E. Grüner

Einführung in R

(7.7.2009)

Was ist R?

- R ist eine umfassende Software für die Analyse und Visualisierung von Daten.
- R ist "GNU S", d.h. eine unter der GNU-Lizenz stehende Implementierung von S. Damit ist R freie Software, d.h. auch kostenlos.
- S wurde bei AT&T Bell Laboratories von John Chambers und Mitarbeitern entwickelt[1].
- S-Plus ist eine kommerzielle Implementierung von S.
- R ist ein Dialekt von S: äußerlich weitgehend identisch, innere Unterschiede sind aber vorhanden.
- Die meisten S- bzw. S-Plus-Programme laufen auch unter R.

Eigenschaften von R

- **Funktionsvielfalt**
  Differenzierte Datentypen und -strukturen, viele eingebaute Funktionen, Funktionsbibliotheken für viele Anwendungen
- **Erweiterbarkeit**
  Entwicklungsumgebung und Programmiersprache
- **Objektorientiertheit**
  Alle Größen (auch die Ausgaben von Funktionen) sind Objekte, auf die man Zugriff hat.
- **Offenheit**
  Datenschnittstelle zu SPSS, SAS u.a. Statistikdatenformaten
- **Open Source**
  Freie Software, Quelltext verfügbar, kostenfrei
- **Qualität**
  Weltweite renommierte Entwicklergemeinde
- **Aktualität**
  Neue Algorithmen sind im allgemeinen implementiert
- **Universalität**
  Läuft unter allen gängigen Betriebssystemen
- **Umfangreiche Dokumentation, Support**
  Bücher und Artikel, pdf-Dateien, Online-Hilfen, Mailinglisten

Nachteile von R

- R ist komplex und sehr mächtig.

[1] Titel des "Blue Book" (Becker, Chambers & Wilks): "The New S Language. A Programming Environment for Data Analysis and Graphics"

- R ist weniger benutzerfreundlich als beispielsweise SPSS.
- Die graphische Benutzungsoberfläche "Rgui.exe" bietet nur rudimentäre Unterstützung[2].
- Die volle Leistungsfähigkeit wird nur durch Verwendung von Kommandos erreicht: man muss die Funktionen kennen und die Syntax von R beachten.

Packages zu Büchern

MASS Funktionen/Daten zu "Modern Applied Statistics with S" (Venables & Ripley)

DAAG Funktionen/Daten zu "Data Analysis And Graphics" (Maindonald & Braun)

ISwR Funktionen/Daten zu "Introductory Statistics with R" (Dalgaard)

UsingR Funktionen/Daten zu "Using R for Introductory Statistics" (Verzani)

car Funktionen/Daten zu "An R and S-PLUS Companion to Applied Regression" (J. Fox)

faraway Funktionen/Daten zu "Linear Models with R" u.a. (Faraway)

... ...

Sonstige Packages

foreign Prozeduren zum Lesen von Fremdformaten

nlme hierarchische Regressionsmodelle

nortest Normalverteilungstests

psy verschiedene Psychometrie-Prozeduren

sem Strukturgleichungsmodelle

survival Survival-Analyse

tseries Zeitreihenanalyse

lattice Trellis Graphiken

rimage Bildbearbeitungsprozeduren

R2HTML HTML-Ausgabe

Rcmdr R Commander (GUI)

... ...

Literatur zu S/S-Plus und R

- Venables, W. N., & Ripley, B. D. (2002). *Modern Applied Statistics with S*. (4th Ed.). Springer-Verlag.
- Venables, W. N., & Ripley, B. D. (2000). *S Programming*. Springer-Verlag.
- Dalgaard, P. (2002). *Introductory Statistics with R*. Springer-Verlag.
- Venables, W. N., & Smith, D. M. () *An Introduction to R*. Network Theory Ltd.
- Fox, J. (). *An R and S-Plus Companion to Applied Regression*. Sage Publications.

[2]Die Entwicklung von GUIs zu R schreitet aber voran, siehe z.B. "R Commander".

- Pinheiro, J. C., & Bates, D. M. (2001). *Mixed-Effects Models in S and S-PLUS*. Springer-Verlag.

Weitere Dokumentation in Form von PDF-Dateien findet man im Internet, vor allem auch auf der CRAN-Seite.

Arbeitsweise von R

- R ist ein Interpreter.
- Im interaktiven Modus arbeitet R in einer sog. 'Read-Eval-Print'-Loop:
  1. **Eingabe**: der Benutzer gibt einen Ausdruck ein
  2. **Evaluierung**: das System wertet den Ausdruck aus
  3. **Ausgabe**: das System gibt das Ergebnis (oder eine Fehlermeldung) auf dem Bildschirm aus

     (weiter mit 1.)
- R kann auch im 'Batch-Modus' betrieben werden.

Fenster und Menüs

- Rgui.exe startet den interaktiven Modus unter Windows.
- Die graphische Benutzungsoberfläche enthält:
  - **Hauptfenster** mit einer Menüleiste und evtl. einer Symbolleiste sowie weiteren Fenstern
  - **Konsolfenster** für Benutzereingaben und Systemausgaben

Tastatureingaben

- Eingaben werden mit der RETURN-Taste abgeschlossen.
- Sollen mehrere Kommandos bzw. Ausdrücke auf einmal eingegeben werden, so sind sie mit Semikolon zu trennen.
- Der Eingabeprompt ist

  `>`
- Bei syntaktisch unvollständigen Eingaben erscheint als Fortsetzungprompt

  `+`
- Das Kommando für das Beendigen von R ist

  `q()`

Hilfen

- Hilfe zu einem Objekt (z.B. einer Funktion) erhält man mit der Eingabe
  `help(`*`objekt`*`)` oder kürzer mit
  `?objekt`

- Informationen zu einem Paket:
  `help(package=`*packagename*`)` oder
  `library(,`*packagename*`)`
- Auflistung aller Funktionen, deren Namen einen gegebenen String enthalten:
  `apropos(`*string*`)`
- Beispiel zu einem angegebenen Thema:
  `example(`*topic*`)`
- Aufruf der HTML-Hilfe:
  `help.start()`
- Über das Help-Menü können darüber hinaus noch weitere Hilfen aufgerufen werden:
  - Frequently Asked Questions (FAQ)
  - HTML-Hilfen
  - Die Handbücher zu R als PDF-Dateien

Die Syntax von R

- Syntax für **Bezeichner** (Variablen-, Funktionsnamen usw.)
  - Folge von Buchstaben, Ziffern und Punkten (“.”).
  - An erster Stelle muß ein Buchstabe stehen.
  - Groß-/Kleinschreibung wird unterschieden (R ist ’case-sensitive’).
- Syntax für **numerische Literale** (Zahlen)
  - Eventuell Vorzeichen, gefolgt von Ziffern.
  - Das Dezimaltrennzeichen ist der Punkt.
  - Die Exponentialschreibweise kann verwendet werden (z.B. 1.234e-3 (=0.001234)).
- Syntax für **Stringliterale** (Zeichenketten)
  - Zeichenfolge, eingefaßt in doppelte oder einfache Anführungszeichen.
- Syntax für **Dateiangaben**
  - Gemäß den Betriebssystemkonventionen eine Zeichenkette incl. evtl. Laufwerk und Pfad[3].
- Syntax für **Funktionsaufrufe**
  - Funktionsname, gefolgt von der Liste der Argumente in runden Klammern[4].
- Das Zeichen “#” leitet einen **Kommentar** ein (gilt bis zum Ende der Zeile).
- Sog. ’white spaces’ (Leerzeichen, Tabs, Zeilenvorschübe) können - und sollten auch - verwendet werden.

Metasprache zur Darstellung der Syntax von R

- Sprachkonstanten (“terminale Symbole”): sie müssen wörtlich übernommen werden.

[3] Bei der Pfadangabe können als Trennzeichen entweder **zwei** Backslashes oder ein einfacher Slash verwendet werden.
[4] Ist die Liste der Argumente leer, so ist ein leeres Klammernpaar zu setzen.

- *Kursivschrift*: metasprachliche Variablen, die durch entsprechende terminale Symbole ersetzt werden müssen.
- Optionale Teile werden durch kursiv gedruckte eckige Klammern gekennzeichnet.

