Grundlagen der Datenanalyse mit R
(R 1)
Sommersemester 2013

und

Statistik und Simulation mit R
(R 2)
Wintersemester 2013/2014

und

Lineare Modelle mit R: Regression und Varianzanalyse
(R 3)
Sommersemester 2014

sowie

Ausgewählte statistische Verfahren mit R
(R 4)
Wintersemester 2014/2015

Dr. Gerrit Eichner
Mathematisches Institut der
Justus-Liebig-Universität Gießen
Arndtstr. 2, D-35392 Gießen, Tel.: 0641/99-32104
E-Mail: gerrit.eichner@math.uni-giessen.de

Inhaltsverzeichnis

1 Einführung **1**
1.1 Was ist **R**, woher kommt es und wo gibt es Informationen darüber? 1
1.2 Vor dem Start eines Projektes mit **R** (unter Microsoft-Windows) 2
1.3 Aufruf von **R** unter Microsoft-Windows 3
1.4 Eingabe und Ausführung von **R**-Befehlen 3
1.5 Benutzerdefinierte Objekte: Zulässige Namen, speichern und löschen 6
1.5.1 Die "command history" 7
1.5.2 Demos 7
1.5.3 Die Online-Hilfe 7
1.5.4 Beenden von **R** 9
1.6 **R**s "graphical user interface" unter Microsoft-Windows 9
1.7 Installation von Zusatzpaketen 12
1.8 Einführungsliteratur 12

2 Datenobjekte: Strukturen, Attribute, elementare Operationen **14**
2.1 Konzeptionelle Grundlagen 14
2.1.1 Atomare Strukturen/Vektoren 14
2.1.2 Rekursive Strukturen/Vektoren 15
2.1.3 Weitere Objekttypen und Attribute 15
2.1.4 Das Attribut „Klasse" ("`class`") 15
2.2 `numeric`-Vektoren: Erzeugung und elementare Operationen 15
2.2.1 Beispiele regelmäßiger Zahlenfolgen: `seq()` und `rep()` 16
2.2.2 Elementare Vektoroperationen 18
2.3 Arithmetik und Funktionen für `numeric`-Vektoren 19
2.3.1 Elementweise Vektoroperationen: Rechnen, runden, formatieren 20
2.3.2 Zusammenfassende und sequenzielle Vektoroperationen: Summen, Produkte, Extrema 21
2.3.3 "Summary statistics" (`summary()` etc.) 22
2.3.4 Mathematische Funktionen 23
2.4 `logical`-Vektoren und logische Operatoren 24
2.4.1 Elementweise logische Operationen 24
2.4.2 Zusammenfassende logische Operationen 26
2.5 `character`-Vektoren und elementare Operationen 27
2.5.1 Zusammensetzen von Zeichenketten: `paste()` 27
2.5.2 Benennung und „Entnennung" von Vektorelementen: `names()` & `unname()` 28
2.5.3 Weiter Operationen: `strsplit()`, `nchar()`, `substring()`, `abbreviate()` & Co. 28
2.6 Indizierung und Modifikation von Vektorelementen: `[ ]` 30
2.6.1 Indexvektoren 30
2.6.2 Zwei spezielle Indizierungsfunktionen: `head()` und `tail()` 31
2.6.3 Indizierte Zuweisungen 31
2.7 Faktoren und geordnete Faktoren: Definition und Verwendung 32
2.7.1 Erzeugung von Faktoren (`factor()`, `gl()`) und Levelabfrage (`levels()`) . 33
2.7.2 Änderung der Levelsortierung (`relevel()` & (`reorder()`), Zusammenfassung von Levels (`leve`
2.7.3 Erzeugung von geordneten Faktoren: `ordered()`, `gl()` 34
2.7.4 Änderung der Levelordnung, Zusammenfassung von Levels, Löschen unnötiger Levels bei geord
2.7.5 Klassierung numerischer Werte und Erzeugung geordneter Faktoren: `cut()` 35
2.7.6 Tabellierung von Faktoren und Faktorkombinationen: `table()` 36
2.7.7 Aufteilung gemäß Faktor(en)gruppen, faktor(en)gruppierte Funktionsanwendungen: `split()`, `t`
2.8 Matrizen: Erzeugung, Indizierung, Modifikation und Operationen 38
2.8.1 Grundlegendes zu Arrays 38
2.8.2 Erzeugung von Matrizen: `matrix()` 38

2.8.3 Be-/Entnennung von Spalten und Zeilen: `dimnames()`, `colnames()`, `rownames()`, `unname()` 39
2.8.4 Erweiterung um Spalten oder Zeilen: `cbind()`, `rbind()` 40
2.8.5 Matrixdimensionen und Indizierung von Elementen: `dim()`, `[]`, `head()` & `tail()` 40
2.8.6 Einige spezielle Matrizen: `diag()`, `col()` & `row()`, `lower.tri()` & `upper.tri()` 42
2.8.7 Ein paar wichtige Operationen der Matrixalgebra 43
2.8.8 Effiziente Berechnung von Zeilen- bzw. Spaltensummen oder -mittelwerten (auch gruppiert): `c`
2.8.9 Spaltenweise Standardabweichungen sowie Kovarianz und Korrelation zwischen Spalten: `sd()`
2.8.10 Zeilen-/Spaltenweise Anwendung von Operationen:texttapply(), `sweep()`, `scale()` 44
2.8.11 Erzeugung spezieller Matrizen mit Hilfe von `outer()` 45
2.9 Listen: Konstruktion, Indizierung und Verwendung 46
2.9.1 Erzeugung und Indizierung: `list()`, `[[ ]]`, `head()` bzw. `tail()` 46
2.9.2 Benennung von Listenelementen und ihre Indizierung: `names()` und `$` . . 47
2.9.3 Komponentenweise Anwendung von Operationen: `lapply()`, `sapply()` & Co. 48
2.10 Data Frames: Eine Klasse „zwischen“ Matrizen und Listen 49
2.10.1 Indizierung: `[ ]`, `$`, `head()` und `tail()` sowie `subset()` 49
2.10.2 Erzeugung: `data.frame()`, `expand.grid()` 50
2.10.3 Zeilen-/Spaltennamen: `dimnames()`, `row.names()`, `case.names()`, `col.names()`, `names()` 52
2.10.4 “Summary statistics” und Struktur eines Data Frames: `summary()` und `str()` 52
2.10.5 Komponentenweise Anwendung von Operationen: `lapply()`, `sapply()` . . 53
2.10.6 Anwendung von Operationen auf Faktor(en)gruppierte Zeilen: `by()` . . . 53
2.10.7 „Organisatorisches“ zu Data Frames und dem Objektesuchpfad: `attach()`, `detach()` und `sea`
2.10.8 Nützliche Transformationen: `stack()`, `reshape()`, `merge()`, das Paket `reshape` 57
2.11 Abfrage und Konversion der Objektklasse, Abfrage von `NA`, `NaN`, `Inf`, `NULL` 57

3 Import und Export von Daten bzw. ihre Ausgabe am Bildschirm 60
3.1 Import aus einer Datei: `scan()`, `read.table()` & Co. 60
3.1.1 Die Funktion `scan()` 60
3.1.2 Die Beispieldaten “SMSA” 63
3.1.3 Die Funktion `read.table()` und ihre Verwandten 64
3.2 Bildschirmausgabe und ihre Formatierung: `print()`, `cat()` & Helferinnen 67
3.3 Export in eine Datei: `sink()`, `write()`, `write.table()` 68
3.4 Dateiausgabe im TEX-, HTML- oder “Open-Document”-Format 69

4 Elementare explorative Grafiken 70
4.1 Grafikausgabe am Bildschirm und in Dateien 70
4.2 Explorative Grafiken für univariate Daten 70
4.2.1 Diskrete Häufigkeitsverteilungen: Balken-, Flächen-, Kreisdiagramme, Dot Charts 71
4.2.2 Metrische Verteilungen: Histogramme, Kern-Dichteschätzer, “stem-and-leaf”-Diagramme, Boxp
4.2.3 Zur Theorie und Interpretation von Boxplots und Q-Q-Plots 78
4.3 Explorative Grafiken für multivariate Daten 80
4.3.1 Bivariate diskrete Häufigkeitsverteilungen: Mosaikplots 80
4.3.2 Multivariate metrische Verteilungen: Streudiagramme 81
4.3.3 Trivariat metrische Daten: Bedingte Streudiagramme (“co-plots”) 84
4.3.4 Weitere Möglichkeiten und Hilfsmittel: `stars()`, `symbols()` 86

5 Wahrscheinlichkeitsverteilungen und Pseudo-Zufallszahlen 87
5.1 Die „eingebauten“ Verteilungen 87
5.2 Bemerkungen zu Pseudo-Zufallszahlen in **R** 89

6 Programmieren in R **90**
6.1 Definition neuer Funktionen: Ein Beispiel . . . 90
6.2 Syntax der Funktionsdefinition . . . 91
6.3 Verfügbarkeit einer Funktion und ihrer lokalen Objekte . . . 91
6.4 Rückgabewert einer Funktion . . . 92
6.5 Spezifizierung von Funktionsargumenten . . . 92
6.5.1 Argumente mit default-Werten . . . 93
6.5.2 Variable Argumentezahl: Das „Dreipunkteargument“ . . . 93
6.5.3 Zuordung von Aktual- zu Formalparametern beim Funktionsaufruf . . . 94
6.5.4 Nützliches für den Zugriff auf Argumentelisten und Argumente sowie auf den Quellcode existie
6.6 Kontrollstrukturen: Bedingte Anweisungen, Schleifen, Wiederholungen . . . 97

7 Weiteres zur elementaren Grafik **100**
7.1 Grafikausgabe . . . 100
7.2 Elementare Zeichenfunktionen: `plot()`, `points()`, `lines()` & Co. . . . 100
7.3 Die Layoutfunktion `par()` und Grafikparameter für `plot()`, `par()` et al. . . . 102
7.4 Achsen, Überschriften, Untertitel und Legenden . . . 104
7.5 Einige (auch mathematisch) nützliche Plotfunktionen . . . 106
7.5.1 Stetige Funktionen: `curve()` . . . 106
7.5.2 Geschlossener Polygonzug: `polygon()` . . . 106
7.5.3 Beliebige Treppenfunktionen: `plot()` in Verbindung mit `stepfun()` . . . 107
7.5.4 Die empirische Verteilungsfunktion: `plot()` in Verbindung mit `ecdf()` . . . 108
7.5.5 „Fehlerbalken“: `errbar()` im Package `Hmisc` . . . 108
7.5.6 Mehrere Polygonzüge „auf einmal“: `matplot()` . . . 108
7.6 Interaktion mit Plots . . . 109

8 Zur para- und nicht-parametrischen Inferenzstatistik in Ein- und Zweistichprobenproblemen
8.1 Auffrischung des Konzepts statistischer Tests . . . 111
8.1.1 Motivation anhand eines Beispiels . . . 111
8.1.2 Null- & Alternativhypothese, Fehler 1. & 2. Art . . . 111
8.1.3 Konstruktion eines Hypothesentests im Normalverteilungsmodell . . . 113
8.1.4 Der p-Wert . . . 115
8.2 Konfidenzintervalle für die Parameter der Normalverteilung . . . 117
8.2.1 Der Erwartungswert μ . . . 117
8.2.2 Die Varianz σ^2 . . . 119
8.2.3 Zur Fallzahlschätzung für Konfidenzintervalle mit vorgegebener „Präzision“ 119
8.3 Eine Hilfsfunktion für die explorative Datenanalyse . . . 120
8.4 Ein Einstichproben-Lokationsproblem . . . 122
8.4.1 Der Einstichproben-t-Test . . . 122
8.4.2 Wilcoxons Vorzeichen-Rangsummentest . . . 124
8.4.3 Wilcoxons Vorzeichentest . . . 127
8.5 Zweistichproben-Lokations- und Skalenprobleme . . . 127
8.5.1 Der Zweistichproben-F-Test für der Vergleich zweier Varianzen . . . 127
8.5.2 Der Zweistichproben-t-Test bei unbekannten, aber gleichen Varianzen . . . 129
8.5.3 Die Welch-Modifikation des Zweistichproben-t-Tests . . . 130
8.5.4 Wilcoxons Rangsummentest (Mann-Whitney U-Test) . . . 131
8.6 Das Zweistichproben-Lokationsproblem für verbundene Stichproben . . . 134
8.6.1 Die Zweistichproben-t-Tests bei verbundenen Stichproben . . . 135
8.6.2 Wilcoxons Vorzeichen-Rangsummentest für verbundene Stichproben . . . 137
8.7 Tests auf Unabhängigkeit . . . 139
8.7.1 Der Pearsonsche Korrelationskoeffizient . . . 141
8.7.2 Der Spearmansche Rangkorrelationskoeffizient . . . 142

8.7.3 Der Kendallsche Rangkorrelationskoeffizient . . . 143
8.8 Die einfache lineare Regression . . . 147
8.9 Die Formelversionen der Funktionen für die Zweistichprobentests . . . 149
8.10 Zu Tests für Quotienten von Erwartungswerten der Normalverteilung . . . 151
8.11 Zu Verfahren zur p-Wert-Adjustierung bei multiplen Tests . . . 151
8.12 Testgüte und Fallzahlschätzung für Lokationsprobleme . . . 152
8.12.1 Der zweiseitige Einstichproben-Gaußtest . . . 152
8.12.1.1 Herleitung der Gütefunktion . . . 152
8.12.1.2 Interpretation und Veranschaulichung der Gütefunktion . . . 153
8.12.1.3 Verwendungen für die Gütefunktion . . . 155
8.12.1.4 Das Problem der unbekannten Varianz . . . 156
8.12.2 Der zweiseitige Einstichproben-t-Test . . . 157
8.12.2.1 Herleitung der Gütefunktion . . . 157
8.12.2.2 Verwendung der Gütefunktion . . . 158
8.12.3 Der einseitige Einstichproben-t-Test . . . 160
8.12.3.1 Gütefunktion: Herleitung, Eigenschaften und Veranschaulichung 160
8.12.3.2 Verwendung der Gütefunktion . . . 161
8.12.4 Die Zweistichproben-t-Tests . . . 162
8.12.4.1 Zwei verbundene Stichproben . . . 162
8.12.4.2 Zwei unverbundene Stichproben . . . 164

9 Zur Inferenzstatistik und Parameterschätzung für Nominaldaten 166
9.1 Bernoulli-Experimente mit `sample()` . . . 166
9.2 Einstichprobenprobleme im Binomialmodell . . . 167
9.2.1 Der exakte Test für die Auftrittswahrscheinlichkeit p: `binom.test()` . . . 167
9.2.2 Der approximative Test für p: `prop.test()` . . . 168
9.2.3 Konfidenzintervalle für p . . . 169
9.2.4 Hinweis zur Fallzahlschätzung für Konfidenzintervalle für p . . . 171
9.3 Mehrstichprobentests im Binomialmodell . . . 172
9.3.1 Zur Theorie der approximativen k-Stichproben-Binomialtests (Pearsons X^2-Tests) 172
9.3.2 Zur Implementation der k-Stichproben-Binomialtests: `prop.test()` . . . 174
9.3.2.1 Der Fall $k = 2$ Stichproben . . . 174
9.3.2.2 Der Fall $k \geq 3$ Stichproben . . . 175
9.4 Testgüte und Fallzahlschätzung im Binomialmodell . . . 176
9.4.1 Einseitiger und zweiseitiger Einstichprobentest . . . 176
9.4.2 Einseitiger und zweiseitiger Zweistichprobentest: `power.prop.test()` . . . 177
9.5 Tests im Multinomialmodell . . . 179
9.5.1 Multinomial-Experimente mit `sample()` . . . 179
9.5.2 Der approximative χ^2-Test im Multinomialmodell: `chisq.test()` . . . 180
9.6 Kontingenztafeln . . . 181
9.6.1 χ^2-Test auf Unabhängigkeit zweier Faktoren und auf Homogenität . . . 181
9.6.1.1 Zum Fall der Unabhängigkeit . . . 182
9.6.1.2 Zum Fall der Homogenität . . . 183
9.6.1.3 Der approximative χ^2-Test auf Unabhängigkeit und der approximative χ^2-Test auf H
9.6.2 Fishers Exakter Test auf Unabhängigkeit zweier Faktoren . . . 187
9.6.2.1 Die Implementation durch `fisher.test()` . . . 188
9.6.2.2 Der Spezialfall der (2×2)-Tafel: Die Odds Ratio . . . 189
9.6.3 Kontingenztafeln für $k \geq 2$ Faktoren und ihre Darstellung: `xtabs()` & `ftable()` 192
9.6.3.1 Der Fall bereits registrierter absoluter Häufigkeiten . . . 192
9.6.3.2 Der Fall explizit aufgeführter Levelkombinationen . . . 195

10 Einführung in die lineare Regression **197**
10.1 Einige Resultate aus der Theorie der normalen linearen Modelle 197
10.2 Die einfache lineare Regression: Modellanpassung & -zusammenfassung 200
10.3 Die multiple lineare Regression: Anpassung & -zusammenfassung 202
10.3.1 Komponenten und Diagnoseplots eines linearen Modells 204
10.4 Zur Syntax von Modellformeln 205
10.5 Zur Interaktion stetiger Covariablen 207
10.6 Modelldiagnose I: Residualanalyse und Transformationen des Modells 210
10.6.1 Grafische Residualanalyse 210
10.6.2 Varianz stabilisierende Transformationen 212
10.6.3 Linearisierende Transformationen 213
10.6.4 Symmetrisierung des Designs, spezielle linearisierbare Regressionsfunktionen214
10.7 Modifizierung eines linearen Regressionsmodells 215
10.7.1 Die Funktion `update()` 216
10.7.2 Das Entfernen eines Terms: `drop1()` 216
10.7.3 Das Hinzufügen eines Terms: `add1()` 219
10.7.4 Beschränkung auf eine Teilmenge der Daten: das `lm`-Argument `subset` . . 220
10.7.5 Exkurs: Akaikes Informationskriterium AIC 221
10.7.5.1 Die Diskrepanz 221
10.7.5.2 Die Kullback-Leibler-Diskrepanz und AIC 223
10.7.5.3 AIC im Modell der linearen Regression 224
10.8 Modelldiagnose II: Ausreißer, Extrempunkte, einflussreiche Punkte, Residualanalyse226
10.8.1 Grafische Identifizierung 226
10.8.2 Inferenzstatistische Residualanalyse 227
10.8.3 Quantitative Identifizierung einflussreicher Punkte, Quantifizierung ihres Einflusses229
10.8.3.1 Einfluss eines Punktes auf $\hat{\mathbf{Y}} = (\hat{Y}_1, \ldots, \hat{Y}_n)'$ 229
10.8.3.2 Einfluss eines Punktes auf $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \ldots, \hat{\beta}_{p-1})'$ 230
10.8.3.3 Einfluss eines Punktes auf $\hat{\sigma}^2$ 231
10.8.4 Zusammenfassung und Umsetzung 231
10.8.4.1 Die Funktion `influence.measures()` und Co. 231
10.8.4.2 Die Funktion `summary.lm()` 233
10.8.5 Zur Unabhängigkeitsannahme der Fehler 234
10.9 Schätz- und Prognosewerte sowie Konfidenz- und Toleranzintervalle 235
10.9.1 Schätzwerte für die Regressionsfunktion und grafische Darstellung 235
10.9.2 Punktweise Konfidenzintervalle für die Regressionsfunktion und ihre Parameter239
10.9.3 Simultane Konfidenzintervalle & -bereiche für die Regressionsfunktion bzw. ihren Parameterve
10.9.4 Ein Konfidenzband für die Regressionsfunktion 245
10.9.5 Punktweise und simultane Toleranzintervalle für zukünftige Response-Werte246
10.9.6 Die Formeln im Spezialfall der einfachen linearen Regression 249
10.9.6.1 Konfidenzintervalle für die Parameter der Regressionsgeraden . . 249
10.9.6.2 Konfidenzintervalle für die Regressionsgerade 250
10.9.6.3 Toleranzintervalle zukünftiger Response-Werte 250
10.10 Polynomiale Regression 253
10.11 Faktorvariablen und Interaktionsterme im linearen Regressionsmodell 258
10.11.1 Ungeordneter Faktor und metrische Variable ohne Interaktion: „Parallele“ Regression259
10.11.2 Ein ungeordneter Faktor und eine metrische Variable mit Interaktionen . 259
10.11.2.1 Das faktorielle (= “crossed”) Modell 260
10.11.2.2 Das hierarchische (= “nested”) Modell 260
10.11.2.3 Modifikationen der beiden Interaktionsmodelle 261
10.11.3 Die Modelle im Überblick 261
10.11.4 Modellparametrisierung ungeordneter Faktoren durch Kontraste 263
10.11.4.1 “Treatment”-Kontraste: Definition und Eigenschaften 264

10.11.4.2 Treatment-Kontraste im Beispiel der parallelen Regression . . . 265
10.11.4.3 Treatment-Kontraste im faktoriellen Modell 267
10.11.4.4 Treatment-Kontraste im hierarchischen Modell 268
10.11.4.5 Helmert-Kontraste: Definition und Eigenschaften 269
10.11.4.6 Helmert-Kontraste im Beispiel der parallelen Regression 270
10.11.4.7 Helmert-Kontraste im faktoriellen Modell 272
10.11.4.8 Helmert-Kontraste im hierarchischen Modell 273
10.11.5 Modellparametrisierung geordneter Faktoren durch Polynom-Kontraste . 273
10.12 F-Tests gewisser linearer Hypothesen: `anova()` 277
10.12.1 Nur stetige Covariablen 278
10.12.1.1 ANOVA für den Vergleich hierarchischer Modelle 279
10.12.1.2 Sequenzielle ANOVA für die Terme eines Modells 281
10.12.2 Stetige und Faktor-Covariablen 282

11 Einführung in die Varianzanalyse 285
11.1 Die einfaktorielle Varianzanalyse ("One-way-ANOVA") 285
11.1.1 "cell means"-Modell, Inferenz und ANOVA-Tabelle 285
11.1.2 Explorative Datenanalyse und konkrete Durchführung der Varianzanalyse mit `aov()` 287
11.1.3 Parametrisierung als Faktoreffekte-Modell und Parameterschätzer mit `model.tables()` 290
11.1.4 Modelldiagnose 290
11.1.5 Bartletts k-Stichprobentest auf Varianzhomogenität und Welchs k-Stichprobentest auf Erwartu
11.1.6 Testgüte und Fallzahlschätzung in der balancierten einfaktoriellen Varianzanalyse 293
11.2 Die zweifaktorielle Varianzanalyse ("Two-way-ANOVA") 296
11.2.1 "cell means"-Modell und Inferenz 296
11.2.1.1 „Interessante" Hypothesen 297
11.2.1.2 Interaktion oder nicht? Grafische Darstellung 299
11.2.1.3 Zur Interpretation der Testergebnisse 302
11.2.2 Der balancierte Versuchsplan 303
11.2.2.1 Faktoreffekte-Parametrisierung und ihre Schätzer 303
11.2.2.2 Die „interessanten" Hypothesen 304
11.2.2.3 Orthogonale Varianzzerlegung und ANOVA-Tabelle 304
11.2.2.4 Aufbereitung der Daten und explorative Analyse 306
11.2.2.5 Test auf Varianzhomogenität 309
11.2.2.6 Durchführung der Varianzanalyse: `aov()` 310
11.2.2.7 Parameterschätzer mit `model.tables()` 311
11.2.2.8 Modelldiagnose 312
11.2.3 Genau eine Beobachtung pro Levelkombination 313
11.2.4 Das einfache, randomisierte Blockexperiment 313
11.2.5 Hinweise zu unbalancierten Versuchsplänen 318
11.3 Einige nicht-parametrische Mehrstichprobentests 319
11.3.1 Lokationsprobleme bei unabhängigen Stichproben 319
11.3.1.1 Der Kruskal-Wallis-Test: `kruskal.test()` 319
11.3.1.2 Der Jonckheere-Terpstra-Test auf Trend (= geordnete Alternativen) 321
11.3.2 Lokationsprobleme bei verbundenen Stichproben 322
11.3.2.1 Der Friedman-Test: `friedman.test()` 322
11.3.2.2 Der Test von Page auf Trend (= geordnete Alternativen) 324
11.3.3 Hinweise zu Skalenproblemen 325
11.3.4 Hinweis zur nicht-parametrischen Analyse in mehrfaktoriellen Versuchsplänen 325
11.4 Multiple Mittelwertvergleiche 326
11.4.1 Multiple Vergleiche mit einer Kontrolle 328
11.4.1.1 Zweiseitige Vergleiche und Konfidenzintervalle 328
11.4.1.2 Einseitige Vergleiche und Konfidenzschranken 330

11.4.1.3 Weitere Möglichkeiten, **C** zu spezifizieren 332
11.4.1.4 Weitere Prozeduren für multiple Vergleich mit einer Kontrolle . 334
11.4.2 Alle paarweisen Vergleiche 336
11.4.2.1 Zweiseitige Vergleiche und Konfidenzintervalle 336
11.4.2.2 Einseitige Vergleiche und Konfidenzschranken 339
11.4.2.3 Weitere Prozeduren für alle paarweisen Vergleich 342
11.4.3 Zu nichtparametrischen multiplen Vergleichen 344

12 Exkurs: Einführung in die Lebensdaueranalyse ("Survival Analysis") 345
12.1 Das Modell der zufälligen Rechtszensierung 345
12.2 Nicht-Parametrik: Der Kaplan-Meier-Schätzer für die Survival-Funktion 347
12.2.1 Motivation, Definition und Berechnung 347
12.2.2 Eigenschaften des Kaplan-Meier-Schätzers; speziell Konfidenzintervalle . . 349
12.2.3 Schätzung der mittleren Survival-Zeit 351
12.2.4 Der Kaplan-Meier-Schätzer in R: `survfit()` 352
12.2.5 Der Kaplan-Meier-Schätzer für gruppierte Survival-Daten 355
12.2.6 Vergleich gruppierter Survival-Daten und die Funktion `survdiff()` 357
12.3 Semi-Parametrik: Das Cox Proportional Hazards Modell 364
12.3.1 Definition des Proportional Hazards Modells 364
12.3.2 Parameterschätzung und Inferenz 365
12.3.3 Das Cox Proportional Hazards Modell in **R**: `coxph()` 367

Literatur 371

1 Einführung

1.1 Was ist R, woher kommt es und wo gibt es Informationen darüber?

R ist eine nicht-kommerzielle „Umgebung“ für die Bearbeitung, grafische Darstellung und (in der Hauptsache statistische) Analyse von Daten. Es besteht aus einer Programmiersprache (die interpretiert und nicht kompiliert wird) und einer Laufzeitumgebung (unter anderem mit Grafik, einem “Debugger”, Zugriff auf gewisse Systemfunktionen und der Möglichkeit, Programmskripte auszuführen). **R** bietet eine flexible Grafikumgebung für die explorative Datenanalyse und eine Vielzahl klassischer sowie moderner statistischer und numerischer Verfahren für die Datenanalyse und Modellierung. Viele sind in die “base distribution” (Grundausstattung) von **R**, kurz “base **R**”, integriert, aber zahlreiche weitere stehen in aktuell (April 2013) über 4430 von NutzerInnen beigesteuerten, sogenannten “(add-on-)packages” zur Verfügung, die bedarfsweise sehr leicht zusätzlich zu installieren sind. Außerdem können benutzereigene, problemspezifische Funktionen einfach und effektiv programmiert werden.

Die offizielle Homepage des **R**-Projektes ist `http://www.r-project.org`. Sie ist die Quelle aktuellster Informationen sowie zahlreicher weiterführender Links, von denen einige im Folgenden noch genannt werden, wie zum Beispiel zu Manualen, Büchern und den – sehr empfehlenswerten – “Frequently Asked Questions”, kurz FAQs. (Siehe auf jener Homepage in der Rubrik “Documentation” die Punkte “Manuals”, “Books” und “FAQs”.) Die Software **R** selbst wird unter `http://cran.r-project.org` sowohl für verschiedene Unix-„Derivate“, für Microsoft-Windows als auch für Mac OS X bereitgehalten. Außerdem ist dort hinter “Contributed” ein Link zu noch mehr von Nutzern beigesteuerten Manualen und Tutorien. Sehr nützlich sind auch die unter dem Link “Task Views” zu großen Themen zusammengestellten so genannten “CRAN Task Views” (bisher 31 an der Zahl). Sie können einen strukturierten und kommentierten, themenspezifischen ersten Überblick verschaffen über einen Teil der oben bereits erwähnten hohen Anzahl an Zusatzpaketen und liefern ein Werkzeug für die automatische Installation aller zu den jeweiligen Themen gehörenden Pakete.

Weitere evtl. sehr interessante Informationsquellen:

- Es existiert ein Wiki unter `http://rwiki.sciviews.org` mit vielen Hilfeseiten zu **R**-spezifischen Themen und Problemen.
- Auf der Seite “Stack Overflow” (die zum “Stack Exchange”-Netzwerk von “free, community-driven Q & A sites” gehört und eine solche für “professional and enthusiast programmers” ist) sind unter dem “tag” R derzeit mehr als 27500 (!) **R**-spezifische Fragen und Antworten gesammelt, zu denen man via `http://stackoverflow.com/questions/tagged/r` direkt gelangt. Sie sind dort noch feiner kategorisierbar.
- Bei `http://gallery.r-enthusiasts.com/` befindet sich die “R Graph Gallery” zahlreicher, mit **R** angefertigter, exzellenter Grafiken (samt ihres **R**-Codes), die nach verschiedenen Kriterien durchsuch- und sortierbar sind.
- `http://journal.r-project.org` ist die Web-Site der offiziellen, referierten Zeitschrift des **R**-Projektes für “statistical computing”.
- Die hoch aktive E-Mail-Liste **R-help** ist ein hervorragendes Medium, um Diskussionen über und Lösungen für Probleme mitzubekommen bzw. selbst Fragen zu stellen (auch wenn der Umgangston gelegentlich etwas rauh ist). Zugang zu dieser Liste ist über den Link “Mailings Lists” auf der o. g. Homepage des **R**-Projektes möglich, oder aber direkt via `http://stat.ethz.ch/mailman/listinfo/r-help`. Wer sie nicht abonnieren, aber trotzdem (gelegentlich) nutzen will, kann z. B. über Gmane auf sie wie auf eine Newsgroup (über verschiedene Arten von Schnittstellen) zugreifen und dort auch Fragen aufgeben (“posten”); der direkte Link ist `http://dir.gmane.org/gmane.comp.lang.r.general`, wo man dann eine Zugriffsart auswählen kann.

Etwas zur Geschichte: Mitte bis Ende der 1970er Jahre wurde in den AT&T Bell Laboratorien (heute Lucent Technologies, Inc.) die „Statistik-Sprache“ **S** entwickelt, um eine interaktive Umgebung für die Datenanalyse zu schaffen. 1991 erschien eine Implementation von **S**, für die heute die Bezeichnung “S engine 3” (kurz S3) in Gebrauch ist. 1998 wurde eine völlig neu konzipierte “S engine 4” (kurz S4) veröffentlicht. Für ausführlichere historische Informationen siehe `http://en.wikipedia.org/wiki/S_programming_language`.

Ab 1988 ist unter dem Namen S-PLUS eine kommerzielle (und binäre sowie nicht gerade billige) Version von **S** mit massenhaft zusätzlichen Funktionen entwickelt bzw. implementiert worden. Seit 2007 existiert S-PLUS in der Version 8 (basierend auf S4) mit mehreren 1000 implementierten Statistik- und anderen Funktionen (siehe hierzu `http://en.wikipedia.org/wiki/S-PLUS`).

R ist ebenfalls eine Implementation der Sprache **S** (quasi ein „Dialekt“) und entstand ab 1993. Sie ist kostenlose, “open source” Software mit einem GNU-ähnlichen Urheberrecht und ist offizieller Bestandteil des GNU-Projektes „GNU-S“. **R** ähnelt „äußerlich“ sehr stark **S** und „innerlich“ (= semantisch) der Sprache “Scheme”. Faktisch gibt es derzeit also drei Implementationen von **S**: Die alte “S engine 3”, die neue “S engine 4” und **R**.

Im Jahr 2000 wurde die **R**-Version 1.0.0 und im Jahr 2004 Version 2.0.0 veröffentlicht; inzwischen ist – nach der Version 2.15.3 – ganz aktuell (3. April 2013) Version 3.0.0 erschienen. Etwa jedes Frühjahr gibt es (bzw. gab es zumindest bisher) ein “upgrade” von x.y.z auf x.y+1.z und während des folgenden Jahres ein bis drei kleinere, jeweils von x.y.z auf x.y.z+1.

R-Code wird prinzipiell über eine Kommandozeilenschnittstelle eingegeben und ausgeführt. Allerdings gibt es inzwischen mehrere grafische Benutzerschnittstellen (“graphical user interfaces” = GUIs), Editoren und ganze integrierte Entwicklungsumgebungen (“integrated development environments” = IDEs), die **R** unterstützen (siehe auch am Ende von Abschnitt 1.6). Unter `http://en.wikipedia.org/wiki/R_programming_language` sind umfangreiche weitere Daten und Fakten verfügbar.

1.2 Vor dem Start eines Projektes mit R (unter Microsoft-Windows)

Empfehlung: Vor dem Beginn der eigentlichen Arbeit an einem konkreten Projekt sollten Sie sich dafür ein eigenes (Arbeits-)Verzeichnis anlegen und dafür sorgen, dass alle mit dem Projekt zusammenhängenden Dateien darin gespeichert sind bzw. werden, auch die von **R** automatisch generierten. (Dies hat nichts mit **R** direkt zu tun, dient lediglich der eigenen Übersicht, der Vereinfachung der Arbeit und wird „außerhalb“ von **R** gemacht.) Damit dies geschieht, können Sie sich wie folgt eine Verknüpfung zu **R** in jenes Arbeitsverzeichnis legen und diese geeignet konfigurieren:

Nach der (erfolgreichen) Installation von **R** befindet sich üblicherweise ein **R**-Icon (genauer eine Verknüpfung zu **R**) auf dem Windows-Desktop, für das sich beim Anklicken mit der rechten Maustaste ein Menü öffnet. Darin wird (u. a.) „Verknüpfung erstellen“ angeboten, was Sie auswählen. Eine evtl. auftauchende Frage nach dem Erstellen einer solchen auf dem Desktop bejahen Sie, um genau das zu erreichen. Das sodann auf dem Desktop neu erstellte, zweite **R**-Icon verschieben Sie in das zuvor angelegte Arbeitsverzeichnis. Hier wird dieses Icon mit der rechten Maustaste angeklickt, um in dem erscheinenden Menü unter dem Reiter „Verknüpfung“ das Feld „Ausführen in“ modifizieren zu können, denn dorthinein muss der *vollständige* Pfad zum Arbeitsverzeichnis kopiert werden, den Sie sich (wohl am einfachsten) aus der Adressleiste des Windows-Explorers – durch geschicktes Anklicken derselbigen – holen.

Wie **R** veranlasst wird, in ein gewünschtes Arbeitsverzeichnis zu wechseln, *ohne* dass darin bereits eine Verknüpfung angelegt worden ist, wird auf Seite 11 in Abschnitt 1.6 beschrieben.

1.3 Aufruf von R unter Microsoft-Windows

Starten Sie **R** mit Hilfe des jeweiligen **R**-Icons in Ihrem Arbeitverzeichnis (oder finden Sie **R** in Windows' Start-Menü und wechseln Sie wie in Abschnitt 1.6 beschrieben in das gewünschte Arbeitsverzeichnis). Das – Windows-spezifische! – **R**-GUI (= "graphical user interface") öffnet sich in einem eigenen Fenster, worin in einem Kommandozeilen-(Teil-)Fenster namens "**R** Console" eine Begrüßungsmeldung ausgegeben wird, die *sehr* nützliche Hinweise darüber enthält, wie man an Informationen über und Hilfe für **R** kommt und wie man es zitiert. Darunter wird der **R**-Prompt „>" dargestellt, der anzeigt, dass **R** ordnungsgemäß gestartet ist und nun **R**-Befehle eingegeben werden können (siehe Abb. 1).

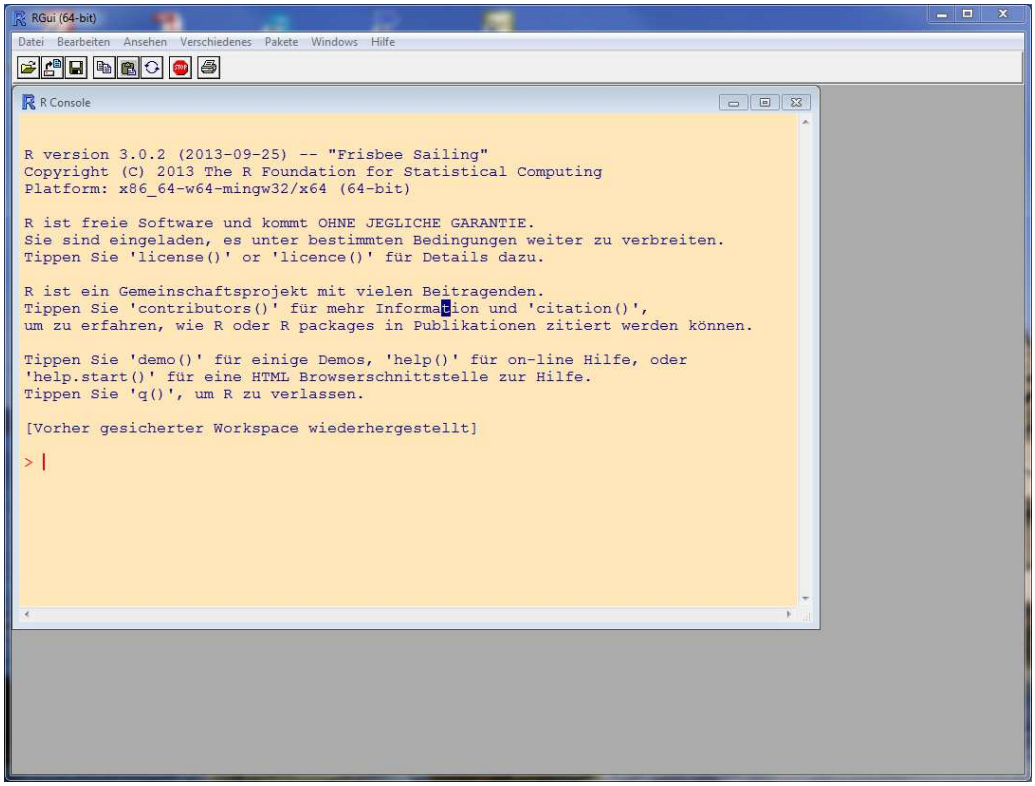


Abbildung 1: Das **R**-GUI (in seiner Voreinstellung) unter Windows mit der **R**-Console und ihrer Begrüßungsmeldung wie es im Wesentlichen auch für die aktuellste Version von **R** aussieht.

1.4 Eingabe und Ausführung von R-Befehlen

S und damit **R** sind funktionale, objektorientierte Sprachen, in der alles sogenannte *Objekte* sind. Dabei werden Groß- und Kleinschreibung beachtet, d. h., `A` und `a` bezeichnen zwei *verschiedene* Objekte. Jeder Befehl (ist selbst ein Objekt und) besteht – vereinfacht dargestellt – entweder aus einem Ausdruck ("expression") oder einer Zuweisungsanweisung ("assignment"), die jeweils sofort nach deren Eingabe am **R**-Prompt und ihrem Abschluss durch die Return-Taste ausgeführt werden. (Es handelt sich bei **R** um einen Interpreter, d. h., die Eingaben müssen nicht erst kompiliert, sprich in Maschinensprache übersetzt werden; dies geschieht automatisch nach dem Tippen der Return-Taste, falls sie syntaktisch korrekt und vollständig waren.)

Ausdrücke: Ist ein Befehl ein Ausdruck, so wird dieser ausgewertet, das Resultat am Bildschirm ausgedruckt und vergessen. Beispielsweise liefert die Eingabe von `17 + 2` (gefolgt vom Druck der Return-Taste) folgendes:

```
> 17 + 2
[1] 19
```

(Hierbei deutet `[1]` vor der Ausgabe an, dass die Antwort von **R** mit dem ersten Element eines – hier nur einelementigen – Vektors beginnt. Dazu später mehr.)

Zuweisungsanweisungen: Eine Zuweisungsanweisung, gekennzeichnet durch den Zuweisungsoperator `<-`, der aus den zwei Zeichen `<` und `-` ohne Leerzeichen dazwischen (!) besteht, wertet den auf der rechten Seite von `<-` stehenden Ausdruck aus und weist den Resultatwert dem Objekt links von `<-` zu. (Das Resultat wird nicht automatisch ausgegeben.) Zum Beispiel:

```
> x <- 119 + 2
```

Die Eingabe des Objektnamens, der selbst ein Ausdruck ist, veranlasst die Auswertung desselben und liefert als Antwort den „Wert“ des Objektes (hier eben eine Zahl):

```
> x
[1] 121
```

Will man also eine Zuweisung durchführen und ihr Ergebnis gleich prüfen, kann das wie eben gezeigt geschehen. Dieses zweischrittige Vorgehen (erst Zuweisung und dann Auswertung des erzeugten Objektes) kann abgekürzt werden, indem die Zuweisungsanweisung in runde Klammern `( )` gepackt wird. Sie erzwingen, dass nach der Ausführung des „Inneren“ der Klammern das Ergebnis dieser Ausführung, also das entstandene Objekt ausgewertet wird:

```
> (x <- 170 + 2)
[1] 172
```

Mehrere Befehle, Kommentare: Mehrere Befehle können zeilenweise, also durch einen Zeilenumbruch mittels Return-Taste getrennt, eingegeben werden oder gemeinsam in einer Zeile durch einen Strichpunkt (;) getrennt. Befindet sich irgendwo in der Zeile das Zeichen `#` (das Doppelkreuz), so wird jeglicher Text rechts davon bis zum Ende dieser Zeile als Kommentar aufgefasst und ignoriert. Im folgenden Beispiel stehen zwei Zuweisungsanweisungen (worin der arithmetische Divisionsoperator `/` und die **R**-Funktion `sqrt()` zur Berechnung der Quadratwurzel verwendet werden) und ein Kommentar in einer Zeile:

```
> (kehrwert <- 1/x);    (wurzel <- sqrt( x))   # Ignorierter Kommentar.
[1] 0.005813953
[1] 13.11488
```

Die Auswertung (von syntaktisch korrekten und vollständigen Ausdrücken) beginnt immer erst nach einer Eingabe von Return und verläuft dann stets sequenziell. Hier wurde also zunächst `kehrwert` erzeugt, dann ausgewertet und schließlich ausgegeben, sodann wurde `wurzel` erzeugt sowie ebenfalls ausgewertet und ausgegeben.

Befehlsfortsetzungsprompt: Ist ein Befehl nach Eingabe von Return oder am Ende der Zeile syntaktisch noch nicht vollständig, so liefert **R** einen „Befehlsfortsetzungsprompt“, nämlich das Zeichen `+`, und erwartet weitere Eingaben in der nächsten Zeile. Dies geschieht so lange bis der Befehl syntaktisch korrekt abgeschlossen ist:

```
> sqrt( pi * x^2          # Die schliessende Klammer fehlt! Das "+" kommt von R.
+ )                       # Hier wird sie "nachgeliefert" und ...
[1] 304.8621
```

...der nun syntaktisch korrekte Ausdruck ausgewertet.

Objektnamenvervollständigung: In vielen Systemen ist eine nützliche (halb-)automatische Objektnamenvervollständigung am **R**-Prompt möglich, die mit der Tabulator-Taste erzielt wird und sowohl eingebaute als auch benutzereigene Objekte einbezieht. Beispiel: In unserer laufenden Sitzung nach der Erzeugung des Objektes `kehrwert` (in der Mitte von Seite 4) liefert das Eintippen von

```
> keh
```

gefolgt vom Druck der Tabulator-Taste die Vervollständigung der Eingabe zu `kehrwert`, da `keh` den gesamten Objektnamen schon eindeutig bestimmt. Bei Mehrdeutigkeiten passiert beim einmaligen Druck der Tabulator-Taste nichts, aber ihr *zweimaliger* Druck liefert eine Auswahl der zur Zeichenkette passenden Objektnamen. Beispiel:

```
> ke
```

gefolgt vom *zweimaligen* Druck der Tabulator-Taste bringt (hier)

```
kehrwert   kernapply  kernel
```

als Erwiderung, an der man erkennt, dass die bisherige Zeichenkette für Eindeutigkeit nicht ausreicht. Jetzt kann durch das nahtlose Eintippen weiterer Buchstaben und die erneute Verwendung der Tabulator-Taste gezielt komplettiert werden.

Ausführung von R-Skripten: Umfangreicheren **R**-Code, wie er sich schnell bei etwas aufwändigeren Auswertungen oder für Simulationen ergibt, wird man häufig in einer Textdatei als **R**-Skript (= **R**-Programm) speichern. Seine vollständige Ausführung lässt sich am **R**-Prompt durch die Funktion `source()` komfortabel erzielen. Ihr Argument muss der Dateiname des Skripts in Anführungszeichen sein bzw. der Dateiname samt Pfad, wenn sich die Datei nicht im aktuellen Arbeitsverzeichnis befindet. Zum Beispiel versucht

```
> source( "Simulation")
```

den Inhalt der Datei „`Simulation`“ aus dem aktuellen Arbeitsverzeichnis (unsichtbar) einzulesen und ihn als (hoffentlich fehlerfreien) **R**-Code auszuführen, während

```
> source( "Analysen2008/Versuch_007")
```

dasselbe mit der Datei „`Versuch_007`“ aus dem Unterverzeichnis „`Analysen2008`“ des aktuellen Arbeitsverzeichnisses macht. (Falls Sie nicht wissen, welches das aktuelle Arbeitsverzeichnis ist und welche Dateien sich darin befinden, bekommen Sie es z. B. mit `getwd()` genannt bzw. mit `dir()` oder `list.files()` seinen Inhalt widergegeben. Allerdings könnte auch hier nach der Eingabe von, z. B., `source( "Analysen2008/Ver` durch den *zweimaligen* Druck der Tabulator-Taste eine (halb-)automatische Dateinamenvervollständigung aktiviert werden!)

Dringende Empfehlung zur Lesbarkeit von R-Code: Zur Bewahrung oder Steigerung der Lesbarkeit von umfangreicherem **R**-Code sollten *unbedingt* Leerzeichen und Zeilenumbrüche geeignet verwendet werden! Hier als Beispiel ein Stück **R**-Code, wie man ihn *auf keinen Fall* produzieren sollte:

```
ifelse(x<z,W$Sub[v],A/(tau*(exp((omega-x)/(2*tau))+exp(-(omega-x)/(2*tau)))^2))
```

Und hier derselbe Code, der durch die Verwendung von Leerzeichen, Zeilenumbrüchen und Kommentaren deutlich besser lesbar geworden ist:

```
ifelse( x < z, W$Sub[ v],
        A / ( tau * (exp( (omega - x) / (2*tau) ) +
                     exp(-(omega - x) / (2*tau) )  )^2
            ) # Ende des Nenners von A / (....)
      ) # Ende von ifelse( ....)
```

Die Suche nach unvermeidlich auftretenden Programmierfehlern wird dadurch sehr erleichtert.

1.5 Benutzerdefinierte Objekte: Zulässige Namen, speichern und löschen

Alle benutzerdefinierten Objekte (Variablen, Datenstrukturen und Funktionen) werden von **R** während der laufenden Sitzung gespeichert. Ihre Gesamtheit wird "workspace" genannt. Eine Auflistung der Namen der aktuell im workspace vorhandenen Objekte erhält man durch den Befehl

```
> objects()
```

Dasselbe erreicht man mit dem kürzeren Aufruf `ls()`. (Zusätzliche Informationen über die Struktur der Objekte liefert `ls.str()`.)

Zulässige Objektnamen bestehen aus Kombinationen von Klein- und Großbuchstaben, Ziffern und „`.`" sowie „`_`", wobei sie mit einem Buchstaben oder „`.`" beginnen müssen. In letzterem Fall darf das zweite Namenszeichen keine Ziffer sein und diese Objekte werden „unsichtbar" gespeichert, was heißt, dass sie beim Aufruf von `objects()` oder `ls()` nicht automatisch angezeigt werden. Memo: Groß-/Kleinschreibung wird beachtet! Beispiele: `p1`, `P1`, `P.1`, `Bloodpool.Info.2008`, `IntensitaetsKurven`, `Skalierte_IntensitaetsKurven`

Es empfiehlt sich zur Verbesserung der Code-Lesbarkeit durchaus, lange und aussagefähige Objektnamen zu verwenden, womit übrigens auch das in der folgenden Warnung beschriebene Problem vermieden werden kann.

Warnung vor Maskierung: Objekte im benutzereigenen workspace haben i. d. R. Priorität über **R**-spezifische Objekte mit demselben Namen! Das bedeutet, dass **R** die ursprüngliche Definition möglicherweise nicht mehr zur Verfügung hat und stattdessen die neue, benutzereigene zu verwenden versucht. Man sagt, die benutzereigenen Objekte „maskieren" die **R**-spezifischen. Dies kann Ursache für (zunächst seltsam erscheinende) Warn- oder Fehlermeldungen oder – schlimmer – augenscheinlich korrektes Verhalten sein, welches aber unerkannt (!) falsche Resultate liefert.

Vermeiden Sie daher Objektnamen wie z. B. `c`, `s`, `t`, `C`, `T`, `F`, `matrix`, `glm`, `lm`, `range`, `tree`, `mean`, `var`, `sin`, `cos`, `log`, `exp` und `names`. Sollten seltsame Fehler(-meldungen) auftreten, kann es hilfreich sein, sich den workspace mit `objects()` oder `ls()` anzusehen, gegebenenfalls einzelne, verdächtig benannt erscheinende Objekte zu löschen und dann einen neuen Versuch zu starten. Ob ein Objektname bereits vergeben ist und ein Zugriffskonflikt oder eine Maskierung drohen würde, können Sie vor seiner ersten Verwendung z. B. dadurch überprüfen, dass Sie ihn einfach am **R**-Prompt eingeben und auszuwerten versuchen lassen.

Gelöscht werden Objekte durch die Funktion `rm()` (wie "remove"). Sie benötigt als Argumente die durch Komma getrennten Namen der zu löschenden Objekte:

```
> rm( a, x, Otto, Werte.neu)
```

löscht die Objekte mit den Namen `a`, `x`, `Otto` und `Werte.neu`, egal ob es Variablen, Datenstrukturen oder Funktionen sind.

Eine Löschung *aller* benutzerdefinierten Objekte auf einen Schlag erzielt man mit dem Befehl

```
> rm( list = objects())     # Radikale Variante: Loescht (fast) alles!
```

der aber natürlich mit Vorsicht anzuwenden ist. Ebenfalls vollständig „vergessen“ werden die in der aktuellen Sitzung zum workspace *neu hinzugekommenen* Objekte und die an im workspace bereits existierenden Objekten durchgeführten Änderungen, wenn man beim Verlassen von **R** die Frage “`Save workspace image? [y/n/c]`” mit `n` beantwortet (siehe auch §1.5.4, Seite 9). Also Vorsicht hierbei!

Eine permanente Speicherung der benutzerdefinierten Objekte des workspaces wird beim Verlassen von **R** durch die Antwort `y` auf die obige Frage erreicht. Sie führt dazu, dass alle Objekte des momentanen workspaces in einer Datei namens `.RData` im aktuellen Arbeitsverzeichnis gespeichert werden. Wie man sie in der nächsten **R**-Session „zurückholt“ und weitere Details werden in §1.5.4 beschrieben.

1.5.1 Die “command history”

R protokolliert die eingegebenen Befehle mit und speichert sie in einer Datei namens `.RHistory` im aktuellen Arbeitsverzeichnis, wenn beim Verlassen von **R** der workspace gespeichert wird. Am **R**-Prompt kann man mit Hilfe der Cursor-Steuertasten (= Pfeiltasten) in dieser “command history” umherwandern, um so frühere Kommandos „zurückzuholen“, sie zu editieren, falls gewünscht, und sie durch Tippen von Return (an jeder beliebigen Stelle im zurückgeholten Kommando) erneut ausführen zu lassen.

1.5.2 Demos

Um sich einen ersten Einblick in die Fähigkeiten von **R** zu verschaffen, steht eine Sammlung von halbautomatisch ablaufenden Beispielen („Demos“) zur Verfügung, die mit Hilfe der Funktion `demo()` gestartet werden können. Der Aufruf von `demo()` ohne Angabe eines Arguments liefert eine Übersicht über die (in der Basisversion) verfügbaren Demos. Die Angabe des Demo-Namens als Argument von `demo()` startet die genannte Demo. Beispiel:

```
> demo( graphics)
```

startet eine Beispielesammlung zur Demonstration von **R**sGrafikfähigkeiten. Beendet wird sie durch Eintippen von `q` (ohne Klammern).

1.5.3 Die Online-Hilfe

R hat eine eingebaute Online-Dokumentation. Sie ist in den meisten Installationen per Voreinstellung ein HTML-basiertes Hilfesystem, das bei einer Anfrage den jeweils voreingestellten Web-Browser startet, falls er noch nicht läuft, und die angeforderte Hilfeseite darin in einem neuen „Tab“ darstellt. Die Hilfe wird objektspezifisch mit `help(....)` aufgerufen, wobei anstelle von „`....`“ der Name eines Objektes, sprich einer Funktion, eines Datensatzes oder etwas anderem steht, worüber man Informationen haben will. Zum Beispiel liefert

```
> help( mean)
```

ausführliche Informationen über die (eingebaute) Funktion `mean()`. Die Kurzform `?mean` liefert dasselbe wie `help( mean)`. (Ohne Argument, also durch `help()`, erhält man übrigens Hilfe zur Funktion `help()` selbst.)

Durchaus hilfreich in diesem Zusammenhang kann die stichwortbasierte Suche mittels der Funktion `help.search()` sein. Sie erlaubt auf verschiedene Arten die Angabe von (auch vagen)

„Textmustern“ oder „Schlüsselwörten“, nach denen in **R**s Hilfeseiten – an gewissen Stellen – gesucht werden soll, falls man den exakten Objektnamen nicht (mehr) oder noch nicht weiß.

Sicher *sehr* hilfreich und absolut empfehlenswert, da oft sehr lehrreich, sind die Beispiele, die eine jede Hilfeseite (so man sie gefunden hat) an ihrem Ende im Abschnitt “Examples” üblicherweise bereithält. Sie sind mit der Funktion `example()` automatisch ausführbar, wenn man ihr als Argument den Namen des interessierenden Objektes übergibt, also z. B. für die Funktion `mean()` durch `example( mean)`.

Allgemein wird die Startseite von **R**s Online-Hilfesystem aufgerufen mit

```
> help.start()
```

Dies sollte den jeweils voreingestellten Web-Browser starten, falls er noch nicht läuft, und – nach wenigen Sekunden – darin die in Abb. 2 zu sehende Seite anzeigen. Wird ein anderer als der voreingestellte Browser bevorzugt, kann seine Verwendung durch

```
> help.start( browser = "....")
```

erzwungen werden (falls er installiert ist), wobei anstelle von `....` der Name des Browser-Programmes stehen muss.

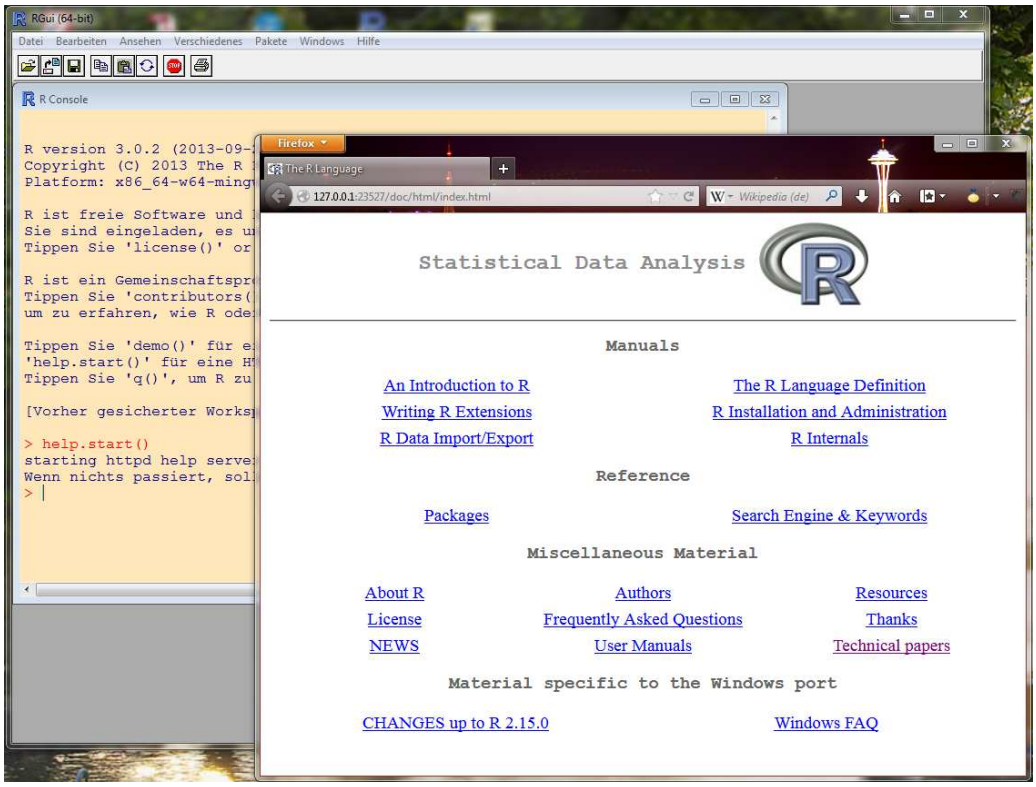


Abbildung 2: Startseite von **R**sBrowser-basierter Online-Hilfe.

Empfehlungen:

1. Hinter dem Link An Introduction to R (in der Startseite der Online-Hilfe in Abb. 2 links unterhalb von `Manuals` zu sehen) steckt eine gute, HTML-basierte, grundlegende Einführung in die Arbeit mit **R** einschließlich einer Beispielsitzung in ihrem Appendix A.

2. Search Engine & Keywords unterhalb von `Reference` führt zu einer nützlichen Suchmaschine für die Online-Hilfeseiten, die die Suche nach “keywords”, Funktionsnamen, Datennamen und Text in den Überschriften jener Hilfeseiten erlaubt. Im Fall einer erfolgreichen

Suche wird eine Liste von Links zu den betreffenden Hilfeseiten gezeigt, auf denen umfangreiche Informationen zu finden sind.

3. Unter Frequently Asked Questions (FAQs) unterhalb von `Miscellaneous Material` sind die Antworten auf einige der typischen Fragen, die den neuen "useR" und die neue "useRin" plagen könnten, zu finden und Lösungen für gewisse Probleme angedeutet.

1.5.4 Beenden von R

Um **R** zu beenden, tippen Sie am Prompt `q()` ein (dabei die leeren Klammern nicht vergessen!):

```
> q()
```

Bevor **R** wirklich beendet wird, werden Sie hier *stets* gefragt, ob die Daten, genauer: die Objekte dieser Sitzung gespeichert werden sollen. Sie können `y`(es), `n`(o) oder `c`(ancel) eingeben, um die Daten (vor dem Verlassen von **R**) permanent speichern zu lassen bzw. um **R** zu verlassen, ohne sie zu speichern, bzw. um die Beendigung von **R** abzubrechen (und sofort zu **R** zurückzukehren). Die permanente Speicherung der Objekte des aktuellen workspaces geschieht in der Datei `.RData` des aktuellen Arbeitsverzeichnisses (das Sie im Fall, dass Sie nicht mehr wissen, „wo Sie sind", mit `getwd()` abfragen können; vgl. Seite 5).

Jener workspace und damit die vormals gespeicherten Objekte können beim nächsten Start von **R** aus dieser Datei `.RData` rekonstruiert und wieder zur Verfügung gestellt werden. Dies geschieht im gewünschten Arbeitsverzeichnis entweder automatisch, indem man **R** durch einen Doppelklick auf das dortige Icon der Datei `.RData` startet (falls `.RData`-Dateien mit **R** verknüpft sind, was bei einer ordnungsgemäßen Windows-Installation automatisch der Fall sein sollte), oder durch durch einen Doppelklick auf die entsprechend angelegte Verknüpfung, oder schließlich „von Hand", indem man eine „irgendwo und irgendwie" gestartete **R**-Session veranlasst (z. B. mit Hilfe des **R**-GUIs wie im folgenden Abschnitt 1.6 beschrieben), das gewünschte Verzeichnis als das aktuelle Arbeitsverzeichnis zu wählen und den vormaligen workspace wieder einzulesen.

1.6 Rs "graphical user interface" unter Microsoft-Windows

Es folgt eine rudimentäre Beschreibung einiger Funktionen des Windows-spezifischen und selbst sehr rudimentären **R**-GUIs.

Das **GUI-Menü** (zu sehen am oberen Rand links in Abb. 1 oder Abb. 3) enthält sechs Themen (von „Datei" bis „Hilfe"), von denen die folgenden drei für die neue "use**R**in" und den neuen "use**R**" wichtig oder interessant sind:

1. „Datei": Offeriert einen einfachen (Skript-)Editor, erlaubt etwas Datei-Management und bietet das Speichern und Wiederherstellen ehemaliger **R**-Sitzungen (bestehend aus ihrem workspace und ihrer command history).

2. „Bearbeiten": Bietet Möglichkeiten, **R**-Code, der im Editor eingetippt worden ist, ausführen zu lassen.

3. „Hilfe": Enthält mehrere Punkte zur Hilfe und zu Hintergrundinformationen.

Etwas detailliertere Erläuterungen folgen:

Rs Skripteditor: Zu finden im R-GUI unter „Datei" → „Neues Skript" oder „Öffne Skript ..." (wie in Abb. 3 angedeutet). Ein zweites (Teil-)Fenster mit dem Titel „**R** Editor" öffnet sich (rechts in Abb. 3). Das GUI-Menü-Thema "Windows" bietet übrigens Möglichkeiten, die

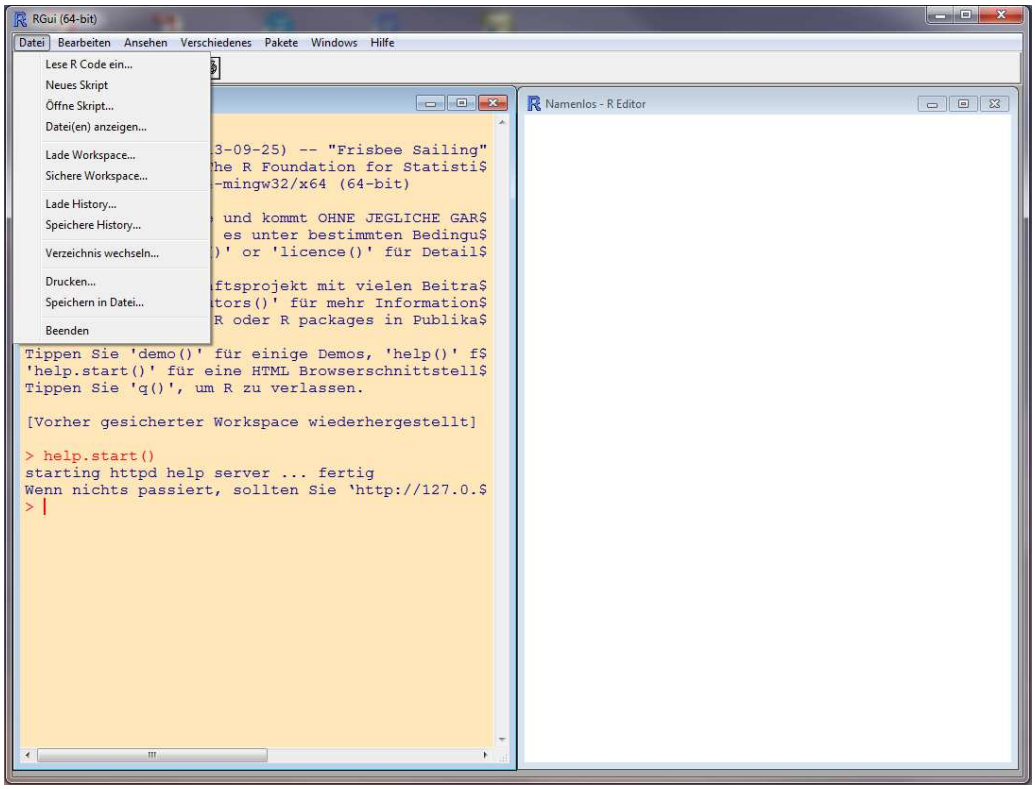


Abbildung 3: Aufruf des **R**-Editors mit einem neuen, also leeren Skript.

Teilfenster innerhalb des GUIs schnell anzuordnen.

Der **R**-Editor kann verwendet werden für die Entwicklung und das Abspeichern von **R**-Code (zusammen mit Kommentaren), der später wieder genutzt oder weiterentwickelt werden soll. Editiert wird darin wie in jedem primitiven Texteditor. Die übliche, einfache Funktionalität steht zur Verfügung: "Copy, cut & paste" mit Hilfe der Tastatur oder der Maus, Ctrl-Z bzw. Strg-Z als Rückgängig-Aktion etc. (Dies und anderes ist auch im GUI-Menü-Thema „Bearbeiten" zu finden.)

Das **Ausführen von R-Code**, der im **R**-Editor steht, kann auf mindestens vier (zum Teil nur geringfügig) verschiedene Methoden erreicht werden (beachte dazu Abb. 4 und siehe auch das GUI-Menü-Thema „Bearbeiten"):

1. Markiere den Code(-Ausschnitt), der ausgeführt werden soll, z. B. mit dem Cursor, und verwende den üblichen "copy & paste"-Mechanismus, um ihn an den Prompt der **R**-Console zu „transportieren". Dort wird er sofort ausgeführt. (Nicht zu empfehlen, da mühselig!)

2. Wenn eine ganze, aber einzelne Zeile des Codes ausgeführt werden soll, platziere den Cursor in eben jener Zeile des Editors und tippe Ctrl-R bzw. Strg-R oder klicke auf das *dann vorhandene* dritte Icon von links unter dem GUI-Menü (siehe die Icons oben in Abb. 4, worin der **R**-Editor das aktive Fenster ist, und vergleiche sie mit denen oben in Abb. 2, in der die **R**-Console das aktive Fenster ist).

3. Soll ein umfangreicherer Teil an Code ausgeführt werden, markiere ihn im Editor wie in Abb. 4 rechts zu sehen und tippe Ctrl-R bzw. Strg-R oder nutze das in Punkt 2 erwähnte Icon.

4. `source( ....)`, wie auf Seite 5 in Abschnitt 1.4 beschrieben, funktioniert natürlich auch hier, wenn der aktuelle Inhalt des **R**-Editors bereits in einer Datei abgespeichert wurde

und deren Dateiname (nötigenfalls inklusive vollständiger Pfadangabe) als Argument an `source()` übergeben wird. Dies führt i. d. R. sogar zu einer schnelleren Code-Ausführung als die obigen Varianten.

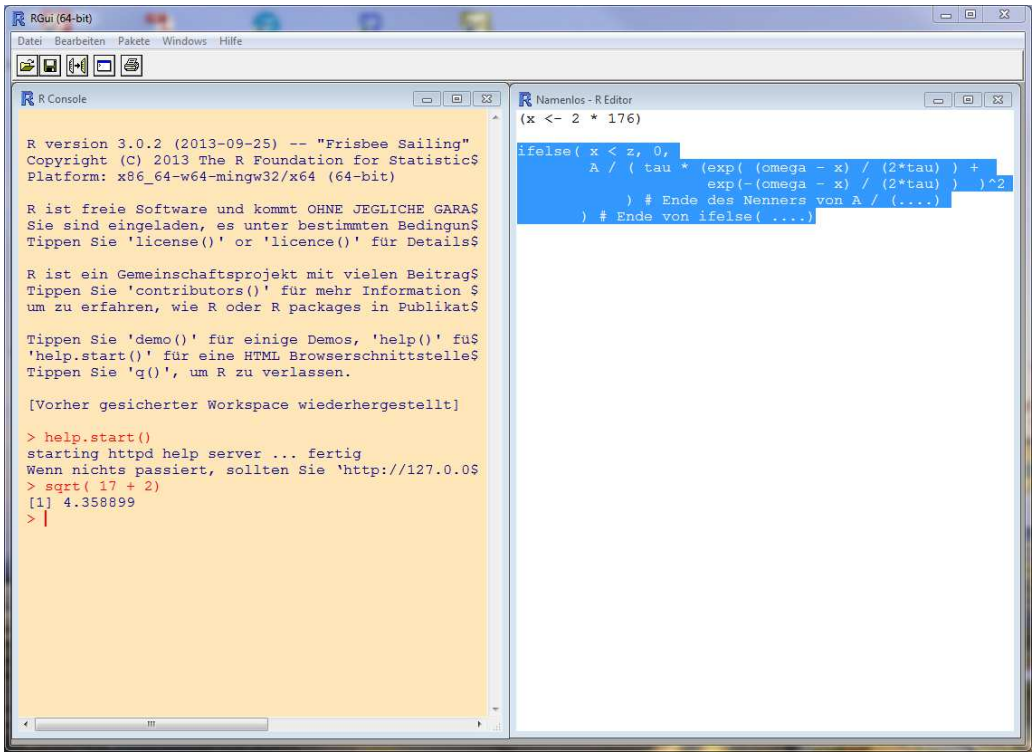


Abbildung 4: Links: direkt am Prompt eingegebener und ausgeführter **R**-Code. Rechts: markierter **R**-Code im **R**-Editor, der ausgeführt werden soll.

Wechsel des Arbeitsverzeichnisses und **Laden eines workspaces:** Im GUI-Menü-Thema „Datei“ sind auch die Punkte zu finden, die für einen Wechsel des Arbeitsverzeichnisses und das Laden eines workspaces nützlich sind, falls **R** nicht durch einen Doppelklick auf die `.RData`-Datei oder auf seiner dortige Verknüpfung im gewünschten Arbeitsverzeichnis gestartet wurde. Es sind dies die Punkte „Verzeichnis wechseln ...“ bzw. „Lade Workspace ...“ (siehe hierfür nochmal Abb. 3).

Rs **Online-Hilfe:** Zusätzlich zu der in §1.5.3 beschriebenen Methode, Online-Hilfe zu bekommen, findet sie sich Browser-basiert auch unter „Hilfe“ → „HTML Hilfe“ (zu sehen in Abb. 2). Beachte die Empfehlungen hierzu auf Seite 8 am Ende von §1.5.3. Darüberhinaus ist unter „Hilfe“ → „Manuale (PDF)“ aber auch eine PDF-Version des Manuals “An Introduction to **R**” zu finden.

Bemerkungen:

- Der **R**-eigene Editor braucht nicht verwendet zu werden; **R**-Code kann selbstverständlich jederzeit auch direkt am **R**-Prompt eingetippt werden.
- Der Editor kann natürlich auch dazu genutzt werden, **R**-Ausgaben zu speichern, indem man “copy & paste” von der **R**-Console in den **R**-Editor (oder jeden beliebigen anderen Editor) verwendet.
- Es existieren einige, durchweg kostenlose „Alternativen“ zum **R**-GUI, als da wären in alphabetischer Reihenfolge z. B.

- Eclipse + StatET (`http://www.walware.de/goto/statet`), d. h. die IDE „Eclipse“ (direkt erhältlich über die Web-Site `http://www.eclipse.org`) mit dem Plug-in „StatET“ (zu finden über `http://www.walware.de/goto/statet`). Eine gute Einführung samt hilfreichen Installationshinweisen liefert Longhow Lams “Guide to Eclipse and the R plug-in StatET”, der zu finden ist unter `http://www.splusbook.com/RIntro/R_Eclipse_StatET.pdf`.
- Emacs Speaks Statistics (kurz ESS, `http://ess.r-project.org`);
- JGR im Paket „JGR“ (sprich “jaguar”, `http://www.rforge.net/JGR`);
- R Commander im Paket „Rcmdr“ (mit näheren Informationen auf seiner Projekt-Homepage `http://socserv.socsci.mcmaster.ca/jfox/Misc/Rcmdr`);
- RStudio (`http://www.rstudio.org`);
- Tinn-R (`http://www.sciviews.org/Tinn-R`) mit einer guten Einführung in dem kostenlosen e-book, das man sich unter `www.rmetrics.org/ebooks-tinnr` herunterladen kann;

und andere. (Auf eine Informationquelle dafür ist am Ende von Abschnitt 1.1 hingewiesen worden.) Ihre Installationen erfordern allerdings einen gewissen Zusatzaufwand, den wir uns sparen, da uns bis auf Weiteres das **R**-GUI ausreicht.

1.7 Installation von Zusatzpaketen

Ist ein Zusatzpaket (“add-on package” oder kurz “package”) zu installieren, so lässt sich dies, falls man eine funktionierende Internetverbindung hat, i. d. R. völlig problemlos erledigen entweder

- mit Hilfe des **R**-GUIs, indem man im GUI-Menü-Thema „Pakete“ den Punkt „Installiere Paket(e)“ anklickt (evtl. einen – möglichst nahegelegenen – “Mirror“ aussucht) und dann den Namen des gewünschten Paketes auswählt, oder
- am **R**-Prompt mit dem Befehl `install.packages()`, z. B. wie in

```
> install.packages( "fortunes")
```

1.8 Einführungsliteratur

Einführende Literatur zum Umgang, zur Statistik und zum Programmieren mit **R** (in alphabetischer Reihenfolge der Autorennamen und mit Preisen laut Amazon im April 2012, falls dort „derzeit nicht verfügbar“; sonst April 2013):

- Braun, W. J., Murdoch, D. J.: *A First Course in Statistical Programming with R.* Cambridge University Press, 2007. (∼ 30 €, paperback; „derzeit nicht verfügbar“)
- Crawley, M. J.: *The R Book.* 2nd ed., John Wiley & Sons, Inc., 2013. (∼ 72 € für über 1050 (!) Seiten))
- Dalgaard, P.: *Introductory Statistics with R.* 2nd ed., Springer-Verlag, 2008. (∼ 50 €; „derzeit nicht verfügbar“)
- Everitt, B. S., Hothorn, T.: *A Handbook of Statistical Analyses Using R.* 2nd ed., Chapman & Hall/CRC, Boca Raton, 2010. (∼ 48 €; „derzeit nicht verfügbar“)
- Hatzinger, R., Hornik, K., Nagel, H.: *R – Einführung durch angewandte Statistik.* Pearson Studium, München, 2011. (∼ 30 €, Taschenbuch)

- Ligges, U.: *Programmieren mit R.* 3., überarbeitete und erweiterte Auflage, Springer-Verlag, 2008. (∼ 35 €, Taschenbuch)
- Maindonald, J., Braun, J.: *Data Analysis and Graphics Using R. An Example-based Approach.* 3rd ed., Cambridge University Press, 2010. (∼ 66 €; scheint „derzeit nicht verfügbar“ zu sein).
- Venables, W. N., Ripley, B. D.: *S Programming.* Corr. 2nd printing, Springer-Verlag, New York, 2000. (∼ 97 €, paperback, March 2012)
- Verzani, J.: *Using R for Introductory Statistics.* Chapman & Hall/CRC Press, Boca Raton/Florida, 2004. (∼ 46 €)

2 Datenobjekte: Strukturen, Attribute und elementare Operationen

Eine umfassende und präzise Sprachdefinition für **R** enthält das Dokument "The **R** Language Definition", das über die Startseite von **R**s Online-Hilfe zu erreichen ist (vgl. Abb. 2 auf Seite 8). Wir geben hier nur einen kurzen Abriss über die für unsere Zwecke ausreichenden Grundlagen.

Alles in **R** ist ein *Objekt* und jedes Objekt hat eine gewisse Struktur. Alle **R**-Strukturen sind Vektoren (= geordnete, endliche Mengen), die sowohl einen (wohldefinierten) Typ, genannt *Modus* ("*mode*"), als auch eine (wohldefinierte) nicht-negative, endliche Länge haben. Daten, die verarbeitet werden sollen, müssen in Objekten zusammengefasst gespeichert werden. Wir wollen hierbei von *Datenobjekten* sprechen (um sie von anderen Objekten zu unterscheiden, die später noch eingeführt werden). Es folgt, dass auch jedes Datenobjekt einen Modus und eine Länge hat.

2.1 Konzeptionelle Grundlagen

Ein Vektor enthält $n \geq 0$ eindeutig indizierbare *Elemente*. Dabei wird n die Länge des Vektors genannt und die Elemente können sehr allgemeiner Natur sein. (Vektoren der Länge 0 sind möglich und Skalare sind Vektoren der Länge 1; doch dazu später mehr.) Es gibt verschiedene Spezialisierungen für Vektoren, von denen wir die wichtigsten hier in einer Übersicht aufzählen:

2.1.1 Atomare Strukturen/Vektoren

Die einfachste Vektorform umfasst die sogenannten „atomaren Vektoren", die sich dadurch auszeichnen, dass alle Elemente eines solchen Vektors vom selben Modus wie der ganze Vektor sind. Es gibt atomare Vektoren der folgenden Modi:

- `logical`: Mögliche Elemente sind die booleschen Werte TRUE (abgekürzt T) und FALSE (abgekürzt F).

- `numeric`: Hierbei handelt es sich um einen Oberbegriff der zwei Modi `integer` und `double`:

  - `integer`: Die Elemente sind ganze Zahlen wie $0, 1, -3$ usw.
  - `double`: Fließkommazahlen mit der Notation `3.5`, `-6.0`, `8.4e10`, `-5e-7`. Zur Abkürzung kann eine ganze Zahl, die als Fließkommazahl aufgefasst werden soll, unter Auslassung von Nachkomma-Nullen lediglich mit Dezimalpunkt notiert werden, also z. B. `4.` (statt `4.0`). Ähnlich können Dezimalbrüche zwischen 0 und 1 durch Weglassen der „Vorkomma-Null" abgekürzt werden, also `.35` statt `0.35`. Des Weiteren ist `8.4e10` $= 8.4 \cdot 10^{10}$ und `-5e-7` $= -5 \cdot 10^{-7}$.

  Beachte: `double`-Werte sind im Allgemeinen keine reellen Zahlen, da jedes digitale System grundsätzlich nur eine endliche numerische Rechengenauigkeit besitzt! Informationen zu den vom jeweiligen Computer-System abhängigen, numerischen Eigenschaften einer **R**-Implementation sind in der Variablen `.Machine` gespeichert und auf ihrer Online-Hilfeseite (zu erreichen via `?.Machine`) erläutert.

- `complex`: Dies repräsentiert komplexe Zahlen $a + b \cdot i$, wobei a und b Zahlen des Modus' `numeric` sind und zwischen b und dem Symbol `i` für die imaginäre Einheit $i = \sqrt{-1}$ kein Leerzeichen stehen darf. Bsp.: `3 + 7i` oder `-1.5 + 0.8i`.

- `character`: Hiermit werden (nahezu) beliebige Zeichenketten gespeichert. Sie werden durch Paare von `"` oder `'` begrenzt, wie z. B. `"Otto"` und `'auto2002'`.

Der Modus eines Objektes `x` kann mit der Funktion `mode()` durch `mode( x)` abgefragt werden.

Sprechweise: Einen Vektor des Modus' `numeric` nennen wir kurz `numeric`-Vektor. Für die anderen Vektormodi gilt Analoges.

2.1.2 Rekursive Strukturen/Vektoren

Eine besondere Form von Vektoren sind die sogenannten „rekursiven" Vektoren: Ihre Elemente sind Objekte beliebiger Modi. Dieser Vektormodus heißt `list`. Ein `list`-Objekt (kurz: eine Liste) ist demnach ein Vektor, dessen Elemente beliebige Objekte verschiedener Modi sein können (also insbesondere selbst wieder Listen). Listen sind mit die wichtigsten (Daten-)Objekte in **R** und werden uns häufig begegnen.

Eine weitere wichtige rekursive Struktur heißt `function`. Sie erlaubt – wie der Name sagt – die Implementation (neuer) benutzerspezifischer Funktionen, die als **R**-Objekte im workspace gespeichert werden können und so **R** quasi erweitern.

2.1.3 Weitere Objekttypen und Attribute

Außer den grundlegenden Eigenschaften Modus und Länge (den sogenannten „intrinsischen Attributen") eines Objektes gibt es noch weitere „Attribute", die Objekten gewisse Struktureigenschaften verleihen. In **R** stehen neben den schon erwähnten Vektoren und Listen viele weitere Objekttypen zur Verfügung, die durch gewisse Attribute „generiert" werden. Beispiele:

- `array` bzw. `matrix`: Sie dienen der Realisierung mehrfach indizierter Variablen und haben ein Attribut „Dimension" (`dim`) und optional ein Attribut „Dimensionsnamen" (`dimnames`).
- `factor` bzw. `ordered` (`factor`): Dies sind `character`-Vektoren, deren Elemente als Ausprägungen einer nominal- bzw. einer ordinal-skalierten Variablen interpretiert werden. Sie haben ein Attribut `levels`, das alle möglichen Ausprägungen der Variablen aufzählt und im Fall `ordered` gleichzeitig die (Rang-)Ordnung dieser Ausprägungen auf der Ordinalskala beschreibt.

2.1.4 Das Attribut „Klasse" ("`class`")

Ein Attribut der besonderen Art ist die „Klasse" (`class`) eines Objektes. Sie wird für die objektorientierte Programmierung in **R** verwendet. Die Klasse eines Objektes entscheidet häufig darüber, wie gewisse Funktionen mit ihnen „umgehen". Viele **R**-Objekte haben ein `class`-Attribut. Falls ein Objekt kein (explizites) `class`-Attribut hat, so besitzt es stets eine implizite Klasse: Es ist dies `matrix`, `array` oder das Ergebnis von `mode( x)`.

Eine spezielle Klasse, die eine Struktur gewissermaßen *zwischen* `list` und `matrix` implementiert, ist die Klasse `data.frame`. Sie dient der strukturierten Zusammenfassung von, sagen wir, p Vektoren gleicher Länge n, aber verschiedener Modi, in ein $(n \times p)$-matrixförmiges Schema. "Data frames" werden in **R** häufig im Zusammenhang mit dem Anpassen statistischer Modelle verwendet. Dabei werden die Vektoren (= Spalten des Schemas) als Variablen interpretiert und die Zeilen des Schemas als die p-dimensionalen Datenvektoren der n Untersuchungseinheiten.

Auf die oben genannten Datenobjekte und anderen Strukturen (außer `complex`) gehen wir in den folgenden Abschnitten detaillierter ein.

2.2 `numeric`-Vektoren: Erzeugung und elementare Operationen

Anhand einfacher Beispiele wollen wir Methoden zur Erzeugung von und elementare Operationen für `numeric`-Vektoren vorstellen. Durch die Zuweisungsanweisung

```
> hoehe <- c( 160, 140, 155)
```

wird dem Objekt `hoehe` ein Vektor mit den Elementen 160, 140 und 155 zugewiesen, indem die Funktion `c()` (vom englischen "concatenation", d. h. „Verkettung") diese Werte zu einem Vektor zusammenfasst. Da es sich bei den Werten auf der rechten Seite von `<-` nur um `integer`- und damit um `numeric`-Größen handelt, wird `hoehe` (automatisch) zu einem `numeric`-Vektor. Skalare werden als Vektoren mit nur *einem* Element aufgefasst und können ohne die Funktion `c()` zugewiesen werden:

```
> eine.weitere.hoehe <- 175
```

Mittels der Funktion `c()` lassen sich Vektoren einfach aneinanderhängen:

```
> c( hoehe, eine.weitere.hoehe)
[1] 160 140 155 175
```

Eine Zuweisung, auf deren linker Seite ein Objekt steht, welches auch auf der rechten Seite auftaucht, wie in

```
> (hoehe <- c( hoehe, eine.weitere.hoehe))
[1] 160 140 155 175
```

bewirkt, dass zunächst der Ausdruck rechts vom `<-` ausgewertet wird und erst dann das Resultat zugewiesen wird. Der ursprüngliche Wert des Objektes wird durch den neuen überschrieben. (Memo: Zur Wirkungsweise der äußeren Klammern siehe Seite 4.)

Die Funktion `length()` liefert die Anzahl der Elemente eines Vektors, also seine Länge:

```
> length( hoehe)
[1] 4
```

Obiges gilt analog für atomaren Vektoren der anderen Modi.

