmen ebenfalls zur Verfügung. Dazu benötigt man *`libary(coin)`*. Die exakte Variante des Wilcoxon-Tests findet sich dort etwa als *`wilcox_test()`*.

Auf den Rängen basierende verteilungsunabhängige Verfahren zu charakterisieren und mit den früher vorgestellten verteilungsunabhängigen Monte-Carlo-Verfahren und deren Varianten zu vergleichen ist ein klassischer Teil der Statistik. Literatur dazu findet man unter den Schlagworten “Rangtests” oder “verteilungsfreie Verfahren”. Zusätzliche R-Funktionen finden sich in *libary(coin)* sowie in einigen speziellen Paketen.

Natürlich stellt sich die Frage nach dem Informationsverlust. Wenn wir uns auf die Daten beschränken und keine oder geringe Verteilungsannahmen machen, haben wir weniger Information als in einem Modell mit expliziten Verteilungsannahmen. Wenn wir die Daten auf die Ränge reduzieren, verschenken wir zusätzlich möglicherweise Information. Dieser Informationsverlust kann z.B. durch die asymptotische relative Effizienz gemessen werden. Dies ist (asymptotisch) der Stichprobenumfang eines optimalen Tests, der benötigt wird, eine vergleichbare Güte wie ein konkurrierender Test zu erreichen. Beim Wilcoxon-Test unter Normalverteilung hat dies einen Wert von 94%. Gilt also die Normalverteilungsannahme, so benötigt der (optimale) t-Test nur 94% des Stichprobenumfangs, die der Wilcoxon-Test benötigt. 6% des Stichprobenumfangs sind die Kosten für die Reduzierung auf Ränge. Gilt die Normalverteilungsannahme nicht, so kann der t-Test möglicherweise zusammenbrechen. Der Wilcoxon-Test bleibt ein valider Test auf die Shift-Alternative.

---

help(wilcox.test)

`wilcox.test` *Wilcoxon Rank Sum and Signed Rank Tests*

Description.

Performs one and two sample Wilcoxon tests on vectors of data; the latter is also known as ‘Mann-Whitney’ test.

Usage.

```
wilcox.test(x, ...)

## Default S3 method:
wilcox.test(x, y = NULL,
            alternative = c("two.sided", "less", "greater"),
            mu = 0, paired = FALSE, exact = NULL, correct = TRUE,
            conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'formula':
wilcox.test(formula, data, subset, na.action, ...)
```

Arguments.

`x`	numeric vector of data values. Non-finite (e.g. infinite or missing) values will be omitted.
`y`	an optional numeric vector of data values.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`. You can specify just the initial letter.
`mu`	a number specifying an optional location parameter.
`paired`	a logical indicating whether you want a paired test.
`exact`	a logical indicating whether an exact p-value should be computed.

`correct`	a logical indicating whether to apply continuity correction in the normal approximation for the p-value.
`conf.int`	a logical indicating whether a confidence interval should be computed.
`conf.level`	confidence level of the interval.
`formula`	a formula of the form `lhs ~ rhs` where `lhs` is a numeric variable giving the data values and `rhs` a factor with two levels giving the corresponding groups.
`data`	an optional data frame containing the variables in the model formula.
`subset`	an optional vector specifying a subset of observations to be used.
`na.action`	a function which indicates what should happen when the data contain `NA`s. Defaults to `getOption("na.action")`.
`...`	further arguments to be passed to or from methods.

Details.

The formula interface is only applicable for the 2-sample tests.

If only `x` is given, or if both `x` and `y` are given and `paired` is `TRUE`, a Wilcoxon signed rank test of the null that the distribution of `x` (in the one sample case) or of `x-y` (in the paired two sample case) is symmetric about `mu` is performed.

Otherwise, if both `x` and `y` are given and `paired` is `FALSE`, a Wilcoxon rank sum test (equivalent to the Mann-Whitney test: see the Note) is carried out. In this case, the null hypothesis is that the distributions of `x` and `y` differ by a location shift of `mu` and the alternative is that they differ by some other location shift.

By default (if `exact` is not specified), an exact p-value is computed if the samples contain less than 50 finite values and there are no ties. Otherwise, a normal approximation is used.

Optionally (if argument `conf.int` is true), a nonparametric confidence interval and an estimator for the pseudomedian (one-sample case) or for the difference of the location parameters `x-y` is computed. (The pseudomedian of a distribution F is the median of the distribution of $(u+v)/2$, where u and v are independent, each with distribution F. If F is symmetric, then the pseudomedian and median coincide. See Hollander & Wolfe (1973), page 34.) If exact p-values are available, an exact confidence interval is obtained by the algorithm described in Bauer (1972), and the Hodges-Lehmann estimator is employed. Otherwise, the returned confidence interval and point estimate are based on normal approximations.

With small samples it may not be possible to achieve very high confidence interval coverages. If this happens a warning will be given and an interval with lower coverage will be substituted.

Value.

A list with class `"htest"` containing the following components:

`statistic`	the value of the test statistic with a name describing it.
`parameter`	the parameter(s) for the exact distribution of the test statistic.
`p.value`	the p-value for the test.
`null.value`	the location parameter `mu`.
`alternative`	a character string describing the alternative hypothesis.
`method`	the type of test applied.
`data.name`	a character string giving the names of the data.
`conf.int`	a confidence interval for the location parameter. (Only present if argument `conf.int = TRUE`.)

`estimate` an estimate of the location parameter. (Only present if argument `conf.int = TRUE`.)

Warning.

This function can use large amounts of memory and stack (and even crash R if the stack limit is exceeded) if `exact = TRUE` and one sample is large (several thousands or more).

Note.

The literature is not unanimous about the definitions of the Wilcoxon rank sum and Mann-Whitney tests. The two most common definitions correspond to the sum of the ranks of the first sample with the minimum value subtracted or not: R subtracts and S-PLUS does not, giving a value which is larger by $m(m+1)/2$ for a first sample of size m. (It seems Wilcoxon's original paper used the unadjusted sum of the ranks but subsequent tables subtracted the minimum.)

R's value can also be computed as the number of all pairs `(x[i], y[j])` for which `y[j]` is not greater than `x[i]`, the most common definition of the Mann-Whitney test.

References.

Myles Hollander & Douglas A. Wolfe (1973), *Nonparametric statistical inference*. New York: John Wiley & Sons. Pages 27–33 (one-sample), 68–75 (two-sample).
Or second edition (1999).

David F. Bauer (1972), Constructing confidence sets using rank statistics. *Journal of the American Statistical Association* **67**, 687–690.

See Also.

`psignrank`, `pwilcox`.

`wilcox.exact` in **exactRankTests** covers much of the same ground, but also produces exact p-values in the presence of ties.

`kruskal.test` for testing homogeneity in location parameters in the case of two or more samples; `t.test` for an alternative under normality assumptions [or large samples]

Examples.

```
## One-sample test.
## Hollander & Wolfe (1973), 29f.
## Hamilton depression scale factor measurements in 9 patients with
##  mixed anxiety and depression, taken at the first (x) and second
##  (y) visit after initiation of a therapy (administration of a
##  tranquilizer).
x <- c(1.83,  0.50,  1.62,  2.48, 1.68, 1.88, 1.55, 3.06, 1.30)
y <- c(0.878, 0.647, 0.598, 2.05, 1.06, 1.29, 1.06, 3.14, 1.29)
wilcox.test(x, y, paired = TRUE, alternative = "greater")
wilcox.test(y - x, alternative = "less")    # The same.
wilcox.test(y - x, alternative = "less",
            exact = FALSE, correct = FALSE) # H&W large sample
                                            # approximation

## Two-sample test.
```

```
## Hollander & Wolfe (1973), 69f.
## Permeability constants of the human chorioamnion (a placental
##  membrane) at term (x) and between 12 to 26 weeks gestational
##  age (y).  The alternative of interest is greater permeability
##  of the human chorioamnion for the term pregnancy.
x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
y <- c(1.15, 0.88, 0.90, 0.74, 1.21)
wilcox.test(x, y, alternative = "g")        # greater
wilcox.test(x, y, alternative = "greater",
            exact = FALSE, correct = FALSE) # H&W large sample
                                            # approximation

wilcox.test(rnorm(10), rnorm(10, 2), conf.int = TRUE)

## Formula interface.
boxplot(Ozone ~ Month, data = airquality)
wilcox.test(Ozone ~ Month, data = airquality,
            subset = Month %in% c(5, 8))
```

Aufgabe 3.12	
	Benutzen Sie den Wilcoxon-Test, um die Resultate des rechts/links *click*-Experiments zu vergleichen.

Aufgabe 3.13	
***	Beim rechts/links *click*-Experiment sind mehrere Effekte vermischt. Einige Probleme: • Die Antwortzeit beinhaltet Reaktionszeit, Zeit für die Grob-Bewegung der Maus, Zeit für die Fein-Adjustierung etc. • Für rechts-links-Bewegungen reicht in der Regel ein Schwenken der Hand aus. Für vorwärts-rückwärts-Bewegungen ist in der Regel eine Arm-Bewegung nötig. Es ist nicht zu erwarten, dass beide vergleichbares statistisches Verhalten haben. • Bei aufeinanderfolgenden Registrierungen kann es zum einen Trainings- zum anderen Ermüdungseffekte geben. Können Sie Experiment und Auswertung so modifizieren, dass Unterschiede in der Reaktionszeit untersucht werden können? Können Sie Experiment und Auswertung so modifizieren, dass Unterschiede in der Genauigkeit der Endposition untersucht werden können?
***	Untersuchen und dokumentieren Sie für sich rechts-links-Unterschiede in der Reaktionszeit und in der Genauigkeit. Formulieren Sie ihr Resultat als Bericht.

Aufgabe 3.14	
	Betrachten Sie als Verteilungsfamilien die Shift/Skalenfamilien von $N(0,1)$ und $t(3)$. Entwerfen Sie ein Szenario, um den Wilcoxon-Test mit dem t-Test jeweils innerhalb dieser Familien zu vergleichen. Führen Sie diesen Test in einer Simulation für Stichprobenumfänge $n_1 = n_2 = 10, 20, 50, 100$ durch und fassen Sie die Resultate zusammen. Führen Sie eine analoge Simulation für die Lognormal-Verteilungen durch.

3.4. Ergänzungen

Lage- und Skalenparameter können auch als Versuch verstanden werden, die Verteilung auf eine Referenzgestalt zu transformieren. Lage- und Skalenparameter erfassen nur lineare Transformationen.

3.4.1. Ergänzung: Box-Cox-Transformationen. Die ***Box -Cox -Transformationen***

$$y^{(\lambda)} = \begin{cases} \frac{y^\lambda - 1}{\lambda} & \text{für} \lambda \neq 0, \\ \log(y) & \text{für} \lambda = 0 \end{cases}$$

sind eine Famile, die so skaliert ist, dass die Logarithmus-Transformation stetig in Potenz-Transformationen eingebettet ist.

Generalisierte lineare Modelle sind so erweitert, dass sie bestimmte Transformationen schon im Modell berücksichtigen können.

Zu diesen Themen finden Sie weiter Information in den Büchern von Venables&Ripley [**VR02**].

3.4.2. Ergänzung: Kolmogoroff-Smirnoff-Test. In Kapitel 1 haben wir den Kolmogoroff-Smirnoff-Test zum Vergleich einer Stichprobe $(X_i)_{i=1,\ldots,n}$ und der zugehörigen empirischen Verteilung F_n mit einer (festen, vorgegebenen) Verteilung F kennengelernt. Die kritische Testgröße ist dabei

$$\sup |F_n - F|.$$

Wir können diesen Test etwas modifizieren, um zwei empirische Verteilungen zu vergleichen. Anstelle der Modellverteilung F tritt nun eine zweite empirische Verteilung G_m von Beobachtungen $(Y_j)_{j=1,\ldots,m}$ mit zu Grunde liegender (unbekannter) Verteilung G. Die kritische Testgröße ist dann

$$\sup |F_n - G_m|.$$

Der darauf basierende Test ist in der Literatur als 2-Stichproben-Kolmogoroff-Smirnoff-Test zu finden. Dieser Test korrespondiert zum PP-Plot und erlaubt es, Bänder zum PP-Plot zu konstruieren.

Wir können Bänder auch durch Simulation bestimmen. Im Gegensatz zum 1-Stichproben-Test haben wir jetzt keine vorgegebene Verteilung, aus der wir simulieren können. Unter der Hypothese, dass die Verteilungen F und G sich nicht unterscheiden, verhält sich jedoch bei unabhängigen Beobachtungen der gemeinsame Vektor $(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ wie ein Vektor von $n + m$ unabhängigen Zufallszahlen mit identischer Verteilung $F = G$. Bei gegebenen Daten kann diese Beziehung zur Simulation genutzt werden. Durch eine Permutation π der Indizes erzeugt man aus dem Vektor $Z = (X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ einen

neuen Vektor Z' mit $Z'_i = Z_{\pi(i)}$. Die ersten n Komponenten benutzen wir als simulierte Werte $(X'_i)_{i=1,\ldots,n}$, die übrigen m Komponenten als simulierte Werte $(Y'_j)_{j=1,\ldots,m}$.

Aufgabe 3.15	
*	Programmieren Sie diesen Algorithmus und ergänzen Sie den PP-Plot durch simulierte PP-Plots für eine kleine Anzahl (19?) von Permutation.
	Bestimmen Sie die Permutationsverteilung von $\sup\|F_n - G_m\|$ aus den Simulation und berechnen Sie diesen Wert für die ursprünglichen Daten. Können Sie diesen Vergleich benutzen, um ein Testverfahren zu definieren?
	Der implementierte Kolmogoroff-Smirnoff-Test beinhaltet eine Approximation für den 2-Stichprobenfall. In unserer Simulation wissen wir, dass wir unter der Hypothese simulieren, die Hypothese also zutrifft. Untersuchen Sie die Verteilung des nominellen Niveaus unter den simulierten Bedingungen.

3.5. Statistische Zusammenfassung

Als Leitbeispiel diente in diesem Kapitel der Vergleich von Stichproben. In einfachen Fällen unterscheiden sich Stichproben nur um eine Verschiebung des Mittelwerts. In diesem Fall können die Probleme auf die Ansätze aus Kapitel 2 reduziert werden. In diesem reduzierten Fall stimmen die um den Mittelwert zentrierten Verteilungen überein. Für den allgemeineren Fall, den wir jetzt untersucht haben, gilt diese Vereinfachung nicht. Ein wichtiges Beispiel ist etwa die Untersuchung von Therapie-Studien. Hat eine Behandlung einen homogenen Effekt, so können wir diesen mit den Mitteln von Kapitel 2 untersuchen. Häufig aber gibt es unter einer Behandlung eine Aufspaltung in "Responder" und "Nicht-Responder". Dies geht über die in Kapitel 2 skizzierten Modelle hinaus, und die allgemeineren Ansätze aus diesem Kapitel 3 werden nötig.

Wir haben uns hier auf den Vergleich von zwei Stichproben beschränkt. Die Praxis führt oft auf andere Probleme. So ist ein typischer Fall, dass eine neue Behandlung mit einer bekannten Referenz-Behandlung verglichen werden soll, wobei für die neue Behandlung nur eine Stichprobeninformation, für die Referenz-Behandlung aber umfassendere Vorinformation bereit steht. Oder eine Referenz-Behandlung soll mit einer Serie von Alternativ-Behandlungen verglichen werden. Diese Probleme gehen über den Rahmen unserer Einführung hinaus. Hier kann nur auf weiterführende Literatur, z.B [**Mil81**] verwiesen werden.

Literatur und weitere Hinweise:

[**VR02**] Venables, W.N.; Ripley, B.D. (2002): Modern Applied Statistics with S. Heidelberg: Springer

[**VR00**] Venables, W.N.; Ripley, B.D. (2000): S Programming. Heidelberg: Springer

[**Mil81**] Miller, R. G. (1981): Simultaneous Statistical Inference. Heidelberg: Springer

KAPITEL 4

1, 2, 3, Infinity

Wenn wir von einer Dimension zu höheren Dimensionen gehen, gibt es sowohl für die theoretische Untersuchung als auch für die grafische Darstellung neue Herausforderungen. Die linearen Modelle können wieder als warnendes oder leitendes Beispiel dienen.

Die Herausforderungen können von ernsthaften Problemen stammen. So können Verteilungen auf höherdimensionalen Räumen selbst unter Regularitätsvoraussetzungen unüberschaubar komplex sein. Die Klassifikations- und Identifikationsprobleme für Funktionen und Räume aus Analysis und Geometrie geben einen Vorgeschmack davon, was bei der Untersuchung von Wahrscheinlichkeitsverteilungen zu bewältigen ist. Daneben gibt es hausgemachte Probleme: Eigentore, die durch selbstgetroffene Wahlen erst erzeugt werden.