Arithmetische Operatoren

Operator	Wirkung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
^	Exponentiation
%/%	Integerdivision
%%	Modulo

Vergleichsoperatoren

Operator	Wirkung
==	gleich
!=	ungleich
<	kleiner
>	größer
<=	kleiner oder gleich
>=	größer oder gleich

Logische Operatoren

Operator	Wirkung
!	Negation
& oder &&	Konjunktion (UND)
| oder ||	Disjunktion (ODER)

Numerische Funktionen

Name	Funktion
`abs`	Absolutbetrag
`sqrt`	Quadratwurzel
`sin`, `cos`, `tan`	Trigonometrische Funktionen
`asin`, `acos`, `atan`	Inverse trig. Funktionen
`sinh`, `cosh`, `tanh`	Hyperbolische trig. Funktionen
`exp`	Exponentialfunktion
`ceiling`, `floor`, `round`	Rundungsfunktionen
`trunc`, `signif`	Abschneiden der Kommastellen
`signif`	Auf signifikante Ziffern reduzieren
`factorial`	Fakultät

Spezielle Werte

Name	Bedeutung
`NA`	Undefiniert (missing data)
`Inf`	Unendlich
`NaN`	Not a Number
`T` oder `TRUE`	Wahr
`F` oder `FALSE`	Falsch
`pi`	π
`LETTERS`	Großbuchstaben
`letters`	Kleinbuchstaben
`month.names`	Monatsnamen
`month.abb`	Monatsnamen (abgekürzt)

Variablen

- Mit Hilfe von Variablen lassen sich Werte für die spätere Verwendung speichern.
- Im Gegensatz zu Konstanten wird der Wert von Variablen bei der Abarbeitung eines Programms im allgemeinen verändert.
- Variablen haben einen Namen und einen Wert. Der Name repräsentiert die Adresse eines Speicherbereichs, der Wert dessen Inhalt.
- Variablen können verschiedene Gültigkeitsbereiche (global vs. lokal) und darüber hinaus auch verschiedene Lebensdauern besitzen.
- In R brauchen Variablen vor dem Gebrauch nicht deklariert zu werden.

Wertezuweisung

- Einer Variable mit Namen *var* kann auf folgende Weise ein Wert zugewiesen werden:
  var `<-` *wert*
  var ist ein frei wählbarer Name für die Variable, *wert* kann ein beliebig komplizierter evaluierbarer Ausdruck sein.
- Durch Angabe des Namens einer Variablen kann auf deren Wert zugegriffen werden (Dereferenzierung).

Datentypen („Modi“)

- Numerisch
- Character
- Logisch
- Factor
- Ordered Factor

Datenstrukturen

- Vektoren
- Matrizen
- Arrays
- Listen
- Data Frames

Vektoren

- Der Vektor ist die einfachste Datenstruktur in R.
- Ein Vektor ist eine geordnete Menge von Daten des gleichen Typs.
- Variablen im Sinne der Statistik werden in R als Vektoren dargestellt.

Eingabe eines Vektors

- Die Werte eines Vektors können mit Hilfe der Funktion `c()` definiert werden:
  `c(`*`werteliste`*`)`
  `Werteliste`: Aufzählung der einzelnen Werte, getrennt durch Kommata.
- Daneben gibt es noch den Kolon-Operator zur Generierung einer Zahlenfolge:
  $a:b$
  Dies erzeugt die Zahlenfolge a, $a+1$, $a+2$, ..., b
  (bzw. a, a-1, a-2, ..., b, falls $a>b$).
- Weitere Möglichkeiten eröffnen die Funktionen `seq()` und `rep()`.

Die Funktion `seq()`

Parameter	**Bedeutung**
`from`	Startwert (default: 1)
`to`	End- bzw. Maximalwert
`by`	Schrittweite (default: 1 bzw. -1)
`length`	Anzahl der Werte
...	...

Die Funktion `rep()`

Zweck: Erzeugung eines Vektors durch Replizieren

Parameter	**Bedeutung**
`x`	Zahl oder Vektor
`times`	Anzahl der Replikationen von `x`
`length.out`	Länge des Ausgabevektors
`each`	Anzahl der Replikationen jedes Elementes von `x`

Hinweise

- Ist `times` eine Zahl, so wird `x` entsprechend oft wiederholt.
- Ist `x` ein Vektor und `times` ebenfalls ein Vektor, wird das i-te Element von `x` so oft wiederholt, wie das i-te Element von `times` angibt[5].

Funktionen für Vektoren

Funktionsname	**Zweck**
`length(`x`)`	Länge des Vektors x
`length(`x`)<-`n	x wird auf Länge n gesetzt
`order(`x`)`	Indexvektor (zum Sortieren)
`sort(`x`)`	Sortieren von x
`cumsum(`x`)`	kumulative Summe
`cumprod(`x`)`	kumulatives Produkt
`unique(`x`)`	x ohne doppelte Elemente
`diff(`x`,lag=1,diff=1)`	Differenzenvektor
`rev(`x`)`	gespiegelter Vektor
`min(`x`)`	Minimum
`max(`x`)`	Maximum
`range(`x`)`	Minimum und Maximum
...	...

Deskriptive Statistiken

Funktionsname	**Zweck**
`sum`	Summe
`mean`	Mittelwert
`median`	Median
`var`	Varianz bzw. Kovarianzmatrix
`sd`	Standardabweichung
`min,max`	Minimum, Maximum
`range`	Spannweite
`quantile`	Quantile
`summary`	Verschiedene Statistiken
`fivenum`	Minimum, Quartile, Maximum
`cov`	Kovarianz bzw. Kovarianzmatrix
`cor`	Korrelation bzw. Korrelationsmatrix
...	...

[5] Voraussetzung dabei ist, dass beide Vektoren gleich viele Elemente haben.

Graphikfunktionen

Funktionsname	**Zweck**
`plot`	generische Graphikfunktion
`barplot`	Balkendiagramm
`dotchart`	Dot Chart
`boxplot`	Boxplot
`stem`	Stem-and-Leaf-Plot
`hist`	Histogramm
`pie`	Kreisdiagramm
`qqnorm`	Normalverteilungsplot (Q-Q-Plot)
`pairs`	Scatterplotmatrix
`scatter.smooth`	Streudiagramm mit Glättungskurve
...	...

Hinzufügen von Elementen zu einer Graphik

Mit Hilfe der folgenden Funktionen lassen sich zu einer bestehenden Graphik verschiedene Elemente hinzufügen.

Funktionsname	**Hinzugefügte Elemente**
`points`	Punkte
`lines`	Polygonzug
`qqline`	Gerade mit den erwarteten NV-Werten
`abline`	Gerade
`segments`	Strecken
`arrows`	Pfeile
`text`	Text
...	...

Die Funktion `plot()`

Die Funktion `plot()` ist eine sog. „generische“ Plotfunktion, d.h. erzeugt einen zum übergebenen Objekt passenden Plot.

Parameter	**Bedeutung**
`x`	x-Koordinate oder das zu plottende Objekt
`y`	y-Koordinate (optional, abh. von `x`)
`main`, `sub`	Titel, Untertitel
`xlab`, `ylab`	Achsenbeschriftung
`xlim`, `ylim`	x-/y-Bereich
`axes`	Achsen zeichnen
`type`	Typ des Plots
`lty`	Linientyp
`pch`	Plotsymbol
`cex`	Skalierungsfaktor
...	...

Der Plottyp (Parameter `type`)

Mit dem Parameter `type` wird der Typ der zu erzeugenden Graphik angegeben:

Wert	**Wirkung**
`'p'`	Punkte
`'l'`	Linien
`'b'`	Punkte, verbunden mit Linien
`'o'`	Punkte und Linien
`'s'`, `'S'`	Treppe
`'h'`	Vertikale Linien
`'n'`	nichts Plotten
...	...

Die Funktion `par()`

- Die Funktion `par()` dient zur Festlegung von Voreinstellungen für Graphiken.
- Die Funktion kennt über 60 Parameter (für eine Auswahl siehe unten).
- Der Aufruf von `par()` wird den eigentlichen Graphikbefehlen vorangestellt.
- Beim Aufruf der Funktion ohne Parameter erhält man die Liste der aktuellen Einstellungen.

Parameter	Bedeutung
`adj`	Ausrichtung von geplottetem Text (0=linksbündig,1=rechtsbündig,0.5=zentriert)
`ask`	Warten vor Ausgabe einer neuen Graphik
`bg`	Hintergrundfarbe
`cex`	Skalierungsfaktor für Plottext
`col`	Zeichenfarbe
`fin`	Größe der Figur (in Inch) (Breite, Höhe)
`lab`	Kontrolle der Anzahl und Größe der Ticks
`las`	Orientierung der Achsenbeschriftung (0=parallel,1=waagrecht,3=senkrecht)
`lty`	Linientyp (1=durchgezogen,2=gestrichelt,3=punktiert,...)
`lwd`	Linienstärke
`mar`	Größe der 4 Ränder in Zeilen (unten,links,oben,rechts)
`mai`	Größe der 4 Ränder in Inch (unten,links,oben,rechts)
`mex`	Skalierungsfaktor für Koordinatenbeschriftung
`mfrow`	Mehrfachplot mit zeilenweiser Anordnung (Anzahl Zeilen, Anzahl Spalten)
`mfcol`	Mehrfachplot mit spaltenweiser Anordnung (Anzahl Zeilen, Anzahl Spalten)
`mgp`	Abstand der Achsen- und Achsenskalenbeschriftung, Achsenlinie
`oma`	Äußere Ränder eines Mehrfachplots in Zeilen
`omi`	Äußere Ränder eines Mehrfachplots in Inch
`pch`	Symbol für das Plotten von Punkten
`pin`	Größe des aktuellen Plots in Inch (Breite, Höhe)
`ps`	Schriftgröße für Text und Symbole
`pty`	Typ des Plotbereichs (`'m'`=maximal,`'s'`=quadratisch)
`tck`	Länge der Tickmarks
`tcl`	Länge der Tickmarks
`type`	Typ der Graphik
`xlog`	logarithmische x-Achse
`ylog`	logarithmische y-Achse
...	...