2.2.1 Beispiele regelmäßiger Zahlenfolgen: `seq()` und `rep()`

Gewisse, häufig benötigte Vektoren, deren Elemente spezielle Zahlenfolgen bilden, lassen sich in **R** recht einfach erzeugen. Hierzu stehen die Funktionen `seq()` und `rep()` zur Verfügung.

Die Funktion `seq()` (vom englischen "sequence") bietet die Möglichkeit, regelmäßige Zahlenfolgen zu erzeugen. Sie hat mehrere Argumente, von denen die ersten vier sehr suggestiv `from`, `to`, `by` und `length` lauten. In einem Aufruf von `seq()` dürfen (verständlicherweise) nicht alle vier Argumente gleichzeitig beliebig spezifiziert werden, sondern höchstens drei von ihnen. Der vierte Wert wird von **R** aus den drei angegebenen passend hergeleitet. Ein paar Beispiele sollen ihre Verwendung demonstrieren:

```
> seq( from = -2, to = 8, by = 1)
[1] -2 -1  0  1  2  3  4  5  6  7  8

> seq( from = -2, to = 8, by = 0.8)
[1] -2.0 -1.2 -0.4  0.4  1.2  2.0  2.8  3.6  4.4  5.2  6.0  6.8  7.6

> seq( from = -2, to = 8)
[1] -2 -1  0  1  2  3  4  5  6  7  8

> seq( from = 7, to = -1)
[1]  7  6  5  4  3  2  1  0 -1

> seq( from = -5, length = 12, by = 0.2)
[1] -5.0 -4.8 -4.6 -4.4 -4.2 -4.0 -3.8 -3.6 -3.4 -3.2 -3.0 -2.8
```

```
> seq( from = -5.5, to = 3.5, length = 12)
[1] -5.5000000 -4.6818182 -3.8636364 -3.0454545 -2.2272727 -1.4090909
[7] -0.5909091  0.2272727  1.0454545  1.8636364  2.6818182  3.5000000

> seq( to = -5, length = 12, by = 0.2)
[1] -7.2 -7.0 -6.8 -6.6 -6.4 -6.2 -6.0 -5.8 -5.6 -5.4 -5.2 -5.0
```

Ist die Differenz zwischen Endpunkt `to` und Startwert `from` kein ganzzahliges Vielfaches der Schrittweite `by`, so endet die Folge beim letzten Wert *vor* `to`. Fehlen die Angaben einer Schrittweite `by` und einer Folgenlänge `length`, wird `by` automatisch auf 1 oder -1 gesetzt. Aus drei angegebenen Argumentwerten wird der fehlende vierte automatisch passend bestimmt.

Bemerkung: Argumentwerte können auch über die *Position* im Funktionsaufruf gemäß `seq( from, to, by, length)` übergeben werden, was zahlreiche Abkürzungsmöglichkeiten bietet (aber den Benutzer für die Korrektheit der Funktionsaufrufe verantwortlich macht und einige Gefahren birgt):

```
> seq( -2, 8)
[1] -2 -1  0  1  2  3  4  5  6  7  8

> seq( -2, 8, 0.8)
[1] -2.0 -1.2 -0.4  0.4  1.2  2.0  2.8  3.6  4.4  5.2  6.0  6.8  7.6
```

Zu den Details der Argumentübergabemöglichkeiten in Funktionsaufrufen gehen wir im Abschnitt 6.5 „Spezifizierung von Funktionsargumenten“ ein.

Für Zahlenfolgen, deren Schrittweite 1 ist, gibt es noch eine weitere Abkürzung, den Doppelpunkt-Operator:

```
> 2:11
[1]  2  3  4  5  6  7  8  9 10 11

> -1:10
[1] -1  0  1  2  3  4  5  6  7  8  9 10

> -(1:10)
[1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10

> 2:-8
[1]  2  1  0 -1 -2 -3 -4 -5 -6 -7 -8
```

Offenbar liefert er dasselbe wie die Verwendung von lediglich den zwei Argumenten `from` und `to` in `seq()`. An Obigem zeigt sich auch schon, dass in Ausdrücken auf die Priorität der verwendeten Operatoren (hier das unäre Minus und der Doppelpunkt-Operator) zu achten ist und nötigenfalls Klammern zu verwenden sind, um Prioritäten zu verändern.

Mit der Funktion `rep()` (von “repeat”) lassen sich Vektoren erzeugen, deren Elemente aus Strukturen entstehen, die (möglicherweise auf komplizierte Art und Weise) wiederholt werden. Die Argumente von `rep()` lauten `x`, `times`, `length.out` und `each`, wobei `x` den Vektor der zu replizierenden Elemente erhält und die anderen Argumente spezifizieren, wie dies zu geschehen hat. In folgendem Beispiel der einfachsten Form

```
> rep( x = 1:3, times = 4)
[1] 1 2 3 1 2 3 1 2 3 1 2 3
```

werden vier hintereinandergehängte Kopien des Vektors `1:3` erzeugt.

Wird dem `times`-Argument ein Vektor der gleichen Länge wie `x` übergeben, dann erwirkt jedes Element des `times`-Arguments eine entsprechende Vervielfältigung des korrespondierenden Elements von `x`:

```
> rep( x = c( -7, 9), times = c( 3, 5))
[1] -7 -7 -7  9  9  9  9  9
```

Das Argument `length.out` gibt an, wie lang der Ergebnisvektor sein soll; die Elemente von `x` werden so oft zyklisch repliziert, bis diese Länge erreicht ist:

```
> rep( x = -1:1, length.out = 11)
[1] -1  0  1 -1  0  1 -1  0  1 -1  0
```

Mit `each` wird angegeben, wie oft *jedes* Element von `x` wiederholt werden soll, was, falls noch andere Argumente angegeben sind, stets *vor* den übrigen Replikationsoperationen gemacht wird:

```
> rep( x = c( -1, 1), each = 3)
[1] -1 -1 -1  1  1  1

> rep( x = c( -1, 0), times = 3, each = 2)
[1] -1 -1  0  0 -1 -1  0  0 -1 -1  0  0

> rep( x = c( -1, 0), length.out = 13, each = 5)
[1] -1 -1 -1 -1 -1  0  0  0  0  0 -1 -1 -1
```

2.2.2 Elementare Vektoroperationen

Wir fassen obige Beispiele in folgender Übersicht zusammen und zählen weitere Funktionen und Operationen auf, welche in **R** für Vektoren zur Verfügung stehen:

R-Befehl und Resultat	Bedeutung/Bemerkung
`> y <- c( 4, 1, 4, 8, 5, 3)`	Die Argumente der Funktion `c()` werden zu einem Vektor zusammengefasst und durch `<-` als Objekt unter dem Namen `y` abgespeichert.
`> length( y)` `[1] 6`	Länge des Vektors `y` (= Anzahl der Vektorelemente).

`> seq( from, to, by, length, along)`
`....`
`> seq( along = y)`
`[1] 1 2 3 4 5 6`
	Für Beispiele bzgl. der ersten vier Argumente siehe oben. `seq( along = y)` liefert dasselbe wie `1:length( y)`, außer wenn `y` die Länge 0 hat, dann liefert es 0.

`> rep( x, times, length.out, each)` `....`	Für Beispiele siehe oben.
`> rev( y)` `[1] 3 5 8 4 1 4`	Kehrt die Reihenfolge der Elemente eines Vektors um.
`> unique( y)` `[1] 4 1 8 5 3`	Liefert die Elemente des Eingabevektors ohne Wiederholungen.
`> sort( y)` `[1] 1 3 4 4 5 8`	Sortiert die Elemente aufsteigend (je nach Modus z. B. numerisch oder lexikografisch) und liefert also die „Ordnungsstatistiken“.

`> rank( y)` `[1] 3.5 1.0 3.5 6.0 5.0 2.0`	Bildet den zum Eingabevektor gehörenden Rangvektor. Bindungen liefern (per Voreinstellung) mittlere Ränge (Engl.: "midranks").
`> order( y)` `[1] 2 6 1 3 5 4`	Liefert den Vektor der *Indizes* der Eingabedaten für deren aufsteigende Sortierung: Das erste Element von `order( y)` ist der Index des kleinsten Wertes in `y`, das zweite der des zweitkleinsten usw.

Empfehlung: Viel mehr und detailliertere Informationen zu den einzelnen Funktionen liefert jeweils die Online-Hilfe.

2.3 Arithmetik und Funktionen für `numeric`-Vektoren

Die **R**-Arithmetik und viele andere Funktionen für `numeric`-Vektoren operieren auf den Vektoren *elementweise* (also *vektorisiert*), was gelegentlich etwas gewöhnungsbedürftig, oft suggestiv, aber auf jeden Fall sehr leistungsstark ist:

```
> breite <- c( 25, 17, 34, 6);   breite * hoehe    # "hoehe" von Seite 16 oben
[1] 4000 2380 5270 1050
```

Vektoren, die im selben Ausdruck auftreten, brauchen nicht die gleiche Länge zu haben. Kürzere Vektoren in diesem Ausdruck werden durch zyklische (möglicherweise unvollständige) Wiederholung ihrer Elemente auf die Länge des längsten Vektors gebracht. Dies geschieht ohne Warnung, wenn die kleinere Vektorlänge ein Teiler der größeren ist! Bei unterschiedlichen Längen der beteiligten Vektoren hat das Resultat also die Länge des längsten Vektors in diesem Ausdruck. Ein Skalar (= Vektor der Länge 1) wird einfach entsprechend oft repliziert:

```
> fahrenheit <- c( 17, 32, 0, 104, -12)
> (celsius <- (fahrenheit - 32) * 5/9)   # Umrechnung Fahrenheit in Celsius
[1]  -8.333333   0.000000 -17.777778  40.000000 -24.444444
> breite * hoehe * c( 2, 3) + 12
[1]  8012  7152 10552  3162
```

Wir listen in den folgenden Tabellen verschiedene Funktionen und Operationen auf, welche in **R** für `numeric`-Vektoren zur Verfügung stehen und teilweise speziell für die `integer`-Vektoren die Ganzzahlarithmetik realisieren. Hierzu verwenden wir drei Beispielvektoren:

```
> x <- c( -0.3691, 0.3537, -1.0119, -2.6563, NA, 11.3351)
> y <- c( 4, 1, 4, 8, 5, 3);    z <- c( 2, 3)
```

`NA` steht dabei für "not available" und bedeutet, dass der Wert dieses Elements schlicht fehlt, also im statistischen Sinn ein "missing value" ist. Im Allgemeinen liefert eine beliebige Operation, in der irgendwo ein `NA` auftaucht, insgesamt den Wert `NA` zurück. Für manche elementweisen Operationen ist es jedoch sinnvoll, wenn sie im Resultatvektor lediglich an denjenigen Stellen ein `NA` ergeben, wo sich in einem der Eingabevektoren ein `NA` befand.

Es gibt noch eine weitere Sorte von "not available", und zwar im Sinne von „numerisch nicht definiert", weil "not a number": `NaN`. Sie ist das Resultat von Operationen wie $0/0$ oder $\infty - \infty$, wobei ∞ (= Unendlich) in **R** durch das Objekt `Inf` implementiert ist. Für mehr Details siehe Abschnitt 2.11, Seite 57.)

2.3.1 Elementweise Vektoroperationen: Rechnen, runden, formatieren

Für die Verknüpfung arithmetischer Operationen gelten in der Mathematik gewisse Operatorpräzedenzen á la „Punkt- vor Strichrechnung“ oder bei Gleichwertigkeit „von rechts nach links“, die von **R** (natürlich) berücksichtigt werden. Allerdings treten in **R** arithmetische Operatoren häufig in Kombination mit anderen, „nicht-arithmetischen“ Operatoren auf. Hier ist die Operatorpräzedenz nicht unbedingt offensichtlich. Für Klarheit sorgt hier die Online-Hilfeseite, zu der `?Syntax` führt. Im Übrigen ist die Verwendung von Paaren runder Klammern stets möglich, um eine gewisse, evtl. von der üblichen Ordnung abweichende Auswertungsreihenfolge zu erzwingen (oder einfach nur für sich selbst sichtbar sicherzustellen).

Die Hilfeseite zu den arithmetischen Operatoren erreicht man durch `?Arithmetic`.

`> y + z`
`[1] 6 4 6 11 7 6`
`> y + x`
`[1] 3.6309  .... NA 14.3351`
	Elementweise Addition (mit zyklischer Replikation des kürzeren Vektors und mit rein elementweiser Beachtung des NA-Status’)

`> y - z` `[1] 2 -2 2 5 3 0`	Elementweise Subtraktion
`> y * z` `[1] 8 3 8 24 10 9`	Elementweise Multiplikation
`> y / z` `[1] 2.0000 0.3333 2.0000 ....`	Elementweise Division
`> y^z` `[1] 16 1 16 512 25 27`	Elementweise Exponentiation (siehe auch das „Beachte“ am Ende dieser Tabelle)
`> y %/% z` `[1] 2 0 2 2 2 1`	Elementweise ganzzahlige Division
`> y %% z` `[1] 0 1 0 2 1 0`	Elementweise Modulo-Funktion, d. h. Rest bei ganzzahliger Division.

`> pmax( x, y)`
`[1]  4.0000  1.0000  4.0000`
`[4]  8.0000  NA 11.3351`
`> pmin( x, y)`
`[1] -0.3691  0.3537 -1.0119`
`[4] -2.6563  NA  3.0000`
	`pmax()` bzw. `pmin()` ergibt das elementweise (oder auch „parallele“) Maximum bzw. Minimum mehrerer Vektoren (und nicht nur zweier wie hier im Beispiel).

`> pmin( x, y, na.rm = TRUE)`
`[1] -0.3691  0.3537 -1.0119`
`[4] -2.6563  5.0000  3.0000`
	Mit dem Argument `na.rm = TRUE` werden NAs ignoriert. Ausnahme: Ist in allen Vektoren dasselbe Element NA, so ist auch im Resultatvektor dieses Element NA.

`> ceiling( x)` `[1] 0 1 -1 -2 NA 12`	Rundet elementweise zur nächsten ganzen Zahl *auf*.
`> floor( x)` `[1] -1 0 -2 -3 NA 11`	Rundet elementweise zur nächsten ganzen Zahl *ab*, ist also die Gaußklammer.
`> trunc( x)` `[1] 0 0 -1 -2 NA 11`	Rundet elementweise zur nächsten ganzen Zahl *in Richtung Null*, d. h., liefert den ganzzahligen Anteil.

`> round( x)`
`[1] 0 0 -1 -3 NA 11`
`> round( c( 2.5, 3.5))`
`[1] 2 4`
	Rundet elementweise auf die nächste ganze Zahl, wobei Werte der Art „$k.5$“ auf die nächste gerade ganze Zahl gerundet werden.

`> round( x, digits = 1)`
`[1] -0.4 0.4 -1.0 -2.7 NA 11.3`
`> round( c( 1.25, 1.35), 1)`
`[1] 1.2 1.4`
	Das zweite (optionale) Argument `digits` gibt, falls positiv, die Zahl der Nachkommastellen, falls negativ, die Zehnerstelle vor dem Komma an, auf die gerundet werden soll. Werte der Art „...$k5$...“ werden an der Stelle k auf die nächste gerade ganze Ziffer gerundet.

`> signif( x, digits = 2)`
`[1] -0.37  0.35 -1.00 -2.70`
`[5]    NA 11.00`
	Rundet auf die für `digits` angegebene Gesamtzahl an signifikanten Stellen (wobei Nullen lediglich für ein einheitliches Druckformat angehängt werden).

`> print( x, digits = 2)`
`[1] -0.37  0.35 -1.01 -2.66`
`[5]    NA 11.34`
	Druckt `x` mit der für `digits` angegebenen Anzahl an Nachkommastellen (gerundet) in einheitlichem Druckformat.

`> format( x, digits = 2)`
`[1] "-0.37" " 0.35" "-1.01"`
`[4] "-2.66" "   NA" "11.34"`
	Erzeugt eine `character`-Darstellung des `numeric`-Vektors `x` in einheitlich langer, rechtsbündiger Formatierung der Elemente von `x` mit der durch `digits` angegebenen Zahl an (gerundeten) Nachkommastellen. Dazu wird mit führenden Leerzeichen aufgefüllt.

`> format( y, nsmall = 2)`
`[1] "4.00" "1.00" "4.00" "8.00"`
`[5] "5.00" "3.00"`
	`nsmall` gibt die Mindestanzahl an Ziffern rechts vom Dezimalpunkt an. Erlaubt: $0 \leq$ `nsmall` ≤ 20. Es wird rechts mit Nullen aufgefüllt.

`> format( (-2)^c( 3,10,21,32),`
`+ scientific = TRUE, digits = 4)`
`[1] "-8.000e+00" " 1.024e+03"`
`[3] "-2.097e+06" " 4.295e+09"`
	`scientific = TRUE` veranlasst die Darstellung in wissenschaftlicher Notation, wobei `digits` wie eben funktioniert. (Viel mehr Informationen liefert die Online-Hilfe. Nützlich ist auch `formatC()`.)

Beachte: Die Berechnung der elementweisen Exponentiation `x^n` ist für kleine ganzzahlige Werte von n (≥ 2) effizienter durch die explizite Angabe als Produkt $x * \cdots * x$ zu erreichen. Also insbesondere in sehr rechenintensiven Simulationen sollte man z. B. `x^2` und `x^3` explizit als `x * x` bzw. `x * x * x` programmieren. Dies kann durch einen reduzierten Laufzeitaufwand die Geduld des/der „Simulanten/in“ ein wenig entlasten.

2.3.2 Zusammenfassende und sequenzielle Vektoroperationen: Summen, Produkte, Extrema

Memo: `x` und `y` sind die Objekte von Seite 19.

`> sum( y)` `[1] 25`	Summe der Vektorelemente.

`> sum( x)`
`[1] NA`
`> sum( x, na.rm = TRUE)`
`[1] 7.6515`
	Konsequenz eines NA-Elements hierbei.
	Aber wenn das logische Argument `na.rm` auf TRUE gesetzt wird, führt dies zum Ausschluss der NA-Elemente *vor* der Summenbildung.

`> prod( y)` `[1] 1920`	Produkt der Vektorelemente. Argument `na.rm` steht auch hier zur Verfügung.
`> cumsum( y)` `[1] 4 5 9 17 22 25`	Kumulative Summen,
`> cumprod( y)` `[1] 4 4 16 128 640 1920`	kumulative Produkte,
`> cummax( y)` `[1] 4 4 4 8 8 8`	kumulative Maxima bzw.
`> cummin( y)` `[1] 4 1 1 1 1 1`	kumulative Minima der Vektorelemente. (Das Argument `na.rm` steht **nicht** zur Verfügung.)
`> diff( y)` `[1] -3  3  4 -3 -2`	Sukzessive Differenzen der Vektorelemente: Element i ist der Wert $y_{i+1} - y_i$ (d. h. der Zuwachs von y_i auf y_{i+1}).
`> diff( y, lag = 2)` `[1]  0  7  1 -5`	Sukzessive Differenzen der Vektorelemente, die *lag* Elemente voneinander entfernt sind: Element i ist der Wert $y_{i+lag} - y_i$.

2.3.3 "Summary statistics" (`summary()` etc.)

Statistikspezifische zusammenfassende Vektoroperationen – im Englischen auch "summary statistics" genannt – wollen wir separat aufführen: Die hierbei verwendeten Funktionen zur Bestimmung von Maximum, Minimum, Spannweite, Mittelwert, Median, Varianz und Quantilen eines Datensatzes bzw. der Korrelation zweier Datensätze würden im Fall fehlender Werte (NAs) in den Daten ebenfalls NA zurückliefern oder die Berechnung mit einer Fehlermeldung abbrechen. Dies kann durch das Argument `na.rm` geändert werden.
Memo: `x` und `y` stammen von Seite 19.

`> max( x)`
`[1] NA`
`> max( x, na.rm = TRUE)`
`[1] 11.3351`
`> min( x, na.rm = TRUE)`
`[1] -2.6563`
	Wie erwartet, führt ein NA bei `max()` etc. ebenfalls zu NA als Resultat.
	Das Argument `na.rm = TRUE` erzwingt, dass alle NAs in dem Vektor `x` ignoriert werden, bevor die jeweilige Funktion angewendet wird. Die Voreinstellung ist `na.rm = FALSE`.

`> which.max( x)`
`[1] 6`
`> which.min( x)`
`[1] 4`
	Liefert den Index des ersten (!) Auftretens des Maximums (bzw. Minimums) in den Elementen eines Vektors. NAs im Vektor werden ignoriert.

`> range( x, na.rm = TRUE)` `[1] -2.6563 11.3351`	Minimum und Maximum der Werte in `x` (ohne NAs).

`> mean( x, na.rm = TRUE)`
`[1] 1.5303`
`> mean( x, trim = 0.2,`
`+ na.rm = TRUE)`
`[1] -0.3424333`
	Arithmetisches Mittel der Nicht-NA-Werte in `x`. Das Argument `trim` mit Werten in $[0, 0.5]$ spezifiziert, welcher Anteil der sortierten Werte in `x` am unteren und oberen Ende *jeweils* weggelassen (getrimmt) werden soll.

`> median( x, na.rm = TRUE)` `[1] -0.3691`	Der Median der Nicht-NA-Werte in `x`. Zu Details seiner Implementation siehe `?median`.

`> quantile( x, na.rm = TRUE)`
`  0%  25%  50%  75%`
`-2.6563 -1.0119 -0.3691  0.3537`
`  100%`
`11.3351`
`> quantile( x, na.rm = TRUE,`
`+ probs = c( 0, 0.2, 0.9))`
`  0%  20%  90%`
`-2.65630 -1.34078  6.94254`
	Empirische Quantile der Werte in `x`. Die Voreinstellung liefert Minimum, unteres Quartil, Median, oberes Quartil und Maximum der `x`-Elemente. Das Argument `probs` erlaubt die Bestimmung jedes beliebigen Quantils. Es sind neun (!) verschiedene Quantildefinitionen und die entsprechenden Algorithmen zu ihrer Bestimmung über das nicht gezeigte Argument `type` wählbar; siehe Online-Hilfe. Per Voreinstellung (`type = 7`) wird zwischen den Ordnungsstatistiken von `x` linear interpoliert, wobei $x_{i:n}$ als `(i-1)/(length(x)-1)`-Quantil angenommen wird. Für `type = 1` wird als Quantilfunktion die Inverse der empirischen Verteilungsfunktion F_n genutzt: $F_n^{-1}(p) = \inf\{x : F_n(x) \geq p\}$, $0 \leq p \leq 1$.

`> summary( x)`
`  Min. 1st Qu.  Median  Mean`
`-2.6560 -1.0120 -0.3691  1.5300`
`3rd Qu.  Max.  NA's`
`0.3537 11.3400  1.0000`
	Das Resultat von `summary()` angewendet auf einen `numeric`-Vektor `x` ist eine Auswahl von Resultaten obiger Funktionen nach Zählung sowie Elimination der NA-Elemente.

`> var( x, na.rm = TRUE)`
`[1] 31.27915`
 	
`> sd( x, na.rm = TRUE)`
`[1] 5.592777`
	Die Funktionen `var()` und `sd()` zur Berechnung der empirischen Varianz bzw. Standardabweichung der Elemente eines Vektors besitzen das Argument `na.rm`. Es muss zum Ausschluss von NA-Elementen auf `TRUE` gesetzt werden.

`> cov( x, y, use = "complete.obs")`
`[1] -5.75535`
`> var( x, y, use = "complete.obs")`
... *(Dasselbe wie bei* ***var()****)*
 	
`> cor( x, y, use = "complete.obs")`
`[1] -0.4036338`
	`cov()` und `var()` bzw. `cor()` zur Bestimmung der empirischen Kovarianz bzw. Korrelation der Elemente zweier Vektoren besitzen das Argument `use`. Es gibt die Behandlung von NAs an. `use = "complete. obs"` sorgt dafür, dass, wenn eines von zwei i-ten Elementen NA ist, beide ausgeschlossen werden. (Für andere Möglichkeiten siehe die Online-Hilfe.)

Beachte im Vorgriff auf Kapitel 2.8 „Matrizen: Erzeugung, Indizierung, Modifikation und Operationen“: `sum()`, `prod()`, `max()`, `min()`, `range()`, `mean()`, `median()`, `quantile()` und `summary()` können auch direkt auf Matrizen angewendet werden. Sie wirken dann jedoch nicht spalten- oder zeilenweise, sondern auf die als Vektor aufgefasste Gesamtheit der Matrixeinträge. Die Funktionen `cov()` und `var()` bzw. `cor()` ergeben allerdings die empirische Kovarianz- bzw. Korrelations*matrix* der Spaltenvektoren, wohingegen `sd()` pro Matrixspalte die empirische Standardabweichung ihrer Einträge liefert.

2.3.4 Mathematische Funktionen

Selbstverständlich stehen auch einige eher „rein mathematische“ Funktionen zur Verfügung, die ebenfalls elementweise auf `numeric`-Vektoren angewendet werden:

Name			Funktion
`sqrt`			Quadratwurzel
`abs`			Absolutbetrag
`sin`	`cos`	`tan`	Trigonometrische Funktionen
`asin`	`acos`	`atan`	Inverse trigonometrische Funktionen
`sinh`	`cosh`	`tanh`	Hyperbolische trigonometrische Funktionen
`asinh`	`acosh`	`atanh`	Inverse hyperbolische trigonometrische Funktionen
`exp`	`log`		Exponentialfunktion und natürlicher Logarithmus
`log10`	`logb`		Dekadischer Logarithmus und Logarithmus zu freier Basis
`beta`	`lbeta`		Betafunktion und ihr natürlicher Logarithmus
`gamma`	`lgamma`		Gammafunktion Γ und ihr natürlicher Logarithmus
`factorial`	`lfactorial`		Ist $\Gamma(x+1)$ für $x \in \mathbb{R}$ und die Fakultätsfunktion $x!$, falls $x \in \mathbb{N}_0$, bzw. ihr natürlicher Logarithmus
`choose`	`lchoose`		Binomialkoeffizienten bzw. ihr natürlicher Logarithmus

Für weitere Informationen über die aufgeführten Funktionen siehe die Online-Hilfe direkt via ihre Namen wie z. B. `?sqrt` und `?log` oder teilweise auch via `?Trig` oder `?Specials`.

Beachte:

- $\Gamma(x) = (x-1)!$, falls x eine natürliche Zahl ist.
- Um die Basis der Logarithmus-Funktion zu wählen, ist für `logb()` das Argument `base` da. Beispielsweise liefert `logb( x, base = 2)` den binären Logarithmus. Natürlich ist `logb( x, base = 10)` gerade gleich `log10( x)`.
- π ist unter dem Namen `pi` verfügbar und e ist `exp( 1)`.

- Für aufwändigere mathematische Simulationen kann es sich hinsichtlich der Rechenzeit lohnen, die wiederholte Anwendung obiger Funktionen auf denselben oder einen leicht variierten Vektor zu vermeiden. Beispiel: Die Berechnung von `exp( x) + exp( -x)` für einen langen `numeric`-Vektor `x` ist effizienter durch `z <- exp( x); z + 1/z` zu erzielen, weil die Kehrwertbildung eine „einfachere“ Funktion ist als die Exponentialfunktion.

- Zum interessanten Thema Rechengenauigkeit im Sinne von numerischer Präzision finden sich in der Antwort zur R-FAQ 7.31 wichtige Informationen und im **R**-Wiki, wenn man dort nach dem Begriff “`r_accuracy`” suchen lässt (siehe `http://rwiki.sciviews.org`). Das für sich allein schon interessante Thema ist in der Arbeit [39, Goldberg (1991)] deutlich weiter ausgeführt, die auch auf der Hilfeseite `?Arithmetic`.

- `choose( n, k)` liefert die *Anzahl* aller k-elementigen Teilmengen einer n-elementigen Grundmenge. Die explizite Erzeugung dieser Teilmengen hingegen – und in gewissem Umfang deren Weiterverarbeitung – ermöglicht die Funktion `combn()` (siehe ihre Online-Hilfe).

2.4 `logical`-Vektoren und logische Operatoren

Mit den „reservierten Worten“ `FALSE` und `TRUE` werden in **R** die logischen Konstanten „falsch“ und „wahr“ bezeichnet. Die äquivalent erscheinenden Objekte `F` und `T` hingegen sind globale Variablen, deren Werte per Voreinstellung zwar auf `FALSE` bzw. `TRUE` gesetzt sind, vom Benutzer aber überschrieben werden können! Vorsicht also mit benutzerdefinierten Objekten namens `F` oder `T`, da sie eventuell die eigentlich beabsichtigten Werte `FALSE` oder `TRUE` maskieren. (Siehe hierzu auch den Punkt zum Unterschied zwischen `TRUE` und `T` auf Seite 26.)

Ein logischer Wert ist das Resultat der Auswertung eines booleschen Ausdrucks (d. h. einer Bedingung). Im folgenden Beispiel wird der Vektor `x` (wie üblich) elementweise mit einem Skalar verglichen. Außerdem werden zwei Vektoren `a` und `b` direkt erzeugt. Sowohl der Resultatvektor für `x > 175` als auch `a` und `b` haben den Modus `logical`:

```
> x <- c( 160, 145, 195, 173, 181);    x > 175
[1] FALSE FALSE  TRUE FALSE  TRUE

> (a <- c( T, T, F, F));    (b <- c( T, FALSE, TRUE, F))
[1]  TRUE  TRUE FALSE FALSE
[1]  TRUE FALSE  TRUE FALSE
```

In den folgenden beiden Tabellen finden sich (fast) alle logischen Operatoren, die in **R** für logische Vektoren und Ausdrücke zur Verfügung stehen. Zur Online-Hilfe über verschiedene logische Operatoren oder die logischen Konstanten kommen Sie via `?Logic` bzw. `?logical`, oder z. B. durch `?"!"`, also durch das `?` angewendet auf den Operator in Hochkommata.

2.4.1 Elementweise logische Operationen

Operator	Bedeutung und Bemerkung	Anwendungsbeispiel mit `a`, `b`, `x` von oben
$<$	kleiner	`> x < 190` `[1] TRUE TRUE FALSE TRUE TRUE`
$>$	größer	`> x > 173` `[1] TRUE FALSE TRUE FALSE TRUE`
$==$	gleich	`> x == 145` `[1] FALSE TRUE FALSE FALSE FALSE`

<=	kleiner oder gleich	`> x <= 190` `[1] TRUE TRUE FALSE TRUE TRUE`
>=	größer oder gleich	`> x >= 173` `[1] FALSE FALSE TRUE TRUE TRUE`

!	nicht (= Negation)	`> !a`
		`[1] FALSE FALSE TRUE TRUE`
		`> !(x > 173)`
		`[1] TRUE TRUE FALSE TRUE FALSE`

! =	ungleich	`> x != 145` `[1] TRUE FALSE TRUE TRUE TRUE`
&	und	`> a & b` `[1] TRUE FALSE FALSE FALSE`
|	oder	`> a | b` `[1] TRUE TRUE TRUE FALSE`
xor	exclusives oder (entweder–oder)	`> xor( a, b)` `[1] FALSE TRUE TRUE FALSE`

Beachte:

- In arithmetischen Ausdrücken findet eine automatische Konversion (Engl.: "coercing") von `logical` in `numeric` statt, und zwar wird `TRUE` zu 1 und `FALSE` zu 0, was – mit Umsicht – sehr effizient genutzt werden kann:

```
> a * 2                    > (x > 175) * 1          > sum( x <= 190)
[1] 2 2 0 0                [1] 0 0 1 0 1            [1] 4
```

- Diese automatische Konversion findet im Prinzip sogar in beide Richtungen statt, fallweise also auch von `numeric` in `logical`! Finden Sie heraus, was die folgenden Ausdrücke im einzelnen liefern und machen Sie sich die Ursache(n) für die jeweiligen Resultate klar:

```
> !0      > !1      > !2      > !0 + !0      > !0 - !0      > !0 * !0
?         ?         ?         ?              ?              ?

> !0 + !0 == !0 - !0          > !0 * !0 == !0^2
?                             ?
```

  Aber:

```
> (!0) + (!0)         > (!0) - (!0)         > (!0) * (!0)         > (!0)^2
?                     ?                     ?                     ?
```

- Beim Vergleich mathematisch reellwertiger Größen, die in **R** als `numeric` gespeichert sind, ist die begrenzte, digitale Maschinengenauigkeit zu berücksichtigen. Beispiel:

```
> (0.2 - 0.1) == 0.1;    (0.3 - 0.2) == 0.1
[1] TRUE
[1] FALSE
```

  Das ist kein **R**-spezifischer „Fehler", sondern ein fundamentales, computerspezifisches Problem (was z. B. auch unter `http://cran.r-project.org/doc/FAQ/R-FAQ.html` in der R-FAQ 7.31 "Why doesn't R think these numbers are equal?" besprochen wird).) Hier können die Funktionen `all.equal()` und `identical()` weiterhelfen, bezüglich deren Arbeitsweise wir aber auf die Online-Hilfe verweisen.

- Beispiel zum Unterschied zwischen `TRUE` und `T` (bzw. zwischen `FALSE` und `F`):

```
> TRUE <- 5
Fehler in TRUE <- 5 : ungueltige (do_set) linke Seite in Zuweisung
> T <- 5; T
[1] 5
```

2.4.2 Zusammenfassende logische Operationen

Operator	Bedeutung und Bemerkung	Anwendungsbeispiel mit `a`, `b` und `x` von oben

&&	„sequenzielles“ und: Ergebnis ist `TRUE`, wenn beide Ausdrücke `TRUE` sind. Ist aber der linke Ausdruck `FALSE`, wird der rechte gar nicht erst ausgewertet.	`> min( x-150) > 0 &&`
		`+ max( log( x-150)) < 1`
		`[1] FALSE`

||	„sequenzielles“ oder: Ergebnis ist `TRUE`, wenn ein Ausdruck `TRUE`. Ist aber der linke Ausdruck schon `TRUE`, wird der rechte nicht mehr ausgewertet.	`> min( x-150) < 0 ||`
		`+ max( log( x-150)) < 1`
		`[1] TRUE`

all	Sind alle Elemente `TRUE`? (Das Argument `na.rm` steht zur Verfügung.)	`> all( a)` `[1] FALSE`
any	Ist mindestens ein Element `TRUE`? (`na.rm` steht zur Verfügung.)	`> any( a)` `[1] TRUE`
which	Welche Elemente sind `TRUE`?	`> which ( b)` `[1] 1 3`

Memos:

- Leerzeichen sind als Strukturierungshilfe sehr zu empfehlen, da sie nahezu beliebig zwischen alle **R**-Ausdrücke eingestreut werden können. Gelegentlich sind sie sogar zwingend notwendig: `sum(x<-1)` kann durch Leerzeichen zwei Ausdrücke mit völlig verschiedenen Bedeutungen ergeben, nämlich `sum(x < -1)` und `sum(x <- 1)`. **R** verwendet für `sum(x<-1)` die zweite Interpretation, also muss man für die erste durch die entsprechende Verwendung von Leerzeichen sorgen (oder von Klammern wie in `sum(x<(-1))`, was aber ziemlich unübersichtlich werden kann).

- An dieser Stelle erinnern wir noch einmal an die bereits am Anfang von §2.3.1 auf Seite 20 erwähnten Regeln zur Operatorpräzedenz und an die Hilfeseite zur Syntax.

2.5 character-Vektoren und elementare Operationen

Vektoren des Modus character (kurz: character-Vektoren) bestehen aus Elementen, die Zeichenketten ("strings") sind. Zeichenketten stehen in paarweisen doppelten Hochkommata, wie z. B. "x-Werte", "Peter, Paul und Mary", "NEUEDATEN2002" und "17". Paarweise einfache Hochkommata können auch verwendet werden, allerdings nicht gemischt mit doppelten innerhalb einer Zeichenkette. character-Vektoren werden wie andere Vektoren mit der Funktion c() zusammengesetzt:

```
> (namen <- c( "Peter", "Paul", "Mary"))
[1] "Peter" "Paul"  "Mary"
> weitere.namen <- c( 'Tic', 'Tac', 'Toe')
> (alle.namen <- c( namen, weitere.namen))
[1] "Peter" "Paul"  "Mary"  "Tic"   "Tac"   "Toe"
```

Werden character-Vektoren mit Vektoren anderer Modi, wie z. B. numeric oder logical verknüpft, so werden alle Elemente in den Modus character konvertiert:

```
> c( 1.3, namen, TRUE, F)
[1] "1.3"   "Peter" "Paul"  "Mary"  "TRUE"  "FALSE"
```

2.5.1 Zusammensetzen von Zeichenketten: paste()

Das „nahtlose" Zusammensetzen von Zeichenketten aus korrespondierenden Elementen mehrerer character-Vektoren kann mit der Funktion paste() ("to paste" = kleben) bewerkstelligt werden (siehe Beispiele unter der folgenden Auflistung):

- Eine beliebige Anzahl an Zeichenketten (d. h. an einelementigen character-Vektoren) wird zu einer einzelnen Zeichenkette (in einem einelementigen character-Vektor) zusammengesetzt.
- Vorher werden dabei automatisch logische Werte in "TRUE" bzw. "FALSE" und Zahlen in Zeichenketten, die ihren Ziffernfolgen entsprechen, konvertiert.
- Zwischen die zusammenzusetzenden Zeichenketten wird mit Hilfe des Arguments sep ein (frei wählbares) Trennzeichen eingebaut; Voreinstellung ist das Leerzeichen ("blank"), also sep = " ".
- character-*Vektoren* werden elementweise (unter zyklischer Wiederholung der Elemente der kürzeren Vektoren) zu character-Vektoren zusammengesetzt, derart dass die Zeichenketten korrespondierender Elemente zusammengesetzt werden.

Beispiele:

```
> paste( "Peter", "ist", "doof!")                      # Aus drei Zeichenketten
[1] "Peter ist doof!"                                  # wird eine.

> paste( "Tic", "Tac", "Toe", sep = ",")               # Dito, aber mit "," als
[1] "Tic,Tac,Toe"                                      # Trennzeichen.

> paste( "Hoehe", hoehe, sep = "=")                    # Aus numeric wird cha-
[1] "Hoehe=160" "Hoehe=140" "Hoehe=155" "Hoehe=175"    # racter und Vektoren ...

> paste( "Data", 1:4, ".txt", sep = "")                # werden zyklisch (teil-)
[1] "Data1.txt" "Data2.txt" "Data3.txt" "Data4.txt"    # repliziert bis ...

> paste( c( "x", "y"), rep( 1:5, each = 2), sep = "") # alle Laengen zueinan-
[1] "x1" "y1" "x2" "y2" "x3" "y3" "x4" "y4" "x5" "y5" # der passen.
```

2.5.2 Benennung und „Entnennung“ von Vektorelementen: `names()` & `unname()`

Eine besondere Anwendung von Zeichenketten ist die Benennung der Elemente eines Vektors mit Hilfe der Funktion `names()` (wobei diese auf der linken Seite des Zuweisungsoperators `<-` verwendet wird). Dies ändert nichts an den Werten der Elemente des Vektors:

```
> alter <- c( 12, 14, 13, 21, 19, 23)
> names( alter) <- alle.namen;   alter
 Peter Paul Mary Tic Tac Toe
    12   14   13  21  19  23
```

Damit ist der Vektor `alter` nach wie vor ein `numeric`-Vektor, hat aber nun zusätzlich das Attribut `names`, wie die Funktion `attributes()` zeigt.

```
> attributes( alter)
$names
[1] "Peter" "Paul"  "Mary"  "Tic"   "Tac"   "Toe"
```

Es ist auch möglich, die Elemente eines Vektors schon bei seiner Erzeugung zu benennen:

```
> c( Volumen = 3.21, Masse = 15.8)
Volumen   Masse
   3.21   15.80
```

Sollte die Benennung von Vektorelementen stören (z. B. weil sie bei langen Vektoren mit außerdem langen Elementenamen viel Speicherplatz in Anspruch nimmt), kann sie mit `unnname()` entfernt werden (was in der Regel auch den Speicherplatzbedarf des Vektors stark reduziert):

```
> object.size( alter)   # liefert ca. die Groesse seines Argumentes in Bytes.
[1] 576
> unname( alter);     object.size( unname( alter))
[1] 12 14 13 21 19 23
[1] 88
```

Bemerkung: Der Speicherplatzbedarf von `numeric`-Vektoren mit mindestens 16 unbenannten Elementen ist acht Bytes pro Element plus ein konstanter “Overhead” von 40 Bytes. Pro Elementname kommen (bei kurzen Elementnamen) ebenfalls *in etwa* acht Bytes hinzu, was die Größe eines solchen Vektors also ungefähr verdoppelt. (Für Vektoren mit weniger als 16 Elementen ist der Speicherplatzbedarf überproportional größer, was i. d. R. aber völlig irrelevant ist.)

2.5.3 Weiter Operationen: `strsplit()`, `nchar()`, `substring()`, `abbreviate()` & Co.

Ein paar Funktionen, die für die Arbeit mit `character`-Vektoren von Nutzen sein können, sind in der folgenden Tabelle aufgeführt:

`> paste( ...., sep, collapse)`
`> paste( weitere.namen,`
`+ collapse = " und ")`
`[1] "Tic und Tac und Toe"`
`> paste( weitere.namen,`
`+ sep = " und ")`
`[1] "Tic" "Tac" "Toe"`
	Für verschiedene Beispiele siehe oben.
	Die Angabe des Arguments `collapse` führt zum „Kollaps“ aller Eingabezeichenketten zu *einer* Zeichenkette, wobei der Wert von `collapse` eingefügt wird. Beachte den Unterschied zur Wirkung von `sep = " und "`!

`> strsplit( ....)` `....`	Die sehr nützliche „Umkehrung“ von `paste()`, für deren – etwas komplexere – Funktionsweise wir auf die Online-Hilfe verweisen.

`> nchar( alle.namen)` `[1] 5 4 4 3 3 3`	Liefert die Längen der in seinem Argument befindlichen Zeichenketten (falls nötig, nach Konversion des Arguments in einen `character`-Vektor).

`> substring( text = namen,`
`+ first = 2, last = 3)`
`[1] "et" "au" "ar"`
 	
`> substring( namen, 2)`
`[1] "eter" "aul"  "ary"`
 	
`> substring( namen, 2, 3) <- "X"`
`> namen`
`[1] "PXer" "PXl"  "MXy"`
	Produziert einen Vektor von Teil-Zeichenketten der Eingabe an `text`. Das Argument `first` bestimmt die Startposition und `last` die Endposition der Teil-Zeichenketten in den Eingabezeichenketten. Ohne Wert für `last` gehen die Teil-Zeichenketten bis zum Ende der Eingabezeichenketten.
	Mit einer Zuweisungsanweisung werden im Eingabevektor die jeweiligen Teile durch die rechte Seite von `<-` ersetzt.
	(Die Werte für `first` und `last` können Vektoren sein; dann wird auf dem Eingabevektor entsprechend elementweise operiert.)

`> abbreviate( names = alle.namen,`
`+ minlength = 2)`
`Peter Paul Mary  Tic  Tac  Toe`
`"Pt" "Pl" "Mr" "Tic" "Tac" "To"`
	Generiert (mit Hilfe eines fürs Englische maßgeschneiderten Algorithmus') aus den Zeichenketten des Eingabevektors eindeutige Abkürzungen der Mindestlänge `minlength`. Der Resultatvektor hat benannte Elemente, wobei die Elementnamen die Zeichenketten des Eingabevektors sind.

Hinweise:

- Für weitere, sehr leistungsfähige, aber in ihrer Umsetzung etwas kompliziertere Zeichenkettenbearbeitungen verweisen wir auf die Online-Hilfe der Funktion `gsub()` und ihre dort genannten „Verwandten“.
- Das Zusatzpaket `stringr` bietet eine Sammlung an Zeichenkettenfunktionen, die konsistenter und einfacher zu verwenden sein sollen als die in “base **R**” bereits zur Verfügung stehenden.
- Mit `tolower()`, `toupper()` und `chartr()` lassen sich ein paar spezielle, einfachere Umwandlungen elegant bewerkstelligen (siehe deren Online-Hilfe).
- Gelegentlich ganz nützlich sind die in **R** schon vorhandenen `character`-Vektoren `letters`, `LETTERS`, `month.name` und `month.abb`. Sie enthalten als Elemente das Alphabet in Klein- bzw. Großbuchstaben sowie die Monatsnamen bzw. deren Abkürzungen:

```
> letters
 [1] "a" "b" "c" .... "z"
> LETTERS
 [1] "A" "B" "C" .... "Z"
> month.name
 [1] "January"   "February"  "March" .... "December"
> month.abb
 [1] "Jan" "Feb" "Mar" .... "Dec"
```

2.6 Indizierung und Modifikation von Vektorelementen: [ ]

Der Zugriff auf einzelne Elemente eines Vektors wird durch seine Indizierung mit der Nummer des gewünschten Elements erreicht. Dies geschieht durch das Anhängen der Nummer in eckigen Klammern `[ ]` an den Namen des Vektors: `x[`i`]` für $i > 0$ liefert das i-te Element des Vektors `x`, sofern dieser mindestens i Elemente besitzt; andernfalls erhält man den Wert `NA` zurück. Die Indizierung der Elemente beginnt mit 1 (im Gegensatz zur Sprache C, wo sie mit 0 beginnt). Der Index 0 (Null), also `x[0]` liefert einen leeren Vektor zuück.

2.6.1 Indexvektoren

Durch die Angabe eines *Indexvektors* kann auf ganze Teilmengen von Vektorelementen zugegriffen werden. Für die Konstruktion eines Indexvektors gibt es vier Methoden:

1. **Indexvektor aus positiven Integer-Werten:** Die Elemente des Indexvektors müssen aus der Menge $\{1, \ldots,$ `length( x)`$\}$ sein. Diese Integer-Werte indizieren die Vektorelemente und liefern sie *in der Reihenfolge*, wie sie im Indexvektor auftreten, zu einem Vektor zusammengesetzt zurück:

```
> alter[ 5]
 Tac
  19
> alter[ c( 4:2, 13)]
 Tic Mary Paul
  21   13   14  NA
> c( "Sie liebt mich.", "Sie liebt mich nicht.")[ c( 1,1,2,1,1,2)]
[1] "Sie liebt mich."   "Sie liebt mich."   "Sie liebt mich nicht."
[4] "Sie liebt mich."   "Sie liebt mich."   "Sie liebt mich nicht."
```

2. **Indexvektor aus negativen Integer-Werten:** Die Menge der Elemente des Indexvektors (d. h. duplizierte Negativindizes werden ignoriert) spezifiziert in diesem Fall die Elemente, die *ausgeschlossen* werden:

```
> alle.namen[ -(1:3)]
[1] "Tic" "Tac" "Toe"
> alter[ -length( alter)]
 Peter Paul Mary Tic Tac
    12   14   13  21  19
```

3. **Logischer Indexvektor:** Ein Indexvektor mit logischen Elementen wählt die Vektorelemente aus, an deren Position im Indexvektor ein `TRUE`-Wert steht; `FALSE`-Werten entsprechende Elemente werden ausgelassen. Ein zu kurzer Indexvektor wird (nötigenfalls unvollständig) zyklisch repliziert, ein zu langer liefert NA für die überzähligen Indizes zurück:

```
> alter[ c( TRUE, TRUE, FALSE, FALSE, TRUE)]
Peter  Paul   Tac   Toe   <NA>
   12    14    19    23     NA
> x[ x > 180]             # x von Seite 24
[1] 195 181
> alle.namen[ alter >= 21]
[1] "Tic" "Toe"
> alle.namen[ alter >= 14 & alter < 18]
[1] "Paul"

> letters[ c( FALSE, FALSE, TRUE)]   # Jeder dritte Buchstabe (von 26)
[1] "c" "f" "i" "l" "o" "r" "u" "x"
```

4. **Indexvektor aus Zeichenketten:** Diese Möglichkeit besteht nur, wenn der Vektor, dessen Elemente indiziert werden sollen, benannte Elemente besitzt (also das Attribut `names` hat, wie z. B. nach Anwendung der Funktion `names()`; vgl. §2.5.2). In diesem Fall können die Elementenamen zur Indizierung der Elemente verwendet werden. Ein unzutreffender Elementename liefert ein NA zurück:

```
> alter[ "Tic"]
 Tic
  21
> alter[ c( "Peter", "Paul", "Mary", "Heini")]
Peter  Paul  Mary  <NA>
   12    14    13    NA
> alter[ weitere.namen]
 Tic Tac Toe
  21  19  23
```

2.6.2 Zwei spezielle Indizierungsfunktionen: `head()` und `tail()`

Gelegentlich möchte man auf die ersten oder letzten k Elemente eines Vektors `x` zugreifen, was in ersterem Fall recht leicht durch `x[1:k]` bewältigt wird. Im zweiten Fall bedarf es jedoch der Bestimmung und etwas unübersichtlichen Verwendung der Vektorlänge, z. B. wie in `x[(length( x)-k+1):length( x)]`. Die Funktionen `head()` und `tail()` erleichtern diese Zugriffe auf kompakte Weise (zumindest im Fall von `tail()`):

`> head( alter, n = 2)`
`Peter  Paul`
`  12  14`
`> tail( alter, n = 1)`
`Toe`
`  23`
	Bei positivem `n` vorderer bzw. hinterer Teil der Länge `n` eines Vektors. Voreinstellung für `n`: Jeweils sechs Elemente. (Weiterer Vorteil dieser Funktionen: Beide sind z. B. auch auf Matrizen und Data Frames anwendbar. Siehe die entsprechenden Abschnitte 2.8, Seite 38 und 2.10, Seite 49.)

`> head( alter, n = -2)`
`Peter  Paul  Mary  Tic`
`  12  14  13  21`
`> tail( alter, n = -1)`
`Paul Mary  Tic  Tac  Toe`
`  14  13  21  19  23`
	Bei einem negativen Wert für `n` liefert `head()` alle Elemente *ohne die letzten* |`n`| Stück und `tail()` alle Elemente *ohne die ersten* |`n`| Stück.

2.6.3 Indizierte Zuweisungen

Zuweisungsanweisungen dürfen auf der linken Seite von `<-` ebenfalls einen indizierten Vektor der Form `vektor[ indexvektor]` enthalten. Dabei wird die Zuweisung nur auf die indizierten Elemente dieses Vektors angewendet. Hat das Auswertungsresultat der rechten Seite nicht dieselbe Länge wie der Indexvektor, so wird bei „zu kurzer rechter Seite“ wieder zyklisch aufgefüllt und bei „zu langer rechter Seite“ die rechte Seite abgeschnitten sowie in den Fällen, wo dies nicht „aufgeht“, eine Warnung ausgegeben. Beispiele (mit `x` von Seite 24):

```
> x;    x[ 3] <- 188;   x
[1] 160 145 195 173 181
[1] 160 145 188 173 181
> x[ x > 180] <- c( -1, -2, -3)
Warning message:
number of items to replace is not a multiple of replacement length
> x
[1]  160  145   -1  173   -2
```

Natürlich funktioniert das bei jeder Art von Vektor:

```
> (Farben <- c( "rot", "gruen", "blau")[ c( 1,1,2,1,3,3,1,2,2)])
[1] "rot"   "rot"   "gruen" "rot"   "blau"  "blau"  "rot"  "gruen" "gruen"

> Farben[ Farben == "rot"] <- "rosa";    Farben
[1] "rosa"  "rosa"  "gruen" "rosa"  "blau"  "blau"  "rosa" "gruen" "gruen"
```

Beachte die Bedeutung leerer eckiger Klammern (`[]`): Sie dienen zur Indizierung *aller* Vektorelemente gleichzeitig, im Gegensatz zur Wirkung einer Zuweisung ganz ohne eckige Klammern, die ein Überschreiben des ganzen Objektes zur Folge hat:

```
> x[] <- 99;   x
[1] 99 99 99 99 99
> x <- 2;   x
[1] 2
```

Frage: Welche Wirkung hat `x[ x < 0] <- -x[ x < 0]` auf einen `numeric`-Vektor `x`?

2.7 Faktoren und geordnete Faktoren: Definition und Verwendung

In **R** ist es möglich, nominalskalierte und ordinalskalierte Daten zu charakterisieren. Nominale Daten sind Werte aus einer Menge von sogenannten "Levels" *ohne* Ordnungsrelation, wie sie beispielsweise bei einem Merkmal wie dem Geschlecht, der Blutgruppe, der Automarke oder dem Beruf auftreten. Solche Merkmale werden in **R** als „Faktoren" bezeichnet. Bei ordinalen Daten sind die Levels *mit* einer Ordnungsrelation versehen, wie z. B. bei den Merkmalen Note, Ausbildungsabschluss, soziale Schicht, Altersklasse etc. Diese Merkmale werden **R** „geordnete Faktoren" genannt. In beiden Fällen sind nur endliche Mengen von Levels zugelassen. Der Sinn dieses Konzepts in **R** liegt im Wesentlichen in der adäquaten Interpretation und Behandlung derartiger Variablen in statistischen Modellen und Funktionen.

Die k möglichen Levels eines (geordneten oder ungeordneten) Faktors werden in **R** durch die natürlichen Zahlen von 1 bis k codiert, sind aber mit Zeichenketten assoziierbar, sodass die Bedeutung der Levels (für den Menschen) besser erkennbar bleibt. Die Daten werden in **R** als `numeric`-Vektoren von Levelcodes gespeichert. Diese Vektoren besitzen zwei Attribute: Das Attribut `levels`, das die Menge der k Zeichenketten enthält, welche alle *zulässigen* Levels des Faktor beschreiben, und das Attribut `class`.

Im Fall eines (ungeordneten) Faktors hat das `class`-Attribut den Wert `"factor"` und macht dadurch kenntlich, dass es sich um einen Vektor mit Werten eines (ungeordneten) Faktors handelt. Das `class`-Attribut eines geordneten Faktors hingegen hat den Wert `c( "ordered", "factor")` und die *Reihenfolge* der Levels im `levels`-Attribut spiegelt die jeweilige Ordnungsrelation der Levels wider. Wir wollen solche Vektoren fortan kurz `factor`- bzw. `ordered`-Vektoren nennen. Beachte, dass der Modus eines `factor`- oder `ordered`-Vektors `numeric` ist!

Anhand von Beispieldaten sollen die Erzeugung und die Arbeit mit Vektoren der beiden Klassen (Faktor bzw. geordneter Faktor) erläutert werden. Angenommen, wir haben die folgenden Vektoren:

```
> alter
[1] 35 39 53 14 26 68 40 56 68 52 19 23 27 67 43
> geschlecht
[1] "m" "m" "w" "w" "m" "w" "w" "m" "m" "w" "m" "m" "w" "w" "w"
```

```
> blutgruppe
[1] "A" "B" "B" "0" "A" "AB" "0" "AB" "B" "AB" "A" "A" "AB" "0" "B"
> gewicht
[1] 82 78 57 43 65 66 55 58 91 72 82 83 56 51 61
> groesse
[1] 181 179 153 132 166 155 168 158 188 176 189 179 167 158 174
> rauchend
[1] "L" "G" "X" "S" "G" "G" "X" "L" "S" "X" "X" "L" "X" "X" "S"
```

Die Vektoren `alter`, `gewicht` und `groesse` enthalten metrische Daten und haben den Modus `numeric`. Die Daten in den `character`-Vektoren `geschlecht` und `blutgruppe` sind in unserer Interpretation von nominalem Typ, während hinter den Daten in `rauchend` eine Ordnung steckt, es sich also um ordinale Daten handelt („X“ ≙ NichtraucherIn, „G“ ≙ GelegenheitsraucherIn, „L“ ≙ LeichteR RaucherIn, „S“ ≙ StarkeR RaucherIn).

2.7.1 Erzeugung von Faktoren (`factor()`, `gl()`) und Levelabfrage (`levels()`)

Wir wollen die obigen nominalen Datenvektoren des Modus' `character` in solche der Klasse `factor` umwandeln und den ordinalen Datenvektor in einen der Klasse `ordered`. Das Alter der Personen (in `alter`) wollen wir in vier Intervalle gruppieren und diese Gruppierung dann in einem `ordered`-Vektor speichern:

`> (ge <- factor( geschlecht))`
`[1] m m w w m w w m m w m m w w w`
`Levels: m w`
`> (blut <- factor( blutgruppe))`
`[1] A B B 0 A AB 0 AB ....`
`Levels: 0 A AB B`
	Erzeugung von (ungeordneten) Faktoren aus den `character`-Vektoren `geschlecht` und `blutgruppe`. Die Ausgabe von Faktoren erfolgt ohne Hochkommata, um zu zeigen, dass es keine `character`-Vektoren sind. Außerdem werden die Levels dokumentiert.

`> levels( ge)`
`[1] "m" "w"`
`> levels( blut)`
`[1] "0" "A" "AB" "B"`
	Die Menge der Levels ist per Voreinstellung automatisch alphanumerisch sortiert worden. Dies impliziert *per se* noch keine Ordnung, muss aber beachtet werden! (Siehe unten bei der Vergabe von Levelnamen durch das Argument `labels` und bei der Erzeugung geordneter Faktoren.)

`> blut2 <- factor( blutgruppe,`
`+ levels = c( "A", "B", "AB", "0"))`
`> levels( blut2)`
`[1] "A" "B" "AB" "0"`
	Es ist bei der Faktorerzeugung möglich, eine Levelordnung vorzugeben, indem das Argument `levels` verwendet wird. Für jeden Wert im Ausgangsvektor (hier `blutgruppe`), der *nicht* im `levels`-Argument auftaucht, würde `NA` vergeben.

`> (ge2 <- factor( geschlecht,`
`+ levels = c( "m", "w"),`
`+ labels = c( "Mann", "Frau")))`
`[1] Mann Mann Frau Frau Mann ....`
`Levels: Mann Frau`
	Das Argument `labels` erlaubt die freie Umbenennung der Faktorlevels gleichzeitig mit der Faktorerzeugung.

Bemerkung: Eine sehr nützliche, da effiziente Funktion zur Erzeugung von Faktorvektoren, in deren Elementen die Faktorlevels in gewissen Mustern aufeinanderfolgen, ist `gl()`. Für Details verweisen wir auf ihre Online-Hilfe und auf die Ausgabe von `example( gl)`.

2.7.2 Änderung der Levelsortierung (`relevel()` & (`reorder()`), Zusammenfassung von Levels (`levels()`), Löschen unnötiger Levels (`droplevels()`)

`> (blut <- factor( blut, levels =`
`+ c( "A", "B", "AB", "0"))`
`[1] A  B  B  0  A  AB 0  AB ....`
`Levels: A B AB 0`
	Die Levelsortierung kann geändert werden, indem der Faktor quasi neu erzeugt wird und dabei dem `levels`-Argument die Levels in der gewünschten Reihenfolge übergeben werden.

`> relevel( blut, ref = "0")`
`[1] A  B  B  0  A  AB 0  AB ....`
`Levels: 0 A B AB`
	Soll nur eines der Levels zum „ersten“ gemacht und die anderen „nach hinten verschoben“ werden, reicht es, `relevel()` das neue „Referenzlevels“ über sein Argument `ref` anzugeben.

`> reorder( ....)`	Sind Faktorlevels in Abhängigkeit von Werten einer anderen (i. d. R. `numeric`-)Variablen zu sortieren, kann `reorder()` die Lösung sein; siehe Online-Hilfe.

`> levels( blut2) <- c( "non0",`
`+ "non0", "non0", "0")`
`> blut2`
`[1] non0 non0 non0  0 ....`
`Levels: non0 0`
	Levels werden zusammengefasst (und auch umbenannt) durch Zuweisung eines entsprechenden `character`-Vektors passender Länge an das `levels`-Attribut. Die Zuordnung der alten Levels zu den neuen geschieht über ihre Elementepositionen.

`> blut[ 1:5]  # = head( blut, 5)`
`[1] A B B 0 A`
`Levels: A B AB 0`
	Die Levels eines Faktors sind gegenüber (jeglicher Form von) Indizierung invariant, d. h., nicht mehr im Faktorvektor auftretende Levels werden nicht gelöscht, ...

`> droplevels( blut[ 1:5])`
`[1] A B B 0 A`
`Levels: A B 0`
`> blut[ 1:5, drop = TRUE]`
`[1] A B B 0 A`
`Levels: A B 0`
	sondern müssen, sofern dies gewünscht oder nötig ist, entweder explizit mit `droplevels()` oder durch Verwendung der Indizierungsoption `drop = TRUE` entfernt werden.

2.7.3 Erzeugung von geordneten Faktoren: `ordered()`, `gl()`

`> (rauch <- ordered( rauchend))`
`[1] L G X S G G X L S X X L X X S`
`Levels: G < L < S < X`
	Erzeugung eines geordneten Faktors aus dem `character`-Vektor `rauchend`. Dabei wird für die Levelordnung die alphabetische Levelsortierung verwendet.

`> (rauch <- ordered( rauchend,`
`+ levels = c( "X", "G", "L", "S")))`
`[1] L G X S G G X L S X X L X X S`
`Levels: X < G < L < S`
	Die Vorgabe einer Levelordnung bei Erzeugung des geordneten Faktors geschieht durch das Argument `levels`. Dabei bestimmt die Levelreihenfolge die Ordnung. Jeder Wert im Ausgangsvektor, der nicht im `levels`-Argument auftaucht, liefert `NA`.

`> (rauch2 <- ordered( rauchend,`
`+ levels = c( "X", "G", "L", "S"),`
`+ labels = c( "NR", "gel.", "leicht",`
`+ "stark")))`
`[1] leicht gel.  NR  stark ....`
`Levels: NR < gel. < leicht < stark`
	Die direkte Erzeugung eines geordneten Faktors mit vorgegebener Levelordnung *und* freier Namensgebung für die Levels ist durch die kombinierte Verwendung der Argumente `levels` und `labels` möglich.

`> ordered( blut)`
`[1] B  O  O  A  B  AB A  AB O ....`
`Levels: A < B < AB < O`
	Die Anwendung von `ordered()` auf einen (ungeordneten) `factor`-Vektor liefert einen geordneten Faktor, dessen Levelordnung und -bezeichnungen vom `factor`-Vektor übernommen werden. (Hier ein unsinniges Beispiel.)

Bemerkung: Die in §2.7.1 bereits erwähnte Funktion `gl()` besitzt ein logisches Argument namens `ordered`, das die einfache Generierung geordneter Faktorvektoren mit Levelmustern erlaubt (siehe ihre Online-Hilfe).

2.7.4 Änderung der Levelordnung, Zusammenfassung von Levels, Löschen unnötiger Levels bei geordneten Faktoren

`> (rauch <- ordered( rauch,`
`+ levels = c( "S", "L", "G", "X")))`
`[1] L G X S G G X L S X X L X X S`
`Levels: S < L < G < X`
	Die Levelordnung kann geändert werden, indem der geordnete Faktor mit der gewünschten Levels-Reihenfolge mittels `ordered()` erneut erzeugt wird. (Beachte: `relevel()` funktioniert bei einem geordneten Faktor nicht!)

`> levels( rauch2) <- c( "NR", "R",`
`+ "R", "R");  rauch2`
`[1] R  R  NR R  R  R  NR R  R ....`
`Levels: NR < R`
	Das Zusammenfassen (und Umbenennen) von Levels geschieht wie bei ungeordneten Faktoren (siehe §2.7.2). Hierbei bleibt die Eigenschaft, ein geordneter Faktor zu sein, erhalten!

`> tail( rauch, 7)`
`[1] S X X L X X S`
`Levels: X < G < L < S`
`> droplevels( tail( rauch, 7))`
`[1] S X X L X X S`
`Levels: X < L < S`
	Für das Entfernen von nicht mehr im Faktorvektor auftretenden Levels gilt dasselbe wie bei ungeordneten Faktoren (siehe §2.7.2), wobei die Eigenschaft, ein geordneter Faktor zu sein, erhalten bleibt.

Als beispielhafte Bestätigung dessen, was am Anfang von Abschnitt 2.7 gesagt wurde:

```
> mode( blut); class( blut);
+ attributes( blut)
[1] "numeric"
[1] "factor"

$levels
[1] "A"  "B"  "AB" "O"
$class
[1] "factor"
```

```
> mode( rauch); class( rauch);
+ attributes( rauch)
[1] "numeric"
[1] "ordered" "factor"

$levels
[1] "S" "L" "G" "X"
$class
[1] "ordered" "factor"
```

2.7.5 Klassierung numerischer Werte und Erzeugung geordneter Faktoren: `cut()`

`> (AKlasse <- cut( alter, breaks`
`+ = c( 0, 25, 45, 60, Inf)))`
`[1] (25,45]  (25,45]  (45,60]`
`[4] (0,25]  (25,45]  (60,Inf]`
`[7] (25,45]  (45,60]  (60,Inf]`
`[10] (45,60]  (0,25]  (0,25]`
`[13] (25,45]  (60,Inf] (25,45]`
`Levels:`
`(0,25] (25,45] (45,60] (60,Inf]`
	Aus dem `numeric`-Vektor `alter` wird durch die Funktion `cut()` ein `factor`-Vektor (hier `AKlasse`) erzeugt, indem sie gemäß der im Argument `breaks` angegebenen Intervallgrenzen (per Voreinstellung) links offene und rechts abgeschlossene Klassen $(b_i, b_{i+1}]$ bildet und daraus (per Voreinstellung) entsprechende Faktorlevels konstruiert (`Inf` $= +\infty$). Jedes Element des Ausgangsvektors wird im Faktor durch das Intervall/Level codiert, in das es fällt. (Levels können durch das Argument `labels` beliebig benannt werden.)

`> (AKlasse <- ordered( AKlasse))`
`[1] (25,45] (25,45] (45,60] ....`
`Levels:`
`(0,25] < (25,45](45,60] < (60,Inf]`
	Die Anwendung von `ordered()` auf den resultierten `factor`-Vektor liefert den geordneten Faktor, dessen Levelordnung und -bezeichnungen aus dem `factor`-Vektor übernommen werden.

2.7.6 Tabellierung von Faktoren und Faktorkombinationen: `table()`

```
> table( AKlasse)
AKlasse
 (0,25]  (25,45]  (45,60] (60,Inf]
      3        6        3        3

> table( ge, AKlasse)
   AKlasse
ge  (0,25] (25,45] (45,60] (60,Inf]
  m      2       3       1        1
  w      1       3       2        2
> table( AKlasse, ge, rauch)
,,rauch = S   ,,rauch = L   ,,rauch = G   ,,rauch = X
         ge            ge            ge            ge
AKlasse  m w  AKlasse  m w  AKlasse  m w  AKlasse  m w
(0,25]   0 1  (0,25]   1 0  (0,25]   0 0  (0,25]   1 0
(25,45]  0 1  (25,45]  1 0  (25,45]  2 0  (25,45]  0 2
(45,60]  0 0  (45,60]  1 0  (45,60]  0 0  (45,60]  0 2
(60,Inf] 1 0  (60,Inf] 0 0  (60,Inf] 0 1  (60,Inf] 0 1
```

`table()` erstellt ein `table`-Objekt mit der Tabelle der absoluten Häufigkeiten jedes Levels in `AKlasse`.

Für zwei Argumente wird die Kontingenztafel aller Levelkombinationen erstellt.

Für mehr als zwei Argumente besteht die Tabellierung aller Levelkombinationen aus mehreren Kontingenztafeln und ist etwas unübersichtlicher.

Hinweise:

- Siehe auch die Online-Hilfe zu `prop.table()` und `addmargins()`, mit deren Hilfe eine bestehende Tabelle absoluter Häufigkeiten in eine solche mit relativen Häufigkeiten überführt werden kann bzw. um nützliche Marginalien (= „Ränder“) ergänzt werden kann.
- Beachte, dass `NA`s und `NaN`s – per Voreinstellung – durch `table( x)` *nicht* mittabelliert werden! Dazu müssen `NA` und `NaN` im zu tabellierenden Faktor `x` durch etwas wie `table( factor( x, exclude = NULL))` explizit zu einem Level gemacht oder `table( x, useNA = "ifany")` oder `table( x, useNA = "always")` verwendet werden (siehe die Online-Hilfe). `summary( x)` kann auch verwendet werden, liefert aber – per Voreinstellung – möglicherweise nicht die gesamte Tabelle.

2.7.7 Aufteilung gemäß Faktor(en)gruppen und faktor(en)gruppierte Funktionsanwendungen: `split()`, `tapply()` & `ave()`

```
> split( gewicht, AKlasse)
$`(0,25]`
[1] 43 82 83

$`(25,45]`
[1] 82 78 65 55 56 61

$`(45,60]`
[1] 57 58 72

$`(60,Inf]`
[1] 66 91 51
```

Ist `g` ein Faktorvektor derselben Länge des Vektors `x`, so teilt `split( x, g)` die Elemente von `x` in Gruppen ein, die durch wertgleiche Elemente in `g` definiert sind. Rückgabewert von `split()` ist eine Liste (siehe hierzu Abschnitt 2.9, Seite 46) von Vektoren der gruppierten `x`-Elemente. Die Komponenten der Liste sind benannt durch die Levels des gruppierenden Faktors. Hier geschieht eine Gruppierung der `gewicht`-Elemente gemäß der durch `AKlasse` indizierten Altersgruppen.

`> split( gewicht, list( ge, AKlasse))`
`$‘m.(0,25]‘`
`[1] 82 83`
`$‘m.(45,60]‘`
`[1] 58`
`$‘w.(0,25]‘`
`[1] 43`
`$‘w.(45,60]‘`
`[1] 57 72`
`$‘m.(25,45]‘`
`[1] 82 78 65`
`$‘m.(60,Inf]‘`
`[1] 91`
`$‘w.(25,45]‘`
`[1] 55 56 61`
`$‘w.(60,Inf]‘`
`[1] 66 51`
	Ist `g` eine Liste von Faktorvektoren (alle derselben Länge von `x`), werden die Elemente von `x` in Gruppen eingeteilt, die durch die Levelkombinationen der Faktoren, die in `g` zusammengefasst sind, definiert werden. Rückgabewert ist eine Liste von Vektoren der gruppierten `x`-Elemente, deren Komponenten durch die aufgetretenen Levelkombinationen der gruppierenden Faktoren (getrennt durch einen Punkt „`.`“) benannt sind.

Die Funktion `tapply()` ist eine Erweiterung von `split()`, indem sie die Anwendung einer im Prinzip frei wählbaren Funktion auf die gruppierten Elemente ermöglicht: Falls `g` ein Faktor derselben Länge des Vektors `x` ist, wendet `tapply( x, g, FUN)` die Funktion `FUN` auf Gruppen der Elemente von `x` an, wobei diese Gruppen durch gleiche Elemente von `g` definiert sind.

Ist `g` eine Liste (vgl. Abschnitt 2.9, Seite 46) von Faktoren (alle derselben Länge von `x`), wird `FUN` auf Gruppen der Elemente von `x` angewendet, die durch die Levelkombinationen der Faktoren, die in `g` genannt sind, definiert werden:

`> tapply( gewicht, AKlasse, mean)`
`  (0,25]  (25,45]  (45,60] (60,Inf]`
`69.33333 66.16667 62.33333 69.33333`
`> tapply( gewicht, AKlasse, sd)`
`  (0,25]  (25,45]  (45,60] (60,Inf]`
`22.81082 11.37395 8.386497 20.20726`
	Faktorgruppierte Anwendung der Funktion `mean()` zur Bestimmung der mittleren Gewichte für jede durch `AKlasse` indizierte Altersgruppe. Dito mittels `sd()` zur Berechnung der empirischen Standardabweichungen.

`> tapply( gewicht, list( ge, AKlasse),`
`+ mean)`
`  (0,25]  (25,45] (45,60] (60,Inf]`
`m  82.5 75.00000  58.0  91.0`
`w  43.0 57.33333  64.5  58.5`
	Faktorengruppierte Anwendung von `mean()` auf die durch jede Levelkombination von `ge` mit `AKlasse` indizierten Einträge in `gewicht`. (`list()` wird in Abschnitt 2.9, Seite 46 erläutert.)

Ein Funktionsaufruf der Art `ave( x, g1, ...., gk)` für einen `numeric`-Vektor `x` und Faktor-Vektoren `g1` bis `gk` macht etwas ähnliches wie `tapply()`, ist aber per Voreinstellung auf die Berechnung von Mittelwerten für Gruppen von `x`-Elementen eingestellt, wobei diese Gruppen durch gleiche Kombinationen von Elementen von `g1` bis `gk` definiert sind. Außerdem ist das Ergebnis von `ave( x, g1, ...., gk)` ein Vektor derselben Länge wie `x`, dessen Elemente, die zur selben Gruppe gehören, alle auch denselben Wert, nämlich den betreffenden Gruppenmittelwert haben. Beispiele:

```
> ave( gewicht, AKlasse)
 [1] 66.16667 66.16667 62.33333 69.33333 66.16667 69.33333 66.16667 62.33333
 [9] 69.33333 62.33333 69.33333 69.33333 66.16667 69.33333 66.16667

> ave( gewicht, ge, AKlasse)
 [1] 75.00000 75.00000 64.50000 43.00000 75.00000 58.50000 57.33333 58.00000
 [9] 91.00000 64.50000 82.50000 82.50000 57.33333 58.50000 57.33333
```

2.8 Matrizen: Erzeugung, Indizierung, Modifikation und Operationen

In **R** stehen mehrdimensional indizierte Felder (Englisch: "arrays") zur Verfügung, für die wir den Terminus „Array" übernehmen. Dies sind Vektoren mit einem Attribut `dim` (und evtl. zusätzlich mit einem Attribut `dimnames`); sie bilden die Klasse `array`. Ein Spezialfall hierin sind wiederum die *zwei*dimensionalen Arrays, genannt Matrizen; diese haben die Klasse `matrix`. Sowohl Arrays als auch Matrizen sind intern als Vektoren gespeichert und unterscheiden sich von „echten" Vektoren nur durch die zwei schon genannten Attribute `dim` und `dimnames`.

2.8.1 Grundlegendes zu Arrays

Arrays werden mit der Funktion `array()` erzeugt. Sie benötigt als Argumente einen Datenvektor, dessen Elemente in das Array eingetragen werden sollen, sowie einen Dimensionsvektor, der das `dim`-Attribut wird und dessen Länge k die Dimension des Arrays bestimmt. Die Elemente des Dimensionsvektors sind positive `integer`-Werte und die Obergrenzen eines jeden der k Indizes, mit denen die Elemente des Arrays indiziert werden. Es sei zum Beispiel `z` ein Vektor mit 1500 Elementen. Dann erzeugt die Anweisung

```
> a <- array( z, c( 3, 5, 100))
```

ein offenbar dreidimensionales $(3 \times 5 \times 100)$-Array, dessen Elemente die Elemente von `z` (in einer gewissen Reihenfolge) sind und dessen Element a_{ijk} mit `a[i, j, k]` indiziert wird. Dabei muss $i \in \{1, 2, 3\}, j \in \{1, 2, 3, 4, 5\}$ und $k \in \{1, \ldots, 100\}$ sein.

Die Elemente des Datenvektors `z` werden in das Array `a` eingetragen, indem sukzessive die Elemente `a[i, j, k]` gefüllt werden, wobei der erste Index (`i`) seinen Wertebereich am schnellsten und der letzte Index (`k`) seinen Wertebereich am langsamsten durchläuft. D. h., für das dreidimensionale Array `a` sind die Elemente `z[1], ..., z[1500]` von `z` sukzessive in die Elemente `a[1,1,1]`, `a[2,1,1]`, `a[3,1,1]`, `a[1,2,1]`, `a[2,2,1]`, ..., `a[2,5,100]` und `a[3,5,100]` „eingelaufen".

Unsere häufigsten Anwendungen werden jedoch nicht mehrdimensionale Arrays benötigen, sondern Matrizen, auf die wir uns i. F. konzentrieren werden. Nichtsdestotrotz sind Arrays kennenswerte, da sehr leistungsfähige Datenstrukturen; `?array` führt zu weiteren Informationen.

2.8.2 Erzeugung von Matrizen: `matrix()`

Eine Matrix wird mit der Funktion `matrix()` erzeugt. (Dies ist auch als zweidimensionales Array mit `array()` möglich, aber `matrix()` ist „maßgeschneidert" für Matrizen.) Sie erwartet als erstes Argument einen Datenvektor, dessen Elemente *spaltenweise* in die Matrix eingetragen werden, und in mindestens einem weiteren Argument eine Angabe, wie viele Zeilen (bzw. Spalten) die Matrix haben soll. Folgende Beispiele sollen die Funktionsweise von `matrix()` erläutern:

```
> z <- c( 130, 26, 110, 24, 118, 25, 112, 25)
> (Werte <- matrix( z, nrow = 4))
     [,1] [,2]
[1,]  130  118
[2,]   26   25
[3,]  110  112
[4,]   24   25
```

Hier wird aus dem Datenvektor `z` eine Matrix mit `nrow = 4` Zeilen (Englisch: "rows") erzeugt. Die Spaltenzahl wird automatisch aus der Länge des Datenvektors ermittelt. Der Datenvektor füllt die Matrix dabei spaltenweise auf. (Dies ist die Voreinstellung.)

Ist die Länge des Datenvektors kein ganzzahliges Vielfaches der für `nrow` angegebenen Zeilenzahl, werden (höchstens) so viele Spalten angelegt, bis der Datenvektor in der Matrix vollständig enthalten ist. Die dann noch leeren Elemente der Matrix werden durch zyklische Wiederholung

der Datenvektorelemente aufgefüllt (und es wird dann eine Warnung ausgegeben). In folgendem Beispiel wird der achtelementige Datenvektor `z` in eine dreizeilige Matrix eingetragen, was durch eine Warnung quittiert wird:

```
> matrix( z, nrow = 3)
     [,1] [,2] [,3]
[1,]  130   24  112
[2,]   26  118   25
[3,]  110   25  130
Warning message:
data length [8] is not a sub-multiple or multiple of the number of rows
[3] in matrix
```

Die hier und in den nächsten Abschnitten folgenden Tabellen enthalten Anweisungen und Operationen, die zur Erzeugung, Spalten- und Zeilenbenennung, Indizierung, Erweiterung von und Rechnung mit Matrizen zur Verfügung stehen.

`> Werte <- matrix( z, nrow = 4)`	(Siehe obiges Beispiel.)

`> matrix( z, ncol = 4)`
`     [,1] [,2] [,3] [,4]`
`[1,]  130  110  118  112`
`[2,]   26   24   25   25`
	Der Datenvektor `z` wird in eine Matrix mit `ncol = 4` Spalten ("columns") geschrieben. Die Zeilenzahl wird automatisch aus der Länge von `z` ermittelt.

`> matrix( z, nrow = 4, ncol = 5)`
`  ....`
`> (Werte <- matrix( z, ncol = 2,`
`+ byrow = TRUE))`
`     [,1] [,2]`
`[1,]  130   26`
`[2,]  110   24`
`[3,]  118   25`
`[4,]  112   25`
	`ncol` und `nrow` können gleichzeitig angegeben werden; `z` wird dann zyklisch verwendet, wenn zu kurz, und abgeschnitten, wenn zu lang. Mit dem Argument `byrow = TRUE` wird der Datenvektor *zeilenweise* in die Matrix eingelesen. Voreinstellung ist `byrow = FALSE`, also spaltenweises Einlesen. (Ohne jegliche Angabe von `ncol` und `nrow` wird eine einspaltige Matrix erzeugt; ohne Angabe eines Datenvektors eine Matrix mit `NA`-Einträgen.)

2.8.3 Be- & „Entnennung" von Spalten und Zeilen: `dimnames()`, `colnames()`, `rownames()`, `unname()`

`> dimnames( Werte) <- list( c( "Alice",`
`+ "Bob", "Carol", "Deborah"),`
`+ c( "Gewicht", "Alter"));  Werte`
`        Gewicht Alter`
`  Alice     130    26`
`    Bob     110    24`
`  Carol     118    25`
`Deborah     112    25`
	Mit der Funktion `dimnames()` werden den Zeilen einer Matrix die Namen zugewiesen, die sich im ersten `character`-Vektor der Liste (dazu mehr in Abschnitt 2.9, Seite 46) auf der rechten Seite befinden. Die Spalten bekommen die Namen, die im zweiten Vektor sind.

`> Werte <- matrix( z, ncol = 2, byrow = TRUE,`
`+ dimnames = list( c( "Alice", "Bob",`
`+ "Carol", "Deborah"), c( "Gewicht",`
`+ "Alter")))`
	Eine Zeilen- und Spaltenbenennung kann auch schon bei der Erstellung der Matrix über das Argument `dimnames` erfolgen.

`> dimnames( Werte)`
`[[1]]`
`[1] "Alice"  "Bob"  "Carol"  "Deborah"`
 	
`[[2]]`
`[1] "Gewicht" "Alter"`
	Der Zugriff auf die Dimensionsnamen durch `dimnames()` liefert eine Liste, deren zwei Komponenten die Vektoren der Zeilen- bzw. Spaltennamen enthalten, falls vorhanden; ansonsten jeweils das NULL-Objekt.

`> dimnames( Werte) <- list( NULL,`
`+ c( "Gewicht", "Alter"));  Werte`
`  Gewicht Alter`
`[1,]  130  26`
`[2,]  110  24`
`[3,]  118  25`
`[4,]  112  25`
	Die Zuweisung des NULL-Objekts an Stelle eines Zeilennamenvektors führt zur Löschung vorhandener Zeilennamen. An der Stelle des zweiten Vektors würde es zur Löschung der Spaltennamen führen.

`> colnames( Werte)`
`....`
`> rownames( Werte) <- c( "A", "B",`
`+ "C", "D")`
	Zugriff auf oder Zuweisung an Spalten- bzw. Zeilennamen allein ist hiermit kompakter möglich.

`> unname( Werte)`
`  [,1] [,2]`
`[1,]  130  26`
`[2,]  110  24`
`[3,]  118  25`
`[4,]  112  25`
	Spalten- und Zeilennamen einer Matrix können mit `unnname()` entfernt werden (z. B. wenn sie störend zu lang sind). Hier geschieht dies nur „temporär“ für die Ausgabe; damit sie permanent entfernt würden, wäre das Ergebnis von `unname( Werte)` natürlich an `Werte` zuzuweisen.

Beachte: Bei der Verwendung von Dimensionsnamen für Matrizen entsteht ein erhöhter Speicherplatzbedarf, der bei großen Matrizen erheblich sein kann und insbesondere bei aufwändigeren Berechnungen (vor allem in Simulationen) die Geschwindigkeit von `R` negativ beeinflusst. Daher kann es sinnvoll sein, solche Dimensionsnamen vorher zu entfernen. (Siehe hierzu auch die Bemerkung zum Speicherplatzbedarf von Elementenamen bei Vektoren in §2.5.2.)

2.8.4 Erweiterung um Spalten oder Zeilen: `cbind()`, `rbind()`

`> groesse <- c( 140, 155, 142, 175)`
`> (Werte <- cbind( Werte, groesse))`
`  Gewicht Alter groesse`
`A  130  26  140`
`B  110  24  155`
`C  118  25  142`
`D  112  25  175`
	Eine bestehende Matrix lässt sich um zusätzliche Spalten und Zeilen erweitern, indem an sie ein passender Vektor „angebunden“ wird. `cbind()` erledigt dies spaltenweise, `rbind()` zeilenweise. Die Länge des Vektors muss – je nach Art des „Anbindens“ – mit der Zeilen- bzw. Spaltenzahl der Matrix übereinstimmen.

`> (Werte <- rbind( Werte,`
`+ E = c( 128, 26, 170)))`
`  Gewicht Alter groesse`
`A  130  26  140`
`B  110  24  155`
`C  118  25  142`
`D  112  25  175`
`E  128  26  170`
	Ist der anzubindende Vektor in einem Objekt gespeichert, so wird dessen Name als Spalten- bzw. Zeilenname übernommen. Ansonsten kann durch *name* = *wert* ein Spalten- bzw. Zeilenname angegeben werden.
	Matrizen mit gleicher Zeilenzahl werden durch `cbind()` spaltenweise aneinandergehängt, solche mit gleicher Spaltenzahl mit `rbind()` zeilenweise.

2.8.5 Matrixdimensionen und Indizierung von Elementen: `dim()`, `[]`, `head()` & `tail()`

`> dim( Werte)`
`[1] 5 3`
`> nrow( Werte); ncol( Werte)`
`[1] 5`
`[1] 3`
	`dim()` liefert die Zeilen- und Spaltenzahl zusammen als zweielementigen Vektor.
	`nrow()` bzw. `ncol()` liefern sie einzeln. (`Werte` stammt aus §2.8.4.)