Ein Beispiel für hausgemachte Probleme kann an linearen Modellen illustriert werden. Die Interpretation des Gauß-Markoff-Schätzers als lineare Projektion zeigt, dass nur scheinbar Koeffizienten für einzelne Regressoren geschätzt werden. Eigentlich wird ein Vektor im von den Regressoren aufgespannten Raum geschätzt; die Zuordnung zu den einzelnen Regressoren ist dann nur lineare Algebra. Diese hängt nicht von dem Einfluss des einzelnen Regressors ab, sondern von der gemeinsamen Geometrie der Regressoren. Nur wenn die Regressoren eine Orthogonalbasis bilden, gibt es eine direkte Interpretation der Koeffizienten.

Wird im linearen Modell die Liste der Regressoren z.B. dupliziert, so ändert sich der Raum nicht. Die Rechnungen in Koordinaten werden etwas komplizierter, weil die Regressoren nun auf keinen Fall eine Basis bilden, aber von einem abstrakten Standpunkt bleibt die Situation unverändert. Gibt es aber kein echtes Duplikat, sondern geringfügige Abweichungen (durch minimale “Fehler”, Rundungen, Transformationen), so ändert sich die Situation drastisch. Für den Gauß-Markoff-Schätzer ist nur der von den Regressoren aufgespannte Raum relevant, und selbst durch minimale Änderungen im Duplikat kann sich dessen Dimension verdoppeln. Dies ist ein Beispiel für ein hausgemachtes Problem.

Dies und andere Beispiele sind ein Grund, die Beziehungen zwischen den Variablen genauer zu untersuchen. Bei der Regression etwa betrifft dies nicht nur die Beziehung zwischen Respons und Regressor, sondern, wie durch das letzte Beispiel illustriert, auch die Beziehungen zwischen den Regressoren.

Um die Verbindung zu den Regressionsproblemen zu halten und auf die Erfahrungen in diesem Bereich zurückzugreifen, betten wir formal die Regressionsprobleme in einen allgemeineren Rahmen ein. Bei der Regression hatten wir eine herausgehobene Variable, die Respons Y, deren Verteilung in Abhängigkeit von den Werten der übrigen Variablen, der Regressoren X, modelliert werden sollte. Wir fassen jetzt Respons und Regressor zu einem Datenvektor $\mathrm{Z} = (\mathrm{Y}; \mathrm{X})$ zusammen und werden auch die gemeinsame Verteilung von Z diskutieren. Wir finden das Regressionsproblem in diesem allgemeineren Rahmen wieder. Beim Regressionsproblem suchten wir nach einem Schätzer für die Mittelwertsfunktion m im Modell

$$Y = m(X) + \varepsilon.$$

Im allgemeineren Rahmen berücksichtigen wir eine gemeinsame Verteilung von X und Y. Das Regressionsmodell wird damit zum Modell

$$Y = E(Y|X) + \varepsilon$$

und wir haben zunächst die Identifizierung $m(X) = E(Y|X)$.

Wenn wir tatsächlich am ursprünglichen Regressionsmodell interessiert sind, müssen wir weitere Arbeit leisten. Eine Schätzung des bedingten Erwartungswerts $E(Y|X)$ ist nicht dasselbe wie die Schätzung einer Regressionsfunktion $m(X)$. Bei dem Regressionsproblem haben wir keine Annahmen über die Verteilung von X gemacht. Um von $E(Y|X)$ (oder einem Schätzer dafür) auf $m(X)$ zurückzuschließen, müssen wir überprüfen, dass die Schätzung von Verteilungsannahmen über X unabhängig ist. Für unsere augenblicklichen Zwecke ist diese Unterscheidung aber nicht relevant. Wir können uns eine Ignoranz auf Zeit erlauben.

4.1. Nichtlinearität und Dimensionen

Haben wir im wentlichen lineare Strukturen, so können wir oft auch höher–dimensionale Strukturen mit Methoden analysieren, die für eindimensionale Modelle entwickelt sind. Wir müssen die Methoden evtl. modifizieren oder iteriert anwenden. Sie helfen uns jedoch, die wesentlichen Merkmale zu erkennen. Sie versagen jedoch, wenn sich höhere Dimensionalität mit Nichtlinearität verbindet.

4.1.1. Spitzen-Nichtlinearität. Das einfachste Beispiel kann im Hinblick auf physikalische Anwendungen illustriert werden. In physikalischen Systemen hängen Wahrscheinlichkeitsverteilungen oft mit Energiezuständen zusammen; (lokale) Minima der Energie entsprechen dabei den Moden der Verteilung. Ein typischer Zusammenhang ist: verhält sich die Energie wie $\varphi(y)$, so verhält sich die Verteilung nach Standardisierung wie $e^{-\varphi(y)}$. Ist $\varphi(y)$ in der Nähe des Minimums quadratisch, so erhalten wir (bis auf Skalentransformation) Verteilungen aus der Familie der Normalverteilungen.

Die Differentialtopologie lehrt uns, dass auch bei kleinen Störungen oder Variationen dieses qualitative Bild erhalten bleibt. Die Energie bleibt zumindest lokal approximativ quadratisch, und die Normalverteilungen bleiben zumindest approximativ eine geeignete Verteilungsfamilie.

Das Verhalten ändert sich drastisch, wenn das Potential sich lokal wie y^4 verhält. Schon geringe Variationen können dazu führen, dass das Potential lokal quadratisch ist. Aber sie können auch dazu führen, dass das lokale Minimum aufbricht und zu zwei Minima führt. Das typische Bild ist von der Gestalt

$$\varphi(y; u, v) = y^4 + u \cdot y^2 + v \cdot y. \tag{4.1}$$

Dabei sind die Variationen durch die Parameter u, v repräsentiert. Am einfachsten lässt sich Situation dynamisch interpretieren: wir stellen uns vor, dass u, v äußere Parameter sind, die sich verändern können. Dieses Bild kennen wir von der magnetischen Hysterese: y gibt die Magnetisierung in einer Richtung an, u spielt die Rolle der Temperatur; v die eines äußeren Magnetfelds. Bei hoher Temperatur folgt die Magnetisierung direkt dem äußeren Magnetfeld. Sinkt die Temperatur, so zeigt das Material Gedächtnis: die Magnetisierung hängt nicht nur vom äußeren Magnetfeld ab, sondern auch von der vorhergehenden Magnetisierung.

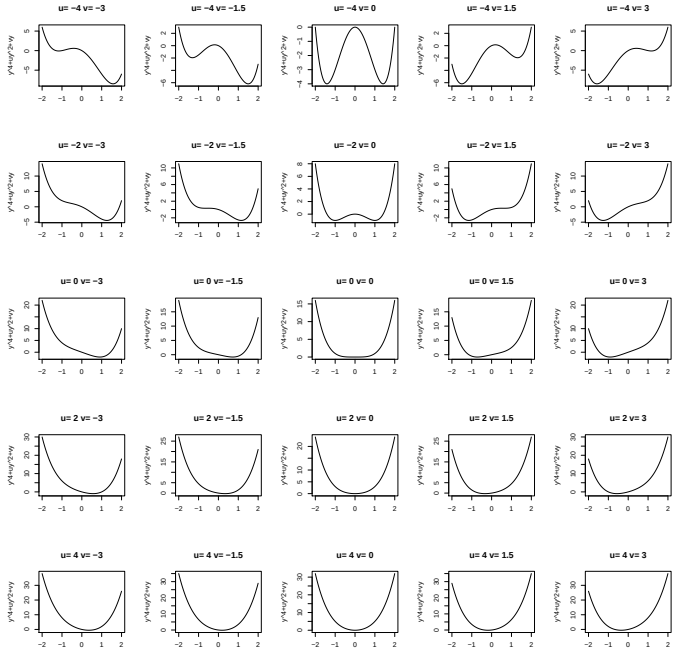


ABBILDUNG 4.1. Entfaltung von y^4: $\varphi(y;u,v) = y^4 + u \cdot y^2 + v \cdot y$

Ähnliche "Gedächtniseffekte" kennen wir auch in anderen Bereichen. Man stelle sich einen Markt vor mit Preisen y, Kosten v und einem "Konkurrenzdruck" u. Bei ausreichender Konkurrenz folgen die Preise (mehr oder weniger) den Kosten bei sonst gleichen Bedingungen. Bei Monopol-Situationen scheinen die Preise ein Gedächtnis zu haben: sind sie einmal gestiegen, so sinken sie erst, wenn die Kosten drastisch reduziert sind.

Die in Formel 4.1 angegebenen "Entfaltung" des Potentials y^4 hat eine typische Form. Aus

$$\varphi'(y;u,v) = 4y^3 + 2u \cdot y + v = 0 \tag{4.2}$$

erhält man die kritischen Punkte (siehe Abb 4.2).

Projiziert auf die u,v-Ebene gibt dies eine Spitze (engl.: "cusp")

$$u = -\frac{27}{8}v^2$$

Bei Parametern im inneren dieser Spitze gibt es zwei lokale Minima; außerhalb der Spitze gibt es nur einen Extremalwert.

Die diesen Potentialen entsprechenden Verteilungen sind – bis auf Skalentransformation zur Normalisierung –

$$p(y;u,v) \propto e^{-(y^4+u\cdot y^2+v\cdot y)} \tag{4.3}$$

Die Struktur der Potentiale spiegelt sich auch in den entsprechenden Verteilungen wieder; der exponetielle Abfall macht allerdings die kritische Grenze etwas komplizierter.

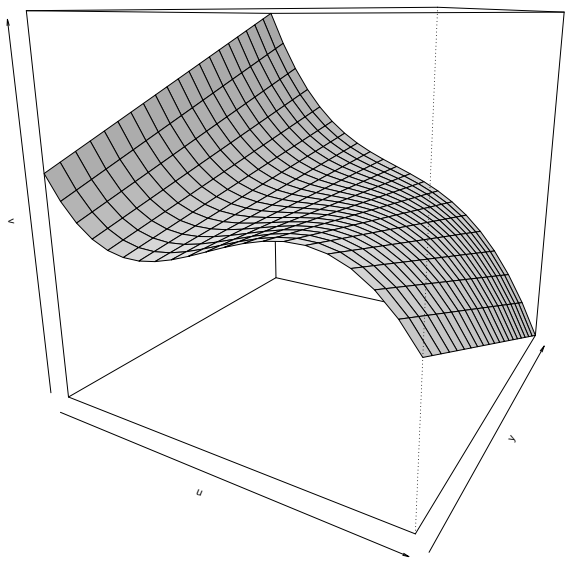


ABBILDUNG 4.2. Kritische Punkte $\varphi'(y; u, v) = 4y^3 + 2u \cdot y + v = 0$

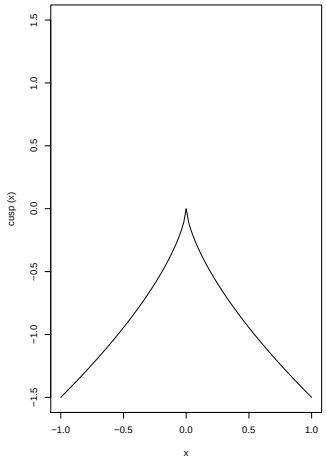


ABBILDUNG 4.3. Grenze zwischen Uni– und Bimodalität im (u,v)–Raum

Die Situation erscheint hier noch harmlos: der Parameterraum (der Raum der Regressoren) x=(u,v) hat nur zwei Dimensionen. Die Verteilung ist eindimensional mit einer glatten Dichte. Aber die Situation kann mit linearen Methoden nur unzureichend erfasst werden. Der typische nichtlineare Effekt wird nicht erkannt, wenn man darauf nicht vorbereitet ist. Erst das Gesamtbild im Dreidimensionalen vermittelt die eigentliche Struktur.

Dieses einfache Beispiel ist eine Herausforderung. Wie kann eine derartige Struktur diagnostiziert werden?

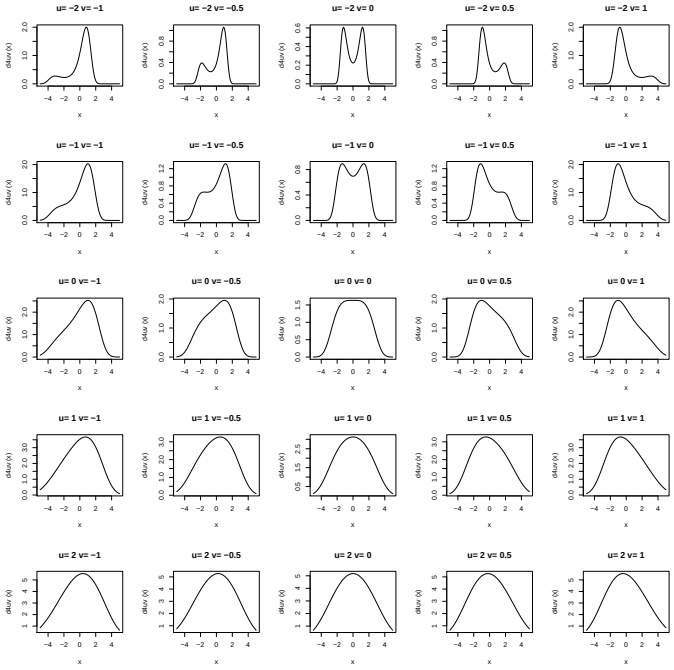


ABBILDUNG 4.4. $p(y; u, v) \propto e^{-(y^4+u\cdot y^2+v\cdot y)}$

Aufgabe 4.1	
	Schreiben Sie eine Funktion `dx4exp(x,u,v)`, die die zentrierte Wahrscheinlichkeitsdichte zu (4.3) berechnet. Dazu müssen Sie die Dichte aus (4.3) integrieren, um die Normierungskonstante zu bestimmen, und den Erwartungswert berechnen, um die Dichte zu zentrieren. Benutzen Sie für beides eine numerische Integration mit `integrate()`.
***	Simulieren Sie zu Werten u, v auf einem Gitter in $u = -2 \ldots 2$ und $v = -1 \ldots 1$ je 100 Zufallszahlen aus `dx4exp(x,u,v)`. Untersuchen Sie diese mit den Methoden aus Kapitel 2. Können Sie Hinweise auf nicht–lineare Abhängigkeit erkennen? Ist die Bimodalität erkennbar? Wie weit können Sie die Struktur identifizieren?

4.2. Koordinatensysteme

Koordinatensystem sind nicht kanonisch vorgegeben. Dies trifft schon auf univariate Probleme zu. Bei univariaten Problemen können wir jedoch Koordinatensysteme relativ einfach transformieren. Die Modellierung der Fehlerverteilung einerseits und die Transformation der Daten auf eine Standard-Verteilung sind in gewisser Weise austauschbar.

In mehrdimensionalen Situationen sind geeignete Transformationsfamilien bisweilen nicht verfügbar oder nicht zugänglich, und die Struktur des Problem kann kritisch von der Wahl geeigneter Koordinaten abhängig sein. Hier hat eine sachorientierte Wahl der Koordinatendarstellung oft den Vorzug vor automatischen Selektionen.

Dieses kann an Anderson's Iris-Datensatz illustriert werden. Der Datensatz hat fünf Dimensionen: vier quantitative Variable (Länge und Breite von Blütenblatt (engl. petal) und Kelchblatt (engl. sepal) von Iris-Blüten) und eine kategorielle Variable (die Spezies: iris setosa canadensis, iris versicolor, iris virginica). Gesucht ist eine Klassifikation der Spezies anhand der vier quantiativen Variablen.

TABELLE 4.2. Iris Spezies. Photos: The Species Iris Group of North America. Mit freundlicher Genehmigung.

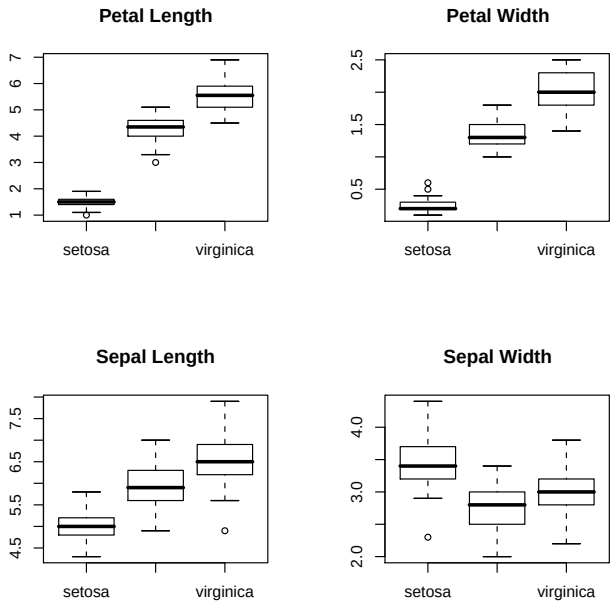
		
Iris setosa	Iris versicolor	Iris virginica

Um eine erste Übersicht zu bekommen ist es naheliegend, die vier Variablen getrennt nach Spezies zu betrachten. Die Standard-Konventionen von R machen dies umständlich. Die Spezies ist eine kategorielle Variable. Dies veranlasst R, bei der `plot`-Funktion von einer Punkt-Darstellung zu Box&Whisker-Plots über zu gehen.