Die Funktion `text()`

Die Funktion `text()` dient dazu, einen Text in eine bestehende Graphik zu schreiben.

Parameter	**Bedeutung**
`x,y`	Koordinaten für den Text
`labels`	auszugebender Text
`adj`	Textausrichtung (siehe Funktion `par()`)
`pos`	Alternative zu `adj`: 1=unterhalb, 2=links, 3=oberhalb, 4=rechts der Koordinaten
`offset`	falls `pos` spezifiziert ist: Offset zu den Koordinaten
`vfont`	Angaben für einen Hershey Vektorfont
`cex`	Skalierungsfaktor für die Schriftgröße
`col`	Textfarbe
`font`	Textfont
`...`	weitere Graphikparameter

Die Funktion `mtext()`

Die Funktion `mtext()` dient dazu, einen Text in einen der Seitenränder des aktuellen Plotrahmens zu schreiben.

Parameter	**Bedeutung**
`text`	auszugebender Text
`side`	Seitenrand (1=unten/2=links/3=oben/4=rechts)
`line`	Nummer der Zeile, in die geschrieben werden soll (die Zählung geht von innen nach außen, beginnend bei 0)
`outer`	T: in den äußeren Rand schreiben, falls vorhanden (z.B. bei einem Mehrfachplot)
`at`	Position in Benutzerkoordinaten
`adj`	Textausrichtung in Leserichtung (0=links bzw. unten/1=rechts bzw. oben)
`padj`	Textausrichtung senkrecht zur Leserichtung (0=rechts bzw. oben/1=links bzw. unten)
`cex`	Skalierungsfaktor für den Text
`col`	Textfarbe
`font`	Textfont
`...`	weitere Graphikparameter

Modellanpassung in R

In R gibt es eine Reihe von Funktionen zur Modellanpassung:

- `lm()`: lineare Regression
- `glm()`: allgemeines lineares Modell
- `aov()`: Varianzanalyse
- `manova()`: multivariate Varianzanalyse
- `gam()`: allgemeines additives Modell
- `nls()`: nichtlineares Modell
- `lsfit()`: Anpassung eines Modells nach der Methode der kleinsten Quadrate
- `lme()`: lineare Modelle mit gemischten Effekten
- `nlme()`: nichtlineare Modelle mit gemischten Effekten
- ...

Der Aufruf dieser verschiedenen Funktionen erfolgt nach einem ähnlichen Muster:

- Die Daten werden normalerweise in Form eines Data Frames erwartet.
- Das anzupassende Modell wird mit Hilfe einer Formel spezifiziert.
- Je nach Funktion gibt es noch einige speziellen Angaben für die Modellanpassung.

Die Formelsyntax von R

Sowohl bei Modellen mit quantitativen Variablen (z.B. bei einer linearen Regression) als auch bei solchen für qualitative Variablen (z.B. bei einer ANOVA) ist für die Modellspezifikation eine Formel zu verwenden.

Allgemeiner Aufbau einer Formel:

response~*ausdruck-mit-erklärenden-termen*

In der folgenden Übersicht bedeutet R die Responsevariable (Kriteriumsvariable) und F, Fa und Fb irgendwelche einfachen oder zusammengesetzten Terme.

Ausdruck	**Bedeutung**
~	wird modelliert durch ...
R~F	R wird mit Hilfe von F modelliert
Fa+Fb	Fa und Fb einschließen
Fa-Fb	alle Terme von Fa ohne jene von Fb
Fa:Fb	Wechselwirkung zwischen Fa und Fb
Fa*Fb	Fa+Fb+Fa:Fb
Fb %in% Fa	Fb genestet unter Fa
Fa/Fb	Fa+(Fb %in% Fa)
F^m	alle Terme von F bis zur Ordnung m
1	konstanter Term
0	Modell ohne additive Konstante
.	das gefittete Modell bzw. dessen Response

Lineare Regression

Mit Hilfe der Funktion `lm()` läßt sich eine einfache oder multiple lineare Regressionsanalyse durchführen.

Parameter	**Bedeutung**
`formula`	Modellformel
`data`	Data Frame
`subset`	Vektor für die Auswahl von Beobachtungen
`weights`	Gewichte für Weighted Least Squares
...	...

- Obligat ist lediglich der erste Parameter.
- Zurückgegeben wird ein `lm`-Objekt, das man unter einem Variablenamen abspeichern kann.
- Ein `lm`-Objekt ist eine Liste, die aus mehreren (benannten) Komponenten besteht.
- Auf die einzelnen Komponenten des `lm`-Objekts kann man mit dem $-Operator zugreifen.

Zugriff auf ein gefittetes Modell

Ein gefittetes Objekt ist eine Liste, in dessen Elementen alle wichtigen Informationen über das Objekt gespeichert sind. Die Namen dieser Elemente sind mit Hilfe der Funktions `names()` zu erfahren. Man kann auf diese Elemente also mit dem $-Operator zugreifen.

Statistiken und Diagnoseplots zu einem Modell

Hinweis: als Argument für die nachfolgenden Funktionen ist jeweils ein `lm`-Objekt anzugeben.

Funktion	**Wirkung**
`print`	Minimalausgabe
`summary`	Residuenstatistiken, Parameterschätzungen, F, ...
`anova`	Varianzanalyse mit F-Tests bzw. Vergleich eines Modells mit einem Submodell
`plot`	Diagnoseplots
`coefficients`, `coef`	Parameterschätzungen
`fitted.values`, `fitted`	Gefittete Werte
`residuals`, `resid`	Residuen
`effects`	Schätzungen der Effekte
`confint`	Konfidenzintervalle für die Parameter
`predict`	Gefittete Werte, Konfidenzintervalle, u.a.
`deviance`	Residual-QS (RSS)
`ls.diag`	Diagnosestatistiken
`influence`, `lm.influence`	Einfluss-Statistiken
`formula`	Modellformel

Funktionen für die Modellanpassung

Funktion	**Wirkung**
`update`	Modifizierung eines linearen Regressionsmodells
`add1`	Test auf Hinzufügen eines einzelnen Effekts
`drop1`	Test auf Entfernen eines einzelnen Effekts
`step`	schrittweise Regression
...	...

Die Funktion `predict()`

Mit Hilfe der Funktion `predict()` können verschiedene Größen zu einem Modell berechnet werden:

- gefittete Werte für die Originaldaten
- gefittete Werte für neue Daten
- Standardfehler
- Konfidenzintervalle
- Vorhersageintervalle
- u.a.

Parameter	**Bedeutung**
`object`	Modell
`newdata`	Data Frame mit neuen Daten (optional)
`se.fit`	Standardfehler berechnen
`interval`	Intervalle, die berechnet werden sollen `"none"`: keine `"confidence"`: Konfidenzintervalle `"prediction"`: Vorhersageintervalle
`level`	Konfidenzniveau
...	...

Funktionen im Anschluss an die Regressionsanalyse

Funktionsname	**Zweck**
`resid(`*lm-obj*`)`	Residuen
`fitted(`*lm-obj*`)`	gefittete Werte
`coef(`*lm-obj*`)`	Koeffizienten
`summary(`*lm-obj*`)`	verschiedene Statistiken
`anova(`*lm-obj*`)`	Varianzanalyse
`plot(`*lm-obj*`)`	vier verschiedene Diagnostikplots
`qqnorm(`*lm-obj*`)`	Normalverteilungsplot der Residuen
`abline(`*lm-obj*`)`	Regressionsgerade zum Streudiagramm hinzufügen
...	...

Wahrscheinlichkeitsverteilungen

Verteilung	**Name in R**	**Parameter (mit Voreinstellung)**
Binomialverteilung	`binom`	`size`, `prob`
χ^2-Verteilung	`chisq`	`df`
Exponentialverteilung	`exp`	`rate=1`
F-Verteilung	`f`	`df1`, `df2`
Geometrische Verteilung	`geom`	`prob`
Hypergeometrische Verteilung	`hyper`	`m`, `n`, `k`
Normalverteilung	`norm`	`mean=0`, `sd=1`
Poissonverteilung	`pois`	`lambda`
t-Verteilung	`t`	`df`
Gleichverteilung	`unif`	`min=0`, `max=1`
...	...	...