Die Indizierung von Matrizen funktioniert analog zu der von Vektoren (vgl. §2.6.1):

`> Werte[ 3,2]` `[1] 25`	Matrixelemente werden durch Doppelindizes indiziert: `[i,j]` liefert das Element in der i-ten Zeile und j-ten Spalte. Ein Spalten- oder Zeilenname wird nicht mitgeliefert.

`> Werte[ 2,]`
`Gewicht Alter groesse`
`  110  24  155`
`> Werte[ ,1]`
`  A  B  C  D  E`
`130 110 118 112 128`
	Spalten (bzw. Zeilen) werden als Ganzes ausgelesen, wenn der jeweils andere Index unspezifiziert bleibt. Ein Komma muss verwendet werden, um die gewünschte Dimension zu spezifizieren. Das Resultat ist (per Voreinstellung) stets ein Vektor und Namen werden übernommen.

`> Werte[ c( 1,2), 2]`
`  A  B`
`26 24`
`> Werte[, c( 1,3)]`
`  Gewicht groesse`
`A  130  140`
`B  110  155`
`C  118  142`
`D  112  175`
`E  128  170`
	Durch Angabe eines Index*vektors* für den Spalten- bzw. Zeilenindex erhält man die angegebenen Spalten bzw. Zeilen.
	Das Resultat ist ein Vektor, wenn einer der Indizes eine einzelne Zahl ist; es ist eine Matrix, wenn beide Angaben Indexvektoren sind. Namen werden übernommen.

`> Werte[ -2, -3]`
`  Gewicht Alter`
`A  130  26`
`C  118  25`
`D  112  25`
`E  128  26`
	Negative Integer-Indizes wirken hier genauso wie bei Vektoren, nämlich ausschließend.

`> Werte[ c( F,F,T,F,F),]`
`Gewicht  Alter groesse`
`  118  25  142`
`> Werte[ c( T,T,F,F,F), c( T,F,T)]`
`  Gewicht groesse`
`A  130  140`
`B  110  155`
	Die Auswahl von Elementen und ganzen Spalten sowie Zeilen kann auch durch logische Indexvektoren geschehen. Sie wirken analog wie bei Vektoren.

`> Werte[ c( F,T), c(T,F,T)]`
`  Gewicht groesse`
`B  110  155`
`D  112  175`
	Sind logische Indexvektoren nicht lang genug, werden sie zyklisch repliziert.

`> Werte[ 1, "Gewicht"]`
`  Gewicht`
`  130`
`> Werte[, c( "Gewicht", "Alter")]`
`  Gewicht Alter`
`A  130  26`
`B  110  24`
`C  118  25`
`D  112  25`
`E  128  26`
	Die Auswahl von Elementen und ganzen Spalten sowie Zeilen kann, wenn eine Benennung vorliegt, auch mit den Namen der Spalten und Zeilen geschehen.

`> Werte[ Werte[, "groesse"] > 160,`
`+ "Gewicht"]`
`  D  E`
`112 128`
	Verschiedene Indizierungsmethoden sind kombinierbar.

`> head( Werte, n = 2)`
`  Gewicht Alter groesse`
`A  130  26  140`
`B  110  24  155`
 	
`> tail( Werte, n = 2)`
`  Gewicht Alter groesse`
`D  112  25  175`
`E  128  26  170`
	Analog zur Anwendung auf Vektoren (vgl. §2.6.2) liefern `head()` und `tail()` die oberen bzw. unteren `n` *Zeilen* einer Matrix. Voreinstellung für `n` ist 6.
	Negative Werte für `n` liefern den "head" bis auf die letzten bzw. den "tail" bis auf die ersten |`n`| Zeilen der Matrix.

2.8.6 Einige spezielle Matrizen: `diag()`, `col()` & `row()`, `lower.tri()` & `upper.tri()`

`> (y <- diag( 1:3))`
`  [,1] [,2] [,3]`
`[1,]  1  0  0`
`[2,]  0  2  0`
`[3,]  0  0  3`
 	
`> diag( y)`
`[1] 1 2 3`
	Erhält `diag()` einen Vektor als Argument, wird eine Diagonalmatrix mit den Elementen dieses Vektors auf der Hauptdiagonalen erzeugt.
	
	Ist `diag()`s Argument eine Matrix, wird ihre Hauptdiagonale als Vektor extrahiert. Die Zuweisungsform `diag( y) <- 4:6` ersetzt die Hauptdiagonale von `y` durch den zugewiesenen Vektor.

`> col( y)`
`  [,1] [,2] [,3]`
`[1,]  1  2  3`
`[2,]  1  2  3`
`[3,]  1  2  3`
`> row( y)`
`  [,1] [,2] [,3]`
`[1,]  1  1  1`
`[2,]  2  2  2`
`[3,]  3  3  3`
	`col()` erwartet eine Matrix als Argument und generiert dazu eine gleich große Matrix, deren Einträge die Spaltenindizes ihrer Elemente sind. `row()` macht entsprechendes mit Zeilenindizes.

`> lower.tri( y)`
`  [,1]  [,2]  [,3]`
`[1,] FALSE FALSE FALSE`
`[2,]  TRUE FALSE FALSE`
`[3,]  TRUE  TRUE FALSE`
 	
`> upper.tri( y, diag = TRUE)`
`  [,1]  [,2] [,3]`
`[1,]  TRUE  TRUE TRUE`
`[2,] FALSE  TRUE TRUE`
`[3,] FALSE FALSE TRUE`
	Ein schönes „Anwendungsbeispiel“ für die beiden obigen Funktionen sind die Implementationen von `lower.tri()` und `upper.tri()` zur Erzeugung von logischen unteren oder oberen Dreiecksmatizen (je nach Wahl des Argumentes `diag` mit oder ohne der Diagonalen):
	
	`> lower.tri`
	`function( x, diag = FALSE) {`
	`  x <- as.matrix(x)`
	`  if(diag)`
	`  row(x) >= col(x)`
	`  else row(x) > col(x)`
	`  }`

2.8.7 Ein paar wichtige Operationen der Matrixalgebra

In der folgenden Auflistung sei mindestens eines der Objekte `A` und `B` eine `numeric`-Matrix und wenn beide Matrizen sind, dann mit Dimensionen, die zu den jeweils betrachteten Operationen „passen“. D. h., `A` oder `B` können in manchen Situationen auch Vektoren oder Skalare sein, die dann von **R** in der Regel automatisch entweder als für die betrachtete Operation geeignete Zeilen- oder Spaltenvektoren interpretiert werden oder (z. B. im Fall von Skalaren) hinreichend oft repliziert werden, damit die Anzahlen der Elemente der an der Operation beteiligten Objekte zueinander passen:

`A + B`, `A - B`, `A * B`, `A / B`, `A^B`, `A %/% B`, `A %% B`	Die elementaren arithmetischen Operationen arbeiten – wie bei Vektoren (siehe §2.3.1) – *elementweise*, falls die Dimensionen von `A` und `B` „zueinanderpassen“. Sind die Dimensionen von `A` und `B` verschieden, werden die Elemente von `B` zyklisch repliziert, falls es möglich und sinnvoll ist (indem die Matrizen als aus ihren Spalten zusammengesetzte Vektoren aufgefasst werden). Beachte, dass der Hinweis auf Seite 21 zur Effizienz beim Potenzieren mit natürlichen Exponenten hier ganz besonders nützlich sein kann, da `A^n` alle Elemente von `A` potenziert.
`t( A)`	A', also die Transponierte der Matrix `A`.
`A %*% B`	Matrixprodukt der Matrizen `A` und `B`.
`crossprod( A, B)`	Kreuzprodukt von `A` mit `B`, d. h. $A'B$, aber i. d. R. etwas schneller als `t( A) %*% B`. Dabei ist `crossprod( A)` dasselbe wie `crossprod( A, A)` und *deutlich* effizienter als die Kombination von `t()` und `%*%`.
`outer( A, B)` *Operatorform:* `A %o% B` *(Dabei ist* `o` *das kleine „O“ und nicht die Null!)*	Äußeres Produkt von `A` mit `B`, d. h., jedes Element von `A` wird mit jedem Element von `B` multipliziert, was $(a_{ij}B)_{ij}$ liefert. Für Spaltenvektoren ist es dasselbe wie `A %*% t( B)` und ergibt dann auch eine Matrix. Für die Verknüpfung der Elemente kann durch das hier nicht gezeigte Argument `FUN` jede binäre Operation angegeben werden; daher heißt `outer()` auch *generalisiertes* äußeres Produkt. Voreinstellung ist `FUN = "*"` (Multiplikation), was für die Operatorform `%o%` die feste Einstellung ist. (Für Beispiele mit anderen binären Operationen siehe Seite 45.)
`solve( A, B)`	Lösung des linearen Gleichungssystems `A %*% X = B`. Beachte: Die Berechnung von $AX^{-1}B$ durch `A %*% solve( X, B)` ist effizienter als durch `A %*% solve( X) %*% B`.
`solve( A)`	Dies liefert die zu `A` inverse Matrix A^{-1}, falls sie existiert.
`chol( A)`	Choleski-Zerlegung von `A`.
`eigen( A)`	Eigenwerte und Eigenvektoren von `A`, d. h. Skalare λ_i und Vektoren v_i mit der Eigenschaft $Av_i = \lambda_i v_i$.
`kappa( A)`	Eine Konditionszahl von `A`.
`qr( A)`	QR-Zerlegung von `A` und „nebenbei“ Bestimmung des Rangs von `A`.
`svd( A)`	Singulärwertzerlegung von `A`.

Bemerkungen: Wichtige Informationen über weitere optionale Argumente, numerische Implementationen, gelegentliche „Verwandte“ etc. der obigen Funktionen sind in ihrer Online-Hilfe zu finden.

Für speziell strukturierte Matrizen, wie z. B. Dreiecksmatrizen, symmetrische, dünn oder dicht besetzte, die typischerweise auch noch – sehr – groß sind, gibt es ein Paket namens `Matrix`, das **R**'s `matrix`-Konzept um Funktionen für den Zugriff auf sehr effiziente Algorithmen erweitert.

2.8.8 Effiziente Berechnung von Zeilen- bzw. Spaltensummen oder -mittelwerten (auch gruppiert): `colSums()` & Verwandte sowie `rowsum()`

`colSums( x)`
`rowSums( x)`
`colMeans( x)`
`rowMeans( x)`
	Spalten- bzw. zeilenweise Summen und arithmetische Mittel für ein `numeric`-Array `x` (also auch eine `numeric`-Matrix). Falls `x` eine Matrix ist, entsprechen die Funktionen (in dieser Reihenfolge) `apply( x, 2, sum)`, `apply( x, 1, sum)`, `apply( x, 2, mean)` bzw. `apply( x, 1, mean)` von §2.8.10, allerdings sind die `apply`-Versionen *erheblich* langsamer. Allen diesen Funktionen steht das Argument `na.rm` zur Verfügung. (In der "base distribution" von **R** scheinen keine derartigen, optimierten Funktionen für die Varianz zu existieren.)

`rowsum( x, g)`	Die Zeilen der `numeric`-Matrix `x` werden für jedes Level des (Faktor-)Vektors `g` in Gruppen zusammengefasst. Dann wird für jede dieser Zeilengruppen die Summe einer jeden Spalte berechnet. Das Ergebnis ist eine Matrix, die so viele Zeilen hat wie verschiedene Werte in `g` sind und so viele Spalten wie `x` hat. (Beachte, dass offenbar zeilengruppierte *Spalten*summen gebildet werden, was mit dem Namen der Funktion etwas schwierig assoziierbar erscheint, insbesondere im Vergleich mit den obigen Funktionen für Zeilen- oder Spaltensummen.)

2.8.9 Spaltenweise Standardabweichungen sowie Kovarianz und Korrelation zwischen Spalten: `sd()` sowie `cov()` und `cor()`

`> sd( Werte)`
`Gewicht  Alter  groesse`
`9.09945 0.83666 15.88395`
	`sd()` liefert für eine Matrix die empirischen Standardabweichungen ihrer Spalten. (Memo: `Werte` stammt aus §2.8.4.)

`> cov( Werte)  # = var( Werte)`
`Gewicht Alter groesse`
`Gewicht  82.8  7.10  -40.30`
`Alter  7.1  0.70  -0.35`
`groesse  -40.3 -0.35  252.30`
 	
`> cor( Werte)`
`Gewicht Alter  groesse`
`Gewicht  1.0000  0.9326 -0.2788`
`Alter  0.9326  1.0000 -0.0263`
`groesse -0.2788 -0.0263  1.0000`
	`cov()` und `var()` bzw. `cor()` liefern für eine Matrix die empirische Kovarianz- bzw. Korrelationsmatrix ihrer Spalten. Zur Behandlung von `NA`s steht beiden das Argument `use` zur Verfügung. Wird ihm der Wert `"complete.obs"` übergeben, werden die Zeilen, in denen ein `NA` auftritt, ganz eliminiert. Der Wert `"pairwise.complete.obs"` erzwingt die maximal mögliche Nutzung aller verfügbaren Elemente pro Variable (= Spalte). Details hierzu stehen in der Online-Hilfe. (Für weitere auf Matrizen anwendbare Funktionen siehe auch „Beachte“ auf S. 23 oben.)

Zur Erinnerung: Die empirische Kovarianz zweier (Daten-)Vektoren $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ mit $n \geq 2$ und $\mathbf{x} \equiv (x_1, \ldots, x_n)'$ ist $\hat{\sigma}(\mathbf{x}, \mathbf{y}) := \frac{1}{n-1} \sum_{r=1}^{n} (x_r - \bar{x}.)\,(y_r - \bar{y}.)$, wobei $\bar{x}. \equiv \frac{1}{n} \sum_{r=1}^{n} x_r$ ist.
Zu $p \geq 2$ Vektoren $\mathbf{x}_1, \ldots, \mathbf{x}_p \in \mathbb{R}^n$, welche oft spaltenweise als Matrix $\mathbf{M} \equiv (\mathbf{x}_1 | \cdots | \mathbf{x}_p)$ zusammengefasst werden, ist ihre empirische Kovarianzmatrix gegeben durch die $(p \times p)$-Matrix $\widehat{\mathrm{Cov}}(\mathbf{M}) := (\hat{\sigma}(\mathbf{x}_i, \mathbf{x}_j))_{1 \leq i,j, \leq p}$. Analog ist die empirische Korrelationsmatrix $\widehat{\mathrm{Cor}}(\mathbf{M})$ definiert.

2.8.10 Zeilen- und spaltenweise Anwendung von (nahezu) beliebigen Operationen: `apply()`, `sweep()` & `scale()`

Das Kommando `apply( matrix, dim, FUN)` wendet die Funktion `FUN` auf die in `dim` spezifizierte Dimension von `matrix` an. (Die Argumente von `apply()` haben eigentlich andere, aber weniger suggestive Namen.)

Die Funktion `scale()` erledigt für eine Matrix wahlweise entweder das spaltenweise Zentrieren (Subtraktion des Spaltenmittelwertes) oder das spaltenweise Standardisieren (Division durch die Spaltenstandardabweichung) oder beides.

`> apply( Werte, 2, mean)`
`Gewicht Alter groesse`
`  119.6  25.2  156.4`
`> apply( Werte, 2, sort)`
`  Gewicht Alter groesse`
`[1,]  110  24  140`
`[2,]  112  25  142`
`[3,]  118  25  155`
`[4,]  128  26  170`
`[5,]  130  26  175`
	Im ersten Beispiel wird die Funktion `mean()` wegen dem Argument `2` auf die zweite Dimension, also die Spalten der Matrix `Werte` (stammt aus §2.8.4) angewendet, was die spaltenweisen Mittelwerte liefert. (Beachte aber §2.8.8!) (Die Zeilen sind die erste Dimension einer Matrix.)
	Im zweiten Beispiel wird jede Spalte von `Werte` sortiert und ...

`> t( apply( Werte, 1, sort))`
`  Alter Gewicht groesse`
`A  26  130  140`
`B  24  110  155`
`C  25  118  142`
`D  25  112  175`
`E  26  128  170`
`> apply( Werte, 2, mean,`
`+ na.rm = TRUE)`
`....`
	...im dritten (zugegebenermaßen ziemlich unsinnigen) Beispiel wegen des Argumentes `1` jede Zeile. Beachte die Verwendung von `t()`. Begründung: Das Resultat von `apply()` ist stets eine Matrix von *Spalten*vektoren. Daher muss das Ergebnis bedarfsweise transponiert werden.
	Benötigt die anzuwendende Funktion `FUN` weitere Argumente, so können diese in benannter Form an `apply()` übergeben werden; sie werden sozusagen „durchgereicht“. Beispiel: Das `na.rm`-Argument der Funktion `mean()`.

`> sweep( Werte, 2, 1:3 * 10)`
`  Gewicht Alter groesse`
`A  120  6  110`
`B  100  4  125`
`C  108  5  112`
`D  102  5  145`
`E  118  6  140`
	Der Befehl `sweep( matrix, dim, vector, FUN = "-")` „kehrt“ (“to sweep” = kehren) den Vektor `vector` elementweise entlang der Dimension `dim` aus der Matrix `matrix` entsprechend der Funktion `FUN` aus. Im Beispiel wird der Vektor $(10, 20, 30)'$ elementweise von jeder Spalte von `Werte` subtrahiert (= Voreinstellung). (Siehe auch obige Bemerkung zu `scale()`.)

2.8.11 Erzeugung spezieller Matrizen mit Hilfe von `outer()`

`> outer( 1:3, 1:5)`
`  [,1] [,2] [,3] [,4] [,5]`
`[1,]  1  2  3  4  5`
`[2,]  2  4  6  8  10`
`[3,]  3  6  9  12  15`
	Das äußere Produkt liefert für zwei Vektoren die Matrix der Produkte aller möglichen paarweisen Kombinationen der Vektorelemente, d. h. das Produkt aus einem Spaltenvektor mit einem Zeilenvektor. (Hier ein Teil des kleinen „1×1“.)

`> outer( 1:3, 1:5, FUN = "+")`
`  [,1] [,2] [,3] [,4] [,5]`
`[1,]  2  3  4  5  6`
`[2,]  3  4  5  6  7`
`[3,]  4  5  6  7  8`
`> outer( 1:3, 1:5, paste, sep = ":")`
`  [,1] [,2] [,3] [,4] [,5]`
`[1,] "1:1" "1:2" "1:3" "1:4" "1:5"`
`[2,] "2:1" "2:2" "2:3" "2:4" "2:5"`
`[3,] "3:1" "3:2" "3:3" "3:4" "3:5"`
	Das generalisierte äußere Produkt erlaubt an Stelle der Multiplikation die Verwendung einer binären Operation oder einer Funktion, die mindestens zwei Vektoren als Argumente verwendet. Im zweiten Beispiel ist dies die Addition und im dritten die Funktion `paste()`, der außerdem noch das optionale Argument `sep = ":"` mit übergeben wird. (Die Ergebnisse sind „selbsterklärend“.)

`> x <- c( -1, 0, -2)`
`> y <- c( -1, 2, 3, -2, 0)`
`> 1 * (outer( x, y, FUN = "<="))`
`  [,1] [,2] [,3] [,4] [,5]`
`[1,]  1  1  1  0  1`
`[2,]  0  1  1  0  1`
`[3,]  1  1  1  1  1`
	Die Matrix der Indikatoren $1_{\{x_i \leq y_j\}}$ für alle $1 \leq i \leq 3$ und $1 \leq j \leq 5$ für die Beispielvektoren `x` und `y`.

2.9 Listen: Konstruktion, Indizierung und Verwendung

Die bisher kennengelernten Datenobjekte Vektor und Matrix (bzw. Array) enthalten jeweils (atomare) Elemente desselben Modus' (`numeric`, `logical` oder `character`). Es kann nötig und sinnvoll sein, auch (nicht-atomare) Elemente unterschiedlicher Modi in einem neuen Objekt – einer „Liste" – zusammenzufassen. Dies wird durch den rekursiven Vektormodus `list` ermöglicht, der durch die Funktion `list()` erzeugt wird. Insbesondere für Funktionen (eingebaut oder selbstdefiniert) sind Listen der bevorzugte Objekttyp, um komplexere Berechnungsergebnisse an die aufrufende Stelle zurückzugeben.

Die Länge einer Liste ist die Anzahl ihrer Elemente. Die Elemente einer Liste sind nummeriert und unter Verwendung eckiger Klammern (wie Vektoren) indizierbar. Dabei liefern einfache eckige Klammern `[ ]` die indizierten Elemente wieder als Liste zurück. Falls nur ein Listenelement (z. B. durch `[ 2]`) indiziert wurde, erhält man also eine Liste der Länge 1. Um auf ein einzelnes Listenelement zuzugreifen, ohne es in eine Listenstruktur eingebettet zu bekommen, muss der Index des gewünschten Elements in *doppelte* eckige Klammern `[[ ]]` gesetzt werden. Eine Benennung der Listenelemente ist ebenfalls möglich; sie heißen dann Komponenten. Der Zugriff auf sie geschieht durch den jeweiligen Komponentennamen in einfachen oder doppelten eckigen Klammern oder mit Hilfe des Komponentenoperators `$`.

Beispiele mögen das Konzept der Listen und die Indizierungsmethoden erläutern, wobei die folgenden Objekte verwendet werden:

```
> personen
[1] "Peter" "Paul"  "Mary"  "Tic"

> alter
[1] 12 14 13 21
```

```
> m
     [,1] [,2] [,3] [,4]
[1,]  130  110  118  112
[2,]   26   24   25   25
```

2.9.1 Erzeugung und Indizierung: `list()`, `[[ ]]`, `head()` bzw. `tail()`

`> (daten <- list( personen, alter, m))`
`[[1]]:`
`[1] "Peter" "Paul"  "Mary"  "Tic"`
 	
`[[2]]:`
`[1] 12 14 13 21`
 	
`[[3]]:`
`     [,1] [,2] [,3] [,4]`
`[1,]  130  110  118  112`
`[2,]   26   24   25   25`
	Die Funktion `list()` fasst ihre Argumente zu einer Liste zusammen. Hier eine dreielementige Liste, deren Elemente ein `character`-Vektor, ein `numeric`-Vektor und eine `numeric`-Matrix sind.
	In der Ausgabe der Liste sind in den doppelten eckigen Klammern die Indices der Listenelemente zu sehen. (Das nachgestellte „`:`" hat keine Bedeutung.)

`> length( daten)` `[1] 3`	Die Länge einer Liste (= Anzahl ihrer Elemente) bestimmt `length()`.

`> daten[ 2]`
`[[1]]:`
`[1] 12 14 13 21`
	`[i]` ist die *Liste*, die das i-te Element der Ausgangsliste enthält.

`> daten[[ 1]]` `[1] "Peter" "Paul"  "Mary"  "Tic"`	`[[i]]` liefert das i-te Element der Ausgangsliste (hier also den Vektor `personen`).
`> daten[[ 2]][ 2:3]` `[1] 14 13`	`[[i]][j]` ist das j-te Element des i-ten Elements der Ausgangsliste (hier also das zweite und dritte Element des ursprünglichen Vektors `alter`).

`> daten[[ 3]][ 2, 4]` `[1] 25`	Wenn das i-te Element der Ausgangsliste eine Doppelindizierung zulässt, ist `[[i]][j,k]` das (j,k)-Element jenes i-ten Elements (hier also das $(2,4)$-Element der ursprünglichen Matrix `m`).
`> head( ....)` `> tail( ....)`	`head()` und `tail()` liefern bei Listen die ersten bzw. letzten `n` Listenelemente zurück; vgl. §2.6.2.

2.9.2 Benennung von Listenelementen und ihre Indizierung: `names()` und `$`

`> names( daten) <- c( "Personen", "Alter",`
`+ "Werte");  daten`
`$Personen:`
`[1] "Peter" "Paul"  "Mary"  "Tic"`
 	
`$Alter:`
`[1] 12 14 13 21`
 	
`$Werte:`
`     [,1] [,2] [,3] [,4]`
`[1,]  130  110  118  112`
`[2,]   26   24   25   25`
	Die Benennung der Listenelemente erfolgt mittels der Funktion `names()`. Die benannten Listenelemente werden Komponenten genannt (vgl. §2.5.2). In der Ausgabe der Liste sind die mit doppelten eckigen Klammern versehenen Indices verschwunden und durch die Komponentennamen samt einem vorangestellten `$`-Zeichen und einem nachgestellten „`:`“ ersetzt. Beachte, dass die Komponentennamen nicht in Hochkommata stehen!

`> names( daten)` `[1] "Personen" "Alter"  "Werte"`	Die Anwendung von `names()` liefert die Komponentennamen.
`> daten$Personen` `[1] "Peter" "Paul"  "Mary"  "Tic"`	Zugriff auf die `"Personen"` benannte Komponente von `daten` mittels des Komponentenoperators `$`.
`> daten[[ "Personen"]]` `[1] "Peter" "Paul"  "Mary"  "Tic"`	Analoge Wirkung der Indizierung durch doppelte eckige Klammern und Komponentennamen.

`> daten[ "Personen"]`
`$Personen`
`[1] "Peter" "Paul"  "Mary"  "Tic"`
	Beachte den Unterschied zur Indizierung mit einfachen eckigen Klammern und Komponentennamen.

`> daten$Werte[ 2,2]` `[1] 24`	Kombination der Indizierungsmethoden: `$Werte[2,2]` ist das $(2,2)$-Element der Komponente `"Werte"`.

`> (D <- list( Land = c( "Baden-Wuerttemberg",`
`+ "Bayern", "Berlin", ....),`
`+ Bevoelkerung = c( 10.7, 12.4, 3.4, ....)))`
`$Land:`
`  [1] "Baden-Wuerttemberg"  "Bayern" ....`
 	
`$Bevoelkerung:`
`  [1] 10.7  12.4 ....`
	Erzeugung einer Liste mit zwei Komponenten, die bereits bei der Listenerzeugung die Komponentennamen `Land` und `Bevoelkerung` erhält (also ohne Verwendung der Funktion `names()`).

Bemerkung: Eine weitere, bereits gesehene Beispielanwendung für Listen ist die Vergabe von Zeilen- und Spaltennamen an Matrizen (siehe §2.8.3, Seite 39).

Beachte: Indizierung von Komponenten unter Verwendung von `character`-Variablen, die die Komponentennamen enthalten, geht nur mit `[[ ]]` und nicht mit `$`. Beispiel:

```
> idx <- "Personen";   daten[[ idx]]     # Funkioniert wie gewuenscht.
[1] "Peter" "Paul"  "Mary"  "Tic"
> daten$idx     # Ist die korrekte Antwort, aber wohl nicht das Gewuenschte.
NULL
```

2.9.3 Komponentenweise Anwendung von Operationen: `lapply()`, `sapply()` & Co.

Ähnlich zur Funktion `apply()`, die eine zeilen- und spaltenweise Anwendung von Funktionen auf Matrizen ermöglicht, existieren zwei Funktionen, die dies für die Elemente einer Liste realisieren. Es handelt sich hierbei um `lapply()` und `sapply()`.

`> lapply( daten, class)`
`$Personen:`
`[1] "character"`
 	
`$Alter:`
`[1] "numeric"`
 	
`$Werte:`
`[1] "matrix"`
	`lapply()` erwartet als erstes Argument eine Liste, auf deren jedes Element sie das zweite Argument (hier die Funktion `class()`) anwendet. Das Ergebnis ist eine Liste der Resultate der Funktionsanwendungen (hier die Klassen der Elemente von `daten`). Die Ergebnisliste erhält die Komponentennamen der Eingabeliste, falls jene eine benannte Liste ist.

`> sapply( daten, class)`
`  Personen  Alter  Werte`
`"character" "numeric" "matrix"`
	`sapply()` funktioniert wie `lapply()`, simplifiziert aber die Ergebnisstruktur, wenn möglich; daher ist hier das Ergebnis ein *Vektor*.

`> x <- list( A = 1:10, B = exp( -3:3),`
`+ C = c( TRUE, FALSE, FALSE, TRUE))`
`> lapply( x, mean)`
`$A`
`[1] 5.5`
 	
`$B`
`[1] 4.535125`
 	
`$C`
`[1] 0.5`
	Weitere Beispiele zur Wirkung von `sapply()` und `lapply()`:
	Hier wird die Funktion `mean()` auf jede Komponente der Liste `x` angewendet.

`> lapply( x, quantile, probs = 1:3/4)`
`$A`
`  25%  50%  75%`
`3.25 5.50 7.75`
 	
`$B`
`  25%  50%  75%`
`0.2516074 1.0000000 5.0536690`
 	
`$C`
`25% 50% 75%`
`0.0 0.5 1.0`
	Sollen der komponentenweise anzuwendenden Funktion (hier `quantile()`, vgl. §2.3.3) weitere Argumente (hier `probs = 1:3/4`) mitgeliefert werden, so sind diese in der Form *Name = Wert* durch Komma getrennt hinter dem Namen jener Funktion aufzuführen.

`> sapply( x, quantile, probs = 1:3/4)`
`  A  B  C`
`25%  3.25  0.25160736  0.0`
`50%  5.50  1.00000000  0.5`
`75%  7.75  5.05366896  1.0`
	Dito, aber `sapply()` simplifiziert die Ergebnisstruktur zu einer *Matrix*, da die jeweiligen Ergebnisse (hier `numeric`-)Vektoren derselben Länge sind. Beachte wie die Ergebnisvektoren *spaltenweise* zu einer Matrix zusammengefasst werden und wie Zeilen- und Spaltennamen der Matrix zustandekommen.

Bemerkungen: Es gibt eine rekursive Version namens `rapply()` und eine „multivariate“ Variante namens `mapply()`, für deren mächtige Funktionalitäten wir auf die Online-Hilfe verweisen.

Eine *sehr* nützliche Funktion, um sich die Struktur von umfangreichen Listen (oder anderen Objekten) übersichtlich in abgekürzter Form anzusehen, ist `str()` (siehe §2.10.4).

2.10 Data Frames: Eine Klasse „zwischen“ Matrizen und Listen

Ein Data Frame ist eine Liste der Klasse `data.frame`, die benannte Elemente, also Komponenten hat. Diese Komponenten müssen atomare Vektoren der Modi `numeric`, `logical` bzw. `character` oder der Klassen `factor` bzw. `ordered` (also Faktoren) sein, die alle die gleiche Länge haben. Eine andere Interpretation ist die einer „verallgemeinerten“ Matrix mit benannten Spalten (und Zeilen): Innerhalb einer jeden Spalte stehen zwar (atomare) Elemente des gleichen Modus bzw. derselben Klasse, aber von Spalte zu Spalte können Modus bzw. Klasse variieren.

Data Frames sind das „Arbeitspferd“ vieler statistischer (Modellierungs-)Verfahren in **R**, da sie hervorragend geeignet sind, die typische Struktur p-dimensionaler Datensätze vom Stichprobenumfang n widerzuspiegeln: Jede der n Zeilen enthält einen p-dimensionalen Beobachtungsvektor, der von einer der n Untersuchungseinheiten (auch Fälle oder Englisch “cases” genannt) stammt. Jede der p Komponenten (= Spalten) enthält die n Werte, die für eine der p Variablen (= Merkmale) registriert wurden.

Der im **R**-Paket `rpart` enthaltene Datensatz `cu.summary` (Autodaten aus der 1990er Ausgabe des “Consumer Report”) ist ein Beispiel eines Data Frames (falls `rpart` noch nicht installiert, siehe Abschnitt 1.7):

```
> data( cu.summary,       # Stellt eine Kopie (!) des Objektes "cu.summary" aus
+ package = "rpart")      # dem Paket "rpart" im workspace zur Verfuegung.
> head( cu.summary)
                Price   Country Reliability Mileage  Type
Acura Integra 4 11950     Japan Much better      NA Small
Dodge Colt 4     6851     Japan        <NA>      NA Small
Dodge Omni 4     6995       USA  Much worse      NA Small
Eagle Summit 4   8895       USA      better      33 Small
Ford Escort   4  7402       USA       worse      33 Small
Ford Festiva 4   6319     Korea      better      37 Small
```

Hinweise: Alle bereits in “base-**R**” zur Verfügung stehenden Datensätze werden durch `library( help = "datasets")` aufgelistet. Mit `?`*datensatzname* wird die Online-Hilfe zum jeweiligen Datensatz gezeigt.

2.10.1 Indizierung: `[ ]`, `$`, `head()` und `tail()` sowie `subset()`

Der Zugriff auf einzelne Elemente, ganze Zeilen oder Komponenten eines Data Frames kann wie bei Matrizen mittels der Zeile-Spalte-Indizierung `[i, j]`, `[i, ]` oder `[, j]` geschehen bzw. über Zeilen- und Komponentennamen gemäß `["zname", "sname"]`, `["zname", ]` oder `[, "sname"]`. Die Spalten (und *nur* sie), da sie Komponenten einer Liste sind, können mit Hilfe des Komponentenoperators `$` über ihre Namen indiziert werden (via *dataframe*`$`*name*). Beispiele:

`> cu.summary[ 21,1]`
`[1] 8695`
`> cu.summary[ "Eagle Summit 4", "Mileage"]`
`[1] 33`
	Zugriff auf einzelne Elemente durch matrixtypische Zeile-Spalte-Indizierung bzw. -Benennung.

`> cu.summary[ "GEO Metro  3", ]`
`Price Country Reliability ....`
`GEO Metro  3  6695  Japan  <NA> ....`
	Matrixtypische Auswahl einer ganzen Zeile durch Zeilenbenennung. Beachte, dass die Spaltennamen übernommen werden.

`> cu.summary$Country`
`  [1] Japan  Japan  USA  ....`
`  ....`
`[115] Japan  Japan  Germany`
`10 Levels: Brazil England France .... USA`
	Listentypische Auswahl einer Komponente über ihren Namen. Beachte, dass beim `$`-Operator keine Hochkommata verwendet werden.

`> head( ....)` `> tail( ....)`	Sie liefern bei Data Frames (wie bei Matrizen; vgl. §2.8.5) die ersten bzw. letzten `n` Zeilen zurück.

`> subset( cu.summary, subset = Price < 6500,`
`+ select = c( Price, Country, Reliability))`
`  Price Country Reliability`
`Ford Festiva 4  6319  Korea  better`
`Hyundai Excel 4  5899  Korea  worse`
`Mitsubishi Precis 4  5899  Korea  worse`
`Subaru Justy 3  5866  Japan  <NA>`
`Toyota Tercel 4  6488  Japan Much better`
	`subset()` wählt über ihr Argument `subset`, das (im Ergebnis) einen `logical`-Vektor erwartet, die Zeilen des Data Frames aus und über ihr Argument `select`, das (im Ergebnis) einen die Spalten des Data Frames indizierenden Vektor erwartet, eben jene aus.

2.10.2 Erzeugung: `data.frame()`, `expand.grid()`

Data Frames können zusammengesetzt werden aus bereits existierenden Vektoren, Matrizen, Listen und anderen Data Frames, und zwar standardmäßig mit der Funktion `data.frame()`. Dabei müssen die „Dimensionen“ der zusammenzusetzenden Objekte i. d. R. zueinanderpassen. Ausnahmen sind in der Online-Hilfe beschrieben.

Beim „Einbau“ von Matrizen, Listen und anderen Data Frames liefert jede Spalte bzw. Komponente eine eigene Komponente des neuen Data Frames. Dazu müssen Matrizen und Data Frames dieselbe Zeilenzahl haben und die Komponenten von Listen dieselbe Länge (die wiederum alle mit der Länge eventuell hinzugefügter Vektoren übereinstimmen müssen). `character`-Vektoren werden beim Einbau in einen Data Frame (per Voreinstellung) zu Faktoren konvertiert.

Im Fall von Vektoren können für die Komponenten (= Spalten) explizit Namen angegeben werden, gemäß der Form *Name* = *Wert*. Wird dies nicht getan, übernehmen die Data-Frame-Komponenten den Objektnamen des jeweiligen Vektors oder werden von **R** automatisch mit einem syntaktisch zulässigen und eineindeutigen (aber häufig furchtbaren) Namen versehen, der gelegentlich mit einem `X` beginnt.

Als Zeilennamen des Data Frames werden die Elementenamen (eines Vektors) oder Zeilennamen der ersten Komponente, die so etwas besitzt, übernommen, ansonsten werden die Zeilen durchnummeriert.

In der nächsten Tabelle werden folgende Beispielobjekte verwendet:

```
> hoehe
[1] 181 186 178 175
> Gewicht
[1] 130 110  63  59
> Diaet
[1]  TRUE FALSE FALSE  TRUE
```

```
> Werte
     Gewicht Alter groesse
[1,]     105    26     181
[2,]      83    24     186
[3,]      64    25     178
[4,]      58    25     175
```

`> (W.df <- data.frame( Groesse = hoehe,`
`+ Gewicht, Diaet, BlutG = c( "A", "A", "AB",`
`+ "0"), Rh = c( "R+", "R-", "R-", "R-")))`
`  Groesse Gewicht Diaet BlutG Rh`
`1  181  130  TRUE  A R+`
`2  186  110 FALSE  A R-`
`3  178  63 FALSE  AB R-`
`4  175  59  TRUE  0 R-`
	Erzeugung eines Data Frames mit den explizit benannten Komponenten `Groesse`, `BlutG` und `Rh` sowie den Komponenten `Gewicht` und `Diaet`, die als Namen den Objektnamen des entsprechenden Vektors erhalten.

`> (W2.df <- data.frame( Werte))`
`  Gewicht Alter groesse`
`1  105  26  181`
`2  83  24  186`
`3  64  25  178`
`4  58  25  175`
	Erzeugung eines Data Frames aus einer bereits existierenden Matrix. Die Namen von Spalten werden von der Matrix übernommen, falls sie existieren.

Bei Betrachtung des obigen Beispiels mit dem `character`-Vektor der Blutgruppen `c("A", "A", "AB", "0")` fällt auf, dass in dem Data Frame `W.df` die Hochkommata des Modus' `character` verloren gegangen sind. Hier ist zu wiederholen, dass `character`-Vektoren bei Übergabe in einen Data Frame gemäß Voreinstellung *automatisch* zu Faktoren konvertiert werden. Um einen `character`-Vektor – warum auch immer – als solchen in einen Data Frame zu übernehmen, ist die Funktion `I()` zu verwenden; sie sorgt für die unveränderte (identische oder auch "as is" genannte) Übernahme des Objektes. Die betreffende Komponente des Data Frames erhält dann den Modus `character` und die Klasse `AsIs`:

`> (W3.df <- data.frame( W.df, Spitzname =`
`+ I( c( "Hulk", "Richi", "Rote Zora",`
`+ "Zimtzicke"))))`
`  Groesse Gewicht Diaet BlutG Rh Spitzname`
`1  181  130  TRUE  A R+  Hulk`
`2  186  110 FALSE  A R-  Richi`
`3  178  63 FALSE  AB R- Rote Zora`
`4  175  59  TRUE  0 R- Zimtzicke`
	Erzeugung eines Data Frames, wobei ein `character`-Vektor als solcher übernommen wird. Bei der Ausgabe sind allerdings auch hier die Hochkommata „verloren“, sodass so kein Unterschied zwischen einer Faktor- und einer `character`-Komponente zu erkennen ist. (Siehe aber den folgenden §2.10.4.)

In manchen Situationen ist es notwendig, zu k (Faktor-)Vektoren mit je $n_1, n_2, \ldots, n_k$ verschiedenen Elementen (bzw. Levels) alle möglichen $n_1 \cdot n_2 \cdots n_k$ Kombinationen zu generieren – und zwar nützlicherweise, wie sich noch herausstellen wird, als Zeilen eines Data Frames.

Dies könnte z. B. der Fall sein für die Erfassung von Daten aus Experimenten, in denen eine Messgröße von Interesse unter verschiedenen Bedingungen erhoben wurde, welche durch die Kombinationen von verschiedenen Levels mehrerer (Einfluss-)Faktoren charakterisiert werden. Oder wenn man eine Funktion, die von mindestens zwei Variablen abhängt, auf einem vollständigen endlichen *Gitter* aller Kombinationen ausgewählter Werte dieser Variablen auswerten will, aber die dabei zu durchlaufenden Werte der Variablen als separate Vektoren gespeichert hat.

Hier ist die Funktion `expand.grid()` das Werkzeug der Wahl, denn sie erzeugt aus jenen separaten Vektoren das entsprechende Gitter (Engl.: "grid").

Beispiel: Soll eine Funktion mit drei Argumenten für alle Kombinationen der drei Strahlungswerte in $S = \{0, 50, 100\}$ mit den fünf Temperaturwerten in $T = \{0, 10, 20, 30, 40\}$ und den zwei Wind-„Werten“ in $W = \{$„ja“, „nein“$\}$ ausgewertet werden, und sind S, T und W als separate Vektoren à la

```
> Strahl <- seq( 0, 100, by = 50);     Temp <- seq( 0, 40, by = 10)
> Wind <- c( "ja", "nein")
```

gespeichert, dann erhalten wir das gewünschte Gitter $S \times T \times W$ mit seinen $3 \cdot 5 \cdot 2 = 30$ Elementen als Data Frame mit (frei benennbaren Komponenten) durch

```
> expand.grid( Strahlung = Strahl, Temperatur = Temp, Wind = Wind)
  Strahlung Temperatur Wind
1         0          0   ja
2        50          0   ja
3       100          0   ja
```

```
4          0         10   ja
5         50         10   ja
....
15       100         40   ja
16         0          0 nein
....
29        50         40 nein
30       100         40 nein
```

Offenbar durchläuft die erste Komponente des resultierenden Data Frames ihren Wertebereich vollständig, bevor die zweite Komponente auf ihren nächsten Wert „springt“. Analog ändert sich der Wert einer j-ten Komponente, falls vorhanden, erst auf den nächsten, wenn sämtliche Kombinationen in den ersten $j-1$ Komponenten durchlaufen sind, usw.

Hinweis: Zu den verwandten Problemen der Erzeugung aller möglichen m-elementigen Kombinationen, sprich Teilmengen bzw. ungeordneten Auswahlen aus *einer* n-elementigen (Grund-) Menge und der Berechnung ihrer Anzahl mit Hilfe der Funktionen `combn()` bzw. `choose()` siehe §2.3.4.

2.10.3 Zeilen- & Spaltennamen: `dimnames()`, `row.names()` & `case.names()`, `col.names()` & `names()`

Auf Zeilen- und Spaltennamen von Data Frames kann analog zu denen von Matrizen mit `dimnames()`, `rownames()` bzw. `colnames()` zugegriffen werden. Allerdings gibt es ein paar auf Data Frames spezialisierte Versionen, nämlich `row.names()` oder `case.names()` für die Zeilen-, sprich Fallnamen bzw. `colnames()` oder (einfach) `names()` für die Spalten-, sprich Variablennamen. Diese Funktionsnamen orientieren sich an der am Anfang von Abschnitt 2.10 kurz erwähnten Interpretation der Struktur von Data Frames.

2.10.4 “Summary statistics” und Struktur eines Data Frames: `summary()` und `str()`

Die Funktion `summary()` hat eine spezielle Methode für Data Frames, sodass sie „direkt“ auf einen Data Frame anwendbar ist und geeignete “summary statistics” einer jeden Komponente des Data Frames in recht übersichtlicher Form liefert (siehe auch Seite 22 bzgl. `summary()`):

Für `numeric`-Komponenten werden die arithmetischen “summary statistics” bestimmt. Für Faktoren und geordnete Faktoren werden die Häufigkeitstabellen ihrer Levels bzw. Ausschnitte daraus angegeben, falls die Tabellen zu groß sind, weil mehr als sechs Levels auftreten. (Die Anzahl der maximal anzuzeigenden Faktorlevels lässt sich aber mit dem `summary()`-Argument `maxsum` auch ändern.) Die Reihenfolge der Levels in einer Häufigkeitstabelle entspricht dabei der Leveldordnung des Faktors. (Für Komponenten der Klasse `AsIs` (hier nicht gezeigt) wird die Länge dieser Komponente, ihre Klasse und ihr Modus, also `character` ausgegeben.) Beispiel:

```
> summary( cu.summary)
    Price            Country        Reliability    Mileage           Type
Min.   : 5866   USA       :49   Much worse :18   Min.   :18.00   Compact:22
1st Qu.:10125   Japan     :31   worse      :12   1st Qu.:21.00   Large  : 7
Median :13150   Germany   :11   average    :26   Median :23.00   Medium :30
Mean   :15743   Japan/USA: 9   better     : 8   Mean   :24.58   Small  :22
3rd Qu.:18900   Korea     : 5   Much better:21   3rd Qu.:27.00   Sporty :26
Max.   :41990   Sweden    : 5   NA's       :32   Max.   :37.00   Van    :10
                (Other)   : 7                    NA's   :57.00
```

Die bereits im Zusammenhang mit Listen (auf Seite 48) erwähnte Funktion `str()`, wie Struktur, ist auch für größere Data Frames eine *hervorragende* Möglichkeit, Informationen über ihre Struktur und ihren Inhalt in übersichtlicher, abgekürzter Form darzustellen:

```
> str( cu.summary)
'data.frame':   117 obs. of  5 variables:
 $ Price      : num  11950  6851  6995  8895  7402 ...
 $ Country    : Factor w/ 10 levels "Brazil","England",..: 5 5 10 10 10 ...
 $ Reliability: Ord.factor w/ 5 levels "Much worse"<"worse"<..: 5 NA 1 4 2 ...
 $ Mileage    : num  NA NA NA 33 33 37 NA NA 32 NA ...
 $ Type       : Factor w/ 6 levels "Compact","Large",..: 4 4 4 4 4 4 4 4 ...
```

2.10.5 Komponentenweise Anwendung von Operationen: `lapply()`, `sapply()`

Analog zur Anwendung einer Funktion auf die Elemente einer Liste durch `lapply()` oder `sapply()` (siehe Seite 48) lässt sich dies auch (*nur*) für die Komponenten eines Data Frames mit `lapply()` oder `sapply()` realisieren, da sie als Listenelemente auffassbar sind:

`> lapply( cu.summary, class)`
`$Price:`
`[1] "numeric"`
 	
`$Country:`
`[1] "factor"`
 	
`$Reliability:`
`[1] "ordered" "factor"`
 	
`$Mileage:`
`[1] "numeric"`
 	
`$Type:`
`[1] "factor"`
	Hier wendet `lapply()` auf die Komponenten (d. h. die Listenelemente) des Data Frames `cu.summary` die Funktion `class()` an, was als Resultat eine Liste mit den Klassen der Komponenten (= Listenelemente) liefert.
	
	(Memo: Zu Klasse und Modus von Faktoren siehe bei Bedarf nochmal am Anfang von Abschnitt 2.7.)

`> sapply( cu.summary, mode)`
`     Price   Country Reliability   Mileage`
`  "numeric" "numeric"   "numeric" "numeric"`
`      Type`
`  "numeric"`
	`sapply()` macht das Analoge zu `lapply()` (hier mit `mode()`), vereinfacht aber nach Möglichkeit die Struktur des Resultats; hier zu einem Vektor.

Natürlich ist auch die komponentenweise Anwendung von `summary()` auf einen Data Frame möglich, wie z. B. durch `lapply( cu.summary, summary)`, was eine Liste mit den “summary statistics” einer jeden Komponente des Data Frames liefert. Allerdings ist diese nicht so übersichtlich wie das Resultat der direkten Anwendung von `summary()` auf einen Data Frame.

2.10.6 Anwendung von Operationen auf Faktor(en)gruppierte Zeilen: `by()`

Die Funktion `by()` bewerkstelligt die zeilenweise Aufteilung eines Data Frames gemäß der Werte eines Faktors – oder mehrerer Faktoren – in (wiederum) Data Frames und die nachfolgende Anwendung jeweils einer (und derselben) Funktion auf diese Unter-Data Frames. (Vgl. auch §2.7.7 zur Arbeitsweise von `tapply()`, welches die interne Grundlage für `by()` ist.)

Im folgenden Beispiel teilt `by()` den Data Frame `cu.summary` zeilenweise gemäß der verschiedenen Werte des Faktors `cu.summary$Type` zunächst in Unter-Data Frames auf und wendet dann jeweils die Funktion `summary()` auf diese an. Das Resultat wird hier nur teilweise gezeigt:

```
> by( cu.summary, cu.summary$Type, summary)
cu.summary$Type: Compact
     Price            Country       Reliability    Mileage           Type
 Min.   : 8620   USA      :7   Much worse :2    Min.   :21.00   Compact:22
 1st Qu.:10660   Germany  :4   worse      :5    1st Qu.:23.00   Large  : 0
 Median :11898   Japan/USA:4   average    :3    Median :24.00   Medium : 0
 Mean   :15202   Japan    :3   better     :4    Mean   :24.13   Small  : 0
 3rd Qu.:18307   Sweden   :3   Much better:5    3rd Qu.:25.50   Sporty : 0
 Max.   :31600   France   :1   NA's       :3    Max.   :27.00   Van    : 0
                 (Other)  :0                    NA's   : 7.00
------------------------------------------------------------
cu.summary$Type: Large
     Price          Country       Reliability    Mileage           Type
 Min.   :14525   USA    :7   Much worse :2    Min.   :18.00   Compact:0
 1st Qu.:16701   Brazil :0   worse      :0    1st Qu.:19.00   Large  :7
 Median :20225   England:0   average    :5    Median :20.00   Medium :0
 Mean   :21500   France :0   better     :0    Mean   :20.33   Small  :0
 3rd Qu.:27180   Germany:0   Much better:0    3rd Qu.:21.50   Sporty :0
 Max.   :27986   Japan  :0                    Max.   :23.00   Van    :0
                 (Other):0                    NA's   : 4.00
------------------------------------------------------------
....
```

Bemerkungen: Die bloße Aufteilung eines Data Frames in Unter-Data Frames gemäß der Werte eines Faktors oder mehrerer Faktoren (ohne die direkte Anwendung einer sonstigen Operation) ist mithilfe der Funktion `split()` möglich (vgl. auch §2.7.7). Eine weitere Funktion zur Berechnung von "summary statistics" für Teilmengen – typischwerweise – eines Data Frames ist `aggregate()`. (Für beide Funktionen verweisen wir auf ihre Online-Hilfe.)

2.10.7 „Organisatorisches" zu Data Frames und dem Objektesuchpfad: `attach()`, `detach()` und `search()`, `with()`, `within()` und `transform()`

Die Komponenten eines Data Frames sind, wie auf den vorherigen Seiten vorgestellt, mittels des Komponentenoperators `$` über ihre Namen indizierbar. Sie sind Bestandteile des Data Frames und keine eigenständigen Objekte. D. h., jedes Mal, wenn z. B. auf die Komponente `Price` in `cu.summary` zugegriffen werden soll, ist stets auch der Name des Data Frames `cu.summary` anzugeben, damit **R** weiß, wo es die Komponente (sprich Variable) `Price` herholen soll.

Gelegentlich jedoch ist der Zugriff auf eine Komponente wiederholt oder auf mehrere Komponenten gleichzeitig notwendig (oder beides). Dann wird die Verwendung von *dataframe*`$`*komponentenname* schnell sehr aufwändig und unübersichtlich. Um hier Erleichterung zu schaffen, kann man den Data Frame, auf dessen Komponenten zugegriffen werden soll, vorher „öffnen", sodass die Angabe seines Namens und die Verwendung des Operators `$` nicht mehr nötig ist, sondern die Komponenten des Data Frames gewissermaßen zu eigenständigen Objekten werden.

- Das „Öffnen" eines Data Frames bewerkstelligt die Funktion `attach()` und das „Schließen" die Funktion `detach()`. Durch `attach( DF)` wird der Data Frame `DF` (falls er existiert) in der zweiten Position des Objektesuchpfads von **R** hinzugefügt (Engl.: "attached"). An der ersten Stelle dieses Suchpfades steht der aktuelle "workspace" (namens `.GlobalEnv`) und beim Aufruf eines Objektes über seinen Namen sucht **R** zuerst in diesem workspace. Wenn **R** ein solchermaßen benanntes Objekt dort nicht findet, wird im nächsten Eintrag des Suchpfades, also hier im Data Frame `DF` weitergesucht. Führt dies zum Erfolg, wird das Objekt zur Verfügung gestellt; anderenfalls wird eine entsprechende Fehlermeldung

ausgegeben. Soll der Data Frame `DF` aus dem Suchpfad wieder entfernt werden, so geschieht dies durch `detach( DF)`.

Wie der aktuelle Suchpfad aussieht, zeigt die Funktion `search()`: Es werden alle geladenen Pakete und anderen “attach”-ten Datenbanken oder Objekte aufgezählt.

Beachte: Die obige Methodik birgt natürlich eine Gefahr: Angenommen, es soll auf eine Komponente des Data Frames `DF` zugegriffen werden, deren Name genauso lautet, wie der eines Objektes im workspace. Dann wird **R** bereits im workspace fündig, beendet (!) seine Suche und stellt das gefundene Objekt zur Verfügung (und zwar *ohne* dem/der Benutzer/in mitzuteilen, dass es noch ein weiteres Objekt des gleichen Namens gibt). Fazit: Es obliegt dem/der Benutzer/in zu kontrollieren, dass das „richtige“ Objekt verwendet wird. (Dies ist ein weiteres Beispiel für die Notwendigkeit, den eigenen workspace öfter einmal aufzuräumen. Siehe hierzu Abschnitt 1.5, Seite 6.)

Am Beispiel eines weiteren, bereits in “base-**R**” zur Verfügung stehenden Data Frames namens `airquality` (mit Messwerten zur Luftqualität in New York von Mai bis September 1973) soll die oben beschriebene Funktionsweise von `attach()` und Co. dargestellt werden:

```
> head( airquality, 5)                    # Fuer Infos siehe ?airquality
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
```

`> Ozone[ 41:42]` `Error: Object "Ozone" not found`	Offenbar gibt es kein Objekt namens `Ozone` im **R**-Suchpfad.

`> attach( airquality)`
`> Ozone[ 41:42]`
`[1] 39 NA`
`> summary( Solar.R)`
`Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's`
`  7.0  115.8  205.0 185.9  258.8 334.0  7.0`
	Nachdem der Data Frame `airquality` dem **R**-Suchpfad hinzugefügt worden ist, sind `Ozone` und `Solar.R` sowie alle anderen Komponenten von `airquality` als eigenständige Objekte unter ihrem Komponentennamen verfügbar, denn ...

`> search()`
`[1] ".GlobalEnv"  "airquality"`
`[3] "package:methods"  "package:stats"`
`[5] "package:graphics"  "package:grDevices"`
`[7] "package:utils"  "package:datasets"`
`[9] "Autoloads"  "package:base"`
	... wie der aktuelle Suchpfad zeigt, befindet sich der Data Frame `airquality` im Augenblick darin an zweiter Position, aber ...

`> detach( airquality)`
`> summary( Solar.R)`
`Error in summary(Solar.R): Object "Solar.R" not found`
	... nach dem Entfernen des Data Frames aus dem Suchpfad eben nicht mehr (wie ein erneutes `search()` explizit bestätigen würde).

- Weitere, sehr nützliche Funktionen zur Arbeit mit und Modifikation von Data Frames sind auch `with()`, `within()` und `transform()`, für deren detailierte Funktionsweise wir nachdrücklich auf ihre Online-Hilfe verweisen und hier nur kurze Beschreibungen sowie jeweils ein kleines Anwendungsbeispiel liefern:

▷ `with( data, expr)` wertet den Ausdruck `expr` in der „lokalen“ Umgebung `data` aus, indem `data` quasi vorübergehend in den Suchpfad eingehängt wird. Dabei ist `data` typischerweise ein Data Frame, kann aber auch eine Liste sein. Das Resultat des `with()`-Aufrufs ist der Wert von `expr`, aber jegliche Zuweisung innerhalb von `expr` ist nur lokal gültig, d. h., findet nicht im benutzereigenen workspace statt und ist daher außerhalb des `with()`-Aufrufs vergessen.

▷ `within( data, expr)` arbeitet ähnlich zu `with()`, versucht allerdings, die in `expr` ausgeführten Zuweisungen und anderen Modifikationen in der Umgebung `data` permanent zu machen. Der Wert des `within()`-Aufrufs ist der Wert des Objekts in `data` mit den durch `expr` resultierenden Modifikationen.

▷ `transform( '_data', ...)` arbeitet ähnlich zu `within()`, allerdings dürfen in `...` nur Ausdrücke der Art `name = expr` auftreten. Jede `expr` wird in der durch den Data Frame `_data` definierten Umgebung ausgewertet und ihr Wert nach Möglichkeit der jeweils zu `name` passenden Komponente des Data Frames `_data` zugewiesen. Wenn keine passende Komponente in `_data` existiert, wird der erhalten Wert an `_data` als weitere Komponente angehängt. Der Wert des `transform()`-Aufrufs ist der modifizierte Wert von `_data`.

`> with( airquality, {`
`+  TempC <- (Temp - 32) * 5/9`
`+  summary( TempC)`
`+  summary( Ozone)`
`+  })`
`  Min. 1st Qu. Median  Mean 3rd Qu.`
`  1.00  18.00  31.50 42.13  63.25`
`  Max.  NA's`
`168.00  37.00`
	Die in `{....}` stehenden Anweisungen sind der Ausdruck, der in der durch `airquality` lokal definierten Umgebung auszuwerten ist, in der `Temp` und `Ozone` bekannt sind. Der Wert dieser `with()`-Anweisung ist der Wert des letzten Ausdrucks in `{....}`, also `summary( Ozone)`, wohingegen `TempC` danach genauso wieder verschwunden ist wie das – gar nicht ausgegebene – Ergebnis von `summary( TempC)`.

`> air2 <- within( airquality, {`
`+  TempC <- (Temp - 32) * 5/9`
`+  Month <- factor( month.abb[ Month])`
`+  rm( Day)`
`+  })`
`> head( air2, 2)`
`  Ozone Solar.R Wind Temp Month  TempC`
`1  41  190  7.4  67  May 19.444`
`2  36  118  8.0  72  May 22.222`
	Die Anweisungen in `{....}` werden in der durch `airquality` lokal definierten Umgebung ausgewertet, worin `Temp`, `Month` und `Day` bekannt sind. Das Ergebnis dieses `within()`s ist der Wert von `airquality` nach Ausführung der Modifikationen, also mit neu angehängter Komponente `TempC`, neuem Wert für die bereits existierende Komponente `Month` und ohne die Komponente `Day`. Beachte: `airquality` selbst bleibt unverändert.

`> air3 <- transform( airquality,`
`+  NegOzone = -Ozone,`
`+  Temp = (Temp - 32) * 5/9)`
`> head( air3, 2)`
`  Ozone Solar.R Wind  Temp Month Day`
`1  41  190  7.4 19.444  5  1`
`2  36  118  8.0 22.222  5  2`
`  NegOzone`
`1  -41`
`2  -36`
	Die Ausdrücke `name = expr` werden in der durch `airquality` lokal definierten Umgebung ausgewertet, worin `Ozone` und `Temp` bekannt sind. Das Ergebnis dieses `transform()`s ist der Wert von `airquality` nach Ausführung der Modifikationen, also mit neu angehängter Komponente `NegOzone` und neuem Wert für die bereits existierende Komponente `Temp`. Beachte: `airquality` selbst bleibt unverändert.

2.10.8 Nützliche Transformationen: stack(), reshape(), merge() und das Paket reshape

Die Arbeit mit einem komplexeren Datensatz, der in der Regel aus Gründen der Datenerfassung und -verwaltung zunächst in Form einer Tabelle arrangiert wurde wie sie nicht für die direkte Verarbeitung mit **R** geeignet ist, bedingt oft das – wiederholte – Umformen dieser Datenstruktur, die in **R** typischerweise als Data Frames gespeichert wird. Zu diesem Zweck stehen in "base **R**" und in dem einen oder anderen **R**-Package mehrere, unterschiedlich leistungsfähige und komplizierte Funktionen zur Verfügung, deren Beschreibung hier zu weit führte. Stattdessen zählen wir nur einige auf und verweisen auf ihre Online-Hilfe bzw. auf sonstige Beschreibungen:

- `stack()` stapelt Spalten oder vektorielle Komponenten eines Objektes zu einem neuen Vektor übereinander und generiert einen Faktor, der die Herkunftsspalte oder -komponente eines jeden Elementes des neuen Vektors angibt. `unstack()` kehrt die Operation um und „entstapelt" einen solchen Vektor entsprechend (siehe die Online-Hilfe).

- `reshape()` ermöglicht die Transformation von (auch komplexeren) Data Frames aus dem sogenannten "wide"-Format (bei dem – an derselben Untersuchungseinheit erhobene – Messwiederholungen in separaten Spalten ein und derselben Zeile gespeichert sind) ins "long"-Format (bei dem jene Messwiederholungen in einer Spalte, aber auf verschiedene Zeilen verteilt sind) und umgekehrt. Ist z. B. für die Umformung von Data Frames mit Longitudinaldaten geeignet. (Umfangreiche Details sind in der Online-Hilfe zu finden.)

- `merge()` erlaubt das Zusammensetzen von Data Frames mithilfe von Vereinigungsoperationen wie sie für Datenbanken verwendet werden (siehe die Online-Hilfe).

- Das Package `reshape` (beachte, dass hier nicht die Funktion `reshape()` gemeint ist!) stellt zwei Funktionen namens `melt()` und `cast()` zur Verfügung, durch die die Funktionalität von `tapply()` (vgl. §2.7.7), `by()`, `aggregate()` (für beide vgl. §2.10.6), `reshape()` und anderen in ein vereinheitlichtes Konzept zusammengefasst worden ist/sei. Siehe hierzu die ausführliche und beispielbewehrte Darstellung in [83, Wickham (2007)].

2.11 Abfrage und Konversion der Objektklasse sowie Abfrage von NA, NaN, Inf und NULL

Es gibt die Möglichkeit, Informationen über die Klasse eines Datenobjekts abzufragen. Dies wird durch die Funktion `class()` erreicht (wie in obigem Beispiel schon gesehen). Sie liefert die entsprechende Information für ihr Argument. Die Überprüfung, ob ein Objekt einer speziellen Klasse angehört, geht mit der Funktion `is( object, class)`. Ihr Argument `object` bekommt das zu prüfende Objekt und ihr Argument `class` den Namen (als Zeichenkette) der Klasse. Das Resultat ist `TRUE`, wenn das Objekt der genannten Klasse angehört, `FALSE` anderenfalls. Die Funktion `as( object, class)` konvertiert das Objekt `object` in eins der Klasse `class`.

Es folgt eine (unvollständige) Auswahl an Möglichkeiten für das Argument `class`, wie sie in `is()` und `as()` Anwendung finden können:

`"array"`	Arrays	`"list"`	Listen
`"character"`	Zeichenkettenobjekte	`"logical"`	Logische Objekte
`"complex"`	Komplexwertige Objekte	`"matrix"`	Matrixobjekte
`"function"`	Funktionen	`"name"`	Benannte Objekte
`"integer"`	Ganzzahlige Objekte	`"numeric"`	Numerische Objekte
		`"vector"`	Vektoren

Ausnahmen sind die Klassen `data.frame`, `factor` und `ordered`: Für eine Konversion dorthinein gibt es die Funktionen `as.data.frame()`, `as.factor()` und `as.ordered()`.

Die symbolischen Konstanten `NA`, `NaN`, `Inf` und `NULL` nehmen Sonderrollen in **R** ein:

- `NA` (Abkürzung für "not available") als Element eines Objektes repräsentiert einen „fehlenden" Wert an der Stelle dieses Elements.
- `NaN` (steht für "not a number") und repräsentiert Elemente die „numerisch nicht definiert" sind, weil sie das Resultat von Operationen wie $0/0$ oder $\infty - \infty$ sind.
- `Inf` implementiert ∞ (= Unendlich) in **R**.
- `NULL` ist sozusagen das leere Objekt (Länge = 0).

Der Test, ob ein Objekt `NA` oder `NaN` enthält, wird mit der Funktion `is.na()` geklärt. Soll konkret nur auf `NaN` gecheckt werden, ist `is.nan()` zu verwenden. Wo sich in einem Vektor die Einträge `Inf` oder `-Inf` befinden, zeigt `is.infinite()`; mit `is.finite()` wird Endlichkeit geprüft. Ob ein Objekt das `NULL`-Objekt ist, klärt die Funktion `is.null()`.

Beachte: In einem `character`-Vektor ist der "missing value" `NA` verschieden von der Zeichenkette `"NA"`, die aus den zwei Buchstaben „N" und „A" besteht.

Ein paar Anwendungsbeispiele für und Falltüren in obige/n Konzepte/n:

```
> a <- c( "1", "2", "3", "4");     class( a)
[1] "character"

> a + 10
Error in a + 10 : non-numeric argument to binary operator
> is( a, "numeric")
[1] FALSE

> a <- as( a, "numeric");     class( a);     a
[1] "numeric"
[1] 1 2 3 4

> as.numeric( factor( 1:5)) + 1
[1] 2 3 4 5 6
> as.numeric( factor( 11:15)) + 1   # Memo: Faktoren sind vom Modus numeric!
[1] 2 3 4 5 6

> class( cu.summary)
[1] "data.frame"
> is( cu.summary, "data.frame");     is( cu.summary, "list")
[1] TRUE
[1] FALSE

> x <- c( 2, -9, NA, 0);     class( x);     is.na( x)
[1] "numeric"
[1] FALSE FALSE  TRUE FALSE

> (y <- x/c( 2,0,2,0))
[1]    1 -Inf   NA  NaN
```

```
> is.na( y);     is.nan( y)
[1] FALSE FALSE  TRUE  TRUE
[1] FALSE FALSE FALSE  TRUE

> is.finite( y);     is.infinite( y)
[1]  TRUE FALSE FALSE FALSE
[1] FALSE  TRUE FALSE FALSE

> is.na( c( "A", NA, "NA"))
[1] FALSE  TRUE FALSE

> (xx <- as( x, "character"));     is.na( xx)
[1] "2"  "-9" NA   "0"
[1] FALSE FALSE  TRUE FALSE
```

Hinweis: Nützlich ist die Kombination der Funktionen `is.na()` und `which()` (siehe hierfür nochmal §2.4.2), um z. B. herauszufinden, an welchen Positionen eines Vektores oder einer Matrix ein Eintrag `NA` steht. Für die Anwendung auf Matrizen ist das `which()`-Argument `arr.ind` sinnvollerweise auf `TRUE` zu setzen (siehe auch die Online-Hilfe).

3 Import und Export von Daten bzw. ihre Ausgabe am Bildschirm

Üblicherweise wird man große Datensätze, die es darzustellen und zu analysieren gilt, aus externen Dateien in **R** einlesen, was Datenimport genannt wird. Umgekehrt ist es gelegentlich notwendig, **R**-Datenobjekte in eine Datei auszulagern oder sogar in einen speziellen Dateityp zu exportieren, um sie für andere Programme lesbar zu machen.

3.1 Datenimport aus einer Datei: `scan()`, `read.table()` und Co.

Zum Zweck des Datenimports aus einer Datei stehen verschieden leistungsfähige Funktionen zur Verfügung. Wir konzentrieren uns hier auf den Datenimport aus ASCII-Textdateien mit Hilfe der Funktionen `scan()` sowie `read.table()` und ihrer Verwandten. Hinsichtlich des Imports anderer Dateiformate verweisen wir auf das **R**-Manual "R Data Import/Export" (siehe z. B. die **R**-Online-Hilfe und das **R**-Paket `foreign`).

3.1.1 Die Funktion `scan()`

Mit Hilfe der Funktion `scan()` kann man aus einer ASCII-Datei Daten einlesen, die dort in einem Tabellenformat (= Zeilen-Spalten-Schema) vorliegen. Dabei stehen Optionen zur Verfügung, die steuern, wie die Datei zu lesen ist und wie die resultierende Datenstruktur in **R** auszusehen hat. Die Funktion `scan()` ist recht schnell, allerdings muss der Benutzer/die Benutzerin für eine korrekte Felder- und Datenmodi-Zuordnung sorgen. `scan()`'s unvollständige Syntax mit ihren Voreinstellungen lautet

```
scan( file = "", what = double(0), nmax = -1, sep = "", dec = ".", skip = 0,
      flush = FALSE)
```

Zur Bedeutung der aufgeführten Argumente:

- `file`: Erwartet in Hochkommata den Namen (nötigenfalls mit Pfadangabe) der Quelldatei, aus der die Daten gelesen werden sollen. (Es kann sogar eine vollständige URL sein!) Die Voreinstellung `""` liest die Daten von der Tastatur ein, wobei dieser Vorgang durch die Eingabe einer leeren Zeile oder von <Ctrl-D> (bzw. <Strg-D>) unter Unix oder <Ctrl-Z> (bzw. <Strg-Z>) unter Windows abgeschlossen wird.
- `what` dient als Muster, gemäß dessen die Daten zeilenweise einzulesen sind, wobei die Voreinstellung `double(0)` bedeutet, dass versucht wird, *alle* Daten als `numeric` (in „doppelter Genauigkeit") zu interpretieren. An Stelle von `double(0)` sind (u. a.) `logical(0)`, `numeric(0)`, `complex(0)` oder `character(0)` möglich, bzw. Beispiele vom jeweiligen Modus, also `TRUE`, `0`, `0i` oder `""`, was praktikabler, weil kürzer ist. `scan()` versucht dann, alle Daten in dem jeweiligen Modus einzulesen. Das Resultat von `scan()` ist ein Vektor des Modus' der eingelesenen Daten.
  Bekommt `what` eine Liste zugewiesen, wird angenommen, dass jede Zeile der Quelldatei ein Datensatz ist, wobei die Länge der `what`-Liste als die Zahl der Felder eines einzelnen Datensatzes interpretiert wird und jedes Feld von dem Modus ist, den die entsprechende Komponente in `what` hat. Das Resultat von `scan()` ist dann eine Liste, deren Komponenten Vektoren sind, und zwar jeweils des Modus' der entsprechenden Komponenten in `what`.
- `nmax` ist die Maximalzahl einzulesender Werte oder, falls `what` eine Liste ist, die Maximalzahl der einzulesenden Datensätze. Bei Weglassung oder negativem Wert wird bis zum Ende der Quelldatei gelesen.
- `sep` gibt in Hochkommata das Trennzeichen der Felder der Quelldatei an: `"\t"` für den Tabulator oder `"\n"` für *newline* (= Zeilenumbruch). Bei der Voreinstellung `""` trennt jede beliebige Menge an "white space" (= Leerraum) die Felder.

- `dec` bestimmt das in der Datei für den Dezimalpunkt verwendete Zeichen und ist auf `"."` voreingestellt.
- `skip` gibt die Zahl der Zeilen an, die am Beginn der Quelldatei vor dem Einlesen übersprungen werden sollen (wie beispielsweise Titel- oder Kommentarzeilen). Voreinstellung ist `skip = 0`, sodass ab der ersten Zeile gelesen wird.
- `flush = TRUE` veranlasst, dass jedes Mal nach dem Lesen des letzten in der `what`-Liste geforderten Feldes in einem Datensatz der Rest der Zeile ignoriert wird. Man kann so hinter diesem letzten Feld in der Quelldatei z. B. Kommentare erlauben, die von `scan()` nicht gelesen werden. Die Voreinstellung `flush = FALSE` sorgt dafür, dass jede Zeile vollständig gelesen wird.

(Für weitere Details siehe die Online-Hilfe.)

Die folgenden Beispiele sollen verdeutlichen, was mit `scan()` möglich ist. Dazu verwenden wir zwei ASCII-Textdateien namens `SMSA0` und `SMSAID0`, von denen hier jeweils ein Ausschnitt zu sehen ist (und mit denen wir uns noch öfter beschäftigen werden). Die Bedeutung der darin enthaltenen Daten ist auf Seite 63 etwas näher erläutert.

Zunächst ein Ausschnitt aus der Datei `SMSA0`. Darin sind die ersten vier Zeilen Titel- bzw. Leerzeilen und erst ab Zeile fünf stehen die eigentlichen Daten, deren Leerräume aus verschieden vielen Leerzeichen ("blanks") bestehen:

```
Datensatz "SMSA" (Standardized Metropolitan Settlement Areas) aus
Neter, Wasserman, Kuttner: "Applied Linear Models".

ID   2    3    4     5    6     7     8    9      10    11     12
  1  1384 9387  78.1 12.3 25627 69678 50.1 4083.9 72100 709234 1
  2  4069 7031  44.0 10.0 15389 39699 62.0 3353.6 52737 499813 4
....
141   654  231  28.8  3.9   140  1296 55.1   66.9  1148  15884 3
```

Es folgt ein Ausschnitt aus der Datei `SMSAID0`. Darin beinhalten die Daten selbst Leerräume (nämlich die "blanks" in den Städtenamen und vor den Staatenabkürzungen) und die Datenfelder sind durch jeweils genau einen (unsichtbaren) Tabulator `\t` getrennt:

```
Identifikationscodierung (Spalte ID) des Datensatzes "SMSA" aus
Neter, Wasserman, Kuttner: "Applied Linear Models".
1    NEW YORK, NY         48  NASHVILLE, TN     95   NEWPORT NEWS, VA
2    LOS ANGELES, CA      49  HONOLULU, HI      96   PEORIA, IL
....
19   SAN DIEGO, CA        66  WILMINGTON, DE    113  COLORADO SPRINGS, CO
....
47   OKLAHOMA CITY, OK    94  LAS VEGAS, NV     141  FAYETTEVILLE, NC
```

Wir gehen im Folgenden davon aus, dass sich die Dateien `SMSA0` und `SMSAID0` in dem Verzeichnis befinden, in dem **R** gestartet wurde, sodass sich eine Pfadangabe erübrigt.

Daten einlesen mit `scan()`	

`> (smsa <- scan( "SMSA0", skip = 4))`
`Read 1692 items`
`  [1]  1.0  1384.0  9387.0`
`  [3]  78.1  12.3  25627.0`
`  ....`
`[1690] 1148.0  15884.0  3.0`
	In der Datei `SMSA0` werden wegen `skip = 4` die ersten vier Zeilen ignoriert und der gesamte danach folgende Inhalt zeilenweise als `numeric`-Vektor (Voreinstellung) eingelesen, wobei jeglicher Leerraum (per Voreinstellung) als Trennzeichen fungiert und das Resultat (hier) als `smsa` gespeichert wird.

```
> scan( "SMSA0", nmax = 11, skip = 4)
Read 11 items
 [1]     1.0   1384.0    9387.0  78.1
 [5]    12.3  25627.0   69678.0  50.1
 [9]  4083.9  72100.0  709234.0
```

Wegen des Arguments `nmax = 11` werden (wg. `skip = 4` ab der fünften Zeile) lediglich die ersten 11 Felder eingelesen. (Ohne Zuweisung werden die Daten nicht gespeichert, sondern nur am **R**-Prompt ausgegeben.)

```
> scan( "SMSA0", what = list( ID = "",0,0,0,0,
+ 0,0,0,0,0,0,Code = ""), nmax = 2, skip = 4)
Read 2 records
$ID:
[1] "1" "2"

[[2]]:
[1] 1384 4069

[[3]]:
[1] 9387 7031

[[4]]:
[1] 78.1 44.0

[[5]]:
[1] 12.3 10.0

[[6]]:
[1] 25627 15389

[[7]]:
[1] 69678 39699

[[8]]:
[1] 50.1 62.0

[[9]]:
[1] 4083.9 3353.6

[[10]]:
[1] 72100 52737

[[11]]:
[1] 709234 499813

$Code:
[1] "1" "4"
```

Die Liste, die hier `what` zugewiesen wird, spezifiziert, dass jeder Datensatz aus 12 Feldern besteht und das erste Feld jeweils den Modus `character` hat, gefolgt von 10 `numeric`-Feldern und wiederum einem `character`-Feld. Die (wg. `skip = 4` ab der fünften Zeile) einzulesenden Felder der `nmax = 2` Datensätze (!) werden demnach so interpretiert, wie die Komponenten des `what`-Arguments beim zyklischen Durchlauf. Das Ergebnis ist eine Liste von Vektoren des Modus' der entsprechenden `what`-Komponente. Ihre Komponenten haben dabei die Namen, die ihnen im `what`-Argument (evtl.) gegeben wurden.

```
> scan( "SMSAID0", what = list( Nr1 = 0,
+ City1 = "", Nr2 = 0, City2 = "", Nr3 = 0,
+ City3 = ""), nmax = 2, sep = "\t", skip = 2)
Read 2 records
$Nr1
[1] 1 2

$City1
[1] "NEW YORK, NY"    "LOS ANGELES, CA"

$Nr2
[1] 48 49

$City2
[1] "NASHVILLE, TN" "HONOLULU, HI"

$Nr3
[1] 95 96

$City3:
[1] "NEWPORT NEWS, VA" "PEORIA, IL"
```

Hier werden in der Datei `SMSAID0` (wg. `skip = 2` ab der dritten Zeile) sechs Felder pro Datensatz erwartet, die abwechselnd den Modus `numeric` und `character` haben sowie durch jeweils genau einen Tabulator als Trennzeichen separiert sind (`sep = "\t"`). Es sollen `nmax = 2` Datensätze eingelesen werden und die Komponentennamen der Ergebnisliste lauten `Nr1`, `City1`, `Nr2`, `City2`, `Nr3` und `City3`. (Auch hier werden keine Daten gespeichert, sondern nur ausgegeben.)

3.1.2 Die Beispieldaten "SMSA"

Die von uns verwendeten SMSA-Daten stammen aus [66, Neter, Wasserman und Kuttner (1990)]. Hier die dort zu findenden Hintergrundinformationen und Beschreibungen. Zitat:

This data set provides information for 141 large Standard Metropolitan Statistical Areas (**SMSA**s) in the United States. A standard metropolitan statistical area includes a city (or cities) of specified population size which constitutes the central city and the county (or counties) in which it is located, as well as contiguous counties when the economic and social relationships between the central and contiguous counties meet specified criteria of metropolitan character and integration. An SMSA may have up to three central cities and may cross state lines.

Each line of the data set has an identification number and provides information on 11 other variables for a single SMSA. The information generally pertains to the years 1976 and 1977. The 12 variables are:

Variable Number	*Variable Name*	*Description*
1	Identification number	1 – 141
2	Land area	In square miles
3	Total population	Estimated 1977 population (in thousands)
4	Percent of population in central cities	Percent of 1976 SMSA population in central city or cities
5	Percent of population 65 or older	Percent of 1976 SMSA population 65 years old or older
6	Number of active physicians	Number of professionally active nonfederal physicians as of December 31, 1977
7	Number of hospital beds	Total number of beds, cribs, and bassinettes during 1977
8	Percent high school	Percent of adult population (persons 25 years old or older) who completed 12 or more years of school, according to the 1970 Census of the Population
9	Civilian labor force	Total number of persons in civilian labor force (person 16 years old or older classified as employed or unemployed) in 1977 (in thousands)
10	Total personal income	Total current income received in 1976 by residents of the SMSA from all sources, before deduction of income and other personal taxes but after deduction of personal contributions to social security and other social insurance programs (in millions of dollars)
11	Total serious crimes	Total number of serious crimes in 1977, including murder, rape, robbery, aggravated assault, burglary, larceny-theft, and motor vehicle theft, as reported by law enforcement agencies
12	Geographic region	Geographic region classification is that used by the U.S. Bureau of the Census, where: 1 = NE, 2 = NC, 3 = S, 4 = W

Data obtained from: U.S. Bureau of the Census, *State and Metropolitan Area Data Book, 1979* (a Statistical Abstract Supplement).

Zitat Ende.

3.1.3 Die Funktion `read.table()` und ihre Verwandten

Eine leistungsfähigere (aber etwas langsamere) Funktion, die eine automatische Datenmodi-Erkennung versucht und als Resultat einen Data Frame liefert, ist `read.table()`. Sie liest Daten aus einer ASCII-Textdatei, falls diese darin im Tabellenformat angelegt sind. Die unvollständige Syntax von `read.table()` mit ihren Voreinstellungen lautet:

```
read.table( file, header = FALSE, sep = "", dec = ".", row.names, col.names,
            as.is = FALSE, nrows = -1, skip = 0)
```

Zur Bedeutung der aufgeführten Argumente:

- `file`: Erwartet in Hochkommata den Namen (nötigenfalls mit Pfadangabe) der Quelldatei, aus der die Daten gelesen werden sollen. (Es kann sogar eine vollständige URL sein!) Jede Zeile der Quelldatei ergibt eine Zeile im resultierenden Data Frame.

- Ist `header = TRUE`, so wird versucht, die erste Zeile der Quelldatei für die Variablennamen des resultierenden Data Frames zu verwenden. Voreinstellung ist `FALSE`, aber wenn sich in der Kopfzeile (= erste Zeile) der Datei ein Feld weniger befindet als die restliche Datei Spalten hat (und somit die Einträge in der ersten Dateizeile als Variablennamen interpretiert werden können), wird `header` automatisch auf `TRUE` gesetzt.

- `sep` erhält das Zeichen, durch das die Datenfelder in der Quelldatei getrennt sind (siehe bei `scan()`, Seite 60; Voreinstellung: Jede beliebige Menge Leerraum).

- `dec` bestimmt das in der Datei für den Dezimalpunkt verwendete Zeichen und ist auf `"."` voreingestellt.

- `row.names` ist eine optionale Angabe für Zeilennamen. Fehlt sie und hat die Kopfzeile ein Feld weniger als die Spaltenzahl der restlichen Datei, so wird die erste Tabellenspalte für die Zeilennamen verwendet (sofern sie keine Duplikate in ihren Elementen enthält, denn ansonsten gibt es eine Fehlermeldung). Diese Spalte wird dann als Datenspalte entfernt. Anderenfalls, wenn `row.names` fehlt, werden Zeilennummern als Zeilennamen vergeben.
  Durch die Angabe eines `character`-Vektors (dessen Länge gleich der eingelesenen Zeilenzahl ist) für `row.names` können die Zeilen explizit benannt werden. Die Angabe einer einzelnen Zahl wird als Spaltennummer bzw. die einer Zeichenkette als Spaltenname interpretiert und die betreffende Spalte der Tabelle wird als Quelle für die Zeilennamen festgelet. Zeilennamen, egal wo sie herkommen, müssen eindeutig sein!

- `col.names` ist eine optionale Angabe für Variablennamen. Fehlt sie, so wird versucht, die Information in der Kopfzeile der Quelldatei für die Variablennamen zu nutzen (so, wie bei `header` beschrieben). Wenn dies nicht funktioniert, werden die Variablen mit den Namen `V1`, `V2` usw. (bis zur Nummer des letzten Feldes) versehen.
  Durch die Angabe eines `character`-Vektors (passender Länge) für `col.names` können die Variablen explizit benannt werden. Variablennamen, egal wo sie herkommen, müssen eindeutig sein!

- `as.is = FALSE` (Voreinstellung) bedeutet, dass alle Spalten, die nicht in `logical`, `numeric` oder `complex` konvertierbar sind, zu Faktoren werden. Ein `logical`-, `numeric`- oder `character`-Indexvektor für `as.is` gibt an, welche Spalten *nicht* in Faktoren umgewandelt werden, sondern `character` bleiben sollen. Beachte, dass sich dieser Indexvektor auf *alle* Spalten der Quelldatei bezieht, also auch auf die der Zeilennamen (falls vorhanden).

- `nrows` ist die Maximalzahl einzulesender Zeilen. Negative Werte führen zum Einlesen der gesamten Quelldatei. (Die Angabe eines Wertes, auch wenn er etwas größer ist als die

tatsächliche Zeilenzahl der Quelldatei, kann den Speicherplatzbedarf des Lesevorgangs reduzieren.)

- `skip` ist die am Beginn der Quelldatei zu überspringende Zeilenzahl (siehe bei `scan()`, Seite 61; Voreinstellung: `skip = 0`, sodass ab der ersten Zeile gelesen wird).

Für Anwendungsbeispiele mögen die Dateien `SMSA1`, `SMSA2`, `SMSA3` und `SMSAID` dienen, von denen unten jeweils ein Ausschnitt gezeigt ist. `SMSA1` bis `SMSA3` enthalten im Wesentlichen dieselben Daten; lediglich das Vorhandensein einer Überschriftszeile bzw. die Einträge in der letzten Spalte unterscheidet/n die Dateien. Ihre Spalten sind durch den (unsichtbaren) Tabulator getrennt. Die Datei `SMSAID` enthält bis auf die einleitenden Kommentarzeilen dieselben Informationen wie `SMSAID0` von Seite 61; die „Spalten“ sind auch hier durch einzelne (unsichtbare) Tabulatoren getrennt.