Beispiel 4.1:

Eingabe

```
oldpar <- par(mfrow = c(2, 2))
plot(iris$Species, iris$Petal.Length, ylab = "",
    main = "Petal Length")
plot(iris$Species, iris$Petal.Width, ylab = "", main = "Petal Width")
plot(iris$Species, iris$Sepal.Length, ylab = "",
    main = "Sepal Length")
plot(iris$Species, iris$Sepal.Width, ylab = "", main = "Sepal Width")
par(oldpar)
```



Wir könnten die R-Funktionen modifizieren, um das gewünschte Ergebnis zu erhalten. Anstatt uns diese Mühe zu machen, greifen wir lieber auf die *grid*-Grafik zurück, die mit dem Paket *lattice* eine weitgehende Unterstützung für multivariate Darstellungen liefert. *grid* ist dabei die Basis. Das ursprüngliche Grafiksystem von R implementiert ein Modell, dass an der Vorstellung von Stift und Papier orientiert ist. Ein Grafik-Port (Papier) wird eröffnet und darauf werden Linien, Punkte/Symbole gezeichnet. *grid* ist ein zweites Grafiksystem, dass an einem Kamera/Objekt-Modell orientiert ist. Grafische Objekte in unterschiedlicher Lage und Richtung werden in einem visuellen Raum abgebildet. Auf der *grid* baut *lattice* auf. In `<http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/>` sind die Grundideen zur Visualisierung multidimensionaler Daten dokukumentiert, die in *lattice* implementiert sind.

Mit *grid* und *lattice* benutzen wir die Funktion *stripplot*. Weil bei der gegebenen Messgenauigkeit Werte vielfach auftreten, benutzen wir ein 'jitter': wir 'verwackeln' vielfache Werte, um sie getrennt darzustellen.

Beispiel 4.2:

Eingabe

```
library("lattice")
print(stripplot(Petal.Length ~ Species, data = iris,
    jitter = TRUE, ylab = "", main = "Petal Length"))
```

Eingabe

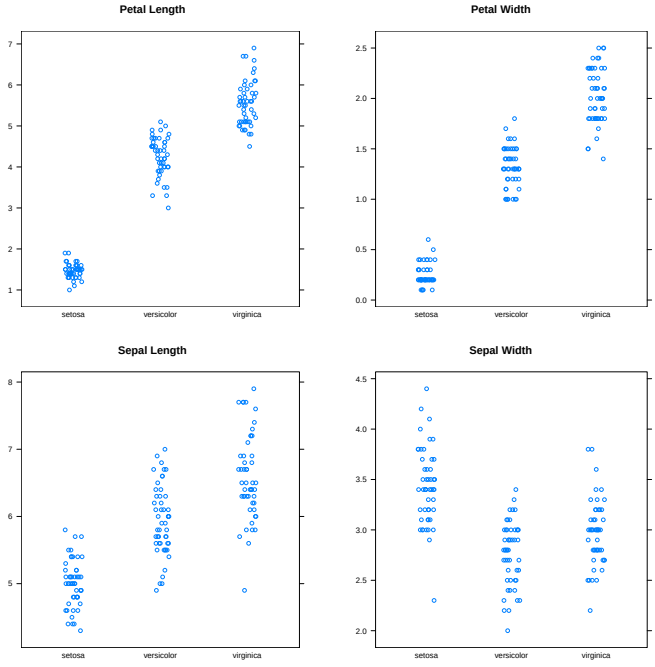
```
print(stripplot(Petal.Width ~ Species, data = iris,
    jitter = TRUE, ylab = "", main = "Petal Width"))
```

Eingabe

```
print(stripplot(Sepal.Length ~ Species, data = iris,
    jitter = TRUE, ylab = "", main = "Sepal Length"))
```

Eingabe

```
print(stripplot(Sepal.Width ~ Species, data = iris,
    jitter = TRUE, ylab = "", main = "Sepal Width"))
```



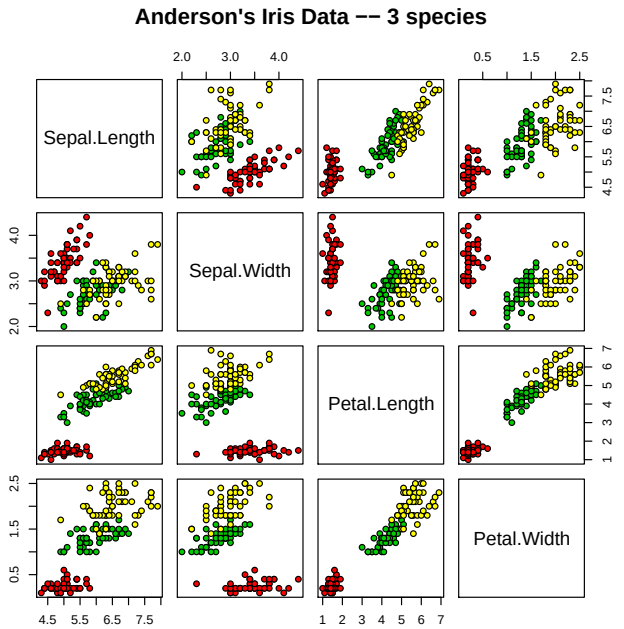
Die eindimensionalen Randverteilugen geben noch wenig Hinweis darauf, wie die drei Gruppen zu trennen sind.

Eine Übersicht über alle zweidimensionalen marginalen Verteilungen erhält man mit `pairs()`. Wir benutzen hier eine Farbcodierung, um die unterschiedlichen Spezies zu kennzeichen. Auch die zweidimensionale Darstellung hilft wenig weiter.

Beispiel 4.3:

Eingabe

```
pairs(iris[1:4], main = "Anderson's Iris Data -- 3 species",
    pch = 21, bg = c("red", "green3", "yellow")[unclass(iris$Species)])
```

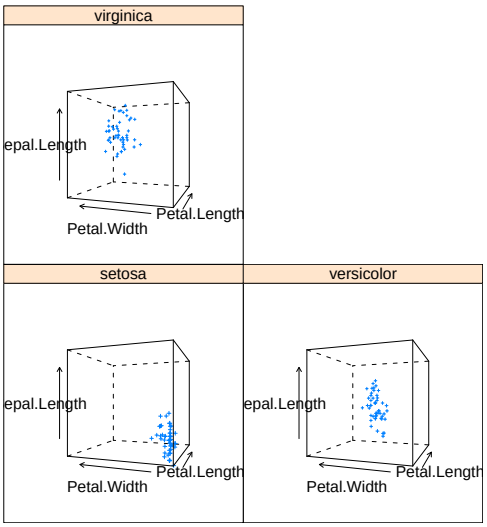


Die Funktion `cloud()` aus der Bibliothek `lattice` kann einen Eindruck von dreidimensionalen Aspekten geben, wieder mit einer Farbcodierung, um die unterschiedlichen Spezies zu kennzeichen.:

Beispiel 4.4:

Eingabe

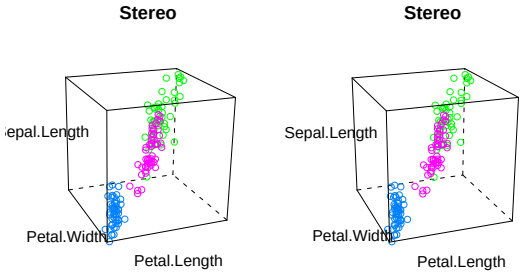
```
library("lattice")
print(cloud(Sepal.Length ~ Petal.Length * Petal.Width |
    Species, data = iris, screen = list(x = -90,
    y = 70), distance = 0.4, zoom = 0.6))
```



Beispiel 4.5:

Eingabe

```
par.set <- list(axis.line = list(col = "transparent"),
    clip = list(panel = FALSE))
print(cloud(Sepal.Length ~ Petal.Length * Petal.Width,
    data = iris, cex = 0.8, groups = Species, main = "Stereo",
    screen = list(z = 20, x = -70, y = 3), par.settings = par.set),
    split = c(1, 1, 2, 1), more = TRUE)
print(cloud(Sepal.Length ~ Petal.Length * Petal.Width,
    data = iris, cex = 0.8, groups = Species, main = "Stereo",
    screen = list(z = 20, x = -70, y = 0), par.settings = par.set),
    split = c(2, 1, 2, 1))
```



Mit formalen Methoden wie der Diskriminanzanalyse kann die Klassifikation anhand der ursprünglichen Variablen gefunden werden. Die Trennung der Spezies ist nicht trivial.

Die ursprünglichen Variablen repräsentieren jedoch nur den Aspekt der Daten, der technisch am einfachsten erhebbar ist. Biologisch gesehen würde man jedoch anders parametrisieren: die Variablen spiegeln Größe und Form der Blätter wieder. Eine erste Approximation wäre

$$area = length \cdot width \tag{4.4}$$

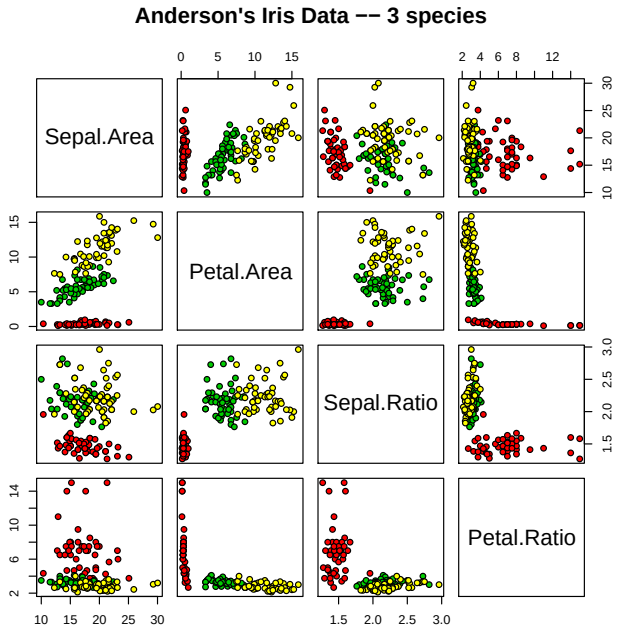
$$aspectratio = length/width. \tag{4.5}$$

Damit erhält man die Darstellung

Beispiel 4.6:

Eingabe

```
iris$Sepal.Area <- iris$Sepal.Length * iris$Sepal.Width
iris$Petal.Area <- iris$Petal.Length * iris$Petal.Width
iris$Sepal.Ratio <- iris$Sepal.Length/iris$Sepal.Width
iris$Petal.Ratio <- iris$Petal.Length/iris$Petal.Width
pairs(iris[6:9], main = "Anderson's Iris Data -- 3 species",
    pch = 21, bg = c("red", "green3", "yellow")[unclass(iris$Species)])
```



In der Marginalverteilung der Flächengrössen sind die Spezies fast vollständig getrennt. In diesen mehr biologischen Koordinaten sieht man, dass zur Klassifikation allein die Blattfläche zur Klassifikation ausreicht - mit zwei Grenzfällen. Jedes kompliziertere formale Verfahren muss sich mit dieser trivialen Klassifikationsregel erst einmal messen.

4.3. Projektionen und Schnitte

Eine Strategie zur Analyse höherdimensionaler Probleme ist die Dimensionsreduktion. Geometrisch gibt es zwei wesentliche Werkzeuge: Projektionen und Schnitte. Das Wechselspiel von Projektionen und Schnitten ist in [**FB94**] untersucht. Statistisch führen Projektionen zu Marginalverteilungen und Schnitte zu bedingten Verteilungen.

Sind der Anwendung angepasste Koordinatensysteme gewählt, so können bisweilen Projektionen bzw. Schnitte entlang der Koordinatenachsen helfen. Beispiel 4.6 benutzt die Art als Schnittvariable und zeigt 2d-Projektionen eins 3d-Unterraums (die Kelchblatt-Breite `Sepal.Width` ist nicht gezeigt. Beispiel 4.5 benutzt die Art als Schnittvariable, die durch Farbe codiert ist, und zeigt alle marginalen 2d-Verteilungen.

Ohne angepasste Koordinatensysteme ist eine umfassende Suche nach interessanten Projektionen und Schnitten nötig. Die Anzahl der Möglichkeiten steigt rasch mit der Dimension. Zu Illustration: Um einen Kubus zu identifizieren müssen zumindest die Eckpunkte erkannt werden. In d Dimensionen sind dies 2^d Eckpunkte. Die Anzahl steigt exponentiell mit der Dimension. Dies ist ein Aspekt des als "Curse of dimension" bekannten Problems.

4.4. Marginale Verteilungen und Scatterplot-Matrizen

Wie wir gesehen haben gibt R eine einfache Möglichkeit, zumindest die marginalen Beziehungen (d.h. die Beziehungen zwischen je zwei Variablen) darzustellen. In einer Scatterplot-Matrix erscheinen alle paarweisen Plots von je zwei Variablen.

---

help(pairs)

`pairs` *Scatterplot Matrices*

Description.

A matrix of scatterplots is produced.

Usage.

```
pairs(x, ...)

## S3 method for class 'formula':
pairs(formula, data = NULL, ..., subset,
      na.action = stats::na.pass)

## Default S3 method:
pairs(x, labels, panel = points, ...,
      lower.panel = panel, upper.panel = panel,
      diag.panel = NULL, text.panel = textPanel,
      label.pos = 0.5 + has.diag/3,
      cex.labels = NULL, font.labels = 1,
      row1attop = TRUE, gap = 1)
```

Arguments.

`x`	the coordinates of points given as numeric columns of a matrix or dataframe. Logical and factor columns are converted to numeric in the same way that `data.matrix` does.
`formula`	a formula, such as `~ x + y + z`. Each term will give a separate variable in the pairs plot, so terms should be numeric vectors. (A response will be interpreted as another variable, but not treated specially, so it is confusing to use one.)
`data`	a data.frame (or list) from which the variables in `formula` should be taken.
`subset`	an optional vector specifying a subset of observations to be used for plotting.

`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is to pass missing values on to the panel functions, but `na.action = na.omit` will cause cases with missing values in any of the variables to be omitted entirely.
`labels`	the names of the variables.
`panel`	`function(x,y,...)` which is used to plot the contents of each panel of the display.
`...`	arguments to be passed to or from methods. Also, graphical parameters can be given as can arguments to `plot` such as `main`. `par(öma")` will be set appropriately unless specified.
`lower.panel, upper.panel`	separate panel functions to be used below and above the diagonal respectively.
`diag.panel`	optional `function(x, ...)` to be applied on the diagonals.
`text.panel`	optional `function(x, y, labels, cex, font, ...)` to be applied on the diagonals.
`label.pos`	`y` position of labels in the text panel.
`cex.labels, font.labels`	graphics parameters for the text panel.
`row1attop`	logical. Should the layout be matrix-like with row 1 at the top, or graph-like with row 1 at the bottom?
`gap`	Distance between subplots, in margin lines.

Details.

The ijth scatterplot contains `x[,i]` plotted against `x[,j]`. The "scatterplot" can be customised by setting panel functions to appear as something completely different. The off-diagonal panel functions are passed the appropriate columns of `x` as `x` and `y`: the diagonal panel function (if any) is passed a single column, and the `text.panel` function is passed a single `(x, y)` location and the column name.

The graphical parameters `pch` and `col` can be used to specify a vector of plotting symbols and colors to be used in the plots.

The graphical parameter `oma` will be set by `pairs.default` unless supplied as an argument.

A panel function should not attempt to start a new plot, but just plot within a given coordinate system: thus `plot` and `boxplot` are not panel functions.

By default, missing values are passed to the panel functions and will often be ignored within a panel. However, for the formula method and `na.action = na.omit`, all cases which contain a missing values for any of the variables are omitted completely (including when the scales are selected). (The latter was the default behaviour prior to R 2.0.0.)

Author(s).

Enhancements for R 1.0.0 contributed by Dr. Jens Oehlschlaegel-Akiyoshi and R-core members.

References.

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

Examples.