Dem Namen der Verteilung wird nun als Präfix einer der Buchstaben `d`, `p`, `q`, `r` vorangestellt, um eine bestimmte Funktion der Wahrscheinlichkeitsverteilung aufzurufen:

Funktionsaufruf	**Wirkung**
`d`*name*`(x, ...)`	Dichtefunktion an der Stelle `x`
`p`*name*`(q, ...)`	Verteilungsfunktion an der Stelle `q`
`q`*name*`(p, ...)`	Quantilfunktion an der Stelle `p`
`r`*name*`(n, ...)`	`n` Pseudozufallszahlen

name ist der Name der Wahrscheinlichkeitsverteilung in R (s.o.).

Data Frames

- In R und S-Plus liegen Datensätze normalerweise als Data Frames vor.
- Der Data Frame ist die Datenstruktur, die auch von den meisten Funktionen zur statistischen Modellbildung vorausgesetzt wird.
- Ein Data Frame ist eine Liste von Vektoren gleicher Länge.
- Ein Data Frame ist von der Gestalt her ähnlich einer Matrix, d.h. besteht aus Zeilen und Spalten.
- Anders als bei einer Matrix dürfen die einzelnen Spalten eines Data Frame aber unterschiedlichen Datentyp haben.
- Innerhalb einer Spalte müssen die Elemente vom gleichen Datentyp sein.

Funktionen zur Erzeugung von Data Frames

Funktionsname	**Zweck**
`data.frame()`	Erzeugung eines Data Frame aus Vektoren (oder Matrizen)
`as.data.frame()`	Umwandlung einer Matrix in einen Data Frame
`read.table()`	ASCII-Datei einlesen und einen Data Frame erzeugen
...	...

Die Funktion `data.frame()`

- Mit der Funktion `data.frame()` wird aus Vektoren (oder Matrizen) ein Data Frame erzeugt.
- An erster Stelle in der Liste der Argumente (`...`) werden die Größen aufgeführt, die zu dem Data Frame zusammengefasst werden sollen.
- Vektoren in dieser Liste können auch in der Form *name=vektor* angegeben werden. Damit wird der entsprechenden Spalte des Data Frame ein Name zugewiesen.

Parameter	**Bedeutung**
`...`	Größen, aus denen der Data Frame gebildet werden soll
`row.names`	Zeilennamen
...	...

Die Funktion `read.table()`

Parameter	**Bedeutung**
`file`	Dateiangabe
`header`	T: die 1. Zeile enthält Variablennamen
`sep`	Spalten-Trennzeichen
`dec`	Dezimaltrennzeichen
`row.names`	Vektor von Zeilennamen oder Nr. bzw. Name der Spalte, die die Zeilennamen enthält
`col.names`	Vektor von Spaltennamen (Variablennamen)
`nrows`	Anzahl der einzulesenden Zeilen
`skip`	Anzahl der zu überspringenden Zeilen am Dateianfang
...	...

- Falls keine Zeilennamen angegeben sind und die Headerzeile um 1 kürzer ist als die Anzahl der Spalten, so wird die erste Spalte für Zeilennamen verwendet.

- Neben der Funktion `read.table()` gibt es noch einige weitere Funktionen zum Lesen von ASCII-Dateien: `read.csv()`, `read.csv2()`, `read.delim()`, `read.delim2()` sowie die komplexe, sehr flexible Funktion `scan()` (siehe Online-Hilfen).

Faktoren

- Ein Objekt vom Typ Faktor enthält Elemente, die zwar numerisch sein können, jedoch kategorialer Natur sind (z.B. Farbe, Geschlecht, Schicht).
- Sind die Werte einer Variablen vom Typ Character, so ist die Variable automatisch vom Typ Faktor.
- Die Werte eines Faktors werden nicht als Strings, sondern als Symbole ohne Anführungszeichen ausgegeben, auch wenn die Werte eigentlich wie Character aussehen.
- Eine Faktorvariable benötigt weniger Speicherplatz als eine Charactervariable, da die Stufen nur als natürliche Zahlen gespeichert werden[6] und die dazugehörigen Symbole in einer internen Tabelle festgehalten sind.
- Ein spezieller Typ einer Faktorvariable ist der **geordnete Faktor ('ordered')**.
- Bei diesem existiert zwischen den einzelnen Werten eine inhärente Ordnungsrelation (z.B. Schulbildung, Altersklasse).
- Defaultmäßig sind bei einem geordneten Faktor die Stufen in alphabetischer Reihenfolge aufsteigend abgelegt (entsprechend dem ASCII-Code).

Die Funktion `factor()`

Mit Hilfe dieser Funktion kann man aus einem Datenvektor einen Vektor vom Typ 'factor' bzw. 'ordered' erzeugen.

Parameter	**Bedeutung**
`x`	Datenvektor
`levels`	Vektor der Faktorstufen (aufsteigend geordnet)
`labels`	Labels für die Faktorstufen
`exclude`	Vektor von Werten, die ausgeschlossen werden sollen
`ordered`	T: es wird ein geordneter Faktor erzeugt

- Obligat ist lediglich der erste Parameter.
- Neben `factor()` gibt es noch eine Funktion `ordered()`, die einen geordneten Faktor erzeugt.

[6] Dies zeigt sich, wenn man auf einen Faktor die Funktion `as.integer()` anwendet.

Weitere Funktionen für Faktoren

Funktionsname	**Zweck**
`cut`	Kreieren eines Faktors durch Klasseneinteilung
`gl`	Kreieren von Faktoren durch Angabe der Stufen und Replikationen
`split`	Erzeugung eines Faktors aus einem Vektor durch Klasseneinteilung
`unsplit`	`split()` rückgängig machen
`levels`	Faktorstufen abfragen oder definieren
`nlevels`	Anzahl der Faktorstufen abfragen
`relevel`	Referenzstufe definieren
`expand.grid`	Erzeugen eines Data Frames aus allen Kombinationen von abgegebenen Variablen
...	...

Varianzanalyse

Mit Hilfe der Funktion `aov()` läßt sich ein Varianzanalysemodell anpassen.

Parameter	**Bedeutung**
`formula`	Modellformel
`data`	Data Frame
`contrasts`	Kontraste
...	...

- Obligat ist lediglich der erste Parameter.
- Zurückgegeben wird ein `aov`-Objekt, das man unter einem Variablenamen abspeichern kann.
- Ein `aov`-Objekt ist wie ein `lm`-Objekt eine Liste, die aus mehreren (benannten) Komponenten besteht.
- Auf die einzelnen Komponenten kann man also mit dem $-Operator zugreifen.
- Auf `aov`-Objekte kann mit denselben Funktionen zugegriffen werden wie auf ein `lm`-Objekt (siehe oben).

Konvertierungsfunktionen

- In R hat jedes Objekt einen Typ oder, wie man in R sagt, einen sog. "**Mode**" (bzw. eine Objektklasse).
- Mit Hilfe von Konvertierungsfunktionen kann man prüfen, ob ein Objekt von einem bestimmten Mode ist oder es kann ihm ein neuer Mode zugewiesen werden.
  Funktionen zum Prüfen eines Objekts:
  `is.`*mode*`(`*objekt*`)`
  Funktionen zum Konvertieren eines Objekts:
  `as.`*mode*`(`*objekt*`)`
  Nach dem Punkt ist jeweils der gewünschte Modename einzusetzen.

Die Objektmodi von R

Modename	**Bedeutung**
`call`	Funktionsaufruf
`character`	Character (Zeichen)
`complex`	komplexe Zahl
`double`	doppelt genaue Zahl
`expression`	Ausdruck
`factor`	Faktor
`function`	Funktion
`integer`	ganze Zahl
`list`	Liste
`logical`	logisch
`matrix`	Matrix
`name`	Name
`NULL`	Null-Objekt
`numeric`	numerisch
`ordered`	geordneter Faktor
`single`	einfache Genauigkeit
`symbol`	Symbol
`table`	Tabelle
`ts`	Zeitreihe
`vector`	Vektor
...	...

Funktionen für Zufallszahlen

- Mit Hilfe der Funktionen `runif()`, `rnorm()` usw. kann man (Pseudo-)Zufallszahlen erzeugen.
- Darüber hinaus gibt es in R noch zwei weitere interessante Zufallsfunktionen: `sample()` und `set.seed()`.

Die Funktion `sample()`

- Mit Hilfe der Funktion `sample()` kann man eine Zufallsstichprobe aus einem Vektor von Werten ziehen.
- Dabei kann man zwischen Ziehen mit und Ziehen ohne Zurücklegen[7] wählen.

Parameter	**Bedeutung**
`x`	Vektor oder positive ganze Zahl
`size`	Anzahl der Elemente, die gezogen werden sollen
`replace`	Ziehen ohne (F) bzw. mit (T) Zurücklegen
`prob`	Vektor von Wahrscheinlichkeiten

Ist `x` eine Zahl, so ist die Wirkung dieselbe wie bei Eingabe von `1:x`.

Die Funktion `set.seed()`

- Mit Hilfe der Funktion `set.seed` kann der Pseudozufallszahlengenerator initialisiert, d.h. auf einen festen Startwert gesetzt werden.
- Auf diese Weise ist es möglich, dieselbe Folge von Pseudozufallszahlen zu erzeugen, um z.B. die Ergebnisse von Monte Carlo Studien zu überprüfen.