Die Datei SMSA1:

```
1       1384    9387    78.1    12.3    ....    72100   709234  1
2       4069    7031    44.0    10.0    ....    52737   499813  4
3       3719    7017    43.9    9.4     ....    54542   393162  2
....
141     654     231     28.8    3.9     ....    1148    15884   3
```

Die Datei SMSA2:

```
        LArea   TPop    CPop    OPop    ....    PIncome SCrimes GReg
1       1384    9387    78.1    12.3    ....    72100   709234  1
2       4069    7031    44.0    10.0    ....    52737   499813  4
3       3719    7017    43.9    9.4     ....    54542   393162  2
....
```

Die Datei SMSA3:

```
ID      LArea   TPop    CPop    OPop    ....    PIncome SCrimes GReg
1       1384    9387    78.1    12.3    ....    72100   709234  NE
2       4069    7031    44.0    10.0    ....    52737   499813  W
3       3719    7017    43.9    9.4     ....    54542   393162  NC
....
```

Die Datei SMSAID:

```
1    NEW YORK, NY         48  NASHVILLE, TN     95   NEWPORT NEWS, VA
2    LOS ANGELES, CA      49  HONOLULU, HI      96   PEORIA, IL
....
47   OKLAHOMA CITY, OK    94  LAS VEGAS, NV     141  FAYETTEVILLE, NC
```

Es folgen verschiedene Möglichkeiten, die Dateien `SMSA1`, `SMSA2`, `SMSA3` sowie `SMSAID` einzulesen. Dabei gehen wir davon aus, dass diese in dem Verzeichnis liegen, in dem **R** gestartet wurde, sodass sich eine Pfadangabe erübrigt.

ASCII-**Dateien einlesen mit** `read.table()`	

`> read.table( "SMSA1")`
`  V1  V2  V3  V4 .... V12`
`1  1 1384 9387 78.1 ....  1`
`2  2 4069 7031 44.0 ....  4`
`....`
	`SMSA1` enthält Daten in Form einer Tabelle ohne Kopfzeile. Das Resultat von `read.table()` ist ein Data Frame mit Nummern als Zeilenbenennung und `V1` bis `V12` als Variablennamen. (Ohne Zuweisung werden die eingelesenen Daten nicht gespeichert, sondern nur am **R**-Prompt ausgegeben.)

```
> read.table( "SMSA1", row.names = 1)
       V2   V3    V4 .... V12
1    1384 9387  78.1 ....   1
2    4069 7031  44.0 ....   4
....
```

Durch `row.names =` n wird die n-te Spalte der Quelldatei zur Zeilenbenennung ausgewählt und (offenbar *nach* dem Einlesen) als Datenspalte entfernt (beachte die Variablennamen).

```
> read.table( "SMSA1", row.names = 1,
+ col.names = LETTERS[ 1:12])
        B    C     D .... L
1    1384 9387  78.1 .... 1
2    4069 7031  44.0 .... 4
....
```

Mittels `col.names` werden die Spalten *vor* dem Einlesen explizit benannt. Zeilennamen werden dann der `row.names`-Spalte der Datei entnommen.

```
> read.table( "SMSA2")
    LArea TPop  CPop .... GReg
1    1384 9387  78.1 ....    1
2    4069 7031  44.0 ....    4
....
```

`SMSA2` hat eine Kopfzeile mit einem Eintrag weniger als die Zeilen der restlichen Tabelle, weswegen ihre Einträge zu Variablennamen werden. Automatisch wird die erste Spalte der Quelldatei zur Zeilenbenennung ausgewählt und als Datenspalte entfernt.

```
> (SMSA <- read.table( "SMSA3",
+ header = TRUE, row.names = "ID"))
    LArea TPop  CPop .... GReg
1    1384 9387  78.1 ....   NE
2    4069 7031  44.0 ....    W
....

> sapply( SMSA, class)
    LArea      TPop      CPop      OPop
"integer" "integer" "numeric" "numeric"
    APhys     HBeds    HSGrad   CLForce
"integer" "integer" "numeric" "numeric"
  PIncome  SCrimes      GReg
"integer" "integer"  "factor"
```

`SMSA3` enthält in allen Zeilen gleich viele Einträge, weswegen durch `header = TRUE` die Verwendung der Kopfzeileneinträge als Variablennamen „erzwungen“ werden muss. Wegen `row.names = "ID"` ist die Spalte „ID“ der Quelldatei zur Zeilenbenennung ausgewählt und als Datenspalte entfernt geworden. Außerdem enthält die letzte Dateispalte nicht-numerische Einträge, sodass sie zu einem Faktor konvertiert wird, wie die Anwendung von `class()` zeigt.

Im folgenden Beispiel wird die ASCII-Datei `SMSAID` eingelesen, deren „Datenspalten“ durch den Tabulator getrennt sind (`sep = "\t"`). Dabei werden die Variablen des resultierenden Data Frames mittels `col.names` explizit benannt. Außerdem sollen die Spalten 2, 4 und 6 der Quelldatei, die dort vom Modus `character` sind, *nicht* zu Faktoren konvertiert werden (wie für `as.is` angegeben):

```
> (SMSAID <- read.table( "SMSAID", sep = "\t", col.names = c( "Nr1", "City1",
+ "Nr2", "City2", "Nr3", "City3"), as.is = c( 2,4,6)))
   Nr1              City1 Nr2            City2 Nr3                City3
1    1       NEW YORK, NY  48    NASHVILLE, TN  95     NEWPORT NEWS, VA
2    2    LOS ANGELES, CA  49     HONOLULU, HI  96           PEORIA, IL
....
47  47  OKLAHOMA CITY, OK  94    LAS VEGAS, NV 141     FAYETTEVILLE, NC

> sapply( SMSAID, class)
        Nr1       City1         Nr2       City2         Nr3       City3
  "integer" "character"   "integer" "character"   "integer" "character"
```

Hinweise: (Für Details siehe die Online-Hilfe.)

- Zu `read.table()` gibt es die vier „spezialisierte“ Varianten `read.csv()`, `read.csv2()`, `read.delim()` und `read.delim2()`, die genau wie `read.table()` funktionieren und lediglich andere Voreinstellungen haben:
  `read.csv()` ist für das Einlesen von “comma separated value”-Dateien (CSV-Dateien) voreingestellt und `read.csv2()` ebenso, wenn in Zahlen das Komma anstelle des Dezimalpunkts verwendet werden und gleichzeitig das Semikolon als Feldtrenner fungiert.
  Völlig analog wirken `read.delim()` und `read.delim2()`, außer dass sie als Feldtrenner den Tabulator erwarten.

  Beachte, dass in allen vier Fällen `header = TRUE` gesetzt ist (sowie `fill = TRUE` gilt und das Kommentarzeichen deaktiviert ist; zur Bedeutung dieses Sachverhalts siehe die Online-Hilfe).

- `read.fwf()` erlaubt das Einlesen von ASCII-Dateien, die im tabellarischen “**f**ixed **w**idth **f**ormat” arrangiert sind.

3.2 Datenausgabe am Bildschirm und ihre Formatierung: `print()`, `cat()` & Helferinnen

Für die Datenausgabe am Bildschirm stehen in **R** z. B. die Funktionen `print()` und `cat()` zur Verfügung.

- `print()`: Bisher bekannt ist, dass die Eingabe eines Objektnamens am **R**-Prompt den „Inhalt“ bzw. Wert des Objektes als Antwort liefert. Intern wird dazu automatisch die Funktion `print()` – genauer eine geeignete „Methode“ von `print()` – aufgerufen, die „weiß“, wie der Objektinhalt bzw. -wert für eine Bildschirmdarstellung zu formatieren ist, und zwar je nachdem, ob es ein Vektor oder eine Matrix welchen Modus’ auch immer ist, eine Liste oder ein Data Frame oder – was wir noch kennenlernen werden – z. B. eine Funktion oder das Ergebnis einer Funktion zur Durchführung eines Hypothesentests.

  Der explizite Aufruf von `print()` erlaubt i. d. R. die Modifikation gewisser Voreinstellungen der Ausgabe, wie z. B. die minimale Anzahl der gezeigten signifikanten Ziffern durch das `integer`-Argument `digits` (1 - 22) oder die *Nicht*-Verwendung von Anführungszeichen bei Zeichenketten mittels des `logical`-Arguments `quote = FALSE`.

  Notwendig ist die explizite Verwendung von `print()`, wenn aus dem Rumpf einer (benutzereigenen) Funktion oder aus einer `for`-Schleife heraus z. B. zu „Protokollzwecken“ eine Ausgabe an den Bildschirm erfolgen soll. (Details hierzu folgen in Kapitel 6.)

- `cat()` liefert eine viel rudimentärere und weniger formatierte Ausgabe der Werte der an sie übergebenen (auch mehreren) Objekte. Sie erlaubt dem/der Benutzer/in aber die eigenständigere Formatierung und außerdem die Ausgabe der Objektwerte in eine Datei, falls an `cat()`’s `character`-Argument `file` ein Dateiname übergeben wird. Existiert die Datei schon, wird sie entweder *ohne Warnung* überschrieben oder die aktuelle Ausgabe an ihren bisherigen Inhalt angehängt, wozu jedoch `cat()`’s `logical`-Argument `append` auf `TRUE` zu setzen ist.

  Per Voreinstellung fügt `cat()` an jedes ausgegebene Element ein Leerzeichen an, was durch ihr `character`-Argument `sep` beliebig geändert werden kann.

  Beachte: Sehr nützliche Sonderzeichen (“escape character”), die `cat()` „versteht“, sind (z. B.) der Tabulator `\t` und der Zeilenumbruch `\n`. Den “backslash” \ erhält man durch `\\`.

Beachte: Die Funktionen `format()`, `formatC()` und `prettyNum()` können sehr nützlich sein, wenn (hauptsächlich `numeric`-)Objekte für die Ausgabe „schön“ formatiert werden sollen.

Zu allen oben aufgeführten Funktionen ist – wie immer – eine (auch wiederholte) Konsultation der Online-Hilfe zu empfehlen.

3.3 Datenexport in eine Datei: `sink()`, `write()` und `write.table()`

Mit `sink()` kann man die Bildschirmausgabe „ein zu eins“ solange in eine anzugebende Datei „umleiten“ – also auch die Ausgabe von `print()` – bis ein erneuter `sink()`-Aufruf (ohne Argument) die Umleitung wieder aufhebt.
Für den Datenexport in eine Datei stehen ebenfalls verschiedene Funktionen zur Verfügung: `cat()` unter Verwendung ihres Argumentes `file`, wie bereits in Abschnitt 3.2 erwähnt, und `write()` als „Umkehrungen“ von `scan()` sowie `write.table()` als „Umkehrung“ der Funktion `read.table()`. (Für nähere Informationen zu all diesen Funktionen siehe die Online-Hilfe.)

Nur ein paar kurze Beispiele. Angenommen, wir haben

```
> alter
 [1] 35 39 53 14 26 68 40 56 68 52 19 23 27 67 43
> gewicht
 [1] 82 78 57 43 65 66 55 58 91 72 82 83 56 51 61
> rauchend
 [1] "L" "G" "X" "S" "G" "G" "X" "L" "S" "X" "X" "L" "X" "X" "S"
> m <- cbind( alter, gewicht)     # also eine (15x2)-Matrix
```

Dann kann der Export der obigen Vektoren und der Matrix `m` in Dateien wie folgt geschehen:

Umleiten der R-Bildschirmausgabe in eine Datei mit `sink()`

```
> sink( "RauchTab");   table( rauchend);   sink()
```

Erzeugt, falls sie noch nicht existiert, die Datei `RauchTab` (in dem Verzeichnis, in dem **R** gestartet wurde) und öffnet sie zum Schreiben. Die Ausgaben *aller* folgenden Befehle – hier nur `table(rauchend)` – werden „eins zu eins“ in diese Datei geschrieben, und zwar bis zum nächsten `sink()`-Befehl mit leerer Argumenteliste. Ab da geschieht wieder normale Bilschirmausgabe. (Voreinstellung von `sink()` ist, dass der ursprüngliche Inhalt der Zieldatei überschrieben wird. Mit dem Argument `append = TRUE` erreicht man, dass die Ausgaben am Zieldateiende angehängt werden.) Der Inhalt der (vorher leeren) Datei `RauchTab` ist nun:

```
 G L S X
 3 3 3 6
```

Daten mit `write()` in eine Datei ausgeben

`> write( alter,`
`+ file = "Alter",`
`+ ncolumns = 5)`
	Schreibt den Vektor `alter` zeilenweise in die ASCII-Datei `Alter` (in dem Verzeichnis, worin **R** gestartet wurde), und zwar in der 5-spaltigen Form
	`35 39 53 14 26`
	`68 40 56 68 52`
	`19 23 27 67 43`.

`> write( rauchend,` `+ file = "Rauch")`	Der `character`-Vektor `rauchend` wird als eine Spalte in die ASCII-Datei `Rauch` geschrieben.

`> write( t( m),`
`+ file = "Matrix",`
`+ ncolumns = ncol( m))`
	Die (15×2)-Matrix `m` wird (nur so) als (15×2)-Matrix in die Datei `Matrix` geschrieben. Beachte die Notwendigkeit des Transponierens von `m`, um *zeilen*weise in die Zieldatei geschrieben zu werden, denn intern werden Matrizen *spalten*weise gespeichert und demzufolge auch so ausgelesen!

Daten mit `write.table()` in eine ASCII-Datei ausgeben

Aus

```
> cu.summary[ 1:3,]
                Price Country Reliability Mileage  Type
Acura Integra 4 11950   Japan Much better      NA Small
Dodge Colt 4     6851   Japan        <NA>      NA Small
Dodge Omni 4     6995     USA  Much worse      NA Small
```

wird durch

```
> write.table( cu.summary[ 1:3,], "cu.txt")
```

die ASCII-Datei `cu.txt` in dem Verzeichnis erzeugt, in dem **R** gerade läuft. Sie sieht dann wie folgt aus:

```
"Price" "Country" "Reliability" "Mileage" "Type"
"Acura Integra 4" 11950 "Japan" "Much better" NA "Small"
"Dodge Colt 4" 6851 "Japan" "NA" NA "Small"
"Dodge Omni 4" 6995 "USA" "Much worse" NA "Small"
```

`write.table()` wandelt ihr erstes Argument zunächst in einen Data Frame um (falls es noch keiner ist) und schreibt diesen dann zeilenweise in die angegebene Zieldatei. Dabei werden die Einträge einer jeden Zeile per Voreinstellung durch ein Leerzeichen (`" "`) getrennt. Beachte die vom Variablentyp abhängige, unterschiedliche Verwendung von Anführungszeichen im Zusammenhang mit NAs, wie z. B. beim Faktor `Reliability` und der `numeric`-Variablen `Mileage`. Mit Hilfe des Arguments `sep` kann das Trennzeichen für die Einträge einer jeden Zeile beliebig festgelegt werden, wie z. B. bei `sep = "\t"` der Tabulator verwendet wird.

Für sehr große Data Frames mit sehr vielen Variablen kann die Funktion `write.table()` recht langsam sein. Die Funktion `write.matrix()` im **R**-Paket `MASS` ist für große `numeric`-Matrizen oder Data Frames, die als solche darstellbar sind, effizienter; siehe ihre Online-Hilfe.

3.4 Datenausgabe in Dateien mit Formatierung in TEX, HTML oder im "Open-Document-Format" (ODF)

Auf den ersten Blick – und auch auf den zweiten – sind die von **R** gelieferten Ausgaben, seien es Datenstrukturen wie Data Frames oder Resultatelisten von Funktionen wie wir sie noch zuhauf kennenlernen werden, layouttechnisch wenig berauschend, was der diesbzgl. mangelnden Leistungsfähigkeit der **R**-Console geschuldet ist.

Allerdings gibt es sehr leistungsfähige Werkzeuge, **R**-Strukturen in LATEX- oder HTML-Code „übersetzen" zu lassen, um diesen danach in entsprechende Dokumente einzubinden. Speziell für Objektdarstellungen in LATEX steht z. B. im **R**-Paket `Hmisc` die Funktion `latex()` zur Verfügung. Soll hingegen HTML-Code erzeugt werden, ist z. B. die Funktion `HTML()` des Pakets `R2HTML` verwendbar. Geht es im Wesentlichen „nur" um die Darstellung von tabellen-ähnlichen **R**-Objekten, kann z. B. `xtable()` aus dem gleichnamigen Paket Ausgaben wahlweise in *beiden* Formaten liefern.

Für weitere Informationen zu obigem und zu dem mit diesem Thema verwandten, sehr interessanten Aspekt der (halb-)automatischen Reportgenerierung in TEX (mit Hilfe der Funktion `Sweave()`, die bereits in "base-**R**" vorhanden ist; siehe Online-Hilfe) oder im ODF (mit Hilfe von `odfWeave()` des genauso benannten **R**-Pakets) siehe auch den Task View "Reproducible Research" auf CRAN (`http://cran.r-project.org/` → Task Views → Reproducible Research).

Alles in allem ist aber zu warnen, dass das Einarbeiten in die oben erwähnten Werkzeuge eine gewisse Zeit in Anspruch nehmen kann.

4 Elementare explorative Grafiken

Wir geben einen Überblick über einige in **R** zur Verfügung stehende grafische Möglichkeiten, univariate oder bi- bzw. multivariate Datensätze gewissermaßen „zusammenfassend“ darzustellen. Die hierbei zum Einsatz kommenden Verfahren hängen nicht nur von der Dimensionalität, sondern auch vom Skalenniveau der Daten ab.

Die im Folgenden vorgestellten Funktionen erlauben dank ihrer Voreinstellungen in den meisten Anwendungsfällen ohne großen Aufwand die schnelle Erzeugung relativ guter und aussagefähiger Grafiken. Andererseits bieten alle Funktionen der Benutzerin und dem Benutzer viele weitere, zum Teil sehr spezielle Einstellungsmöglichkeiten für das Layout der Grafiken. Die Nutzung dieser Optionen kann dann jedoch zu ziemlich komplizierten Aufrufen führen, deren Diskussion hier zu weit ginge. In Kapitel 7 „Weiteres zur elementaren Grafik“ gehen wir diesbezüglich auf einige Aspekte näher ein, aber detaillierte Informationen zur Syntax der Funktionsaufrufe und genauen Bedeutung der im Einzelnen zur Verfügung stehenden Funktionsargumente sollten der Online-Hilfe entnommen werden.

4.1 Grafikausgabe am Bildschirm und in Dateien

Um eine grafische Ausgabe am Bildschirm oder in eine Datei zu ermöglichen, ist für den ersten Fall zunächst ein Grafikfenster und für den zweiten eine Grafikdatei zu öffnen. Jedes grafische Ausgabemedium, egal ob Bildschirm-Grafikfenster oder Grafikdatei, wird “device” genannt. Für die Verwendung dieser Devices dienen (unter anderem) die folgenden Befehle:

Öffnen und Schließen von Grafik-Devices:	
`> X11()` `> windows()`	`x11()`, `X11()` und `windows()` (letzteres nicht unter Unix) öffnet bei jedem Aufruf ein neues Grafikfenster. Das zuletzt geöffnete Device ist das jeweils aktuell *aktive*, in welches die Grafikausgabe erfolgt.
`> dev.list()`	Listet Nummern und Typen aller geöffneten Grafik-Devices auf.
`> dev.off()`	Schließt das zuletzt geöffnete Grafik-Device *ordnungsgemäß*.
`> graphics.off()`	Schließt alle geöffneten Grafik-Devices auf einen Schlag.
`> postscript( file)` .... (Grafikbefehle)	Öffnet bei jedem Aufruf eine neue (EPS- (= “Encapsulated PostScript”-) kompatible) PostScript-Datei mit dem als Zeichenkette an `file` übergebenen Namen. Das zuletzt geöffnete Device ist das jeweils aktuelle, in welches die Grafikausgabe erfolgt.
`> dev.off()`	Schließt das zuletzt geöffnete Grafik-Device und *muss* bei einer PostScript-Datei unbedingt zur Fertigstellung verwendet werden, denn erst danach ist sie vollständig und korrekt interpretierbar.

`> pdf( file)`
.... (Grafikbefehle) ....
`> dev.off()`
	Völlig analog zu `postscript()`, aber eben für PDF-Grafikdateien.

Hinweise: Der Aufruf einer grafikproduzierenden Funktion, *ohne* dass vorher ein Grafik-Device explizit geöffnet wurde, aktiviert automatisch ein Grafik*fenster*, sodass ein `x11()`, `X11()` oder `windows()` für ein erstes Grafik*fenster* nicht nötig ist.
Die Funktionen `postscript()` und `pdf()` haben eine Vielzahl weiterer Argumente, die insbesondere dann Verwendung finden (können), wenn beabsichtigt ist, die Grafikdateien später z. B. in LaTeX-Dokumente einzubinden; wir verweisen hierfür auf die Online-Hilfe. Für einen Überblick über alle derzeit in **R** verfügbaren Formate grafischer Ausgabe ist `?Devices` hilfreich.

4.2 Explorative Grafiken für univariate Daten

In diesem Abschnitt listen wir einige Funktionen auf, die zur Darstellung und explorativen Analyse *univariater* Datensätze verwendet werden können. Zunächst geschieht dies für **(endlich)**

diskrete oder **nominal-** bzw. **ordinalskalierte Daten**, wo es im Wesentlichen um die Darstellung der beobachteten (absoluten oder relativen) Häufigkeiten der Werte geht. Dabei verwenden wir die Daten aus dem folgenden ...

Beispiel: In einer Studie zum Mundhöhlenkarzinom wurden von allen Patienten Daten über den bei ihnen aufgetretenen maximalen Tumordurchmesser in Millimeter (mm) und ihren „ECOG-Score“ erhoben. Letzterer ist ein qualitatives Maß auf der Ordinalskala 0, 1, 2, 3 und 4 für den allgemeinen physischen Leistungszustand eines Tumorpatienten (der von 0 bis 4 schlechter wird). Die Skala der Tumordurchmesser wurde in die vier Intervalle $(0, 20]$, $(20, 40]$, $(40, 60]$ und $(60, 140)$ eingeteilt.

Die rechts gezeigte `numeric`-Matrix `HKmat` enthält die gemeinsamen absoluten Häufigkeiten der gruppierten, maximalen Tumordurchmesser (pro Spalte) und der Realisierungen 0 bis 4 des ECOG-Scores (pro Zeile). Offenbar hat `HKmat` (durch sein `dimnames`-Attribut) selbsterklärend benannte Zeilen und Spalten.

```
> HKmat
      (0,20] (20,40] (40,60] (60,140)
ECOG0   1301    1976     699      124
ECOG1    173     348     189       59
ECOG2     69     157     104       34
ECOG3     16      27      18        7
ECOG4      5       6       4        2
```

Die Resultate der im Folgenden beschriebenen Funktionen zur grafischen Darstellung dieser Daten sind ab Seite 72 anhand von Beispiel-Plots zu bestaunen (hier “plot” (Engl.) = Diagramm).

Bemerkung: Die Verwendung deutscher Umlaute in der Textausgabe für Grafiken ist etwas komplizierter, wenn sie nicht auf der Tastatur vorkommen. In den folgenden Beispielen wird aus Demonstrationszwecken daher häufig Gebrauch von sogenannten „Escape-Sequenzen“ für die ASCII-Nummer (in Oktalsystemschreibweise) gemacht, um deutsche Umlaute in der Grafikausgabe zu generieren. Der Vollständigkeit halber listen wir die Escape-Sequenzen der deutschen Sonderzeichen in der folgenden Tabelle auf:

Sonderzeichen	ä	Ä	ö	Ö	ü	Ü	ß
Oktale Escape-Sequenz	`\344`	`\304`	`\366`	`\326`	`\374`	`\334`	`\337`

Hinweis: Dateien aus einer anderen „Locale“ (d. h. regional- und sprachspezifischen Einstellung) als der eigenen können auf einem „fremden“ Zeichensatz basieren, also eine andere Enkodierung (= “encoding”) für `character`-Vektoren besitzen. Für die Konversion von Enkodierungen kann die Information unter `?Encodings` und die Funktion `iconv()` hilfreich und nützlich sein.

4.2.1 Die Häufigkeitsverteilung diskreter Daten: Balken-, Flächen- und Kreisdiagramme sowie Dot Charts

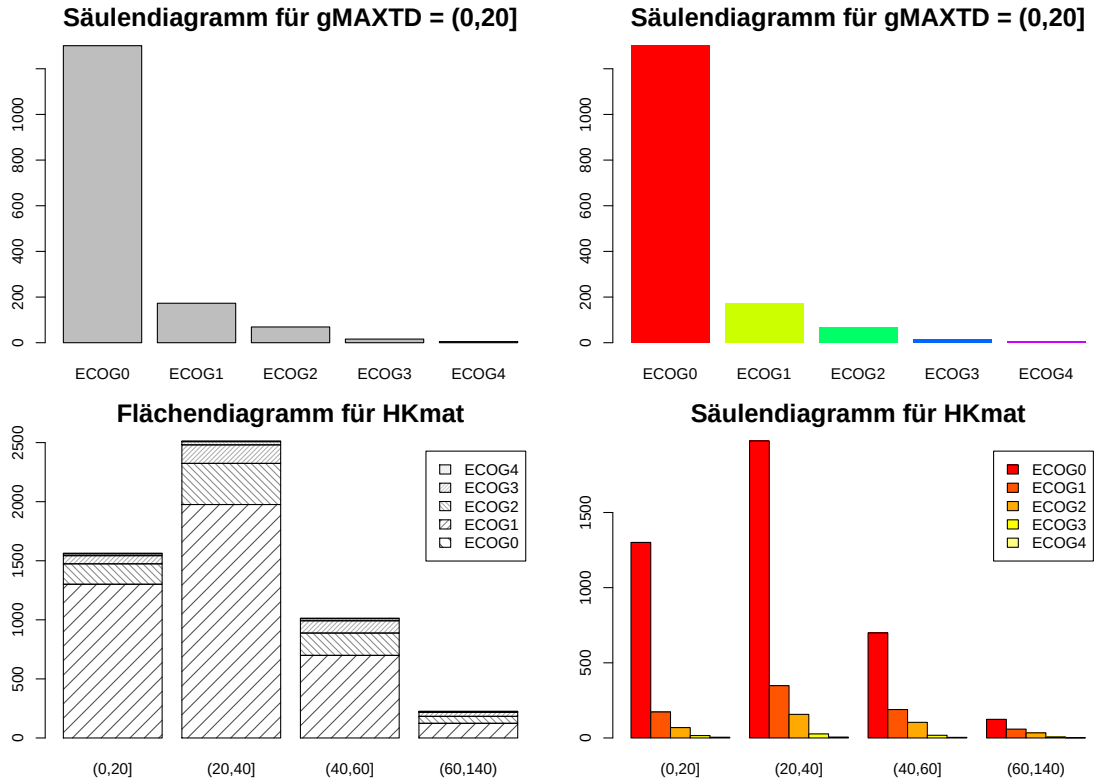
`> barplot( HKmat[, 1],`
`+ main = "S\344ulendiagramm`
`+ f\374r gMAXTD =(0,20]")`
 	
(Memo: `\374` ≙ ä, `\374` ≙ ü*)*
	`barplot()` mit einem Vektor als erstem Argument erzeugt ein **Balken-** (auch genannt **Säulen-**)**Diagramm** mit Balkenhöhen, die diesem Vektor (hier: `HKmat[, 1]`) entnommen werden. Die Balkenbeschriftung geschieht automatisch über das `names`-Attribut des Vektors, falls vorhanden (oder explizit mit dem nicht gezeigten Argument `names.arg`). Die Plot-Überschrift wird durch `main` festgelegt. (Siehe im Bild auf Seite 72 oben links.)

`> barplot( HKmat[, 1],`
`+ col = rainbow( 5),`
`+ border = NA, ....)`
	`col` erlaubt die Vergabe von Farben für die Balken (wobei hier durch `rainbow( 5)` fünf Farben aus dem Spektrum des Regenbogens generiert werden; siehe Online-Hilfe), `border = NA` schaltet die Balkenumrandung aus. (Im Bild auf Seite 72 oben rechts. Details zu derartigen und weiteren, allgemeinen Grafik-Argumenten folgen in Kapitel 7.)

`> barplot( HKmat,`
`+ angle = c( 45, 135),`
`+ density = (1:5)*10,`
`+ legend.text = TRUE,`
`+ main = "Flächendiagramm`
`+ für HKmat")`
	Eine Matrix als erstes Argument liefert für *jede Spalte* ein **Flächen-Diagramm**: Darin ist jeder Balken gemäß der Werte in der jeweiligen Matrixspalte vertikal in Segmente unterteilt. Deren Schraffur-Winkel und -Dichten werden durch `angle` bzw. `density` bestimmt. Balkenbeschriftung: Matrix*spalten*namen (oder `names.arg`). Wegen `legend.text = TRUE` wird aus den Matrix*zeilen*namen eine entsprechende Legende erzeugt (auch explizit angebbar). Überschrift: `main`. (Siehe im Bild unten links.)

`> barplot( HKmat,`
`+ beside = TRUE,`
`+ col = heat.colors( 5),`
`+ legend.text = TRUE, ....)`
	`beside = TRUE` lässt für eine Matrix nebeneinander stehende Balken entstehen, `col` steuert (zyklisch) die Farben innerhalb dieser Balkengruppen, `legend.text = TRUE` führt zur Legende. (Im Bild unten rechts.)

Hinweis: Die Online-Hilfe zu `barplot()` zählt weitere ihrer Argumente auf, u. a. das für Aussehen und Platzierung der Legende.

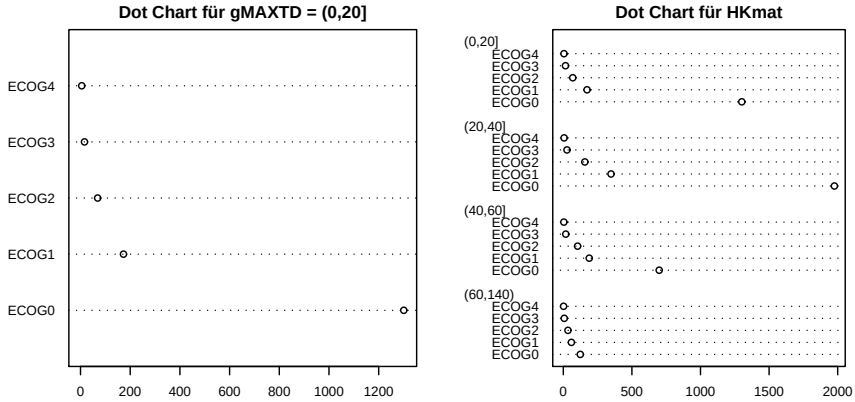


`> dotchart( HKmat[, 1],`
`+ main = "Dot Chart für`
`+ gMAXTD = (0,20]")`
	`dotchart()` erzeugt zu einem Vektor für die Werte seiner Elemente einen **"Dot Chart"**. (Dies ist eine im Uhrzeigersinn um 90 Grad gedrehte und auf das absolut Nötigste reduzierte Art von Balkendiagramm.) Die „zeilenweise" Chart-Beschriftung geschieht automatisch über das `names`-Attribut des Vektors (oder explizit mit dem Argument `labels`). Überschrift: `main`. (Bild auf Seite 73 oben links.)

`> dotchart( HKmat,`
`+ main = "Dot Chart für`
`+ HKmat")`
	Aus einer Matrix wird für jede Spalte ein gruppierter Dot Chart angefertigt. Dessen gruppeninterne „Zeilen"-Beschriftung geschieht einheitlich durch die Matrix*zeilen*namen. Die Charts werden in ein gemeinsames Koordinatensystem eingezeichnet und durch die Matrix*spalten*namen beschriftet. Überschrift: `main`. (Bild auf Seite 73 oben rechts.)

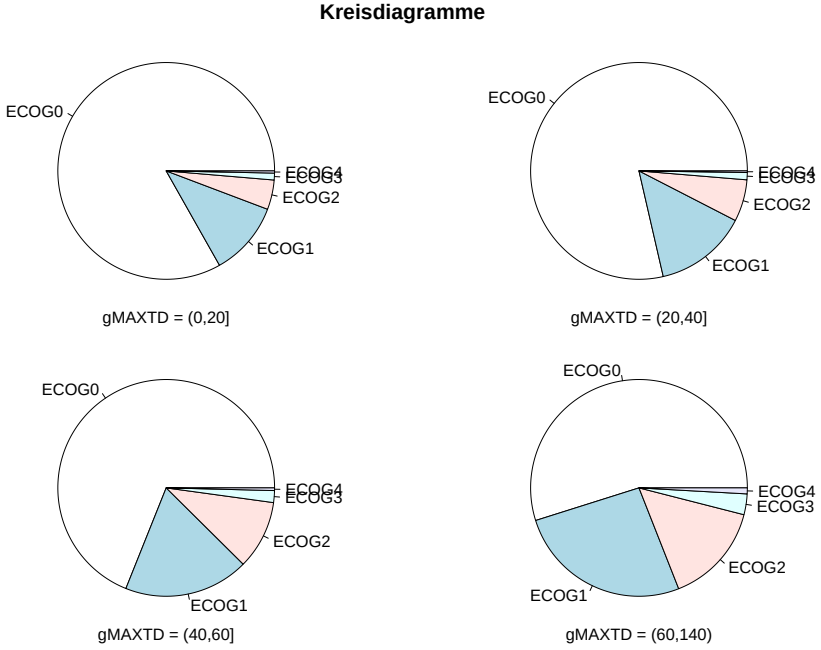
Hinweise:

- Für einen Vektor erlaubt das einen Faktor erwartende `dotchart()`-Argument `groups` die beliebige Gruppierung seiner Elemente gemäß des Faktors. Die Faktorlevels beschriften die Gruppen und mit dem weiteren `dotchart()`-Argument `labels` ließen sich die gruppeninternen Zeilen benennen. (Nicht gezeigt; siehe Online-Hilfe.)
- Die – eigentlich naheliegende – Übergabe des Resultates von `table()` (siehe §2.7.6) oder von `xtabs()` (siehe Online-Hilfe) führt im Falle einer *ein*dimensionalen Häufigkeitstabelle zu einer Warnmeldung, weil `dotchart()` einen numerischen Vektor (oder eine solche Matrix) erwartet, in den (bzw. die) ein `table`-Objekt erst noch konvertiert wird.




```
> pie( HKmat[, 1],
+ sub = paste( "gMAXTD =",
+  colnames( HKmat)[ 1]))
```

`pie()` erstellt zu einem Vektor (hier `HKmat[, 1]`) ein **Kreisdiagramm**, dessen Sektorenwinkel proportional zu den Werten der Vektorelemente sind. Die Einfärbung der Sektoren geschieht automatisch (oder explizit durch das Argument `col`) und ihre Benennung über das `names`-Attribut des Vektors (oder explizit mit dem Argument `labels`). (Mithilfe der Argumente `angle` und `density` können die Sektoren auch unterschiedlich schraffiert werden.) Einen Untertitel für den Plot liefert `sub` und `main` die Überschrift. (Siehe unten, Diagramm links oben. Die übrigen drei Diagramme wurden analog hergestellt, indem nacheinander die anderen Spalten von `HKmat` indiziert wurden.)



Bemerkung: Kreisdiagramme sind zur Informationsdarstellung oft weniger gut geeignet als Balkendiagramme oder Dot Charts, weswegen letztere vorzuziehen sind ([20, Cleveland (1985)]).

4.2.2 Die Verteilung metrischer Daten: Histogramme, Kern-Dichteschätzer, "stem-and-leaf"-Diagramme, Boxplots, "strip charts" und Q-Q-Plots

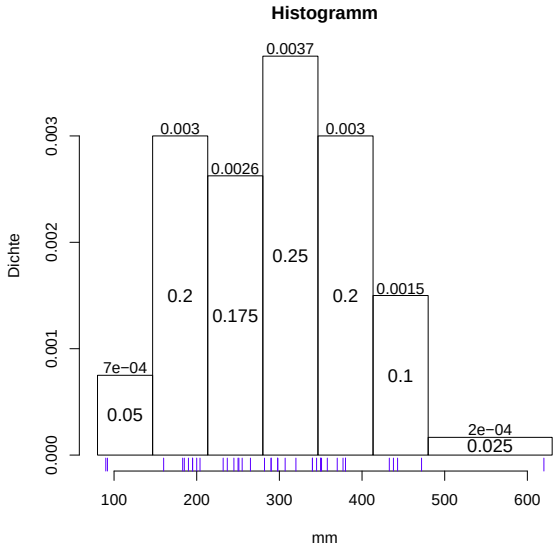
Für die im Folgenden verwendeten Beispieldaten

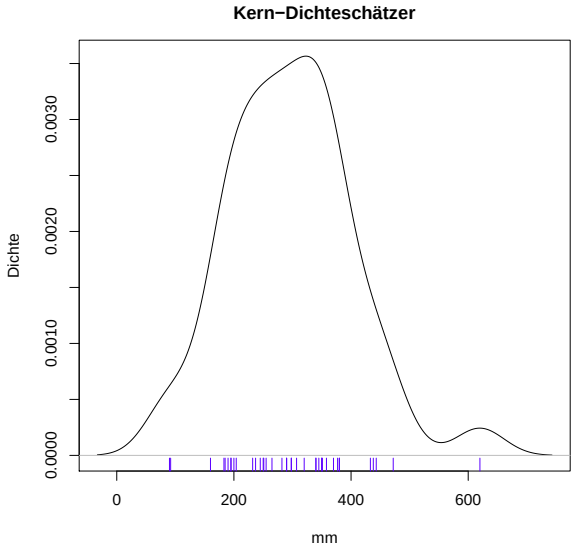
```
> X1 <- c( 298, 345, 183, 340, 350, 380, 190, 351, 443, 290, 160, 298, 185,
+          370, 245, 377,  92, 380, 195, 265, 232, 358, 290, 307, 433, 255,
+          320, 237, 472, 438, 250, 340, 204, 282, 195, 251,  90, 200, 350, 620)
```

findet sich in §4.3.2 eine Kurzbeschreibung.

`> brks <- c( seq( 80, 480,`
`+ length = 7), 630),`
`> hist( X1, freq = FALSE,`
`+ breaks = brks,`
`+ main = "Histogramm",`
`+ xlab = "mm", ylab = "Dichte")`
	`hist()` erzeugt zu einem Vektor (hier `X1`) ein wegen `freq = FALSE` auf 1 flächennormiertes **Histogramm** für die Werte in diesem Vektor. (Bei `freq = TRUE` sind die Balkenhöhen absolute Klassenhäufigkeiten.) Die Klasseneinteilung der x-Achse erfolgt gemäß der Angabe von Klassengrenzen für `breaks`; wenn sie fehlt, werden aus den Daten automatisch Grenzen bestimmt. Überschrift: `main`; Achsenbeschriftung: `xlab` und `ylab`. (Bild links)

`> plot( density( X1),`
`+ main = "Kern-Dichteschätzer",`
`+ xlab = "mm", ylab = "Dichte")`
	`density()` berechnet einen sogenannten (nicht-parametrischen) Kern-Schätzer für die Dichte der den Daten zugrundeliegenden Verteilung (und ist quasi ein „geglättetes" Histogramm, wobei der Grad der Glättung anhand eines gewissen (Optimalitäts-)Kriteriums datenabhängig gewählt wird). Die Zeichnung des Graphen der geschätzten Dichte muss explizit durch `plot()` angefordert werden (Bild rechts). (Weitere wenige Details in Abschnitt 6.1 oder nochmal in 8.3.)





Hinweis: Das gezeigte Histogramm wurde durch folgende Befehle mit den zusätzlichen (blauen) Markierungen der Rohdatenwerte sowie den Zusatzinformationen zu Säulenhöhen und -flächen versehen. (Der Plot des Kerndichteschätzer wurde nur durch `rug()` ergänzt.)

```
> histwerte <- hist( X1, ....(wie oben))
> rug( X1, col = "blue")
> with( histwerte, {
+  text( x = mids, y = density/2, cex = 1.2, labels = round( density * diff( brks), 4))
+  text( x = mids, y = density, pos = 3, offset = 0.1, labels = round( density, 4))
+  })
```

Zum Rückgabewert von `hist()` siehe ihre Online-Hilfe. `with()` wurde kurz schon in §2.10.7 auf S. 56 angesprochen und auf `text()` gehen wir kurz in Abschnitt 7.2 ein bzw. verweisen auf ihre Online-Hilfe.

`> stem( X1)`
`The decimal point is 2 digit(s)`
`to the right of the |`
`0 | 99`
`1 | 6899`
`2 | 00003455567899`
`3 | 001244555567888`
`4 | 3447`
`5 |`
`6 | 2`
 	
`> stem( X1, scale = 2)`
`The decimal point is 2 digit(s)`
`to the right of the |`
`0 | 99`
`1 |`
`1 | 6899`
`2 | 000034`
`2 | 55567899`
`3 | 001244`
`3 | 555567888`
`4 | 344`
`4 | 7`
`5 |`
`5 |`
`6 | 2`
	Ein **"stem-and-leaf"-Diagramm** ist eine halbgrafische Darstellung klassierter Häufigkeiten der Werte in einem Vektor, ähnlich der Art eines Histogramms. Allerdings gehen dabei die Werte selbst nicht „verloren" (im Gegensatz zum Histogramm), sondern werden so dargestellt, dass sie bis auf Rundung rekonstruierbar sind. (Siehe z. B. [77, Tukey (1977)].) Dazu werden die Daten aufsteigend sortiert und die führenden Stellen der Werte zur Klasseneinteilung verwendet. Sie liefert den Stamm ("stem") des Diagramms, der durch den senkrechten Strich „`|`" angedeutet wird. Hier sind es Hunderterklassen, was aus "`The decimal point is 2 digit(s) to the right of the |`" folgt. Die nächste Stelle (in diesem Fall also die – gerundete – Zehnerstelle) der Daten bildet die Blätter ("leafs"), die rechts am Stamm „angeheftet" werden. Der Strich markiert also die Grenze zwischen Stamm und Blättern (hier zwischen Hundertern und Zehnern).
	
	Das Argument `scale` erlaubt, die „Auflösung" des Wertebereichs einzustellen: Je größer sein Wert, desto mehr Klassen werden gebildet. (Dadurch wird auch die „Höhe" des Stamms gesteuert.)

Boxplots und "Strip Charts" dienen nicht nur der – mehr oder minder übersichtlich zusammenfassenden – grafischen Darstellung metrischer Daten, sondern auch der qualitativen Beurteilung von Verteilungsannahmen über ihren „Generierungsmechanismus". (Die auf der nächsten Seite folgenden Boxplots wurden hier zur Erläuterung zum Teil mit Zusatzinfomationen versehen.)

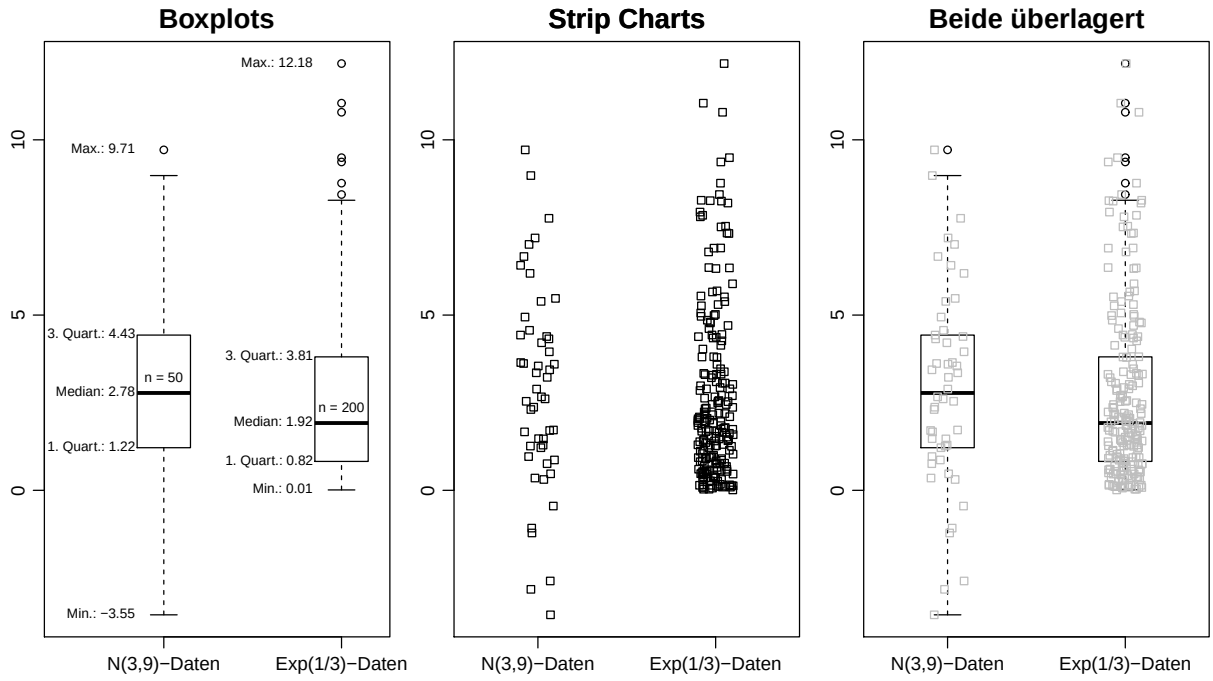
`> x1 <- rnorm( 50, 3, 3)` `> x2 <- rexp( 200, 1/3)`	(`x1` und `x2` erhalten 50 bzw. 200 pseudo-zufällige $\mathcal{N}(3, 3^2)$- bzw. $Exp(1/3)$-verteilte, künstliche Beispieldaten. Für Details siehe Kapitel 5.)
`> boxplot( x1)`	Ein `numeric`-Vektor als erstes Argument liefert den Boxplot nur für seinen Inhalt. (Keine Grafik gezeigt.)

`> boxplot( list( x1, x2),`
`+ names = c( "N(3,9)-Daten",`
`+ "Exp(1/3)-Daten"),`
`+ main = "Boxplots",`
`+ boxwex = 0.3)`
	`boxplot()` erzeugt für jede Komponente von `list( x1, x2)` einen **Boxplot** (siehe §4.2.3), in einem *gemeinsamen* Koordinatensystem nebeneinander angeordnet und durch die Zeichenketten in `names` benannt. Überschrift: `main`. Steuerung der relativen Breite der Boxen: `boxwex`. (Siehe im Bild auf Seite 76 links.)

`> boxplot( split(` `+ SMSA$SCrimes, SMSA$GReg))`	`split()` erzeugt eine für `boxplot()` geeignete Liste, indem sie ihr erstes Argument (`numeric`) gemäß des zweiten (`factor`) auf Listenkomponenten aufteilt. (Ohne Bild.)
`> boxplot( SCrimes ~ GReg,` `+ data = SMSA)`	Das erste Argument von `boxplot()` darf auch eine „Formel" sein. Der Formeloperator `~` (Tilde) bedeutet, dass seine linke (hier: `numeric`-)Variable auf der Ordinate abgetragen wird, und zwar „aufgeteilt" entlang der Abszisse gemäß seiner rechten (hier: `factor`-)Variablen. Quelle der Variablen: Der Data Frame für `data`. (Kein Bild gezeigt.)

`> stripchart( list( x1, x2),`
`+ vertical = TRUE,`
`+ method = "jitter",`
`+ group.names = c( "N(3,9)-`
`+ Daten", "Exp(1/3)-Daten"),`
`+ ....)`
	`stripchart()` erzeugt für jede Komponente von `list( x1, x2)` ein hier wg. `vertical = TRUE` senkrechtes, eindimensionales Streudiagramm (**Strip Chart**) der Daten, die hier infolge von `method = "jitter"` zur besseren Unterscheidbarkeit horizontal „verwackelt“ sind (im Bild in der Mitte). Das erste Argument kann auch ein Vektor oder eine Formel sein (siehe die Online-Hilfe).

Beachte: Überlagerung von Boxplot und Strip Chart mittels `add = TRUE` (und evtl. `at`) in `stripchart()` möglich (im Bild rechts). Dies ist empfehlenswert nur und gerade bei wenigen Daten.



Hinweis: Die gezeigte linke Boxplots-Grafik wurde (in etwa) durch die folgenden Befehle bzw. -ergänzungen mit den Zusatzinformationen versehen:

```
> xlist <- list( "N(3,9)-Daten" = x1, "Exp(1/3)-Daten" = x2)
> boxplot( xlist, main = "Boxplots", boxwex = 0.3, xlim = c( 0.4, 2.2))
> bp <- boxplot( xlist, plot = FALSE, range = 0)
> shift <- 0.15  # = boxwex/2
> with( bp, {
+       text( x = stats[ 3, ], labels = paste( "n =", n), pos = 3, cex = 0.7)
+       for( i in 1:ncol( stats)) {
+        text( x = rep( i - shift, nrow( stats)), y = stats[, i],
+              labels = paste( c( "Min.", "1. Quart.", "Median", "3. Quart.",
+                                 "Max."),
+                              round( stats[, i], 2), sep = ": "),
+              pos = 2, offset = 0.1, cex = 0.7)
+        } } )
```

Zu den Argumenten und dem Rückgabewert von `boxplot()` siehe ihre Online-Hilfe, auf `with()` wurde kurz schon auf Seite 56 in §2.10.7 hingewiesen, auf `text()` gehen wir kurz in Abschnitt 7.2 ein bzw. verweisen auf die Online-Hilfe und `for()` widmen wir uns in Abschnitt 6.6 bzw. empfehlen die Online-Hilfe via `?"for"` (beachte die hier notwendigen Hochkommata!).

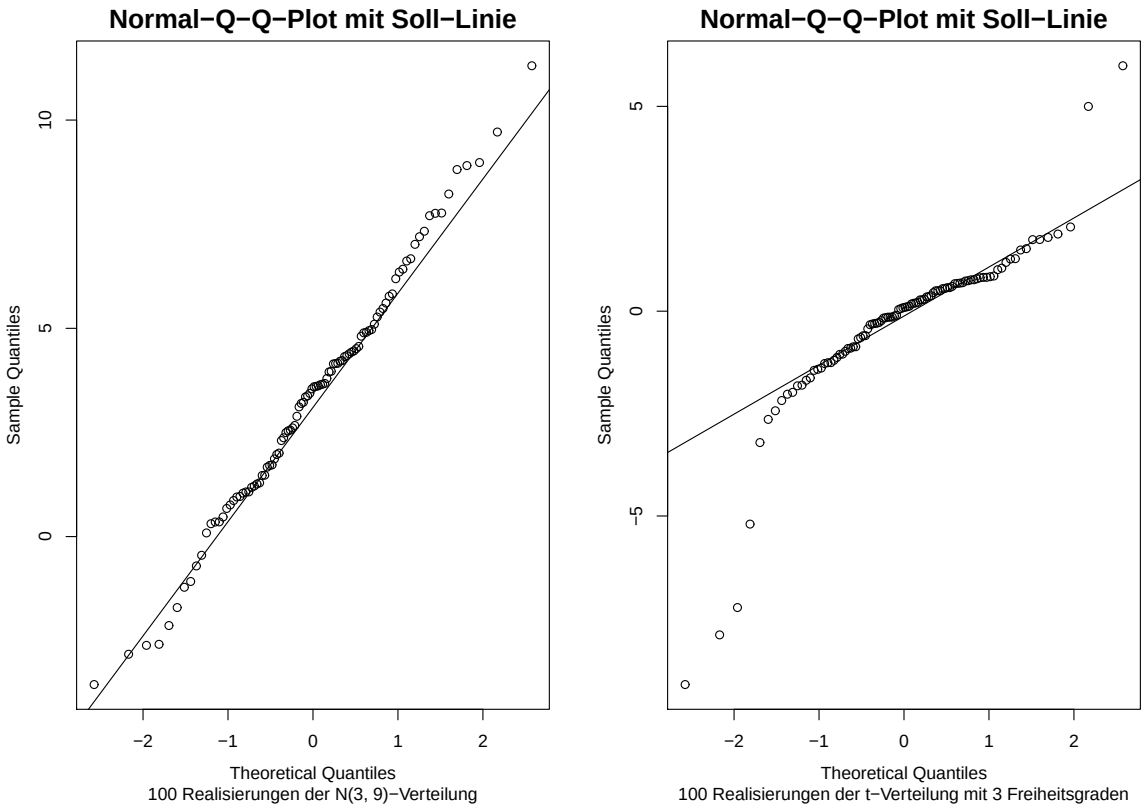
Normal-Q-Q-Plots wurden speziell dafür entwickelt, eine qualitative Beurteilung der Normalverteilungsannahme vorzunehmen:

`> x <- rnorm( 100, 3, 3)`	(`x` erhält 100 pseudo-zufällige $\mathcal{N}(3, 3^2)$-verteilte, künstliche Beispieldaten; Details folgen in Kapitel 5.)

`> qqnorm( x, main = "Normal-`
`+ Q-Q-Plot mit Soll-Linie",`
`+ sub = "100 Realisierungen`
`+ der N(3, 9)-Verteilung")`
	`qqnorm( x, ....)` liefert einen **Normal-Q-Q-Plot**, d. h. einen Plot der `x`-Ordnungsstatistiken gegen die entsprechenden Quantile der Standardnormalverteilung zur visuellen Prüfung der Normalverteiltheit der Werte in `x` (zur Theorie siehe §4.2.3). Überschrift und Untertitel: `main` und `sub`. (Siehe Grafik unten links.) Mit dem (nicht gezeigten) Argument `datax = TRUE` lässt sich erreichen, dass die Daten nicht auf der vertikalen, sondern auf der horizontalen Achse abgetragen werden.

`> qqline( x)`	`qqline()` zeichnet die „Soll-Linie“ ein, in deren Nähe die Punkte im Fall normalverteilter Daten theoretisch zu erwarten sind.

Die Grafik links unten zeigt einen Normal-Q-Q-Plot für 100 Realisierungen der $\mathcal{N}(3, 3^2)$-Verteilung und die rechte einen solchen für 100 Realisierungen der t-Verteilung mit 3 Freiheitsgraden, die im Vergleich zur Normalverteilung bekanntermaßen mehr Wahrscheinlichkeitsmasse in den Rändern hat. Dies schlägt sich im Normal-Q-Q-Plot nieder durch ein stärkeres „Ausfransen“ der Punktekette am linken Rand nach unten und am rechten Rand nach oben. (Zur Theorie siehe Punkt 2 im folgenden §4.2.3.)



Bemerkung: Im Package `car` wird eine Funktion `qqPlot()` zur Verfügung gestellt, die die Funktionalität von `qqnorm()` und `qqline()` erweitert, wie z. B. durch ein punktweises Konfidenzband um die Soll-Linie, wodurch die Beurteilung der Zulässigkeit der Normalverteilungsannahme etwas unterstützt wird.

4.2.3 Zur Theorie und Interpretation von Boxplots und Q-Q-Plots

1. Boxplots: Sie sind eine kompakte Darstellung der Verteilung eines metrischen Datensatzes mit Hervorhebung der wichtigen Kenngrößen Minimum, 1. Quartil, Median, 3. Quartil und Maximum. Sie erlauben die schnelle Beurteilung von Streuung und Symmetrie der Daten. Darüber hinaus werden potenzielle *Ausreißer* („relativ“ zur Normalverteilung) in den Daten markiert, wobei ihre Identifikation gemäß des folgenden Kriteriums geschieht: Ist

$$\begin{aligned} X_i &> \text{3. Quartil} + 1.5 \cdot \text{Quartilsabstand} = X_{\frac{3n}{4}:n} + 1.5 \cdot \left(X_{\frac{3n}{4}:n} - X_{\frac{n}{4}:n} \right) \quad \text{oder} \\ X_i &< \text{1. Quartil} - 1.5 \cdot \text{Quartilsabstand} = X_{\frac{n}{4}:n} - 1.5 \cdot \left(X_{\frac{3n}{4}:n} - X_{\frac{n}{4}:n} \right), \end{aligned}$$

so gilt X_i als potenzieller Ausreißer und wird durch ein isoliertes Symbol (hier ein Kreis) markiert. Die senkrechten, gestrichelten Linien (“whiskers” genannt) erstrecken sich nach oben bis zum größten noch nicht als Ausreißer geltenden Datum und nach unten bis zum kleinsten noch nicht als Ausreißer geltenden Datum, die beide durch einen waagrechten Strich markiert sind.

Begründung: Die obige Ausreißer-Identifikation ist durch die folgende, im Normalverteilungsfall gültige Approximation motivierbar (vgl. (1) und (3) unten):

$$X_{\frac{3n}{4}:n} + 1.5 \cdot \left(X_{\frac{3n}{4}:n} - X_{\frac{n}{4}:n} \right) \approx 4\sigma \cdot \Phi^{-1}(3/4) + \mu \approx 2.7 \cdot \sigma + \mu$$

Da $\mathbb{P}(|\mathcal{N}(\mu, \sigma^2) - \mu| > 2.7 \cdot \sigma) \approx 0.007$, sollten also im Fall $X_i \sim \mathcal{N}(\mu, \sigma^2)$ weniger als 1 % der Daten außerhalb der obigen Schranken liegen.

Zur Erinnerung hier auch die „$k\sigma$-Regel“ der Normalverteilung:

$$\begin{array}{lll} k = \frac{2}{3}: & \mathbb{P}\left(\mu - \frac{2}{3}\sigma < X \leq \mu + \frac{2}{3}\sigma\right) = 0.4950 \approx 0.5 & \\ k = 1: & \mathbb{P}\left(\mu - \sigma < X \leq \mu + \sigma\right) = 0.6826 \approx \frac{2}{3} & \text{„}\sigma\text{-Intervall“} \\ k = 2: & \mathbb{P}(\mu - 2\sigma < X \leq \mu + 2\sigma) = 0.9544 \approx 0.95 & \text{„}2\sigma\text{-Intervall“} \\ k = 3: & \mathbb{P}(\mu - 3\sigma < X \leq \mu + 3\sigma) = 0.9973 \approx 0.997 & \text{„}3\sigma\text{-Intervall“} \end{array}$$

2. Q-Q-Plots: Der Satz von Glivenko-Cantelli besagt für unabhängige Zufallsvariablen $X_1, \ldots, X_n \sim F$ mit beliebiger Verteilungsfunktion F, dass ihre empirische Verteilungsfunktion F_n mit Wahrscheinlickeit 1 (also „fast sicher“) gleichmäßig gegen F konvergiert. Kurz:

$$\sup_{q \in \mathbb{R}} |F_n(q) - F(q)| \longrightarrow 0 \text{ fast sicher für } n \to \infty$$

Daraus ist eine Aussage für die Konvergenz der empirischen Quantilfunktion F_n^{-1} herleitbar:

$$F_n^{-1}(p) \longrightarrow F^{-1}(p) \text{ fast sicher für } n \to \infty \text{ an jeder Stetigkeitsstelle } 0 < p < 1 \text{ von } F^{-1}$$

D. h., für hinreichend großes n ist $F_n^{-1}(p) \approx F^{-1}(p)$ an einem jeden solchen p. Und alldieweil wir für jede Ordnungsstatistik $X_{i:n}$ die Identität $F_n^{-1}(i/n) = X_{i:n}$ haben, muss gelten:

$$X_{i:n} \approx F^{-1}(i/n) \quad \text{für } i = 1, \ldots, n \text{ mit hinreichend großem } n \tag{1}$$

Die Approximation (1) verbessert sich, wenn man die Quantilfunktion etwas „shiftet“:

$$\begin{aligned} X_{i:n} &\approx F^{-1}((i - 1/2)/n) \quad \text{für } n > 10 \quad \text{bzw.} \\ X_{i:n} &\approx F^{-1}((i - 3/8)/(n + 1/4)) \quad \text{für } n \leq 10 \end{aligned} \tag{2}$$

Aufgrund dieser Approximationen sollten sich in einem Q-Q-Plot genannten Streudiagramm der – auch empirische Quantile heißenden – Ordnungsstatistiken $X_{i:n}$ gegen die theoretischen Quantile $F^{-1}((i - 1/2)/n)$ bzw. $F^{-1}((i - 3/8)/(n + 1/4))$ die Punkte in etwa entlang der Identitätslinie $y = x$ aufreihen. Damit haben wir ein Vehikel, um die Verteilungsannahme für die

X_i zu beurteilen: Sollte die „Punktekette“ des Q-Q-Plots *nicht* in etwa an der Identitätslinie entlang verlaufen, so können die Daten nicht aus der Verteilung zu F stammen.

Im **Fall der Normalverteilung** ist $F = \Phi_{\mu,\sigma^2} \equiv \Phi\left(\frac{\cdot-\mu}{\sigma}\right)$, wobei Φ die Verteilungsfunktion der Standardnormalverteilung ist, μ der Erwartungswert und σ^2 die Varianz. Aus $\Phi_{\mu,\sigma^2}(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$ folgt

$$\Phi^{-1}_{\mu,\sigma^2}(u) = \sigma\ \Phi^{-1}(u) + \mu, \tag{3}$$

sodass unter X_i i.i.d. $\sim \mathcal{N}(\mu, \sigma^2)$ für die Ordnungsstatistiken $X_{1:n}, \ldots, X_{n:n}$ gelten muss:

$$\begin{aligned} X_{i:n} &\approx \sigma\ \Phi^{-1}\left((i-0.5)/n\right) + \mu \quad \text{für } n > 10 \qquad \text{bzw.} \\ X_{i:n} &\approx \sigma\ \Phi^{-1}\left((i-3/8)/(n+1/4)\right) + \mu \quad \text{für } n \leq 10 \end{aligned} \tag{4}$$

Offenbar stehen diese empirischen Quantile $X_{i:n}$ aus einer beliebigen Normalverteilung in einer approximativ *linearen* Beziehung zu den theoretischen Quantilen $\Phi^{-1}((i-0.5)/n)$ bzw. $\Phi^{-1}((i-3/8)/(n+1/4))$ der Standardnormalverteilung, wobei Steigung und „y-Achsenabschnitt“ dieser Beziehung gerade σ bzw. μ sind. In einem Streudiagramm dieser Quantile, das Normal-Q-Q-Plot genannt und durch die Funktion `qqnorm()` geliefert wird, sollte die resultierende Punktekette demnach einen approximativ linearen Verlauf zeigen. (Dabei ist irrelevant, wo und wie steil diese Punktekette verläuft, denn die Zulässigkeit statistischer Verfahren hängt oft nur davon ab, dass die Daten überhaupt aus einer Normalverteilung stammen.)

Fazit: Zeigt sich im Normal-Q-Q-Plot *kein* approximativ linearer Verlauf, so ist die Normalverteilungsannahme für die X_i *nicht* zulässig.

Ergänzung: Der Erwartungswert μ und die Standardabweichung σ sind in der Praxis – auch unter der Normalverteilungsannahme – in der Regel unbekannt, lassen sich aber durch das arithmetische Mittel $\hat{\mu}$ und die Stichprobenstandardabweichung $\hat{\sigma}$ konsistent schätzen. Die Soll-Linie $y(x) = \sigma\ x + \mu$ für den Normal-Q-Q-Plot der $X_{i:n}$ gegen $\Phi^{-1}((i-0.5)/n)$ oder $\Phi^{-1}((i-3/8)/(n+1/4))$ *könnte* also durch $y(x) = \hat{\sigma}\ x + \hat{\mu}$ approximiert und zur Beurteilung des linearen Verlaufs der Punktekette als Referenz eingezeichnet werden, was jedoch *nicht* geschieht. Stattdessen wird aus Gründen der Robustheit (durch die Funktion `qqline()`) diejenige Gerade eingezeichnet, die durch die ersten und dritten empirischen und theoretischen Quartile verläuft, also durch die Punkte $(\Phi^{-1}(1/4), X_{\frac{n}{4}:n})$ und $(\Phi^{-1}(3/4), X_{\frac{3n}{4}:n})$. Sie hat die Gleichung

$$y(x) = \frac{X_{\frac{3n}{4}:n} - X_{\frac{n}{4}:n}}{\Phi^{-1}(3/4) - \Phi^{-1}(1/4)}\ x + \frac{X_{\frac{3n}{4}:n} + X_{\frac{n}{4}:n}}{2}$$

Was hat diese robuste Gerade mit der eigentlichen, linearen Beziehung zu tun? Antwort(en):

1. Für symmetrische Verteilungen ist das arithmetische Mittel von erstem und zweitem empirischen Quartil ein guter Schätzer des Medians, der im Normalverteilungsfall gleich dem Erwartungswert μ ist. Also schätzt der y-Achsenabschnitt dieser robusten Geraden das μ.

2. Der empirische Quartilsabstand $X_{\frac{3n}{4}:n} - X_{\frac{n}{4}:n}$ ist ein Schätzer für $F^{-1}(3/4) - F^{-1}(1/4)$, wofür im Normalverteilungsfall gemäß (3) gilt: $F^{-1}(3/4) - F^{-1}(1/4) = \sigma\ (\Phi^{-1}(3/4) - \Phi^{-1}(1/4))$. Damit schätzt hier die Steigung dieser robusten Geraden das σ.

Bemerkungen:

- Als die „Bibel“ der explorativen Datenanalyse gilt [77, Tukey (1977)]. Eine Einführung in die grafische Datenanalyse geben auch [18, Chambers et al. (1983)]. In beiden Referenzen werden auch die obigen Verfahren beschrieben.
- Zum praktischen Sinn oder Unsinn eines statistischen Tests der Hypothese, dass Daten aus einer exakten Normalverteilung kommen, ist die Funktion `SnowsPenultimateNormalityTest()` des **R**-Paketes `TeachingDemos` und ihre Online-Hilfe eine humorvoll gehaltene Ermahnung. Auch lesenswert zu diesem Thema ist
  `http://stackoverflow.com/questions/7781798/seeing-if-data-is-normally-distributed-in-r`

4.3 Explorative Grafiken für multivariate Daten

Für **multivariate Datensätze** ist die Darstellung aufwändiger und schwieriger (bis unmöglich). Die Häufigkeitstabelle eines zweidimensionalen, endlich-diskreten oder nominal- bzw. ordinalskalierten Datensatzes lässt sich noch durch ein multiples Balkendiagramm oder durch Mosaikplots veranschaulichen und die Verteilung von zwei- bzw. dreidimensionalen metrischskalierten Daten durch ein zwei- bzw. dreidimensionales Streudiagramm. Ab vier Dimensionen jedoch erlauben im metrischen Fall nur noch paarweise Streudiagramme aller bivariaten Kombinationen der Dimensionen der multivariaten Beobachtungsvektoren eine grafisch einigermaßen anschauliche Betrachtung; Häufigkeitstabellen multivariat nominal- bzw. ordinalskalierter Daten werden schnell völlig unübersichtlich.

4.3.1 Die Häufigkeitsverteilung bivariat diskreter Daten: Mosaikplots

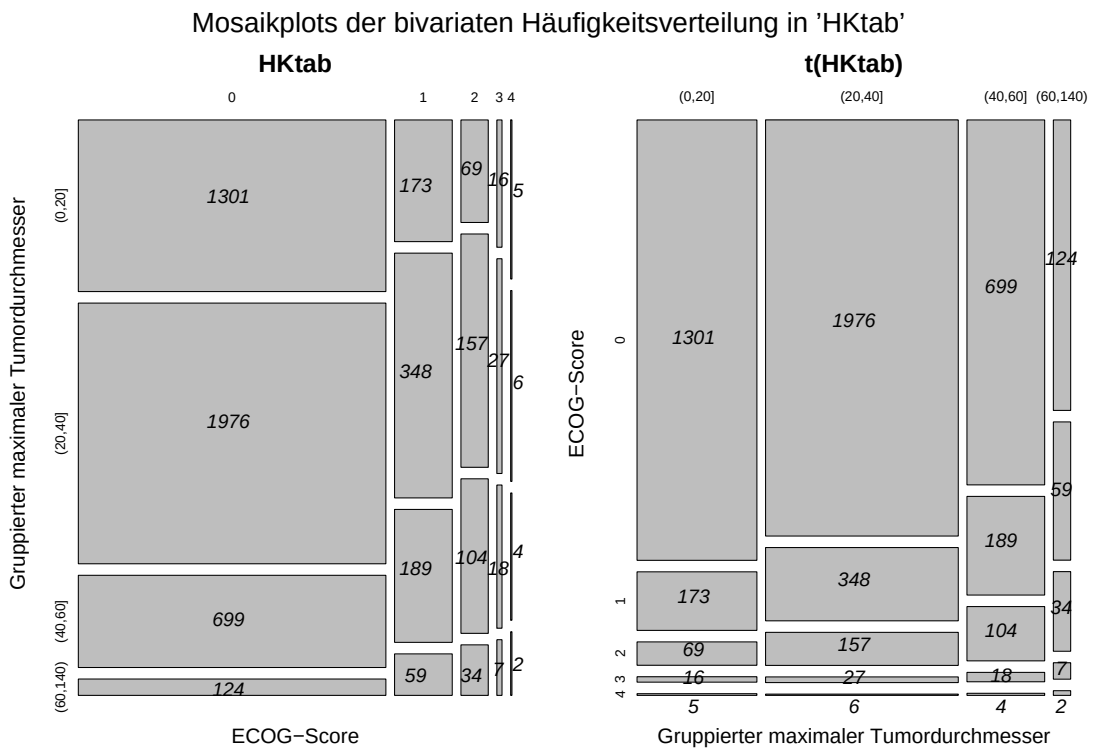
Für bivariate endlich-diskrete oder nominal- bzw. ordinalskalierte Daten sind Mosaikplots eine mögliche Darstellung der Häufigkeitstabelle der Wertepaare (auch Kontingenztafel genannt). Als Beispiel verwenden wir wieder die bereits als absolute Häufigkeiten in der `numeric`-Matrix `HKmat` vorliegenden Mundhöhlenkarzinomdaten (vgl. S. 71):

`> HKtab <- as.table( HKmat)`
`> rownames( HKtab) <- 0:4`
`> dimnames( HKtab)`
`[[1]]`
`[1] "0" "1" "2" "3" "4"`
`[[2]]`
`[1] "(0,20]"  "(20,40]"`
`[3] "(40,60]"  "(60,140)"`
	Zunächst wandelt `as.table()` das `matrix`-Objekt `HKmat` in ein `table`-Objekt um. (Sinnvollerweise erzeugt man solche Häufigkeitstabellen von vorneherein mit `table()`, wie wir es später in Abschnitt 9.6 öfter tun werden. Außerdem verkürzen wir die Zeilennamen, damit die grafische Ausgabe übersichtlicher wird.)

`> mosaicplot( HKtab,`
`+ xlab = "ECOG-Score",`
`+ ylab = "Gruppierter`
`+ maximaler`
`+ Tumordurchmesser")`
	`mosaicplot()` erstellt für ein `table`-Objekt als erstem Argument (hier `HKtab`) den **Mosaikplot** auf der nächsten Seite oben links. Darin ist die Fläche von „Fliese“ (i, j) proportional zu H_{ij}/n, der relativen Häufigkeit in Tabellenzelle (i, j). Um dies zu erreichen, ist die Fliesenbreite proportional zu $H_{\cdot j}/n$, der relativen Spaltenhäufigkeit, und die Fliesenhöhe proportional zu $H_{ij}/H_{\cdot j}$, der Zellenhäufigkeit relativ zur Spaltenhäufigkeit.

`> mosaicplot( t( HKtab)),`
`+ xlab = "Gruppierter`
`+ maximaler`
`+ Tumordurchmesser",`
`+ ylab = "ECOG-Score")`
	Daher ist die Darstellung nicht „transponierinvariant“, wie der Vergleich des Ergebnisses für `t( HKtab)` auf der nächsten Seite oben rechts mit der Grafik links daneben schnell klar macht.
	Die „Zeilen“- und „Spalten“-Beschriftung wird dem `dimnames`-Attribut des `table`-Objekts entnommen, die „Achsen“-Beschriftung steuern `xlab` und `ylab` und die Überschrift ist mit `main` möglich. (Die gezeigten Grafiken haben eine eigene, nicht durch `mosaicplot()` generierte Zusatzbeschriftung durch Zellenhäufigkeiten.)

Bemerkung: Durch Mosaikplots lassen sich auch höher- als zweidimensionale Häufigkeitsverteilungen endlich-diskreter oder nominal- bzw. ordinalskalierter Datensätze darstellen. (Siehe z. B. [35, Friendly, M. (1994)]: *Mosaic displays for multi-way contingency tables.* Journal of the American Statistical Association, 89, pp. 190 - 200.)



4.3.2 Die Verteilung multivariat metrischer Daten: Streudiagramme

Als ein Beispiel für multivariate metrisch skalierte Daten dienen uns sechsdimensionale Messungen $(X_1, \ldots, X_6)$ an 40 Exemplaren des „Brillenschötchens“ (biscutella laevigata) (aus [75, Timischl (1990)], Seite 4), die in den jeweils 40-elementigen Vektoren `X1`, ..., `X6` gespeichert seien. Diese Variablen enthalten die folgenden (namensgleichen) Merkmale, gefolgt von einem Ausschnitt aus der Tabelle der Rohdaten:

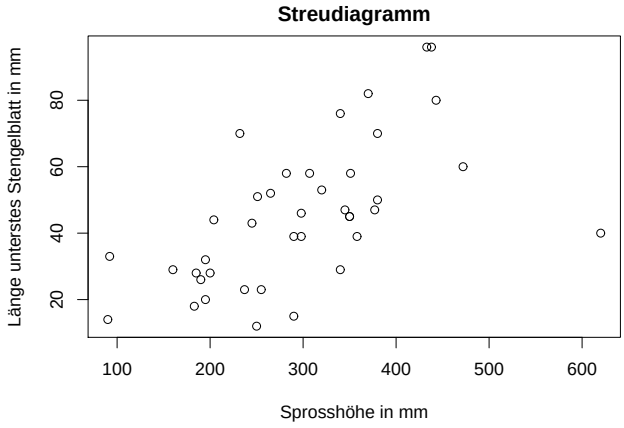
X_1 : Sprosshöhe in mm
X_2 : Länge des größten Grundblattes in mm
X_3 : Anzahl der Zähne des größten Grundblattes an einem Blattrand
X_4 : Anzahl der Stengelblätter am Hauptspross
X_5 : Länge des untersten Stengelblattes in mm
X_6 : Spaltöffnungslänge in μm
X_7 : Chromosomensatz (d = diploid, t = tetraploid)
X_8 : Entwicklungsstand (1 = blühend, 2 = blühend und fruchtend,
3 = fruchtend mit grünen Schötchen,
4 = fruchtend mit gelben Schötchen)

i	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
1	298	50	1	6	39	27	d	4
2	345	65	2	7	47	25	d	1
3	183	32	0	5	18	23	d	3
⋮				...				
20	265	63	4	6	52	23	d	4

i	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
21	232	75	3	6	70	26	d	2
22	358	64	2	11	39	28	t	4
23	290	48	0	12	39	30	t	1
⋮				...				
40	620	48	4	10	40	26	d	2

Ein (zweidimensionales) Streudiagramm (Engl.: “scatter plot”) für die Realisierungen der zwei Variablen X_1 und X_5 (genauer: der Elemente in `X5` gegen die Elemente in `X1`) kann wie folgt mit Hilfe der Funktion `plot()` erzeugt werden:

```
> plot( X1, X5, xlab = "Sprosshöhe in mm",
+ ylab = "Länge unterstes Stengelblatt in mm", main = "Streudiagramm")
```

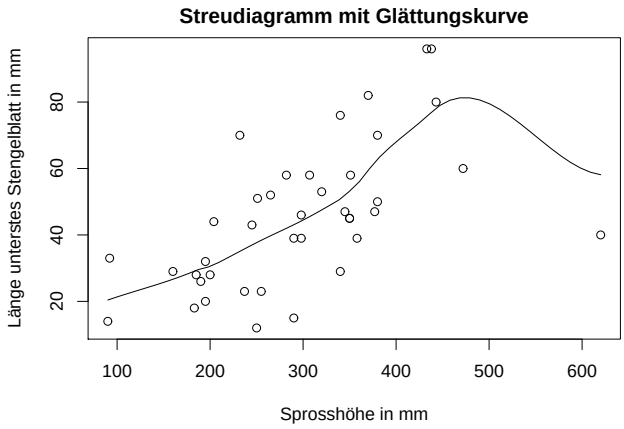


Das erste Argument von `plot()` ist der Vektor der waagrechten Koordinaten, das zweite der Vektor der senkrechten Koordinaten der zu zeichnenden Punkte. `xlab` und `ylab` liefern die Achsenbeschriftungen; `main` die Überschrift. (Mit dem hier nicht verwendeten Argument `sub` wäre noch ein Untertitel möglich.)

Hinweis: Sollte es – anders als hier – in einem Streudiagramm z. B. aufgrund gerundeter metrischer Koordinaten zu starken Überlagerungen der Punkte kommen, kann ein "sunflower plot" hilfreich sein. Siehe `example( sunflowerplot)` und die Online-Hilfe zu `sunflowerplot()`. Für *sehr* umfangreiche und dicht liegende zweidimensionale Datensätze lohnt sich ein Blick auf die Darstellungen, die `example( smoothScatter)` präsentiert, oder ein Blick auf `example( hexbinplot)` des **R**-Paketes `hexbin` (nachdem man es installiert hat).

Zusätzlich kann in ein Streudiagramm – sagen wir zur Unterstützung des optischen Eindrucks eines möglichen Zusammenhangs der dargestellten Variablen – eine sogenannte (nicht-parametrische) Glättungskurve (Engl.: "smooth curve") eingezeichnet werden. Dies ermöglicht die Funktion `scatter.smooth()`. Sie bestimmt mit der sogenannten „loess"-Methode eine lokal-lineare oder lokal-quadratische Regressionskurve und zeichnet den dazugehörigen Plot. („loess" könnte von dem deutschen Begriff „Löss" (= kalkhaltiges Sediment des Pleistozäns, das sich oft sanft wellig zeigt) kommen, oder eine Abkürzung für "locally estimated scatter smoother" sein; auf die technischen Details gehen wir hier nicht ein. Siehe z. B. [21, Cleveland, W. S., Grosse, E., Shyu, W. M. (1992): *Local regression models.* Chapter 8 of *Statistical Models in S*, eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole]. Beispiel:

```
> scatter.smooth( X1, X5, span = 2/3, degree = 1, xlab = "Sprosshöhe in mm",
+ ylab = "Länge unterstes Stengelblatt in mm",
+ main = "Streudiagramm mit Glättungskurve")
```



Die ersten zwei Argumente enthalten die x- und y-Koordinaten der Datenpunkte. Die Argumente `span` und `degree` steuern Grad und Art der Glättung: `span` ist der Anteil der Daten, der in die lokale Glättung eingehen soll. Je größer `span`, umso „glatter" die Kurve. `degree = 1` legt eine lokal-lineare und `degree = 2` eine lokal-quadratische Glättung fest. `xlab`, `ylab` und `main` (und `sub`) funktionieren wie bei `plot()`.

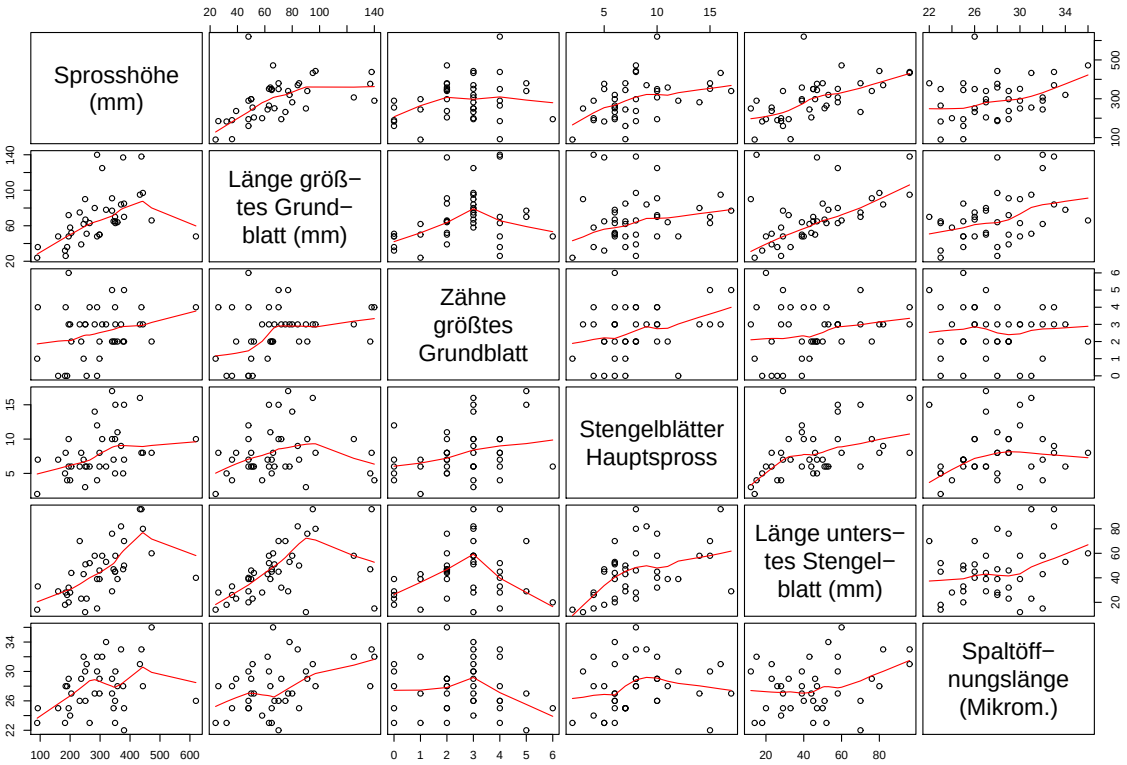
Ein Plot der Streudiagramme *aller* bivariaten Kombinationen der Komponenten multivariater Beobachtungsvektoren (hier 6-dimensional) befindet sich auf der nächsten Seite. Es ist ein sogenannter Pairs-Plot für die Matrix `M <- cbind( X1, ...., X6)`, deren Zeilen als die multivariaten Beobachtungsvektoren aufgefasst werden. Er wird durch die Funktion `pairs()` erzeugt,

die außerdem durch ihr Argument `labels` eine explizite Beschriftung der Variablen im Plot ermöglicht (dabei ist „`\n`“ das Steuerzeichen für den Zeilenumbruch).

Auch in einen Pairs-Plot kann in jedes der Streudiagramme eine (nicht-parametrische) lokale Glättungskurve eingezeichnet werden. Dazu muss dem Argument `panel` der Funktion `pairs()` die Funktion `panel.smooth()` übergeben werden. Im folgenden Beispiel geschieht dies gleich in Form einer “in-line”-definierten Funktion mit einem speziell voreingestellten Argument für den Glättungsgrad (`span = 2/3`), nur um anzudeuten, wie dieser (und andere) Parameter hier variiert werden könnte(n). (Der gewählte Wert 2/3 ist sowieso auch der **R**-Voreinstellungswert.)

Die in `panel.smooth()` zum Einsatz kommende lokale Glättungsmethode ist allerdings *nicht* die loess-Methode, sondern die sogenannte „lowess“-Methode (= “locally weighted scatter smoother”), die eine robuste, lokal gewichtete Glättung durchführt. (Siehe z. B. [19, Cleveland, W. S. (1981): *LOWESS: A program for smoothing scatterplots by robust locally weighted regression.* The American Statistician, 35, p. 54].) Die folgenden Befehle führten zu der unten gezeigten Grafik:

```
> var.labels <- c( "Sprosshöhe\n(mm)", "Länge größ-\ntes Grund-\nblatt (mm)",
+ "Zähne\ngrößtes\nGrundblatt","Stengelblätter\nHauptspross",
+ "Länge unters-\ntes Stengel-\nblatt (mm)",
+ "Spaltöff-\nnungslänge\n(Mikrom.)")
> pairs( M, labels = var.labels,
+ panel = function( x, y) { panel.smooth( x, y, span = 2/3) } )}
```



Bemerkungen:

- An der „Kreuzung“ von erster Spalte und fünfter Zeile des obigen Arrangements findet sich das Streudiagramm von Seite 82 unten wieder. Offensichtlich sind die jeweils darin eingezeichneten Glättungskurven verschieden.
- Für weitere, sehr leistungsfähige Argumente der Funktion `pairs()` verweisen wir (wieder einmal) auf die Online-Hilfe.