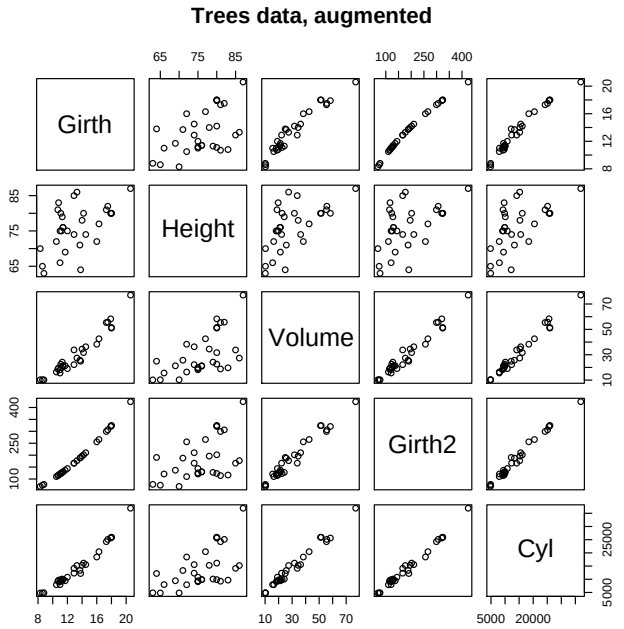
```
pairs(iris[1:4], main = "Anderson's Iris Data -- 3 species",
      pch = 21, bg = c("red", "green3", "blue")[unclass(iris$Species)])

## formula method
pairs(~ Fertility + Education + Catholic, data = swiss,
      subset = Education < 20, main = "Swiss data, Education < 20")

pairs(USJudgeRatings)

## put histograms on the diagonal
panel.hist <- function(x, ...)
{
    usr <- par("usr"); on.exit(par(usr))
    par(usr = c(usr[1:2], 0, 1.5) )
    h <- hist(x, plot = FALSE)
    breaks <- h$breaks; nB <- length(breaks)
    y <- h$counts; y <- y/max(y)
    rect(breaks[-nB], 0, breaks[-1], y, col="cyan", ...)
}
pairs(USJudgeRatings[1:5], panel=panel.smooth,
      cex = 1.5, pch = 24, bg="light blue",
      diag.panel=panel.hist, cex.labels = 2, font.labels=2)

## put (absolute) correlations on the upper panels,
## with size proportional to the correlations.
panel.cor <- function(x, y, digits=2, prefix="", cex.cor)
{
    usr <- par("usr"); on.exit(par(usr))
    par(usr = c(0, 1, 0, 1))
    r <- abs(cor(x, y))
    txt <- format(c(r, 0.123456789), digits=digits)[1]
    txt <- paste(prefix, txt, sep="")
    if(missing(cex.cor)) cex <- 0.8/strwidth(txt)
    text(0.5, 0.5, txt, cex = cex * r)
}
pairs(USJudgeRatings, lower.panel=panel.smooth, upper.panel=panel.cor)
```

Um die Situation besser zu verstehen gehen wir zunächst zu einem einfacheren Regressionsproblem. Der Datensatz “trees” enthält Daten über Stammduchmesser, Höhe und Zimmerholz-Volumen von 31 ehemaligen Kirschbäumen. Die Variable *Girth* dabei ist ein standardisierter Kennwert: der Durchmesser (in Zoll) gemessen in der Höhe 4 Fuß 6”. Details erhält man mit *help(trees)*. Das Holzvolumen, im wesentlichen aus dem Stamm gebildet, ist aus der Sicht der Autoren das Wichtige. Naive Vorstellungen können den Stamm als Zylinder oder Kegel modellieren - auf jeden Fall führen sie zu einem Modell, das den Stammquerschnitt als Parameter enthält; nur die Koeffizienten unterscheiden sich. Bis auf Konstanten ist dies das Quadrat des Durchmessers, und bis auf Konstanten ist das Volumen in diesen Modellen das Produkt aus Querschnitt und Höhe. Wir erweitern den Datensatz entsprechend durch eine zwei abgeleitete Variable $Girth2$ und Cyl.

Beispiel 4.7:

Eingabe

```
data(trees)
trees$Girth2 <- trees$Girth * trees$Girth
trees$Cyl <- trees$Girth2 * trees$Height
pairs(trees, main = "Trees data, augmented")
```



In dieser Situation wissen wir, dass $Girth2$ genau das Quadrat von $Girth$ ist - wir haben die Variable so generiert. Hätten wir diese Information nicht, so wären die Plots von $Girth$ gegen $Girth2$ unsere Informationsquelle, und diese zeigen eine annähernd lineare Gestalt: Die Scatterplots der Zielvariablen gegen die "Regressoren" zeigen eine ähnlich klare lineare Beziehung von $Volume$ gegen Cyl wie gegen $Girth2$ und sind wenig hilfreich. Wenn wir hier an ein lineares Modell denken, so wäre

$$Volume = a_0 + a_1 Cyl + a_2 Girth + a_3 Girth2 + a_4 Height + \varepsilon$$

ein sehr allgemeiner Kandidat. Zu vermuten ist, dass die simple Geometrie hinter dem Beitrag $a_1 Cyl$ den wesentlichen Beitrag beschreibt, wobei a_1 die eigentliche Form widerspiegelt. (Selbstverständlich ist $Volume = 0$, fallls $Girth = 0$ oder $Height = 0$. Aber wir schätzen in einem einfachen Modell aus einem Datenmaterial mit Werten, die weit von null entfernt sind. Wir erwarten oder verlangen nicht, dass die Extrapolation exakt durch den Ursprung geht, sondern erlauben einen konstanten Term a_0.)

Ein wesenlich einfacheres Modell wäre

$$Volume = a_0 + a_1 Cyl + err.$$

Bringt das kompliziertere Modell eine wesentliche Verbesserung gegenüber dem einfachen Modell? Welche Parameter bzw. welche abgeleitete Variable sollten in das Modell einbezogen werden? Die Lehre aus den linearen Modellen ist, dass nicht der Wert des einzelnen

Parameters den Beitrag im Modell bestimmt, sondern dass die durch die Parameter bestimmten Räume die wesentlichen Grössen sind. An dieser Stelle sind angepasste Strategien gefragt. Wir können mit einfachen Modellen beginnen und fragen, ob zusätzliche Parameter einen weiteren Beitrag liefern. Dadurch erreichen wir einen besseren Fit, aber erhöhen die Varianz unserer Schätzungen. Oder wir können mit einem relativ komplexen Modell beginnen, und fragen, ob wir Parameter fortlassen können. Dadurch wird zwar der Restfehler erhöht, wir gewinnen aber an Verlässlichkeit der Schätzungen.

Beide Strategien führen im abstrakten linearen Regressionsmodell zu einem Vergleich von zwei Modellräumen $\mathscr{M}_{X'} \subset \mathscr{M}_X$. Die entsprechenden Schätzer sind $\pi_{\mathscr{M}_{X'}}(Y)$ und $\pi_{\mathscr{M}_X}(Y)$. Die Beziehung zwischen beiden wird klar, wenn wir die orthogonale Zerlegung $\mathscr{M}_X = \mathscr{M}_{X'} \oplus L_X := M_0, L_X := \mathscr{M}_X \ominus \mathscr{M}_{X'}$ von $\mathscr{M}_X$ wählen. Dann ist $\pi_{\mathscr{M}_X}(Y) = \pi_{\mathscr{M}_{X'}}(Y) + \pi_{L_X}(Y)$.

4.5. Partielle Residuen und Added-Variable-Plots

In der Regression sind $\mathscr{M}_{X'}$ und $\mathscr{M}_X$ Räume, die von den Regressor-Variablenvektoren aufgespannt werden. In unserer Situation interessiert uns der Spezialfall

$$X' = \text{span}(X'_1, \ldots, X'_p); X = \text{span}(X_1, \ldots, X_p)$$

mit $p > p'$. Dann wird aber L_X aufgespannt von den Vektoren

$$R_{p'+1} = X_{p'+1} - \pi_{\mathscr{M}'_X}(X_{p'+1}), \ldots, R_p = X_p - \pi_{\mathscr{M}_{X'}}(X_p).$$

Wenn wir also (formal) eine lineare Regression der zusätzlichen Regressoren nach den bereits in X' enthaltenen durchführen, sind die dabei entstehenden Residuen ein Erzeugendensystem für L_X. Eine weitere Regression von Y nach diesen Residuen liefert uns den Term $\pi_{L_{X'}}(Y)$, der den Unterschied zwischen den Modellen beschreibt. Nach Konstruktion wissen wir, dass $\pi_{\mathscr{M}_{X'}}(Y)$ orthogonal zu L_X ist. Bei dieser zweiten Regression wird deshalb dieser Anteil auf null abgebildet. Wir können diesen Anteil gleich eliminieren und uns auf die Regression von $Y' = Y - \pi_{\mathscr{M}_{X'}}(Y)$ nach $R_{p'+1}, \ldots, R_p$ beschränken.

Im Beispiel der "tree"-Daten ist das Modell

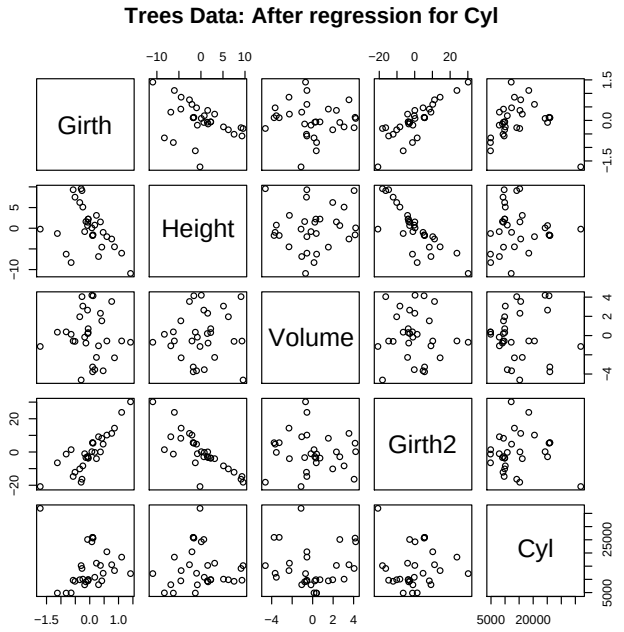
$$Volume = a_0 + a_1 Cyl + \varepsilon$$

ein sehr plausibles Ausgangsmodell, und die Strategiewahl ist einfach: wir untersuchen, ob zusätzliche Parameter in das Modell aufgenommen werden sollten. Anstelle der Scatterplot-Matrix der ursprünglichen Daten betrachten wir die Scatterplots der (formalen) Residuen aus diesem einfachen Modell. Diese Scatterplots werden ***Added-Variable-Plots*** genannt.

Beispiel 4.8:

Eingabe

```
treesresid <- trees
treesresid$Girth <- lm(Girth ~ Cyl, data = trees)$residuals
treesresid$Height <- lm(Height ~ Cyl, data = trees)$residuals
treesresid$Volume <- lm(Volume ~ Cyl, data = trees)$residuals
treesresid$Girth2 <- lm(Girth2 ~ Cyl, data = trees)$residuals
pairs(treesresid, main = "Trees Data: After regression for Cyl")
```



Die Scatterplot-Matrix der Residuen nach Regression für Cyl zeigt keine erkennbare lineare Struktur in der Beziehung von "Volume" zu einer der anderen (adjustierten) Variablen. Im Rahmen des linearen Modell erscheint das einfache Modell

$$Volume = a_0 + a_1\, Cyl + \varepsilon$$

als ausreichend. Eine erkennbare (nichtlineare) Beziehung besteht zwischen den adjustierten Werten für Height und Volume. Bei größerer Höhe zeigen die Residuen für das Volumen eine deutlich größere Variabilität. Eine Keilform ist in den Scatterplots zu sehen. Wenn ein detallierteres Modell gesucht wird, so wäre hier ein Ansatzpunkt.

Um den Unterschied zur Scatterplot-Matrix der Ausgangsdaten zu betonen: lineare Strukturen im Scatterplot der Ausgangsdaten sind ein klarer Hinweis auf lineare Abhängigkeiten. Nichtlineare Strukuren, wie z.B. die Dreiecksgestalt in einigen der Scatterplots können eine entsprechende Abhängigkeit widerspiegeln; sie können aber auch Artefakte sein, die als Folge der Verteilungs- und Korrelationsstruktur der Regressoren auftreten. Sie haben in der Regel keine einfache Deutung. Im Gegensatz dazu sind die Darstellungen in der Matrix der Added-Variable-Plots für lineare Effekte der vorausgehenden Variablen adjustiert. Dadurch hängen sie von der Wahl der Reihenfolge ab, in der Variable einbezogen

werden. Sie korrigieren aber für lineare Effekte, die aus den Korrelationen zu vorausgehenden Variablen kommen. Dadurch wird eine ganze Reihe von Artefakten vermieden und sie können unter Berücksichtigung des Zusammenhangs unmittelbar interpretiert werden.

Aufgabe 4.2	
	Modifizieren Sie die nachfolgende Prozedur *pairslm()* so, dass sie für alle Variablen in der ursprünglichen Matrix *x* die Residuen der Regression nach der neuen Variablen *x$fit* berechnet und eine Scatterplot-Matrix dieser Residuen zeigt. *pairslm <- function(model, x,...)* *{ x$fit <- lm(model, x)$fit; pairs(x,...)}* Fügen Sie auch Titel, Legenden etc. hinzu. Benutzen Sie den "trees"-Datensatz als Beispiel.

Wir haben an diesem Beispiel den Übergang von p' zu $p'+1$ Variablen untersucht (in unserem Fall: $p'=2$ für die Variablen const= 1 und Cyl). Die Scatterplot-Matrix erlaubt uns einen schnellen Überblick über eine (nicht zu) große Zahl von Kandidaten (bei uns drei mögliche zusätzliche Regressoren). Der Übergang von p zu $p-1$, zur Elimination einer Variablen, ist in gewisser Weise dual dazu. Dies entspricht der zweiten Strategie, der schrittweisen Elimination.

Statt eine einzelne Variable als Leitvariable auszuwählen ist es effizienter, Kombinationen von Variablen als synthetische Leitvariablen zu benutzen. Dies geht über den Rahmen dieser Einführung hinaus. Entsprechende Methoden werden in der Theorie als Hauptkomponentenanlyse behandelt und durch die Funktion *princomp()* in der Bibliothek *mva()* bereitgestellt.

Das Beispiel der linearen Modelle lehrt uns, dass die marginalen Beziehungen nur die halbe Wahrheit sind. Anstelle die einzelnen Regressoren zu betrachten, müssen wir im linearen Modell schrittweise orthogonalisieren. Komponentenweise Interpretationen sind damit fragwürdig - sie sind weitgehend von der Reihenfolge abhängig, in der Variablen einbezogen werden. Bei den "tree"-Daten haben wir eine schrittweise Hinzunahme von Variablen als Strategie verfolgt.

In komplexeren Situtationen führen formale Methoden oft nur in die Irre. Handwerkliches Geschick ist hier notwendig. Leider sind die Kennntnisse darüber, wie handwerkliche Eingriffe die Gültigkeit formaler Methoden beeinflussen, noch sehr beschränkt. Deshalb ist es gerade hier wichtig, gewählte Strategien anhand von Simulationen kritisch zu beurteilen.

Scatterplot-Matrizen gehören zu einer Familie von Verfahren, die versuchen, höherdimensionale Probleme durch Projektion auf ausgewählte marginale Aspekte zu analysieren. Die einfachen Scatterplot-Matrizen benutzen dazu die Projektionen auf die durch Paare von Variablen definierten zweidimensionalen Ränder. Dies ist ein erster Ansatz. Anstelle der Variablen haben wir Linearkombinationen benutzt, um uns die Zielräume der Projektion zu definieren: die Räume, die durch Kleinste-Quadrate-Schätzung und schrittweise orthogonale Komplemente bestimmt sind. Das hat zu den Added-Variable-Plots geführt, die wir als marginale Projektionen in einem transformierten Datensatz, den Residuen, erhalten haben. Nachdem die Transformation durchgeführt war, können wir wieder die Standard-Algorithmen für die Scatterplot-Matrix benutzen.

Die Funktion `pairs` kontrolliert nur das "Layout" der Matrix, die Auswahl und Anordnung der Projektionen. Die Darstellung in den Plot-Feldern wird durch den Aufruf gesteuert. Die Default-Belegungen führen dazu, dass in der Diagonale die Namen der Variablen und ausserhalb der Diagonale die paarweisen Scatterplots gezeigt werden.

Aufgabe 4.3	
	Generieren Sie eine Scatterplot-Matrix für den tree-Datensatz, die in der Diagonale die Histogramme der jeweiligen Variablen zeigt. *Hinweis:* Siehe `help(pairs)`.
	Fügen Sie eine Beschriftung mit den Variablennamen hinzu.

Bei nichtlinearen Beziehungen können gemeinsame Abhängigkeiten eine noch größere Bedeutung haben. Im allgemeinen erfordert dies Umsicht bei der Modellbildung. Nichtlineare Beziehungen können in Projektionen versteckt sein. Artefakte der (linearen) Projektion können ein Bild vermittlen, das nicht den ursprünglichen Beziehungen entspricht.