Parameter	**Bedeutung**
`seed`	Initialisierungswert
`kind`	Arbeitsmodus

Der Initialwert dient zum Initialisieren, der Arbeitsmodus definiert den Algorithmus, der bei der "Berechnung" der Zufallszahlen verwendet werden soll.

[7] d.h. damit erzeugt man eine zufällige Permutation.

Indizierung

Zugriff auf Elemente eines Vektors

Auf einzelne Elemente eines Vektors wird mit Hilfe von Indizes zugegriffen:

vektor[*indexvektor*]

- Für *indexvektor* kann eine Zahl oder ein numerischer Vektor verwendet werden.
- Für einen Indexvektor gibt es mehrere Möglichkeiten:
  - Positiver numerischer Vektor: die entsprechenden Elemente werden ausgewählt.
  - Negativer numerischer Vektor: die entsprechenden Elemente werden weggelassen.
  - Logischer Vektor: die Elemente an der TRUE-Position werden genommen
  - Ist der Indexvektor kürzer als die Länge von `vektor`, so wird er zyklisch erweitert.
- Haben die Vektorelemente einen Namen, so können auch diese zur Indizierung verwendet werden.

Zugriff auf Teile eines Data Frame bzw. einer Matrix

Die Syntax ist

matrix[*zeilenindexvektor*,*spaltenindexvektor*]
bzw.
dataframe[*zeilenindexvektor*,*spaltenindexvektor*]

- Die Indizes sind entweder Zahlen oder Vektoren.
- Fehlt der Zeilen- bzw. der Spaltenindex, so wird die gesamte Zeile bzw. Spalte ausgewählt.
- Ist ein Zeilen- bzw. Spaltenindex negativ, so werden die entsprechenden Elemente weggelassen.
- Wie bei den Vektoren ist auch bei Matrizen die Indizierung mit Hilfe von logischen Werten oder Zeilen- bzw. Spaltenamen möglich.

Matrizen und Arrays

- Eine Matrix ist eine zweidimensionale Anordnung von Elementen, also eine Verallgemeinerung eines Vektors.
- Im Unterschied zu einem Data Frame sind bei einer Matrix sämtliche Elemente vom gleichen Typ ('Mode').
- Ein Array ist eine weitere Verallgemeinerung, nämlich eine drei- oder mehrdimensionale Anordnung von Elementen des gleichen Typs ('Modes')
- Ein Array kann mit Hilfe der Funktion `array()` erzeugt werden.

Attribute einer Matrix

Attribut	**Bedeutung**
`dim`	Dimension der Matrix
`rownames`	Zeilennamen
`colnames`	Spaltennamen

Indizierung von Matrixelementen

Die Syntax ist: *matrix*[*zeilenindizes*,*spaltenindizes*]

- Die Indizes sind entweder einzelne Zahlen oder Vektoren.
- Fehlen die Zeilen- bzw. Spaltenindizes, so wird die gesamte Zeile bzw. Spalte ausgewählt.
- Negative Zeilen- bzw. Spaltenindizes bewirken, dass die entsprechenden Elemente weggelassen werden.
- Wie bei den Vektoren ist auch bei Matrizen die Indizierung mit Hilfe von logischen Werten oder Zeilen- bzw. Spaltenamen möglich.

Matrizenoperationen

Operation	**Bedeutung**
`+`	elementweise Addition
`-`	elementweise Subtraktion
`*`	elementweise Multiplikation
`/`	elementweise Division
`^`	elementweise Exponentiation
`%*%`	Matrizenmultiplikation
`%o%`	Äußeres Produkt (mit der Multiplikation als binärer Operation)

Funktionen für Matrizen

Funktion	**Bedeutung**
`matrix`	Erzeugung einer Matrix aus einem Vektor
`as.matrix`	Umwandlung in eine Matrix
`is.matrix`	Prüfung, ob das Objekt eine Matrix ist
`cbind`	spaltenweises Zusammenfügen
`rbind`	zeilenweises Zusammenfügen
`subset`	Auswahl
`diag`	Erzeugung einer Diagonalmatrix bzw. Extraktion der Diagonale
`solve`	Inverse bzw. Lösung eines linearen Gleichungssystems
`t`	Transponierte
`det`	Determinante
`outer`	äußeres Produkt
`kronecker`	Kroneckerprodukt
`apply`	Mappingfunktion
...	...

Die Funktion `apply()`

Die Funktion `apply()` erlaubt es, eine beliebige Funktion auf alle Zeilen oder Spalten oder Elemente einer Matrix anzuwenden.

Parameter	Bedeutung
`X`	Matrix
`MARGIN`	Dimension(en), auf die die Funktion angewandt werden soll
`FUN`	Name bzw. Rumpf der anzuwendenden Funktion
`...`	Optionale Argumente für die Funktion

Funktionen in R

- Das Arbeiten mit R besteht in der Hauptsache im Aufruf von Funktionen.
- Die Mächtigkeit von R steckt im wesentlichen in seinen eingebauten und benutzerdefinierten Funktionen.
- Eine der Stärken von R ist es, dass es kein starres System ist. Vielmehr kann der Benutzer dieses System durch Definition eigener Funktionen erweitern und an seine Bedürfnisse anpassen.
- In R gibt es eine Reihe von Funktionen, die sog. "primitives", die in einer höheren Programmiersprache (z.B. in C) geschrieben und vorkompiliert sind. Andere Funktionen sind in R selbst geschrieben[8].
- Der Quellcode einer in R geschriebenen Funktion wird ausgegeben, wenn man den Funktionsnamen ohne Klammern eingibt.

Syntax eines Funktionsaufrufs

funktionsname(*argumente*)

- Es wird der Name der Funktion angegeben, gefolgt von der Liste der Argumente in runden Klammern.
- Ist die Liste der Argumente leer, so ist ein leeres Klammernpaar zu setzen.

Parameter von Funktionen

- Man unterscheidet Stellungs- und Schlüsselwortparameter.
- Stellungsparameter sind definiert durch die Stellung, d.h. die Reihenfolge innerhalb der Parameterliste.
- Schlüsselwortparameter sind definiert durch den Namen des Parameters.
- In R sind beide Aufrufmodi verwendbar (auch gemischt), d.h. Funktionsparameter sind sowohl durch ihren Namen als auch durch ihre Reihenfolge ansprechbar.
- Schlüsselwortparameter werden in folgender Weise spezifiziert:
  schlüsselwortname=wert
- Der Schlüsselwortnamen kann abgekürzt werden, sofern die Eindeutigkeit gewahrt ist.
- Weiterhin unterscheidet man obligate vs. optionale Parameter.
- Bei manchen Parametern gibt es Voreinstellungen (defaults). Eine Voreinstellung tritt in Kraft, falls das entsprechende Argument weggelassen wird.
- Manche Funktionen rufen ihrerseits andere Funktionen auf. An diese können irgendwelche Argumente weitergereicht werden, falls in der Funktionsdefinition der Parameter '...' angegeben ist.

Benutzerdefinierte Funktionen

Definition eines Funktionsobjekts

`function(`*`parameter`*`)` *`funktionsrumpf`*

Der Funktionsrumpf besteht entweder aus einer einzelnen Anweisung oder aus einem Block von mehreren Anweisungen. In diesem Fall sind geschweifte Klammern zu verwenden:

[8] R ist selbst eine Programmiersprache.

```
function(parameter) {
  anweisungen
}
```

In der bisherigen Form ist die Funktion eine anonyme Funktion.

Um eine Funktion auch verwenden zu können, wird das Funktionsobjekt einer Variable zugewiesen, und so erhält man eine benannte Funktion:

```
funktionsname <- funktionsobjekt
```

Ein benutzerdefinierte Funktion wird im allgemeinen also folgende Gestalt haben:

```
funktionsname <- function(parameter) {
  anweisungen
}
```

Parameter und Voreinstellungen

- In der Funktionsdefinition können auch Voreinstellungen für Parameter definiert werden.
- Voreinstellungen werden in der Parameterliste (d.h. innerhalb der runden Klammern) definiert. Dazu lässt man nach dem Namen des Parameters dessen Voreinstellung folgen:
  `parameter=default`
- Wird dann beim Aufruf der Funktion das entsprechende Argument weggelassen, so wird der voreingestellte Wert genommen.

Parameterübergabe an Funktionen

- Beim Aufruf einer Funktion werden die Werte der Aufrufargumente auf die Funktionsparameter kopiert.
- Die Zuordnung eines Arguments zu einem bestimmten Parameter erfolgt entweder anhand seiner Stellung innerhalb der Argumentenliste oder anhand des entsprechenden Schlüsselworts.
- Die meisten Funktionen geben einen Wert zurück, nämlich den Wert des letzten in der Funktion evaluierten Ausdrucks.
- Für die explizite Werterückgabe gibt es auch eine eigene Funktion, nämlich die Funktion `return()`.