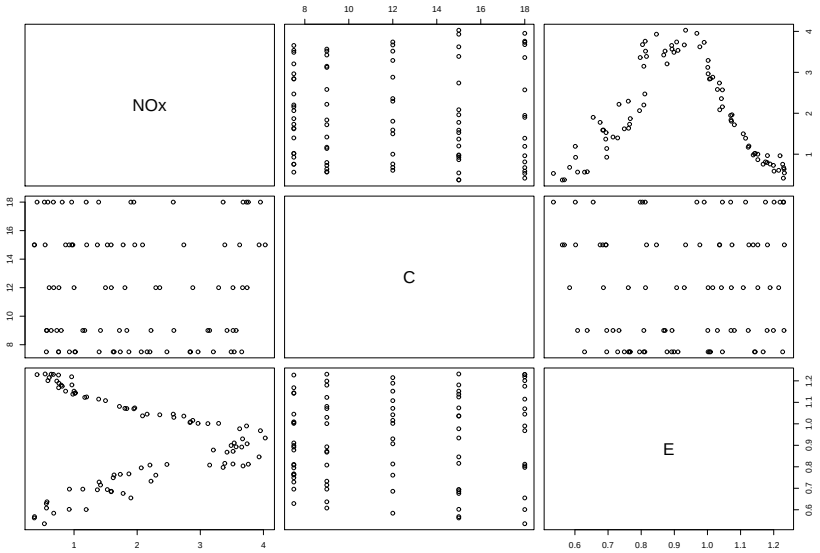
4.3.3 Die Verteilung trivariat metrischer Daten: Bedingte Streudiagramme ("co-plots")

Eine weitere Möglichkeit, eventuelle Zusammenhänge in dreidimensionalen Daten grafisch zweidimensional zu veranschaulichen, bieten die sogenannten "conditioning plots" (kurz: "co-plots") oder bedingten Plots. Hier werden paarweise Streudiagramme zweier Variablen (zusammen mit nicht-parametrischen Glättungskurven) unter der Bedingung geplottet, dass eine dritte Variable in ausgewählten Wertebereichen liegt. Realisiert wird dies durch die Funktion `coplot()` und anhand eines Beispiels soll das Verfahren erläutert werden:

Im – i. d. R. mit "base **R**" mit-installierten – **R**-Paket `lattice` ist ein Datensatz namens `ethanol` eingebaut, der Daten aus einem Experiment mit einem Ein-Zylinder-Testmotor zur Abhängigkeit der NO_x-Emmission (`NOx`) von verschiedenen Kompressionswerten (`C`) und Gas-Luft-Gemischverhältnissen (`E`) enthält. Er wird durch `data()` zur Verfügung gestellt und der folgende Pairs-Plot liefert einen ersten Eindruck der Daten:

```
> data( ethanol, package = "lattice")
> pairs( ethanol)
```

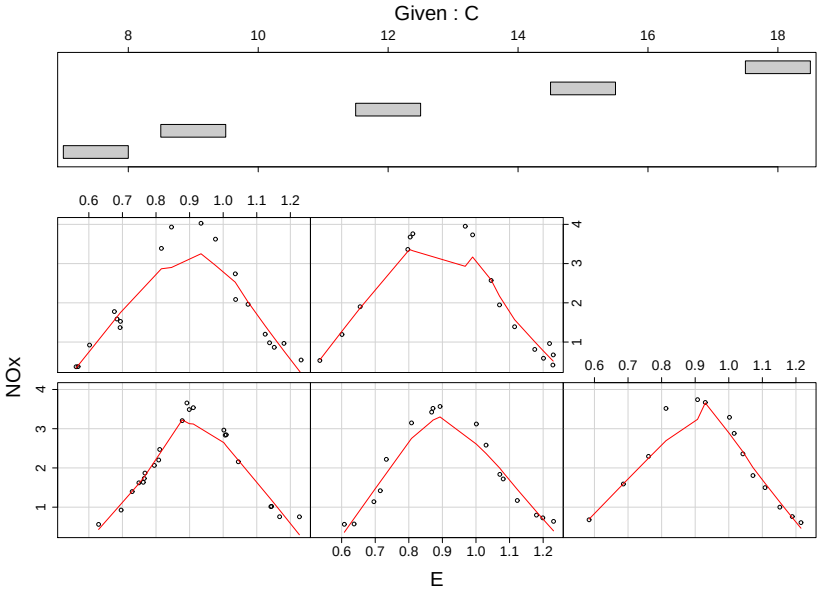
ergibt nebenstehenden Plot. Eine deutliche (evtl. quadratische) Abhängigkeit des `NOx` von `E` wird sofort offenkundig, was sich für `NOx` und `C` nicht sagen lässt. Auch ein Einfluss von `C` *auf die Abhängigkeit* des `NOx` von `E`, also eine *Wechselwirkung* zwischen `C` und `E`, ist so nicht zu entdecken.
Eine Möglichkeit hierzu bieten die unten beschriebenen co-plots, da sie eine detailliertere Betrachtung erlauben.



Zunächst betrachten wir den Fall, dass die **Variable** `C` (die nur fünf verschiedene Werte angenommen hat und somit als diskret aufgefasst werden kann) **als bedingende Größe** verwendet wird: Erst werden die `C`-Werte bestimmt, nach denen die Abhängigkeit von `NOx` und `E` bedingt werden soll (siehe `C.points` unten). Dann wird in `coplot()` „`NOx` an `E` modelliert, gegeben `C`" (mittels der Modellformel `NOx ~ E | C`), wozu die gegebenen `C`-Werte dem Argument `given.values` zugewiesen werden. Das Argument `data` erhält den Data Frame mit den in der Modellformel verwendeten Variablen und das Argument `panel` die Funktion `panel.smooth()` für die lowess-Glättungskurve (vgl. hierzu Seite 83 im vorherigen Paragraphen):

```
> C.points <- sort( unique( ethanol$C))
> coplot( NOx ~ E | C, given.values = C.points, data = ethanol,
+ panel = panel.smooth)
```

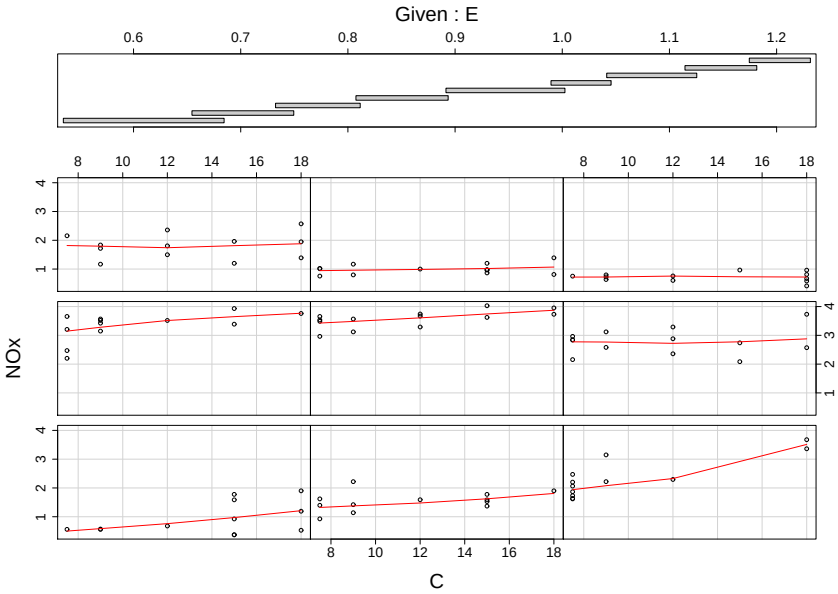
Das Resultat (auf der nächsten Seite oben) ist wie folgt zu „lesen": Die Werte(bereiche) der Bedingungsvariablen `C` sind in dem oberen „Panel" (mit der Überschrift `Given : C`) als graue Balken dargestellt, denen die darunter gezeigten `NOx`-`E`-Streudiagramme so zugeordnet sind, dass diese (beginnend links unten) von links nach rechts und von unten nach oben zu den Balken im `Given`-Panel von link nach rechts gehören.



Beobachtung: Über den gesamten Wertebereich von `C` hinweg scheint die Abhängigkeit von `NOx` von `E` vom selben (quadratischen?) Typ zu sein und dies auch gleichermaßen stark.

Nun wollen wir **nach der stetigen Variablen `E` bedingen**. Dazu wird mit der Funktion `co.intervals()` der `E`-Wertebereich in `number` Intervalle aufgeteilt, die sich zu `overlap` überlappen (siehe `E.ints` unten) und nach denen die Abhängigkeit von `NOx` und `C` bedingt werden soll. Dann wird „`NOx` an `C` modelliert, gegeben `E`“, wozu die gegebenen `E`-Intervalle dem Argument `given.values` zugewiesen werden. Die Argumente `data` und `panel` fungieren wie eben, wobei hier an `panel` eine in-line-definierte Funktion mit spezieller Einstellung für den Glättungsparameter `span` in `panel.smooth()` übergeben wird. Da hier `span = 1` ist, wird stärker geglättet als bei der Voreinstellung 2/3.

```
> E.ints <- co.intervals( ethanol$E, number = 9, overlap = 0.25)
> coplot( NOx ~ C | E, given.values = E.ints, data = ethanol,
+ panel = function( x, y, ...) { panel.smooth( x, y, span = 1, ...) } )
```



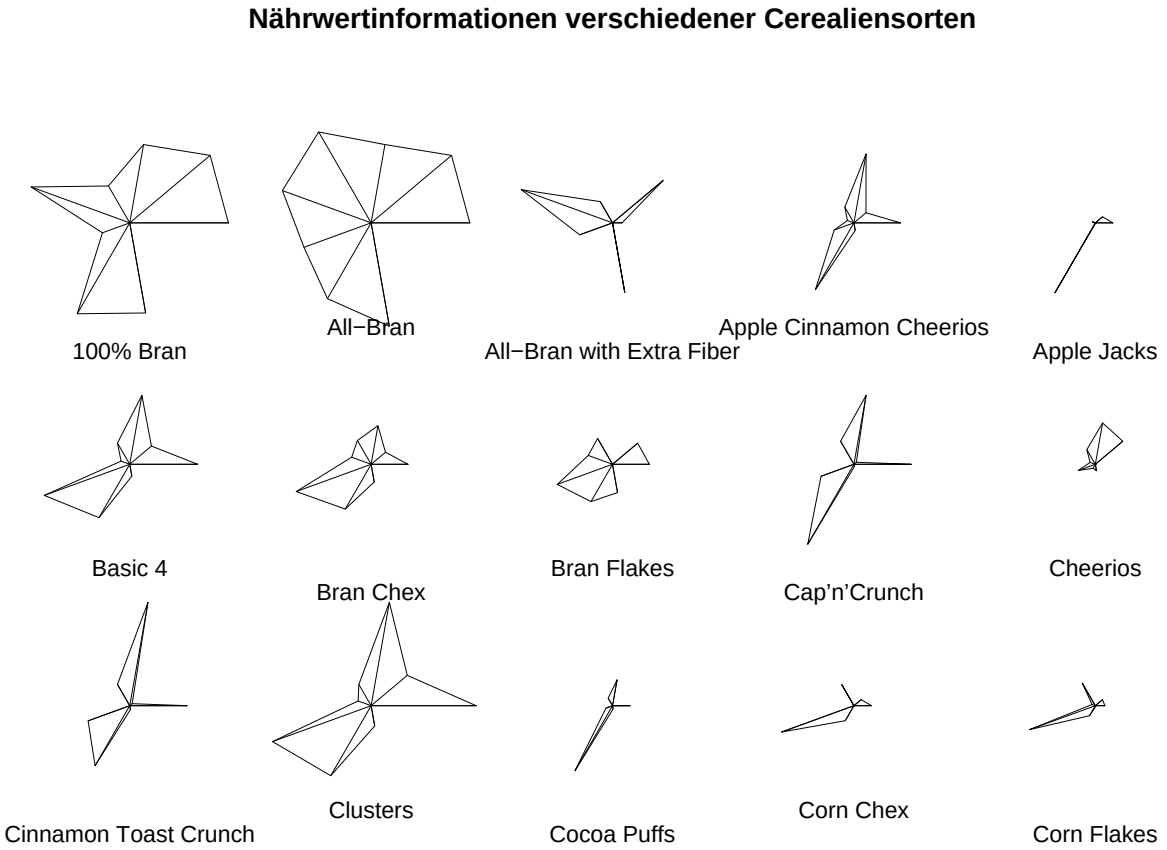
Beobachtung: Für niedrige Werte von `E` nimmt `NOx` mit steigendem `C` zu, während es für mittlere und hohe `E`-Werte als Funktion von `C` (!) konstant zu sein scheint.

Hinweis: Weitere und zum Teil etwas komplexere Beispiele zeigt Ihnen `example( coplot)`.

4.3.4 Weitere Möglichkeiten und Hilfsmittel für multivariate Darstellungen: `stars()`, `symbols()`

Eine weitere Möglichkeit, jede einzelne multivariate Beobachtung zu veranschaulichen, und zwar als „Sternplot“, bietet `stars()` (manchmal auch „Spinnen-“ oder „Radarplot“ genannt). Anhand eines Ausschnitts aus dem Datensatz `UScereal` im **R**-Paket `MASS` wird die Methode kurz vorgestellt, aber näher wollen wir hier nicht darauf eingehen. Zu Details siehe `?stars` und außerdem `help( UScereal, package = "MASS")`.

```
> data( UScereal, package = "MASS")
> stars( UScereal[ -c( 1, 9)][ 1:15,], nrow = 3, ncol = 5, cex = 1.1,
+ main = "Nährwertinformationen verschiedener Cerealiensorten")
```



Andere oder in der Ausgestaltung ausgefeiltere oder komplexere Varianten liefern Ihnen die – empfehlenswerten – Beispiele in der Online-Hilfe; siehe also `example( stars)`.

Über die Fähigkeiten und die Anwendung der Funktion `symbols()` sollten Sie sich durch ihre via `example( symbols)` erhältliche Beispielekollektion sowie die zugehörige Online-Hilfe informieren.

Amüsant und interessant, auch weil 100 % ernst gemeint, ist das Darstellungskonzept der “Chernoff faces” für multivariate Daten, wozu Sie z. B. unter `http://de.wikipedia.org/wiki/Chernoff_faces` mehr finden.

Bemerkungen: Ein sehr empfehlenswertes sowie weit- und tiefgehendes Buch zum Thema Grafik in **R** ist “R Graphics” ([64, Murrell (2005)] bzw. seine zweite Auflage [65, Murrell (2011)]). Dasselbe gilt für das exzellente Buch “Lattice. Multivariate Data Visualization with R” ([69, Sarkar (2008)]), das die **R**-Implementation und -Erweiterung des hervorragenden “Trellis-Grafik”-Systems vorstellt, auf das wir hier nicht eingehen.

5 Wahrscheinlichkeitsverteilungen und Pseudo-Zufallszahlen

R hat bereits in der "base distribution" für viele Wahrscheinlichkeitsverteilungen sowohl die jeweilige Dichtefunktion (im Fall einer stetigen Verteilung) bzw. Wahrscheinlichkeitsfunktion (im diskreten Fall) als auch Verteilungs- und Quantilfunktion implementiert; fallweise natürlich nur approximativ. Ebenso sind Generatoren für Pseudo-Zufallszahlen implementiert, mit denen Stichproben aus diesen Verteilungen simuliert werden können. Wenn wir im Folgenden (zur Abkürzung) von der Erzeugung von Zufallszahlen sprechen, meinen wir stets *Pseudo*-Zufallszahlen. (Beachte: In der **R**-Terminologie wird nicht zwischen Dichtefunktion einer stetigen Verteilung und Wahrscheinlichkeitsfunktion einer diskreten Verteilung unterschieden, sondern beides als "density function" bezeichnet.)

5.1 Die „eingebauten" Verteilungen

Die **R**-Namen der obigen vier Funktions*typen* setzen sich wie folgt zusammen: Ihr erster Buchstabe gibt den Funktionstyp an, der Rest des Namens identifiziert die Verteilung, für die dieser Typ zu realisieren ist. Der Aufruf eines Funktionstyps einer Verteilung benötigt (neben eventuell anzugebenden spezifischen Verteilungsparametern) die Angabe der Stelle, an der eine Funktion ausgewertet werden soll, bzw. die Anzahl der zu generierenden Zufallszahlen. Etwas formaler gilt allgemein für eine beliebige, implementierte Verteilung *dist* (mit möglichen Parametern „. . . .") mit Verteilungsfunktion (VF) $F_{....}$ und Dichte- bzw. Wahrscheinlichkeitsfunktion $f_{....}$:

`d`*`dist`*`( x, ....)`	$= f_{....}(x)$	: Dichte-/Wahrscheinlichkeitsfkt.	der Verteilung *dist*
`p`*`dist`*`( x, ....)`	$= F_{....}(x)$	: VF (also $\mathbb{P}(X \leq x)$ für $X \sim F$)	
`q`*`dist`*`( y, ....)`	$= F^{-1}_{....}(y)$	: Quantilfunktion	
`r`*`dist`*`( n, ....)`	liefert n Zufallszahlen aus		

Beispiele: Für den stetigen Fall betrachten wir die (Standard-)Normalverteilung:

`dnorm( x)`	$= \phi(x)$	: Dichte	der Standardnormalverteilung
`pnorm( x)`	$= \Phi(x)$	: VF	
`qnorm( y)`	$= \Phi^{-1}(y)$	: Quantilfunktion	
`rnorm( n)`	liefert n Zufallszahlen aus		

Gemäß der Voreinstellung liefern Aufrufe der Funktionstypen aus der Familie der Normalverteilung immer Resultate für die *Standard*normalverteilung, wenn die Parameter `mean` und `sd` nicht explizit mit Werten versehen werden:

Die Normalverteilung	
`> dnorm( c( 7, 8), mean = 10)` `[1] 0.004431848 0.053990967`	Werte der Dichte der $\mathcal{N}(10, 1)$-Verteilung an den Stellen 7 und 8.
`> pnorm( c( 1.5, 1.96))` `[1] 0.9331928 0.9750021`	Werte der Standardnormalverteilungsfunktion Φ an den Stellen 1.5 und 1.96.
`> qnorm( c( 0.05, 0.975))` `[1] -1.644854  1.959964`	Das 0.05- und das 0.975-Quantil der Standardnormalverteilung, also $\Phi^{-1}(0.05)$ und $\Phi^{-1}(0.975)$.

`> rnorm( 6, mean = 5, sd = 2)`
`[1] 5.512648 8.121941 5.672748`
`[4] 7.665314 4.081352 4.632989`
	Sechs Zufallszahlen aus der $\mathcal{N}(5, 2^2)$-Verteilung.

Für den diskreten Fall betrachten wir die Binomialverteilung:

$$\left.\begin{array}{lll}
\texttt{dbinom(k, size =}\ m\texttt{, prob =}\ p\texttt{)} & = \mathbb{P}(X = k) & : \text{W.-Funktion} \\
\texttt{pbinom(k, size =}\ m\texttt{, prob =}\ p\texttt{)} & = F(k) := \mathbb{P}(X \leq k) & : \text{VF} \\
\texttt{qbinom(y, size =}\ m\texttt{, prob =}\ p\texttt{)} & = F^{-1}(y) & : \text{Quantilfunktion} \\
\texttt{rbinom(n, size =}\ m\texttt{, prob =}\ p\texttt{)} & \text{liefert } n \text{ Zufallszahlen aus} &
\end{array}\right\} \ldots$$

... der Binomial(m, p)-Verteilung (d. h. für $X \sim \text{Bin}(m, p)$).

Hier zwei Tabellen vieler der in **R** zur Verfügung stehenden Verteilungen, ihrer jeweiligen Funktionsnamen und Parameter (samt Voreinstellungen, sofern gegeben; beachte auch die Ergänzung auf Seite 89 oben):

Diskrete Verteilungen		
... (-)Verteilung	**R**-Name	Verteilungsparameter
Binomial	`binom`	`size, prob`
Geometrische	`geom`	`prob`
Hypergeometrische	`hyper`	`m, n, k`
Multinomial	`multinom`	`size, prob` (nur `r....` und `d....`)
Negative Binomial	`nbinom`	`size, prob`
Poisson	`pois`	`lambda`
Wilcoxons Vorzeichen-Rangsummen	`signrank`	`n`
Wilcoxons Rangsummen	`wilcox`	`m, n`

Stetige Verteilungen		
... (-)Verteilung	**R**-Name	Verteilungsparameter
Beta	`beta`	`shape1, shape2, ncp = 0`
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, rate = 1`
Log-Normal	`lnorm`	`meanlog = 0, sdlog = 1`
Logistische	`logis`	`location = 0, scale = 1`
Multivariate Normal (im package `mvtnorm`)	`mvnorm`	`mean = rep(0,`*`d`*`), sigma = diag(`*`d`*`)` (mit *d* = Dimension)
Multivariate t (im package `mvtnorm`)	`mvt`	(etwas komplizierter; siehe seine Online-Hilfe)
Normal	`norm`	`mean = 0, sd = 1`
Students t	`t`	`df, ncp = 0`
Uniforme	`unif`	`min = 0, max = 1`
Weibull	`weibull`	`shape, scale = 1`

Hinweise: Mehr Informationen über die einzelnen Verteilungen – wie z. B. die Beziehung zwischen den obigen **R**-Funktionsargumenten und der mathematischen Parametrisierung der Dichten – liefert die Online-Hilfe, die etwa mit dem Kommando `?d`*dist* konsultiert werden kann, wenn etwas über die Verteilung namens *dist* (vgl. obige Tabelle) in Erfahrung gebracht werden soll. Beachte: `?`*dist* funktioniert *nicht* oder liefert nicht das Gewünschte! Aber `?distribution` liefert eine Übersicht über alle Verteilungen im Paket `stats` mit Links auf deren Hilfe-Seiten. (Zusätzliche Möglichkeit: Mittels `help.search( "distribution")` erhält man u. A. Hinweise

auf alle Hilfedateien, in denen etwas zum Stichwort "distribution" steht.) Ist man an einem umfangreichen Überblick über die in "base-**R**" und anderen **R**-Paketen zur Verfügung gestellten Verteilungen interessiert, lohnt sich ein Blick auf den Task View "Probability Distributions" auf CRAN unter `http://cran.r-project.org` → Task Views → Distributions.

Ergänzung: Urnenmodelle sind in **R** ebenfalls realisierbar, und zwar mithilfe der Funktion `sample()`. Mit ihr kann das Ziehen von Elementen aus einer (endlichen) Grundmenge mit oder ohne Zurücklegen simuliert werden. Es kann auch festgelegt werden, mit welcher Wahrscheinlichkeit die einzelnen Elemente der Grundmenge jeweils gezogen werden sollen, wobei die Bedeutung dieser Festlegung beim Ziehen mit Zurücklegen eine andere ist als beim Ziehen ohne Zurücklegen. Die Voreinstellung von `sample()` produziert eine zufällige Permutation der Elemente der Grundmenge. In Abschnitt 9.1 „Bernoulli-Experimente mit `sample()`" gehen wir etwas näher darauf ein; vorerst verweisen wir für Details auf die Online-Hilfe.

5.2 Bemerkungen zu Pseudo-Zufallszahlen in R

Die Generierung von Pseudo-Zufallszahlen aus den verschiedenen Verteilungen basiert auf uniformen Pseudo-Zufallszahlen, welche mit einem Zufallszahlengenerator (Engl.: "random number generator" = RNG) erzeugt werden. In **R** stehen im Prinzip mehrere verschiedene RNGs zur Verfügung. Per Voreinstellung ist es der „Mersenne-Twister", ein uniformer RNG, der eine sehr lange, aber letztendlich doch periodische Zahlensequenz (der Länge $2^{19937} - 1$) im offenen Intervall $(0, 1)$ erzeugt. Diesbzgl. wird in der Online-Hilfe die Publikation [59, Matsumoto und Nishimura (1998)] zitiert. Weitere Details und zahlreiche Literaturverweise liefert `?RNG`.

Der Zustand des RNGs wird von **R** in dem Objekt `.Random.seed` im workspace (gewissermaßen unsichtbar) gespeichert. Vor dem **aller**ersten Aufruf des RNGs existiert `.Random.seed` jedoch noch nicht, z. B. wenn **R** in einem neuen Verzeichnis zum ersten Mal gestartet wird. Den Startzustand des RNGs leitet **R** dann aus der aktuellen Uhrzeit ab, zu der der RNG zum ersten Mal aufgerufen wird. Bei der Erzeugung von Zufallszahlen ändert sich der Zustand des RNGs und der jeweils aktuelle wird in `.Random.seed` dokumentiert. Daher führen wiederholte Aufrufe des RNGs, insbesondere in verschiedenen **R**-Sitzungen, zu verschiedenen Zufallszahlen(folgen).

Für die Überprüfung und den Vergleich von Simulationen ist es jedoch notwendig, Folgen von Zufallszahlen reproduzieren (!) zu können. Um dies zu erreichen, kann der Zustand des RNGs von der Benutzerin oder dem Benutzer gewählt werden, wozu die Funktion `set.seed()` dient. Wird sie vor dem Aufruf einer der obigen `r....`-Funktionen (und unter Verwendung desselben RNGs) jedesmal mit demselben Argument (einem `integer`-Wert) aufgerufen, so erhält man stets die gleiche Folge an Zufallszahlen. Beispiel:

```
> runif( 3)                          # Drei uniforme Zufallszahlen (aus dem
[1] 0.1380366 0.8974895 0.6577632    # RNG mit unbekanntem Startzustand).

> set.seed( 42)                      # Wahl eines Startzustandes des RNGs.
> runif( 3)                          # Drei weitere uniforme Zufallszahlen.
[1] 0.9148060 0.9370754 0.2861395

> set.seed( 42)                      # Wahl *desselben* RNG-Startzustandes
> runif( 3)                          # wie eben => Replizierung der drei
[1] 0.9148060 0.9370754 0.2861395    # uniformen Zufallszahlen von eben.

> runif( 3)                          # Ausgehend vom aktuellen (uns "unbe-
[1] 0.8304476 0.6417455 0.5190959    # kannten") Zustand liefert der RNG
                                     # drei andere uniforme Zufallszahlen.
```

6 Programmieren in R

Schon in "base **R**" (also **R**s „Grundausstattung") gibt es eine Unmenge eingebauter Funktionen, von denen wir bisher nur einen Bruchteil kennengelernt haben. Viele der Funktionen sind durch eine Reihe von Optionen über Argumente im Detail ihrer Arbeitsweise modifizierbar, aber sie sind nichtsdestotrotz vorgefertigte „Konserven". Häufig ist es gewünscht oder nötig, eine solche Funktion mit einer langen Liste von speziellen Optionen oder eine ganze Gruppe von Funktionen in immer derselben Abfolge wiederholt aufzurufen. Auch sind gelegentlich (beispielsweise bei der Simulation statistischer Verfahren) problemspezifische Algorithmen umzusetzen, was keine der eingebauten Funktionen allein kann, sondern nur eine maßgeschneiderte Implementation. Diese Ziele lassen sich durch das Programmieren neuer Funktionen in **R** erreichen.

6.1 Definition neuer Funktionen: Ein Beispiel

Anhand einer Beispielfunktion für die Explorative Datenanalyse (EDA) sollen die Grundlagen der Definition neuer, problemspezifischer Funktionen erläutert werden: Viele inferenzstatistische Verfahren hängen stark von Verteilungsannahmen über die Daten ab, auf die sie angewendet werden sollen. Für die Beurteilung, ob diese Annahmen gültig sind, ist die EDA hilfreich. Sie verwendet grafische Methoden, die wir zum Teil bereits kennengelernt haben, und es kann sinnvoll sein, diese in einer neuen Funktion zusammenzufassen.

Als Beispieldaten verwenden wir Messungen des amerikanischen Physikers A. A. Michelson aus dem Jahr 1879 zur Bestimmung der Lichtgeschwindigkeit in km/s (die folgenden Werte resultieren aus den eigentlichen Geschwindigkeitswerten durch Subtraktion von 299000 km/s):

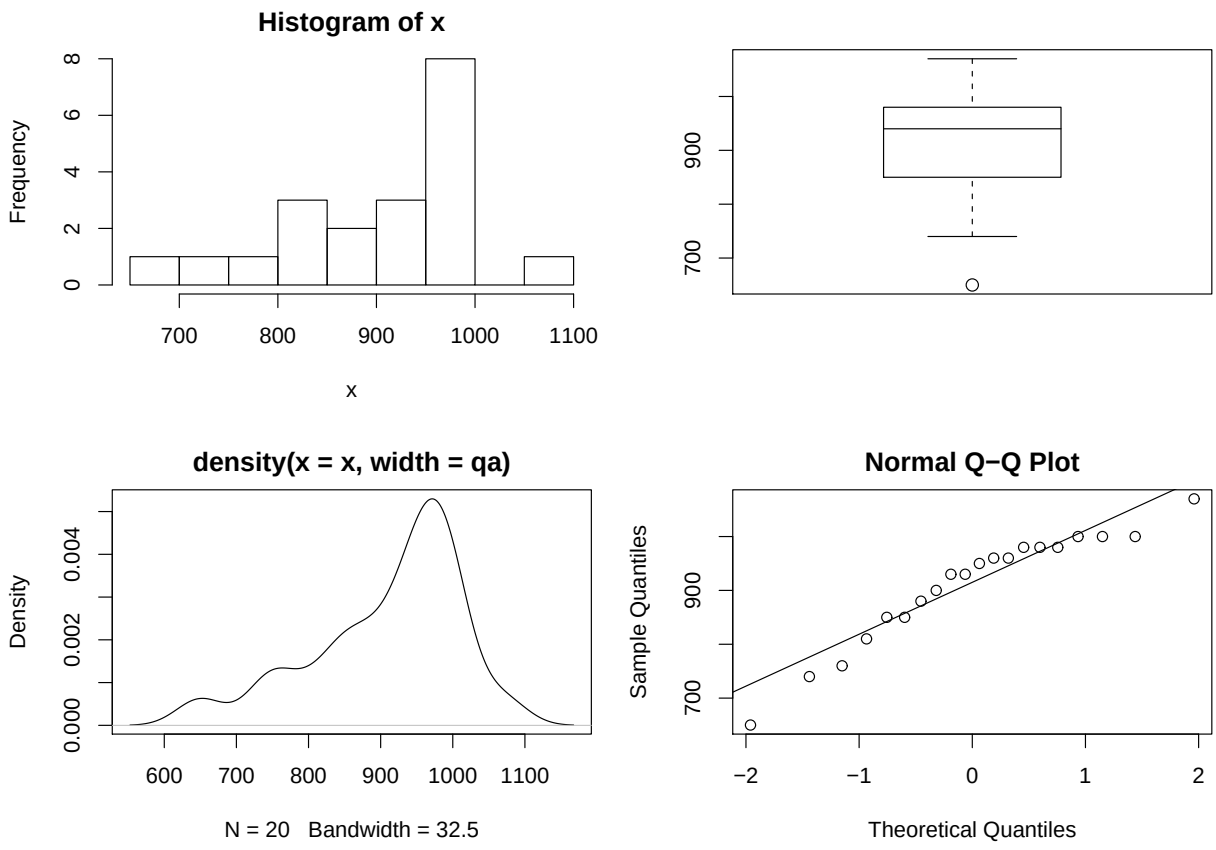
```
> v <- c( 850, 740, 900, 1070, 930, 850,  950,  980, 980, 880,
+        1000, 980, 930,  650, 760, 810, 1000, 1000, 960, 960)
```

Eine neu definierte Funktion für die explorative Datenanalyse

```
> eda <- function( x) {
+ par( mfrow = c( 2,2),
+      cex = 1.2)
+ hist( x)
+ boxplot( x)
+ qa <- diff( quantile( x,
+             c( 1/4, 3/4)))
+ dest <- density( x,
+                 width = qa)
+ plot( dest)
+ qqnorm( x)
+ qqline( x)
+ summary( x)
+ }

> eda( v)
 Min. 1st Qu. Median Mean
  650     850    940  909

 3rd Qu. Max.
    980 1070
```

Die Zuweisung `eda <- function( x) {....}` definiert eine Funktion namens `eda`, die bei ihrem Aufruf ein funktionsintern mit `x` bezeichnetes Argument erwartet. Die in `{....}` stehenden Befehle werden beim Aufruf von `eda()` sequenziell abgearbeitet:
Erst „formatiert" `par()` ein Grafikfenster passend (und öffnet es, wenn noch keins offen war; Details hierzu in Kapitel 7). Als nächstes wird ein Histogramm für die Werte in `x` geplottet (womit klar ist, dass der an `x` übergebene Wert ein `numeric`-Vektor sein muss). Als drittes entsteht ein Boxplot. Dann wird der Quartilsabstand der Daten berechnet und funktionsintern in `qa` gespeichert. Hernach bestimmt `density()` einen Kern-Dichteschätzer (per Voreinstellung mit Gaußkern), wobei `qa` für die Bandbreite verwendet wird; das Resultat wird `dest` zugewiesen und dann geplottet. Des Weiteren wird ein Normal-Q-Q-Plot mit Soll-Linie angefertigt. Zuletzt werden arithmetische "summary statistics" berechnet und da dies der letzte Befehl in `eda()` ist, wird dessen Ergebnis als Resultatwert von `eda()` an die Stelle des Funktionsaufrufes zurückgegeben (wie `eda( v)` zeigt).

Das Resultat der Anwendung der Funktion `eda()` auf `v` ist die Ausgabe der "summary statistics" für `v` und als Nebenwirkung die folgenden Grafiken in einem Grafikfenster:



6.2 Syntax der Funktionsdefinition

Allgemein lautet die Syntax der Definition einer neuen **R**-Funktion

neuefunktion `<-` `function(` *argumenteliste*`)` `{` *funktionsrumpf* `}`

Dabei ist

- *neuefunktion* der (im Prinzip frei wählbare) Name des Objektes, als das die neue Funktion gespeichert wird und das zur Klasse `function` zählt;
- `function` ein reservierter **R**-Ausdruck;
- *argumenteliste* die stets in runde Klammern `( )` zu packende, möglicherweise leere Argumenteliste von durch Kommata getrennten Argumenten, die beim Aufruf der Funktion mit Werten versehen und innerhalb der neuen Funktion verwendet werden (können);
- *funktionsrumpf* der Funktionsrumpf, welcher aus einem zulässigen **R**-Ausdruck oder einer Sequenz von zulässigen **R**-Ausdrücken besteht, die durch Semikola oder Zeilenumbrüche getrennt sind. Er ist, wenn er aus mehr als einem Ausdruck besteht, notwendig in geschweifte Klammern `{ }` zu packen; bei nur einem Ausdruck ist die Verwendung von `{ }` zwar nicht nötig, aber dennoch empfehlenswert.

6.3 Verfügbarkeit einer Funktion und ihrer lokalen Objekte

Durch die Ausführung obiger **R**-Anweisung wird unter dem Namen *neuefunktion* die *argumenteliste* und die Gesamtheit der in *funktionsrumpf* stehenden **R**-Ausdrücke als **ein** neues Objekt der Klasse `function` zusammengefasst und im aktuellen workspace gespeichert. In unserem Eingangsbeispiel ist dies das `function`-Objekt mit dem Namen `eda`.

Die in *funktionsrumpf* stehenden **R**-Ausdrücke dürfen insbesondere auch Zuweisungsanweisungen sein. Diese Zuweisungsanweisungen haben i. d. R. jedoch nur „lokalen“ Charakter, d. h., sie sind außerhalb der Funktion ohne Effekt und werden nach dem Abarbeiten der Funktion im Allgemeinen wieder vergessen. Mit anderen Worten, jedes innerhalb des Funktionsrumpfs *erzeugte* Objekt (auch wenn sein Name mit dem eines außerhalb der Funktion bereits existierenden Objekts übereinstimmt) hat eine rein temporäre und völlig auf das „Innere“ der Funktion eingeschränkte, eigenständige Existenz. Diese Objekte werden daher auch lokale Variablen oder Objekte genannt. In unserem `eda`-Beispiel sind `qa` und `dest` solche Objekte, die nur während der Abarbeitung der Funktion existieren und auch nur innerhalb des Rumpfes der Funktion bekannt sind.

6.4 Rückgabewert einer Funktion

Jede **R**-Funktion liefert einen Rückgabewert. Er ist das Resultat, d. h. der Wert des im Funktionsrumpf als letztem ausgeführten Ausdrucks, wenn nicht vorher ein Aufruf der Funktion `return()` erreicht wird, der die Abarbeitung des Funktionsrumpfes sofort beendet und deren Argument dann der Rückgabewert ist. Sind mehrere Werte/Objekte zurückzugeben, können sie in eine Liste (z. B. mit benannten Komponenten) zusammengefasst werden.

War die Aufrufstelle der Funktion der Prompt der **R**-Console und ist keine andere Vorkehrung getroffen, wird der Rückgabewert der Funktion einfach in der Console angezeigt und vergessen. Soll er *nicht* angezeigt werden, so ist im Funktionsrumpf der Ausdruck, dessen Wert zurückzugeben ist, in die Funktion `invisible()` einzupacken. Nichtsdestotrotz wird der Rückgabewert der Funktion an die Aufrufstelle übergeben, nur eben „unsichtbar“, d. h., er kann nach wie vor an ein Objekt zugewiesen und so gespeichert werden.

Beispiel: Im Fall von `eda()` in Abschnitt 6.1 ist der Rückgabewert das Ergebnis von `summary( x)`. Obiges bedeutet, dass der Aufruf `eda( v)` bzw. einer jeden neu definierten Funktion auch auf der rechten Seite einer Zuweisungsanweisung stehen darf: `michelson.summary <- eda( v)` ist also eine zulässige **R**-Anweisung, die als Haupteffekt dem Objekt `michelson.summary` den Rückgabewert des Funktionsaufrufs `eda( v)`, also die “summary statistics” für `v` zuweist und als Nebeneffekt die jeweiligen Grafiken erzeugt. Sollte das Ergebnis von `eda()`, also `summary( x)`, unsichtbar zurückgegeben werden, müsste der letzte Ausdruck in `eda()`s Rumpf `return( invisible( summary( x)))` oder kürzer `invisible( summary( x))` lauten. In diesem Fall hätte `michelson.summary <- eda( v)` also dieselbe Wirkung wie zuvor, aber `eda( v)` allein würde den Rückgabewert einfach „auf Nimmerwiedersehen“ verschwinden lassen.

6.5 Spezifizierung von Funktionsargumenten

Es ist guter und sicherer Programmierstil, möglichst alle Informationen, die innerhalb des Funktionsrumpfs verarbeitet werden sollen, durch die Argumenteliste des Funktionsaufrufs an die Funktion zu übergeben (und nicht von innen auf „außerhalb“ der Funktion vorhandene Objekte zuzugreifen). Dadurch wird die Argumenteliste jedoch schnell zu einem recht langen „Flaschenhals“, sodass eine gut durchdachte Argumenteliste entscheidend zur Flexibilität und Praktikabilität einer Funktion beitragen kann.

Die in der Argumenteliste einer Funktions*definition* auftretenden Argumente sind ihre Formalparameter und die beim Funktions*aufruf* tatsächlich an die Argumenteliste übergebenen Objekte heißen Aktualparameter. Entsprechend haben Formalparameter Formalnamen und die Aktualparameter Aktualnamen.

6.5.1 Argumente mit default-Werten

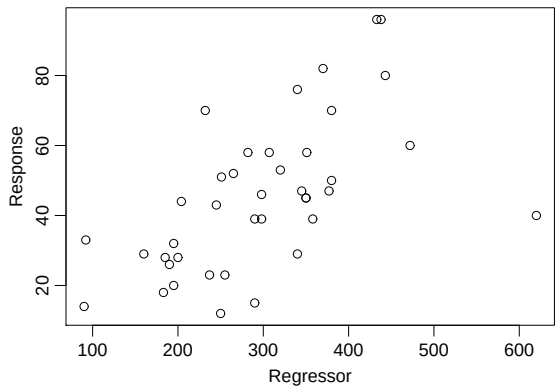
Um eine Funktion großer Flexibilität zu konstruieren, ist in ihrer Definition häufig eine umfangreiche Argumenteliste vonnöten, was ihren Aufruf rasch recht aufwändig werden lässt. Oft sind jedoch für mehrere Argumente gewisse Werte als dauerhafte „Voreinstellungen“ wünschenswert, die nur in gelegentlichen Aufrufen durch andere Werte zu ersetzen sind. Um diesem Aspekt Rechnung zu tragen, ist es schon in der Funktionsdefinition möglich, Argumenten Voreinstellungswerte (= “default values”) zuzuweisen, sodass diese bei einem konkreten Aufruf der Funktion nicht in der Argumenteliste des Aufrufs angegeben zu werden brauchen (wenn die voreingestellten Werte verwendet werden sollen). Dazu sind den jeweiligen Formalparametern in der Argumenteliste die gewünschten Voreinstellungswerte mittels Ausdrücken der Art *formalparameter* = *wert* zuzuweisen.

In §4.3.2 hatten wir auf Seite 81 das Beispiel eines Streudiagramms für die Werte in zwei Vektoren `X1` und `X5`, in dem die x- und y-Achsenbeschriftung durch die Argumente `xlab` und `ylab` der Funktion `plot()` festgelegt wurden. Angenommen, wir sind hauptsächlich an Streudiagrammen im Zusammenhang mit der einfachen linearen Regression interessiert und wollen häufig die Beschriftung „Regressor“ und „Response“ für die x- bzw. y-Achse verwenden. Dann könnten wir uns eine Funktion `rplot()` wie folgt definieren:

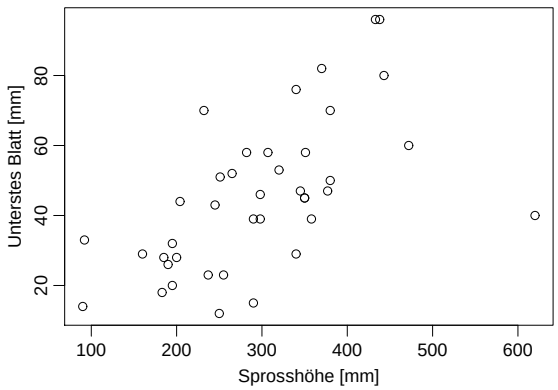
```
> rplot <- function( x, y, xlabel = "Regressor", ylabel = "Response") {
+ plot( x, y, xlab = xlabel, ylab = ylabel)
+ }
```

In der Definition von `rplot()` sind die zwei Argumente `xlabel` und `ylabel` auf die default-Werte „Regressor“ bzw. „Response“ voreingestellt und brauchen deshalb beim Aufruf von `rplot()` in seiner Argumenteliste nicht mit Werten versehen zu werden, wenn diese Achsenbeschriftung verwendet werden soll. Wenn jedoch `xlabel` und `ylabel` im Aufruf von `rplot()` mit Werten versehen werden, „überschreibt“ dies (temporär!) deren default-Werte. Vergleiche die beiden folgenden Aufrufe und die resultierenden Plots darunter:

```
> rplot( X1, X5)
```




```
> rplot( X1, X5,
+ xlabel = "Sprosshöhe [mm]",
+ ylabel = "Unterstes Blatt [mm]")
```



6.5.2 Variable Argumentezahl: Das „Dreipunkteargument“

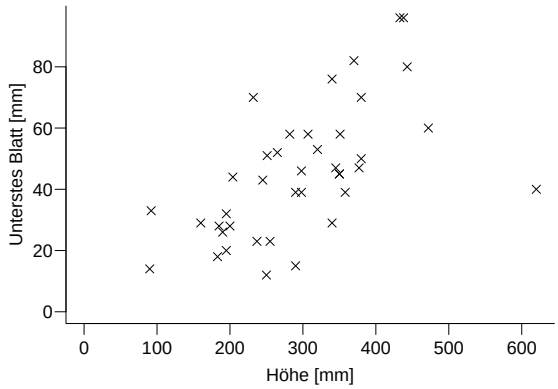
Gelegentlich ist es nötig, innerhalb einer neu definierten Funktion wie `rplot()` eine weitere Funktion (oder mehrere Funktionen) aufzurufen, an die gewisse Argumente von der Aufrufstelle von `rplot()` einfach nur „durchgereicht“ werden sollen, ohne dass sie durch `rplot()` verändert oder benutzt werden. Insbesondere kann auch die Anzahl der durchzureichenden Argumente variieren. Um dies zu ermöglichen, muss in der Argumenteliste der Funktionsdefinition von

`rplot()` der spezielle, „Dreipunkteargument“ (oder “dot-dot-dot”) genannte Formalparameter „`...`“ stehen. Er spezifiziert eine variable Anzahl beliebiger Argumente für diese Funktion (und zwar zusätzlich zu den in der Argumenteliste bereits angegebenen). In einem solchen Fall können dieser Funktion unter Verwendung der Syntax *formalparameter* `=` *wert* beliebige und beliebig viele Argumente übergeben werden. Innerhalb des Funktionsrumpfes wird der Formalparameter „`...`“ typischerweise nur in den Argumentelisten weiterer Funktionsaufrufe verwendet (siehe aber auch den knappen Hinweis zum Dreipunkteargument auf Seite 96 oben).

In unserem Beispiel `rplot()` könnten durch eine solche Erweiterung beispielsweise Argumente an die Funktion `plot()` durchgereicht werden, die das Layout des zu zeichnenden Koordinatensystems, die zu verwendenden Linientypen, Plot-Zeichen usw. beeinflussen:

```
> rplot <- function( x, y, xlabel = "Regressor", ylabel = "Response", ...) {
+ plot( x, y, xlab = xlabel, ylab = ylabel, ...)
+ }
```

```
> rplot( X1, X5,
+ xlabel = "Höhe [mm]",
+ ylabel = "Unterstes Blatt [mm]",
+ xlim = c( 0, max( X1)),
+ ylim = c( 0, max( X5)),
+ las = 1, bty = "l", pch = 4)
```



In dem obigen Aufruf von `rplot()` werden die Aktualparameter `X1`, `X5`, `xlabel` und `ylabel` von `rplot()` „abgefangen“, weil die ersten beiden über ihre Position im Funktionsaufruf den beiden ersten Formalparametern `x` und `y` von `rplot()` zugewiesen werden und die beiden anderen mit Formalnamen in der Argumenteliste von `rplot()` in Übereinstimmung gebracht werden können. Die Argumente `xlim`, `ylim`, `las`, `bty` und `pch` sind in der Argumenteliste von `rplot()` nicht aufgeführt und werden daher vom Dreipunkteargument „`...`“ übernommen und im Rumpf von `rplot()` unverändert an `plot()` weitergereicht. (Zur Bedeutung der Layout-Parameter `xlim` bis `pch` siehe Kapitel 7 und zu den Details der Parameterübergabe den folgenden Abschnitt.)

6.5.3 Zuordung von Aktual- zu Formalparametern beim Funktionsaufruf

Hier zählen wir **R**s wesentliche Regeln und ihre Anwendungsreihenfolge für die Zuordnung von Aktual- zu Formalparametern beim Funktionsaufruf auf, wobei die vier Schritte anhand der Funktion `mean( x, trim = 0, na.rm = FALSE, ...)` und eines Beispielvektors `z <- c( 9, 14, NA, 10, 8, 7, 5, 11, 1, 3, 2)` erläutert werden:

1. **Zuordnung über vollständige Formalnamen**: Zunächst wird für jedes Argument der Bauart *argumentname* `=` *wert*, bei dem *argumentname* vollständig mit einem Formalnamen in der Argumenteliste der Funktion übereinstimmt, *wert* diesem Formalparameter zugewiesen. Die Reihenfolge der Argumente mit vollständigen Formalnamen ist dabei im Funktionsaufruf irrelevant:

   ```
   > mean( na.rm = TRUE, x = z)
   [1] 7
   ```

2. **Zuordnung über unvollständige Formalnamen**: Sind nach Schritt 1 noch Argumente der Art *argumentname* = *wert* übrig, so wird für jedes verbliebene Argument, dessen *argumentname* mit den Anfangszeichen eines noch „unversorgten“ Formalnamens übereinstimmt, *wert* dem entsprechenden Formalparameter zugewiesen. Auch hier spielt die Reihenfolge der Argumente im Funktionsaufruf keine Rolle:

```
> mean( na.rm = TRUE, x = z, tr = 0.1)
[1] 6.875
```

```
> mean( x = z, na = TRUE, tr = 0.1)
[1] 6.875
```

3. **Zuordnung über Argumentepositionen**: Sind auch nach Schritt 2 noch Argumente übrig, werden die Werte **unbenannter** Argumente von links beginnend, der Reihe nach den noch übrigen „unversorgten“ Formalparametern zugewiesen:

```
> mean( na.rm = TRUE, tr = 0.1, z)
[1] 6.875
```

```
> mean( z, 0.1, TRUE)
[1] 6.875
```

4. **Zuordnung über das Dreipunkteargument**: Sind nach dem 3. Schritt immer noch Argumente übrig, werden sie dem Dreipunkteargument zugewiesen, falls es in der Funktionsdefinition vorhanden ist, ansonsten gibt es eine Fehlermeldung:

```
> mean( z, na = TRUE, nochwas = "Unsinn")          # Offenbar besitzt mean()
[1] 7                                               # das Dreipunkteargument,
> median( z, na = TRUE, nochwas = "Unsinn")        # aber median() nicht.
Error in median(z, na = TRUE, nochwas = "Unsinn") :
        unused argument(s) (nochwas ...)
```

Beachte: Für benannte Argumente *hinter* dem Dreipunkteargument klappt eine Zuordnung über unvollständige Formalnamen nicht, d. h. deren Argumentnamen sind beim Funktionsaufruf nicht abkürzbar.

6.5.4 Nützliches für den Zugriff auf Argumentelisten und Argumente sowie auf den Quellcode existierender Funktionen

Es folgen einige **knappe Hinweise** zur Existenz von Funktionen, die für die „direkte“ Arbeit mit Argumentelisten und Argumenten einer Funktion im Rumpf derselben nützlich sein können. Details sind unbedingt der Online-Hilfe zu entnehmen.

- Argumente können einen Vektor als default-Wert haben, dessen Elemente mit Hilfe von `match.arg()` auf „Passgenauigkeit“ mit dem beim Funktionsaufruf übergebenen Aktualparameter untersucht werden können, was einerseits sehr flexible und gleichzeitig äußerst knappe Funktionsaufrufe ermöglicht und andererseits die Wertemenge zulässiger Aktualparameter beschränkt und eine „integrierte Eingabefehlerkontrolle“ implementiert:

```
> rplot <- function( x, y, xlabel = c( "Regressor", "Unabhängige Variable"),
+                    ylabel = c( "Response", "Abhängige Variable"), ...) {
+ xlabel <- match.arg( xlabel)
+ ylabel <- match.arg( ylabel)
+ plot( x, y, xlab = xlabel, ylab = ylabel, ...)
+ }
```

  `match.arg()` prüft, ob und wenn ja, zu welchem der default-Werte der tatsächlich übergebene Aktualparameter passt und liefert den (nötigenfalls vervollständigten) Wert zurück.

- `hasArg()` und `missing()` dienen im Rumpf einer Funktion der Prüfung, ob Formalparameter beim Aufruf jener Funktion mit Aktualparametern versorgt wurden bzw. ob diese fehlten.

`formals()` und `args()` liefern die gesamte Argumenteliste einer Funktion in zwei verschiedenen Formaten.

- Zugriff auf den „Inhalt“ des Dreipunktearguments erhält man mittels `list(...)`. Das Ergebnis ist eine Liste mit den in `...` auftauchenden Paaren *argumentname* = *wert* als Komponenten.

  Evtl. recht instruktiv für die Funktionsweise des Dreipunktearguments könnte auch die kleine Aufgabe in einer E-Mail von Bert Gunter vom 30. 1. 2013 auf R-help sein, zu finden z. B. unter `http://article.gmane.org/gmane.comp.lang.r.general/285906/match=how+to+use`.

- `deparse()` und `substitute()` in geeigneter Kombination im Rumpf einer Funktion erlauben, an die Aktualnamen der an die Formalparameter übergebenen Aktualparameter heranzukommen, was z. B. genutzt werden kann für die flexible Beschriftung von Grafiken, die „innerhalb“ von benutzereigenen Funktionen angefertigt werden:

  ```
  > myplot <- function( x, y) {
  + plot( x, y, xlab = deparse( substitute( x)), ylab = deparse( substitute( y)))
  + }
  ```

  `deparse( substitute( y))` ermittelt in obigem Fall z. B. die Zeichenkette, die den Objektnamen des an den Formalparameter `y` übergebenen Aktualparameter darstellt.

- Will man sich den Quellcode einer *selbstdefinierten* Funktion ansehen, so erreicht man dies durch Eingabe des Funktionsnamens – ohne das Klammernpaar `()` – am `R`-Prompt (da der Code ja der Wert des `function`-Objektes mit dem betreffenden Namen ist). Für bereits in “base **R**” oder in **R**-Paketen existierende Funktionen ist es gelegentlich jedoch geringfügigst aufwändiger, an ihren Quellcode zu kommen, da sie aus technischen Gründen „versteckt“ sein können. In [53, Ligges (2006)] wird genauer beschrieben, was dahinter steckt und warum dann wie vorzugehen ist.

  Hier nur ein paar Beispiele:

  – Für die Funktion `sd()` zur Berechnung der Standardabweichung ist es ganz einfach. Die Eingabe ihres Namens liefert ihre Definition und eine Zusatzinformation (die hier aus Platzgründen nur unvollständig und leicht editiert abgedruckt werden):

    ```
    > sd
    function (x, na.rm = FALSE) {
        if (is.matrix(x)) {
            msg <- "sd(<matrix>) is deprecated.\n Use apply(*, 2, sd) instead."
            warning(paste(msg, collapse = ""), call. = FALSE, domain = NA)
            apply(x, 2, sd, na.rm = na.rm)
        } else
    ....
    ```

  – Bei `mean()` geschieht dies eigentlich auch, aber das Ergebnis ist – zunächst – wenig erhellend, denn es sagt uns letztendlich nur, dass es mehrere sogenannte Methoden für die „generische“ Funktion `mean()` gibt und eine adäquate aufgerufen wird:

    ```
    > mean
    function (x, ...)
    UseMethod("mean")
    <bytecode: 0x0555f5e0>
    <environment: namespace:base>
    ```

    Welche Methoden es gibt, bekommt man durch die Funktion `methods()` aufgelistet:

```
> methods( mean)
[1] mean.data.frame   mean.Date    mean.default   mean.difftime   mean.POSIXct
[6] mean.POSIXlt
```

Und die konkrete Abfrage der interessierenden Methode liefert schließlich ihre Definition (die aus Platzgründen hier ebenfalls nicht vollständig abgedruckt wird):

```
> mean.default
function (x, trim = 0, na.rm = FALSE, ...) {
    if (!is.numeric(x) && !is.complex(x) && !is.logical(x)) {
        warning("argument is not numeric or logical: returning NA")
        return(NA_real_)
    }
....
```

– Schwieriger wird's wenn einige der Methoden "invisible" sind, wie z. B. die `ecdf`-Methode der generischen Funktion `summary()`, nämlich `summary.ecdf()`, denn an sie kommt man nicht durch lediglich die Eingabe ihres Namens:

```
> methods( summary)
 [1] summary.aov            summary.aovlist        summary.aspell*
 [4] summary.connection     summary.data.frame     summary.Date
 [7] summary.default        summary.ecdf*          summary.factor
....
   Non-visible functions are asterisked

> summary.ecdf
Fehler: Objekt 'summary.ecdf' nicht gefunden
```

– Die Online-Hilfe zu `summary.ecdf()` zeigt uns jedoch (in der linken oberen Ecke ihrer Hilfeseite), dass diese Methode zum Paket `stats` gehört, sodass das Voranstellen des Paketnamens und zweier oder dreier Doppelpunkte den Zugriff auf den Funktionscode erlaubt:

```
> stats:::summary.ecdf
function (object, ...) {
    header <- paste("Empirical CDF:\t ", environment(object)$n,
        "unique values with summary\n")
....
```

6.6 Kontrollstrukturen: Bedingte Anweisungen, Schleifen, Wiederholungen

Die Abarbeitungsreihenfolge von **R**-Ausdrücken in einer Funktion ist sequenziell. Kontrollstrukturen wie bedingte Anweisungen und Schleifen erlauben Abweichungen davon:

Bedingte Anweisungen: `if`, `ifelse`, `switch`	

`> if(` *bedingung* `) {`
`+` *ausdruck1*
`+ } else {`
`+` *ausdruck2*
`+ }`
	Die `if`-Anweisung erlaubt die alternative Auswertung zweier Ausdrücke abhängig vom Wert des Ausdrucks *bedingung*, der ein skalares `logical`-Resultat liefern muss: ist es `TRUE`, wird *ausdruck1* ausgewertet, anderenfalls *ausdruck2*. Der „`else`-Zweig" (`else{....}`) kann weggelassen werden, wenn für den Fall *bedingung* = `FALSE` keine Aktion benötigt wird. Der Resultatwert der `if`-Anweisung ist der des tatsächlich ausgeführten Ausdrucks.

`> a <- if( n > 10) {`
`+  print( "Shift 1")`
`+  (1:n - 1/2)/n`
`+  } else {`
`+  print( "Shift 2")`
`+  (1:n - 3/8)/`
`+  (n + 1/4)`
`+  }`
	Die Bestandteile der gesamten „`if( ){ }else{ }`“-Struktur und insbesondere ihre vier geschweiften Klammern sollten stets so auf verschiedene Zeilen verteilt werden, wir links gezeigt, um Lesbarkeit und Eindeutigkeit der `if`-Struktur zu gewährleisten. (Handelt es sich bei *ausdruck1* oder *ausdruck2* um einen einzelnen Ausdruck, so können die geschweiften Klammern `{ }` jeweils weggelassen werden, was jedoch fehleranfällig ist.)

`> ifelse(` *test*`,`
`+` *ausdr1*`,` *ausdr2*`)`
 	
`> ifelse(`
`+  runif( 4) > 1/2,`
`+  letters[ 1:4],`
`+  LETTERS[ 1:4])`
`[1] "A" "b" "C" "D"`
	Die Funktion `ifelse()` ist eine vektorisierte Version von `if`, wobei *test* ein `logical`-Objekt ist oder ergibt; dieses gibt auch die Struktur des Resultates vor. *ausdr1* und *ausdr2* müssen vektorwertige Resultate derselben Länge wie *test* liefern (können also schon Vektoren sein). Für jedes Element von *test*, das `TRUE` ist, wird das korrespondierende Element von *ausdr1* in das entsprechende Element des Resultates eingetragen, für jedes Element, das `FALSE` ist, das korrespondierende Element von *ausdr2*.

`> switch(` *ausdr*`,`
`+` *argname.1* `=` *ausdr.1*`,`
`+  ....,`
`+` *argname.N* `=` *ausdr.N*
`+ )`
 	
`> switch( Verteilung,`
`+  normal = rnorm(1),`
`+  cauchy = rcauchy(1),`
`+  uniform = runif(1),`
`+  stop( "Unbekannt")`
`+  )`
	Die Funktion `switch()` ist, wie ihr Name nahelegt, ein Schalter, der abhängig vom Wert des (Steuer-)Ausdrucks *ausdr* die Ausführung von höchstens einem von mehreren, alternativen Ausdrücken veranlasst. Der Resultatwert von `switch()` ist der Wert des tatsächlich ausgeführten Ausdrucks: *ausdr* muss sich zu `numeric` oder `character` ergeben. Ist das Ergebnis `numeric` mit dem Wert i, wird *ausdr.i* ausgewertet; ist es `character`, so wird derjenige Ausdruck ausgeführt, dessen Formalname (*argname.1* bis *argname.N*) mit *ausdr* exakt übereinstimmt. Falls der Wert von *ausdr* nicht zwischen 1 und N liegt oder mit keinem Formalnamen übereinstimmt, wird `NULL` zurückgegeben oder, falls ein letzter unbenannter Ausdruck existiert, dessen Wert: In `switch(` *ausdr*`,` *f1* `=` *a1*`,` `....,` *fN* `=` *aN*`, aX)` wäre es dann also `aX`.

Die `for`-Schleife	

`> for(` *variable* `in` *werte*`) {`
`+` *ausdruck*
`+  }`
 	
`> X <- numeric( 100)`
`> e <- rnorm( 99)`
`> for( i in 1:99) {`
`+  X[ i+1] <- alpha *`
`+  X[ i] + e[ i]`
`+  }`
	Die `for`-Schleife weist *variable* sukzessive die Elemente in *werte* zu und führt dabei *ausdruck* jedes Mal genau einmal aus. *variable* muss ein zulässiger Variablenname sein, `in` ist ein notwendiges Schlüsselwort, *werte* ein Ausdruck, der ein `vector`-Objekt liefert, und *ausdruck* ein beliebiger Ausdruck (oder eine Sequenz von Ausdrücken). Innerhalb von *ausdruck* steht *variable* zur Verfügung, braucht aber nicht verwendet zu werden. Der Rückgabewert der gesamten `for`-Schleife ist der Wert von *ausdruck* in ihrem letzten Durchlauf.

Vielfach wiederholte Ausdruckauswertung mit `replicate()`	

`> replicate( n,` *ausdruck*`)`
 	
`> replicate( 10, {`
`+  x <- rnorm( 30)`
`+  y <- rnorm( 30)`
`+  plot( x, y)`
`+  } )`
	Für `n` $\in \mathbb{N}$ wird *ausdruck* `n`-mal ausgewertet und die `n` Ergebnisse in einer möglichst einfachen Struktur (z. B. als Vektor) zusammengefasst zurückgegeben. `replicate()` ist faktisch nur eine “wrapper”-Funktion für eine typische Anwendung von `sapply()` (vgl. S. 48) und insbesondere geeignet für Simulationen, in denen wiederholt Stichproben von Zufallszahlen zu generieren und jedes Mal auf dieselbe Art und Weise zu verarbeiten sind.

Hinweis: Für detailliertere Informationen und weitere Anwendungsbeispiele zu obigen Kontrollstrukturen bzw. auch zu anderen Schleifentypen (`repeat` und `while`) sowie Kontrollbefehlen für Schleifen (`next` und `break`) siehe die Online-Hilfe. Beachte: Um die Online-Hilfe zu `if` oder zur `for`-Schleife zu bekommen, müssen Anführungszeichen verwendet werden: `?"if"` bzw. `?"for"`.

Warnungen, Tipps & weitere – knappe – Hinweise:

- **Warnung:** `for`-Schleifen können in **R** *ziemlich ineffizient* und daher relativ langsam sein. Für gewisse, insbesondere iterative Berechnungen sind sie zwar nicht zu umgehen, sollten aber, wann immer möglich, vermieden und durch eine vektorisierte Version des auszuführenden Algorithmus ersetzt werden!

  Für unvermeidbar hoch-iterative (oder auch rekursive) Algorithmen ist in Erwägung zu ziehen, sie in C oder Fortran zu programmieren und die C- bzw. Fortran-Schnittstelle von **R** zu nutzen. Siehe hierzu die unten empfohlenen Bücher oder in der Online-Hilfe in der Dokumentation "Writing R Extensions" den Abschnitt 6 "System and foreign language interfaces" (vgl. §1.5.3, Bild 2).

  Was man *auf jeden Fall* zu vermeiden versuchen sollte, weil es extrem ineffizient (und schlechter **R**-Programmierstil) ist:

  - Vektorisierbare Berechnungen in einer Schleife der Art `for( i in ....) { y[ i] <- ....}` durchführen zu lassen.
  - Objekte (Vektoren, Matrizen, Listen etc.) durch oder in eine/r `for`-Schleife „wachsen", d. h. dynamisch entstehen zu lassen wie z. B. in `for( i in ....) { y <- c( y, ....)}`, wobei für `cbind()` oder `rbind()` anstelle von `c()` hierbei dasselbe gilt.

    Selbst wenn man nicht weiß, wie groß das resultierende Objekt schließlich genau sein wird, man aber immerhin eine obere Schranke kennt, so lohnt es sich oft, diese Schranke zu nutzen, indem man ein zu großes Objekt kreiert, durch geeignete Indizierung (nur teilweise) füllt und das erhaltene Objekt schließlich auf die endgültig benötigte Dimension verkleinert.

- Einige sehr nützliche Hilfsfunktionen, um Eigenschaften von Programmcode wie Schnelligkeit und Objektgrößen zu kontrollieren bzw. um die Fehlersuche zu erleichtern, sind die folgenden:

  - `system.time()` ermittelt die CPU-Zeit (und andere Zeiten), die zum Abarbeiten eines an sie übergebenen **R**-Ausdrucks benötigt wurde.
  - `object.size()` ergibt den ungefähren Speicherplatzbedarf in Bytes des an sie übergebenen Objektes.
  - `browse()` erleichtert die Fehlersuche z. B. im Rumpf einer Funktion, indem sie erlaubt, die Auswertungsumgebung der Stelle, an der sie aufgerufen wurde, zu inspizieren.

    `debug()` ermöglicht noch mehr, z. B. das „schrittweise" Abarbeiten der Ausdrücke in einem Funktionsrumpf und (ebenfalls) das zwischenzeitliche Inspizieren sowie nötigenfalls sogar das Modifizieren der dabei verwendeten oder erzeugten Objekte.

    Für die Funktionsweise und Nutzung von `browse()` und `debug()` siehe jeweils ihre Online-Hilfe.
  - `Rprof()` erlaubt das zeitlich diskrete, aber relativ engmaschige "profiling" (eine Art Protokollieren) der Abarbeitung von **R**-Ausdrücken. Siehe die Online-Hilfe.

- Gute und nützliche, einführende, aber auch schon tiefgehende Bücher zur Programmierung in **R** bzw. **S** sind „Programmieren mit R" ([54, Ligges (2008)]) und "S Programming" ([80, Venables & Ripley (2004)]).

  Noch tiefer- und weitergehend sind "Software for Data Analysis: Programming with R" ([17, Chambers (2008)]) und das exzellente "The Art of R Programming" ([60, Matloff (2011)]).

7 Weiteres zur elementaren Grafik

Die leistungsfähige Grafik von **R** zählt zu seinen wichtigsten Eigenschaften. Um sie zu nutzen, bedarf es eines speziellen Fensters bzw. einer speziellen Ausgabedatei zur Darstellung der grafischen Ausgabe. Wie bereits gesehen, gibt es viele Grafikfunktionen, die gewisse Voreinstellungen besitzen, welche es erlauben, ohne großen Aufwand aufschlussreiche Plots zu erzeugen. Grundlegende Grafikfunktionen, wie sie im Folgenden beschrieben werden, ermöglichen dem/der BenutzerIn die Anfertigung sehr spezieller Plots unter Verwendung verschiedenster Layoutfunktionen und Grafikparameter. Für die *interaktive* Nutzung der Grafikausgabe stehen ebenfalls (mindestens) zwei Funktionen zur Verfügung.

7.1 Grafikausgabe

Zur Vollständigkeit sind hier nochmal die in 4.1 aufgeführten Funktionen wiederholt. (Beachte auch die dortigen Hinweise.)

Öffnen und Schließen von Grafik-Devices:	

`> X11()`
`> windows()`
.... (Grafikbefehle) ....
	`X11()` öffnet unter Unix und `windows()` unter Windows bei jedem Aufruf ein neues Grafikfenster. Das zuletzt geöffnete Device ist das jeweils aktuell *aktive*, in welches die Grafikausgabe erfolgt.

`> dev.list()`	Listet Nummern und Typen aller geöffneten Grafik-Devices auf.
`> dev.off()`	Schließt das zuletzt geöffnete Grafik-Device *ordnungsgemäß*.
`> graphics.off()`	Schließt alle geöffneten Grafik-Devices auf einen Schlag.
`> postscript( file)` .... (Grafikbefehle)	Öffnet bei jedem Aufruf eine neue (EPS- (= "Encapsulated PostScript"-) kompatible) PostScript-Datei mit dem als Zeichenkette an `file` übergebenen Namen. Das zuletzt geöffnete Device ist das jeweils aktuelle, in welches die Grafikausgabe erfolgt.
`> dev.off()`	Schließt das zuletzt geöffnete Grafik-Device und *muss* bei einer PostScript-Datei unbedingt zur Fertigstellung verwendet werden, denn erst danach ist sie vollständig und korrekt interpretierbar.

`> pdf( file)`
.... (Grafikbefehle) ....
`> dev.off()`
	Völlig analog zu `postscript()`, aber eben für PDF-Grafikdateien.

7.2 Elementare Zeichenfunktionen: `plot()`, `points()`, `lines()` & Co.

Eine Funktion zur „Erzeugung" eines Streudiagramms samt Koordinatensystem (= „Kosy"):

Koordinatensysteme & Streudiagramme	

`> x <- runif( 50)`
`> y <- 2*x + rnorm( 50)`
`> z <- runif( 50)`
`> X11()`
	(Synthetische Beispieldaten einer einfachen linearen Regression $y_i = 2x_i + \varepsilon_i$ mit ε_i i.i.d. $\sim \mathcal{N}(0,1)$, weiterer z_i i.i.d. $\sim \mathcal{U}(0,1)$ und Öffnen eines neuen Grafikfensters.)

`> plot( x, y)`
 	
`> plist <- list( x = x, y = y)`
`> plot( plist)`
 	
`> pmat <- cbind( x, y)`
`> plot( pmat)`
	Erstellt ein Kosy und zeichnet („plottet") zu den `numeric`-Vektoren `x` und `y` das Streudiagramm der Punkte (`x[i], y[i]`) für i=1, ..., `length( x)`. Statt `x` und `y` kann eine Liste übergeben werden, deren Komponenten `x` und `y` heißen, oder eine zweispaltige Matrix, deren erste Spalte die x- und deren zweite Spalte die y-Koordinaten enthalten.

`> DF <- data.frame( y, x, z)` `> plot( y ~ x + z, data = DF)`	`plot()` akzeptiert auch eine Formel (vgl. `boxplot()` und `stripchart()` in §4.2.2 auf S. 76): `y ~ x1 + .... + xk` bedeutet, dass die „linke“ `numeric`-Variable auf der Ordinate abgetragen wird und die „rechte(n)“ `numeric`-Variable(n) entlang der Abszisse eines jeweils eigenen Kosys. Quelle der Variablen ist der Data Frame für `data`.
`> plot( y)` `> plot( y ~ 1, data = DF)`	Für einen Vektor allein oder die Formel `y ~ 1` wird das Streudiagramm der Punkte (`i, y[i]`) gezeichnet.

Funktionen, mit denen einfache grafische Elemente in ein *bestehendes* Koordinatensystem eingezeichnet werden können:

Punkte, Linien, Pfeile, Text & Gitter in ein Koordinatensystem einzeichnen	

`> points( x, y)`
`> points( plist)`
`> points( pmat)`
`> points( x)`
	In ein *bestehendes* (durch `plot()` erzeugtes) Kosy werden für die `numeric`-Vektoren `x` und `y` an den Koordinaten (`x[i], y[i]`) Punkte eingetragen. Ebenso können (wie bei `plot()`) eine Liste (mit Komponenten namens `x` und `y`) oder eine zweispaltige Matrix verwendet werden. Für alleiniges `x` werden die Punkte an (`i, x[i]`) eingezeichnet.

`> lines( x, y)`
`> lines( plist)`
`> lines( pmat)`
`> lines( x)`
	Zeichnet einen Polygonzug durch die Punkte (`x[i], y[i]`) in ein *existierendes* Kosy. Eine x-y-Liste, eine zweispaltige Matrix oder ein alleiniges `x` funktionieren analog zu `points()`.

`> abline( a, b)`
`> abline( a)`
`> abline( h = const)`
`> abline( v = const)`
	Zeichnet eine Gerade mit y-Achsenabschnitt `a` und Steigung `b` in ein *bestehendes* Kosy ein. Bei alleiniger Angabe eines *zwei*elementigen Vektors `a` wird `a[1]` der y-Achsenabschnitt und `a[2]` die Steigung. Alleinige Angabe von `h = const` liefert horizontale Geraden mit den Elementen des `numeric`-Vektors `const` als Ordinaten; `v = const` führt zu vertikalen Geraden mit den Abszissen aus `const`.

`> segments( x0, y0,` `+ x1, y1)`	Fügt für die gleich langen `numeric`-Vektoren `x0`, `y0`, `x1` und `y1` in ein *vorhandenes* Kosy für jedes i=1, ..., `length( x0)` eine Strecke von (`x0[i], y0[i]`) bis (`x1[i], y1[i]`) ein.
`> arrows( x0, y0,` `+ x1, y1)`	`arrows()` wirkt wie `segments()`; zusätzlich wird an jedem Endpunkt (`x1[i], y1[i]`) eine Pfeilspitze gezeichnet. Durch `length` wird die Größe (in Zoll) des bzw. der zu zeichnenden Pfeilspitzen gesteuert; die Voreinstellung ist mit 0.25 Zoll ziemlich groß. (Eine Angabe `code = 3` liefert Pfeilspitzen an beiden Enden.)
`> text( x, y, labels,` `+ adj, pos)`	Für die `numeric`-Vektoren `x` und `y` und den `character`-Vektor `labels` wird der Text `labels[i]` (per Voreinstellung horizontal und vertikal zentriert) an die Stelle (`x[i], y[i]`) in ein *bestehendes* Kosy geschrieben. Mit `adj` oder `pos` lässt sich die Ausrichtung der Textes relativ zur Stelle (`x[i], y[i]`) variieren. Voreinstellung für `labels` ist `1:length( x)`, d. h., an den Punkt (`x[i], y[i]`) wird `i` geschrieben.
`> grid( nx, ny)`	Zeichnet ein (`nx` × `ny`)-Koordinatengitter in einen bestehenden Plot ein, das sich an seiner Achseneinteilung orientiert.

Bemerkung: Alle obigen Funktionen habe zahlreiche weitere Argumente. Eine große Auswahl davon wird im folgenden Abschnitt beschrieben, aber nicht alle; speziell interessant für `points()` ist `pch` und für `lines()` sind es `lty` und `col` auf Seite 103. Die Online-Hilfe der einzelnen Funktionen ist – wie immer – eine wertvolle Informationsquelle.

7.3 Die Layoutfunktion `par()` und Grafikparameter für `plot()`, `par()` et al.

Viele Grafikfunktionen, die einen Plot erzeugen, besitzen optionale Argumente, die bereits beim Funktionsaufruf die Spezifizierung einiger Layoutelemente (wie Überschrift, Untertitel, Legende) erlauben. Darüber hinaus stehen bei Plotfunktionen, die ein Koordinatensystem erzeugen, Argumente zur Verfügung, mit denen man dessen Layout spezifisch zu steuern vermag (Achsentypen, -beschriftung und -skalen, Plottyp). Weitere Grafikparameter werden durch die Funktion `par()` gesetzt und steuern grundlegende Charakteristika aller nachfolgenden Plots. Sie behalten so lange ihre (neue) Einstellung bis sie explizit mit `par()` wieder geändert oder beim Aufruf eines neuen Grafik-Devices (Grafikfenster oder -datei) automatisch für diese Grafikausgabe auf ihre Voreinstellung zurückgesetzt werden. Zusätzlich gibt es einige „eigenständige“ Layoutfunktionen (wie `title()`, `mtext()`, `legend()`), die zu einem bestehenden Plot (bereits erwähnte) Layoutelemente auch nachträglich hinzuzufügen erlauben. Die folgende Auflistung ist *nicht* vollständig!

Seitenlayoutparameter der Funktion `par()`	
`mfrow = c( m, n)`	`mfrow = c( m,n)` teilt eine Grafikseite in $m \cdot n$ Plotrahmen ein, die in m Zeilen und n Spalten angeordnet sind. Dieser Mehrfachplotrahmen wird links oben beginnend *zeilenweise* mit Plots gefüllt.
`mfcol = c( m, n)`	Wie `mfrow`, nur wird der Mehrfachplotrahmen *spaltenweise* gefüllt.
`oma = c( u, l, o, r)`	Spezifiziert die maximale Textzeilenzahl für den unteren (`u`), linken (`l`), oberen (`o`) und rechten (`r`) „äußeren“ Seitenrand eines Mehrfachplotrahmens. Voreinstellung: `oma = c( 0,0,0,0)`. Für die Beschriftung dieser Ränder siehe `mtext()` und `title()` in Abschnitt 7.4.
`mar = c( u, l, o, r)`	Bestimmt die Breite der Seitenränder (unten, links, oben und rechts) eines einzelnen Plotrahmens in Zeilen ausgehend von der Box um den Plot. Voreinstellung: `mar = c( 5,4,4,2)+0.1`, was etwas „verschwenderisch“ ist.
`pty = "c"`	Setzt den aktuellen Plotrahmentyp fest. Mögliche Werte für `c` sind `s` für einen quadratischen Plotrahmen und `m` für einen maximalen, typischerweise rechteckigen Plotrahmen (Voreinstellung).

Hinweis: Flexiblere und komplexere Einteilungen von Grafik-Devices erlaubt u. a. die Funktion `layout()`, die jedoch inkompatibel mit vielen anderen Layoutfunktionen ist; siehe ihre Online-Hilfe und die dortigen Beispiele.

Koordinatensystemparameter der Funktionen `plot()` oder `par()`	
`axes = L`	Logischer Wert; `axes = FALSE` unterdrückt jegliche Achsenkonstruktion und -beschriftung. Voreinstellung: `TRUE`. (Siehe auch `axis()` in Abschnitt 7.4.)
`xlim = c( x1, x2)` `ylim = c( y1, y2)`	Erlaubt die Wahl der Achsenskalen: `x1, x2, y1, y2` sind approximative Minima und Maxima der Skalen, die für eine „schöne“ Achseneinteilung automatisch gerundet werden.
`log = "c"`	Definiert eine logarithmische Achsenskaleneinteilung: `log = "xy"`: doppelt-logarithmisches Achsensystem; `log = "x"`: logarithmierte x- und „normale“ y-Achse; `log = "y"`: logarithmierte y- und „normale“ x-Achse.
`lab = c( x, y, len)`	Kontrolliert die Anzahl der Achseneinteilungsstriche (“ticks”) und ihre Länge (jeweils approximativ): `x`: Anzahl der tick-Intervalle der x-Achse; `y`: Anzahl der tick-Intervalle der y-Achse; (`len`: tick-Beschriftungsgröße beider Achsen. *Funktionslos!*) Voreinstellung: `lab = c( 5,5,7)`.

tcl = x	Länge der ticks als Bruchteil der Höhe einer Textzeile. Ist x negativ (positiv), werden die ticks außerhalb (innerhalb) des Kosys gezeichnet; Voreinstellung: -0.5.
las = n	Orientierung der Achsenskalenbeschriftung: las = 0: parallel zur jeweiligen Achse (Voreinstellung); las = 1: waagrecht; las = 2: senkrecht zur jeweiligen Achse; las = 3: senkrecht.
mgp = c(x1, x2, x3)	Abstand der Achsenbeschriftung (x1), der -skalenbeschriftung (x2) und der -linie (x3) vom Zeichenbereich in relativen Einheiten der Schriftgröße im Plotrand. Voreinstellung: mgp = c(3,1,0). Größere Werte führen zur Positionierung weiter vom Zeichenbereich entfernt, negative zu einer innerhalb des Zeichenbereichs.

Linien-, Symbol-, Textparameter für par(), plot() & Co.	
type = "c"	Spezifiziert als plot()-Argument den Plottyp. Mögliche Werte für c sind p: Punkte allein; l: Linien allein; b: beides, wobei die Linien die Punkte aussparen; o: beides, aber überlagert; h: vertikale "high-density"-Linien; s, S: Stufenplots (zwei Arten Treppenfunktionen); n: „nichts“ außer ein leeres Koordinatensystem.
pch = "c" pch = n	Verwendetes Zeichen c bzw. verwendeter Zeichencode n für zu plottende Punkte. Voreinstellung: o. Andere Symbole können der Online-Hilfe zu points() entnommen werden (am besten durch example(points)).
lty = n lty = "text"	Linientyp (geräteabhängig) als integer-Code n oder als character-Wert. Normalerweise ergibt lty = 1 (Voreinstellung) eine durchgezogene Linie und für höhere Werte erhält man gestrichelte und/oder gepunktete Linien. Für text sind englische Ausdrücke zugelassen, die den Linientyp beschreiben wie z. B. solid oder dotted. Details: Online-Hilfe zu par().
col = n col = "text" col.axis col.lab col.main col.sub	Linienfarbe als integer-Code oder character: col = 1 ist die Voreinstellung (i. d. R. schwarz); 0 ist stets die Hintergrundfarbe. Hier sind englische Farbnamen zugelassen, z. B. red oder blue. Die Parameter col.axis, col.lab, col.main, col.sub beziehen sich speziell auf die Achsenskalenbeschriftung, die Achsenbeschriftung, die Überschrift bzw. den Untertitel, wie durch title() erzeugt werden (siehe Abschnitt 7.4).
cex = x cex.axis cex.lab cex.main cex.sub	Zeichen-„Expansion“ relativ zur Standardschriftgröße des verwendeten Gerätes: Falls x > 1, ein Vergrößerungs-, falls x < 1, ein Verkleinerungsfaktor. Für cex.axis, cex.lab, cex.main und cex.sub gilt dasselbe wie für col.axis usw. Details zu col und cex: Online-Hilfe zu par().
adj = x	Ausrichtung von (z. B. durch text(), title() oder mtext(), siehe Abschnitt 7.4) geplottetem Text: adj = 0: linksbündig, adj = 0.5: horizontal zentriert (Voreinstellung), adj = 1: rechtsbündig. Zwischenwerte führen zur entsprechenden Ausrichtung zwischen den Extremen.

Beschriftungsparameter der Funktion plot()	
main = "text"	Plotüberschrift text; 1.5-fach gegenüber der aktuellen Standardschriftgröße vergrößert. Voreinstellung: NULL.
main = "text1\ntext2"	\n erzwingt einen Zeilenumbruch und erzeugt (hier) eine zweizeilige Plotüberschrift. (Analog für das Folgende.)
sub = "text"	Untertitel text, unterhalb der x-Achse. Voreinstellung: NULL.

`xlab = "text"` `ylab = "text"`	x- bzw. y-Achsenbeschriftung (`text`). Voreinstellung ist jeweils der Name der im Aufruf von `plot()` verwendeten Variablen.

7.4 Achsen, Überschriften, Untertitel und Legenden

Wurde eine Grafik angefertigt, ohne schon beim Aufruf von `plot()` Achsen, eine Überschrift oder einen Untertitel zu erstellen (z. B. weil `plot()`'s Argumenteliste `axes = FALSE, ylab = "", xlab = ""` enthielt und weder `main` noch `sub` mit einem Wert versehen wurden), kann dies mit den folgenden Funktionen nachgeholt und bedarfsweise auch noch genauer gesteuert werden, als es via `plot()` möglich ist. Darüberhinaus lässt sich eine Grafik um eine Legende ergänzen.