Abstrakt gesprochen sind wir beim Problem, eine höherdimensionale Verteilung zu untersuchen. Projektionen reduzieren die Verteilung auf Randverteilungen. Wo diese aussagekräftig sind, helfen Projektionen weiter. Wenn die Verteilungen nicht durch Randverteilungen eindeutig definiert ist, stoßen Projektionen an Grenzen.

4.6. Bedingte Verteilungen und Coplots

Alternativ können wir höherdimensionale Verteilungen durch Schnitte beschreiben. Abstrakt sind dies bedingte Verteilungen, und sie sind nur dort zuverlässig, wo die Bedingung ein positives Maß hat. Um die Idee der Reduktion auf bedingte Verteilungen auch auf Daten anwenden zu können, dicken wir die Schnitte auf. Abstrakt: anstelle bedingte Verteilungen des Typs $P(\cdot \mid X = x)$ zu untersuchen, betrachten wir $P(\cdot \mid \Omega \, \|X - x\| < \varepsilon)$, wobei ε auch mit x variieren kann. In grafische Darstellungen von Daten verlangt dies eine Serie von Plots, die jeweils nur den durch die Bedingung eingeschränkten Teildatensatz zeigen.

Schnitte und Projektionen sind in gewissem Sinne komplementär: Projektionen zeigen Strukturmerkmale niedriger Dimension. Schnitte sind geeignet, Strukturmerkmale niedriger Co-Dimension zu entdecken. Beide können zur Datenanalyse kombiniert werden.

Als erstes Hilfsmittel stellt R die Möglichkeit bereit, zwei Variablen ***bedingt*** auf eine oder mehre weitere Variable zu analysieren. Als grafische Darstellung dient dazu der Coplot. Er ist eine Variante der Plot-Matrix und zeigt in jedem Feld den Scatterplot zweier Variabler, gegeben die Bedingung. Wie die Bedingungen zu wählen sind, wird durch den Aufruf gesteuert.

Der Coplot kann nun auf bestimmte Muster untersucht werden. Sind die dargestellten Variablen stochastisch unabhängig von den bedingenden Variablen, so zeigen alle Plot-Elemente dieselbe Gestalt. Dargestellte Variable und bedingende Variable können dann entkoppelt werden.

Stimmt die Gestalt überein, aber Ort und Größe variieren, so weist dies auf eine (nicht notwendig lineare) Shift/Skalenbeziehung hin. Additive Modelle oder Varianten davon können benutzt werden, um die Beziehung zwischen dargestellten Variablen und bedingenden Variablen zu modellieren.

Verändert sich bei Variation der Bedingung die Gestalt, so liegt eine wesentliche Abhängigkeitsstruktur oder Interaktion vor, die genauerer Modellierung bedarf.

---

help(coplot)

`coplot` *Conditioning Plots*

Description.

This function produces two variants of the **co**nditioning plots discussed in the reference below.

Usage.

```
coplot(formula, data, given.values, panel = points, rows, columns,
       show.given = TRUE, col = par("fg"), pch = par("pch"),
       bar.bg = c(num = gray(0.8), fac = gray(0.95)),
       xlab = c(x.name, paste("Given :", a.name)),
       ylab = c(y.name, paste("Given :", b.name)),
       subscripts = FALSE,
       axlabels = function(f) abbreviate(levels(f)),
       number = 6, overlap = 0.5, xlim, ylim, ...)
co.intervals(x, number = 6, overlap = 0.5)
```

Arguments.

`formula`	a formula describing the form of conditioning plot. A formula of the form `y ~ x | a` indicates that plots of `y` versus `x` should be produced conditional on the variable `a`. A formula of the form `y ~ x| a * b` indicates that plots of `y` versus `x` should be produced conditional on the two variables `a` and `b`. All three or four variables may be either numeric or factors. When `x` or `y` are factors, the result is almost as if `as.numeric()` was applied, whereas for factor `a` or `b`, the conditioning (and its graphics if `show.given` is true) are adapted.
`data`	a data frame containing values for any variables in the formula. By default the environment where `coplot` was called from is used.
`given.values`	a value or list of two values which determine how the conditioning on `a` and `b` is to take place. When there is no `b` (i.e., conditioning only on `a`), usually this is a matrix with two columns each row of which gives an interval, to be conditioned on, but is can also be a single vector of numbers or a set of factor levels (if the variable being conditioned on is a factor). In this case (no `b`), the result of `co.intervals` can be used directly as `given.values` argument.
`panel`	a `function(x, y, col, pch, ...)` which gives the action to be carried out in each panel of the display. The default is `points`.
`rows`	the panels of the plot are laid out in a `rows` by `columns` array. `rows` gives the number of rows in the array.
`columns`	the number of columns in the panel layout array.

`show.given`	logical (possibly of length 2 for 2 conditioning variables): should conditioning plots be shown for the corresponding conditioning variables (default `TRUE`)
`col`	a vector of colors to be used to plot the points. If too short, the values are recycled.
`pch`	a vector of plotting symbols or characters. If too short, the values are recycled.
`bar.bg`	a named vector with components `"num"` and `"fac"` giving the background colors for the (shingle) bars, for **num**eric and **fac**tor conditioning variables respectively.
`xlab`	character; labels to use for the x axis and the first conditioning variable. If only one label is given, it is used for the x axis and the default label is used for the conditioning variable.
`ylab`	character; labels to use for the y axis and any second conditioning variable.
`subscripts`	logical: if true the panel function is given an additional (third) argument `subscripts` giving the subscripts of the data passed to that panel.
`axlabels`	function for creating axis (tick) labels when x or y are factors.
`number`	integer; the number of conditioning intervals, for a and b, possibly of length 2. It is only used if the corresponding conditioning variable is not a `factor`.
`overlap`	numeric < 1; the fraction of overlap of the conditioning variables, possibly of length 2 for x and y direction. When overlap < 0, there will be *gaps* between the data slices.
`xlim`	the range for the x axis.
`ylim`	the range for the y axis.
`...`	additional arguments to the panel function.
`x`	a numeric vector.

Details.

In the case of a single conditioning variable `a`, when both `rows` and `columns` are unspecified, a "close to square" layout is chosen with `columns >= rows`.

In the case of multiple `rows`, the *order* of the panel plots is from the bottom and from the left (corresponding to increasing `a`, typically).

A panel function should not attempt to start a new plot, but just plot within a given coordinate system: thus `plot` and `boxplot` are not panel functions.

As from R 2.0.0 the rendering of arguments `xlab` and `ylab` is not controlled by `par` arguments `cex.lab` and `font.lab` even though they are plotted by `mtext` rather than `title`.

Value.

`co.intervals(., number, .)` returns a (`number` $\times$ 2) `matrix`, say `ci`, where `ci[k,]` is the `range` of `x` values for the `k`-th interval.

References.

Chambers, J. M. (1992) *Data for models.* Chapter 3 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Cleveland, W. S. (1993) *Visualizing Data.* New Jersey: Summit Press.

See Also.

`pairs`, `panel.smooth`, `points`.

Examples.

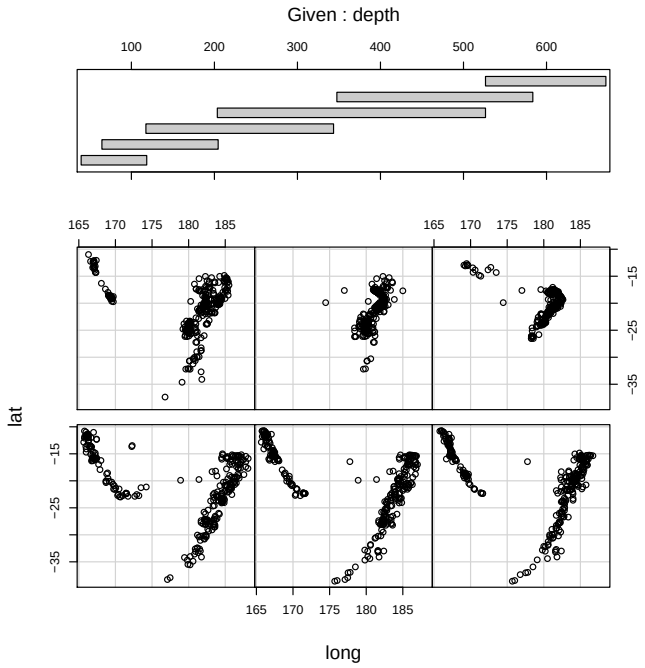
```
## Tonga Trench Earthquakes
coplot(lat ~ long | depth, data = quakes)
given.depth <- co.intervals(quakes$depth, number=4, overlap=.1)
coplot(lat ~ long | depth, data = quakes, given.v=given.depth, rows=1)

## Conditioning on 2 variables:
ll.dm <- lat ~ long | depth * mag
coplot(ll.dm, data = quakes)
coplot(ll.dm, data = quakes, number=c(4,7), show.given=c(TRUE,FALSE))
coplot(ll.dm, data = quakes, number=c(3,7),
       overlap=c(-.5,.1)) # negative overlap DROPS values

## given two factors
Index <- seq(length=nrow(warpbreaks)) # to get nicer default labels
coplot(breaks ~ Index | wool * tension, data = warpbreaks, show.given = 0:1)
coplot(breaks ~ Index | wool * tension, data = warpbreaks,
       col = "red", bg = "pink", pch = 21, bar.bg = c(fac = "light blue"))

## Example with empty panels:
attach(data.frame(state.x77))#> don't need 'data' arg. below
coplot(Life.Exp ~ Income | Illiteracy * state.region, number = 3,
       panel = function(x, y, ...) panel.smooth(x, y, span = .8, ...))
## y ~ factor -- not really sensical, but 'show off':
coplot(Life.Exp ~ state.region | Income * state.division,
       panel = panel.smooth)
detach() # data.frame(state.x77)
```

Wir illustrieren die Coplots mit dem "Quakes"-Datensatz. Dieser Datensatz gibt die geografische Länge und Breite einer Reihe von Erdbeben in der Nähe der Fiji-Inseln, zusammen mit der Tiefe des Erdbebenherdes. Wir benutzen die geografische Länge und Breite als Variablen aud die wir projizieren, und die Tiefe als Covariable, nach der wir Schnitte bilden.

Beispiel 4.9:

Eingabe

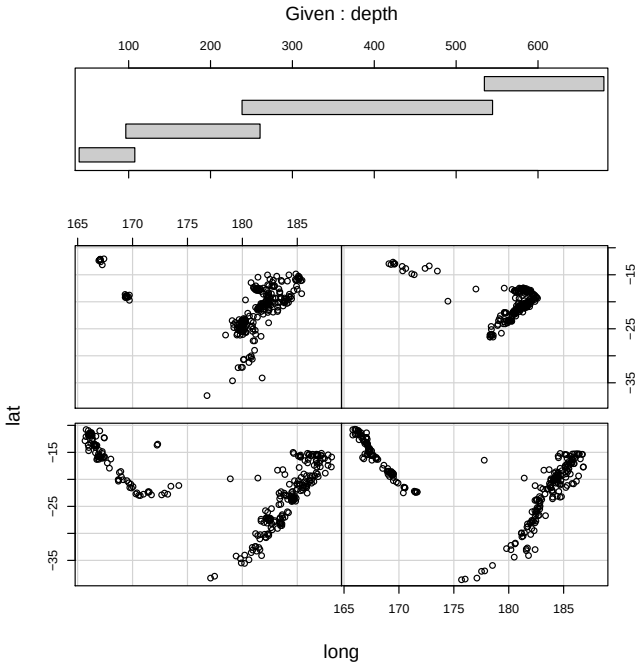
```
data(quakes)
coplot(lat ~ long | depth, data = quakes)
```



Beispiel 4.10:

Eingabe

```
given.depth <- co.intervals(quakes$depth, number = 4,
    overlap = 0.1)
coplot(lat ~ long | depth, data = quakes, given.values = given.depth)
```



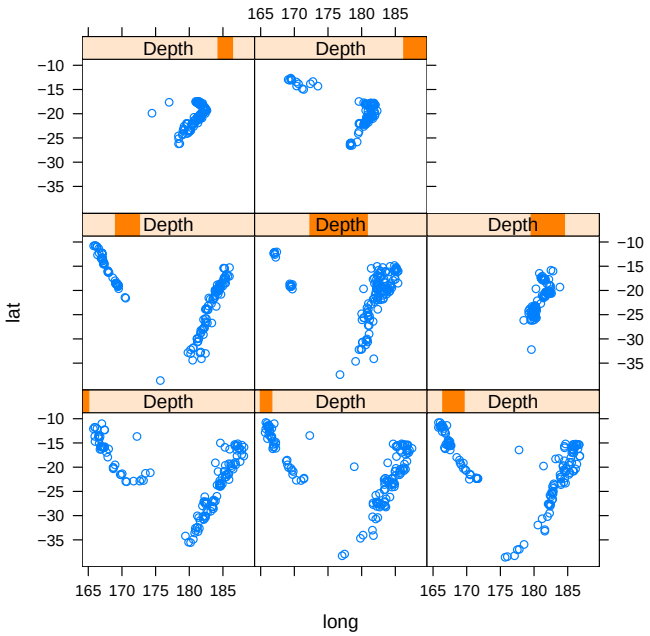
Aufgabe 4.4	
	Analysieren Sie den “quakes”-Datensatz. Fassen Sie Ihre Ergebnisse zusammen. Versuchen Sie, ein formales Modell zu formulieren.

Die Idee der Coplots wird generalisiert in den Trellis-Displays (siehe [**Cle93**]). Trellis-Displays sind in R in *library("lattice")* implementiert.

Beispiel 4.11:

Eingabe

```
library("lattice")
Depth <- equal.count(quakes$depth, number = 8, overlap = 0.1)
print(xyplot(lat ~ long | Depth, data = quakes))
```

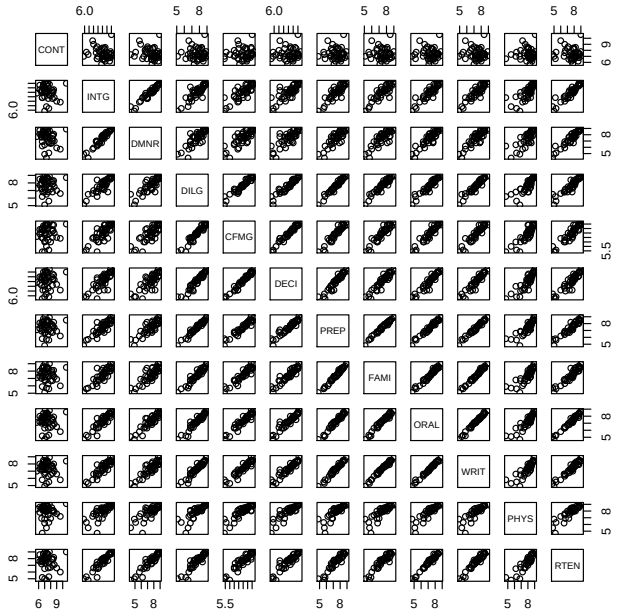


Der USJudgeRating-Datensatz ist ein Beispiel, wo die Strategie der schrittweisen Elimination angewandt werden könnte. Wieder ist die einfache Scatterplotmatrix der Rohdaten nur der erste Schritt. Sie zeigt uns, dass zwischen allen Variablen (ausser CONT) eine starke lineare Abhängigkeit besteht. Sie lässt uns jedoch mit dem Problem alleine, die interne Struktur dieser Variablen zu untersuchen.

Beispiel 4.12:

Eingabe

```
data(USJudgeRatings)
pairs(USJudgeRatings)
```



Dies ist ein zwölfdimensionaler Datensatz. An den Diagonallinien ist eine starke lineare Abhängigkeit zwischen den meisten Variablenpaaren erkennbar. Die Variable "cont"= Anzahl der Klientenkontakte ist dabei eine deutliche Ausnahme.

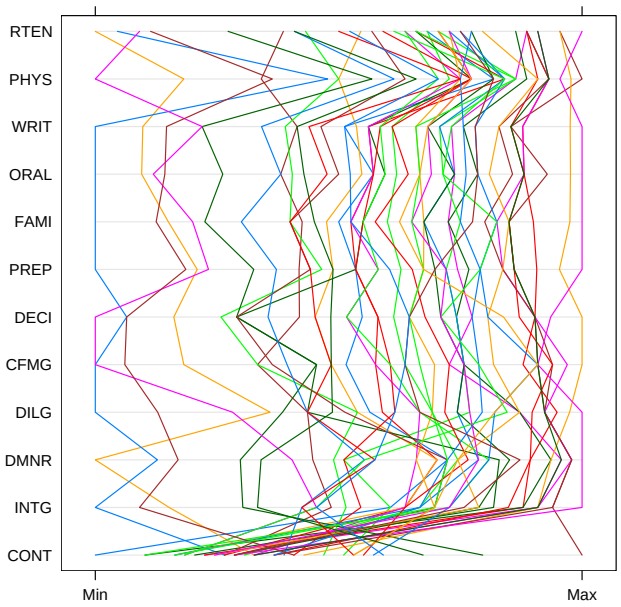
Die Scatterplot-Matrix zeigt uns die marginale paarweise Verteilung aller Variablen. Es ist an sich schon hilfreich, diesen Plot zu untersuchen. Dies über eine blinde Anwendung von Schätzmethoden hinaus. Aber der Wert von Scatterplot-Matrizen ist begrenzt durch die starke Abhängigkeit zwischen den Variablen. Die Funktion *pairs()* ist hier nur von bedingtem Wert.