Die Funktion `return()`

- Mit dem Aufruf der `return`-Funktion wird die laufende Funktion sofort verlassen und der angegebene Wert an die aufrufende Funktion zurückgegeben.
- Normalerweise wird der Rückgabewert auch (auf dem Bildschirm) ausgegeben. Will man einen Wert zwar zurückgeben, aber dessen Ausgabe unterdrücken, so verwendet man die Funktion `invisible()`.

Gültigkeit von Variablen ("scope")

- Die Variablen und deren Werte bilden in R die sogenannte "Umgebung" (environment, namespace), innerhalb deren der Interpreter Ausdrücke auswertet.
- In der interaktiven Umgebung (d.h. in der Kommandozeile) definierte Variablen sind global und

damit in allen Funktionen definiert[9].

- Alle in einer Funktion definierten Variablen sind lokale Variablen.
- Auch die Parameter einer Funktion sind lokale Variablen, d.h. nur innerhalb der Funktion definiert.
- Lokale Variablen verdecken globale Variablen mit gleichem Namen.
- In R existieren verschiedene Funktionen, die es erlauben, die Umgebung zu manipulieren bzw. zu setzen.

[9] Von der Verwendung globaler Variablen in Funktionen ist jedoch abzuraten; statt dessen wird empfohlen, sie als Argumente zu übergeben.

Kontrollstrukturen

Folgende Befehle stehen in R zur Verfügung:

- `{}`: Anweisungsblock
- `if`: Bedingte Anweisung
- `switch`: Fallunterscheidung
- `repeat`-Schleife
- `while`-Schleife
- `for`-Schleife
- `next`: nächster Schleifendurchgang
- `break`: Schleifenabbruch

Neben diesen iterativen Kontrollstrukturen ist es in R auch möglich, bei der Definition von Funktionen die Kontrollstruktur der Rekursion einzusetzen.

Anweisungsblock

```
{
  anweisungen
}
```

- Anweisungen, die in geschweiften Klammern eingeschlossen sind, definieren einen Anweisungsblock.
- Ein Anweisungsblock kann überall dort stehen, wo eine einzelne Anweisung stehen kann.

`if, else`

```
if (condition) expression1 [else expression2]
```

- Ist das Ergebnis von *condition* `TRUE` (oder ein von 0 verschiedener numerischer Wert)[10], so wird *expression1* ausgeführt, ansonsten - falls `else` vorhanden ist - *expression2*.
- Liefert die Bedingung einen Vektor, so wird nur die erste Komponente verwendet und eine Warnung ausgegeben.
- Der `else`-Zweig kann wiederum eine `if`-Anweisung enthalten.
- Sind für *expression1* bzw. *expression2* mehrere Anweisungen auszuführen, so definiert man sie als Anweisungsblock, d.h. setzt sie in geschweifte Klammern.
- Für die Formulierung einer Bedingung stehen neben den Vergleichsoperatoren (`==`, `!=`, `<`, `>`, `<=`, `>=`) auch die logischen Operatoren (`!`, `&&`, `||`) zur Verfügung.

`switch`

```
switch (EXPR,...)
```

- Nach dem Parameter `EXPR` sind unter `...` die verschiedenen Alternativen aufzuführen.

[10] Das Ergebnis darf allerdings nicht `NA` sein.

- Ist der Wert von EXPR eine Zahl n,
  - so wird das n-te Element von ... ausgewertet und das Ergebnis zurückgegeben.
  - Ist n allerdings keine Zahl zwischen 1 und der Anzahl der Argumente bei ..., so wird NULL zurückgegeben.
- Ist der Wert von EXPR ein String,
  - so wird der Wert jener Alternative zurückgegeben, deren Namen genau dem String entspricht; ist deren Wert jedoch leer, so wird der Wert des ersten nachfolgenden nicht-leeren Elements verwendet.
  - Befindet sich unter den Alternativen keine mit dem gesuchten Namen, so wird das erste unbenannte Element zurückgegeben (falls es unbenannte Elemente gibt) bzw. NULL.

`repeat`

```
repeat expression
```

- Diese Konstruktion stellt eigentlich eine Endlosschleife dar.
- Aus einer repeat-Schleife kommt man mit Hilfe einer break-Anweisung heraus, die ihrerseits Teil einer if-Anweisung ist.
- Eine repeat-Schleife wird im allgemeinen also so aussehen:
  ```
  repeat {
  ...
  if ... break
  ...
  }
  ```
- Eine repeat-Schleife wird also mindestens einmal durchlaufen (oder zumindest einmal begonnen).

`while`

```
while (condition) expression
```

- expression wird solange ausgeführt, wie die angegebene Bedingung wahr ist.
- Liefert die Bedingung gleich zu Beginn FALSE, so wird die Schleife überhaupt nicht durchlaufen.
- Jede repeat-Schleife lässt sich auch als while-Schleife formulieren.

`for`

```
for (variable in sequence) expression
```

- Bei der for-Schleife durchläuft der angegebene Ausdruck (bzw. Anweisungsblock) mehrere Iterationen, wobei die angegebene Schleifenvariable der Reihe nach alle Werte von sequence annimmt[11].
- sequence kann eine Variable, ein Vektor, eine Matrix oder auch eine Liste sein.

[11] Im Gegensatz zur while- und repeat-Schleife ist hier die Anzahl der Durchläufe zu Beginn der Schleife bekannt.

- Soll eine Schleife für alle Elemente von x durchlaufen werden, so lässt sich anstatt
  `for (i in 1:length(x))` auch
  `for (y in seq(along=x))`
  verwenden. Das jeweilige Element von x ist dann einfach mit y (anstatt mit `x[i]`) anzusprechen.

`next`

```
next
```

- Die next-Anweisung wird in einer repeat-Schleife verwendet, um den aktuellen Schleifendurchlauf abzubrechen und wieder an den Schleifenanfang zu springen.

`break`

```
break
```

- Die break-Anweisung wird in einer repeat-Schleife verwendet, um die Schleife abzubrechen. Dies geschieht praktisch immer im Zusammenhang mit der Überprüfung eines Abbruchkriteriums mit Hilfe einer if-Anweisung.

Iteration vs. Rekursion

- Die bisher betrachteten Kontrollstrukturen gehören zu Gruppe der iterativen Kontrollstrukturen.
- In R gibt es auch die Möglichkeit, rekursive Funktionen zu definieren.
- Eine rekursive Funktion ist eine Funktion, die sich direkt oder indirekt selbst aufruft.
- Rekursive Funktionen sind zwar eleganter als iterative, aber auch langsamer und speicherintensiver.
- Es gibt Probleme, die sich mit iterativen Funktionen nicht oder nur sehr umständlich lösen lassen (Beispiel: Turm von Hanoi).

Beispiel: Berechnung der Fakultät (iterative Lösung)

```
fak <- function(n) {
  p <- 1
  for ( i in 1:n )
    p <- p * i
  return(p)
}
```

Beispiel: Berechnung der Fakultät (rekursive Lösung)

```
fak <- function(n) {
  if( n == 0 )
    1
  else
    n * fak( n - 1 )
}
```

Listen

- Die Liste ist eine sehr flexible Datenstruktur.
- Sie ist eine geordnete Menge von beliebig vielen Elementen.
- Die Elemente können von beliebigem Typ bzw. beliebiger Struktur sein.
- Ein Data Frame ist übrigens eine Liste von Vektoren gleicher Länge.
- Die Funktionen zur statistischen Modellbildung geben ihr Ergebnis im allgemeinen in Form einer Liste zurück.

Funktionen für Listen

Funktion	**Bedeutung**
`list`	Erzeugung einer Liste aus angegebenen Elementen
`unlist`	Erzeugung eines Vektors (oder einer Liste niedrigerer Ordnung) aus einer Liste
`lapply`	Mappingfunktion
`sapply`	Mappingfunktion mit Vereinfachung des Ergebnisses
...	...

Die Funktion `list()`

Die Funktion `list()` ist die Konstruktorfunktion für eine Liste.

Aufruf der Funktion: `list(...)`

- In der Liste der Argumente (...) werden die Größen aufgeführt, die zu der Liste zusammengefasst werden sollen.
- Die Elemente können auch in der Form *name=element* angegeben werden. Damit wird dem entsprechenden Element innerhalb der Liste ein Name zugewiesen.
- Eine Benennung von Listenelementen kann auch nachträglich mit Hilfe der Funktion `names()` vorgenommen werden.

Zugriff auf Listenelemente

- Mit Hilfe eines Index oder Namens in doppelten eckigen Klammern erhält man das entsprechende Element der Liste:
  liste`[[`*i*`]]` bzw. *liste*`[['`*elementname*`']]`

- Mit Hilfe des $-Operators erhält man das Element mit dem angegebenen Namen:
  liste`$`*elementname*

- Mit Hilfe eines Index bzw. Namens in einfachen eckigen Klammern erhält man eine Liste mit dem entsprechenden Element der Liste[12]:
  liste`[`*i*`]` bzw. *liste*`['`*name*`']`

[12] Hier kann auch ein (positiver oder negativer) Indexvektor oder ein Namensvektor angegeben werden; dann werden die entsprechenden Elemente zusammen in eine Liste gepackt.