„Eigenständige“ Plotbeschriftungsfunktionen	

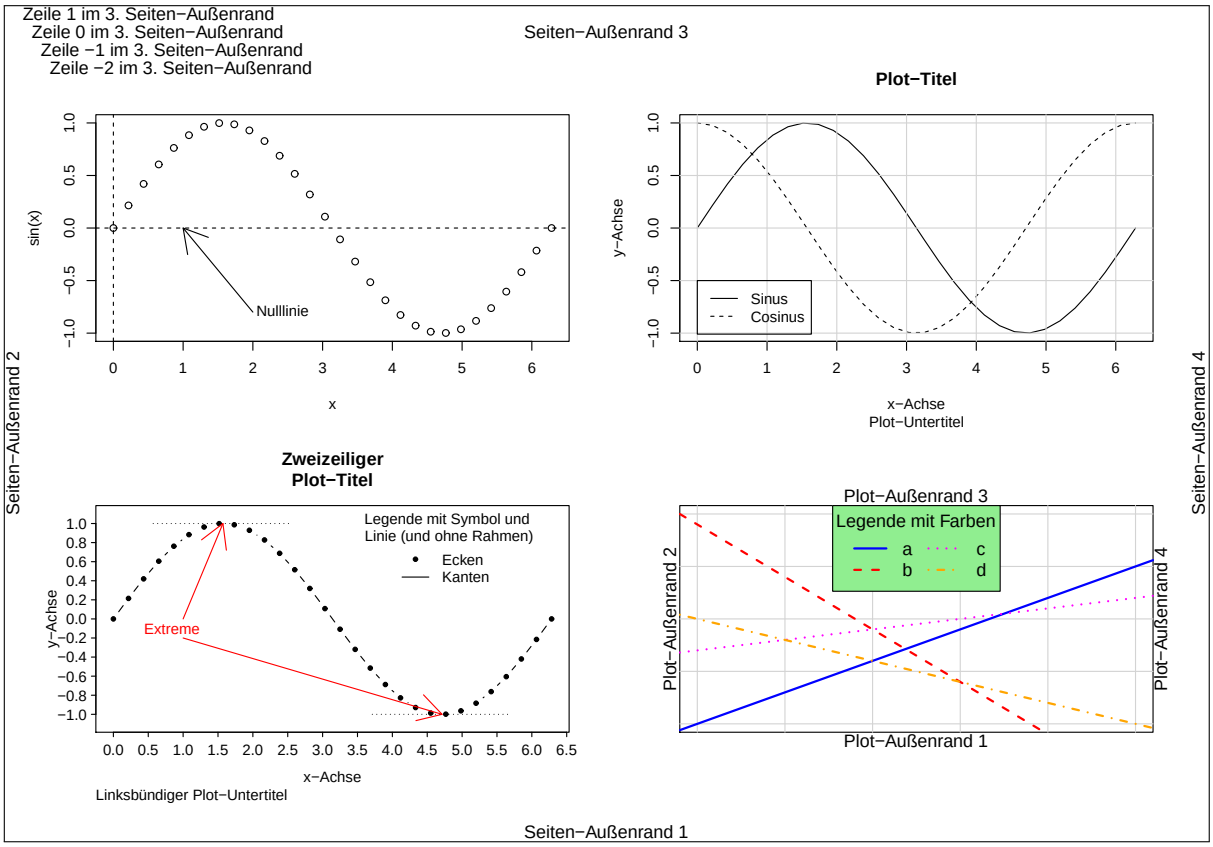
`> axis( side,`
`+ at = NULL,`
`+ labels = TRUE,`
`+ tick = TRUE,`
`+ ....)`
	Fügt zu einem bestehenden Plot eine Achse an der durch `side` spezifizierten Seite (1 = unten, 2 = links, 3 = oben, 4 = rechts) hinzu, die ticks an den durch `at` angebbaren Stellen hat (`Inf`, `NaN`, `NA` werden dabei ignoriert); in der Voreinstellung `at = NULL` werden die tick-Positionen intern berechnet. `labels = TRUE` (Voreinstellung) liefert die übliche numerische Beschriftung der ticks (auch ohne eine Angabe für `at`). Für `labels` kann aber auch ein `character`- oder `expression`-Vektor mit an den ticks zu platzierenden Labels angegeben werden; dann muss jedoch `at` mit einem Vektor derselben Länge versorgt sein. `tick` gibt an, ob überhaupt ticks und eine Achsenlinie gezeichnet werden; in der Voreinstellung geschieht dies.

`> title( main,`
`+ sub,`
`+ xlab,`
`+ ylab,`
`+ line,`
`+ outer = FALSE)`
	Fügt zu einem bestehenden Plot (zentriert in den Rändern des aktuellen Plotrahmens) die Überschrift `main` (1.5-mal größer als die aktuelle Standardschriftgröße) und den Untertitel `sub` unterhalb des Kosys sowie die Achsenbeschriftungen `xlab` und `ylab` hinzu. Der voreingestellte Zeilenabstand ist durch `line` änderbar, wobei positive Werte den Text weiter vom Plot entfernt positionieren und negative Werte näher am Plot. Falls `outer = TRUE`, landen die Texte in den Seiten-Außenrändern.

`> mtext( text,`
`+ side = 3,`
`+ line = 0,`
`+ outer = FALSE)`
	Schreibt den Text `text` in einen der Seitenränder des aktuellen Plotrahmens. Dies geschieht in den Seitenrand `side`: 1 = unten, 2 = links, 3 = oben (Voreinstellung), 4 = rechts. Dort wird in die Zeile `line` (Bedeutung wie bei `title()`) geschrieben. Soll der Text in den (äußeren) Rand eines *Mehrfach*plotrahmens geplottet werden (also beispielsweise als Überschrift für einen Mehrfachplotrahmen), muss `outer = TRUE` angegeben werden.

`> legend( x = 0,`
`+ y = NULL,`
`+ legend,`
`+ lty, pch,`
`+ ncol = 1)`
 	
`> legend( "top",`
`+ legend, ....)`
	In den bestehenden Plot wird eine Legende eingezeichnet, deren linke obere Ecke sich an einer durch `x` und `y` definierten Position befindet und worin die Elemente des `character`-Vektors `legend` auftauchen, gegebenenfalls kombiniert mit den durch `lty` und/oder `pch` spezifizierten Linientypen bzw. Plotsymbolen. (Die nächste Seite und der nächste Abschnitt zeigen Beispiele.) Mit `ncol` wird die (auf 1 voreingestellte) Spaltenzahl variiert. Weitere grafische Parameter sind angebbar; siehe die Online-Hilfe. Anstelle numerischer Werte für `x` und `y` kann an `x` eines der Worte `"bottomright"`, `"bottom"`, `"bottomleft"`, `"left"`, `"topleft"`, `"top"`, `"topright"`, `"right"` oder `"center"` als Ort der Legende (im Inneren des Kosys) übergeben werden.

Die Grafik auf der nächsten Seite oben zeigt das Beispiel einer Seite mit einem (2×2)-Mehrfachplotrahmen und verschiedensten Layouteinstellungen, die durch die darunter folgenden **R**-Kommandos erzeugt wurde.




```
# 30 äquidistante Stützstellen von 0 bis 2*pi als Beispieldaten und
# Formatieren eines Grafikfensters als (2 x 2)-Mehrfachplotrahmen:
#*****************************************************************
x <- seq( 0, 2*pi, length = 30)
par( mfrow = c( 2, 2), oma = c( 1, 1, 2, 1), mar = c( 5, 4, 4, 2) + 0.1)

# Plot links oben:
#*****************
plot( x, sin( x));        abline( v = 0, lty = 2);     abline( h = 0, lty = "dashed")
arrows( 2, -0.8, 1, 0);   text( 2, -0.8, "Nulllinie", pos = 4, offset = 0.2)

# Plot rechts oben:
#******************
plot( x, sin( x), type = "l", main = "Plot-Titel", sub = "Plot-Untertitel",
      xlab = "x-Achse", ylab = "y-Achse")
lines( x, cos( x), lty = "dashed");     grid( lty = "solid")
legend( 0, -0.5, legend = c( "Sinus", "Cosinus"), lty = c( "solid", "dashed"))

# Plot links unten:
#******************
par( mgp = c( 2, 0.5, 0))
plot( x, sin( x), type = "b", pch = 20, las = 1, lab = c( 10, 10, 7), tcl = -0.25,
      ylim = c( -1.1, 1.1), xlab = "x-Achse", ylab = "y-Achse")

text( 1, -0.1, "Extreme", adj = 0.7, col = "red")
arrows( c( 1, 1), c( 0, -0.2), c( pi/2, 3*pi/2), c( 1, -1), col = "red")
segments( c( pi/2, 3*pi/2) - 1, c( 1, -1),
          c( pi/2, 3*pi/2) + 1, c( 1, -1), lty = "dotted")
legend( 3.5, 1, legend = c( "Ecken", "Kanten"), lty = c( -1, 1), pch = c( 20, -1),
        bty = "n", title = "Legende mit Symbol und\nLinie (und ohne Rahmen)")

title( main = "Zweizeiliger\nPlot-Titel", cex = 0.75)
title( sub = "Linksbündiger Plot-Untertitel", adj = 0, cex = 0.6)
```

```
# Plot rechts unten:
#******************
par( mgp = c( 3, 1, 0))
plot( c( 0, 5), c( -1, 1), type = "n", axes = FALSE, xlab = "", ylab = "");    box()
grid( lty = "solid")

abline( -1,   0.3, lty = 1, lwd = 2, col = "blue")
abline( 0.9, -1/2, lty = 2, lwd = 2, col = "red")
abline( -0.3, 0.1, lty = 3, lwd = 2, col = "magenta")
abline( 0,   -0.2, lty = 4, lwd = 2, col = "orange")

legend( "top", legend = letters[ 1:4], lty = 1:4, lwd = 2,
        col = c( "blue", "red", "magenta", "orange"), cex = 1.2, ncol = 2,
        bg = "lightgreen", title = "Legende mit Farben")

mtext( "Plot-Außenrand 1", side = 1);   mtext( "Plot-Außenrand 2", side = 2)
mtext( "Plot-Außenrand 3", side = 3);   mtext( "Plot-Außenrand 4", side = 4)

# Text im äußeren Rahmen der Seite:
#**********************************
mtext( paste("Seiten-Außenrand", 1:4), side = 1:4, outer = TRUE)    # mtext() mit
mtext( paste( "Zeile", 1:-2, "im 3. Seiten-Außenrand"),             # vektoriellen
       outer = TRUE, line = 1:-2, adj = 0:3/100)                    # Argumenten.

# Schließen der Grafikausgabe:
#*****************************
dev.off()
```

7.5 Einige (auch mathematisch) nützliche Plotfunktionen

In mathematisch-statistischen Anwendungen sind mit gewisser Häufigkeit die Graphen stetiger Funktionen, einseitig stetiger Treppenfunktionen oder geschlossener Polygonzüge (mit gefärbtem Inneren) zu zeichnen. Hierfür präsentieren wir einige **R**-Plotfunktionen im Rahmen von Anwendungsbeispielen (und verweisen für Details auf die Online-Hilfe):

7.5.1 Stetige Funktionen: `curve()`

```
> curve( sin( x)/x, from = -20, to = 20, n = 500)
> curve( abs( 1/x), add = TRUE, col = "red", lty = 2)
```

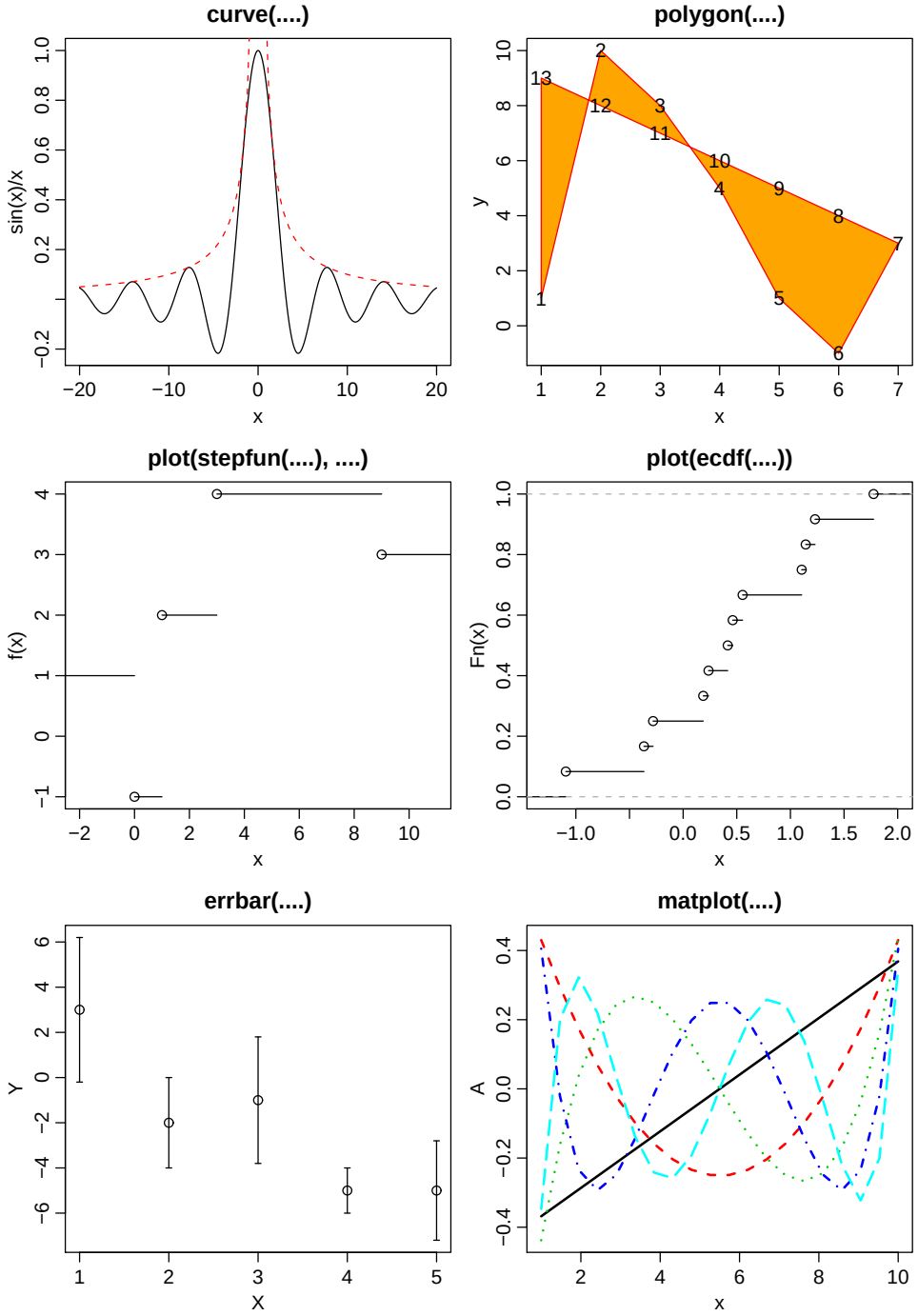
(Plot links oben auf nächster Seite.) Das erste Argument von `curve()` ist der Ausdruck eines Aufrufs einer (eingebauten oder selbstdefinierten) Funktion. Die Argumente `from` und `to` spezifizieren den Bereich, in dem die Funktion ausgewertet und gezeichnet wird. `n` (Voreinstellung: 101) gibt an, wie viele äquidistante Stützstellen dabei zu nutzen sind. Ist `add = TRUE`, wird in ein *bestehendes* Koordinatensystem gezeichnet, wobei dessen x-Achsenausschnitt verwendet wird (und dies, wenn nicht anders spezifiziert, mit `n = 101` Stützstellen). Wie stets, bestimmen `col`, `lty` und `lwd` Linienfarbe, -typ bzw. -dicke.

7.5.2 Geschlossener Polygonzug: `polygon()`

```
> x <- c( 1:7, 6:1);     y <- c( 1, 10, 8, 5, 1, -1, 3:9)
> plot( x, y, type = "n")                          # Generiert das (leere) Kosy.
> polygon( x, y, col = "orange", border = "red")                # Der Polygonzug.
> text( x, y)                                    # Nur zur Beschriftung der Ecken.
```

(Plot rechts oben auf nächster Seite.) `polygon()` zeichnet ein Polygon in ein *bestehendes* Koordinatensystem. Seine ersten beiden Argumente `x` und `y` müssen die Koordinaten der Polygonecken

enthalten, die durch Kanten zu verbinden sind, wobei die letzte Ecke automatisch mit der ersten (`x[1], y[1]`) verbunden wird. Innen- und Randfarbe werden durch `col` bzw. `border` festgelegt.



7.5.3 Beliebige Treppenfunktionen: `plot()` in Verbindung mit `stepfun()`

```
> x <- c( 0, 1, 3, 9);      y <- c( 1, -1, 2, 4, 3)
> plot( stepfun( x, y), verticals = FALSE)
```

(Linker Plot in zweiter „Zeile“ oben auf dieser Seite.) Für einen aufsteigend sortierten Vektor `x` von Sprungstellen und einen *um ein Element längeren* Vektor `y` von Funktionswerten *zwischen* den Sprungstellen generiert `stepfun( x, y)` ein `stepfun`-Objekt, das eine rechtsseitig stetige Treppenfunktion repräsentiert, die links von der ersten Sprungstelle `x[1]` auf dem Niveau `y[1]` startet und an `x[i]` auf `y[i+1]` springt. Sie wird von `plot()` „mathematisch korrekt“ gezeichnet, falls das `plot`-Argument `verticals = FALSE` ist.

7.5.4 Die empirische Verteilungsfunktion: `plot()` in Verbindung mit `ecdf()`

```
> plot( ecdf( rnorm( 12)))     # Empirische Verteilungsfunktion zu 12 standard-
                               # normalverteilten Pseudozufallsvariablen.
```

(Rechter Plot in zweiter „Zeile“ oben auf vorheriger Seite.) Die empirische Verteilungsfunktion F_n zu n Werten ist eine Treppenfunktion, die auf dem Niveau 0 startet und an der i-ten OS auf den Funktionswert i/n springt. Zu einem beliebigen `numeric`-Vektor generiert `ecdf()` ein `stepfun`-Objekt, das diese speziellen Eigenschaften hat und von `plot()` „mathematisch korrekt“ gezeichnet wird, da hierzu automatisch die `plot`-Methode `plot.stepfun()` verwendet wird.

7.5.5 „Fehlerbalken“: `errbar()` im Package `Hmisc`

```
> X <- 1:5;   Y <- c( 3, -2, -1, -5, -5);   delta <- c( 3.2, 2, 2.8, 1, 2.2)
> library( Hmisc)
> errbar( x = X, y = Y, yplus = Y + delta, yminus = Y - delta)
> detach( package:Hmisc)
```

(Plot ganz links unten auf vorheriger Seite.) Mit `library(Hmisc)` wird das Package `Hmisc` in den Suchpfad eingefügt, damit die Funktion `errbar()` zur Verfügung steht. (Vorsicht! Hierdurch wird – temporär – die Funktion `ecdf()` durch eine `Hmisc`-spezifische maskiert, wie eine entsprechende Mitteilung meldet. Das abschließende `detach(package:Hmisc)` revidiert diese Maskierung.) `errbar()` plottet an den für `x` angegebenen Stellen vertikale „Fehlerbalken“. Die Werte für `y` (hier also die Werte in `Y`) werden markiert und die (absoluten) Unter- und Obergrenzen der vertikalen Balken werden von `yminus` bzw. `yplus` erwartet.

7.5.6 Mehrere Polygonzüge „auf einmal“: `matplot()`

`matplot( A, B)` trägt jede Spalte einer n-zeiligen Matrix `B` gegen jede Spalte einer anderen, ebenfalls n-zeiligen Matrix `A` als Polygonzug in ein gemeinsames Koordinatensystem ab, wobei jede Spaltenkombination einen eigenen Linientyp erhält. Bei ungleicher Spaltenzahl wird (spalten-)zyklisch repliziert; also kann jede der Matrizen auch ein n-elementiger Vektor sein.

Beispiel: `B` sei eine $(n \times k)$-Matrix, die in ihren $j = 1, \ldots, k$ Spalten die Funktionswerte $p_j(x_i)$ gewisser Orthonormalpolynome $p_1, \ldots, p_k$ an Stützstellen x_i für $i = 1, \ldots, n$ enthält:

```
> x <- seq( 1, 10, length = 20)
> (B <- poly( x, 5))     # Gewisse Orthonormalpolynome bis zum Grad 5
               1          2           3           4           5
 [1,] -0.36839420  0.43019174 -0.43760939  0.40514757 -0.34694765
 [2,] -0.32961586  0.29434172 -0.16122451 -0.02132356  0.20086443
 [3,] -0.29083753  0.17358614  0.03838679 -0.23455912  0.32260045
....
[20,]  0.36839420  0.43019174  0.43760939  0.40514757  0.34694765
```

Durch

```
> matplot( x, B, type = "l")
```

wird eine Grafik erzeugt, in der für jede Spalte $j = 1, \ldots, k$ ein (farbiger) Polygonzug durch die Punkte (`x[ i]`, `B[ i, j]`), $i = 1, \ldots, n$, gezeichnet ist, wobei die Polygonzüge automatisch verschiedene Linientypen bekommen. (Plot auf der vorherigen Seite ganz unten rechts.)

7.6 Interaktion mit Plots

R erlaubt dem/der BenutzerIn unter Verwendung der Maus mit einem in einem Grafikfenster bestehenden Plot zu interagieren: Durch „Anklicken“ mit der Maus können geplottete Punkte in einem Koordinatensystem (Kosy) identifiziert oder Informationen an beliebigen, „angeklickten“ Positionen in einen Plot eingefügt werden.

Interaktion mit Plots	
`> identify( x, y)` `[1]  4  7 13 19 24`	In den in einem Grafikfenster *bestehenden* Plot von `y` gegen `x` wird nach jedem Klick mit der linken Maustaste die *Indexnummer* des dem Mauszeiger nächstliegenden Punktes in dessen Nähe in das bestehende Kosy platziert. Ein Klick der mittleren Maustaste beendet die Funktion `identify()`. Rückgabewert: Vektor der Indizes der „angeklickten“ Punkte (hier 5 Stück).

`> identify( x, y,`
`+ labels = c( "a",`
`+ "b", ....))`
	Wird `labels` ein `character`-Vektor (derselben Länge wie `x`) zugewiesen, so wird das dem „angeklickten“ Punkt entsprechende Element von `labels` in dessen Nähe gezeichnet.

`> identify( x, y,` `+ plot = FALSE)`	Für `plot = FALSE` entfällt das Zeichnen. (`identify()` gibt aber stets die Indices der angeklickten Punkte zurück.)

`> locator()`
`$x`
`[1] 0.349  4.541`
 	
`$y`
`[1] -0.291 0.630`
	Für einen in einem Grafikfenster bestehenden Plot wird eine Liste von x-y-Koordinaten der mit der linken Maustaste „angeklickten“ Stellen geliefert (hier 2 Stück; Voreinstellung: Maximal 500). Mit einem Klick der mittleren Maustaste wird die Funktion beendet. (Kann z. B. zur Positionierung einer Legende genutzt werden; siehe unten.)

`> locator( n = 10,` `+ type = "c")`	Das Argument `n` bestimmt die Höchstzahl der identifizierbaren Punkte. `type` gibt an, ob bzw. was an den „angeklickten“ Koordinaten geplottet werden soll. Mögliche Werte für `c` sind `p`: Punkte allein; `l`: Die Punkte verbindende Linien (also ein Polygonzug); `o`: Punkte mit verbindenden Linien überlagert; `n`: „Nichts“ (Voreinstellung).

Anwendungsbeispiele für die Interaktion mit Plots	

`> pos <- locator( 1)`
`> legend( pos,`
`+ legend, ....)`
	`pos` speichert die in einem bestehenden Plot in einem Grafikfenster angeklickte Stelle. Dann wird dort die durch `legend()` spezifizierte Legende positioniert.

`> text( locator(),` `+ labels)`	An den durch `locator()` in einem bestehenden Plot in einem Grafikfenster spezifizierten Positionen werden die in `labels` angegebenen Elemente (zyklisch durchlaufend) gezeichnet. Allerdings geschieht das Zeichnen erst, *nachdem* durch einen Klick der mittleren Maustaste die Funktion `locator()` beendet worden ist.

`> mies <- identify( x,`
`+ y, plot = FALSE)`
`> xgut <- x[ -mies]`
`> ygut <- y[ -mies]`
	Die Indexnummern der unter den (`x[i], y[i]`) durch Anklicken mit der Maus identifizierten Punkte werden in `mies` gespeichert, damit diese „miesen“ Punkte danach aus `x` und `y` entfernt werden können.

Hinweise:

- Wörtliche Wiederholung der Bemerkungen von Seite 86 unten: Ein sehr empfehlenswertes sowie weit- und tiefgehendes Buch zum Thema Grafik in **R** ist "R Graphics" ([64, Murrell (2005)]) (bzw. die zweite Auflage [65, Murrell (2011)]). Dasselbe gilt für das exzellente Buch "Lattice. Multivariate Data Visualization with R" ([69, Sarkar (2008)]), das die **R**-Implementation und -Erweiterung des hervorragenden "Trellis-Grafik"-Systems vorstellt, auf das wir hier nicht eingehen.

- Die "**R**-Gallery" (`http://gallery.r-enthusiasts.com/`) wurde bereits in Abschnitt 1.1 erwähnt. In ihr sind zahlreiche, mit **R** angefertigte, exzellente Grafiken (samt des sie produzierenden **R**-Codes) zu finden und sie können dort nach verschiedenen Kriterien sortierbar betrachtet werden.

- **R** stellt auch eine LaTeX-ähnliche Funktionalität für die Erstellung mathematischer Beschriftung von Grafiken zur Verfügung. Die Online-Hilfe dazu erhält man via `?plotmath`. `example( plotmath)` und `demo( plotmath)` liefern einen Überblick über das, was damit möglich ist, und wie es erreicht werden kann. Warnung: Die Syntax ist etwas „gewöhnungsbedürftig".

- Für erheblich leistungsfähigere interaktive und dynamische Grafik stehen z. B. die **R**-Packages `rgl` und `rggobi` zur Verfügung. Sie besitzen auch eigene Websites, zu denen man z. B. aus der Package-Sammlung von **R** kommt: `www://cran.r-project.org/` → "Packages" → `rgl` oder `rggobi` und dort dann der angegebenen URL folgen.

8 Zur para- und nicht-parametrischen Inferenzstatistik in Ein- und Zweistichprobenproblemen für metrische Daten

8.1 Auffrischung des Konzepts statistischer Tests

Die Inferenzstatistik dient ganz allgemein dem Ziel, Aussagen über unbekannte Parameter einer Grundgesamtheit (= Population) auf der Basis von Daten, d. h. einer Stichprobe zu machen. Hin und wieder liegen über die Werte von Parametern Vermutungen oder Annahmen vor; man spricht von „Hypothesen“. Über die Richtigkeit solcher Hypothesen entscheiden zu helfen ist der Zweck statistischer Tests. Die statistische Testtheorie stellt eine Verbindung zwischen Stichprobe und Population, also zwischen dem Experiment als kleinem Ausschnitt aus der Wirklichkeit und der „Gesamtwirklichkeit“ her: Sie beschreibt, wie sich Stichproben wahrscheinlichkeitstheoretisch „verhalten“, wenn eine gewisse Hypothese über die jeweilige Population zutrifft, und liefert Kriterien, die zu entscheiden erlauben, ob die Hypothese angesichts der tatsächlich beobachteten Stichprobe beizubehalten oder zu verwerfen ist.

8.1.1 Motivation anhand eines Beispiels

Aufgrund langjähriger Erfahrung sei bekannt, dass die Körpergewichte (in Gramm) von ausgewachsenen, männlichen, weißen Labormäusen (vgl. z. B. `de.wikipedia.org/wiki/Hausmaus` → „Äußere Merkmale“) einer gewissen Zuchtlinie unter natürlichen Bedingungen normalverteilt sind mit dem Erwartungswert $\mu_0 = 49$ g und der Standardabweichung $\sigma_0 = 4.2$ g. Für das Körpergewicht X einer zufällig ausgewählten solchen Maus gilt also: $X \sim \mathcal{N}(\mu_0, \sigma_0^2)$.

Es wird vermutet, dass ein gewisser Umweltreiz das Körpergewicht beeinflusst. Allerdings sei bekannt, dass auch unter dem Einfluss dieses Umweltreizes das Körpergewicht eine normalverteilte Zufallsvariable – sagen wir Y – mit Varianz σ_0^2 ist. Jedoch ist ihr Erwartungswert μ unbekannt und möglicherweise verschieden von μ_0. Kurz: $Y \sim \mathcal{N}(\mu, \sigma_0^2)$.

Es soll geklärt werden, ob $\mu = \mu_0$ ist oder nicht. Zu diesem Zweck werde in einem Experiment eine Gruppe von n zufällig ausgewählten Mäusen ähnlicher physischer Konstitution dem besagten Umweltreiz unter sonst gleichen Bedingungen ausgesetzt. (Die Mäuse seien nicht aus demselben Wurf, um auszuschließen, dass „aus Versehen“ z. B. eine außergewöhnlich schwache oder starke Linie untersucht wird.) Die nach einer vorher festgelegten Zeitspanne beobachtbaren zufälligen Körpergewichte $Y_1, \ldots, Y_n$ der Mäuse sind also $\mathcal{N}(\mu, \sigma_0^2)$-verteilt mit einem unbekannten μ. Aber gemäß der Erwartungstreue des arithmetischen Mittels gilt $\mathbb{E}[\bar{Y}_n] = \mu$, sodass für konkrete Realisierungen $y_1, \ldots, y_n$ der zufälligen Körpergewichte erwartet werden kann, dass $\bar{y}_n \approx \mu$ ist (egal welchen Wert μ hat). Für den Fall, dass der Umweltreiz *keinen* Einfluss auf das mittlere Körpergewicht hat, ist $\mu = \mu_0$ und demzufolge auch $\bar{y}_n \approx \mu_0$.

Es werden nun $n = 50$ Realisierungen $y_1, \ldots, y_n$ mit einem Mittelwert $\bar{y}_n = 47.46$ ermittelt und es stellt sich die Frage, ob der beobachtete Unterschied zwischen $\bar{y}_n = 47.46$ und $\mu_0 = 49$, also die Abweichung $|\bar{y}_n - \mu_0| = 1.54$

- eine im Rahmen des Szenarios $Y_1, \ldots, Y_n \sim \mathcal{N}(\mu_0, \sigma_0^2)$ gewöhnlich große, auf reinen *Zufalls*schwankungen beruhende Abweichung ist, oder ob sie
- so groß ist, dass sie eine ungewöhnliche Abweichung darstellt, also gegen $\mu = \mu_0$ spricht und vielmehr das Szenario $\mu \neq \mu_0$ stützt.

8.1.2 Null- & Alternativhypothese, Fehler 1. & 2. Art

Zur Beantwortung der eben gestellten Frage kann ein <u>statistischer Test</u> verwendet werden. Dazu formulieren wir das Problem mittels zweier (sich gegenseitig ausschließender) <u>Hypothesen</u>:

$$\text{Die } \underline{\text{Nullhypothese}} \;\; H_0 : \mu = \mu_0 \qquad \text{und} \qquad \text{die } \underline{\text{Alternative}} \;\; H_1 : \mu \neq \mu_0$$

stehen für die möglichen Verteilungsszenarien der Y_i in obigem Beispiel. Dabei wird hier durch die Alternative H_1 nicht spezifiziert, in welche Richtung μ von μ_0 abweicht, geschweige denn, welchen Wert μ stattdessen hat.

Auf der Basis einer Stichprobe soll nach einem (noch zu entwickelnden) Verfahren entschieden werden, ob H_0 abzulehnen ist oder nicht. (Der Name Nullhypothese stammt aus dem Englischen: "to nullify" = entkräften, widerlegen.) Dabei müssen wir uns bewusst sein, dass statistische Tests, also Tests auf der Basis zufälliger Daten, nur *statistische* Aussagen über den „Wahrheitsgehalt" von Hypothesen machen können. Testentscheidungen bergen also immer die Gefahr von Fehlern und wir unterscheiden hierbei zwei Arten:

- **Der Fehler 1. Art:**

  Bei einem Test von H_0 gegen H_1 kann es passieren, dass eine „unglückliche" Stichprobenzusammenstellung, die allein durch natürliche Zufallseinflüsse entstand, uns veranlasst, die Nullhypothese *fälschlicherweise zu verwerfen* (d. h., H_0 zu verwerfen, obwohl H_0 in Wirklichkeit richtig ist). Ein solcher Irrtum heißt Fehler 1. Art. Die Testtheorie stellt Mittel zur Verfügung, die Wahrscheinlichkeit für einen solchen Fehler zu kontrollieren – auszuschließen ist er so gut wie nie! Jedoch kann man sich die (Irrtums-)Wahrscheinlichkeit $\alpha \in (0, 1)$ für den Fehler 1. Art beliebig (aber üblicherweise klein) vorgeben. Sie wird das Signifikanzniveau des Tests genannt.

  Häufig verwendete Werte für α sind 0.1, 0.05 und 0.01, aber an und für sich ist α frei wählbar; es quantifiziert die subjektive Bereitschaft, im vorliegenden Sachverhalt eine Fehlentscheidung 1. Art zu treffen. Also sind auch andere als die genannten Niveaus zulässig, wie z. B. 0.025 oder 0.2, aber je kleiner α ist, umso „zuverlässiger" ist eine Ablehnung von H_0, *wenn* es dazu kommt.

- **Der Fehler 2. Art:**

  Er wird gemacht, wenn man die Nullhypothese *fälschlicherweise beibehält* (d. h., H_0 beibehält, obwohl H_0 in Wirklichkeit falsch ist). Die Wahrscheinlichkeit für diesen Fehler 2. Art wird mit β bezeichnet. Im Gegensatz zu α, das man sich vorgibt, ist β *nicht* so ohne Weiteres kontrollierbar, sondern hängt von α, der Alternative, der den Daten zugrunde liegenden Varianz und dem Stichprobenumfang n ab. (Näheres hierzu in Abschnitt 8.12 „Testgüte und Fallzahlschätzung für Lokationsprobleme im Normalverteilungsmodell".)

Wir geben eine tabellarische Übersicht über die vier möglichen Kombinationen von unbekannter Hypothesenwirklichkeit und bekannter Testentscheidung:

		Unbekannte Wirklichkeit	
		H_0 wahr	H_0 falsch
Datenabhängige Testentscheidung	H_0 verwerfen	Fehler 1. Art*)	Richtige Entscheidung
	H_0 beibehalten	Richtige Entscheidung	Fehler 2. Art**)

$*$): Auftrittswahrscheinlichkeit α ist unter Kontrolle.
$**$): Auftrittswahrscheinlichkeit β ist *nicht* unter Kontrolle.

Offenbar werden H_0 und H_1 nicht symmetrisch behandelt. Allerdings kommt in der Praxis dem Fehler 1. Art auch häufig ein größeres Gewicht zu, da H_0 oft eine Modellannahme ist, die sich bisher bewährt hat. Deshalb will man sich durch Vorgabe eines kleinen αs dagegen schützen, H_0 ungerechtfertigterweise zu verwerfen:

- Die Entscheidung „H_0 verwerfen" ist verlässlich, weil die Wahrscheinlichkeit für eine Fehlentscheidung hierbei unter Kontrolle und üblicherweise klein ist. Man sagt, die Entscheidung „H_0 verwerfen" ist *statistisch signifikant* (auf dem (Signifikanz-)Niveau α).

- Die Entscheidung „H_0 beibehalten“ ist *nicht* verlässlich, denn selbst im Fall, dass H_0 falsch ist, besitzt die (Fehl-)Entscheidung H_0 beizubehalten die möglicherweise hohe Wahrscheinlichkeit β. Man sagt, die Entscheidung „H_0 beibehalten“ ist *nicht signifikant* (auf dem (Signifikanz-)Niveau α).

Fazit: Statistische Tests sind *Instrumente zum Widerlegen von Nullhypothesen*, nicht zu deren Bestätigung. Zu bestätigende Aussagen sind als Alternativen zu wählen.

Wie kommt man jedoch überhaupt zu einer Entscheidung gegen oder für H_0?
Das generelle Konstruktionsprinzip statistischer Tests ist, eine geeignete (Test-)Statistik zu finden, deren wahrscheinlichkeitstheoretisches Verhalten, d. h. deren Verteilung im Fall der Gültigkeit von H_0 (kurz: „unter H_0“) bestimmt werden kann, um darauf basierend eine Entscheidungsregel anzugeben, die für konkrete Daten die Beantwortung der Frage „Pro oder contra H_0?“ erlaubt.

Die Bestimmung des wahrscheinlichkeitstheoretischen Verhaltens einer Teststatistik ist (in der Regel) nur im Rahmen eines wahrscheinlichkeitstheoretischen Modells für den dem Gesamtproblem zugrunde liegenden „Datengenerierungsmechanismus“ möglich. Wir betrachten im folgenden Paragraph beispielhaft das Normalverteilungsmodell.

8.1.3 Konstruktion eines Hypothesentests im Normalverteilungsmodell

Wir wollen zunächst einen **zweiseitigen Test** samt Entscheidungsregel **für eine Hypothese über den Erwartungswert im Normalverteilungsmodell** herleiten. Dazu seien $X_1, \ldots, X_n$ unabhängige und $\mathcal{N}(\mu, \sigma_0^2)$-verteilte Zufallsvariablen, wobei μ unbekannt und σ_0^2 bekannt ist. Zu testen sei für ein gegebenes μ_0

$$H_0 : \mu = \mu_0 \quad \text{gegen} \quad H_1 : \mu \neq \mu_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

Wie in §8.1.1 schon motiviert, ist die Verteilung des zufälligen Differenzbetrages $|\bar{X}_n - \mu_0|$ unter H_0 wichtig, um eine Entscheidung gegen oder für H_0 zu fällen, denn er misst den Unterschied zwischen dem beobachteten Mittelwert und dem unter H_0 erwarteten. Aus diesem Grund machen wir nun die folgende ...

Annahme: **Die Nullhypothese $H_0 : \mu = \mu_0$ sei richtig.**
Dann sind $X_1, \ldots, X_n$ unabhängig $\mathcal{N}(\mu_0, \sigma_0^2)$-verteilt und für $\bar{X}_n = \frac{1}{n}\sum_{i=1}^n X_i$ gilt $\bar{X}_n \sim \mathcal{N}(\mu_0, \sigma_0^2/n)$ und somit:

$$\frac{\bar{X}_n - \mu_0}{\sigma_0/\sqrt{n}} \sim \mathcal{N}(0,1) \qquad \text{für alle } n \geq 1$$

Also ist auch die Verteilung von $|\bar{X}_n - \mu_0|$ vollständig bekannt und für jedes $x > 0$ ist

$$\mathbb{P}_{H_0}\left(\Bigg|\underbrace{\frac{\bar{X}_n - \mu_0}{\sigma_0/\sqrt{n}}}_{\sim\mathcal{N}(0,1)}\Bigg| \leq x\right) = \Phi(x) - \Phi(-x) = 2\Phi(x) - 1 \quad \text{(wg. } \Phi\text{s Symmetrie)}$$

Daraus leitet sich sofort auch eine Wahrscheinlichkeitsaussage über das Ereignis ab, dass die betragliche Differenz $|\bar{X}_n - \mu_0|$ einen Wert *über*schreitet, bzw. (aufgrund der Stetigkeit der Normalverteilung) mindestens so groß ist wie dieser Wert:

$$\mathbb{P}_{H_0}\left(\frac{|\bar{X}_n - \mu_0|}{\sigma_0/\sqrt{n}} \geq x\right) = 2(1 - \Phi(x))$$

Unser Ziel ist nun, zu bestimmen, welche Differenzbeträge unter H_0 „ungewöhnlich groß“ sind: Man *setzt fest (!)*, dass dies der Fall ist, wenn sie auf oder oberhalb einer Schwelle liegen,

die unter H_0 eine *kleine* Überschreitungwahrscheinlichkeit besitzt. Geben wir uns eine kleine Überschreitungswahrscheinlichkeit α vor (mit $0 < \alpha \leq 1/2$) und setzen wir für x das $(1-\alpha/2)$-Quantil der Standardnormalverteilung, also $u_{1-\alpha/2} \equiv \Phi^{-1}(1-\alpha/2)$ ein, so folgt

$$\mathbb{P}_{H_0}\left(\frac{|\bar{X}_n - \mu_0|}{\sigma_0/\sqrt{n}} \geq u_{1-\alpha/2}\right) = 2(1-\Phi(u_{1-\alpha/2})) = \alpha$$

Wir haben also hiermit den Bereich ungewöhnlich großer Differenzbeträge $|\bar{X}_n - \mu_0|$ gefunden: Es sind all diejenigen Werte, für welche $|\bar{X}_n - \mu_0|/(\sigma_0/\sqrt{n})$ mindestens so groß wie der <u>kritische Wert</u> $u_{1-\alpha/2}$ ist.

Mit anderen Worten formuliert bedeutet das obige Resultat:

- $\bar{X}_n$ fällt unter $H_0 : \mu = \mu_0$ tendenziell **in** das Intervall $\left(\mu_0 - \frac{u_{1-\alpha/2}\,\sigma_0}{\sqrt{n}},\ \mu_0 + \frac{u_{1-\alpha/2}\,\sigma_0}{\sqrt{n}}\right)$, nämlich mit der großen Wahrscheinlichkeit $1-\alpha$.

Das *Komplement* von $\left(\mu_0 - \frac{u_{1-\alpha/2}\,\sigma_0}{\sqrt{n}},\ \mu_0 + \frac{u_{1-\alpha/2}\,\sigma_0}{\sqrt{n}}\right)$ wird <u>kritischer Bereich</u> genannt und es liegt insgesamt folgendes Fazit nahe:

- Realisiert sich $\bar{X}_n$ als $\bar{x}_n$ in dem kritischen Bereich, wo unter H_0 nur selten, nämlich mit der kleinen Wahrscheinlichkeit α, ein $\bar{x}_n$ zu erwarten ist, so spricht dies gegen $H_0 : \mu = \mu_0$ und für $H_1 : \mu \neq \mu_0$.

Das führt für eine konkrete Stichprobe $x_1, \ldots, x_n$ zu der folgenden

> Entscheidungsregel für den zweiseitigen Test auf Basis des kritischen Wertes:
>
> $$H_0 \text{ ist zum Niveau } \alpha \text{ zu verwerfen} \quad \Longleftrightarrow \quad \frac{|\bar{x}_n - \mu_0|}{\sigma_0/\sqrt{n}} \geq u_{1-\alpha/2}$$

Dieses Verfahren heißt <u>zweiseitiger Gaußtest</u>; zweiseitig, weil die Alternative $H_1 : \mu \neq \mu_0$ für μ eine Abweichung von μ_0 in *zwei* Richtungen zulässt. Die Größe $|\bar{X}_n - \mu_0|/(\sigma_0/\sqrt{n})$ ist die <u>Teststatistik</u> und $u_{1-\alpha/2}$ ist, wie oben schon gesagt, ihr <u>kritischer Wert</u>. Die Entscheidungsregel garantiert, dass der Fehler 1. Art eine Auftrittswahrscheinlichkeit von (höchstens) α und somit der Test das Signifikanzniveau α hat.

Das **Beispiel** der Körpergewichte von Labormäusen (von Seite 111): Angenommen, das (vorab festgelegte) Signifikanzniveau α des Tests ist 0.05. Dann lautet die Entscheidungsregel wegen $n = 50$, $\sigma_0 = 4.2$ und $u_{1-\alpha/2} = u_{0.975} \doteq 1.96$ für die konkrete Stichprobe $y_1, \ldots, y_n$: „Verwirf H_0, falls $|\bar{y}_n - \mu_0|/(4.2/\sqrt{50}) \geq 1.96$“. Wir hatten $|\bar{y}_n - \mu_0| = 1.54$ und somit $1.54/(4.2/\sqrt{50}) \doteq 2.593$, sodass H_0 hier in der Tat verworfen werden kann.

Im Fall des **einseitigen Gaußtests**

$$H_0' : \mu \overset{(\leq)}{\geq} \mu_0 \qquad \text{gegen} \qquad H_1' : \mu \overset{(>)}{<} \mu_0 \qquad \text{zum Signifikanzniveau } \alpha$$

erfolgt die obige Argumentation analog und führt schließlich zur

> Entscheidungsregel für den einseitigen Test auf Basis des kritischen Wertes:
>
> $$H_0' \text{ ist zum Niveau } \alpha \text{ zu verwerfen} \quad \Longleftrightarrow \quad \frac{\bar{x}_n - \mu_0}{\sigma_0/\sqrt{n}} \overset{(\geq +)}{\leq -} u_{1-\alpha}$$

Beachte, dass hier das $(1-\alpha)$- und nicht das $(1-\alpha/2)$-Quantil der Standardnormalverteilung zum Einsatz kommt!

8.1.4 Der p-Wert

Gleich nochmal zum **Beispiel** der Körpergewichte von Labormäusen: Offenbar und auf gewünschte Weise beeinflusst das Niveau α die Testentscheidung. Wir wollen die Wirkung von α in diesem Zahlenbeispiel mit $n = 50$ und $\sigma_0 = 4.2$ etwas näher untersuchen: Für verschiedene Werte von α erhalten wir verschiedene kritische Werte und abhängig davon entsprechende Testentscheidungen:

α	$u_{1-\alpha/2}$	Testentscheidung
0.1	1.6448	H_0 verwerfen!
0.05	1.9600	H_0 verwerfen!
0.01	2.5758	H_0 verwerfen!
0.005	2.8070	H_0 beibehalten!
0.001	3.2905	H_0 beibehalten!

Generell wächst der kritische Wert $u_{1-\alpha/2}$ monoton, wenn α kleiner wird (hier sogar streng monoton als eine direkte Konsequenz aus dem streng monotonen Wachstum der Quantilfunktion der Standardnormalverteilung). Dieser Sachverhalt gilt nicht nur für die kritischen Werte in zweiseitigen Gaußtests, sondern allgemein für jeden Test:

> *Bei vorliegenden Daten* gibt es ein kleinstes Niveau, auf dem H_0 gerade noch verworfen werden kann. Dieses kleinste Niveau heißt <u>p-Wert</u>.

Demnach ist die Beziehung zwischen dem p-Wert und dem von der Anwenderin bzw. dem Anwender (**vor** der Testdurchführung) fest gewählten Niveau α wie folgt:

> Test-Entscheidungsregel auf Basis des p-Wertes:
>
> H_0 ist zum Niveau α zu verwerfen $\iff$ p-Wert $\leq \alpha$

Software-Pakete berechnen in der Regel den p-Wert zu den eingegebenen Daten und überlassen somit die letztendliche Testentscheidung ihrem/r Anwender/in.

Wir wollen die **Berechnung des p-Wertes beim zweiseitigen Gaußtest** aus §8.1.3 exemplarisch durchführen. Laut obiger Definition des p-Wertes gilt:

$$\text{Der (zweiseitige) } p\text{-Wert ist das kleinste } \alpha \text{ mit } \frac{|\bar{x}_n - \mu_0|}{\sigma_0/\sqrt{n}} \geq u_{1-\alpha/2}.$$

Da $u_{1-\alpha/2}$ streng monoton fallend und stetig in α ist, gleicht der p-Wert gerade demjenigen α, für welches in der vorstehenden Ungleichung Gleichheit gilt. Daraus erhalten wir:

$$\text{Zweiseitiger } p\text{-Wert} = 2\left(1 - \Phi\left(\frac{|\bar{x}_n - \mu_0|}{\sigma_0/\sqrt{n}}\right)\right)$$

Die Formel für diesen p-Wert deckt sich mit seiner folgenden, sehr suggestiven Interpretation (die ebenfalls allgemein für jeden Test gilt):

> Der p-Wert ist die Wahrscheinlichkeit, dass die (zufällige!) Teststatistik unter H_0 einen Wert annimmt, der mindestens so weit „in Richtung H_1“ liegt wie derjenige, der sich für die konkret vorliegenden Daten ergeben hat.

Begründung: Zur Abkürzung sei $T_n := (\bar{X}_n - \mu_0)/(\sigma_0/\sqrt{n})$ die Teststatistik und $t_n^* := (\bar{x}_n - \mu_0)/(\sigma_0/\sqrt{n})$ der von ihr für die konkret vorliegenden Daten angenommene Wert. Dann gilt hier beim zweiseitigen Gaußtest:

$$\begin{aligned}
\mathbb{P}_{H_0}\begin{pmatrix}\text{Teststatistik } T_n \text{ realisiert sich min-}\\ \text{destens so weit „in Richtung } H_1\text{“ wie}\\ \text{der konkret vorliegende Wert } t_n^*.\end{pmatrix} &= \mathbb{P}_{H_0}\left(|T_n| \geq |t_n^*|\right)\\
= \quad 1 - \mathbb{P}_{H_0}\left(|T_n| < |t_n^*|\right) &= 1 - \mathbb{P}_{H_0}\left(-|t_n^*| < T_n < |t_n^*|\right)\\
= \quad 1 - \{\Phi(|t_n^*|) - \Phi(-|t_n^*|)\} &= 2(1 - \Phi(|t_n^*|)) \quad = \quad \text{Zweiseitiger } p\text{-Wert}
\end{aligned}$$

Bemerkungen:

1. Ein kleiner p-Wert bedeutet demnach, dass das Beobachtete unter H_0 unwahrscheinlich gewesen ist und daher gegen H_0 spricht. Als „Eselsbrücke“ zur Interpretation diene der Buchstabe „p“: Der p-Wert quantifiziert gewissermaßen die **P**lausibilität der Nullhypothese derart, dass H_0 umso weniger plausibel erscheint, je kleiner er ist. Anders formuliert:

Je kleiner der p-Wert, umso mehr „Vertrauen“ kann man in die Richtigkeit der Entscheidung haben, H_0 zu verwerfen.

   Ist ein p-Wert „sehr klein“, d. h., unterschreitet er das vorgegebene Signifikanzniveau α, so wird er – stellvertretend für die Testentscheidung H_0 zu verwerfen – oft kurz „signifikant“ (auf dem Niveau α) genannt.

2. Offenbar hängt der p-Wert von der Art der Alternative H_1 ab! Oben haben wir den zweiseitigen Fall betrachtet; den einseitigen diskutieren wir unten, wobei dort noch unterschieden werden muss, ob die Alternative $\mu < \mu_0$ oder $\mu > \mu_0$ lautet.

3. In der Praxis ist es sinnvoll, bei der Angabe eines Testergebnisses den p-Wert mitzuliefern, da er mehr Information enthält als die Testentscheidung allein, wie wir an unserem **Beispiel** illustrieren wollen: Dort hatten wir $n = 50$, $|\bar{x}_n - \mu_0| = 1.54$ und $\sigma_0 = 4.2$. Einsetzen dieser Werte in obige Formel liefert einen p-Wert von 0.009522, sodass ein Verwerfen von H_0 auf dem Niveau 0.01 möglich ist. Allerdings ist es doch ein etwas knapper Unterschied, auf dem diese Entscheidung beruht. Wäre $\bar{x}_n = 46.75$ g und somit $|\bar{x}_n - \mu_0| = 2.25$ beobachtet worden, ergäbe sich ein p-Wert von 0.00015 und das Niveau 0.01 würde *deutlich* unterschritten, sodass ein Verwerfen von H_0 wesentlich besser abgesichert wäre.

4. In der Praxis ist es aber nicht sinnvoll, *nur* den p-Wert anzugeben, denn aus seiner Kenntnis allein lässt sich im Allgemeinen nicht ableiten, warum die Daten zu einem signifikanten oder nicht signifikanten Ergebnis geführt haben: Ist der p-Wert signifikant, kann die Ursache ein wahrhaft großer Unterschied $|\mu - \mu_0|$ sein oder nur ein großes n relativ zur Streuung σ_0; ist er nicht signifikant, kann die Ursache ein wahrhaft kleines $|\mu - \mu_0|$ sein oder nur ein relativ zu σ_0 zu kleines n.

5. Des weiteren ist es sinnvoll bis gefordert, auf Basis der beobachteten Daten ein Konfidenzintervall für den Parameter anzugeben, über den eine Hypothese getestet wurde, da man in der Regel an der Größenordnung des Effektes der betrachteten Behandlung interessiert ist (und nicht nur daran, ob es ein signifikanter Effekt ist). (Vgl. z. B. [37, Gardner & Altman (1986)] und [51, ICH-E9 (1998), §5.5, p. 25 – 26].)

Wir zeigen nun die p-**Wert-Berechnung im Fall des einseitigen Gaußtests** von

$$H_0' : \mu \overset{(\leq)}{\geq} \mu_0 \qquad \text{gegen} \qquad H_1' : \mu \overset{(>)}{<} \mu_0$$

Gemäß der Entscheidungsregel auf Basis des kritischen Wertes (Seite 114, oben) folgt:

$$\text{Der (einseitige) } p\text{-Wert ist das kleinste } \alpha \text{ mit } \frac{\bar{x}_n - \mu_0}{\sigma_0/\sqrt{n}} \overset{(\geq +)}{\leq -} u_{1-\alpha},$$

sodass sich mit einer dem zweiseitigen Fall analogen Argumentation zu Stetigkeit und Monotonie von $u_{1-\alpha}$ in α ergibt:

$$\text{Einseitiger } p\text{-Wert} = 1 - \Phi\left(\overset{(+)}{-} \frac{\bar{x}_n - \mu_0}{\sigma_0/\sqrt{n}}\right) = \begin{cases} \Phi\left(\frac{\bar{x}_n - \mu_0}{\sigma_0/\sqrt{n}}\right) & \text{für} \quad H_1' : \mu < \mu_0 \\ 1 - \Phi\left(\frac{\bar{x}_n - \mu_0}{\sigma_0/\sqrt{n}}\right) & \text{"} \quad H_1' : \mu > \mu_0 \end{cases}$$

Zusammenfassend formulieren wir das allgemein gültige, **prinzipielle Vorgehen bei statistischen Tests**:

1. Formulierung von Hypothese H_0 und Alternative H_1 sowie Festlegung des Signifikanzniveaus α.
2. Wahl des Tests.
3. Erhebung der Daten und explorative Analyse, um die verteilungstheoretischen Voraussetzungen des Tests zu prüfen. (Falls zweifelhaft, zurück zu Punkt 2 oder sogar 1.)
4. Berechnung der Teststatistik und des p-Wertes.
5. Entscheidung für oder gegen H_0 unter Angabe von H_0 und H_1, des Tests und des p-Wertes (sowie eventuell relevanter Informationen zur Teststatistik wie z. B. ihre Freiheitsgrade).

8.2 Konfidenzintervalle für die Parameter der Normalverteilung

Häufig ist es zusätzlich zu einem Punktschätzer oder einem Hypothesentest für einen unbekannten Parameter – wie dem Erwartungswert oder der Varianz – gewünscht oder sogar gefordert, einen Bereichsschätzer mit einer festgelegten Überdeckungswahrscheinlichkeit $1 - \alpha$, also ein Konfidenzintervall (KI) zum (Konfidenz-)Niveau $1 - \alpha$ anzugeben. Dies ist sehr empfehlenswert, da der p-Wert allein (wie vorseitig in der 4. Bemerkung bereits erläutert) keine Aussage über die Größenordnung eines Effektes $\mu - \mu_0$ (oder $\mu_1 - \mu_2$ in noch folgenden Paragraphen) machen kann.

8.2.1 Der Erwartungswert μ

Im Modell unabhängiger Zufallsvariablen $X_1, \ldots, X_n$ mit der identischen Verteilung $\mathcal{N}(\mu, \sigma^2)$ ist bekanntermaßen $\sqrt{n}(\bar{X}_n - \mu)/\sigma \sim \mathcal{N}(0, 1)$ für alle $n \geq 1$, woraus man folgert, dass

$$P\left(\bar{X}_n - \frac{u_{1-\alpha/2}\,\sigma}{\sqrt{n}} \leq \mu \leq \bar{X}_n + \frac{u_{1-\alpha/2}\,\sigma}{\sqrt{n}}\right) = 2\Phi(u_{1-\alpha/2}) - 1 = 1 - \alpha$$

Bei bekanntem σ^2 ist ein Konfidenzintervall für den unbekannten Erwartungswert μ also wie folgt angebbar:

$X_1, \ldots, X_n$ seien unabhängig und identisch $\mathcal{N}(\mu, \sigma^2)$-verteilt. Dann ist

$$\left[\bar{X}_n - \frac{u_{1-\alpha/2}\,\sigma}{\sqrt{n}}, \bar{X}_n + \frac{u_{1-\alpha/2}\,\sigma}{\sqrt{n}}\right] \text{ ein } \underline{\text{exaktes KI}} \text{ für } \mu \text{ zum Niveau } 1-\alpha.$$

Beachte:

- Die nicht zufällige Länge des Konfidenzintervalls hängt vom Niveau $1-\alpha$, der Varianz σ^2 und vom Stichprobenumfang n ab: Je höher das Niveau bzw. je größer die Varianz, umso länger das Konfidenzintervall; je größer der Stichprobenumfang, umso kürzer das Konfidenzintervall.
- Das obige Konfidenzintervall ist symmetrisch um $\bar{X}_n$.

In Anwendungen dürfte es allerdings der Regelfall sein, dass σ^2 unbekannt ist (und somit ein Störparameter, Englisch: nuisance parameter). Das obige Konfidenzintervall ist dann nutzlos, da seine Grenzen σ explizit enthalten. Die Lösung des Problems ist, das unbekannte σ durch seinen Schätzer $\hat{\sigma}_n$ zu ersetzen, denn im betrachteten Normalverteilungsmodell ist $\sqrt{n}(\bar{X}_n - \mu)/\hat{\sigma}_n \sim t_{n-1}$ für alle $n \geq 2$, wobei t_{n-1} die (Studentsche) t-Verteilung mit $n-1$ Freiheitsgraden (kurz: t_{n-1}-Verteilung) ist, deren Verteilungsfunktion wir mit $F_{t_{n-1}}$ bezeichnen werden.

Daraus ergibt sich (analog zum Fall des bekannten σ^2) für jedes $n \geq 2$

$$P\left(\bar{X}_n - \frac{t_{n-1;1-\alpha/2}\,\hat{\sigma}_n}{\sqrt{n}} \leq \mu \leq \bar{X}_n + \frac{t_{n-1;1-\alpha/2}\,\hat{\sigma}_n}{\sqrt{n}}\right) = 2F_{t_{n-1}}(t_{n-1;1-\alpha/2}) - 1 = 1-\alpha,$$

wobei $t_{n-1;1-\alpha/2}$ das $(1-\alpha/2)$-Quantil der t_{n-1}-Verteilung ist.

Bei unbekanntem σ^2 folgt also:

$X_1, \ldots, X_n$ seien unabhängig und identisch $\mathcal{N}(\mu, \sigma^2)$-verteilt. Dann ist

$$\left[\bar{X}_n - \frac{t_{n-1;1-\alpha/2}\,\hat{\sigma}_n}{\sqrt{n}}, \bar{X}_n + \frac{t_{n-1;1-\alpha/2}\,\hat{\sigma}_n}{\sqrt{n}}\right] \text{ ein } \underline{\text{exaktes KI}} \text{ für } \mu \text{ zum Niveau } 1-\alpha.$$

Beachte:

- Die zufällige (!) Länge dieses Konfidenzintervalls hängt vom Niveau $1-\alpha$, der (zufälligen!) Stichprobenvarianz $\hat{\sigma}_n^2$ und vom Stichprobenumfang n ab: Je höher das Niveau bzw. je größer die Stichprobenvarianz, umso länger das Konfidenzintervall; je größer der Stichprobenumfang, umso kürzer das Konfidenzintervall.
- Aufgrund der Tatsache, dass die $(1-\alpha)$-Quantile der t-Verteilungen stets betraglich größer als die der Standardnormalverteilung sind, ist dieses Konfidenzintervall grundsätzlich etwas länger als ein Konfidenzintervall bei bekannter Varianz, selbst wenn (zufällig) $\sigma = \hat{\sigma}_n$ sein sollte. Dies ist gewissermaßen der „Preis“, den man für die Unkenntnis der Varianz zu zahlen hat.

Bemerkungen:

- Am Beispiel des obigen Konfidenzintervalls rekapitulieren wir die generell gültige **Häufigkeitsinterpretation** von Konfidenzintervallen:

Das Ereignis $\left\{\bar{X}_n - \frac{t_{n-1;1-\alpha/2}\,\hat{\sigma}_n}{\sqrt{n}} \leq \mu \leq \bar{X}_n + \frac{t_{n-1;1-\alpha/2}\,\hat{\sigma}_n}{\sqrt{n}}\right\}$ hat die Wahrscheinlichkeit $1-\alpha$. D. h., für $(1-\alpha)\cdot 100$ % *aller* Stichproben $x_1, \ldots, x_n$ trifft die Aussage

$$\bar{x}_n - \frac{t_{n-1;1-\alpha/2}\,s_n}{\sqrt{n}} \leq \mu \leq \bar{x}_n + \frac{t_{n-1;1-\alpha/2}\,s_n}{\sqrt{n}}$$

zu, für die verbleibenden $\alpha \cdot 100$ % *aller* Stichproben jedoch nicht.

- Eine häufig anzutreffende Formulierung (hier mit erfundenen Zahlen) lautet

  „Das wahre μ liegt mit einer Wahrscheinlichkeit von 95 % zwischen 0.507 und 0.547.“ } **DAS IST UNSINN!**

  Nach unserer Modellierung hat das unbekannte μ einen *festen* Wert, also liegt es entweder zwischen 0.507 und 0.547 oder nicht. Nicht μ ist zufällig, sondern die Schranken des Konfidenzintervalls, und zwar natürlich nur *bevor* die Daten erhoben worden sind. Neue Stichproben ergeben also im Allgemeinen *andere* Schranken bei *gleichem* μ. Die **korrekte Formulierung** lautet schlicht „$[0.507, 0.547]$ ist ein 95 %-Konfidenzintervall für μ“ und ihre (Häufigkeits-) Interpretation steht oben.

- Sollen die Grenzen eines Konfidenzintervalls gerundet werden, so ist die Untergrenze stets *ab*- und die Obergrenze stets *auf*zurunden, um zu vermeiden, dass das Konfidenzintervall durch Rundung zu kurz wird und deshalb das nominelle Niveau nicht mehr einhält. Das so erhaltene Konfidenzintervall hat dann *mindestens* das angestrebte Niveau. (Eine Vorgehensweise, die dazu führt, dass Niveaus auf jeden Fall eingehalten und möglicherweise „übererfüllt“ werden, nennt man auch konservativ.)

8.2.2 Die Varianz σ^2

Für unabhängig und identisch $\mathcal{N}(\mu, \sigma^2)$-verteilte Zufallsvariablen $X_1, \ldots, X_n$ ist $(n-1)\hat{\sigma}_n^2/\sigma^2 \sim \chi^2_{n-1}$ für alle $n \geq 2$. Daraus leitet sich (bei unbekanntem Erwartungswert μ) das folgende Konfidenzintervall für die unbekannte Varianz σ^2 ab:

> $X_1, \ldots, X_n$ seien unabhängig und identisch $\mathcal{N}(\mu, \sigma^2)$-verteilt. Dann ist
>
> $$\left[\frac{(n-1)\hat{\sigma}_n^2}{\chi^2_{n-1;1-\alpha/2}}, \frac{(n-1)\hat{\sigma}_n^2}{\chi^2_{n-1;\alpha/2}}\right] \quad \text{ein exaktes KI für } \sigma^2 \text{ zum Niveau } 1-\alpha.$$

Beachte: Die zufällige (!) Länge dieses – um $\hat{\sigma}_n^2$ *asymmetrischen* – Konfidenzintervalls hängt vom Niveau $1-\alpha$, der (zufälligen!) Stichprobenvarianz $\hat{\sigma}_n^2$ und vom Stichprobenumfang n ab: Je höher das Niveau bzw. je größer die Stichprobenvarianz, umso länger das Konfidenzintervall; je größer der Stichprobenumfang, umso kürzer das Konfidenzintervall (was hier nicht offensichtlich ist, aber durch die Beziehung $F_{k,\infty} = \chi^2_k/k$ und das monotone Wachstum des Quantils $F_{k,\infty;\gamma}$ in k nachvollzogen werden kann).

8.2.3 Zur Fallzahlschätzung für Konfidenzintervalle mit vorgegebener „Präzision“

Soll bei der Schätzung des Erwartungswertes μ einer Normalverteilung durch ein Konfidenzintervall eine gewisse „Präzision“ erzielt werden, so ist zunächst zu klären, was unter „Präzision“ zu verstehen ist. Hier gibt es mindestens zwei verschiedene Konzepte, wie z. B. dass der *Erwartungswert* der zufälligen (!) Länge des Konfidenzintervalls eine vorgegebene Schranke nicht überschreiten soll (was natürlich nicht bedeutet, dass das konkret berechnete Intervall diese Längenbeschränkung erfüllt) oder dass das zufällige (!) Konfidenzintervall mit vorgegebener großer Wahrscheinlichkeit (auch Power oder Güte genannt und mit $1-\beta$ bezeichnet) komplett innerhalb eines Intervalls $[\mu - \Delta, \mu + \Delta]$ mit ebenfalls vorgegebenem Δ zu liegen kommt (und dies jeweils weiterhin bei festgelegter Überdeckungswahrscheinlichkeit $1-\alpha$ für μ). Ausführlich sind die Problematik und Lösungen dafür z. B. in [6, Bock (1998)] in Abschnitt 3.2 beschrieben.

8.3 Eine Hilfsfunktion für die explorative Datenanalyse

Viele klassische inferenzstatistische Verfahren für metrische Daten hängen stark von Verteilungsannahmen über die Daten ab, auf die sie angewendet werden sollen. In den in diesem Abschnitt vorgestellten *parametrischen* Verfahren sind dies die Normalverteilungsannahme und die Unabhängigkeit der Daten. Die Beurteilung der Gültigkeit dieser Annahmen kann durch die explorative Datenanalyse (EDA) unterstützt werden, wofür mehrere grafische Methoden zur Verfügung stehen, die wir zum Teil bereits kennengelernt haben.

In Abschnitt 6.1 hatten wir schon einige dieser Methoden in einer neuen Funktion zusammengefasst, die wir im Folgenden noch etwas erweitern:

Eine (erweiterte) Funktion für die explorative Datenanalyse

```
> eda <- function( x) {
+ oldp <- par( mfrow = c( 3,2), cex = 0.8,
+   mar = c( 3.5,3,2.6,0.5)+0.1,
+   mgp = c( 1.7, 0.6,0))
+
+ hist( x, freq = FALSE, ylab = "Dichte",
+       xlab = "", main = "Histogramm")
+ boxplot( x, horizontal = TRUE,
+          main = "Boxplot")
+ set.seed( 42)
+ stripchart( x, add = TRUE, col = "blue",
+             method = "jitter")
+
+ qa <- diff( quantile( x, c( 1/4, 3/4)))
+ dest <- density( x, width = qa)
+ plot( dest, ylab = "Dichte",
+       main = "Dichteschätzer")
+
+ qqnorm( x, datax = TRUE,
+         ylab = "Stichprobenquantile",
+         xlab = "Theoretische Quantile")
+ qqline( x, datax = TRUE)
+ ts.plot( x, ylab = "",
+          xlab = "Zeit bzw. Index",
+          main = "Zeitreihe")
+
+ acf( x, main = "Autokorrelation")
+
+ par( oldp)
+
+ d <- max( 3, getOption( "digits") - 3)
+ c( summary( x),
+    "St.Dev." = signif( sd( x),
+                        digits = d))
+ }
```

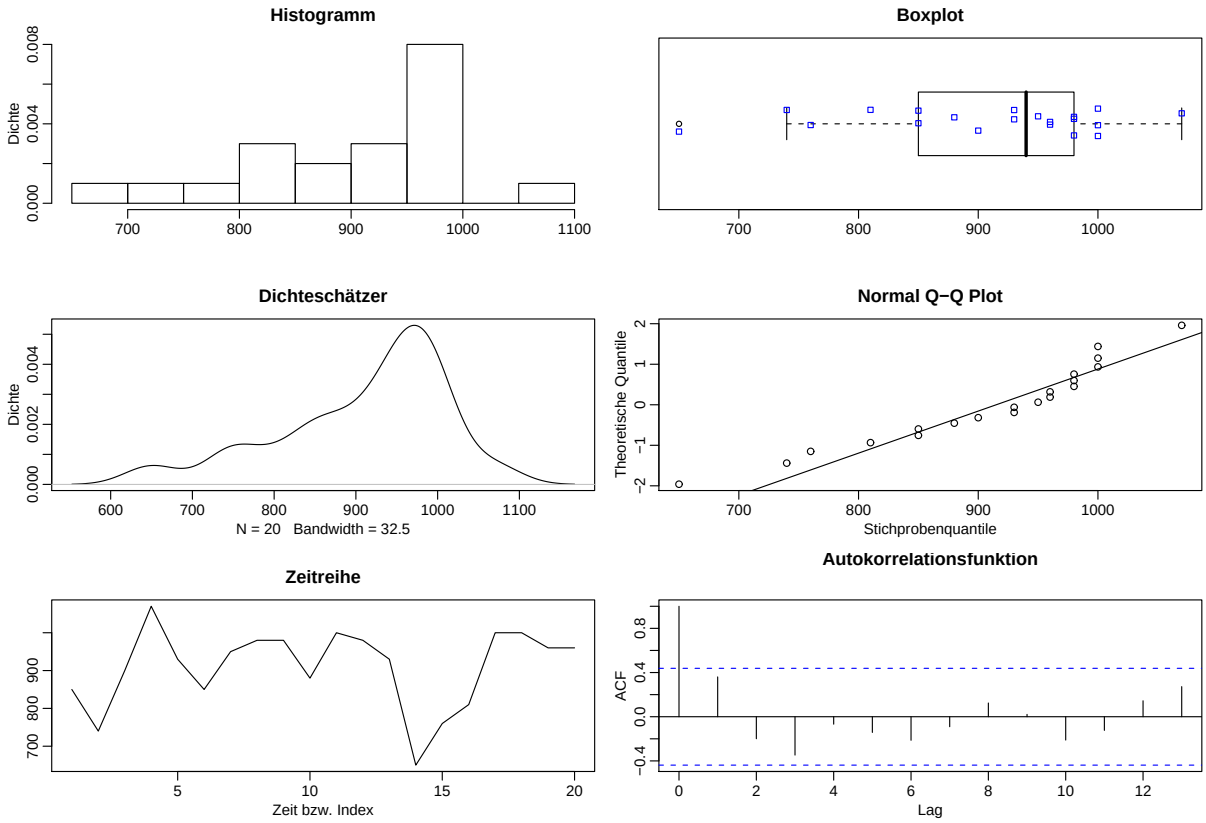
Die hiesige Funktion `eda()` ist gegenüber der in Abschnitt 6.1 definierten insofern erweitert, als dass durch `par( mfrow = c( 3,2), ....)` zunächst ein (3×2)-Mehrfachplotrahmen mit gewissen Modifikationen – vor allem von Abständen – des Standardlayouts angelegt und die *vorherige* `par()`-Einstellung in `oldp` gespeichert wird. Sodann werden in die ersten vier der sechs Plotrahmen – ebenfalls leicht modifiziert im Vergleich zu Abschnitt 6.1 – ein flächennormiertes Histogramm, ein horizontaler Boxplot mit leicht „verwackelt“ überlagerten Rohdaten (in blau), ein Kern-Dichteschätzer (mit Gaußkern und dem Quartilsabstand der Daten als Fensterbreite) sowie ein Normal-Q-Q-Plot (wegen `datax = TRUE` mit den Daten auf der x-Achse und mit vertauschter „Zuständigkeit“ von `xlab` und `ylab` für die Achsenlabels) samt Soll-Linie geplottet.

Zusätzlich werden durch `ts.plot()` ein „Zeitreihenplot“ (`ts` wie “time series”) der Elemente von `x` gegen ihre Indices produziert und mittels `acf()` die Autokorrelationsfunktion der Daten für mehrere “lags” (zusammen mit den 95 %-Konfidenzgrenzen) gezeichnet; diese beiden letzten Plots können eine mögliche serielle Korrelation der Daten zutage fördern. Als vorvorletztes wird die *vorherige* `par()`-Einstellung wieder reaktiviert und als Rückgabewert liefert die Funktion dann den Vektor der “summary statistics” des Arguments `x` ergänzt um die Standardabweichung der Daten mit zur Ausgabe von `summary()` passender Anzahl signifikanter Ziffern.

Zur Demonstration verwenden wir wieder den Datensatz von Michelson zur Bestimmung der Lichtgeschwindigkeit (vgl. Seite 90):

```
> v <- c( 850,  740,  900, 1070,  930,  850,  950,  980,  980,  880,
+        1000,  980,  930,  650,  760,  810, 1000, 1000,  960,  960)
> eda( v)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. St.Dev.
  650.0   850.0   940.0   909.0   980.0  1070.0   104.9
```

Das Resultat der Anwendung der Funktion `eda()` auf `v` ist auch hier die Ausgabe der "summary statistics" für `v` und die folgenden Grafiken:



Natürlich könnte das Layout noch weiter verbessert werden, doch wir wenden uns jetzt den angekündigten Ein- und Zweistichprobenproblemen zu.

An dieser Stelle sei aber zunächst an die Bemerkung zum Sinn oder Unsinn eines statistischen Tests der Normalverteilungshypothese auf Seite 79 in §4.2.3 erinnert.

Außerdem bemerken wir des Weiteren vorher noch, dass z. B. [87, Zuur et al. (2009)] einen Beleg dafür liefert, dass sich – hier in Ökologie und Evolutionsbiologie tätige – angewandte StatistikerInnen nach wie vor intensiv Gedanken darüber machen, dass und wie eine adäquate explorative Datenanalyse zentraler Bestandteil der Auswertung von Daten sein muss (obwohl es sich auf den ersten Blick um Trivialitäten zu handeln scheint).

8.4 Ein Einstichproben-Lokationsproblem

Lokations- (oder eben Lage-)Parameter einer Verteilung charakterisieren ihre „zentrale Lage“. Die bekanntesten Lokationsparameter sind der Erwartungswert (sofern existent) und der (stets existente) Median einer Verteilung.

Wir betrachten in diesem Abschnitt Tests für Lokationshypothesen über *eine* zugrunde liegende Population. Unter der Normalverteilungsannahme wird dazu Students t-Test (mit verschiedenen Alternativen) für Hypothesen über den Erwartungswert verwendet. In Fällen, in denen die Daten *nicht* normalverteilt sind – wie es die explorative Datenanalyse im vorliegenden Beispiel der Michelson-Daten auf der voherigen Seite übrigens nahelegt – sollten eher nicht-parametrische (Rang-)Verfahren, wie z. B. Wilcoxons Vorzeichen-Rangsummentest für Hypothesen über den Median einer immerhin noch stetigen und symmetrischen Verteilung, verwendet werden.

Natürlich untersuchen Tests für Erwartungswerthypothesen nicht dasselbe wie solche für Medianhypothesen! Dies muss bei der Interpretation der Ergebnisse beachtet werden (auch wenn das in der praktischen Anwendung selten geschieht ...). Zu Illustrationszwecken werden wir *beide* Verfahren am Michelson-Datensatz aus dem vorherigen Abschnitt vorführen.

8.4.1 Der Einstichproben-t-Test

Als Referenz und zur Erinnerung hier zunächst eine Wiederholung von etwas Theorie:

<u>Annahmen:</u> $X_1, \ldots, X_n$ sind unabhängig $\mathcal{N}(\mu, \sigma^2)$-verteilt mit unbekannten μ und σ^2.

<u>Zu testen:</u>

$$H_0 : \mu = \mu_0 \qquad \text{gegen} \qquad H_1 : \mu \neq \mu_0 \qquad \text{zum Signifikanzniveau } \alpha$$

bzw.

$$H_0' : \mu \overset{(\leq)}{\geq} \mu_0 \qquad \text{gegen} \qquad H_1' : \mu \overset{(>)}{<} \mu_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

<u>Teststatistik:</u>

$$\frac{\bar{X}_n - \mu_0}{\hat{\sigma}_n/\sqrt{n}} \sim t_{n-1} \qquad \begin{array}{l}\text{unter } H_0 \text{ bzw. unter } \mu = \mu_0, \\ \text{was der „Rand“ von } H_0' \text{ ist,}\end{array}$$

wobei $\bar{X}_n = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} X_i$ ist und $\hat{\sigma}_n^2 = \dfrac{1}{n-1}\displaystyle\sum_{i=1}^{n}(X_i - \bar{X}_n)^2$ der Schätzer für σ^2.

<u>Entscheidungsregel</u> für konkrete Daten $x_1, \ldots, x_n$ auf Basis des kritischen Wertes:

$$\text{Verwirf } H_0 \quad \Longleftrightarrow \quad \frac{|\bar{x}_n - \mu_0|}{s_n/\sqrt{n}} \geq t_{n-1;1-\alpha/2}$$

bzw.

$$\text{verwirf } H_0' \quad \Longleftrightarrow \quad \frac{\bar{x}_n - \mu_0}{s_n/\sqrt{n}} \overset{(\geq +)}{\leq -} t_{n-1;1-\alpha},$$

wobei $\bar{x}_n$ das arithmetische Mittel und s_n^2 die Stichprobenvarianz der Daten sowie $t_{k;\gamma}$ das γ-Quantil der t_k-Verteilung ist.

Alternativ: Entscheidungsregel für konkrete Daten $x_1, \ldots, x_n$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei

$$p\text{-Wert} = \begin{cases} 2\left(1 - F_{t_{n-1}}\left(\frac{|\bar{x}_n - \mu_0|}{s_n/\sqrt{n}}\right)\right) & \text{für} \quad H_1 : \mu \neq \mu_0 \\ F_{t_{n-1}}\left(\frac{\bar{x}_n - \mu_0}{s_n/\sqrt{n}}\right) & \text{"} \quad H_1' : \mu < \mu_0 \\ 1 - F_{t_{n-1}}\left(\frac{\bar{x}_n - \mu_0}{s_n/\sqrt{n}}\right) & \text{"} \quad H_1' : \mu > \mu_0 \end{cases}$$

und F_{t_k} die (um 0 symmetrische) Verteilungsfunktion der t_k-Verteilung ist. (Die Herleitung dieser p-Wert-Formel erfolgt völlig analog zu der des Gaußtests auf den Seiten 115 und 117.)

Die **R**-Funktion `t.test()` berechnet den p-Wert zu den eingegebenen Daten und überlässt die Testentscheidung dem/der Anwender/in (wie die meisten Software-Pakete). Dies geschieht auch bei allen anderen implementierten Tests. Aus diesem Grund werden wir im Folgenden nur noch die Entscheidungsregeln auf Basis des p-Wertes formulieren. Nun zur Durchführung der t-Tests in **R** für die Michelson-Daten:

Students Einstichproben-t-Test	

`> t.test( v, mu = 990)`
 	
`  One-sample t-Test`
 	
`data: v`
`t = -3.4524, df = 19,`
`p-value = 0.002669`
`alternative hypothesis: true mean`
`is not equal to 990`
`95 percent confidence interval:`
`  859.8931 958.1069`
`sample estimates:`
`  mean of x`
`  909`
	Die Funktion `t.test()` führt Students t-Test über den Erwartungswert μ einer als normalverteilt angenommenen Population durch. Ihr erstes Argument erwartet die Daten (hier in `v`) aus jener Population. Dem Argument `mu` wird der hypothetisierte Erwartungswert μ_0 übergeben, zu lesen als H_0 : `mu` $= \mu_0$, hier mit $\mu_0 = 990$. Als Resultat liefert die Funktion den Namen der Datenquelle (`data`), den Wert der Teststatistik (`t`), die Anzahl ihrer Freiheitsgrade (`df`) und den p-Wert (`p-value`) des – hier – zweiseitigen Tests. Außerdem werden die Alternative (Voreinstellung: Zweiseitige Alternative H_1 : `mu` $\neq \mu_0$), das 95 %-Konfidenzintervall für μ (vgl. 118 oben) und das arithmetische Mittel der Daten angegeben. Ohne Angabe für `mu` wird die Voreinstellung `mu` $= 0$ verwendet.

`> t.test( v, mu = 990,`
`+ alternative = "greater")`
`....`
`95 percent confidence interval:`
`  868.4308  Inf`
`....`
 	
`> t.test( v, mu = 990,`
`+ alter = "less")`
`....`
 	
`> t.test( v, mu = 990,`
`+ conf.level = 0.90)`
`....`
	Über das Argument `alternative` lässt sich die Alternative spezifizieren: `alternative = "greater"` bedeutet H_1 : `mu` $> \mu_0$ und `alter = "less"` bedeutet H_1 : `mu` $< \mu_0$. (In diesen Fällen sind die angegebenen, hier aber zum Teil nicht gezeigten Konfidenzintervalle einseitig.) Die Argumentnamen dürfen – wie immer – so weit verkürzt werden, wie eindeutige Identifizierbarkeit gewährleistet ist. Dasselbe gilt für die zugewiesenen Werte `"less"` (max. Abk.: `"l"`) und `"greater"` (max. Abk.: `"g"`).)
	Mit `conf.level` ist das Konfidenzniveau beliebig wählbar (Voreinstellung: 95 %).

8.4.2 Wilcoxons Vorzeichen-Rangsummentest

Wenn die Normalverteilungsannahme *nicht* gerechtfertigt erscheint, dann stehen verschiedene nicht-parametrische (Rang-)Verfahren für das Einstichproben-Lokationsproblem zur Verfügung, wie Wilcoxons Vorzeichentest, der lediglich eine stetige (aber sonst beliebige) Verteilung mit eindeutig bestimmtem Median voraussetzt, und Wilcoxons Vorzeichen-Rangsummentest, der eine stetige und um einen eindeutig bestimmten Median symmetrische Verteilung benötigt. Beispiele hierfür sind jede t-, Cauchy-, Doppelexponential-, $U(-a,a)$- oder $\Delta(-a,a)$-Verteilung. (Beispiele für stetige und symmetrische Verteilungen *ohne* eindeutig bestimmten Median lassen sich z. B. aus zwei Verteilungen mit kompakten Trägern leicht zusammensetzen. Memo: Falls der Erwartungswert einer symmetrischen Verteilung existiert, ist er gleich ihrem Median.)

Empfehlenswerte Referenzen zu dem Thema sind [16, Büning & Trenkler (1994)] und [44, Hettmansperger (1984)]. Bzgl. Tests für beliebige, also auch nicht stetige Verteilungen ist [15, Brunner & Munzel (2013)] mit einer hervorragenden Darstellung der Thematik sowie einer guten Mischung aus Theorie und durchgerechneten Beispielen sehr zu empfehlen.

Wir stellen hier Wilcoxons Vorzeichen-Rangsummentest näher vor, welcher in **R** durch die Funktion `wilcox.test()` realisiert ist. Auf Wilcoxons Vorzeichentest gehen wir in §8.4.3 kurz ein.

Annahmen: $X_1, \ldots, X_n$ sind unabhängig und nach einer stetigen Verteilungsfunktion $F(\cdot - \theta)$ verteilt, wobei F **symmetrisch** um 0 ist und 0 ihr eindeutig bestimmter Median ist (d. h., $F(\cdot - \theta)$ ist symmetrisch um θ und θ ist ihr eindeutig bestimmter Median sowie Erwartungswert, falls letzterer existiert).

Zu testen:

$$H_0 : \theta = \theta_0 \qquad \text{gegen} \qquad H_1 : \theta \neq \theta_0 \qquad \text{zum Signifikanzniveau } \alpha$$

bzw.

$$H_0' : \theta \overset{(\leq)}{\geq} \theta_0 \qquad \text{gegen} \qquad H_1' : \theta \overset{(>)}{<} \theta_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

Teststatistik:

$$W_n^+ := \sum_{i=1}^n 1_{\{X_i - \theta_0 > 0\}} R(|X_i - \theta_0|),$$

wobei $R(z_i)$ der *Rang* von z_i unter $z_1, \ldots, z_n$ ist (d. h., $R(z_i) = j$, falls $z_i = z_{j:n}$). Die exakte Verteilung von Wilcoxons Vorzeichen-Rangsummenstatistik W_n^+ (auch Wilcoxons signierte Rangsummenstatistik genannt) ist unter $\theta = \theta_0$ bekannt, aber für großes n sehr aufwändig explizit zu berechnen. Einfache, kombinatorische Überlegungen zeigen, dass

$$\mathbb{E}[W_n^+] = \frac{n(n+1)}{4} \qquad \text{und} \qquad \mathrm{Var}(W_n^+) = \frac{n(n+1)(2n+1)}{24} \qquad \text{unter } \theta = \theta_0$$

sowie dass W_n^+ symmetrisch um ihren Erwartungswert verteilt ist. Des Weiteren ist nachweisbar, dass W_n^+ unter $\theta = \theta_0$ asymptotisch normalverteilt ist:

$$Z_n := \frac{W_n^+ - \mathbb{E}[W_n^+]}{\sqrt{\mathrm{Var}(W_n^+)}} \overset{n\to\infty}{\longrightarrow} \mathcal{N}(0,1) \quad \text{in Verteilung unter } \theta = \theta_0$$

Treten unter den $|X_i - \theta_0|$ exakte Nullwerte auf, so werden diese eliminiert und n wird auf die Anzahl der von 0 verschiedenen Werte gesetzt. Treten (z. B. aufgrund von Messungenauigkeiten in den Daten) Bindungen unter den (von 0 verschiedenen) $|X_i - \theta_0|$ auf, so wird die Methode der Durchschnittsränge verwendet, was den Erwartungswert von W_n^+ nicht, wohl aber ihre Varianz beeinflusst. In beiden Fällen wird in `wilcox.test()` per Voreinstellung die Normalapproximation verwendet, aber mit einer im Nenner leicht modifizierten Statistik Z_n^{mod} anstelle von Z_n (worauf wir jedoch hier nicht näher eingehen). Z_n^{mod} hat dieselbe Asymptotik wie Z_n.