Neben der starken Abhängigkeit, die das Bild dominiert, ist die Anzahl der Plots so groß, dass es kaum gelingt, einen Überblick zu bekommen. Das letztere Problem kann umgangen werden, indem man als Darstellung Parallel-Koordinaten anstelle von cartesischen Koordinaten wählt. In Parallel-Koordinaten erhält jede Variable eine eigene Koordinaten-Achse. Die Koordinaten-Achsen werden parallel angeordnet. Die zu einem Fall gehörenden Markierungen auf diesen Achsen werden durch einen Linienzug verbunden. Parallel-Koordinaten sind in *library("lattice")* bereitgestellt. Diese Plots sind etwas gewöhnungsbedürftig. Was jedoch schnell auffällt ist, dass die Einstufungen, die von Rechtanwälten mit wenigen Kontakten stammen, eher positiv ausfallen.

Beispiel 4.13:

Eingabe

```
library("lattice")
print(parallel(USJudgeRatings[1:12], data = USJudgeRatings))
```



Aufgabe 4.5	
	Analysieren Sie den "USJudgeRating"-Datensatz. Definieren Sie zu Beginn die Rolle der Variablen *CONT* und *RTEN*.
	Iterieren Sie folgende Strategie: Wählen Sie eine Variable als Leitvariable. Ersetzen Sie diejenigen Variablen, die deutlich linear mit dieser Leitvariablen zusammenhängen, durch die Residuen der entsprechenden Regression.
	Können Sie Fälle identifizieren, die vom allgemeinen Muster abweichen?
	Versuchen Sie "Scores" anzugeben, um die Fälle zu ordnen.
	Finden Sie Indikatoren, die Abweichungen vom allgemeinen Score-Muster diagnostizieren.
	Vergleichen Sie zwei unterschiedliche Anordnungen der Variablen.

Aufgabe 4.6	
	Analysieren Sie den “USJudgeRating”-Datensatz in inverser Richtung. Benutzen Sie die abgegebene Bewertung der Richter, um die Anwälte, nicht die Richter zu analysieren. Gibt es Gruppen? Gibt es Ausreißer? Welche Rolle haben nun die einzelnen Variablen?

4.7. Statistische Zusammenfassung

Die Analyse multivariater Daten konnte in diesem Zusammenhang nur gestreift werden. Multivariate Probleme tauchen implizit schon bei Regressionsproblemen auf (siehe Kapitel 2). Bei den einfachen Regressionsproblemen bezogen sich die multivariaten Aspekte aber nur auf deterministische Parameter. Im allgemeinen Fall haben wir aber eine multivariate statistische Verteilung zu analysieren. An dieser Stelle muss die Einführung abbrechen, und weiteres bleibt weiterführenden Vorlesungen vorbehalten.

R als Programmiersprache: Übersicht

R ist eine interpretierte Ausdruckssprache. Ausdrücke sind zusammengesetzt aus Objekten und Operatoren.

A.1. Namen und Suchpfade

Objekte werden durch Namen identifiziert. Anhand des Namens werden Objekte in einer Kette von Suchbereichen identifiziert. Die aktuellen Suchbereiche können mit `search()` inspiziert werden.

R ***Suchpfade***	
`search()`	Liste der aktuellen Suchbereiche, beginnend mit `.GlobalEnv` bis hinab zum Basis-Paket `package:base`. *Beispiel:* `search()`
`searchpaths()`	Liste der Zugriffspfade zu aktuellen Suchbereichen *Beispiel:* `searchpaths()`
`ls()`	Liste der Objekte in einem Suchbereich *Beispiele:* `ls()` `ls("package:base")`
`help()`	Information über ein Objekt/eine Funktion *Beispiel:* `help(help)`
`help.search()`	Sucht Information über ein Objekt/eine Funktion
`apropos()`	Lokalisiert nach Stichwort
`find()`	Lokalisiert nach Stichwort. Findet auch überlagerte Einträge

Funktionen können sowohl bei Definition als auch bei Aufruf geschachtelt sein. Dies macht eine Erweiterung der Suchpfade nötig. Die dynamische Identifikation von Objekten benutzt Umgebungen (environments), um in Funktionen lokale Variable oder globale Variablen aufzulösen.

R ***Suchpfade (Fortsetzung)***	

(Fortsetzung)→

R ***Suchpfade (Fortsetzung)*** (Fortsetzung)	
`environment()`	Aktuelle Auswertungsumgebung *Beispiel:* `environment()`
`sys.parent()`	Vorausgehende Auswertungsumgebungen *Beispiel:* `sys.parent(1)`

Objekte haben zwei implizite Attribute, die erfragt werden mit `mode()` und `length()`. Die Funktion `typeof()` gibt den (internen) Speichermodus eines Objektes an.

A.2. Basis-Datentypen

R ***Basis-Datentypen***	
numeric	*real* oder *integer*. In R: real ist stets doppelt-genau. Einfache Genauigkeit wird für exterene Aufrufe zu anderen Sprachen mit .C oder .Fortran unterstützt. Funktionen wie *mode()* und *typedef()* können je nach Implementierung auch den Speicherungsmodus (single, double ...) melden. *Beispiele:* *1.0* *2* *3.14E0*
complex	komplex, in cartesischen Koordinaten *Beispiel:* *1.0+0i*
logical	TRUE, FALSE. In R: auch vordefinierte Variable T, F. In S-Plus sind T und F Basis-Objekte.
character	Zeichenketten. Delimiter sind alternativ " oder '. *Beispiel:* *"T"*, '*klm*'
list	Allgemein Liste. Die Listenelemente können auch von unterschiedlichem Typ sein. *Beispiel:* *list(1:10,"Hello")*
function	R-Funktion *Beispiel:* *sin*
NULL	Spezialfall: leeres Objekt *Beispiel:* *NULL*

Zusätzlich zu den Konstanten TRUE und FALSE gibt es drei spezielle Werte für Ausnahmesituationen:

spezielle Konstanten	
TRUE	Alternativ: *T*. Typ: logical.
FALSE	Alternativ: *F*. Typ: logical.
NA	"not available". Typ: logical. NA ist von TRUE und FALSE verschieden
NaN	"not a valid numeric value". Implementationsabhängig. Sollte dem IEEE Standard 754 entsprechen. Typ: numeric. *Beispiel:* 0/0

(Fortsetzung)→

spezielle Konstanten (Fortsetzung)	
`Inf`	unendlich. Implementationsabhängig. Sollte dem IEEE Standard 754 entsprechen. Typ: numeric. *Beispiel:* `1/0`

A.3. Ausgabe von Objekten

Die Objekt-Attribute und weitere Eigenschaften können abgefragt oder mit Ausgaberoutinen angefordert werden. Die Ausgaberoutinen sind in der Regel ***polymorph***, d.h. sie erscheinen in Varianten, die den jeweiligen Objekten angepasst werden.

R ***Inspektion***	
print()	Standard-Ausgabe
summary()	Standard-Ausgabe als Übersicht, insbesondere für Modellanpassungen
plot()	Standard-Grafikausgabe

A.4. Inspektion von Objekten

Die folgende Tabelle fasst die wichtigsten Informationsmöglichkeiten über Objekte zusammen.

Inspektion von Objekten	
str()	Stellt die interne Struktur eines Objekts in kompakter Form dar. *Aufruf:* *str(*⟨object⟩*)*
class()	Objekt-Klasse. Bei neueren Objekten ist die Klasse als Attribut gespeichert. In älteren S oder R-Versionen ist sie durch Typ und andere Attribute implizit bestimmt.
mode()	Modus (Typ) eines Objekts.
storage.mode()	Speichermodus eines Objekts.
typeof()	Modus eines Objekts. Kann vom Speichermodus abweichen. Je nach Implementierung kann etwa eine numerische Variable standardmässig doppelt- oder einfach genau abgespeichert werden.
length()	Länge = Anzahl der Elemente
attributes()	Liest/setzt Attribute eines Objekts
names()	Namen-Attribut für Elemente eines Objekts, z.B. eines Vektors. *Aufruf:* *names(*⟨obj⟩*)* gibt das Namen-Attribut von ⟨obj⟩. *names(*⟨obj⟩*)<-*⟨charvec⟩ setzt es. *Beispiel:* *x<-values* *names(x)<-* ⟨charvec⟩

A.5. Inspektion des Systems

Die folgende Tabelle faßt die wichtigsten Informationsmöglichkeiten über die allgemeine Systemumgebung zusammen.

System-Inspektion	
`ls()`	aktuelle Objekte
`methods()`	generische Methoden *Aufruf:* `methods(`⟨`fun`⟩`)` zeigt spezialisierte Funktionen zu ⟨`fun`⟩, `methods(class=`⟨`c`⟩`)` die klassenspezifischen Funktionen zu class ⟨`c`⟩. *Beispiele:* `methods(plot)` `methods(class=lm)`
`data()`	zugreifbare Daten
`library()`	zugreifbare Bibliotheken
`help()`	allgemeines Hilfe-System

A.6. Komplexe Datentypen

Die Interpretation von Basistypen oder abgeleiteten Typen kann durch ein oder mehrere `class`-Attribute spezifiziert werden. Polymorphe Funktionen wie `print` oder `plot` werten dieses Attribut aus und rufen nach Möglichkeit entsprechend der Klasse spezialisierte Varianten auf (Siehe 2.6.4Seite 2-23).

Zur Speicherung von Datumsangaben und Zeiten stehen entsprechende Klassen bereit. Nähere Information zu diesen Datentypen erhält man mit

```
help(DateTimeClasses).
```

R ist vektor-basiert. Einzelne Konstanten oder Werte sind nur Vektoren der speziellen Länge 1. Sie geniessen keine Sonderbehandlung.

Zusammengesetzte Objekttypen	
Vektoren	R Basis-Datentypen
Matrizen	Vektoren mit zwei-dimensionalem Layout
Arrays	Vektoren mit höherdimensionalem Layout `dim()` definiert Dimensionsvektor *Beispiel:* `x < -runif(100)` `dim(x) < - c(5,5,4)` `array()` konstruiert neuen Vektor mit gegebener Dimensionsstruktur *Beispiel:* `z < - array(0, c(4,3,2))` `rbind()` kettet Reihen an `cbind()` kettet Spalten an

Faktoren	Sonderfall für kategorielle Daten
	`factor()` wandelt Vektor in Faktor um
	Siehe auch Abschnitt 2.2.4
	`ordered()` wandelt Vektor im Faktor mit geordneten Stufen um. Dies ist eine Abkürzung für `factor(x, ..., ordered = TRUE)`
	`levels()` gibt die Stufen eines Faktors an
	Beispiel:
	`> x <- c("a","b", "a", "c", "a")`
	`> xf <- factor(x)`
	`> levels(xf)`
	`[ 1 ] "abc"`
	`tapply()` wendet eine Funktion getrennt für alle Stufen von Faktoren einer Faktorliste an

Listen	Analog Vektoren, mit Elementen auch unterschiedlichen Typs

(Fortsetzung)→

Zusammengesetzte Objekttypen (Fortsetzung)	
	`list()` erzeugt FListe *Aufruf:* `list(`⟨`Komponenten`⟩`) `[[ ]]` Indexweiser Zugriff auf Komponenten `Liste$Komponente` Zugriff nach Namen *Beispiel:* `l <- list(name ="xyz", age =22, fak = "math")` `> l[[2]]` `22` `>l$age` `22`
Datenrahmen	**data frames** Analog Arrays bzw. Listen, mit spaltenweise einheitlichem Typ und einheitlicher Spaltenlänge `data.frame()` analog `list()`, aber Restriktionen müssen erfüllt sein `attach()` fügt Datenrahmen in die aktuelle Suchliste ein, d.h. für Komponenten reicht der Komponentenname. `detach()`

A.7. Zugriff auf Komponenten

Die Länge von Vektoren ist ein dynamisches Attribut. Sie wird bei Bedarf erweitert und gekürzt. Insbesondere gilt implizit eine "Recyling-Regel": Hat ein Vektor nicht die erforderliche Länge für eine Operation, so wird er periodisch bis zur erforderlichen Länge wiederholt.

Auf Vektor-Komponenten kann über Indizes zugegriffen werden. Die Indizes können explizit oder als Regel-Ausdruck angegeben werden.

R ***Index-Zugriff***	
`x[`⟨`indices`⟩`]`	Indizierte Komponenten von x *Beispiel:* `x[1:3]`
`x[-`⟨`indices`⟩`]`	x ohne indizierte Komponenten *Beispiel:* `x[-3]` x ohne 3. Komponente
`x[`⟨`condition`⟩`]`	Komponenten von x, für die ⟨`condition`⟩ gilt. *Beispiel:* `x[x<0.5]`

Vektoren (und andere Objekte) können auf höherdimensionale Konstrukte abgebildet werden. Die Abbildung wird durch zusätzliche Dimensions-Attribute beschrieben. Nach Konvention erfolgt eine spaltenweise Einbettung, d.h. der erste Index variiert zuerst (FORTRAN-Konvention). Operatoren und Funktionen können die Dimensions-Attribute auswerten.

R ***Index-Zugriff***	
`dim()`	Setzt oder liest die Dimensionen eines Objekts *Beispiel:* `x <- 1:12 ; dim (x) <- c(3,4)`
`dimnames()`	Setzt oder liest Namen für die Dimensionen eines Objekts
`nrow()`	Gibt die Anzahl der Zeilen = Dimension 1
`ncol()`	Gibt die Anzahl der Spalten = Dimension 2
`matrix()`	Erzeugt eine Matrix mit vorgegebenen Spezifikationen *Aufruf:* `matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)` *Siehe auch* Beispiel 1.21 (Seite 1-21)
`array()`	Erzeugt eine evtl. höherdimensionale Matrix *Beispiel:* `array(x, dim = length(x),dimnames = NULL)`

R ***Array-Zugriffe***	
`cbind()` `rbind()`	Verkettet Zeilen bzw. Spalten

(Fortsetzung)→

R ***Array-Zugriffe*** (Fortsetzung)	
`split()`	Teilt einen Vektor nach Faktoren auf
`table()`	Erzeugt eine Tabelle von Besetzungszahlen

R ***Iteratoren***	
`apply()`	wendet eine Funktion auf die Zeilen oder Spalten einer Matrix an *Aufruf:* `apply(x, MARGIN, FUNCTION,...)` Margin = 1: Zeilen, Margin = 2: Spalten. *Siehe auch* Beispiel 1.21 (Seite 1-21)
`lapply()`	wendet eine Funktion auf die Elemente einer Liste an *Aufruf:* `lapply(X, FUN, ...)`
`sapply()`	wendet eine Funktion auf die Elemente einer Liste, eines Vektors oder einer Matrix an. Falls mögliche werden Dimensionsnamen übernommen. *Aufruf:* `sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)`
`tapply()`	wendet eine Funktion auf Komponenten eines Objekts in Abhängigkeit von einer Liste von kontrollierenden Faktoren an.
`replicate()`	Wertet eine Ausdruck wiederholt aus (z. Bsp. mit Erzeugung von Zufalllszahlen zur Simulation. *Aufruf:* `replicate(n, expr, simplify = TRUE)`
`outer()`	erzeugt eine Matrix mit allen Paar-Kombinationen aus zwei Vektoren, und wendet eine Funktion auf jedes Paar an. *Aufruf:* `outer(vec1,vec2, FUNCTION, ...)`

A.8. Operatoren

Ausdrücke in R können aus Objekten und Operatoren zusammengesetzt sein. Die folgende Tabelle ist nach Vorrang geordnet (höchster Rang oben).

R ***Basisoperatoren***	
`$`	Komponenten-Selektion *Beispiel:* `list$item`
`[ [[`	Indizierung, Elementzugriff *Beispiel:* `x[i]`
`^`	Potenzierung *Beispiel:* `x^3`
`-`	unitäres Minus
`:`	Folge-Generierung *Beispiele:* `1:5` 5:1
`%`⟨`name`⟩`%`	spezielle Operatoren. Können auch benutzer-definiert sein. *Beispiele:* `"%deg2%"<-function(a,b) a + b^2` `2 %deg2% 4`
`* /`	Multiplikation, Division
`+ -`	Addition, Subtraktion
`< > <= >= == !=`	Vergleichsoperatoren
`!`	Negation
& | && ||	und, oder && , || sind "Shortcut"-Operatoren
`<- ->`	Zuweisung

Operatoren der Form %⟨`name`⟩% können vom Benutzer definiert werden. Die Definition folgt den Regeln für Funktionen.