- Listenelemente werden aus einer Liste entfernt, wenn man ihnen den Wert `NULL` zuweist.
- Wird ein Element aus einer Liste entfernt, so rücken die nachfolgenden Elemente in der Reihenfolge sofort nach[13].

Die Funktion `lapply()`

Analog zur Funktion `apply()`, mit deren Hilfe man eine Funktion auf alle Spalten und/oder Zeilen einer Matrix anwenden kann, gibt es für Listen die Funktion `lapply()`.

Parameter	**Bedeutung**
`x`	Liste
`FUN`	Name bzw. Rumpf der anzuwendenden Funktion
`...`	Argumente für die Funktion (optional)

- Das Ergebnis dieser Funktion ist eine Liste gleicher Länge wie `x`.
- Die Elemente der Rückgabeliste sind jeweils die Ergebnisse der angegebenen Funktion, angewandt auf das jeweilige Listenelement von `x`.

Die Funktion `sapply()`

Die Funktion `sapply()` arbeitet genauso wie die Funktion `lapply()` mit dem Unterschied, dass die Rückgabe bezüglich der Datenstruktur vereinfacht wird, d.h. die Rückgabe kann anstatt in Form einer Liste die Form einer Matrix oder eines Vektor erfolgen.

Parameter	**Bedeutung**
`x`	Liste oder Vektor
`FUN`	Name bzw. Rumpf der anzuwendenden Funktion
`...`	Optionale Argumente für die Funktion
`simplify`	T: wenn möglich, soll ein Vektor zurückgegeben werden
`use.names`	T: ist x ein Charactervektor, wird er zur Namensgebung für die Rückgabe verwen

[13] Dies ist zu beachten, wenn man mehrere Elemente nacheinander löschen will.

Signifikanztests

Funktion	**Wirkung**
`bartlett.test`	Bartlett's Test auf Varianzhomogenität
`binom.test`	Binomialtest
`chisq.test`	Chiquadrattest
`cor.test`	Test einer Korrelation gegen Null
`fisher.test`	Fisher's exakter Test
`friedman.test`	Friedman's Rangsummentest
`kruskal.test`	Rangsummentest nach Kruskal-Wallis
`ks.test`	Test von Kolmogorov-Smirnov für eine bzw. zwei Stichproben
`mcnemar.test`	Test von McNemar
`pairwise.t.test`	paarweise Mittelwertsvergleiche
`pairwise.prop.test`	paarweise Vergleiche von Anteilswerten
`pairwise.wilcox.test`	paarweise Mittelwertsvergleiche (nonparametrisch)
`power.anova.test`	Teststärke für ANOVA
`power.prop.test`	Teststärke des Tests für Anteilswerte (`prob.test()`)
`power.t.test`	Teststärke des t-Tests
`prop.test`	Test von relativen Häufigkeiten
`oneway.test`	einfache Varianzanalyse
`t.test`	t-Test für eine oder zwei Stichproben
`var.test`	F-Test für zwei Varianzen
`wilcox.test`	Wilcoxon Rangsummen Test
...	...

Stringfunktionen

R verfügt viele, zum Teil sehr mächtige Funktionen für die Arbeit mit Strings.

Funktion	**Wirkung**
`paste`	Objekte in Character umwandeln und verketten
`strsplit`	Strings in Teilstrings zerlegen
`match`, `charmatch`, `pmatch`	Werte in Strings suchen
`substr`, `substring`	Teilstring extrahieren bzw. ersetzen
`grep`, `regexpr`	Suchen mit Hilfe eines regulären Ausdrucks
`sub`, `gsub`	Suchen und Ersetzen mit einem regulären Ausdruck
`tolower`, `toupper`	In Klein- bzw. Großbuchstaben umwandeln
`chartr`	Zeichen ersetzen
`nchar`	Anzahl der Zeichen in einem Charactervektor
...	...

Die Funktion `paste()`

Parameter	**Bedeutung**
...	Objekte
`sep`	Trennstring für die Verkettung der Objekte
`collapse`	Trennstring für die Verkettung von Ergebnisvektoren

- Die angegebenen Objekte werden in Strings umgewandelt und zu einem einzelnen String verkettet.
- Bei der Verkettung wird zwischen die einzelnen Teilstrings die bei `sep` spezifizierte Zeichenkette gesetzt.
- Sind die angegebenen Objekte Vektoren, so werden diese elementweise parallel - bei Bedarf zyklisch - durchlaufen und entsprechende Teilstrings gebildet, d.h. das Ergebnis ist ein Charactervektor (dieser ist gleich lang wie der längste angegebene Vektor).
- Dieser Charactervektor wird wiederum zu einem Gesamtstring verkettet, falls der Parameter `collapse` angegeben ist (dessen Wert dient dann als Trennstring für die einzelnen Teile).

Ein-/Ausgabe

- Anstatt von der Konsole können Kommandos auch aus einer ASCII-Datei eingelesen und interpretiert werden.
- Ebenso können Datenobjekte oder sogar das gesamte Arbeitsverzeichnis auf externe Dateien geschrieben und von dort wieder gelesen werden.
- Für die Ausgabe von Ergebnissen gibt es weiterhin vielfältige Möglichkeiten der Formatierung.

Ein-/Ausgabe-Funktionen

Funktion	**Wirkung**
`source`	Skriptdatei ausführen
`sink`	Ausgabe auf eine Datei umleiten
`save`	Objekt binär auf eine Datei schreiben
`save.image`	Arbeitsverzeichnis binär auf eine Datei schreiben
`load`	Arbeitsverzeichnis bzw. Objekt von einer Datei einlesen
`write`	Daten (z.B. eine Matrix) im ASCII-Format auf eine Datei schreiben
`write.table`	Objekt als Dataframe im ASCII-Format auf eine Datei schreiben
`read.table`	Dataframe von einer ASCII-Datei lesen
`print`	Ausgabe eines Ausdrucks
`cat`	Umwandlung von Objekten in Strings und deren Ausgabe
`format`	Ausgabe eines Objekts mit vorheriger Formatierung
`formatC`	Formatierung im C-Stil
`unlink`	Datei(en) oder Verzeichnis(se) löschen
...	...

Die Funktion `cat()`

Parameter	**Bedeutung**
`...`	Objekte
`file`	Ausgabedatei (falls leer, Ausgabe auf Bildschirm)
`sep`	Trennzeichen zwischen den einzelnen Objekten
`fill`	F: Zeilenvorschübe werden nicht automatisch erzeugt
`labels`	Zeilenlabels
`append`	T: Ausgabe an Datei anhängen (nur, falls `file` nicht leer)

- Die Funktion `cat()` ist vor allem in Benutzerfunktionen nützlich.
- Sie wandelt die angegebenen Objekte in Strings um, verkettet sie unter Verwendung des Trennzeichens und gibt sie aus.
- Für Zeilenvorschübe verwendet man die Sequenz `"\n"`.

Tabellen

- Die Verteilung zwei- oder mehrdimensionaler kategorialer Daten wird mit Hilfe von Tabellen beschrieben.
- In R gibt es dafür den Objekttyp 'table'.
- Zur Erstellung von Tabellen steht in R die Funktion `table()` u.a. zu Verfügung.
- Die Funktionen `tapply()` und `by()` erlauben die Anwendung beliebiger Funktionen auf Untergruppen von Daten und Rückgabe der Ergebnisse als Tabelle bzw. Liste.

Funktionen für Tabellen

Funktion	**Wirkung**
`table`	Erzeugung einer ein- oder mehrdimensionalen Häufigkeitstabelle
`margin.table`	Erstellung einer Randtabelle
`prop.table`	Erstellung einer Randtabelle von Prozentwerten
`tapply`	Mappingfunktion
...	...

Die Funktion `tapply()`

Die Funktion `tapply()` erlaubt es, eine beliebige Funktion auf alle durch einen bzw. mehrere Faktoren definierten Untergruppen anzuwenden.

Parameter	**Bedeutung**
`X`	Vektor
`INDEX`	Faktor oder Liste von Faktoren (jeweils von gleicher Länge wie X)
`FUN`	Name bzw. Rumpf der anzuwendenden Funktion
`...`	Optionale Argumente für die Funktion
`simplify`	F: Ergebnis als Liste, sonst als Vektor, Matrix bzw. Array

- Für jede Kombination der Faktorstufen (Parameter `INDEX`) wird die angegebene Funktion angewandt.
- Die Rückgabe ist ein Array (Vektor, Matrix, ...) mit genau so vielen Dimensionen, wie Faktoren angegeben wurden.
- Neben `tapply()` gibt es noch eine Funktion, die eine ähnliche Wirkung hat, sich aber auf Dataframes anwenden lässt: `by()`.