Entscheidungsregel für konkrete Daten $x_1, \ldots, x_n$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei für die Berechnung des p-Wertes die folgende Fallunterscheidung zur Anwendung kommt, in welcher w_n der zu $x_1, \ldots, x_n$ realisierte Wert von W_n^+ sei:

1. Keine Bindungen, keine exakten Nullwerte und n „klein“ (was in `wilcox.test()` $n < 50$ bedeutet):

$$p\text{-Wert} = \begin{cases} 2F_n^+(w_n) & \text{für} \quad H_1 : \theta \neq \theta_0, \text{wenn } w_n \leq n(n+1)/4 \\ 2(1 - F_n^+(w_n - 1)) & \text{"} \quad H_1 : \theta \neq \theta_0, \text{wenn } w_n > n(n+1)/4 \\ F_n^+(w_n) & \text{"} \quad H_1' : \theta < \theta_0 \\ 1 - F_n^+(w_n - 1) & \text{"} \quad H_1' : \theta > \theta_0 \end{cases}$$

   wobei F_n^+ die um $n(n+1)/4$ symmetrische Verteilungsfunktion der diskreten (!) Verteilung von W_n^+ unter $\theta = \theta_0$ ist.

2. Bindungen oder exakte Nullwerte oder n „groß“ (in `wilcox.test()` $n \geq 50$):

   Hier wird Z_n^{mod} anstelle von Z_n verwendet und außerdem, um die Approximation an die asymptotische Normalverteilung für endliches n zu verbessern, im Zähler von Z_n^{mod} eine sogenannte Stetigkeitskorrektur vorgenommen. Der Zähler lautet dann

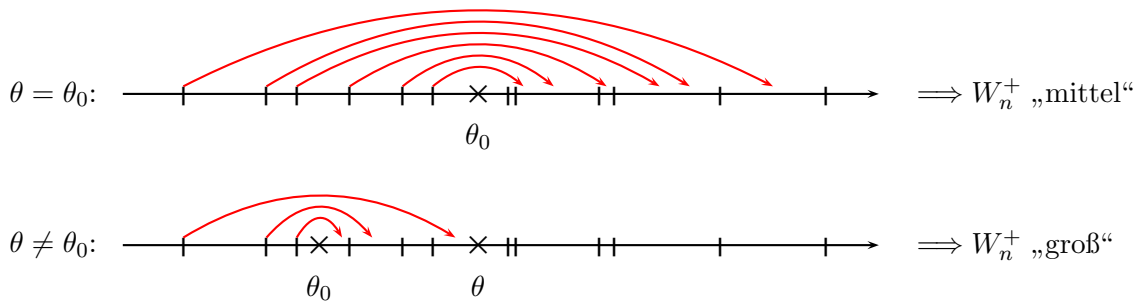
$$W_n^+ - \mathbb{E}[W_n^+] - c, \quad \text{wobei} \quad c = \begin{cases} \operatorname{sign}(W_n^+ - \mathbb{E}[W_n^+]) \cdot 0.5 & \text{für} \quad H_1 : \theta \neq \theta_0 \\ 0.5 & \text{"} \quad H_1' : \theta > \theta_0 \\ -0.5 & \text{"} \quad H_1' : \theta < \theta_0 \end{cases}$$

   Wenn nun z_n^{mod} die konkrete Realisierung von Z_n^{mod} bezeichne, dann ist

$$p\text{-Wert} = \begin{cases} 2\left(1 - \Phi\left(\left|z_n^{mod}\right|\right)\right) & \text{für} \quad H_1 : \theta \neq \theta_0 \\ \Phi\left(z_n^{mod}\right) & \text{"} \quad H_1' : \theta < \theta_0 \\ 1 - \Phi\left(z_n^{mod}\right) & \text{"} \quad H_1' : \theta > \theta_0 \end{cases}$$

Beachte: Die Verteilung von W_n^+ unter H_0 ist sehr empfindlich gegenüber Abweichungen von der Symmetrieannahme (siehe [14, Brunner und Langer (1999)], §7.1.2).

Zur heuristischen Motivation der Funktionsweise von Wilcoxons Vorzeichen-Rangsummenstatistik seien die folgenden zwei „prototypischen“ Szenarien skizziert und darunter ihre Wirkung auf W_n^+ beschrieben:



Durch die Betragsbildung $|X_i - \theta_0|$ werden die links von θ_0 liegenden X_i quasi auf die rechte Seite von θ_0 „gespiegelt“ (im Bild symbolisiert durch die Pfeile). Danach werden Rangzahlen

für die (gespiegelten und ungespiegelten) Werte $|X_i - \theta_0|$ vergeben. W_n^+ „wertet“ jedoch nur die Rangzahlen derjenigen $|X_i - \theta_0|$, deren X_i auch schon *vor* der Spiegelung rechts von θ_0 gelegen haben, und quantifiziert somit den Grad der „Durchmischung“ der $|X_i - \theta_0|$ dergestalt, dass W_n^+ große Werte oder kleine Werte annimmt, wenn der Durchmischungsgrad gering ist (wie im unteren Szenario).

Wilcoxons Vorzeichen-Rangsummentest	

`> wilcox.test( v, mu = 990)`
 	
`  Wilcoxon signed-rank test`
`  with continuity correction`
 	
`data:  v`
`V = 22.5, p-value = 0.00213`
`alternative hypothesis:`
`true mu is not equal to 990`
 	
`Warning message:`
`In wilcox.test.default(v, mu =`
`990) : cannot compute exact`
`  p-value with ties`
	`wilcox.test()` führt in dieser Form Wilcoxons Vorzeichen-Rangsummentest für den Median θ einer als stetig und symmetrisch um θ verteilt angenommenen Population durch, wozu ihr erstes Argument die Daten (hier in `v`) aus jener Population erwartet. An `mu` wird der hypothetisierte Median θ_0 übergeben, zu lesen als H_0 : `mu` $= \theta_0$. Zurückgeliefert werden der Name der Datenquelle, der Wert der Teststatistik W_n^+ (`V`), der p-Wert (`p-value`) des zweiseitigen Tests und die Alternative (Voreinstellung: Zweiseitige Alternative H_1 : `mu` $\neq \theta_0$; “`true mu`” meint θ). Der p-Wert wurde – ohne Hinweis! – über die approximativ normalverteilte Teststatistik Z_n^{mod} berechnet, da in den Michelson-Daten Bindungen auftreten, und dies (per Voreinstellung) `with continuity correction`. Immerhin wird eine entsprechende `Warning message` ausgegeben.
	Ohne Angabe für `mu` würde die Voreinstellung `mu` $= 0$ verwendet. Durch das nicht gezeigte Argument `alternative` von `wilcox.test()` ließe sich (wie bei `t.test()`) die Alternative spezifizieren: `alternative = "greater"` ist H_1 : `mu` $> \theta_0$ und `alter = "less"` ist H_1 : `mu` $< \theta_0$.

`> v2 <- v + runif( length( v))`
`> wilcox.test( v2, mu = 990)`
 	
`  Wilcoxon signed-rank test`
 	
`data:  v2`
`V = 27, p-value = 0.002325`
`alternative hypothesis:`
`true mu is not equal to 990`
	Brechen wir (zur Veranschaulichung) die Bindungen in den Daten künstlich auf, erhalten wir die Ausgabe der „nicht-approximativen“ Version von Wilcoxons Vorzeichen-Rangsummentest: Den Wert der Teststatistik W_n^+ (`V`) und den exakten p-Wert des zweiseitigen Tests sowie die Alternative.
	(Hinweis: `jitter(v)` anstelle von `v + runif(length( v))` ist empfehlenswerter, da zurückhaltender beim Bindungen Aufbrechen; siehe `jitter()`s Online-Hilfe.)

Hinweise:

- Mit dem nicht gezeigten Argument `conf.int = TRUE` erhielte man den Hodges–Lehmann–Schätzer für den „Pseudomedian“, der im hier vorliegenden Fall einer symmetrischen Verteilung gleich ihrem Median θ ist, ergänzt um ein Konfidenzintervall (das ein exaktes ist, wenn auch der p-Wert exakt ist, und ansonsten ein approximatives). (Siehe die Online-Hilfe und für mathematische Details siehe z. B. [44, Hettmansperger (1984), Abschnitt 2.3] oder [48, Hollander & Wolfe (1973), Abschnitte 3.2 und 3.3].)
- Das **R**-Paket `exactRankTests`, das allerdings “no longer under development” ist, bietet durch seine Funktion `wilcox.exact()` auch für Stichproben *mit* Bindungen die Berechnung exakter p-Werte des Wilcoxon-Rangsummentests. Stattdessen und auch unabhängig davon gibt es das **R**-Paket `coin`, dessen Funktion `wilcox_test()` exakte und approximative *bedingte* p-Werte der Wilcoxon-Tests liefert.

8.4.3 Wilcoxons Vorzeichentest

Wilcoxons Vorzeichentest setzt, wie zu Beginn des vorherigen Paragraphen bereits erwähnt, lediglich eine stetige (aber sonst beliebige) Verteilung mit eindeutig bestimmtem Median voraus. Er ist in **R** nicht durch eine eigene Funktion realisiert, da er auf einen einfachen Binomialtest hinausläuft, den wir im Abschnitt 9.2 „Einstichprobenprobleme im Binomialmodell“ ausführlich behandeln. Daher fassen wir uns hier kurz:

Annahmen: $X_1, \ldots, X_n$ sind unabhängig und nach einer stetigen Verteilungsfunktion $F(\cdot - \theta)$ verteilt, wobei 0 der eindeutig bestimmte Median von F ist (und damit θ der eindeutig bestimmte Median von $F(\cdot - \theta)$).

Zu testen:

$$H_0 : \theta = \theta_0 \qquad \text{gegen} \qquad H_1 : \theta \neq \theta_0 \qquad \text{zum Signifikanzniveau } \alpha$$

bzw.

$$H_0' : \theta \overset{(\leq)}{\geq} \theta_0 \qquad \text{gegen} \qquad H_1' : \theta \overset{(>)}{<} \theta_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

Teststatistik:

$$W_n := \sum_{i=1}^{n} 1_{\{X_i - \theta_0 > 0\}}$$

Offenbar sind die Indikatoren $1_{\{X_i - \theta_0 > 0\}}$ unter $\theta = \theta_0$ unabhängig und identisch verteilte Bernoulli-Variablen zum Parameter $p \equiv \mathbb{P}\left(1_{\{X_i - \theta_0 > 0\}}\right) = \mathbb{P}(X_i > \theta_0) = 1/2$ (weil ja θ_0 der Median ist), sodass W_n unter $\theta = \theta_0$ eine Binomial-Verteilung zu den Parametern n und $p = 1/2$ besitzt. Für alles Weitere verweisen wir auf den schon erwähnten Abschnitt 9.2 „Einstichprobenprobleme im Binomialmodell“.

8.5 Zweistichproben-Lokations- und Skalenprobleme

Hier werden Hypothesen über Lokations- bzw. Skalen- (also Streuungs-)*Unterschiede* zwischen zwei zugrunde liegenden Populationen getestet. Unter der Normalverteilungsannahme und der Annahme, dass die (in der Regel unbekannten!) Varianzen in den beiden Populationen gleich sind, wird für den Test auf Erwartungswertegleichheit Students Zweistichproben-t-Test verwendet. Um vor der Durchführung des t-Tests die Annahme der Varianzengleichheit zu testen, gibt es einen F-Test. Bei verschiedenen Varianzen ist Welchs Modifikation des t-Tests angezeigt. In Fällen *nicht* normal-, aber stetig verteilter Daten sollten wiederum nicht-parametrische (Rang-)Verfahren verwendet werden, namentlich Wilcoxons Rangsummentest auf Lokationsunterschied zwischen den beiden Populationen (und z. B. die – hier nicht diskutierten – Tests von Ansari und von Mood auf Skalenunterschied).

8.5.1 Der Zweistichproben-F-Test für der Vergleich zweier Varianzen

Er ist implementiert in der Funktion `var.test()`, aber auch hier zur Erinnerung zunächst wieder etwas Theorie:

Annahmen: $X_1, \ldots, X_n$ sind unabhängig $\mathcal{N}(\mu_X, \sigma_X^2)$-verteilt und unabhängig davon sind $Y_1, \ldots, Y_m$ unabhängig $\mathcal{N}(\mu_Y, \sigma_Y^2)$-verteilt mit unbekannten Varianzen σ_X^2 und σ_Y^2.

Zu testen:

$$H_0 : \frac{\sigma_X^2}{\sigma_Y^2} = r_0 \qquad \text{gegen} \qquad H_1 : \frac{\sigma_X^2}{\sigma_Y^2} \neq r_0 \qquad \text{zum Signifikanzniveau } \alpha$$

bzw.

$$H_0' : \frac{\sigma_X^2}{\sigma_Y^2} \overset{(\leq)}{\geq} r_0 \qquad \text{gegen} \qquad H_1' : \frac{\sigma_X^2}{\sigma_Y^2} \overset{(>)}{<} r_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

Teststatistik:

$$\frac{1}{r_0}\frac{\hat{\sigma}^2_{Xn}}{\hat{\sigma}^2_{Ym}} \sim F_{n-1,m-1} \qquad \text{unter } H_0 \text{ bzw. unter } \sigma^2_X/\sigma^2_Y = r_0, \text{ dem „Rand“ von } H_0',$$

wobei $\hat{\sigma}^2_{Xn} = (n-1)^{-1}\sum_{i=1}^n (X_i - \bar{X}_n)^2$ der Schätzer für σ^2_X ist und für σ^2_Y analoges gilt.

Entscheidungsregel für konkrete Daten $x_1, \ldots, x_n$ und $y_1, \ldots, y_m$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei mit $q^2_{n,m} := 1/r_0 \cdot s^2_{Xn}/s^2_{Ym}$, dem mit $1/r_0$ skalierten Quotienten der Stichprobenvarianzen der x- bzw. y-Daten, gilt:

$$p\text{-Wert} = \begin{cases} 2 \cdot \min\left\{F_{n-1,m-1}\left(q^2_{n,m}\right), 1 - F_{n-1,m-1}\left(q^2_{n,m}\right)\right\} & \text{für} \quad H_1 : \sigma^2_X/\sigma^2_Y \neq r_0 \\ F_{n-1,m-1}\left(q^2_{n,m}\right) & \text{"} \quad H_1' : \sigma^2_X/\sigma^2_Y < r_0 \\ 1 - F_{n-1,m-1}\left(q^2_{n,m}\right) & \text{"} \quad H_1' : \sigma^2_X/\sigma^2_Y > r_0 \end{cases}$$

worin $F_{s,t}$ die (*nicht* um 0 symmetrische) Verteilungsfunktion der F-Verteilung zu s Zähler- und t Nenner-Freiheitsgraden ist.

Als **Beispiel** verwenden wir Daten von [72, Snedecor & Cochran (1980)] über Gewichtszuwächse in zwei verschiedenen Würfen von Ratten, die unterschiedlich proteinhaltige Nahrung erhielten:

```
> protreich <- c( 134, 146, 104, 119, 124, 161, 107, 83, 113, 129, 97,123)
> protarm <- c( 70, 118, 101, 85, 107, 132, 94)
```

Eine explorative Datenanalyse (nicht gezeigt) lässt an der Normalverteiltheit der Daten nicht zweifeln, sodass der F-Test für der Vergleich der zwei Varianzen durchgeführt werden kann:

F-Test für der Vergleich zweier Varianzen	

`> var.test( protreich, protarm,`
`+ ratio = 1)`
 	
`  F test to compare two variances`
 	
`data:  protreich and protarm`
`F = 1.0755, num df = 11,`
`denom df = 6, p-value = 0.9788`
`alternative hypothesis: true ratio`
`of variances is not equal to 1`
`95 percent confidence interval:`
`  0.198811 4.173718`
`sample estimates:`
`ratio of variances`
`  1.07552`
	`var.test()` führt so den F-Test für $H_0 : \sigma^2_x/\sigma^2_y = r_0$ für zwei als normalverteilt angenommene Populationen aus, wobei ihre zwei ersten Argumente die Datenvektoren erwarten. An `ratio` wird der für σ^2_x/σ^2_y hypothetisierte Quotientenwert r_0 übergeben, zu lesen als H_0 : `ratio` $= r_0$. Die (hier explizit gezeigte) Voreinstellung lautet `ratio = 1` und testet damit also auf Gleichheit der Varianzen. Resultate: Namen der Datenquellen, Wert der Teststatistik (`F`), Freiheitsgrade ihres Zählers und Nenners (`num df`, `denom df`), p-Wert (`p-value`). Außerdem: Alternative (Voreinstellung: Zweiseitig, H_1 : `ratio` $\neq r_0$), 95 %-Konfidenzintervall und Schätzwert für σ^2_x/σ^2_y.
	Mit `conf.level` und `alternative` lassen sich Konfidenzniveau bzw. Alternative spezifizieren: `alternative = "greater"` ist H_1 : `ratio` $> r_0$; `alter = "less"` ist H_1 : `ratio` $< r_0$.

Ergebnis: Die Varianzen unterscheiden sich nicht signifikant; somit kann der (im Folgenden vorgestellte) Zweistichproben-t-Test auf obige Daten angewendet werden, wobei aber bedacht werden sollte, dass in obigem F-Test natürlich ein Fehler 2. Art aufgetreten sein könnte ... und wir ja nur feststellen können, dass wir kein stichhaltiges Argument gegen H_0 gefunden haben (nicht, dass wir H_0 „bewiesen“ haben).

8.5.2 Der Zweistichproben-t-Test bei unbekannten, aber gleichen Varianzen

Annahmen: $X_1, \ldots, X_n$ sind unabhängig $\mathcal{N}(\mu_X, \sigma^2)$-verteilt und unabhängig davon sind $Y_1, \ldots, Y_m$ unabhängig $\mathcal{N}(\mu_Y, \sigma^2)$-verteilt mit unbekanntem, aber gleichem σ^2.

Zu testen:

$$H_0 : \mu_X - \mu_Y = \Delta_0 \qquad \text{gegen} \qquad H_1 : \mu_X - \mu_Y \neq \Delta_0 \qquad \text{zum Signifikanzniveau } \alpha$$

bzw.

$$H_0' : \mu_X - \mu_Y \overset{(\leq)}{\geq} \Delta_0 \qquad \text{gegen} \qquad H_1' : \mu_X - \mu_Y \overset{(>)}{<} \Delta_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

Teststatistik:

$$\frac{\bar{X}_n - \bar{Y}_m - \Delta_0}{\sqrt{\left(\frac{1}{n} + \frac{1}{m}\right) \frac{(n-1)\hat{\sigma}^2_{Xn} + (m-1)\hat{\sigma}^2_{Ym}}{n+m-2}}} \sim t_{n+m-2}$$

unter H_0 bzw. unter $\mu_X - \mu_Y = \Delta_0$, dem „Rand“ von H_0'.

Entscheidungsregel für konkrete Daten $x_1, \ldots, x_n$ und $y_1, \ldots, y_m$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \iff p\text{-Wert} \leq \alpha,$$

wobei mit s^2_{Xn} und s^2_{Ym} als Stichprobenvarianzen der x- bzw. y-Daten und

$$t^*_{n,m} := \frac{\bar{x}_n - \bar{y}_m - \Delta_0}{\sqrt{\left(\frac{1}{n} + \frac{1}{m}\right) \frac{(n-1)s^2_{Xn} + (m-1)s^2_{Ym}}{n+m-2}}}$$

als Realisierung der obigen Teststatistik gilt:

$$p\text{-Wert} = \begin{cases} 2\left(1 - F_{t_{n+m-2}}\left(|t^*_{n,m}|\right)\right) & \text{für} \quad H_1 : \mu_X - \mu_Y \neq \Delta_0 \\ F_{t_{n+m-2}}\left(t^*_{n,m}\right) & \text{"} \quad H_1' : \mu_X - \mu_Y < \Delta_0 \\ 1 - F_{t_{n+m-2}}\left(t^*_{n,m}\right) & \text{"} \quad H_1' : \mu_X - \mu_Y > \Delta_0 \end{cases}$$

wobei F_{t_k} die (um 0 symmetrische) Verteilungsfunktion der t_k-Verteilung ist.

Doch nun die praktische Durchführung von Students Zweistichproben-t-Test (ebenfalls) mit Hilfe der Funktion `t.test()` für die Daten von Snedecor und Cochran:

Students Zweistichproben-t-Test	

`> t.test( protreich, protarm,`
`+ mu = 0, var.equal = TRUE)`
 	
`  Two-Sample t-Test`
 	
`data:  protreich and protarm`
`t = 1.8914, df = 17,`
`p-value = 0.07573`
`alternative hypothesis:`
`true difference in means is not equal to 0`
`95 percent confidence interval:`
`  -2.193679  40.193679`
`sample estimates:`
`  mean of x mean of y`
`  120  101`
	In dieser Form führt `t.test()` Students t-Test auf Erwartungswertegleichheit für zwei mit gleichen Varianzen als normalverteilt angenommene Populationen aus, wobei ihre zwei ersten Argumente die Datenvektoren erwarten. `mu` erhält den für $\mu_X - \mu_Y$ hypothetisierten Differenzwert Δ_0, zu lesen als H_0 : `mu` $= \Delta_0$ (hier explizit gezeigt mit der Voreinstellung `mu = 0`, also für $H_0 : \mu_X - \mu_Y = 0$). Beachte: `var.equal = TRUE` ist *nicht* die Voreinstellung. Resultate: Namen der Datenquellen, Wert der Teststatistik (`t`), ihre Freiheitsgrade (`df`), p-Wert (`p-value`). Außerdem: Alternative (Voreinstellung: Zweiseitig, H_1 : `mu` $\neq \Delta_0$), 95 %-Konfidenzintervall für die Differenz $\mu_X - \mu_Y$ und die Schätzer für beide Erwartungswerte. Konfidenzniveau und Alternative wären mit `conf.level` bzw. `alternative` spezifizierbar: `alternative = "greater"` ist H_1 : `mu` $> \Delta_0$; `alter = "less"` ist H_1 : `mu` $< \Delta_0$.

8.5.3 Die Welch-Modifikation des Zweistichproben-t-Tests

Die bisher im Normalverteilungsmodell betrachteten Ein- und Zweistichprobentests halten das Niveau α exakt ein. Für das Zweistichproben-Lokationsproblem mit unbekannten und möglicherweise *verschiedenen* Varianzen ist kein solcher exakter Test bekannt. Es handelt sich dabei um das sogenannte Behrens-Fisher-Problem, wofür es immerhin einen approximativen Niveau-α-Test gibt: Welchs Modifikation des Zweistichproben-t-Tests. Auch dieser Test ist durch `t.test()` verfügbar und ist sogar die Voreinstellung.

Annahmen: $X_1, \ldots, X_n$ sind unabhängig $\mathcal{N}(\mu_X, \sigma_X^2)$-verteilt und unabhängig davon sind $Y_1, \ldots, Y_m$ unabhängig $\mathcal{N}(\mu_Y, \sigma_Y^2)$-verteilt mit unbekannten μ_X, μ_Y, σ_X^2 und σ_Y^2.

Zu testen:

$$H_0 : \mu_X - \mu_Y = \Delta_0 \qquad \text{gegen} \qquad H_1 : \mu_X - \mu_Y \neq \Delta_0 \qquad \text{zum Signifikanzniveau } \alpha$$

bzw.

$$H_0' : \mu_X - \mu_Y \overset{(\leq)}{\geq} \Delta_0 \qquad \text{gegen} \qquad H_1' : \mu_X - \mu_Y \overset{(>)}{<} \Delta_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

Teststatistik:

$$\frac{\bar{X}_n - \bar{Y}_m - \Delta_0}{\sqrt{\frac{\hat{\sigma}_{Xn}^2}{n} + \frac{\hat{\sigma}_{Ym}^2}{m}}} \overset{approx.}{\sim} t_\nu \qquad \text{mit} \quad \nu = \left\lfloor \frac{\left(\frac{\hat{\sigma}_{Xn}^2}{n} + \frac{\hat{\sigma}_{Ym}^2}{m}\right)^2}{\frac{1}{n-1}\left(\frac{\hat{\sigma}_{Xn}^2}{n}\right)^2 + \frac{1}{m-1}\left(\frac{\hat{\sigma}_{Ym}^2}{m}\right)^2} \right\rfloor$$

unter H_0 bzw. unter $\mu_X - \mu_Y = \Delta_0$, dem „Rand“ von H_0'.

Entscheidungsregel für konkrete Daten $x_1, \ldots, x_n$ und $y_1, \ldots, y_m$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei mit s_{Xn}^2 und s_{Ym}^2 als Stichprobenvarianzen der x- bzw. y-Daten sowie mit $t_{n,m}^*$ und ν^* als Realisierungen der obigen Teststatistik bzw. der (ganzzahligen) Freiheitsgrade nach obiger Formel gilt:

$$p\text{-Wert} = \begin{cases} 2\left(1 - F_{t_{\nu^*}}\left(|t_{n,m}^*|\right)\right) & \text{für} \quad H_1 : \mu_X - \mu_Y \neq \Delta_0 \\ F_{t_{\nu^*}}\left(t_{n,m}^*\right) & \text{"} \quad H_1' : \mu_X - \mu_Y < \Delta_0 \\ 1 - F_{t_{\nu^*}}\left(t_{n,m}^*\right) & \text{"} \quad H_1' : \mu_X - \mu_Y > \Delta_0 \end{cases}$$

wobei F_{t_k} die (um 0 symmetrische) Verteilungsfunktion der t_k-Verteilung ist.

Welch-Modifikation des Zweistichproben-t-Tests

```
> t.test( protreich, protarm)

  Welch Two-Sample t-Test

data: protreich and protarm
t= 1.9107, df= 13.082, p-value= 0.0782
alternative hypothesis: true dif-
ference in means is not equal to 0
95 percent confidence interval:
  -2.469073  40.469073
sample estimates:
 mean of x mean of y
       120       101
```

Aufgrund der (nicht explizit gezeigten) Voreinstellung `var.equal = FALSE` und `mu = 0` führt `t.test()` so Welchs t-Test auf Erwartungswertegleichheit für zwei mit möglicherweise ungleichen Varianzen als normalverteilt angenommenen Populationen durch (d. h., den Test für H_0 : `mu` $= 0$, also $H_0 : \mu_X - \mu_Y = \Delta_0$ mit $\Delta_0 = 0$).
Alles Weitere inklusive der Resultate: Analog zum Zweistichproben-t-Test für gleiche Varianzen des vorherigen §8.5.2. Allerdings wird die Anzahl der Freiheitsgrade (`df`) *nicht* abgerundet angegeben, d. h., der Wert ist das Argument von $\lfloor\cdot\rfloor$ in der obigen Definition von ν .

8.5.4 Wilcoxons Rangsummentest (Mann-Whitney U-Test)

Wäre die Normalverteilungsannahme für obige Daten nicht gerechtfertigt, wohl aber die Annahme zweier stetiger und „typgleicher“, aber sonst beliebiger Verteilungen, käme als nichtparametrisches Pendant Wilcoxons Rangsummentest infrage (der äquivalent zum Mann-Whitney U-Test ist; siehe auch die zweite Bemerkung am Ende dieses Abschnitts (Seite 132)). Dieser Test wird durch `wilcox.test()` zur Verfügung gestellt.

Annahmen: $X_1, \ldots, X_n$ sind unabhängig und nach einer stetigen Verteilungsfunktion F verteilt und unabhängig davon sind $Y_1, \ldots, Y_m$ unabhängig nach der (typgleichen!) Verteilungsfunktion $F(\cdot - \theta)$ verteilt (d. h., die Y_j können sich als aus F stammende $\tilde{X}_j$s vorgestellt werden, die um θ zu $Y_j := \tilde{X}_j + \theta$ geshiftet wurden; θ ist also nicht notwendigerweise ein Median o. Ä., sondern lediglich der Unterschied in der Lage der beiden Verteilungen zueinander).

Zu testen:

$$H_0 : \theta = \theta_0 \qquad \text{gegen} \qquad H_1 : \theta \neq \theta_0 \qquad \text{zum Signifikanzniveau } \alpha$$

bzw.

$$H_0' : \theta \overset{(\leq)}{\geq} \theta_0 \qquad \text{gegen} \qquad H_1' : \theta \overset{(>)}{<} \theta_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

Teststatistik:

$$W_{n,m} := \sum_{i=1}^{n} R(X_i + \theta_0) - \frac{n(n+1)}{2},$$

wobei $R(X_i + \theta_0)$ der *Rang* von $X_i + \theta_0$ unter den $N := n + m$ „gepoolten“ Werten $X_1 + \theta_0, \ldots, X_n + \theta_0, Y_1, \ldots, Y_m$ ist. Die Verteilung von Wilcoxons Rangsummenstatistik $W_{n,m}$ ist unter $\theta = \theta_0$ bekannt. Kombinatorische Überlegungen zeigen, dass

$$\mathbb{E}[W_{n,m}] = \frac{nm}{2} \qquad \text{und} \qquad \mathrm{Var}(W_{n,m}) = \frac{nm(N+1)}{12} \qquad \text{unter } \theta = \theta_0$$

ist und dass für $W_{n,m}' := \sum_{j=1}^{m} R(Y_j) - m(m+1)/2$ gilt $W_{n,m} + W_{n,m}' = nm$. Des Weiteren ist $W_{n,m}$ unter $\theta = \theta_0$ asymptotisch normalverteilt:

$$Z_{n,m} := \frac{W_{n,m} - \mathbb{E}[W_{n,m}]}{\sqrt{\mathrm{Var}(W_{n,m})}} \overset{n,m\to\infty}{\longrightarrow} \mathcal{N}(0,1) \quad \text{unter } \theta = \theta_0, \text{ falls } \frac{n}{m} \to \lambda \notin \{0, \infty\}$$

Treten Bindungen zwischen den $X_i + \theta_0$ und Y_j auf, so wird die Methode der Durchschnittsränge verwendet, was den Erwartungswert von $W_{n,m}$ nicht, wohl aber ihre Varianz beeinflusst. In diesem Fall wird in `wilcox.test()` per Voreinstellung die Normalapproximation verwendet, aber mit einer im Nenner leicht modifizierten Statistik $Z_{n,m}^{mod}$ anstelle von $Z_{n,m}$ (worauf wir hier jedoch nicht näher eingehen). $Z_{n,m}^{mod}$ hat dieselbe Asymptotik wie $Z_{n,m}$.

Entscheidungsregel für konkrete Daten $x_1, \ldots, x_n$ und $y_1, \ldots, y_m$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei für die Berechnung des p-Wertes die folgende Fallunterscheidung zur Anwendung kommt, in welcher $w_{n,m}$ der realisierte Wert von $W_{n,m}$ sei:

1. Keine Bindungen und „kleine“ Stichprobenumfänge (in `wilcox.test()` $n < 50$ und $m < 50$):

$$p\text{-Wert} = \begin{cases} 2F_{n,m}(w_{n,m}) & \text{für} \quad H_1 : \theta \neq \theta_0, \text{wenn } w_{n,m} \leq nm/2 \\ 2(1 - F_{n,m}(w_{n,m} - 1)) & \text{"} \quad H_1 : \theta \neq \theta_0, \text{wenn } w_{n,m} > nm/2 \\ F_{n,m}(w_{n,m}) & \text{"} \quad H_1' : \theta < \theta_0 \\ 1 - F_{n,m}(w_{n,m} - 1) & \text{"} \quad H_1' : \theta > \theta_0 \end{cases}$$

   wobei $F_{n,m}$ die um $nm/2$ symmetrische Verteilungsfunktion der diskreten (!) Verteilung von $W_{n,m}$ unter $\theta = \theta_0$ (Wilcoxons Rangsummenverteilung zu den Stichprobenumfängen n und m) ist.

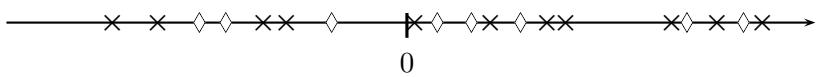
2. Bindungen oder „große“ Stichprobenumfänge (in `wilcox.test()` $n \geq 50$ oder $m \geq 50$):

   Hier wird $Z_{n,m}^{mod}$ anstelle von $Z_{n,m}$ verwendet und außerdem eine Stetigkeitskorrektur (so wie bei Wilcoxons Vorzeichen-Rangsummentest auf Seite 125, nur mit dem hiesigen $W_{n,m}$ an Stelle des dortigen W_n^+), um die Asymptotik von $Z_{n,m}^{mod}$ zu verbessern. Wenn nun $z_{n,m}^{mod}$ die konkrete Realisierung von $Z_{n,m}^{mod}$ bezeichne, dann ist

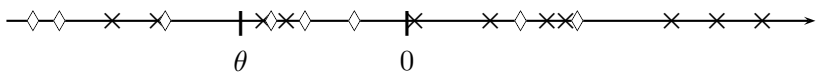
$$p\text{-Wert} = \begin{cases} 2\left(1 - \Phi\left(\left|z_{n,m}^{mod}\right|\right)\right) & \text{für} \quad H_1 : \theta \neq \theta_0 \\ \Phi\left(z_{n,m}^{mod}\right) & \text{"} \quad H_1' : \theta < \theta_0 \\ 1 - \Phi\left(z_{n,m}^{mod}\right) & \text{"} \quad H_1' : \theta > \theta_0 \end{cases}$$

Bemerkung: Ist $\theta > 0$, so ist die Verteilungsfunktion $F(\cdot - \theta)$ von Y gegenüber der Verteilungsfunktion F von X nach rechts verschoben, denn es ist $F(x - \theta) \leq F(x)$ für alle $x \in \mathbb{R}$. Man sagt dann, Y ist <u>stochastisch größer</u> als X (kurz: $Y \geq_{\mathbb{P}} X$). Analog ist im Fall $\theta < 0$ die Verteilungsfunktion $F(\cdot - \theta)$ gegenüber F nach links verschoben und man nennt Y <u>stochastisch kleiner</u> als X (kurz: $Y \leq_{\mathbb{P}} X$). Schließlich bedeutet $\theta \neq 0$, dass $F(\cdot - \theta)$ gegenüber F entweder nach rechts oder nach links verschoben ist, d. h., dass entweder $Y \geq_{\mathbb{P}} X$ oder $Y \leq_{\mathbb{P}} X$ gilt.

Zur heuristischen Motivation der Funktionsweise von Wilcoxons Rangsummenstatistik hier für den Fall $\theta_0 = 0$ zwei „prototypische“ Datenszenarien: Es seien X_i i.i.d. $\sim F$ (markiert durch „$\times$“) und unabhängig davon Y_j i.i.d. $\sim F(\cdot - \theta)$ (markiert durch „$\Diamond$“).

$\theta = 0$: 

$\Rightarrow W_{n,m}$ „mittel“

$\theta \neq 0$: 

$\Rightarrow W_{n,m}$ „groß“

Begründung: $\theta < 0 \;\Rightarrow\; F(x - \theta) \geq F(x) \;\Rightarrow\; Y \leq_{\mathbb{P}} X \;\Rightarrow$ die Y_j haben tendenziell kleinere Ränge als die $X_i \;\Rightarrow\; W_{n,m}$ tendenziell „groß“.

$W_{n,m}$ misst, wie gut die $(X_i + \theta_0)$- und die Y_j-Daten durchmischt sind: Je schlechter die Durchmischung, desto kleiner oder größer ist $W_{n,m}$, wobei kleine Werte auftreten, wenn die $(X_i + \theta_0)$-Gruppe gegenüber der Y_j-Gruppe tendenziell nach links verschoben ist (und somit die $(X_i + \theta_0)$-Ränge tendenziell kleiner als die Y_j-Ränge sind). Große Werte treten im umgekehrten Fall auf (wie im unteren Szenario der vorstehenden Skizze; Memo: Dort ist $\theta_0 = 0$).

Auf der nächsten Seite wird die Umsetzung des Tests durch die Funktion `wilcox.test()` an einem Beispiel erläutert. Aus Platzgründen ziehen wir weitere Bemerkungen hierher vor:

Weitere Bemerkungen:

- Es spielt keine Rolle, welche Stichprobe das erste Argument von `wilcox.test()` ist (sofern man selbst auf das korrekte Vorzeichen beim hypothetisierten θ_0 achtet!), da für $W_{n,m}$ und $W'_{n,m}$ (die Summenstatistik für die Ränge der $Y_1, \ldots, Y_m$) stets $W_{n,m} + W'_{n,m} = nm$ gilt, sodass Wilcoxons Rangsummentest für $W_{n,m}$ und für $W'_{n,m}$ äquivalent ist.

- Faktisch ist in `wilcox.test()` der zu Wilcoxons Rangsummentest äquivalente U-Test von Mann und Whitney realisiert. Tatsächlich ist die Mann-Whitney-Teststatistik U genau so definiert wie die hiesige, angebliche Wilcoxon Rangsummenstatistik $W_{n,m}$. Bei der eigentlichen Wilcoxon Rangsummenstatistik $W := \sum_{i=1}^{n} R(X_i + \theta_0)$ wird nämlich nicht $n(n+1)/2$ abgezogen. Aber damit unterscheiden sich W und U $(= W_{n,m})$ nur durch die Konstante $n(n+1)/2$, sodass beide Verfahren äquivalent sind.

- Mit dem nicht gezeigten Argument `conf.int = TRUE` erhielte man den Hodges–Lehmann–Schätzer für den Shift θ, ergänzt um ein Konfidenzintervall (das ein exaktes ist, wenn auch der p-Wert exakt ist, und ansonsten ein approximatives). (Siehe die Online-Hilfe und für mathematische Details siehe z. B. [48, Hollander & Wolfe (1973), Abschnitte 4.2 und 4.3].)
- (Wörtl. Wiederholung des 2. Hinweises auf S. 126 unten.) Das **R**-Paket `exactRankTests`, das allerdings "no longer under development" ist, bietet durch seine Funktion `wilcox.exact()` auch für Stichproben *mit* Bindungen die Berechnung exakter p-Werte des Wilcoxon-Rangsummentests. Stattdessen und auch unabhängig davon gibt es das **R**-Paket `coin`, dessen Funktion `wilcox_test()` exakte und approximative *bedingte* p-Werte der Wilcoxon-Tests liefert.

Wilcoxons Rangsummentest	

`> wilcox.test( protreich, protarm)`
 	
`  Wilcoxon rank sum test with`
`  continuity correction`
 	
`data:  protreich and protarm`
`W = 62.5, p-value = 0.09083`
`alternative hypothesis: true mu`
`is not equal to 0`
 	
`Warning message:`
`In wilcox.test.default(protreich,`
`protarm) : cannot compute exact`
`  p-value with ties`
	So führt `wilcox.test()` Wilcoxons Rangsummentest auf einen Lokations-Shift zwischen den als stetig und gleichartig angenommenen Verteilungen zweier Populationen durch, wobei die zwei ersten Argumente die Datenvektoren erhalten. Das nicht gezeigte Argument `mu` erwartet den hypothetisierten Shiftwert θ_0 und ist per Voreinstellung auf 0 gesetzt (also wird hier H_0 : `mu` $= \theta_0$ mit $\theta_0 = 0$ getestet). Resultate: Die Datenquellen, der errechnete Wert der Teststatistik $W_{n,m}$ (`W`) und ein p-Wert (`p-value`) sowie die Alternative (Voreinstellung: Zweiseitig, H_1 : `mu` $\neq \theta_0$; "`true mu`" bezieht sich also auf θ). Jedoch wurde aufgrund einer Bindung zwischen den beiden Stichproben für die p-Wert-Berechnung die approximativ normalverteilte Teststatistik $Z_{n,m}^{mod}$ (`with continuity correction`) verwendet und eine entsprechende `Warning message` ausgegeben.
	Die Alternative lässt sich auch hier mit `alternative` spezifizieren: `alternative= "greater"` heißt H_1 : `mu` $> \theta_0$ und `alter= "less"` ist H_1 : `mu` $< \theta_0$.

`> protarm[ 5] <- 107.5`
`> wilcox.test( protreich, protarm)`
 	
`  Wilcoxon rank sum test`
 	
`data:  protreich and protarm`
`W = 62, p-value = 0.1003`
`alternative hypothesis: true mu`
`is not equal to 0`
	Nach dem künstlichen Aufbrechen der Bindung (zu Demonstrationszwecken) erhalten wir die Ausgabe des exakten Wilcoxon-Rangsummentests: Wert von $W_{n,m}$ (`W`) und exakter p-Wert (`p-value`). Alles Weitere ist wie oben.

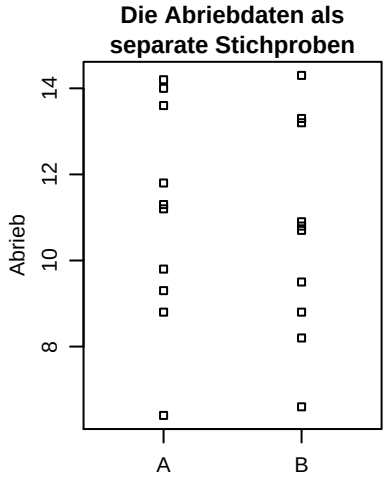
8.6 Das Zweistichproben-Lokationsproblem für verbundene Stichproben

Häufig werden Zweistichprobendaten im Rahmen von sogenannten *vergleichenden Studien* erhoben. Eine derartige Studie hat zum Ziel, einen *Unterschied* zwischen zwei Behandlungen A und B zu entdecken (und weniger, ihre jeweiligen Wirkungen zu quantifizieren). Bestehen zwischen den zu betrachtenden Untersuchungseinheiten jedoch bereits aufgrund individueller Gegebenheiten starke, behandlungsunabhängige Unterschiede, wie z. B. große, behandlungsunabhängige Gewichts- oder Altersunterschiede, so kann dies zu einer sehr hohen *interindividuellen Variabilität* in den Beobachtungen (d. h. zu hohen Varianzen in den A- bzw. B-Daten) führen. Hierdurch wird leicht ein Unterschied zwischen den mittleren Wirkungen der zu vergleichenden Behandlungen „verschleiert“ (*maskiert*).

Diesem Effekt versucht man entgegenzuwirken, indem man (vor der Anwendung der Behandlungen) Paare von möglichst ähnlichen Untersuchungseinheiten bildet und die Behandlungen A und B jeweils auf genau eine der beiden Einheiten pro Paar anwendet. Dabei geschieht die jeweilige Zuordnung von Untersuchungseinheit zu Behandlung „innerhalb“ eines Paares per Randomisierung. Sofern gewährleistet ist, dass sich die beiden Behandlungen gegenseitig nicht beeinflussen, können im Idealfall sogar beide an denselben Untersuchungseinheiten durchgeführt werden, d. h., man bildet „Paare“ aus *identischen* Einheiten, sodass jede Untersuchungseinheit gewissermaßen als ihre eigene Kontrolle fungiert. Man erhält in jedem Fall „verbundene“ (oder „gepaarte“, Engl.: “paired” oder “matched”) Beobachtungen (X_i, Y_i) für jedes Untersuchungseinheitenpaar. (Natürlich sind X_i und Y_i dadurch nicht mehr unabhängig, wohl aber nach wie vor alle Paare (X_i, Y_i) und (X_j, Y_j) mit $i \neq j$.) Schließlich werden die Behandlungs*unterschiede* zwischen den beiden Untersuchungseinheiten jedes Paares, also $D_i := X_i - Y_i$ gebildet und deren Verteilung analysiert.

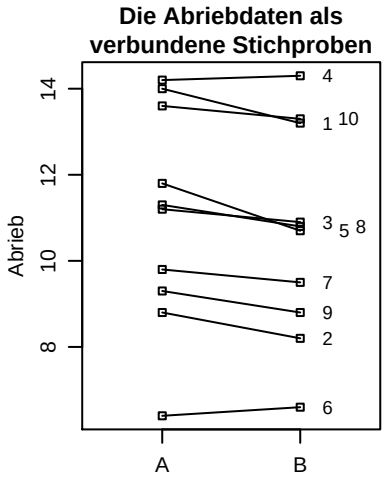
Das folgende **Beispiel** soll den Sachverhalt näher erläutern: In einem Experiment (beschrieben in [9, Box, Hunter und Hunter (1978)]) wurde der Abrieb bei Schuhsohlen für zwei verschiedene Materialien A und B untersucht: Dazu wurde für zehn Paar Schuhe jeweils ein Schuh mit einer Sohle aus Material A und der andere mit einer aus Material B versehen. Welches Material für welchen Schuh eines jeden Paares verwendet wurde, war durch Randomisierung bestimmt worden, wobei mit gleicher Wahrscheinlichkeit Material A auf den linken oder den rechten Schuh kam. Damit sollte ein möglicher, eventuell aus dem menschlichen Gehverhalten resultierender Effekt der Seiten ausgeschaltet werden.

Zehn Jungen trugen jeweils eines der Schuhpaare für eine gewisse Zeit und im Anschluss daran wurden die Abriebe an den Schuhen der Jungen gemessen. Die Frage war, ob die beiden Sohlenmaterialien denselben Abrieb hatten. Der Strip Chart rechts zeigt die beobachteten Abriebwerte für jede Materialsorte. Aufgrund der Unterschiedlichkeit der Jungen (hinsichtlich Gewicht, Laufgewohnheiten u. Ä.) ist sowohl *innerhalb* der Abriebe $X_1, \ldots, X_n$ für Material A als auch *innerhalb* der Abriebe $Y_1, \ldots, Y_n$ für B eine so hohe Variabilität vorhanden, dass ein möglicher Unterschied *zwischen* den mittleren Abrieben der beiden Materialien nicht zu erkennen ist; er wird maskiert.



Dieses Problem *könnte* als übliches Zweistichprobenproblem aufgefasst werden. Im vorliegenden Fall sind die Beobachtungen jedoch verbunden, da jeder Junge jedes Sohlenmaterial getragen hat. (Hier sind die Untersuchungseinheiten, auf die die verschiedenen „Behandlungen“ angewendet wurden, nicht nur ähnliche Individuen oder Objekte, sondern in der Tat identisch!)

Der Strip Chart rechts veranschaulicht durch Linien die verbundenen Beobachtungen, die jeweils zu einem Jungen gehören. (Die Zahlen dienen nur der leichteren Identifizierung der einzelnen Werte.) Dadurch wird sichtbar, dass B tendenziell niedrigere Abriebwerte als A hat. Damit die Verbundenheit adäquat modelliert und ein Unterschied zwischen A und B statistisch auch eher erkannt wird, sollte ein Test für verbundene Stichproben angewendet werden.



8.6.1 Die Zweistichproben-t-Tests bei verbundenen Stichproben

Für die beobachteten Paare $(X_1, Y_1), \ldots, (X_n, Y_n)$ sei von Interesse, ob die Erwartungswerte μ_X der X_i und μ_Y der Y_i gleich sind, bzw. ob sie in einer speziellen Größenbeziehung zueinander stehen. Es geht also wieder um die beiden Testprobleme:

$$H_0 : \mu_X = \mu_Y \quad \text{gegen} \quad H_1 : \mu_X \neq \mu_Y \qquad \text{bzw.} \qquad H_0' : \mu_X \overset{(\leq)}{\geq} \mu_Y \quad \text{gegen} \quad H_1' : \mu_X \overset{(>)}{<} \mu_Y$$

Sie sind äquivalent zu den Testproblemen

$$H_0 : \mu_X - \mu_Y = 0 \quad \text{gegen} \quad H_1 : \mu_X - \mu_Y \neq 0$$

bzw.

$$H_0' : \mu_X - \mu_Y \overset{(\leq)}{\geq} 0 \quad \text{gegen} \quad H_1' : \mu_X - \mu_Y \overset{(>)}{<} 0$$

Diese Umformulierung der Testprobleme und die Tatsache, dass wir an der Entdeckung eines *Unterschiedes* zwischen den Erwartungswerten der Behandlungen (und nicht an den eigentlichen Erwartungswerten) interessiert sind, legt nahe, die <u>Paardifferenzen</u> $D_i = X_i - Y_i$ zu betrachten. Diese haben natürlich den Erwartungswert $\mu_D = \mu_X - \mu_Y$, sodass obige Hypothesen äquivalent zu Hypothesen über μ_D sind. Zwar ist die Unabhängigkeit zwischen den Zufallsvariablen X_i und Y_i, die an ein und derselben Untersuchungseinheit (bzw. an den sehr ähnlichen Untersuchungseinheiten eines Paares) beobachtet wurden, nicht mehr gewährleistet, aber die D_i sind unabhängig, wenn die Untersuchungseinheiten (bzw. -paare) und somit die (X_i, Y_i) als voneinander unabhängig erachtet werden können. Unter der Annahme, dass die Differenzen D_i normalverteilt sind (was übrigens eine schwächere Voraussetzung ist, als die Forderung, dass die X_i und die Y_i normalverteilt sind), können nun die Einstichproben-t-Tests von §8.4.1 auf diese Hypothesen übertragen werden:

<u>Annahmen:</u> $(X_1, Y_1), \ldots, (X_n, Y_n)$ sind unabhängig und identisch verteilt mit marginalen Erwartungswerten μ_X und μ_Y. Die Differenzen $D_i := X_i - Y_i$ für $i = 1, \ldots, n$ seien $\mathcal{N}(\mu_D, \sigma_D^2)$-verteilt mit unbekanntem $\mu_D = \mu_X - \mu_Y$ und unbekanntem σ_D^2.

<u>Zu testen:</u>

$$H_0 : \mu_D = \mu_0 \qquad \text{gegen} \qquad H_1 : \mu_D \neq \mu_0 \qquad \text{zum Signifikanzniveau } \alpha$$

bzw.

$$H_0' : \mu_D \overset{(\leq)}{\geq} \mu_0 \qquad \text{gegen} \qquad H_1' : \mu_D \overset{(>)}{<} \mu_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

Teststatistik:

$$\frac{\bar{D}_n - \mu_0}{\hat{\sigma}_D/\sqrt{n}} \sim t_{n-1} \qquad \text{unter } H_0 \text{ bzw. unter } \mu_D = \mu_0, \text{ dem „Rand“ von } H_0',$$

wobei $\hat{\sigma}_D$ der Schätzer für σ_D und $\bar{D}_n$ das arithmetische Mittel der D_i sind.

Entscheidungsregel für konkrete Daten $x_1, \ldots, x_n$, $y_1, \ldots, y_n$ und $d_i := x_i - y_i$ für $i = 1, \ldots, n$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei mit $\bar{d}_n$ und s_n als arithmetischem Mittel bzw. als Stichprobenstandardabweichung der d_i sowie $t_n^* := (\bar{d}_n - \mu_0)/(s_n/\sqrt{n})$ als Realisierung der obigen Teststatistik gilt:

$$p\text{-Wert} = \begin{cases} 2\left(1 - F_{t_{n-1}}\left(|t_n^*|\right)\right) & \text{für} \quad H_1 : \mu_D \neq 0 \\ F_{t_{n-1}}\left(t_n^*\right) & \text{"} \quad H_1' : \mu_D < 0 \\ 1 - F_{t_{n-1}}\left(t_n^*\right) & \text{"} \quad H_1' : \mu_D > 0 \end{cases}$$

worin F_{t_k} die (um 0 symmetrische) Verteilungsfunktion der t_k-Verteilung ist.

Wir setzen das oben begonnene **Beispiel** fort: Unten folgen die Datenvektoren mit den gemessenen Werten für den Abrieb einer jeden Materialsorte für jeden Jungen (Komponente i enthält jeweils den Wert für Junge i). Zunächst sollte auch hier eine explorative Datenanalyse durchgeführt werden, um zu beurteilen, ob die Testvoraussetzungen erfüllt sind. Diese betreffen *nicht* die einzelnen Beobachtungen $x_1, \ldots, x_n$ und $y_1, \ldots, y_n$ der beiden Stichproben, sondern die Differenzen $d_i = x_i - y_i$ der gepaarten Beobachtungen. Und natürlich handelt es sich dabei um die Annahmen der Normalverteiltheit und der Unabhängigkeit der d_i:

```
> abrieb.A <- c( 14.0, 8.8, 11.2, 14.2, 11.8, 6.4, 9.8, 11.3, 9.3, 13.6)
> abrieb.B <- c( 13.2, 8.2, 10.9, 14.3, 10.7, 6.6, 9.5, 10.8, 8.8, 13.3)
> eda( abrieb.A - abrieb.B)
```

(`eda()`-Resultat nicht gezeigt; Verteilungsannahmen für die Differenzen scheinen aber erfüllt.)

Students t-Test für verbundene Stichproben	

`> t.test( abrieb.A, abrieb.B,`
`+ paired = TRUE)`
 	
`Paired t-test`
 	
`data: abrieb.A and abrieb.B`
`t = 3.3489, df = 9, p-value = 0.008539`
`alternative hypothesis: true difference in means is not equal to 0`
`95 percent confidence interval:`
`0.1330461 0.6869539`
`sample estimates:`
`mean of the differences`
`0.41`
	Students Zweistichproben-t-Test für verbundene Stichproben für den Erwartungswert der als normalverteilt angenommenen Population der Differenzen. Das nicht gezeigte Argument `mu` (mit seiner Voreinstellung `mu` $= 0$) erhält den für μ_D hypothetisierten Wert μ_0 zugewiesen, zu lesen als $H_0 :$ `mu` $= \mu_0$ gegen $H_1 :$ `mu` $\neq \mu_0$ (hier also als $H_0 : \mu_D = 0$ gegen $H_1 : \mu_D \neq 0$).
	Alles Weitere inklusive der Resultate analog zum Zweistichproben-t-Test für *un*verbundene Stichproben in §8.5.2 (bis auf die Angabe von natürlich nur einem `sample estimate`).

Bemerkungen:

1. Students Zweistichproben-t-Test für **un**verbundene Stichproben liefert für dieselben Daten einen p-Wert von 0.71! Ein Vergleich des Boxplots der x_i (Abrieb-Daten für A) mit dem der y_i

(Abrieb-Daten für B) einerseits und ein Boxplot der Differenzen d_i andererseits veranschaulicht deutlich die Varianzreduktion durch Paarbildung (Plots nicht gezeigt). Diese hat sich ja bereits in der Darstellung der Paardaten auf Seite 134 „angekündigt“.

2. „Semi-quantitative“ Diskussion der Ursachen: Wenn die Variabilität der Differenzen zwischen den gepaarten Beobachtungen kleiner ist als die Summe der Variabilitäten innerhalb der separaten Stichproben, sind diese Tests für verbundene Stichproben „empfindlicher“ als die Versionen für den unverbundenen Fall. Man sagt, sie haben eine größere Güte (“power”) als die Tests für unverbundene Stichproben. (Siehe hierzu auch den Abschnitt 8.12.)

   Ein Vergleich der in den beiden Zweistichproben-t-Tests auftauchenden empirischen Varianzen erhellt den Sachverhalt: U_n bezeichne die Teststatistik des t-Tests für zwei unverbundene Stichproben gleichen Umfangs (vgl. §8.5.2) und V_n diejenige des t-Tests für zwei verbundene Stichproben. Für die empirische Varianz im Nenner von V_n gilt (wie man leicht nachrechnet):

$$\hat{\sigma}_D^2 \;=\; \hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - \underbrace{2 \cdot \hat{\sigma}_X \cdot \hat{\sigma}_Y}_{>\,0 \text{ stets}} \cdot \hat{\rho}_{XY} \;<\; \hat{\sigma}_X^2 + \hat{\sigma}_Y^2, \quad \text{falls } \hat{\rho}_{XY} > 0$$

   Die rechte Seite in der obigen Ungleichung ist offenbar die empirische Varianz im Nenner von U_n und außerdem halten wir fest, dass die Zähler von U_n und V_n gleich sind. Wird nun – inkorrekterweise – U_n anstelle von V_n verwendet, so ist im Fall $\hat{\rho}_{XY} > 0$ die Teststatistik $|U_n|$ „zu klein“ im Vergleich zur korrekten Teststatistik $|V_n|$, da der Zähler $|\bar{X}_n - \bar{Y}_n|$ durch eine „zu große“ empirische Varianz geteilt wird.

   Zwar sind die Quantile der t-Verteilung mit $2(n-1)$ Freiheitsgraden auch kleiner als diejenigen der t-Verteilung mit $n-1$ Freiheitsgraden, aber dies kompensiert meist *nicht* die inkorrekte Normierung durch eine zu kleine empirische Varianz. Und außerdem ist wegen der Verletzung der Unabhängigkeitsannahme im verbundenen Fall die Teststatistik U_n nicht $t_{2(n-1)}$-verteilt, weswegen man also einen falschen p-Wert erhielte oder ein ungeeignetes Quantil verwenden würde.

8.6.2 Wilcoxons Vorzeichen-Rangsummentest für verbundene Stichproben

Obwohl die Abriebdaten die (Normal-)Verteilungsannahme zu erfüllen scheinen, demonstrieren wir an ihnen auch ein nicht-parametrisches Pendant des t-Tests für verbundene Stichproben, nämlich Wilcoxons Vorzeichen-Rangsummentest für verbundene Stichproben (verfügbar ebenfalls durch `wilcox.test()`):

Annahmen: $(X_1, Y_1), \ldots, (X_n, Y_n)$ sind unabhängig und identisch verteilt. Die Differenzen $D_i := X_i - Y_i$ für $i = 1, \ldots, n$ seien nach einer stetigen Verteilungsfunktion $F(\cdot - \theta)$ verteilt, wobei F **symmetrisch** um 0 ist und 0 ihr eindeutig bestimmter Median ist (d. h., $F(\cdot - \theta)$ ist symmetrisch um θ und θ ist ihr eindeutig bestimmter Median sowie Erwartungswert, falls letzterer existiert).

Zu testen:

$$H_0 : \theta = \theta_0 \qquad \text{gegen} \qquad H_1 : \theta \neq \theta_0 \qquad \text{zum Signifikanzniveau } \alpha$$

bzw.

$$H_0' : \theta \overset{(\leq)}{\geq} \theta_0 \qquad \text{gegen} \qquad H_1' : \theta \overset{(>)}{<} \theta_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

Teststatistik:

$$W_n^+ := \sum_{i=1}^{n} 1_{\{D_i - \theta_0 > 0\}} R(|D_i - \theta_0|),$$

deren Notation und Verteilungseigenschaften schon auf Seite 124 in §8.4.2 aufgeführt wurden. Wir verzichten daher an dieser Stelle auf Ihre erneute Auflistung.

Entscheidungsregel für konkrete Daten $d_i := x_i - y_i$ für $i = 1, \ldots, n$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei für die Berechnung des p-Wertes dieselbe Fallunterscheidung zur Anwendung kommt, wie auf Seite 125 in §8.4.2, nur dass w_n der zu $d_1, \ldots, d_n$ realisierte Wert von W_n^+ ist. Wir verzichten hier daher auf die langweilige, da langwierige Fallunterscheidung und verweisen auf jene Seite.

Beachte:

- Obige Hypothese $H_0 : \theta = \theta_0$ ist *nicht* äquivalent zu der Hypothese $\theta_X - \theta_Y = \theta_0$, wenn wir mit θ_X und θ_Y die Mediane der (Marginal-)Verteilungen der X_i bzw. Y_i bezeichnen, denn im Allgemeinen ist der Median einer Differenz nicht gleich der Differenz der Mediane: $F_{X-Y}^{-1}(1/2) \neq F_X^{-1}(1/2) - F_Y^{-1}(1/2)$. Insbesondere bedeutet dies, dass der Test von $H_0 : \theta = 0$ im Allgemeinen *nicht* äquivalent zum Test auf Gleichheit der Mediane θ_X und θ_Y ist.

  Sind die (Marginal-)Verteilungen von X und Y jedoch symmetrisch (um ihre Mediane θ_X bzw. θ_Y), so ist auch die Verteilung ihrer Differenz $X - Y$ symmetrisch (um ihren Median $\theta_{X-Y} = \theta_X - \theta_Y$)!

- Wie auch schon in §8.4.2 erwähnt, ist die Verteilung von W_n^+ unter H_0 sehr empfindlich gegenüber Abweichungen von der Symmetrieannahme für D (vgl. [14, Brunner & Langer (1999)], §7.1.2).

- Die Forderung nach Symmetrie der Differenzverteilung ist allerdings keine starke Einschränkung, denn die Situation „Kein Unterschied zwischen Gruppe 1 (z. B. 'Behandlung') und Gruppe 2 (z. B. 'Kontrolle')“ bedeutet für die gemeinsame Verteilung H von (X, Y), dass ihre Argumente vertauschbar sind: $H(x, y) = H(y, x)$ für alle $x, y, \in \mathbb{R}$. Hieraus folgt Verteilungsgleichheit von X und Y, woraus sich wiederum die Symmetrie (um 0) der Verteilung der Differenz $X - Y$ ergibt. (Vgl. Example 2.1.1, p. 30 und Exercise 2.10.3, p. 123 in [44, Hettmansperger (1984)] und Comment 2, p. 30 in [48, Hollander & Wolfe (1973)].)

Wilcoxons Vorzeichen-Rangsummentest für verbundene Stichproben	

`> wilcox.test( abrieb.A, abrieb.B,`
`+ paired = TRUE)`
 	
`  Wilcoxon signed rank test with`
`  continuity correction`
 	
`data:  abrieb.A and abrieb.B`
`V = 52, p-value = 0.01431`
`alternative hypothesis: true mu is`
`not equal to 0`
 	
`Warning message:`
`In wilcox.test.default(abrieb.A,`
`abrieb.B, paired = TRUE) :`
` cannot compute exact p-value with`
` ties`
	Wilcoxons Vorzeichen-Rangsummentest für verbundene Stichproben für den Median θ der stetigen und um θ symmetrischen Verteilung der Population der Differenzen. Das nicht gezeigte Argument `mu` (mit seiner Voreinstellung `mu` $= 0$) erhält den für θ hypothetisierten Wert θ_0 zugewiesen, zu lesen als $H_0 :$ `mu` $= \theta_0$ gegen $H_1 :$ `mu` $\neq \theta_0$ (hier also $H_0 : \theta = 0$ gg. $H_1 : \theta \neq 0$).
	Resultate: Wert der Teststatistik W_n^+ (`V`), p-Wert (`p-value`; wegen Bindungen in den Differenzen über Z_n^{mod} berechnet, per Voreinstellung `with continuity correction`), Alternative (Voreinstellung: zweiseitig, $H_1 :$ `mu` $\neq \theta_0$; “`true mu`” meint θ).
	Wegen der Bindungen erfolgt die entsprechende `Warning message`.

`> abrieb.B[ c( 7, 9, 10)] <- c( 9.55,`
`+ 8.85, 13.25)`
`> wilcox.test( abrieb.A, abrieb.B,`
`+ paired = TRUE)`
 	
`  Wilcoxon signed rank test`
 	
`data:  abrieb.A and abrieb.B`
`V = 52, p-value = 0.009766`
`alternative hypothesis: true mu is`
`not equal to 0`
	Nach dem künstlichen Aufbrechen der Bindungen: Ausgabe der Ergebnisse des exakten Wilcoxon-Vorzeichen-Rangsummentests für verbundene Stichproben.
	
	Die Alternative lässt sich wie stets wieder mit `alternative` spezifizieren: `alternative = "greater"`: H_1 : `mu` $> \theta_0$, `alter = "less"`: H_1 : `mu` $< \theta_0$.

Hinweise:

- Hier gelten dieselben Hinweise wie auf Seite 126 (am Ende von §8.4.2).
- Die höchst empfehlenswerten, weil äußerst allgemeinen und leistungsfähigen, aber dennoch relativ einfachen Verfahren für die nichtparametrische Analyse longitudinaler Daten in [14, Brunner & Langer (1999)] oder [13, Brunner, Domhof & Langer (2002)] stehen im R-Paket `nparLD` zur Verfügung.

8.7 Tests auf Unabhängigkeit

Im Problem der beiden vorherigen Abschnitte sind die verbundenen Beobachtungen X und Y durch die Anlage des Experiments oder der Untersuchung *nicht* unabhängig voneinander. Es gibt jedoch Situationen, in denen nicht klar ist, ob Unabhängigkeit vorliegt oder nicht. Gerade dieser Sachverhalt ist häufig selbst von Interesse: Sind Zufallsvariablen X und Y unabhängig oder nicht?

Zur Erinnerung: Zwei (reellwertige) Zufallsvariablen X und Y sind genau dann unabhängig, wenn für alle $x, y, \in \mathbb{R}$ gilt: $\mathbb{P}(X \leq x \text{ und } Y \leq y) = \mathbb{P}(X \leq x) \cdot \mathbb{P}(Y \leq y)$. (Und: Aus Unabhängigkeit folgt $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$, aber nicht umgekehrt.)

Die Ursachen für die Ungültigkeit der obigen Gleichung und damit für die Verletzung der Unabhängigkeit zweier Zufallsvariablen können vielfältiger Natur sein, denn zwischen X und Y kann eine Abhängigkeit – man spricht auch von einer Assoziation, einem Zusammenhang oder einer Korrelation – auf verschiedene Arten zustande kommen. Für manche Abhängigkeitstypen gibt es Maße für den Grad der Abhängigkeit von X und Y. Diese Maße sind üblicherweise so skaliert, dass ihr Wertebereich $[-1, +1]$ ist und sie im Fall der Unabhängigkeit von X und Y den Wert 0 (Null) liefern. Die Verteilungstheorie ihrer empirischen Pendants erlaubt die Konstruktion von Tests der Hypothese der Unabhängigkeit und z. T. die Angabe von Konfidenzintervallen für den unbekannten Erwartungs- oder korrespondierenden theoretischen Wert des jeweiligen Zusammenhangsmaßes.

In den nächsten drei Paragraphen gehen wir näher auf drei solche Maße samt der auf ihnen basierenden Tests ein. Auf einen vierten, äußerst interessanten Test zur Aufdeckung allgemeinerer Alternativen zur Unabhängigkeitshypothese (ohne ein zugrunde liegendes Zusammenhangsmaß) verweisen wir lediglich:

- Pearsons (Produkt-Moment-)Korrelation, die die <u>lineare Assoziation</u> von X und Y quantifiziert, samt Tests und Konfidenzintervall (§8.7.1).
- Spearmans Rangkorrelation, die eine <u>monotone Assoziation</u> von X und Y (über Rangdifferenzen) quantifiziert, samt Tests (aber ohne Konfidenzintervall; §8.7.2). (Siehe z. B. [16, Bühning und Trenkler (2000)], §8.4 und für interessante technische Details z. B. [44, Hettmansperger (1984)], Abschnitt 4.4, speziell S. 201 - 205.)

- Kendalls Rangkorrelation, die ebenfalls eine monotone Assoziation von X und Y (über die Auszählung von Rang-„Fehlordnungen“) quantifiziert, samt Test (aber ohne Konfidenzintervall; §8.7.3). (Für eine knappe Darstellung siehe z. B. [16, Bühning und Trenkler (2000)], Abschnitt 8.5, Nr. (5) und für eine ausführlichere [7, Bortz et al. (2000)], §§8.2.2 und 8.2.3. Interessante technische Details stehen wieder in [44, Hettmansperger (1984)], Abschnitt 4.4 und [48, Hollander und Wolfe (1973)], Abschnitt 8.1, insbesondere in den “Comments” auf S. 185.)

- Hoeffdings Test zur Aufdeckung allgemeinerer Abhängigkeit als sie z. B. allein durch lineare oder monotone Assoziation zustande kommen kann. (Für eine ausführliche Darstellung siehe [48, Hollander und Wolfe (1973)], Abschnitt 10.2, insbesondere “Comment” 9 auf S. 235.)

Unter der Annahme, dass $(X_1, Y_1), \ldots, (X_n, Y_n)$ unabhängig und identisch wie (X, Y) verteilt sind, lassen sich auf Basis der empirischen Versionen der obigen Korrelationen Tests auf Unabhängigkeit von X und Y durchführen. Insbesondere unter der Voraussetzung einer bivariaten Normalverteilung für (X, Y) kann ein solcher Test unter Verwendung des (empirischen) Pearsonschen Korrelationskoeffizienten durchgeführt werden. Falls diese Voraussetzung nicht erfüllt zu sein scheint, erlauben der (empirische) Spearmansche und der Kendallsche Rangkorrelationskoeffizient je einen nicht-parametrischen Test auf Unabhängigkeit. (Letztere testen jedoch qua definitionem nicht auf lineare, sondern auf monotone Korrelation.) Alle drei Tests sind in der **R**-Funktion `cor.test()` implementiert. Hoeffdings Test ist in der Lage, auch – gewisse – nicht-monotone Abhängigkeiten zu entdecken, die den anderen drei Tests verborgen bleiben. Er ist in der Funktion `hoeffd()` des Paketes `Hmisc` implementiert.

Allgemeine Memos und weitere Bemerkungen: Der empirische Pearsonsche Korrelationskoeffizient r_P ist ein Maß für den *linearen* Zusammenhang zwischen den Realisierungen zweier Merkmale X und Y: Je näher r_P bei $+1$ oder -1 ist, umso enger gruppieren sich die Datenpunkte (x_i, y_i) um eine gedachte Gerade.

Dagegen misst der empirische Spearmansche oder der Kendallsche Rangkorrelationskoeffizient „lediglich“ den Grad des *monotonen* Zusammenhangs zwischen den Realisierungen von X und Y: Je näher der Koeffizient bei $+1$ oder -1 ist, umso enger gruppieren sich die Datenpunkte (x_i, y_i) in einem monotonen Verlauf.

Bei allen drei Korrelationskoeffizienten sagt ihr Wert allerdings nicht viel darüber aus, wie steil die gedachte Gerade verläuft bzw. wie steil der monotone Trend ist.

Außerdem folgt aus „Koeffizient $= 0$“ nicht, dass kein Zusammenhang existiert; es kann z. B. $\rho_P = \mathbb{E}[\hat{\rho}_S] = \tau = 0$ sein, aber X und Y in U-förmiger Beziehung zueinander stehen! Gleiches gilt auch für ihre empirischen Pendants (siehe Beispiele in den Übungen).

Für die Korrelationskoeffizienten gilt in etwa folgende Sprachregelung (wobei r hier irgendeiner der obigen Koeffizienten sei):

$$\left.\begin{array}{rcccl} & |r| & = & 0 & : \text{ keine} \\ 0 < & |r| & \leq & 0.5 & : \text{ schwache} \\ 0.5 < & |r| & \leq & 0.8 & : \text{ mittlere} \\ 0.8 < & |r| & < & 1 & : \text{ starke} \\ & |r| & = & 1 & : \text{ perfekte} \end{array}\right\} \text{(Rang-)Korrelation}$$

8.7.1 Der Pearsonsche Korrelationskoeffizient

Memo 1: Für (X, Y) und $(X_1, Y_1), \ldots, (X_n, Y_n)$ sind

$$\rho_P = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X) \cdot \mathrm{Var}(Y)}} \quad \text{der theoretische Pearsonsche Korrelationskoeffizient von } X \text{ und } Y$$

$$\text{und} \quad \hat{\rho}_P = \frac{\sum_{i=1}^n (X_i - \bar{X}_n)(Y_i - \bar{Y}_n)}{\sqrt{\sum_{i=1}^n (X_i - \bar{X}_n)^2 \sum_{i=1}^n (Y_i - \bar{Y}_n)^2}} \quad \text{der empirische Pearsonsche Korrelationskoeffizient der } (X_i, Y_i).$$

Memo 2: Im Fall eines bivariat *normal*verteilten (X, Y) sind Unkorreliertheit im Sinne von $\rho_P = 0$ und Unabhängigkeit äquivalent. Im Allgemeinen allerdings ist Unabhängigkeit eine stärkere Eigenschaft als Unkorreliertheit: Unabhängigkeit impliziert immer $\rho_P = 0$, aber nicht umgekehrt.

Zu den Tests:
<u>Annahmen:</u> (X, Y) sei bivariat normalverteilt und $(X_1, Y_1), \ldots, (X_n, Y_n)$ seien unabhängig und identisch verteilte Kopien von (X, Y).

<u>Zu testen:</u>

$$\begin{aligned} & H_0 : X \text{ und } Y \text{ sind unabhängig} && (\text{hier: } \Leftrightarrow \rho_P = 0) \\ \text{gegen} \quad & H_1 : X \text{ und } Y \text{ sind abhängig} && (\text{hier: } \Leftrightarrow \rho_P \neq 0) \end{aligned}$$

bzw.

$$\begin{aligned} & H_0' : X \text{ und } Y \text{ sind nicht } \overset{(\text{positiv})}{\text{negativ}} \text{ linear (!) assoziiert} && (\text{hier: } \Leftrightarrow \rho_P \overset{(\leq)}{\geq} 0) \\ \text{gegen} \quad & H_1' : X \text{ und } Y \text{ sind } \overset{(\text{positiv})}{\text{negativ}} \text{ linear (!) assoziiert} && (\text{hier: } \Leftrightarrow \rho_P \overset{(>)}{<} 0) \end{aligned}$$

jeweils zum Signifikanzniveau α.

<u>Teststatistik:</u>

$$\sqrt{n-2}\,\frac{\hat{\rho}_P}{\sqrt{1-\hat{\rho}_P^2}} \sim t_{n-2} \qquad \text{unter } H_0 \text{ bzw. unter } \rho_P = 0, \text{ dem „Rand“ von } H_0'.$$

<u>Entscheidungsregel</u> für konkrete Daten $(x_1, y_1), \ldots, (x_n, y_n)$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \iff p\text{-Wert} \leq \alpha,$$

wobei mit r_P als Realisierung von $\hat{\rho}_P$ und $t_n^* := \sqrt{n-2} \cdot r_P / \sqrt{1 - r_P^2}$ als Realisierung der obigen Teststatistik gilt:

$$p\text{-Wert} = \begin{cases} 2\left(1 - F_{t_{n-2}}(|t_n^*|)\right) & \text{für} \quad H_1 : \rho_P \neq 0, \\ F_{t_{n-2}}(t_n^*) & \text{"} \quad H_1' : \rho_P < 0, \\ 1 - F_{t_{n-2}}(t_n^*) & \text{"} \quad H_1' : \rho_P > 0, \end{cases}$$

worin F_{t_k} die (um 0 symmetrische) Verteilungsfunktion der t_k-Verteilung ist.

Bemerkung: Mit Hilfe der sogenannten z-Transformation von Fisher kann ein asymptotisches $(1-\alpha) * 100$ % Konfidenzintervall für den Pearsonschen Korrelationskoeffizienten ρ_P angegeben werden, wenn $n \geq 4$, nämlich: $\tanh\left(\tanh^{-1}(\hat{\rho}_P) \pm u_{1-\alpha/2}/\sqrt{n-3}\right)$.

8.7.2 Der Spearmansche Rangkorrelationskoeffizient

Memo: Für $(X_1, Y_1), \ldots, (X_n, Y_n)$ sind

$$\hat{\rho}_S = \frac{\sum_{i=1}^n \left(R(X_i) - \frac{n+1}{2}\right)\left(R(Y_i) - \frac{n+1}{2}\right)}{\sqrt{\sum_{i=1}^n \left(R(X_i) - \frac{n+1}{2}\right)^2 \sum_{i=1}^n \left(R(Y_i) - \frac{n+1}{2}\right)^2}}$$

der empirische Spearmansche Rangkorrelationskoeffizient der (X_i, Y_i)

$$\left(\stackrel{!}{=} 1 - \frac{6\sum_{i=1}^n (R(X_i) - R(Y_i))^2}{(n-1)n(n+1)}, \text{ falls keine Bindungen vorliegen} \right)$$

und $\mathbb{E}[\hat{\rho}_S] \stackrel{!}{=} \frac{3}{n+1}(\tau + (n-2)(2\gamma - 1))$ der Erwartungswert des Spearmanschen Rangkorrelationskoeffizienten der (X_i, Y_i),

wobei $\tau = 2\mathbb{P}((X_1 - X_2)(Y_1 - Y_2) > 0) - 1$ Kendalls τ ist (siehe §8.7.3) und $\gamma = \mathbb{P}((X_1 - X_2)(Y_1 - Y_3) > 0)$ (sog. "type 2 concordance", vgl. [44, Hettmansperger (1984)], S. 205).

Zu den Tests:
<u>Annahmen:</u> (X, Y) sei bivariat stetig verteilt und $(X_1, Y_1), \ldots, (X_n, Y_n)$ seien unabhängig und identisch verteilte Kopien von (X, Y).

<u>Zu testen:</u>

	$H_0 : X$ und Y sind unabhängig	$(\Rightarrow \mathbb{E}[\hat{\rho}_S] = 0)$
gegen	$H_1 : \mathbb{E}[\hat{\rho}_S] \neq 0$	($\Rightarrow X$ und Y sind monoton assoziiert, also abhängig)

bzw.

	$H_0' : X$ und Y sind unabhängig	$(\Rightarrow \mathbb{E}[\hat{\rho}_S] = 0)$
gegen	$H_1' : \mathbb{E}[\hat{\rho}_S] \overset{(>)}{<} 0$	($\Rightarrow X$ und Y sind $\overset{\text{(positiv)}}{\text{negativ}}$ monoton assoziiert, also abhängig)

jeweils zum Signifikanzniveau α.

<u>Teststatistiken:</u>

$$\left.\begin{array}{ll} S_n := \frac{(n-1)n(n+1)}{6}(1 - \hat{\rho}_S) & \text{Exakte Verteilung bekannt} \\ \sqrt{n-2}\,\frac{\hat{\rho}_S}{\sqrt{1-\hat{\rho}_S^2}} & \text{approx.} \sim t_{n-2} \\ \sqrt{n-1}\,\hat{\rho}_S & \stackrel{n\to\infty}{\longrightarrow} \mathcal{N}(0,1) \end{array}\right\} \text{unter } H_0^{(')}$$

<u>Entscheidungsregel</u> für konkrete Daten $(x_1, y_1), \ldots, (x_n, y_n)$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \iff p\text{-Wert} \leq \alpha,$$

wobei für die Berechnung des p-Wertes mit r_S als Realisierung von $\hat{\rho}_S$, $s_n := (n-1)n(n+1)/6 \cdot (1 - r_S)$ sowie $t_n^* := \sqrt{n-2} \cdot r_S / \sqrt{1 - r_S^2}$ als Realisierungen der obigen Teststatistiken die folgende Fallunterscheidung zur Anwendung kommt:

1. Für n „klein" (was in `cor.test()` $n \leq 1290$ bedeutet!) und ohne Bindungen:

$$p\text{-Wert} = \begin{cases} 2F_{S_n}(s_n) & \text{für } H_1 : \mathbb{E}[\hat{\rho}_S] \neq 0, \text{wenn } s_n \leq (n^3 - n)/6 \quad (\Leftrightarrow r_S \geq 0), \\ 2(1 - F_{S_n}(s_n - 1)) & \text{"} \quad H_1 : \mathbb{E}[\hat{\rho}_S] \neq 0, \text{wenn } s_n > (n^3 - n)/6 \quad (\Leftrightarrow r_S < 0), \\ F_{S_n}(s_n) & \text{"} \quad H_1' : \mathbb{E}[\hat{\rho}_S] < 0, \\ 1 - F_{S_n}(s_n - 1) & \text{"} \quad H_1' : \mathbb{E}[\hat{\rho}_S] > 0, \end{cases}$$

   worin F_{S_n} die um 0 symmetrische Verteilungsfunktion der diskreten (!) Verteilung von S_n unter $H_0^{(')}$ ist.

2. Für n „groß“ (also in `cor.test()` für $n > 1290$!) oder beim Vorliegen von Bindungen:

$$p\text{-Wert} = \begin{cases} 2\left(1 - F_{t_{n-2}}\left(|t_n^*|\right)\right) & \text{für} \quad H_1 : \mathbb{E}[\hat{\rho}_S] \neq 0, \\ F_{t_{n-2}}\left(t_n^*\right) & \text{"} \quad H_1' : \mathbb{E}[\hat{\rho}_S] < 0, \\ 1 - F_{t_{n-2}}\left(t_n^*\right) & \text{"} \quad H_1' : \mathbb{E}[\hat{\rho}_S] > 0, \end{cases}$$

worin F_{t_k} die (um 0 symmetrische) Verteilungsfunktion der t_k-Verteilung ist.

Bemerkungen: Beim Auftreten von Bindungen wird in `cor.test()` keine Modifikation der Teststatistiken vorgenommen, was beim Vorliegen vieler Bindungen eigentlich empfohlen wird (vgl. [16, Bühning und Trenkler (1994)], S. 237); es wird lediglich eine `warning message` ausgegeben. Die asymptotisch standardnormalverteilte Statistik $\sqrt{n-1}\,\hat{\rho}_S$ kommt in `cor.test()` nicht zum Einsatz.
Der Spearmansche Rangkorrelationskoeffizient ist erheblich robuster gegenüber Ausreißern als der Pearsonsche Korrelationskoeffizient.

8.7.3 Der Kendallsche Rangkorrelationskoeffizient

Memo: Für $(X_1, Y_1), \ldots, (X_n, Y_n)$ sind

$$\begin{aligned} \hat{\tau} &= \binom{n}{2}^{-1} \left(\sum_{1 \le i < j \le n} 1\{(X_i - X_j)(Y_i - Y_j) > 0\} - \sum_{1 \le i < j \le n} 1\{(X_i - X_j)(Y_i - Y_j) < 0\} \right) \\ &\quad \text{der empirische Kendallsche Rangkorrelationskoeffizient der } (X_i, Y_i) \\ &=: \frac{2}{n(n-1)} \cdot (K_n - D_n) \;=\; 1 - \frac{4}{n(n-1)} \cdot D_n, \quad \text{denn } K_n + D_n = \frac{n(n-1)}{2}, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{falls keine Bindungen vorliegen,} \end{aligned}$$

$$\text{und} \quad \tau = \mathbb{E}[\hat{\tau}] = 2\mathbb{P}((X_1 - X_2)(Y_1 - Y_2) > 0) - 1$$

der theoretische Kendallsche Korrelationskoeffizient von X und Y,

wobei $K_n = |\{\text{konkordante } (X_i, Y_i)\text{-}(X_j, Y_j)\text{-Paare}\}|$ und $D_n = |\{\text{diskordante } (X_i, Y_i)\text{-}(X_j, Y_j)\text{-Paare}\}|$ für alle solche Paare mit $i < j$. Zwei Paare (X_i, Y_i) und (X_j, Y_j) heißen <u>konkordant</u>, falls $(X_i - X_j)(Y_i - Y_j) > 0$ und <u>diskordant</u>, falls $(X_i - X_j)(Y_i - Y_j) < 0$.

Zu den Tests:
<u>Annahmen:</u> (X, Y) sei bivariat stetig verteilt und $(X_1, Y_1), \ldots, (X_n, Y_n)$ seien unabhängig und identisch verteilte Kopien von (X, Y).

<u>Zu testen:</u>

	$H_0 : X$ und Y sind unabhängig	$(\Rightarrow \tau = 0)$
gegen	$H_1 : \tau \neq 0$	($\Rightarrow X$ und Y sind monoton assoziiert, also abhängig)

bzw.

	$H_0' : X$ und Y sind unabhängig	$(\Rightarrow \tau = 0)$
gegen	$H_1' : \tau \overset{(>)}{<} 0$	($\Rightarrow X$ und Y sind $\overset{\text{(positiv)}}{\text{negativ}}$ monoton assoziiert, also abhängig)

jeweils zum Signifikanzniveau α.

Teststatistiken:

$$\frac{\hat{\tau}}{\sqrt{\frac{2(2n+5)}{9n(n-1)}}} = \frac{\overset{\hat{\tau}}{K_n - D_n}}{\sqrt{n(n-1)\frac{2n+5}{18}}} \begin{array}{l} \text{Exakte Verteilung bekannt} \\ \overset{n\to\infty}{\longrightarrow} \mathcal{N}(0,1) \end{array} \Bigg\} \text{unter } H_0^{(')}$$

Entscheidungsregel für konkrete Daten $(x_1, y_1), \ldots, (x_n, y_n)$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei für die Berechnung des p-Wertes mit d_n als Realisierung von D_n in $\hat{\tau}$ und $z_n := (k_n - d_n)/\sqrt{n(n-1)(2n+5)/18}$ als Realisierung der obigen, asymptotisch normalverteilten Teststatistik die folgende Fallunterscheidung zur Anwendung kommt:

1. Für n „klein“ (was in `cor.test()` $n < 50$ bedeutet) und ohne Bindungen:

$$p\text{-Wert} = \begin{cases} 2F_{D_n}(d_n) & \text{für} \quad H_1 : \tau \neq 0, \text{wenn } d_n \leq n(n-1)/4 \quad (\Leftrightarrow \hat{\tau} \geq 0), \\ 2(1 - F_{D_n}(d_n - 1)) & \text{"} \quad H_1 : \tau \neq 0, \text{wenn } d_n > n(n-1)/4 \quad (\Leftrightarrow \hat{\tau} < 0), \\ F_{D_n}(d_n) & \text{"} \quad H_1' : \tau < 0, \\ 1 - F_{D_n}(d_n - 1) & \text{"} \quad H_1' : \tau > 0, \end{cases}$$

   worin F_{D_n} die Verteilungsfunktion der diskreten (!) Verteilung von D_n unter $H_0^{(')}$ ist.