Ausdrücke können als Folge mit trennendem Semikolon geschrieben werden. Ausdrucksgruppen können durch {...} zusammengefasst werden.

A.9. Funktionen

Funktionen sind spezielle Objekte. Funktionen können Resultat-Objekte übergeben.

R *Funktions-deklarationen*	
Deklaration	*function* (⟨formale Parameterliste⟩) ⟨Ausdruck⟩ *Beispiel:* *fak <- function(n) prod(1:n)*
Formale Parameter	⟨Parametername⟩ ⟨Parametername⟩ =⟨Default-Wert⟩
Formale Parameterliste	Liste von formalen Parametern, durch Komma getrennt *Beispiele:* *n, mean=0, sd=1*
...	Variable Parameterliste. Variable Parameterlisten können innerhalb von Prozeduren weitergegeben werden. *Beispiel:* *mean.of.all <- function (...)mean(c(...))*
Funktions-Resultate	*return* ⟨Wert⟩ bricht Funktionsauswertung ab und übergibt Wert
	⟨Wert⟩ als letzter Ausdruck in einer Funktionsdeklaration: übergibt Wert
Funktions-Resultate	⟨Variable⟩<<-⟨Wer t⟩ übergibt Wert. Normalerweise wirken Zuweisungen nur auf lokale Kopien der Variablen. Die Zuweisung mit <<- jedoch sucht die Zielvariable in der gesamten Umgebungshierachie.

R *Funktions-aufruf*	
Funktionsaufruf	⟨Name⟩*(*⟨Aktuelle Parameterliste⟩*)* *Beispiel:* *fak(3)*
Aktuelle Parameterliste	Werte werden zunächst der Position nach zugeordnet. Abweichend davon können Namen benutzt werden, um Werte gezielt zuzuordnen. Dabei reichen die Anfangsteile der Namen. Mit der Funktion *missing()* kann überprüft werden, ob für einen formalen Parameter ein entsprechender aktueller Parameter fehlt. *Aufruf:* ⟨Werteliste⟩ ⟨Parametername⟩=⟨Werte⟩ *Beispiel:* *rnorm(10, sd=2)*

Parameter bei Funktionen werden dem Wert nach übergeben, Soll der damit verbundene Aufwand vermieden werden, so kann mit Hilfe der *environment*-Information direkt auf Variable zugegriffen werden. Entstprechende Techniken sind in [**GI00**] beschrieben.

Spezialfall: Funktionen mit Namen der Form `xxx<-` erweitern die Zuweisungsfunktion.
Beispiel:

```
"inc<-" <-function (x, value) x+value
x <- 10
inc(x)<- 3
x
```

In R-Zuweisungsfunktionen **muss** das Wert-Argument "value" heißen.

A.10. Debugging und Profiling

R bietet eine Reihe von Werkzeugen zur Identifizierung von Fehlern. Diese sind besonders im Zusammenhang mit Funktionen hilfreich. Mit *browser()* kann in einen Browser-Modus geschaltet werden. In diesem Modus sind die üblichen R-Anweisungen möglich. Daneben gibt es eine kleine Zahl von speziellen Anweisungen. Der Browser-Modus kann mit *debug()* automatisch bei Eintritt in eine Funktion aktiviert werden. Durch den speziellen Prompt *Browse[xx]>* ist der Browser-Modus erkennbar. *debug*-Kontrolle steht. Fährt mit der Anweisungsausführung fort, falls *browser* direkt aufgerufen wurde.
aufgerufen wurde).
fort.

⟨return⟩: geht zur nächsten Anweisung, falls die Funktion unter *debug*-Kontrolle steht. Fährt mit der Anweisungsausführung fort, falls *browser* direkt aufgerufen wurde.

n: geht zur nächsten Anweisung (auch falls *browser* direkt aufgerufen wurde).

cont: Fährt mit der Anweisungsausführung fort.

c: Kurzform für *cont*. Fährt mit der Anweisungsausführung fort.

where: Zeigt Aufrufverschachtelung.

Q: Stoppt Ausführung und springt in Grundzustand zurück.

Debug-Hilfen	
browser()	Hält die Ausführung an und geht in den Browser-Modus. *Aufruf:* *browser()*
recover()	*recover()* zeigt eine Liste der aktuellen Aufrufe, aus der einer zur *browser()*-Inspektion gewählt werden kann. Mit *c* kehrt man aus dem *browser* zu *recover* zurück. Mit *0* verlässt man *recover()* *Aufruf:* *recover()* *Hinweis:* Mit *options(error=recover)* kann die Fehlerbehandlung so konfiguiert werden, dass im Fehlerfalle automatisch *browser()* aufgerufen wird.
debug()	Markiert eine Funktion zur Debugger-Kontrolle. Bei nachfolgenden Aufrufen der Funktion wird der Debugger aktiviert und schaltet in den Browser-Modus. *Aufruf:* *debug(*⟨Funktion⟩*)*
undebug()	Löscht Debugger-Kontrolle für eine Funktion. *Aufruf:* *undebug(*⟨Funktion⟩*)*

(Fortsetzung)→

Debug-Hilfen (Fortsetzung)	
trace()	Markiert eine Funktion zur Trace-Kontrolle. Bei nachfolgenden Aufrufen der Funktion wird der Aufruf mit seinen Argumenten angezeigt. *Aufruf:* *trace(*⟨Funktion⟩*)*
untrace()	Löscht Trace-Kontrolle für eine Funktion. *Aufruf:* *untrace(*⟨Funktion⟩*)*
traceback()	Im Fehlerfall innerhalb einer Funktion wird die aktuelle Aufrufverschachtelung in einer Variablen *.Traceback* gespeichert. *traceback()* wertet diese Variable aus und zeigt den Inhalt an. *Aufruf:* *traceback()*

Um die Laufzeit in einzelnen Bereichen zu messen, bietet R ein "profiling", das jedoch teilweise nur verfügbar ist, wenn R mit den entsprechnden Optionen compiliert worden ist.

Profiling-Hilfen	
system.time()	Misst die Ausführungszeit einer Anweisung. Diese Funktion ist stets verfügber. *Aufruf:* *system.time(*⟨expr⟩, ⟨gcFirst⟩*)*
Rprof()	Registriert periodisch die jeweils aktiven Funktionen. Diese Funktion ist nur verfügbar, wenn R für "profiling" compiliert ist. *Aufruf:* *Rprof(filename = "Rprof.out", append = FALSE, interval = 0.02)*
summaryRprof()	Fasst die Ausgabe von *Rprof()* zusammen und berichtet den Zeitbedarf je Funktion. *Aufruf:* *summaryRprof(filename = "Rprof.out", chunksize = 5000)*

A.11. Kontrollstrukturen

R ***Kontrollstrukturen***	
if	Bedingte Ausführung *Aufruf:* *if* (⟨log. Ausdruck 1⟩) ⟨Ausdruck2⟩ Der logische Ausdruck 1 darf nur einen logischen Wert ergeben. Für vektorisierten Zugriff benutze man *ifelse*. *Aufruf:* *if* (⟨log. Ausdruck1⟩) ⟨Ausdruck2⟩ *else* ⟨Ausdruck3⟩
ifelse	Elementweise bedingte Ausführung *Aufruf:* *ifelse*(⟨log. Ausdruck1⟩, ⟨Ausdruck2⟩, ⟨Ausdruck3⟩) Wertet den logischen Ausdruck 1 elementweise auf einen Vektor an, und übergibt bei wahrem Resultat den elementweisen Wert von Ausdruck2, sonst von Ausdruck3) *Beispiel:* `trimmedX <- ifelse (abs(x)<2, X, 2)`
switch	Auswahl aus einer Liste von Alternativen *Aufruf:* *switch*(⟨Ausdruck1⟩, ...) Ausdruck1 muss einen numerischen Wert oder eine Zeichenkette ergeben. ... ist eine explizite Liste der Altenativen. *Beispiel:* `centre <- function (x , type) { switch(type, mean =mean(x), median =median(x), trimmed= mean(x, trim= .1)}`
for	Iteration (Schleife) *Aufruf:* *for* (⟨name⟩ *in* ⟨Ausdruck1⟩) ⟨Ausdruck2⟩

repeat	Wiederholung. Muss z.B. mit *break* verlassen werden.
	Aufruf: *repeat* ⟨Ausdruck⟩
	Beispiel:
	`pars<-init`
	`repeat { res<- get.resid (data,pars)`
	`if (converged(res) ) break`
	`pars<-new.fit (data,pars)}`

while	Bedingte Wiederholung *Aufruf:* *while* (⟨log. Ausdruck⟩) ⟨Ausdruck⟩ *Beispiel:* `pars<-init; res <- get.resid (data,pars) while (!converged(res)) { pars<- new.fit(data,pars) res<- get.resid}`
break	verlässt die aktuelle Schleife
next	verlässt einen Schleifenzyklus und springt zum nächsten

A.12. Verwaltung und Anpassung

objects() *ls()*	Liste der aktuellen Objekte
rm()	Löscht die angegebenen Objekte *Aufruf:* *rm(*⟨Objektliste⟩*)*

A.13. Ein- und Ausgabe in Ausgabeströme

R ***Ein/Ausgabe***	
write()	Schreibt Daten in eine Datei. *Aufruf:* *write(val,file)* *Beispiel:* *write(x,file="data")*
source()	Führt die R-Anweisungen aus der angegebenen Datei aus. *Aufruf:* *source*(”⟨Dateiname⟩”) *Beispiel:* *source("cmnds.R")*
sink()	Lenkt Ausgaben in die angegebene Datei. *Aufruf:* *sink*(”⟨Dateiname⟩”) *Beispiel:* *sink()* lenkt die Ausgabe wieder auf die Konsole.
dump()	Schreibt für ein Objekt die definierenden Kommandos. Mit *source()* kann aus der Ausgabe das Objekt regeneriert werden *Aufruf:* *dump(list, file="*⟨dumpdata.R⟩*", append=FALSE)*

A.14. Externe Daten

Zum Editieren und für die Eingabe nach Spreadsheet-Art innerhalb von R gibt es *edit()* (früherer Name: *data.entry()*).

Für den Austausch müssen die Datenformate zwischen allen Beteiligten abgestimmt sein. Zum Import aus Datenbanken und anderen Paketen steht eine Reihe von Bibliotheken zu verfügung, z.B. *stataread* für Stata, *foreign* für SAS, Minitab und SPSS, *RODBC* für SQL. Weitere Information findet sich im Manual "Data Import/Export" ([**R D04b**]).

Innerhalb von R werden vorbereitete Daten üblicherweise als *data frames* bereitgestellt. Sind zusätzliche Objekte wie Funktionen oder Parameter nötig, so können sie gebündelt als Paket bereit gestellt werden (siehe Aufgabe A.15 (Seite A-29)).

Für den Austausch zu R kann ein spezielles Austauschformat benutzt werden. Dateien in diesem Format können mit *save()* generiert werden und haben konventionell die Namensendung *.Rda*.

Daten werden mit der Funktion *data()* geladen. Abhängig von der Endung des Dateinamens der Eingabedatei verzweigt *data()* in mehreren Spezialfällen. Neben den *.Rda* sind übliche Endungen für reine Daten-Eingabedateien *.tab* oder *.txt*. Die online-help-Funktion *help(data)* gibt weitere Auskunft.

Ein- Ausgabe von Daten für R	
save()	Speichert Daten *Aufruf:* *save(⟨Namen der zu speichernden Objekte⟩, file=⟨Dateiname⟩,...)*
data()	Lädt Daten *Aufruf:* *data(... , list =character(0), package = c(.packages(),.Autoloaded), lib.loc = .lib.loc)* *Beispiel:* *data(crimes) # lädt den Datensatz 'crimes'*

Für den flexiblen Austausch mit anderen Programmen werden Daten in der Regel als Text-Dateien bereitgestellt, nach Möglichkeit

- in Tabellenform,
- nur ASCII-Zeichen (z.B. keine Umlaute!)
- Variablen spaltenweise angeordnet
- Spalten durch Tabulator-Sprünge getrennt.
- evtl. Spaltenüberschriften in Zeile 1
- evtl. Zeilennr. in Spalte 1.

Dafür wird zum Lesen die Funktion *read.table()* und zum Schreiben die Funktion *write.table()* bereitgestellt. Neben *read.table()* gibt es eine Reihe von Varianten, die auf andere gebräuchliche Datenformate abgestimmt sind. Diese sind unter *help(read.table)* aufgeführt.

Ein- Ausgabe von Daten zum Austausch	
`read.table()`	Liest Daten-Tabelle *Aufruf:* `read.table(file, header = FALSE, sep = "\t",...)` *Beispiele:* `read.table(`⟨Dateiname⟩`, header=TRUE, sep='\t')` Überschriften in Zeile 1, Zeilennr. in Spalte 1 `read.table(`⟨Dateiname⟩`, header=TRUE, sep='\t')` keine Zeilennr., Überschriften in Zeile 1,
`write.table()`	Schreibt Daten-Tabelle *Aufruf:* `write.table(file, header = FALSE, sep='\t',...)` *Beispiele:* `write.table(`⟨data frame⟩`,`⟨Dateiname⟩`, header=TRUE, sep='\t')` Überschriften in Zeile 1, Zeilennr. in Spalte 1 `write.table(`⟨data frame⟩`,`⟨Dateiname⟩`, header=TRUE, sep='\t')` keine Zeilennr., Überschriften in Zeile 1,

Defaultmäßig konvertiert `read.table()` Daten in `factor`-Variable, falls möglich. Dieses Verhalten kann mit dem Parameter `as.is` beim Aufruf von `read.table()` modifiziert werden. Diese Modifikation ist z. B. nötig, um Datums- und Zeitangaben einzulesen, wie in dem folgen Beispiel aus [**GP04**]:

```
# date col in all numeric format yyyymmdd
df <- read.table("laketemp.txt", header = TRUE)
as.Date(as.character(df$date), "%Y-%m-%d")
# first two cols in format mm/dd/yy hh:mm:ss
# Note as.is= in read.table to force character
library("chron")
df <- read.table("oxygen.txt", header = TRUE,
as.is = 1:2)
chron(df$date, df$time)
```

Für sequentielles Lesen steht `scan()` zur Verfügung. Dateien mit stellengenau fest vorgegebenenem Format können mit `read.fwf()` gelesen werden.

A.15. Libraries, Pakete

Externe Information kann in (Text)-Dateien und Paketen(Packages) gespeichert sein. Bibliotheken und Pakete sind dabei nach speziellen R-Konventionen strukturiert. "Bibliotheken" sind Sammlungen von "Paketen".

Zusätzliche Funktionen werden in der Regel als Pakete bereitgestellt. Pakete werden mit

```
library()
```

geladen. Im Paket enthaltene Datensätze sind dann direkt auffindbar und werden mit

```
data()
```

(ohne Argument) aufgelistet.
Beispiel:

```
library(nls)
data()
data(Puromycin)
```

Pakete	
`library()`	Lädt Zusatzpaket *Aufruf:* `library(package,...)` *Siehe auch* Abschnitt 1.5.6
`require()`	Lädt Zusatzpaket; gibt Warnung bei Fehler. *Aufruf:* `require(package,...)`
`detach()`	Gibt Zusatzpaket frei und entfernt es aus dem Suchpfad. *Aufruf:* `detach(⟨name⟩)`
`install.packages()`	Installiert Pakete in ⟨lib⟩, lädt sie bei Bedarf aus dem Archiv *CRAN* *Aufruf:* `install.packages(pkgs, lib, CRAN = getOption("CRAN"),...)`
`package.manager()`	Falls implementiert: Interface zur Verwaltung installierter Pakete. *Aufruf:* `package.manager()`
`package.skeleton()`	Erstellt das Gerüst für ein neues Paket. *Aufruf:* `package.skeleton(name = "⟨anRpackage⟩", list, ...)`

Detailinformation zur Erstellung von R-Paketen findet man in "Writing R Extensions" ([**R D04e**]).

A.16. Modell-Beschreibungen

Lineare statistische Modelle können durch Angabe einer Design-Matrix X spezifiziert werden und in der allgemeinen Form

$$\mathtt{Y} = \mathtt{X}\beta + \varepsilon$$

dargestellt werden, wobei die Matrix `X` jeweils genauer bestimmt werden muß.

R erlaubt es, Modelle auch dadurch zu spezifizieren, dass die Regeln angegeben werden, nach denen die Design-Matrix gebildet wird.