Daten

Verfügbare Datensätze in R

In R stehen Datensätze aus den verschiedensten Anwendungsgebieten zur Verfügung. Man unterscheidet:

- Eingebaute Datensätze
- Datensätze in Packages

Unterstützte Datenformate

Zur Zeit werden 4 Datenformate unterstützt:

- Dateien mit der Extension `.R` oder `.r` können mit der Funktion `source()` eingelesen und interpretiert werden; das Arbeitsverzeichnis wechselt vorübergehend in das Verzeichnis dieser Datei.
- Dateien mit der Extension `.RData` oder `.rda` werden mit der Funktion `load()` eingelesen.
- Textdateien (Extension `.tab`, `.txt` u.a.) werden mit Hilfe der Funktion `read.table()` eingelesen; das Ergebnis ist ein Data Frame.
- CSV-Dateien[14] werden ebenfalls mit Hilfe der Funktion `read.table()` eingelesen und zwar mit folgendem Aufruf:
  `read.table(`*`dateireferenz`*`,header=TRUE,sep="`*`trennzeichen`*`")`.

Unterstützte Fremdformate

- Das Paket `foreign` enthält Funktionen, um Datendateien von anderen Statistiksystem (S, SAS, SPSS, Stata, Systat, Minitab, dBase, u.a.) zu lesen .
- Andere Pakete erlauben es z.B., Daten aus SQL- bzw. ODBC-Datenbanken einzulesen.

[14] Textdateien mit Datenfeldern, die üblicherweise mit Komma oder Semikolon getrennt sind

Packages

- Ein großer Teil von R liegt in Form von sog. Paketen vor.
- Einige Pakete gehören zur Basisinstallation.
- Im Internet (z.B. im CRAN-Archiv) steht eine große Zahl zusätzlicher Pakete zum Download bereit; diese können auf einfachem Wege nachinstalliert werden.
- Ein Paket kann enthalten:
  - Funktionen, die in R geschrieben sind,
  - DLLs (dynamische Bibliotheken) von kompiliertem Code[15],
  - Datensätze

Funktionen für Pakete

Funktion	**Wirkung**
`library`	Paket laden bzw. Informationen zu einem Paket anfordern
`require`	Paket innerhalb einer Funktion laden
`.packages`	Namen der geladenen bzw. installierten Pakete auflisten
`.libPaths`	Wurzelverzeichnispfad der installierten Pakete
...	...

Hinweis

- Neben den aufgeführten Funktionen existiert noch die Stringvariable `.Library`: in ihr ist das voreingestellte Library-Wurzelverzeichnis, d.h. das '`library`'-Unterverzeichnis des Programmverzeichnisses von R, gespeichert (siehe `.libPaths()`).

[15] Funktionen, die meist in C oder FORTRAN geschrieben sind.

Behandlung von fehlenden Werten in R

Fehlende Werte werden in R durch die Konstante `NA` repräsentiert.

Funktion	**Wirkung**
`is.na`	auf NA testen bzw. NA ersetzen
`complete.cases`	Fälle auf fehlende Werte prüfen
`missing`	Prüfung, ob ein Funktionsargument beim Aufruf fehlte
`na.fail`	haben alle Fälle vollständige Daten, wird das Objekt zurückgegeben, sonst ein Fehler gemeldet
`na.omit`	Fälle mit fehlenden Werten werden entfernt
`na.exclude`	Fälle mit fehlenden Werten werden entfernt; bei `naresid` und `napredict` werden für die entfernten Fälle `NA`s ausgegeben, um die korrekte Länge beizubehalten
`na.pass`	das Objekt wird unverändert zurückgegeben
`na.action`	Informationen über den Umgang mit NA beim Kreieren eines Objekts

Hinweis

- Ersetzung von NA:
  `is.na(`*x*`) <-` *value*

Funktionen für Datum und Zeit

Funktion	Wirkung
`Sys.time`	Datum und Uhrzeit incl. Zeitzone
`Sys.timezone`	Zeitzone
`date`	Datum
`system.time`	Zeitbedarf für die Auswertung eines Ausdrucks
`proc.time`	Zeitverbrauch für den laufenden R-Prozess

Dateisystemfunktionen

Funktion	Wirkung
`dir.create`	Verzeichnis kreieren
`file.create`	Datei(en) kreieren
`file.exists`	Test, ob Datei(en) existieren
`file.info`	Dateiinformationen
`file.access`	Informationen über Zugriffsrechte
`list.files`, `dir`	Liste der Dateien in einem Verzeichnis
`system.file`	vollständiger Pfad eines Packages bzw. Files
`file.rename`	Datei umbenennen
`file.append`	Datei(en) anhängen
`file.copy`	Datei(en) kopieren
`file.symlink`	symbolischen Link erzeugen
`file.remove`	Datei(en) löschen

Systemfunktionen

Funktion	Wirkung
`system`	Aufruf eines Betriebssystemkommandos
`shell`	Kommando unter einer Shell ausführen
`shell.exec`	Übergabe einer Datei an die mit ihr verknüpfte Applikation

Der R Interpreter, Objekte und Umgebungen

- R ist eine Interpretersprache.
- Damit bietet R dem Benutzer eine Reihe von Möglichkeiten bei der Evaluierung von Ausdrücken. Beispielsweise können Anweisungen dynamisch zur Laufzeit erzeugt und ausgeführt werden.
- Bei der Evaluierung eines Ausdrucks wird die sog. Umgebung nach den Variablen abgesucht, die in dem Ausdruck auftauchen, um deren Wert zu ermitteln.
- Zusätzlich verwendet R noch den sog. Suchpfad, um Variablen bzw. Funktionen zu finden.

Frames und Umgebungen

- Ein Frame ist die Menge der lokalen Variablen, die beim Aufruf einer Funktion kreiert werden und beim Verlassen der Funktion wieder verschwinden.
- Eine Umgebung ist eine Verschachtelung von Frames bzw. der innerste Frame incl. seiner umschließenden Umgebung.
- Die globale Umgebung ist das Benutzer-Arbeitsverzeichnis. Bei Wertzuweisungen an der Kommandozeile werden Objekte der globalen Umgebung angesprochen.
- Die Umgebung einer Funktion ist jene Umgebung, die zur Zeit der Definition der Funktion aktiv war[16].
- Umgebungen kann man auch Variablen zuweisen.
- Eine Umgebung kann durch eine Wertzuweisung manipuliert werden.

Der Suchpfad

Ausdrücke in R bestehen aus Namen und Symbolen. Für deren Evaluierung durchläuft der R-Interpreter den sog. Suchpfad, das ist die Liste der zur Zeit definierten Objekte und geladenen Pakete. Es stehen mehrere Funktionen zur Verfügung, um den Suchpfad aufzulisten bzw. ihn zu verändern:

Funktion	**Wirkung**
`search`	Liste der Objekte und Packages im Suchpfad
`searchpaths`	Liste der Objekte und Packages (mit Pfad) im Suchpfad
`attach`	Dataframe in den Suchpfad aufnehmen
`detach`	Dataframe aus dem Suchpfad entfernen

- Der Suchpfad umfasst an erster Stelle die globale Umgebung (`".GlobalEnv"`), dann die geladenen Daten und Pakete und an letzter Stelle das Basispaket (`"package:base"`).
- Die Reihenfolge der einzelnen Einträge im Suchpfad bestimmt die Reihenfolge, in der sie vom Interpreter durchsucht werden.
- Umgebungen werden mit Hilfe der Funktionen `attach()` oder `library()` in den Suchpfad eingefügt.
- Mit Hilfe der Funktion `search()` kann der aktuelle Suchpfad ausgegeben werden.

[16] Man spricht hier von 'lexikalischem Scope'. Daneben gibt es auch 'dynamischen Scope'.

Objekte

Zum Umgang mit Objekten stehen verschiedene Funktionen zur Verfügung:

Funktion	**Wirkung**
`ls`, `objects`	Objekte einer Umgebung auflisten
`ls.str`	detaillierte Informationen über die Objekte einer Umgebung
`class`	Klasse (mode) eines Objekts abfragen bzw. setzen
`str`	detaillierte Informationen über ein Objekt
`apropos`	Objekte auflisten, deren Namen ein angegebenes Muster enthalten
`find`	Pakete auflisten, die angegebene Objekte enthalten
`help.search`	Suche im Online-Hilfesystem
`remove`, `rm`	Objekte löschen

Interpreterfunktionen

Funktion	**Wirkung**
`expression`	Objekte vom Typ `expression` kreieren
`eval`	Ausdruck in einer Umgebung evaluieren
`environment`	Umgebung abfragen, setzen oder kreieren
`patent.frame`	Frame der rufenden Funktion
`sys.frame`	Systemframe (z.B. .GlobalEnv)
`sys.status`	Liste mit `sys.calls`, `sys.parents`, `sys.frames`
`parse`	String in einen Ausdruck umwandeln
`deparse`	Ausdruck in einen String umwandeln
`quote`	Argument zurückgeben, ohne es zu evaluieren
`substitute`	Substitution von Variablen in einem Ausdruck
...	...