Operator	Syntax	Bedeutung	Beispiel
$\sim$	$Y \sim M$	Y hängt von M ab	$Y \sim X$ ergibt $E(Y) = a + bX$
$+$	$M_1 + M_2$	M_1 und M_2 einschliessen	$Y \sim X + Z$ $E(Y) =$ $a + bX + cZ$
$-$	$M_1 - M_2$	M_1 einschliessen, aber M_2 ausschliessen	$Y \sim X - 1$ $E(Y) = bX$
:	$M_1 : M_2$	Tensorprodukt	
% in %	M_1% in %M_2	modifiziertes Tensorprodukt	
$*$	$M_1 * M_2$	"gekreuzt": $M_1 + M_2 + M_1 : M_2$	
/	M_1/M_2	"geschachtelt": $M_1 + M_2$ %in% M_1	
^	$M\hat{}n$	M mit allen "Interaktionen" bis Stufe n	
$I()$	$I(M)$	Interpretiere M. Terme in M behalten ihre ursprüngliche Bedeutung; das Resultat bestimmt das Modell.	$Y \sim (1 + I(X^2))$ $E(Y) = a + bX^2$

Tabelle A.44. Wilkinson-Rogers-Notation für lineare Modelle

Die Modell-Spezifikation ist auch für allgemeinere, nicht lineare Modelle möglich.

Beispiele

`y ~ 1 + x`	entspricht $\mathtt{y_i} = (\mathtt{1\ x_i})(\beta_1\ \beta_2)^\top + \varepsilon$
`y ~ x`	Kurzschreibweise für `y ~1 + x` (Konstanter Term wird implizit angenommen)
`y ~ 0 + x`	entspricht $\mathtt{y_i} = \mathtt{x_i} \cdot \beta + \varepsilon$
`log(y) ~ x1 + x2`	entspricht $\log(\mathtt{y_i}) = (\mathtt{1\ x_{i1}\ x_{i2}})(\beta_1\ \beta_2\ \beta_3)^\top + \varepsilon$ (Konstanter Term wird implizit angenommen)

`y ~ A`	Einweg-Varianzanalyse mit Faktor `A`
`y ~ A + x`	Covarianzanalyse mit Faktor `A` und Covariable `x`
`y ~ A * B`	Zwei-Faktor-Kreuz-Layout mit Faktoren `A` und `B`
`y ~ A/B`	Zwei-Faktor hierarchisches Layout mit Faktor `A` und Subfaktor `B`

Um zwischen verschiedenen Modellen ökonomisch wechseln zu können, steht die Funktion `update()` zur Verfügung.

Modell-Verwaltung	
`formula()`	extrahiert Modellformel aus einem Objekt
`terms()`	extrahiert Terme der Modell-Formal aus einem Objekt
`contrasts()`	spezifiziert Kontraste
`update()`	Wechsel zwischen Modellen
`model.matrix()`	Generiert die Design-Matrix zu einem Modell

Anwendungsbeispiel:

```
lm(y ~poly(x,4), data = experiment)
```

analysiert den Datensatz "experiment" mit einem linearen Modell für polynomiale Regression vom Grade 4.

Standard-Analysen	
`lm()`	lineares Modell *Siehe auch* Kapitel 2
`glm()`	generalisiertes lineares Modell
`nls()`	nicht-lineare kleinste Quadrate
`nlm()`	allgemeine nicht-lineare Minimierung
`update()`	Wechsel zwischen Modellen
`anova()`	Varianz-Analyse

A.17. Grafik-Funktionen

R bietet zwei Grafik-Systeme: Das Basis-Grafiksystem von R implementiert ein Modell, dass an der Vorstellung von Stift und Papier orientiert ist. Das Lattice-Grafiksystem ist ein zusätzliches zweites Grafiksystem, dass an einem Kamera/Objekt-Modell orientiert ist. Information über Lattice erhält man mit `help(Lattice)` (Großbuchstabe L!), eine Übersicht über die Funktionen in Lattice mit library(help = lattice). Informationen über das Basis-Grafiksystem folgen in hier.

Grafik-Funktionen fallen im wesentlichen in drei Gruppen:

"high level"-Funktionen.	Diese definieren eine neue Ausgabe.
"low level"-Funktionen.	Diese modifizieren eine vorhandene Ausgabe.
Parametrisierungen.	Diese modifizieren die Voreinstellungen des Grafik-Systems.

A.17.1. high level Grafik.

"high level"	
`plot()`	Generische Grafikfunktion
`pairs()`	paarweise Scatterplots
`coplot()`	Scatterplots, bedingt auf Covariable
`qqplot()`	Quantil-Quantil-Plot
`qqnorm()`	Gauß-Quantil-Quantil-Plot
`qqline()`	fügt eine Linie zu einem Gauß-Quantil-Quantil-Plot hinzu, die durch das erste und dritte Quantil verläuft.
`hist()`	Histogramm *Siehe auch* Abschnitt 1.3.2, Seite 1-25
`boxplot()`	Box&Whisker-Plot
`dotplot()`	
`curve()`	Wertet eine Funktion oder einen Ausdruck nach Bedarf aus und zeichnet eine Kurve. *Beispiel:* `curve(dnorm,from=-3, to=3)`
`image()`	farbcodiertes z gegen x, y
`contour()`	Contourplot von z gegen x, y
`persp()`	3D-Fläche

A.17.2. low level Grafik. Die high-level-Funktionen haben in der Regel einen Parameter `add`. Wird beim Aufruf `add=FALSE` gesetzt, so können sie auch benutzt werden, um zu einem vorhandenen Plot Elemente hinzu zu fügen. Daneben gibt es eine Reihe von low-level-Funktionen, die voraussetzen, dass bereits eine Plot-Umgebung geschaffen ist.

"low level"	

(Fortsetzung)→

"low level" (Fortsetzung)	
`points()`	Generische Funktion. Markiert Punkte an angegebenen Koordinaten. *Aufruf:* `points(x, ...)`
`lines()`	Generische Funktion. Verbindet Punkte an angegebenen Koordinaten. *Aufruf:* `lines(x, ...)`
`abline`	Fügt Linie (in mehreren Darstellungen) zum Plot hinzu. *Aufruf:* `abline(a, b, ...)`
`polygon()`	Fügt Polygon mit spezifizierten Ecken hinzu.
`axis()`	Fügt Achsen hinzu.

Daneben hat R rudimentäre Möglichkeiten für Interaktion mit Grafik.

Interaktionen	
`locator()`	bestimmt die Position von Mausklicks. Eine aktuelle Grafik muss definiert sein, bevor `locator()` benutzt wird. *Beispiel:* `plot(runif(19))` `locator(n = 3, type = "l")`

A.17.3. Annotationen und Legenden. Die high-level-Funktion bieten in der Regel die Möglichkeiten, Standard-Beschriftungen durch geeignete Parameter zu kontrollieren.

- `main=` Haupt-Überschrift, über dem Plot
- `sub=` Plot-Unterschrift
- `xlab=` Beschriftung der x-Achse
- `ylab=` Beschriftung der y-Achse

Beschreibungen erhält man mit `help(plot.default)`.

Zur Ergänzung stehen low-level-Funktionen bereit.

"low level"	
`title()`	Setz Überschrift, analog high-level-Parametern. *Aufruf:* `title(main = NULL, sub = NULL, xlab = NULL, ylab = NULL,...)`
`text`	Fügt Text an spezifizierten Koordinaten hinzu. *Aufruf:* `text(x, y=NULL, text, ...)`
`legend()`	Fügt einen Block mit einer Legende hinzu. *Aufruf:* `legend(x, y=NULL, text, ...)`

(Fortsetzung)→

"low level" (Fortsetzung)	
`mtext()`	Fügt Randbeschriftung hinzu. *Aufruf:* `mtext(text, side = 3,...)`. Die Ränder werden bezeichnet durch 1=unten, 2=links, 3=oben, 4=rechts)

R gibt auch (eingeschränkte) Möglichkeiten zum Formelsatz. Ist der Text-Parameter eine Zeichenkette, so wird sie direkt übernommen. Ist der Text-Parameter ein (unausgewerteter) R-Ausdruck, so wird versucht, die mathematisch übliche Darstellung zu geben. R-Ausdrücke können mit den Funktionen `expression()` oder `bquote()` erzeugt werden.

Beispiel:

```
text(x,y, expression(paste(bquote("(", atop(n,x),")"), (.p)^x, (.q)^{ n-x})))
```

Ausgabe-Beispiele erhält man mit `demo(plotmath)`.

A.17.4. Grafik-Parameter.

Parametrisierungen	
`par()`	Setzt Parameter des Basis-Grafiksytsems *Aufruf:* siehe `help(par)`

A.18. Einfache Statistische Funktionen

Statistik-Funktionen	
`sum()`	summiert Komponenten eines Vektors
`cumsum()`	bildet kumulierte Summen
`prod()`	multipliziert Komponenten eines Vektors
`cumprod()`	bildet kumulierte Produkte
`length()`	Länge eines Objekts, z.B. Vektors
`max()` `min()`	Maximum, Minimum. Siehe auch `pmax, pmin`
`range()`	Minimum und Maximum
`cummax()` `cummin()`	Kumulatives Maximum, Minimum
`quantile()`	Stichprobenquantile *Aufruf:* Für theoretische Verteilungen: `qxxxx`, z.B. `qnorm`
`median()`	Median
`mean()`	Mittelwert *Aufruf:* auch getrimmte Mittel
`var()`	Varianz, Varianz / Covarianzmatrix
`sort()` `rev()`	Sortierung
`order()`	Sortierung nach Leit-Element
`rank()`	Stichprobenränge

A.19. Verteilungen, Zufallszahlen, Dichten...

Der Basis-Generator für uniforme Zufallszahlen wird von `Random` verwaltet. Verschiedene Basis-Generatoren stehen zur Verfügung. **Für ernsthafte Simulation wird eine Lektüre der Empfehlungen von Marsaglia et al. dringend empfohlen.** (Siehe `help(.Random.seed)`). Alle nicht-uniformen Zufallszahlengeneratoren sind vom aktuellen Basisgenerator abgeleitet. Eine Übersicht über die wichtigsten nicht-uniformen Zufallszahlengeneratoren, ihre Verteilungsfunktionen und ihre Quantile findet sich unten.

R *Zufallszahlen*	

`RngKind()`	RngKind() gibt den Namen des aktuellen Basisgenerators.
	RngKind (⟨`name`⟩) setzt einen Basisgenerator.
	Aufruf: `RngKind()`
	RngKind(⟨`name`⟩)
	Beispiel: `RngKind("Wichmann-Hill")`
	`RngKind("Marsaglia-Multicarry")`
	`RngKind("Super-Duper")`

`sample()`	`sample` zieht eine Zufallsstichprobe aus den im Vektor x angegebenen Werten, mit oder ohne Zurücklegen (je nach Wert von replace).
	Size ist defaultmäßig die Länge von x.
	Optional kann prob ein Vektor von Wahrscheinlichkeiten für die Werte von x sein.
	Aufruf: `sample(x, size, replace = FALSE, prob)`
	Beispiel: Zufällige Permutation:
	`sample(x)`
	`val<-c("H","T")`
	`prob<-c(0.3,0.7)`
	`sample(val,10,`
	`replace=T,prob)`

Die einzelnen Funktionsnamen für die wichtigsten nicht-uniformen Generatoren und Funktionen setzen sich aus einem Präfix und dem Kurznamen zusammen. Allgemeiner Schlüssel: `xxxx` ist der Kurzname

- `rxxxx` erzeugt Zufallszahlen
- `dxxxx` Dichte oder Wahrscheinlichkeit
- `pxxxx` Verteilungsfunktion
- `qxxxx` Quantile

Beispiel:

- `x<-runif(100)` erzeugt 100 $\mathtt{U}(0,1)$-verteilte Zufallsvariable
- `qf(0.95,10,2)` berechnet das 95%-Quantil der $\mathtt{F}(10,2)$-Verteilung.

Verteilungen	***Kurzname***	***Parameter und Default-Werte***
Beta	`beta`	`shape1, shape2, ncp=0`
Binomial	`binom`	`size, prob`
Cauchy	`cauchy`	`location=0, scale=1`
χ^2	`chisq`	`df, ncp=0`
Exponential	`exp`	`rate= 1`
`F`	`f`	`df1, df2 (ncp=0)`
Gamma	`gamma`	`shape, scale=1`
Gauß	`norm`	`mean=0, sd=1`
Geometrisch	`geom`	`prob`
Hypergeometrisch	`hyper`	`m, n, k`
Lognormal	`lnorm`	`meanlog= 0, sdlog= 1`
Logistisch	`logis`	`location=0, scale=1`
Negativ-Binomial	`nbinom`	`size, prob`
Poisson	`pois`	`lambda`
Student's t	`t`	`df`
Tukey Studentised Range	`tukey`	
Uniform	`unif`	`min =0,max=1`
Wilcoxon Signed Rank	`signrank`	`n`
Wilcoxon Rank Sum	`wilcox`	`m, n`
Weibull	`weibull`	`shape, scale = 1`

A.20. Verarbeitung von Ausdrücken

Die Sprachausdrücke von R sind genau so Objekte wie Daten oder Funktionen. Wie diese können sie gelesen oder verändert werden.

Umwandlungen	
`parse()`	Wandelt Eingabe in eine Liste von R-Ausdrücken um. `parse` führt den Parse-Schritt durch, wertet die Ausdrücke aber nicht aus.
`deparse()`	Wandelt einen R-Ausdruck in interner Darstellung in eine Zeichendarstellung um.
`expression()`	erzeugt einen R-Ausdruck in interner Darstellung. *Beispiel:* `integrate <- expression(integral(fun,lims))` *Siehe auch* 1.3.1: Mathematischer Formelsatz in Plot-Beschriftungen
`substitute()`	R-Ausdrücke mit Auswertung aller definierten Terme.
`bquote()`	R-Ausdrücke mit selektiver Auswertung. Terme in `.()` werden ausgewertet. *Beispiele:* `n<-10; bquote( n^2 == .(n*n))`

Auswertung	
`eval()`	wertet einen Ausdruck aus.

Literaturverzeichnis

[BCW88] Richard A. Becker, John M. Chambers, and Allan R. Wilks, *The new S language*, Chapman & Hall, London, 1988.

[CH92] John M. Chambers and Trevor J. Hastie, *Statistical models in S*, Chapman & Hall, London, 1992.

[Cha98] John M. Chambers, *Programming with data*, Springer, New York, 1998, ISBN 0-387-98503-4.

[Cle93] William S. Cleveland, *Visualizing data*, AT&T Bell Laboratories, Murray Hill, 1993.

[FB94] George W. Furnas and Andreas Buja, *Prosection views: dimensional inference through sections and projections*, J. Comput. Graph. Statist. **3** (1994), no. 4, 323–385.

[Gen95] Robert Gentleman, *Statistical computing using R*, Tech. Report Papers 528.188/T187, Auckland: The University, 1995.

[GI00] Robert Gentleman and Ross Ihaka, *Lexical scope and statistical computing*, Journal of Computational and Graphical Statistics **9** (2000), 491–508.

[GP04] Gabor Grothendieck and Thomas Petzoldt, *R help desk: Date and time classes in R*, R News **4** (2004), no. 1, 29–32.

[GS77] P. Gänssler and W. Stute, *Wahrscheinlichkeitstheorie*, Springer, 1977.

[JK70] N.L. Johnson and S. Kotz, *Discrete distributions*, Wiley, New York, 1970.

[Jør93] Bent Jørgensen, *The theory of linear models*, Chapman & Hall, New York-London, 1993.

[Lee95] Allan Lee, *Data analysis - an introduction based on R*, Tech. Report Papers 528.281/288, Auckland: The University, 1995.

[Mil81] R. G. Miller, *Simultaneous statistical inference*, Springer, New York, 1981.

[R D04a] R Development Core Team, *An introduction to R*, Tech. report, R Project, 2004.

[R D04b] ———, *R data import/export*, Tech. report, R Project, 2004.

[R D04c] ———, *The R language definition*, Tech. report, R Project, 2004.

[R D04d] ———, *The R reference index*, Tech. report, R Project, 2004.

[R D04e] ———, *Writing R extensions*, Tech. report, R Project, 2004.

[Rao73] C. Radhakrishna Rao, *Linear statistical inference and its applications*, 2 ed., Wiley, 1973.

[Saw94a] Günther Sawitzki, *Numerical reliability of data analaysis systems*, Camputational Statistics & Data Analysis **18** (1994), no. 2, 269–286.

[Saw94b] Günther Sawitzki, *Report on the numerical reliability of data analysis systems*, Computational Statistics and Data Analysis **18** (1994), no. 2, 289 – 301.

[VR00] William N. Venables and Brian D. Ripley, *S programming*, Springer, 2000, ISBN 0-387-98966-8.

[VR02] ———, *Modern applied statistics with S. fourth edition*, Springer, Heidelberg, 2002.

Index