2. Für n „groß“ (also in `cor.test()` für $n \geq 50$) oder beim Vorliegen von Bindungen:

$$p\text{-Wert} = \begin{cases} 2\left(1 - \Phi\left(|z_n|\right)\right) & \text{für} \quad H_1 : \tau \neq 0, \\ \Phi\left(z_n\right) & \text{"} \quad H_1' : \tau < 0, \\ 1 - \Phi\left(z_n\right) & \text{"} \quad H_1' : \tau > 0, \end{cases}$$

   worin Φ die Standardnormalverteilungsfunktion ist.

Bemerkung: Die Beziehung zwischen $\hat{\rho}_S$ und $\hat{\tau}$ wird in [44, Hettmansperger (1984)] auf S. 203 detailliert präsentiert. Dort offenbart sich (a. a. O. in Ex. 4.4.1, S. 207), dass $\hat{\rho}_S$ zwar meist numerisch extremer als $\hat{\tau}$ ist, aber die Varianz von $\hat{\rho}_S$ kleiner als die von $\hat{\tau}$ ist (a. a. O., S. 205), weswegen auf Basis von $\hat{\tau}$ eine Abweichung von der Unabhängigkeitshypothese tendenziell eher als signifikant erklärt wird.

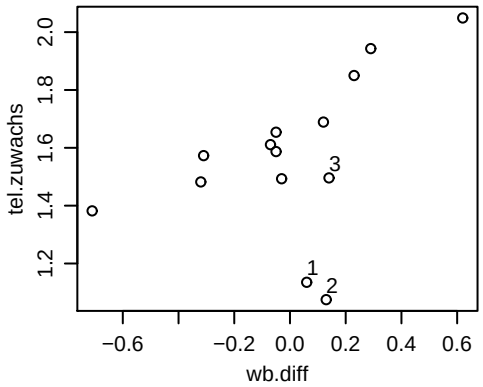
Beispiel: Anhand von Daten zum Wohnungsbau und zur Nachfrage nach privaten Telefonanschlüssen (in einer Region von New York City) werden die Tests auf Korrelation veranschaulicht. Die folgenden Daten stellen die Veränderungen in den Zahlen im Wohnungsbau und der Telefonanschlüsse (in einer speziellen „Verschlüsselung“) für 14 aufeinander folgende Jahre dar:

```
> wb.diff <- c( 0.06,  0.13,  0.14, -0.07, -0.05, -0.31,  0.12,
+               0.23, -0.05, -0.03,  0.62,  0.29, -0.32, -0.71)
> tel.zuwachs <- c( 1.135, 1.075, 1.496, 1.611, 1.654, 1.573, 1.689,
+                   1.850, 1.587, 1.493, 2.049, 1.943, 1.482, 1.382)
```

Zunächst ist auch hier eine explorative Datenanalyse angebracht. Ein Streudiagramm der Daten (x_i, y_i) mit der Identifikation dreier möglicher Ausreißer (Beobachtungen Nr. 1, 2 und eventuell 3) und separate Zeitreihen- sowie Autokorrelationsplots für jeden der Datensätze folgen:

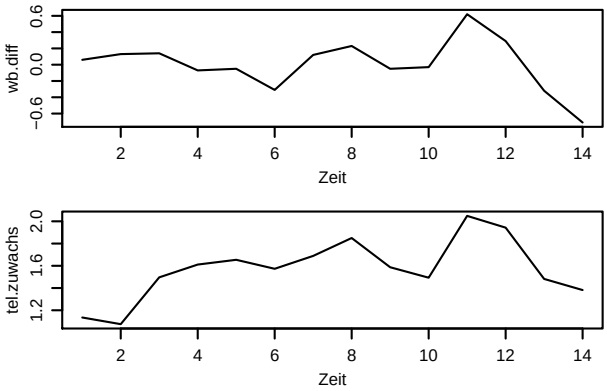
```
> plot( wb.diff, tel.zuwachs)
> identify( wb.diff, tel.zuwachs, n = 3)
```

Im Streudiagramm ist schon eine gewisse Assoziation zwischen den beiden Variablen erkennbar.




```
> ts.plot( wb.diff, xlab = "Zeit",
+ ylab = "wb.diff")
> ts.plot( tel.zuwachs, xlab = "Zeit",
+ ylab = "tel.zuwachs")
```

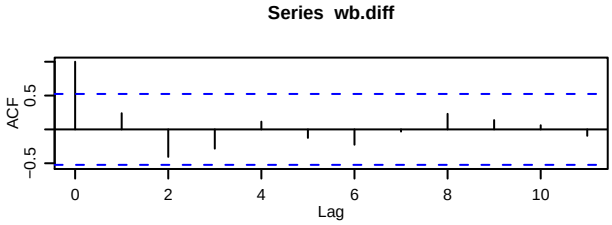
Weder in den X_i noch in den Y_i ist ein Trend oder eine Saisonalität zu sehen.

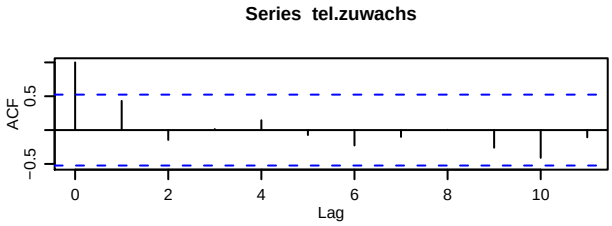



```
> acf( wb.diff)
> acf( tel.zuwachs)
```

Die X_i scheinen nicht autokorreliert zu sein und die Y_i ebenfalls nicht.

Es liegen also keine *offenkundigen* Indizien gegen die Annahme der Unabhängigkeit der (X_i, Y_i) vor.





Tests auf bivariate Unabhängigkeit	

`> cor.test( wb.diff, tel.zuwachs)`
 	
`  Pearson's product-moment correlation`
 	
`data:  wb.diff and tel.zuwachs`
`t = 1.9155, df = 12, p-value = 0.07956`
`alternative hypothesis: true correlation is not equal to 0`
`95 percent confidence interval:`
`  -0.06280425  0.80722628`
`sample estimates:`
`      cor`
`0.4839001`
	t-Test auf Unabhängigkeit auf Basis des Pearsonschen Korrelationskoeffizienten für unabhängig und identisch bivariat normalverteilte Daten ($H_0 : \rho_P = 0$). Resultate: Wert der Teststatistik (`t`), ihre Freiheitsgrade (`df`), p-Wert (`p-value`), Alternative (Voreinstellung: zweiseitig, $H_1 : \rho_P \neq 0$), asymptotisches 95 % Konfidenzintervall für ρ_P auf Basis von Fishers z-Transformation, empirischer Pearson-Korrelationskoeffizient.

`> cor.test( wb.diff, tel.zuwachs,`
`+ method = "spearman")`
 	
`  Spearman's rank correlation rho`
 	
`data:  wb.diff and tel.zuwachs`
`S = 226, p-value = 0.06802`
`alternative hypothesis: true rho is not equal to 0`
`sample estimates:`
`      rho`
`0.5038507`
 	
`Warning message:`
`In cor.test.default(wb.diff, tel.zuwachs, method = "spearman") :`
`  Cannot compute exact p-values with ties`
	Test auf Unabhängigkeit auf Basis der Spearmanschen Rangkorrelation für unabhängig und identisch bivariat stetig verteilte Daten (H_0 : X und Y sind unabhängig).
	Resultate: Wert der Teststatistik S_n (`S`), p-Wert (`p-value`) entweder exakt oder aus der t-Verteilungsapproximation, Alternative (Voreinstellung: zweiseitig, $H_1 : \mathbb{E}[\hat{\rho}_S] \neq 0$, `true rho` meint $\mathbb{E}[\hat{\rho}_S]$), empirischer Spearman-Rangkorrelationskoeffizient.
	
	Außerdem gegebenenfalls die `warning message` aufgrund von Bindungen in den Rängen.

`> cor.test( wb.diff, tel.zuwachs,`
`+ method = "kendall")`
 	
`  Kendall's rank correlation tau`
 	
`data:  wb.diff and tel.zuwachs`
`z = 2.0834, p-value = 0.03721`
`alternative hypothesis: true tau is not equal to 0`
`sample estimates:`
`      tau`
`0.4198959`
 	
`Warning message:`
`In cor.test.default(wb.diff, tel.zuwachs, method = "kendall") :`
`  Cannot compute exact p-values with ties`
	Test auf Unabhängigkeit auf Basis der Kendallschen Korrelation für unabhängig und identisch bivariat stetig verteilte Daten ($H_0 : X$ und Y sind unabhängig).
	Resultate: Entweder Wert der exakten Teststatistik D_n (dann `T`) oder der asymptotisch normalverteilten Teststatistik (dann `z`), p-Wert (`p-value`), Alternative (Voreinstellung: zweiseitig, $H_1 : \tau \neq 0$), empirischer Kendall-Rangkorrelationskoeffizient.
	Außerdem die `warning message` aufgrund von Bindungen in den Rängen.
	
	Für alle drei Tests lässt sich die Alternative mit `alternative` spezifizieren: `alternative = 'greater'`: $H_1 : \rho > 0$, `alter = 'less'`: $H_1 : \rho < 0$ mit $\rho \in \{\rho_P, \mathbb{E}[\hat{\rho}_S], \tau\}$.

Hinweis: Für Berechnungen im Zusammenhang mit der Power des Tests auf Unabhängigkeit auf Basis des Pearsonschen Korrelationskoeffizienten sei auf die Funktion `pwr.r.test()` im **R**-Paket `pwr` verwiesen.

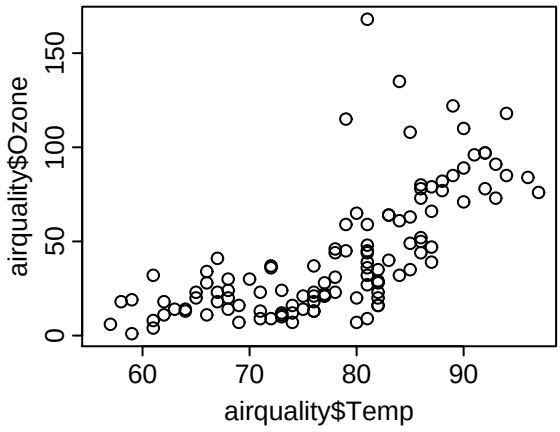
8.8 Die einfache lineare Regression

In §8.7.1 haben wir einen Test für das Vorliegen eines linearen Zusammenhangs zwischen zwei stetigen Zufallsvariablen X und Y auf Basis des empirischen Pearsonschen Korrelationskoeffizienten rekapituliert. Hat man für einen bivariaten metrisch-stetigen Datensatz einen einigermaßen deutlichen linearen Zusammenhang entdeckt, d. h. eine signifikante und im besten Fall auch numerisch hohe empirische Pearsonsche Korrelation, so ist interessant, *wie* diese lineare Beziehung zwischen X und Y aussieht, denn der Wert des Korrelationskoeffizienten besagt diesbezüglich ja nicht viel aus.
Die Erfassung des beobachteten linearen Trends läuft auf die Anpassung einer Ausgleichsgeraden durch die bivariate Punktewolke hinaus, d. h. auf den Fall der einfachen linearen Regression:

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \quad \text{für } i = 1, \ldots, n$$

Zur Illustration verwenden wir den in **R** eingebauten Data Frame `airquality`, der (unter anderem) in seiner Komponente `Ozone` Ozon-Konzentrationen enthält, die jeweils bei Temperaturen beobachtet wurden, die in der Komponente `Temp` verzeichnet sind. Unten ist ein Ausschnitt aus `airquality` und das Streudiagramm von `Ozone` gegen `Temp` gezeigt.

```
> airquality
    Ozone Solar.R Wind Temp Month Day
  1    41     190  7.4   67     5   1
  2    36     118  8.0   72     5   2
  3    12     149 12.6   74     5   3
  4    18     313 11.5   62     5   4
  5    NA      NA 14.3   56     5   5
  6    28      NA 14.9   66     5   6
  7    23     299  8.6   65     5   7
....
153    20     223 11.5   68     9  30
```



Wir wollen den Ozon-Gehalt in Abhängigkeit von der Temperatur durch eine einfache lineare Regression mit normalverteilten Fehlern modellieren, d. h., es soll das Modell

$$\texttt{Ozone}_i = \beta_0 + \beta_1 \cdot \texttt{Temp}_i + \varepsilon_i \qquad \text{für } i = 1, \ldots, n$$

mit $\varepsilon_1, \ldots, \varepsilon_n$ i.i.d. $\sim \mathcal{N}(0, \sigma^2)$ „gefittet“ werden. In **R** wird das durch die Funktion `lm()` unter Verwendung einer *Formel* zur Beschreibung des Modells wie folgt realisiert:

Einfache lineare Regression:	

`> (Oz <- lm( formula = Ozone ~ Temp,`
`+ data = airquality))`
 	
`Call: lm(formula = Ozone ~ Temp,`
`data = airquality)`
 	
`Coefficients:`
`(Intercept)  Temp`
`  -146.995  2.429`
 	
`> lm( Ozone ~ Temp, airquality)`
`....` (Argumentezuweisung per Position geht natürlich auch hier.)
	Fittet eine einfache lineare Regression von `Ozone` an `Temp` (beide aus dem Data Frame `airquality`) und speichert sie im `lm`-Objekt `Oz`. Die Tilde `~` im Argument `formula` bedeutet, dass die links von ihr stehende Variable von der rechts von ihr stehenden „abhängen“ soll. Ist eine dieser Variablen nicht in dem `data` zugewiesenen Data Frame vorhanden, wird unter den Objekten im workspace gesucht.
	Resultate: Der Funktionsaufruf (`Call`) und die Koeffizientenschätzer $\hat{\beta}_0$ (`(Intercept)`) und $\hat{\beta}_1$ (`Temp`).

`> summary( Oz)`
 	
`Call: lm( formula = Ozone ~ Temp, data = airquality)`
	Die Anwendung von `summary()` auf ein `lm`-Objekt liefert Details über das gefittete Modell, auf deren Bedeutung in aller Ausführlichkeit in Kapitel 10 eingegangen wird: Der `Call`, ...

`Residuals:`
`Min 1Q Median 3Q Max`
`-40.73 -17.41 -0.59 11.31 118.27`
	die "summary statistics" der Residuen (`Residuals`) sowie ...

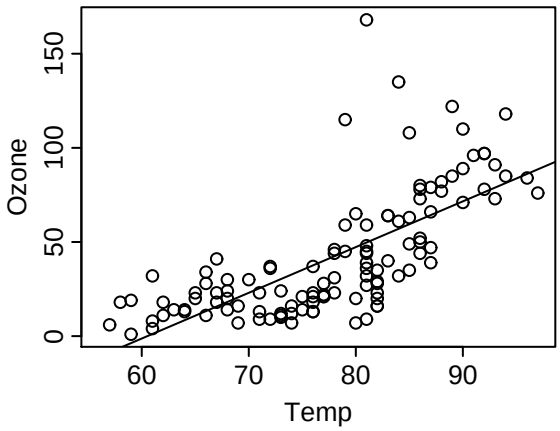
`Coefficients:`
`Estimate Std. Error`
`(Intercept) -146.9955 18.2872`
`Temp 2.4287 0.2331`
 	
`t value Pr(>|t|)`
`-8.038 9.37e-13 ***`
`10.418 < 2e-16 ***`
`---`
	die `Coefficients`-Tabelle, in der zusätzlich zu den Koeffizientenschätzern (`Estimate`) deren geschätzte Standardabweichung (`Std. Error`), die Werte der t-Teststatistiken (`t value`) der zweiseitigen Hypothesentests $H_0 : \beta_j = 0$ (für $j = 0, 1$) und deren p-Werte (`Pr(>|t|)`) stehen.

`Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1`	Die Markierungen durch Sternchen oder andere Symbole sind in der Legende `Signif. codes` darunter erläutert.

`Residual standard error: 23.71 on 114 degrees of freedom`
`(37 observations deleted due to missingness)`
`Multiple R-Squared: 0.4877,`
`Adjusted R-squared: 0.4832`
`F-statistic: 108.5 on 1 and 114 DF,`
`p-value: < 2.2e-16`
	Es folgen die Residuenstandardabweichung $\hat{\sigma}$ (`Residual standard error`) als Schätzer für σ mit ihren $n-2$ Freiheitsgraden, evtl. ein Hinweis auf fehlende Werte, der multiple R^2-Wert (`Multiple R-Squared`, hier: Quadrat des empirischen Pearsonschen Korrelationskoeffizienten der (Y_i, x_i)), sowie das korrigierte R_a^2 (`Adjusted R-Squared`), der Wert der F-Teststatistik (`F-statistic`) des „globalen" F-Tests (hier) $H_0 : \beta_1 = 0$ mit ihren Freiheitsgraden (DF) 1 und $n-2$ sowie dessen p-Wert (`p-value`).

Die Überlagerungung des Streudiagramms der Daten durch die gefittete Regressionsfunktion ist im Fall der einfachen linearen Regression sehr leicht, da es sich um eine *Gerade* (im $\mathbb{R}^2$) handelt: Die Funktion `abline()` kann aus einem `lm`-Objekt die beiden Koeffizienten extrahieren und die dazugehörige Gerade in ein bestehendes Koordinatensystem einzeichnen:

```
> with( airquality, plot( Temp, Ozone))
> abline( Oz)
```



Bemerkung: Anscheinend hat `lm()` keine Probleme damit, dass in `airquality` (mindestens) in der Komponente `Ozone` `NA`s auftreten, wie z. B. in Zeile 5. Begründung: Per Voreinstellung werden für den Modellfit diejenigen Zeilen des Data Frames eliminiert, die in den verwendeten Komponenten (hier `Ozone` und `Temp`) `NA`s enthalten. Daher ist nicht mit 153 Beobachtungen (soviele wie der Data Frame Zeilen hat) gerechnet worden, sondern nur mit den 116, für die gleichzeitig `Ozone` und `Temp` bekannt sind.

8.9 Die Formelversionen der Funktionen für die Zweistichprobentests

Die bisher vorgestellten Funktionen zur Durchführung der verschiedenen *Zwei*stichprobentests der Abschnitte 8.5, 8.6 und 8.7 – als da sind `var.test()`, `t.test()`, `wilcox.test()` und `cor.test()` – stehen auch in Versionen zur Verfügung, die die Beschreibung der Datenstruktur mittels einer Formel erlauben (wie z. B. auch bei der Funktion `boxplot()` auf Seite 74 in §4.2.2 gesehen). Für Einstichprobentests ist dies nicht möglich (und natürlich auch nicht nötig).

In den drei Funktionen `var.test()`, `t.test()` und `wilcox.test()` darf im Zweistichprobenfall das erste Argument eine Formel der Bauart „linke Seite ~ rechte Seite“ sein, wobei auf der linken Seite ein `numeric`-Vektor und auf der rechten Seite ein als `factor` mit zwei Levels interpretierbarer Vektor steht. Der Formeloperator `~` (Tilde) bedeutet dann, dass die Elemente des `numeric`-Vektors links der Tilde gemäß der beiden Levels des Faktors rechts der Tilde zu gruppieren – zu „modellieren“ – sind. (Als Umgebung, aus der diese Vektoren zu holen sind, kann durch das optionale Argument `data` ein Data Frame angegeben werden. Fehlt es, werden die Vektoren unter den Objekten im aktuellen workspace gesucht.)

Lägen die Daten von Snedecor und Cochran (siehe Seite 128) in einem Data Frame vor, wie er unten links gezeigt ist, so könnten die oben erwähnten Tests aus Abschnitt 8.5 wie im Folgenden rechts angedeutet durchgeführt werden:

```
> Protein
     Diaet Gewicht
1  p.reich     123
2  p.reich     134
3  p.reich     124
4    p.arm     107
5    p.arm      70
6  p.reich     119
7  p.reich     129
8  p.reich      97
9    p.arm      94
10   p.arm     132
11 p.reich     107
12 p.reich     104
13   p.arm     101
14 p.reich     161
15 p.reich      83
16 p.reich     146
17   p.arm      85
18   p.arm     118
19 p.reich     113
```

```
> var.test( Gewicht ~ Diaet, data = Protein)
        F test to compare two variances

data:  Gewicht by Diaet
F = 0.9298, num df = 6, denom df = 11,
p-value = 0.9788
alternative hypothesis: true ratio of variances
is not equal to 1
....

> t.test( Gewicht ~ Diaet, data = Protein)
        Welch Two Sample t-test

data:  Gewicht by Diaet
....

> wilcox.test( Gewicht ~ Diaet, data = Protein)
        Wilcoxon rank sum test with ....

data:  Gewicht by Diaet
....
```

Beachte: Die Zeilen des Data Frames brauchen die Daten nicht als homogene Blöcke zu enthalten, sondern dürfen „durcheinander“ sein.

Obiges gilt im Prinzip in analoger Form auch für die Zweistichprobentests für verbundene Stichproben in Abschnitt 8.6. Hier ist jedoch benutzer/innenseits durch die Reihenfolge der Paardaten in der verwendeten Datenstruktur sicherzustellen, dass die paarweise Zuordnung der Daten korrekt geschieht. Außerdem ist unbedingt die unten folgende Warnung zu beachten. Wir verwenden wieder die Abriebdaten von Seite 136:

```
> BHH
   Material Abrieb
1         A   14.0
2         B   13.2
3         A    8.8
4         B    8.2
5         A   11.2
6         B   10.9
7         A   14.2
8         B   14.3
9         A   11.8
10        B   10.7
11        A    6.4
12        B    6.6
13        A    9.8
14        B    9.5
15        A   11.3
16        B   10.8
17        A    9.3
18        B    8.8
19        A   13.6
20        B   13.3
```

```
> t.test( Abrieb ~ Material, data = BHH,
+ paired = TRUE)

        Paired t-test

data:  Abrieb by Material
t = 3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true difference in
means is not equal to 0
....
```

Bemerkung: In den Zeilen des Data Frames brauchen die Paardaten natürlich nicht wie im linken Beispiel abwechselnd zu stehen, sondern können z. B. auch als „Blöcke“ enthalten sein, so wie rechts. Dann ist es aber seitens des Benutzers schwieriger zu kontrollieren, ob die Datenstruktur die korrekte Paarbildung gewährleistet.

```
   Material Abrieb
1         A   14.0
2         A    8.8
3         A   11.2
....
10        A   13.6
11        B   13.2
12        B    8.2
13        B   10.9
....
20        B   13.3
```

Warnung: Treten in den Paardaten „fehlende Werte“, also `NA`s auf, ist größte Vorsicht angeraten, da genau beachtet werden muss, wie die Formelvariante von `t.test()` diese `NA`s eliminiert! In der Variante *ohne* Formel wird nämlich paarweise eliminiert und der Test auf die paarweise vollständigen Daten angewendet (wie man es vernünftigerweise wohl erwarten würde). In der Voreinstellung der Formelvariante hingegen werden – passend zur Interpretation von Data Frames – erst die Zeilen des Data Frames, die `NA`s enthalten, eliminiert, wodurch im Allgemeinen die Paarbindung zerstört wird. Danach wird der Test auf den dann „vollständigen“ Data Frame angewendet und daher kein oder – schlimmer: ohne Warnung – ein unsinniges Resultat erzielt! (Details sind auch in der Diskussion auf “**R**-help” unter `http://tolstoy.newcastle.edu.au/R/e11/help/10/08/4720.html` zu finden.)

Die Formel für die Funktion `cor.test()` sieht etwas anders aus als das, was wir oben kennengelernt haben: Sie hat eine „leere“ linke Seite und rechts der Tilde werden die beiden zu verwendenden `numeric`-Vektoren durch `+` verknüpft „aufgezählt“: `~ X + Y` (d. h., das `+` steht hier nicht für eine Addition). Die beiden Vektoren müssen dieselbe Länge haben und auch hier kann der Data Frame, aus dem diese Vektoren zu holen sind, durch das optionale Argument `data` angegeben werden. Fehlt es, werden sie im aktuellen workspace gesucht.

Wären die Daten zum Wohnungsbau und den Telefonanträgen in New York City (vgl. Seite 144) gespeichert wie in dem unten links gezeigten Data Frame, so könnte ein Test auf Korrelation durchgeführt werden wie daneben zu sehen:

```
> NewYorkCity
     wbd   tzw
1   0.06 1.135
2   0.13 1.075
3   0.14 1.496
.... .... ....
12  0.29 1.943
13 -0.32 1.482
14 -0.71 1.382
```

```
> cor.test( ~ wbd + tzw, data = NewYorkCity)

        Pearson's product-moment correlation

data:  wbd and tzw
t = 1.9155, df = 12, p-value = 0.07956
alternative hypothesis: true correlation is
not equal to 0
....
```

8.10 Zu Tests für Quotienten von Erwartungswerten der Normalverteilung

In einer Vielzahl biomedizinischer Fragestellungen spielen Untersuchungen zu Quotienten von Erwartungswerten normalverteilter Zufallsvariablen oder von Steigungskoeffizienten von Regressionsgeraden eine wichtige Rolle. Auf diese Thematik gehen wir hier nicht ein, sondern verweisen auf das **R**-package `mratios` und seine Beschreibung in [27, Dilba et al. (2007)]) in der Art eines mit Beispielen gespickten Tutoriums.

8.11 Zu Verfahren zur p-Wert-Adjustierung bei multiplen Tests

Bei statistischen Analysen jedweder Komplexität kommt es oft und sehr schnell zur Durchführung nicht nur eines, sondern *zahlreicher* statistischer Tests, von denen jeder einzelne der Gefahr eines Fehlers erster Art ausgesetzt ist, und zwar jeweils auf dem Niveau α. Die Anzahl („Multiplizität“) der Tests führt dazu, dass die Wahrscheinlichkeit für *mindestens* einen Fehler erster Art in der betrachteten Familie von Tests *erheblich* größer als das angestrebte α ist. Dieses Phänomen wird auch α-Fehler-Inflation genannt.

Diese Inflation muss und kann dadurch kompensiert werden, dass die Durchführung der einzelnen Tests insofern „strenger“ wird, als dass entweder die für die einzelnen Test geltenden lokalen Niveaus (zum Teil deutlich) kleiner als das angestrebte, globale Niveau α gewählt werden oder – äquivalent dazu – der p-Wert jedes einzelnen Tests geeignet erhöht, sprich adjustiert wird.

Eine p-Wert-Adjustierung kann in **R** mit Hilfe der Funktion `p.adjust()` explizit für einen beliebigen Satz an p-Werten ausgeführt werden. Es stehen aber auch einige spezielle Funktionen zur Verfügung, wie z. B. zur Durchführung aller paarweisen t-Tests für den Vergleich mehrerer Stichproben, die adjustierte p-Werte automatisch liefern. Es sind dies die Funktionen `pairwise.t.test()` und `pairwise.wilcox.test()`, deren „einfache“ Versionen wir schon kennengelernt haben, sowie `pairwise.prop.test()`, deren einfache Version im §9.2.2 „Der approximative Test für p: `prop.test()`“ vorgestellt wird. Für die genannten Funktionen verweisen wir auf die Online-Hilfe.

Ein recht neues Buch mit dem Titel “Multiple Comparisons Using R” ([11, Bretz et al. (2010)]) bietet eine sehr empfehlenswerte Beschreibung von Theorie und Praxis der multiplen Vergleiche, wie sie auch in einem eigenen **R**-Paket namens `multcomp` implementiert ist. Jenes Paket fungiert für dieses mit Beispielen vollgepackte Buch quasi als „Arbeitspferd“. Darüberhinaus existieren noch weitere **R**-Pakete, die Funktionalität für diese Thematik bereitstellen und auch in dem genannten Buch Erwähnung finden: `HH` und `npmc`.

8.12 Testgüte und Fallzahlschätzung für Lokationsprobleme im Normalverteilungsmodell

In den Bemerkungen zu Tests im §8.1.2 wurde auf Seite 111 nur qualitativ auf

$$\begin{aligned} \alpha &= \mathbb{P}_{H_0}(H_0 \text{ wird verworfen}) = \text{Wahrscheinlichkeit für den Fehler 1. Art und} \\ \beta &= \mathbb{P}_{H_1}(H_0 \text{ wird nicht verworfen}) = \text{Wahrscheinlichkeit für den Fehler 2. Art} \end{aligned}$$

eingegangen, wobei $\mathbb{P}_{H_*}(A)$ die Wahrscheinlichkeit des Ereignisses A bezeichne, wenn H_* wahr ist.

Eine Mindestforderung an statistische Tests ist, dass sie das Signifikanzniveau α einhalten (zumindest asymptotisch). Über die Wahrscheinlichkeit β ist im Allgemeinen nichts bekannt, obgleich sie eine nicht minder wichtige Rolle spielt, wie wir sehen werden. In diesem Zusammenhang eventuell etwas suggestiver als der Fehler 2. Art ist sein Gegenereignis: H_0 wird in der Tat verworfen, wenn H_1 wahr ist. Man sagt, „H_1 wird entdeckt.“ Die Wahrscheinlichkeit dafür, dass ein Test dies leistet, wird offenbar durch $1-\beta$ quantifiziert. Es gilt ganz allgemein die folgende

Definition der Güte (“power” in der englischen Literatur):

$$\text{Güte} := 1-\beta \equiv \mathbb{P}_{H_1}(H_0 \text{ wird verworfen})$$

Natürlich hängt die Güte von der Alternative H_1 ab, aber eben nicht nur von ihr. Man spricht daher auch von der Gütefunktion eines Tests. (Ihr Wert sollte wünschenswerterweise umso größer sein, je „weiter H_1 von H_0 abweicht“.)

8.12.1 Der zweiseitige Einstichproben-Gaußtest

Anhand eines konkreten Testszenarios analysieren wir nun $1-\beta$ im (unrealistischen!) Beispiel des zweiseitigen Einstichproben-Gaußtests. Zur Erinnerung (s. §8.1.3):

Annahmen: $X_1, \ldots, X_n$ u.i.v. $\sim \mathcal{N}(\mu, \sigma^2)$ mit unbekanntem μ und bekanntem (!) σ^2.
Zu testen: $H_0 : \mu = \mu_0$ gegen $H_1 : \mu \neq \mu_0$ zum Signifikanzniveau α.
Teststatistik:

$$\frac{\bar{X}_n - \mu_0}{\sigma/\sqrt{n}} \sim \mathcal{N}(0,1) \quad \text{unter } H_0$$

Entscheidungsregel für konkrete Daten $x_1, \ldots, x_n$ auf Basis des kritischen Wertes:

$$\text{Verwirf } H_0 \quad \Longleftrightarrow \quad \frac{|\bar{x}_n - \mu_0|}{\sigma/\sqrt{n}} \geq u_{1-\alpha/2}$$

8.12.1.1 Herleitung der Gütefunktion

Für obige Entscheidungsregel lässt sich $1-\beta$ nun wie folgt berechnen: Ist H_1 wahr, so haben die X_i als Erwartungswert das unbekannte μ ($\neq \mu_0$) und die obige Teststatistik ist *nicht* standardnormalverteilt, alldieweil sie „falsch“ zentriert wird. Indes gilt natürlich stets

$$\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \sim \mathcal{N}(0,1) \qquad \text{für alle } n \geq 1,$$

da hier korrekterweise mit μ zentriert wird. Dies nutzen wir zur Bestimmung der Verteilung der Teststatistik unter H_1 und damit für die Berechnung von $1-\beta$ aus:

$$
\begin{aligned}
1-\beta \;\equiv\; & \mathbb{P}_{H_1}(H_0 \text{ wird verworfen}) \;=\; \mathbb{P}_{H_1}\left(\frac{|\bar{X}_n-\mu_0|}{\sigma/\sqrt{n}} \geq u_{1-\alpha/2}\right) \\
= \; & \mathbb{P}_{H_1}\left(\frac{\bar{X}_n-\mu_0}{\sigma/\sqrt{n}} \geq \underbrace{u_{1-\alpha/2}}_{=\,-u_{\alpha/2}}\right) + \mathbb{P}_{H_1}\left(\frac{\bar{X}_n-\mu_0}{\sigma/\sqrt{n}} \leq \underbrace{-u_{1-\alpha/2}}_{=\,u_{\alpha/2}}\right) \\
= \; & \mathbb{P}_{H_1}\left(\underbrace{\frac{\bar{X}_n-\mu}{\sigma/\sqrt{n}}}_{\sim\,\mathcal{N}(0,1)\text{ wegen nun korrekter Zentrierung!}} \geq -u_{\alpha/2}-\frac{\mu-\mu_0}{\sigma/\sqrt{n}}\right) + \mathbb{P}_{H_1}\left(\frac{\bar{X}_n-\mu}{\sigma/\sqrt{n}} \leq u_{\alpha/2}-\frac{\mu-\mu_0}{\sigma/\sqrt{n}}\right) \\
= \; & \Phi\left(u_{\alpha/2}+\frac{\mu-\mu_0}{\sigma/\sqrt{n}}\right) + \Phi\left(u_{\alpha/2}-\frac{\mu-\mu_0}{\sigma/\sqrt{n}}\right) \qquad (5) \\
\longrightarrow \; & \begin{cases} 2\Phi\left(u_{\alpha/2}\right)=\alpha, & \text{falls } \sqrt{n}|\mu-\mu_0|/\sigma \longrightarrow 0 \\ 1, & \quad \text{"} \qquad \text{"} \qquad\quad \longrightarrow \infty \end{cases}
\end{aligned}
$$

Fazit: Wie in (5) zu erkennen, hängt die Güte $1-\beta$ im „nicht-asymptotischen“ Fall offenbar vom Niveau α, vom Stichprobenumfang n, der Standardabweichung σ und der unbekannten (!) Differenz

$$\Delta := \mu - \mu_0$$

zwischen wahrem Erwartungswert μ und hypothetisiertem Erwartungswert μ_0 ab. Als ExperimentatorIn hat man jedoch einzig auf n einen wesentlichen Einfluss. Im Prinzip zwar auch auf α (z. B. durch Inkaufnahme eines höheren Risikos für den Fehler 1. Art) und auf σ (durch Ausschaltung möglicher Streuungsquellen wie Messfehler etc.), aber faktisch nur in geringem Maße.

Die Güte hat sich also zu einer Funktion von Δ, σ, n und α konkretisiert:

$$\text{Güte } = 1-\beta(\Delta,\sigma,n,\alpha) \longrightarrow \begin{cases} \alpha, & \text{falls } \sqrt{n}|\Delta|/\sigma \longrightarrow 0 \\ 1, & \quad \text{"} \qquad \text{"} \qquad\quad \longrightarrow \infty \end{cases}$$

Eine genauere Inspektion dieser Gütefunktion zeigt, dass sie symmetrisch in Δ ist und bei Fixierung der jeweils übrigen Argumente ...

$$
\begin{array}{llll}
\ldots\text{in } |\Delta| & \text{streng} & \text{monoton} & \text{wächst,} \\
\text{"} \quad \sigma & \text{"} & \text{"} & \text{fällt,} \\
\text{"} \quad n & \text{"} & \text{"} & \text{wächst und} \\
\text{"} \quad \alpha & \text{"} & \text{"} & \text{wächst.}
\end{array}
\qquad (6)
$$

8.12.1.2 Interpretation und Veranschaulichung der Gütefunktion

- Für kleines $|\Delta|$ (d. h., μ nahe bei μ_0) ist die Güte $\approx \alpha$ (also klein!), was gut interpretierbar ist: Je geringer der Abstand von μ und μ_0, umso schlechter dürfte die Tatsache „$\mu \neq \mu_0$“ aufgrund von Daten zu entdecken sein.

- Für großes $|\Delta|$ ist die Güte ≈ 1. Dies entspricht der subjektiven Einschätzung, dass sich eine große Distanz von μ und μ_0 in den Daten deutlich bemerkbar machen müsste und die Tatsache „$\mu \neq \mu_0$“ demzufolge entdeckt würde.

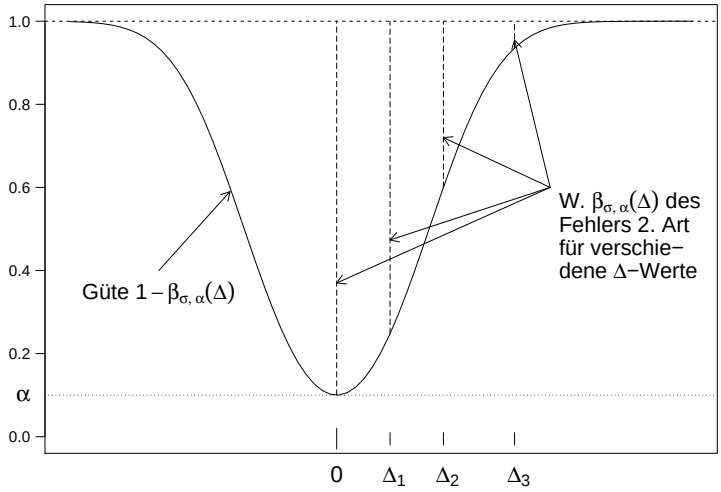
- Für jedes $|\Delta| > 0$ – auch wenn μ noch so weit von μ_0 entfernt ist – „schwächt“ eine große Streuung σ die Güte.

- Ist $|\Delta| > 0$ auch noch so klein, so kann dies dennoch durch ein hinreichend großes n stets entdeckt werden. Mit anderen Worten: Wenn man n nur weit genug erhöht, bekommt man jeden Unterschied $|\Delta| > 0$ signifikant.

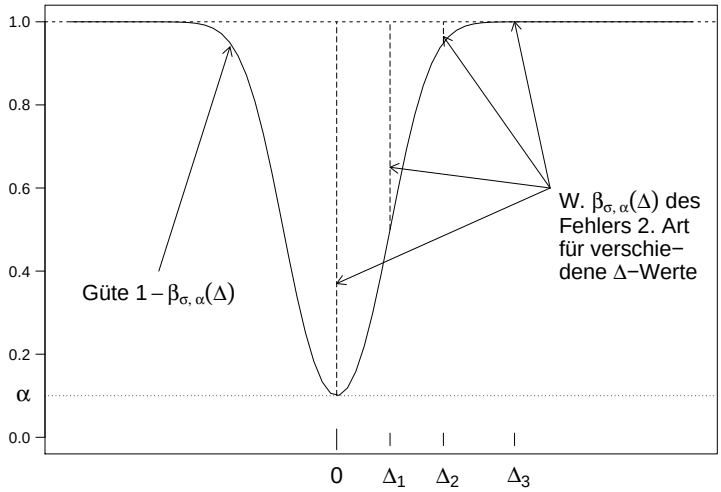
- Genauso wird eine große Streuung σ durch hinreichend viele Beobachtungen n kompensiert: „Unzuverlässigkeit“ in den einzelnen Daten kann also durch einen großen Stichprobenumfang wettgemacht werden.
- Ein „scharfes“ (= kleines) Niveau α macht es dem Test schwer, H_1 zu entdecken.

In der Regel wird das Signifikanzniveau α durch Risikoabwägungen gesteuert festgelegt und die Varianz σ^2 ist durch die Natur unveränderbar vorgegeben (aber hier als bekannt angenommen). Der wahre, aber unbekannte (!) Erwartungswert μ und somit auch Δ entziehen sich jeglichen Einflusses. Die Gütefunktion $1 - \beta_{\sigma,\alpha}(\Delta, n) := 1 - \beta(\Delta, \sigma, n, \alpha)$ quantifiziert dann also in Abhängigkeit vom Stichprobenumfang n und der Differenz Δ, mit welcher Wahrscheinlichkeit eben jenes Δ entdeckt werden kann. Wir fixieren exemplarisch $\sigma = 1$ und $\alpha = 0.1$ und veranschaulichen diese Abhängigkeit von n und Δ:

Für n = 10 ergibt sich für die Güte des obigen Gaußtests als Funktion von Δ der unten gezeigte Verlauf. Deutlich zu erkennen ist einerseits, wie mit betraglich wachsendem Δ die Güte $1 - \beta$ zunimmt (und gleichzeitig die Wahrscheinlichkeit β für den Fehler 2. Art abnimmt). Andererseits offenbart sich für betraglich kleine Δ die niedrige Güte (und gleichzeitig hohe Wahrscheinlichkeit für den Fehler 2. Art) des Tests:



Für den größeren Wert n = 30 zeigt die Güte als Funktion von Δ qualitativ denselben Verlauf. Aber klar zu erkennen ist auch, wie sich bei diesem höheren n – mit Ausnahme in der Stelle 0 – die gesamte Kurve nach oben verschoben hat, sich die Güte also global verbessert hat:



Bemerkung: Gaußtests (und auch die t-Tests) sind so konstruiert, dass die Wahrscheinlichkeit β für den Fehler 2. Art stets *möglichst klein* ist. Sie sind in diesem Sinne optimal.

8.12.1.3 Verwendungen für die Gütefunktion

Generell ist es bei Tests das Ziel, mit kleinstmöglichem Stichprobenumfang n ein betraglich möglichst kleines Δ mit größtmöglicher Wahrscheinlichkeit $1-\beta$ (= Güte) zu entdecken. Die strenge Monotonie von $1-\beta_{\sigma,\alpha}$ in $|\Delta|$ und n lässt sich hierzu auf drei Arten nutzen, die wir nun beispielhaft diskutieren werden, und zwar anhand eines Gaußtests der Hypothese $H_0 : \mu = \mu_0$ gegen die zweiseitige Alternative $H_1 : \mu \neq \mu_0$ zum Niveau $\alpha = 5$ % bei einer angenommenen Standardabweichung von $\sigma = 1$:

1. Bestimme zu gegebenen $|\Delta|$ und n die damit erreichbare Güte $1-\beta$!

   Beispiel 1: Eine medizinische Behandlung gelte als klinisch relevant (!) verschieden von einem „Goldstandard“, wenn der Wirkungsunterschied zwischen ihr und dem Goldstandard eine Mindestgröße $|\Delta|$ hat. Man möchte für eine neue Therapie nachweisen, dass sie dieses Kriterium erfüllt, hat aber nur eine begrenzte Zahl n an Untersuchungseinheiten zur Verfügung. Falls tatsächlich ein Wirkungsunterschied von mindestens $|\Delta|$ vorliegt, so kann der Gaußtest diesen Sachverhalt nur mit einer gewissen Wahrscheinlichkeit entdecken: Es ist dies die Güte $1-\beta$ zum Mindestunterschied $|\Delta|$ und zum Stichprobenumfang n.

   Konkret: Gesucht sei die Güte $1-\beta$, mit der ein Unterschied von $|\Delta| = 0.3$ bei einem Stichprobenumfang von $n = 50$ entdeckt wird.

   Die Antwort gemäß (5) lautet $1-\beta = 56.4$ %.

2. Berechne zu gegebenen $|\Delta|$ und $1-\beta$ den dazu nötigen Stichprobenumfang n!

   Beispiel 2: Will man eine vom Goldstandard um mindestens $|\Delta|$ abweichende Behandlung mit einer (Mindest-)Wahrscheinlichkeit $1-\beta$ entdecken, so benötigt der Gaußtest eine (Mindest-)Menge an Untersuchungseinheiten: Es ist dies der mindestens notwendige Stichprobenumfang n, sodass die Güte beim Mindestunterschied $|\Delta|$ den (Mindest-) Wert $1-\beta$ hat.

   Konkret: Gesucht sei der mindestens notwendige Stichprobenumfang, damit ein Unterschied von $|\Delta| = 0.3$ mit einer (Mindest-)Wahrscheinlichkeit von $1-\beta = 80$ % entdeckt wird.

   Zur Klärung des Problems müsste Gleichung (5) von Seite 153 bei gegebenen $1-\beta$ und Δ nach n aufgelöst werden, was jedoch analytisch nicht möglich ist. Numerisch ginge es zwar, ist aber etwas aufwändig, weswegen man häufig zu einer Approximation übergeht. Es gilt nämlich für die rechte Seite von (5), dass jeder ihrer beiden Summanden positiv ist und somit

$$\text{Güte} = \underbrace{\Phi\left(u_{\alpha/2} + \frac{\Delta}{\sigma/\sqrt{n}}\right)}_{<\alpha/2 \text{ für } \Delta<0} + \underbrace{\Phi\left(u_{\alpha/2} - \frac{\Delta}{\sigma/\sqrt{n}}\right)}_{<\alpha/2 \text{ für } \Delta>0} \geq \Phi\left(u_{\alpha/2} + \frac{|\Delta|}{\sigma/\sqrt{n}}\right), \tag{7}$$

   wobei die rechte Seite in (7) um höchstens $\alpha/2$ kleiner als die linke ist. Löst man also $1-\beta \leq \Phi(u_{\alpha/2} + \sqrt{n}|\Delta|/\sigma)$ nach n auf, so ist man „nahe dran“ und auch noch auf der „sicheren Seite“. Man erhält für den zweiseitigen Gaußtest:

$$n \geq \frac{\sigma^2}{\Delta^2}(u_{1-\beta} - u_{\alpha/2})^2 \tag{8}$$

   Selbstverständlich muss ein Stichprobenumfang eine natürliche Zahl sein und durch

$$n := \left\lceil \frac{\sigma^2}{\Delta^2}(u_{1-\beta} - u_{\alpha/2})^2 \right\rceil \tag{9}$$

   wird auf die nächstgrößere natürliche Zahl *auf*gerundet, um sicherzustellen, dass die tatsächlich erreichte Güte mindestens so groß bleibt, wie die nominell gewünschte. (Diese tatsächliche

Güte kann natürlich wieder mittels (5) unter Verwendung des aufgerundeten n berechnet werden.)

Lösung des oben gestellten konkreten Problems: Gemäß (8) werden (mindestens) $n = 87.21$ Untersuchungseinheiten benötigt, was natürlich zu $n = 88$ aufgerundet wird und zu einer tatsächlichen Güte von 80.4 % führt.

3. Ermittle zu gegebenen n und $1 - \beta$ den kleinsten entdeckbaren Unterschied $|\Delta|$!

   Beispiel 3: Ist die Zahl n an Untersuchungseinheiten vorgegeben und gleichzeitig die (Mindest-)Wahrscheinlichkeit $1 - \beta$, mit der ein Unterschied zwischen Behandlung und Goldstandard entdeckt werden soll, so kann dies nur für einen gewissen *Mindest*unterschied $|\Delta|$ geschehen: Es ist dies der kleinste entdeckbare Unterschied $|\Delta|$ zum Stichprobenumfang n und zur Güte $1 - \beta$.

   Konkret: Gesucht sei der kleinste Unterschied $|\Delta|$, der bei einem Stichprobenumfang von $n = 50$ mit einer Wahrscheinlichkeit von $1 - \beta = 80$ % entdeckt wird.

   Offenkundig lässt sich (5) von Seite 153 bei gegebenen $1 - \beta$ und n analytisch auch nicht nach Δ auflösen, sodass auch hier die Approximation (7) verwendet wird. Man erhält für den zweiseitigen Gaußtest

$$|\Delta| \geq \frac{\sigma}{\sqrt{n}} |u_{1-\beta} - u_{\alpha/2}| \tag{10}$$

   und setzt $|\Delta|$ natürlich *gleich* der rechten Seite. (Klar erkennbar: Der kleinste entdeckbare Unterschied $|\Delta|$ vergrößert sich bei sonst festen Parametern, wenn die Streuung der Daten zunimmt.)

   Die Lösung des konkreten Problems lautet $|\Delta| = 0.3962$.

8.12.1.4 Das Problem der unbekannten Varianz

Die Voraussetzung eines bekannten σ ist *äußerst* unrealistisch. (Daher sind die im vorhergehenden §diskutierten Anwendungen in **R** *nicht* implementiert.) Faktisch muss σ fast immer geschätzt werden, sodass jegliche statistische Inferenz auf die t-Tests hinausläuft, deren Güte in den folgenden Paragraphen analysiert wird. Als „Auswege“ aus diesem Dilemma gibt es verschiedene (approximative) Vorschläge:

1. Ignoriere das Problem und postuliere ein σ; verwende weiterhin die Standardnormalverteilung und somit die Gleichungen (5), (9) sowie (10).

2. Falls aus Vorinformationen eine deterministische (!) obere Schranke σ_+ für σ bekannt ist, verwende diese anstelle von σ in den Gleichungen (5), (9) und (10). Das liefert für die Güte $1 - \beta$ einen Mindestwert, für n einen hinreichend großen Wert, der die zum gegebenen $|\Delta|$ geforderte Testgüte für jedes $\sigma < \sigma_+$ garantiert, sowie für den kleinsten entdeckbaren Unterschied $|\Delta|$ eine untere Schranke, oberhalb derer für jedes $|\Delta|$ zum gegebenen n die geforderte Testgüte garantiert wird.

3. Falls ein Konfidenzintervall $[\hat{\sigma}_-, \hat{\sigma}_+]$ für σ aus einer unabhängigen, aber auf derselben Population beruhenden Vorschätzung existiert, verwende $\hat{\sigma}_+$ als Obergrenze für σ und gehe analog zum Verfahren 2 vor (siehe z. B. [6, Bock (1998)], S. 60 in Verbindung mit S. 32).

4. Gehe vor wie im nächsten Paragraph beschrieben, wobei auch dort keine Lösung für das Problem der unbekannten Varianz angegeben werden kann, sondern nur eine „etwas richtigere“ Methode.

8.12.2 Der zweiseitige Einstichproben-t-Test

Memo (vgl. §8.4.1):
Annahmen: $X_1, \ldots, X_n$ u.i.v. $\sim \mathcal{N}(\mu, \sigma^2)$ mit unbekannten μ und σ^2.
Zu testen: $H_0 : \mu = \mu_0$ gegen $H_1 : \mu \neq \mu_0$ zum Signifikanzniveau α.
Teststatistik:

$$\frac{\bar{X}_n - \mu_0}{\hat{\sigma}_n/\sqrt{n}} \sim t_{n-1} \quad \text{unter } H_0$$

Entscheidungsregel für konkrete Daten $x_1, \ldots, x_n$ auf Basis des kritischen Wertes:

$$\text{Verwirf } H_0 \quad \Longleftrightarrow \quad \frac{|\bar{x}_n - \mu_0|}{s_n/\sqrt{n}} \geq t_{n-1;1-\alpha/2}$$

8.12.2.1 Herleitung der Gütefunktion

Die Güte $1-\beta$ dieses Tests wird analog zu der des Gaußtests in §8.12.1.1 berechnet: Unter H_1 haben die X_i als Erwartungswert das unbekannte μ ($\neq \mu_0$) und die obige Teststatistik ist *nicht* t-verteilt, weil sie nicht korrekt zentriert ist. Es gilt nun, die Verteilung der Teststatistik unter H_1 zu bestimmen, wozu sie mit σ erweitert und ihr Zähler geeignet „teleskopiert“ wird:

$$\frac{\bar{X}_n - \mu_0}{\hat{\sigma}_n/\sqrt{n}} = \frac{\dfrac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} + \overbrace{\dfrac{\mu - \mu_0}{\sigma/\sqrt{n}}}^{=:\,\delta(\Delta,\sigma,n)}}{\dfrac{\hat{\sigma}_n}{\sigma}} \sim \frac{\mathcal{N}(0,1) + \delta(\Delta,\sigma,n)}{\sqrt{\chi^2_{n-1}/(n-1)}} = t_{n-1,\ \delta(\Delta,\sigma,n)}, \tag{11}$$

wobei $t_{k,\zeta}$ die nichtzentrale t-Verteilung zu k Freiheitsgraden mit Nichtzentralitätsparameter ζ (griech.: zeta) ist. Die Verteilungsaussage in (11) gilt sowohl unter H_0 als auch unter H_1 und das für jedes $n \geq 2$. (Unter H_0 ist lediglich $\delta(\Delta,\sigma,n) = \delta(0,\sigma,n) = 0$.) Daraus folgt:

$$\begin{aligned} 1-\beta &= \mathbb{P}_{H_1}\left(\frac{|\bar{X}_n - \mu_0|}{\hat{\sigma}_n/\sqrt{n}} \geq t_{n-1;1-\alpha/2}\right) \\ &= 1 - F_{n-1,\ \delta(\Delta,\sigma,n)}\left(t_{n-1;1-\alpha/2}\right) + F_{n-1,\ \delta(\Delta,\sigma,n)}\left(-t_{n-1;1-\alpha/2}\right), \end{aligned} \tag{12}$$

worin $F_{k,\zeta}$ die Verteilungsfunktion der $t_{k,\zeta}$-Verteilung bezeichnet (die nicht um 0, sondern um ζ symmetrisch ist). Hier gilt also:

$$1-\beta = 1-\beta(\delta(\Delta,\sigma,n),n,\alpha) \longrightarrow \begin{cases} \alpha, & \text{falls } |\delta(\Delta,\sigma,n)| \longrightarrow 0 \\ 1, & \text{"} \quad \text{"} \quad \longrightarrow \infty \end{cases}$$

Diese Gütefunktion verhält sich qualitativ genauso wie die des Gaußtests (wenngleich die Begründung für ihr Verhalten deutlich komplexer und z. B. in [38, Ghosh (1973)] zu finden ist). Daher gelten die Aussagen in (6) auf Seite 153 ebenso wie die in §8.12.1.2 anschließende Interpretation und Veranschaulichung.

Warnung: Die obige Gütefunktion enthält im Nichtzentralitätsparameter „versteckt“ die nach wie vor unbekannte Standardabweichung σ, sodass sie in der Praxis faktisch nicht ausgewertet werden kann, obwohl die Verteilungsfunktion der nichtzentralen t-Verteilung bekannt ist!

Auch Gleichung (12) ist bei gegebenen $1-\beta$ und Δ analytisch nicht nach n aufzulösen, sodass man häufig folgende, zu (7) analoge Approximation nutzt:

$$\underbrace{1 - F_{n-1,\ \delta(\Delta,\sigma,n)}\left(t_{n-1;1-\alpha/2}\right)}_{<\,\alpha/2 \text{ für } \Delta<0} + \underbrace{F_{n-1,\ \delta(\Delta,\sigma,n)}\big(\overbrace{-t_{n-1;1-\alpha/2}}^{=\,t_{n-1;\alpha/2}}\big)}_{<\,\alpha/2 \text{ für } \Delta>0} \geq F_{n-1,\ -\delta(|\Delta|,\sigma,n)}\left(t_{n-1;\alpha/2}\right), \tag{13}$$

wobei die rechte Seite in (13) um höchstens $\alpha/2$ kleiner als die linke ist. Könnte man also $1-\beta \leq F_{n-1,\ -\delta(|\Delta|,\sigma,n)}\left(t_{n-1;\alpha/2}\right)$ nach n auflösen, so wäre man „nahe dran“ und auch noch auf der „sicheren Seite“. Ungünstigerweise steckt n auch im Nichtzentralitätsparameter $-\delta(|\Delta|,\sigma,n)$, was aber über eine Approximation der Quantile der $t_{k,\zeta}$-Verteilung durch diejenigen der (zentralen) t-Verteilung gemäß $t_{k,\zeta;\gamma} \approx t_{k;\gamma} + \zeta$ kompensiert werden kann. Damit leitet man dann analog zu (8) und (10) für den zweiseitigen t-Test die Beziehungen

$$n \geq \frac{\sigma^2}{\Delta^2}(t_{n-1;1-\beta} - t_{n-1;\alpha/2})^2 \qquad \text{und} \qquad |\Delta| \geq \frac{\sigma}{\sqrt{n}}|t_{n-1;1-\beta} - t_{n-1;\alpha/2}| \tag{14}$$

her (vgl. z. B. [6, Bock (1998)], S. 59). Die linke Ungleichung (für n) kann aber immer noch nicht analytisch nach n aufgelöst werden; es muss numerisch geschehen (was in **R** implementiert ist), falls ein (Vor-)Schätzer $\hat{\sigma}$ für σ zur Verfügung ist! Der so erhaltene Stichprobenumfang wird nötigenfalls auf die nächstgrößere natürliche Zahl *auf*gerundet und die damit tatsächlich erreichte Güte kann dann wieder mittels (12) berechnet werden.

8.12.2.2 Verwendung der Gütefunktion

Die drei obigen Anwendungen sind in **R** durch die Funktion `power.t.test()` implementiert. Einige ihrer Argumente sind auf gewisse Standardwerte voreingestellt. Deren Argumentnamen und Voreinstellungswerte nebst den von uns für sie benutzten Bezeichnungen lauten wie folgt:

n	Δ	σ	α	$1-\beta$
`n = NULL`	`delta = NULL`	`sd = 1`	`sig.level = 0.05`	`power = NULL`

H_1	Art des Tests	Exakt
`alternative = "two.sided"`	`type = "two.sample"`	`strict = FALSE`

Der/die Benutzer/in muss dafür sorgen, dass beim Aufruf von `power.t.test()` genau vier der Argumente `n`, `delta`, `sd`, `sig.level` und `power` mit einem von `NULL` verschiedenen Wert versehen werden, damit der Wert des „fehlenden“ Arguments dann aus den Werten der anderen abgeleitet wird. Beachte, dass `sd` und `sig.level` als Voreinstellung *nicht* den Wert `NULL` haben, sodass, wenn einer dieser beiden berechnet werden soll, `NULL` für eben denjenigen explizit anzugeben ist. Mit `alternative` lässt sich H_1 als `"two.sided"` oder `"one.sided"` spezifizieren, worauf wir in noch folgenden Paragraphen eingehen. Der `type` des Tests kann `"two.sample"`, `"one.sample"` oder `"paired"` sein, was wir ebenfalls noch genauer diskutieren werden. (Im aktuellen Paragraph müssen wir `type = "one.sample"` setzen.)
Wird `strict = TRUE` angegeben, führt `power.t.test()` alle Berechnungen auf Basis der exakten Formel (12) durch und nicht unter Verwendung der Approximation (13). (Ohne diese Wahl ist die durch `power.t.test()` berechnete „Güte unter H_0“ nur $\alpha/2$.)
Die jeweiligen Argumentwerte und Berechnungsergebnisse werden schließlich zusammen in einer Liste zurückgegeben (s. u.).

Für die nun folgenden, konkreten Beispiele betrachten wir das gleiche Testszenario wie in §8.12.1.3: $H_0 : \mu = \mu_0$ gegen die zweiseitige Alternative $H_1 : \mu \neq \mu_0$ zum Niveau $\alpha = 5$ %, aber bei einer angenommenen Standardabweichung von $\sigma = 0.9$ (d. h., wir wählen **R**s Voreinstellung für α):

1. Bestimme zu gegebenen $|\Delta|$ und n die damit erreichbare Güte $1-\beta$!

   Beispiel 1: Gesucht sei die Güte $1-\beta$, mit der ein Unterschied von $|\Delta| = 0.3$ bei einem Stichprobenumfang von $n = 50$ entdeckt wird.

```
> power.t.test( n = 50, delta = 0.3, sd = 0.9, type = "one.sample")
     One-sample t test power calculation
              n = 50
```

```
          delta = 0.3
             sd = 0.9
      sig.level = 0.05
          power = 0.6370846
    alternative = two.sided
```

Ein Aufruf mit `delta = -0.3` liefert exakt dieselbe Ausgabe, da `delta` nur mit seinem Betrag in die Berechnungen eingeht.

2. Berechne zu gegebenen $|\Delta|$ und $1-\beta$ den dazu nötigen Stichprobenumfang n!

   Beispiel 2: Gesucht sei der mindestens notwendige Stichprobenumfang, damit ein Unterschied von $|\Delta| = 0.3$ mit einer (Mindest-)Wahrscheinlichkeit von $1-\beta = 80$ % entdeckt wird.

```
> power.t.test( delta = 0.3, power = 0.8, sd = 0.9, type = "one.sample")
     One-sample t test power calculation
              n = 72.58407
          delta = 0.3
             sd = 0.9
      sig.level = 0.05
          power = 0.8
    alternative = two.sided
```

   Das Aufrunden des Stichprobenumfangs bleibt dem/der Benutzer/in überlassen.

3. Ermittle zu gegebenen n und $1-\beta$ den kleinsten entdeckbaren Unterschied $|\Delta|$!

   Beispiel 3: Gesucht sei der kleinste Unterschied $|\Delta|$, der bei einem Stichprobenumfang von $n = 50$ mit einer Wahrscheinlichkeit von $1-\beta = 80$ % entdeckt wird.

```
> power.t.test( n = 50, power = 0.8, sd = 0.9, type = "one.sample")
     One-sample t test power calculation
              n = 50
          delta = 0.3637661
             sd = 0.9
      sig.level = 0.05
          power = 0.8
    alternative = two.sided
```

Bemerkung: `power.t.test()` akzeptiert außer für `delta` für keines seiner Argumente vektorielle Eingabewerte. D. h., man kann nicht (ohne die Verwendung einer Schleife o. Ä.) z. B. zu verschiedenen Gütewerten bei festem Δ die dazu nötigen Stichprobenumfänge mit *einem* Aufruf von `power.t.test()` bestimmen lassen; wohl aber für festes n und verschiedene Δ-Werte die zugehörigen Gütewerte:

```
> power.t.test( n = 50, delta = c( 0, 0.1, 0.2, 0.3, 0.4), sd = 0.9,
+ type = "one.sample")
     One-sample t test power calculation
              n = 50
          delta = 0.0, 0.1, 0.2, 0.3, 0.4
             sd = 0.9
      sig.level = 0.05
          power = 0.0250000, 0.1171034, 0.3374697, 0.6370846, 0.8687654
    alternative = two.sided
```

8.12.3 Der einseitige Einstichproben-t-Test

Wenn ein zweiseitiger Test nicht notwendig ist, sondern einer mit einseitiger Alternative ausreicht, sollte man dies ausnutzen, da es sich günstig auf Stichprobenumfang und/oder Güte auswirkt. Die Argumentation verläuft analog zum zweiseitigen Fall, aber doch etwas einfacher. Zur Erinnerung (vgl. §8.4.1):

<u>Annahmen:</u> $X_1, \ldots, X_n$ u.i.v. $\sim \mathcal{N}(\mu, \sigma^2)$ mit unbekannten μ und σ^2.
<u>Zu testen:</u> $H_0' : \mu \overset{(\leq)}{\geq} \mu_0$ gegen $H_1' : \mu \overset{(>)}{<} \mu_0$ zum Signifikanzniveau α.

<u>Teststatistik:</u>

$$\frac{\bar{X}_n - \mu_0}{\hat{\sigma}_n/\sqrt{n}} \sim t_{n-1} \quad \text{unter dem „Rand“ } \mu = \mu_0 \text{ von } H_0'$$

<u>Entscheidungsregel</u> für konkrete Daten $x_1, \ldots, x_n$ auf Basis des kritischen Wertes:

$$\text{Verwirf } H_0' \quad \Longleftrightarrow \quad \frac{\bar{x}_n - \mu_0}{s_n/\sqrt{n}} \overset{(\geq +)}{\leq -} t_{n-1;1-\alpha}$$

8.12.3.1 Gütefunktion: Herleitung, Eigenschaften und Veranschaulichung

Völlig analog zur Herleitung der Gleichung (12) in §8.12.2.1 folgt:

$$1 - \beta \;=\; \mathbb{P}_{H_1'}\left(\frac{\bar{X}_n - \mu_0}{\hat{\sigma}_n/\sqrt{n}} \overset{(\geq +)}{\leq -} t_{n-1;1-\alpha}\right) \;=\; \overset{(1\ -)}{} F_{n-1,\ \delta(\Delta,\sigma,n)}\big(\overset{(+)}{-}\, t_{n-1;1-\alpha}\big), \tag{15}$$

wobei $F_{k,\zeta}$ wie zuvor die Verteilungsfunktion der $t_{k,\zeta}$-Verteilung bezeichnet und $\delta(\Delta, \sigma, n)$ wie in (11) definiert ist. Auflösen nach n oder Δ geschieht (wie für (14)) mit Hilfe der Approximation der $t_{k,\zeta}$-Quantile gemäß $t_{k,\zeta;\gamma} \approx t_{k;\gamma} + \zeta$, wobei für die Umformungen beachtet werden muss, dass Δ unter $\mu < \mu_0$ negativ ist und unter $\mu > \mu_0$ positiv:

$$n \geq \frac{\sigma^2}{\Delta^2}(t_{n-1;1-\beta} - t_{n-1;\alpha})^2 \qquad \text{und} \qquad \Delta \overset{(\geq +)}{\leq -} \frac{\sigma}{\sqrt{n}}(t_{n-1;1-\beta} - t_{n-1;\alpha}) \tag{16}$$

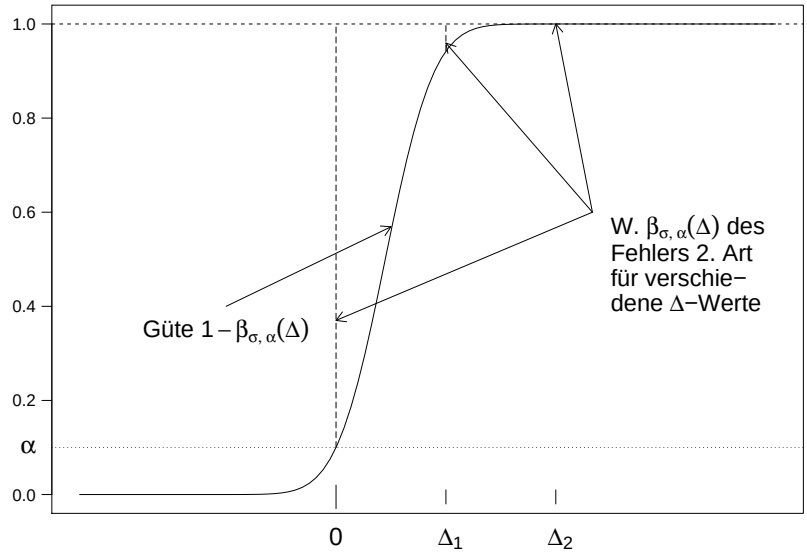
Beachte, dass hier α steht, wo sich in (14) $\alpha/2$ befindet.

Die n-Ungleichung muss man – wie zuvor – numerisch lösen (unter Verwendung eines (Vor-)Schätzers für σ), den so erhaltenen Stichprobenumfang nötigenfalls auf eine natürliche Zahl *auf*runden und, falls gewünscht, die damit tatsächlich erreichte Güte mit (15) berechnen.

Das Verhalten der Gütefunktion lautet wie folgt

$$1 - \beta(\delta(\Delta, \sigma, n), n, \alpha) \;\longrightarrow\; \begin{cases} \alpha, & \text{falls } \delta(\Delta, \sigma, n) \longrightarrow 0 \\ 1, & \text{"} \quad \text{"} \quad \longrightarrow -\infty\ (+\infty) \\ 0, & \text{"} \quad \text{"} \quad \longrightarrow +\infty\ (-\infty) \end{cases}$$

Die Gütefunktion des einseitigen t-Tests $H_0' : \mu \leq \mu_0$ gegen $H_1' : \mu > \mu_0$ als Funktion von Δ hat qualitativ den auf der nächsten Seite oben gezeigten Verlauf (für den Test gegen $H_1' : \mu < \mu_0$ ergibt sich der an der senkrechten Achse $\Delta = 0$ gespiegelte Verlauf):



8.12.3.2 Verwendung der Gütefunktion

In **R** ist die Gütefunktion (15) nur in ihrer Version für $H_0' : \mu \leq \mu_0$ gegen die $H_1' : \mu > \mu_0$ implementiert (da sich die andere Variante im Ergebnis nur im Vorzeichen für Δ unterscheidet). Für die nun folgenden, konkreten Beispiele betrachten wir eben dieses t-Testszenario zum Niveau $\alpha = 5$ %, bei einer angenommenen Standardabweichung von $\sigma = 0.9$ (also **R**s Voreinstellung für α):

1. Bestimme zu gegebenen Δ und n die damit erreichbare Güte $1 - \beta$!

   Beispiel 1′: Gesucht sei die Güte $1 - \beta$, mit der ein Unterschied von $\Delta = 0.3$ bei einem Stichprobenumfang von $n = 50$ entdeckt wird.

```
> power.t.test( n = 50, delta = 0.3, sd = 0.9, alternative = "one.sided",
+ type = "one.sample")
     One-sample t test power calculation
              n = 50
          delta = 0.3
             sd = 0.9
      sig.level = 0.05
          power = 0.7515644
    alternative = one.sided
```

   Ein Aufruf mit `delta = -0.3` ist unsinnig, da wir uns in diesem Fall gar nicht mehr unter der Alternative befinden! Dennoch ist die Funktion (15) definiert, führt aber zu einer winzigen „Güte“:

```
> power.t.test( n = 50, delta = -0.3, sd = 0.9, alternative = "one.sided",
+ type = "one.sample")
     One-sample t test power calculation
              n = 50
          delta = -0.3
             sd = 0.9
      sig.level = 0.05
          power = 3.593939e-05
    alternative = one.sided
```

2. Berechne zu gegebenen Δ und $1-\beta$ den dazu nötigen Stichprobenumfang n!

   Beispiel 2′: Gesucht sei der mindestens notwendige Stichprobenumfang, damit ein Unterschied von $\Delta = 0.3$ mit einer (Mindest-)Wahrscheinlichkeit von $1-\beta = 80$ % entdeckt wird.

```
> power.t.test( delta = 0.3, power = 0.8, sd = 0.9, alter = "one.sided",
+ type = "one.sample")
     One-sample t test power calculation
              n = 57.02048
          delta = 0.3
             sd = 0.9
      sig.level = 0.05
          power = 0.8
    alternative = one.sided
```

   Ein (unsinniger) Aufruf mit `delta = -0.3` liefert eine (kryptische, aber berechtigte) Fehlermeldung in der Nullstellensuche zur Lösung von (15), was daran liegt, dass die Gütefunktion auf $\Delta < 0$ nie den Wert $1-\beta$ (für $1-\beta > \alpha$) erreicht:

```
> power.t.test( delta = -0.3, power = 0.8, sd = 0.9, alter = "one.sided",
+ type = "one.sample")
Fehler in uniroot(function(n) eval(p.body) - power, c(2, 1e+07)) :
        f() values at end points not of opposite sign
```

3. Ermittle zu gegebenen n und $1-\beta$ den kleinsten entdeckbaren Unterschied Δ!

   Beispiel 3′: Gesucht sei der kleinste Unterschied Δ, der bei einem Stichprobenumfang von $n = 50$ mit einer Wahrscheinlichkeit von $1-\beta = 80$ % entdeckt wird.

```
> power.t.test( n = 50, power = 0.8, sd = 0.9, alternative = "one.sided",
+ type = "one.sample")
     One-sample t test power calculation
              n = 50
          delta = 0.3209412
             sd = 0.9
      sig.level = 0.05
          power = 0.8
    alternative = one.sided
```

Bemerkung: Zur Güte des nicht-parametrischen Pendants Wilcoxons Vorzeichen-Rangtest vgl. z. B. [16, Büning & Trenkler (1994)], Seite 100. Hierzu ist in **R** (meines Wissens) bisher nichts implementiert.

8.12.4 Die Zweistichproben-t-Tests

Wir verzichten auf die detaillierte Herleitung und Diskussion der Gütefunktionen der Zweistichproben-t-Tests. Die Vorgehensweise ist analog zu derjenigen im Einstichprobenfall. Zu beachten sind allerdings die im Folgenden zusammengefassten Aspekte:

8.12.4.1 Zwei verbundene Stichproben

Memo (vgl. §8.6.1 auf Seite 135):
<u>Annahmen:</u> $D_i := X_i - Y_i$ u. i. v. $\sim \mathcal{N}(\mu_D, \sigma_D^2)$ mit unbekannten μ_D und σ_D^2.

Zu testen: $H_0 : \mu_D = \mu_0$ gegen $H_1 : \mu_D \neq \mu_0$ bzw. $H_0' : \mu_D \overset{(\leq)}{\geq} \mu_0$ gegen $H_1' : \mu_D \overset{(>)}{<} \mu_0$ zum Signifikanzniveau α.
Teststatistik:

$$\frac{\bar{D}_n - \mu_0}{\hat{\sigma}_n/\sqrt{n}} \sim t_{n-1} \quad \text{unter } \mu_D = \mu_0$$

Entscheidungsregel für konkrete Daten $d_1, \ldots, d_n$ auf Basis des kritischen Wertes:

$$\text{Verwirf } H_0 \iff \frac{|\bar{d}_n - \mu_0|}{s_n/\sqrt{n}} \geq t_{n-1;1-\alpha/2} \quad \text{bzw. verwirf } H_0' \iff \frac{\bar{d}_n - \mu_0}{s_n/\sqrt{n}} \overset{(\geq +)}{\leq -} t_{n-1;1-\alpha}$$

Hier läuft es auf dieselben Gütefunktionen wie im Einstichprobenfall hinaus (für den zweiseitigen Fall siehe (12) auf S. 157 und für den einseitigen (15) auf S. 160), allerdings mit $\Delta = \mu_D - \mu_0$ und $\delta(\Delta, \sigma_D, n), n, \alpha) = \sqrt{n}(\mu_D - \mu_0)/\sigma_D$.

Für die nun folgenden Beispiele betrachten wir den zweiseitigen Fall zum Niveau $\alpha = 5$ % (der Voreinstellung für α in **R**) und einer angenommenen Standardabweichung von $\sigma_D = 0.9$:

1. Bestimme zu gegebenen $|\Delta|$ und n die damit erreichbare Güte $1 - \beta$!

   Beispiel 1″: Gesucht sei die Güte $1 - \beta$, mit der ein Unterschied von $|\Delta| = 0.3$ bei einem Stichprobenumfang von $n = 50$ entdeckt wird.

```
> power.t.test( n = 50, delta = 0.3, sd = 0.9, type = "paired")
     Paired t test power calculation
....
          power = 0.6370846
....
```

   ... (Bis auf den folgenden Hinweis `NOTE` *steht hier qualitativ exakt dieselbe Ausgabe wie für Beispiel 1 auf S. 158) ...*

```
NOTE: n is number of *pairs*, sd is std.dev. of *differences* within pairs
```

2. Berechne zu gegebenen $|\Delta|$ und $1 - \beta$ den dazu nötigen Stichprobenumfang n!

   Beispiel 2″: Gesucht sei der mindestens notwendige Stichprobenumfang, damit ein Unterschied von $|\Delta| = 0.3$ mit einer (Mindest-)Wahrscheinlichkeit von $1 - \beta = 80$ % entdeckt wird.

```
> power.t.test( delta = 0.3, power = 0.8, sd = 0.9, type = "paired")
     Paired t test power calculation
              n = 72.58407
....
```

   ... (Bis auf `NOTE` *qualitativ exakt dieselbe Ausgabe wie für Beispiel 2 auf S. 159) ...*

```
NOTE: n is number of *pairs*, sd is std.dev. of *differences* within pairs
```

3. Ermittle zu gegebenen n und $1 - \beta$ den kleinsten entdeckbaren Unterschied $|\Delta|$!

   Beispiel 3″: Gesucht sei der kleinste Unterschied $|\Delta|$, der bei einem Stichprobenumfang von $n = 50$ mit einer Wahrscheinlichkeit von $1 - \beta = 80$ % entdeckt wird.

```
> power.t.test( n = 50, power = 0.8, sd = 0.9, type = "paired")
     Paired t test power calculation
....
        delta = 0.3637661
....
```

... (Bis auf `NOTE` *qualitativ dieselbe Ausgabe wie für Bsp. 3 auf S. 159) ...*

```
NOTE: n is number of *pairs*, sd is std.dev. of *differences* within pairs
```

8.12.4.2 Zwei unverbundene Stichproben

Hierbei stellt sich die Frage, ob die Varianzen in den beiden Populationen, die den Stichproben zugrunde liegen, als gleich angenommen werden können oder nicht:

Falls ja (vgl. §8.5.2), kann man zeigen, dass die Güte maximal bzw. der Gesamtstichprobenumfang (also $n + m$ in der Notation von §8.5.2) minimal wird, wenn beide Stichproben gleich groß sind (wie z. B. in [6, Bock (1998)], S. 62 ausgeführt). Bei gleich großen Stichproben ergeben sich dann zum Einstichprobenfall baugleiche Gütefunktionen (für den zweiseitigen Fall wie (12) auf S. 157 und für den einseitigen wie (15) auf S. 160), allerdings mit $\Delta = \mu_X - \mu_Y$, Nichtzentralitätsparameter $\delta(\Delta, \sigma, n), n, \alpha) = \sqrt{n}\Delta/(\sqrt{2}\sigma)$ und $2(n-1)$ Freiheitsgraden (und nicht $n-1$) für die t-Verteilung und ihre Quantile! Analog gelten die Abschätzungen (14) und (16), wenn darin σ durch $\sqrt{2}\sigma$ ersetzt wird.

Falls man von ungleichen Varianzen ausgehen und daher Welchs Modifikation des t-Tests zum Einsatz kommen muss (siehe §8.5.3), wird das Problem komplizierter: die optimalen Stichprobenumfänge hängen dann vom unbekannten (!) Verhältnis der Varianzen σ_X^2 und σ_Y^2 ab (siehe [6, Bock (1998)], S. 65). Für diesen Fall ist in base-**R** keine Güte- bzw. Stichprobenumfangsberechnung implementiert, sondern nur für denjenigen, in dem gleiche Varianzen angenommen werden. Siehe hierzu jedoch die Bemerkungen auf der nächsten Seite unten!

Beispiele: Wir betrachten den Test der Hypothese $H_0 : \mu_X = \mu_Y$ gegen die zweiseitige Alternative $H_1 : \mu_X \neq \mu_Y$ und führen die zu den Beispielen $1''$, $2''$ und $3''$ des zweiseitigen Tests im vorherigen §8.12.4.1 analogen Berechnungen durch (mit $\alpha = 5$ % und einer angenommenen Standardabweichung von $\sigma_X = \sigma_Y = \sigma = 0.9$):

1. Bestimme zu gegebenen $|\Delta|$ und n die damit erreichbare Güte $1 - \beta$!

   Beispiel $1'''$: Gesucht sei die Güte $1 - \beta$, mit der Unterschied von $|\Delta| = 0.3$ bei gleichen Stichprobenumfängen $n = m$ entdeckt wird:

```
> power.t.test( n = 50, delta = 0.3, sd = 0.9, type = "two.sample")
     Two-sample t test power calculation
              n = 50
          delta = 0.3
             sd = 0.9
      sig.level = 0.05
          power = 0.3784221
    alternative = two.sided

NOTE: n is number in *each* group
```

2. Berechne zu gegebenen $|\Delta|$ und $1 - \beta$ den dazu nötigen Umfang n je Stichprobe!

   Beispiel $2'''$: Gesucht sei der notwendige Stichprobenumfang, damit ein Unterschied von $|\Delta| = 0.3$ mit einer (Mindest-)Wahrscheinlichkeit von $1 - \beta = 80$ % entdeckt wird.

```
> power.t.test( delta = 0.3, power = 0.8, sd = 0.9, type = "two.sample")
     Two-sample t test power calculation
              n = 142.2466
          delta = 0.3
             sd = 0.9
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

 NOTE: n is number in *each* group
```

Die Antwort lautet demnach $n = 143$, also insgesamt 286 Untersuchungseinheiten.

3. Ermittle zu gegebenen n und $1 - \beta$ den kleinsten entdeckbaren Unterschied $|\Delta|$!

   Beispiel 3‴: Gesucht sei der kleinste Unterschied $|\Delta|$, der bei gleichen Stichprobenumfängen $n = m$ mit einer Wahrscheinlichkeit von $1 - \beta = 80$ % entdeckt wird.

```
> power.t.test( n = 50, power = 0.8, sd = 0.9, type = "two.sample")
     Two-sample t test power calculation
              n = 50
          delta = 0.5092952
             sd = 0.9
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

 NOTE: n is number in *each* group
```

Bemerkungen:

1. Die Probleme mit den in der Regel unbekannten Varianzen σ_X^2 und σ_Y^2 sind dieselben wie die für den Einstichprobenfall in §8.12.1.4 auf Seite 156 diskutierten.

2. Zur Güte von Wilcoxons Rangsummentest als nicht-parametrischem Pendant siehe z. B. [16, Büning & Trenkler (1994)], S. 135, oder die in [86, Zhao et. al (2008)] zitierte Literatur. Hierzu steht in base-**R** (meines Wissens) bisher nichts zur Verfügung. Aber in einem relativ neuen **R**-Paket namens `samplesize` ist eine entsprechende Funktion auf Basis von [86, Zhao et. al (2008)] für den Fall *ordinal* skalierter Daten (also für Daten mit Bindungen!) implementiert, worauf wir hier aber nicht näher eingehen.

3. Für Fälle, in denen verschiedene Stichprobenumfänge $n \neq m$ betrachtet bzw. gewünscht werden, enthält z. B. das **R**-Paket `pwr` ein paar nützliche Funktionen wie `pwr.t2n.test()` oder auch das bereits erwähnte Paket `samplesize` mit seiner Funktion `n.indep.t.test.neq()`.

4. Für den Fall ungleicher Varianzen (also für Welchs t-Test; vgl. §8.5.3) bietet – ebenfalls – das Paket `samplesize` die Funktion `n.welch.test()`, in der die Stichprobenumfangsschätzung gemäß [6, Bock (1998)], S. 65 implementiert ist.

9 Zur Inferenzstatistik und Parameterschätzung für Nominaldaten

In diesem Kapitel geht es zunächst um die Auftrittswahrscheinlichkeit(en) *eines* interessierenden Ereignisses in einer bzw. mehreren Grundgesamtheit/en. Danach wird ein Testverfahren für Hypothesen über die Auftrittswahrscheinlichkeiten von *mehreren*, sich gegenseitig ausschließenden Ereignissen vorgestellt. Sodann werden einige Tests für die Hypothese der Unabhängigkeit endlich-diskreter (und höchstens ordinal skalierter) Variablen behandelt.

9.1 Bernoulli-Experimente mit `sample()`

Ein dichotomes Zufallsexperiment, d. h. eins mit prinzipiell nur zwei möglichen Ergebnissen wird auch *Bernoulli-Experiment* genannt. Zu seiner Modellierung wird häufig eine 0-1-wertige Zufallsvariable, sagen wir B, verwendet, deren mögliche Werte 0 und 1 oft mit „Misserfolg“ (oder „Niete“) bzw. „Erfolg“ (oder „Treffer“) bezeichnet werden. Die Wahrscheinlichkeit p – der *Bernoulli-Parameter* – für das Auftreten des Ereignisses $\{B = 1\}$ wird daher auch *Erfolgswahrscheinlichkeit* genannt.

Hat man es mit einer Reihe von n solchen dichotomen Zufallsexperimenten zu tun, die sich gegenseitig nicht beeinflussen und unter gleichen Bedingungen ablaufen, sodass ihre Modellierung durch eine Folge von n stochastisch unabhängigen und gleichartigen Bernoulli-Experimenten adäquat erscheint, spricht man von einer *Bernoulli-Kette der Länge* n. (Die geforderte „Gleichartigkeit“ der Bernoulli-Experimente bedeutet übrigens nur die stets gleiche Erfolgswahrscheinlichkeit!) Formal wird diese Kette durch unabhängig und identisch Bernoulli(p)-verteilte Zufallsvariablen $B_1, \ldots, B_n$ beschrieben.

Häufig interessiert man sich in diesem Szenario nicht für die konkrete Sequenz der Erfolge und Misserfolge, sondern nur für die Anzahl der Erfolge, also für $X := \sum_{i=1}^{n} B_i$, welches bekanntermaßen eine *Binomialverteilung* zu den Parametern n und p hat (auch Stichprobenumfang bzw. Erfolgswahrscheinlichkeit genannt); kurz: $X \sim \mathrm{Bin}(n, p)$.

In **R** lässt sich eine Bernoulli-Kette der Länge $n \geq 1$ zur Erfolgswahrscheinlichkeit $p \in [0, 1]$ (mit tatsächlich abgeschlossenem Intervall!) generieren mittels der Funktion

```
sample( x, size, replace = FALSE, prob = NULL)
```

Sie kann indes wesentlich mehr realisieren, nämlich das Ziehen einer Stichprobe des Umfangs `size` aus den Elementen von `x`, und zwar entweder <u>mit</u> Wiederholung (für `replace = TRUE`) oder <u>ohne</u> (für `replace = FALSE`, was die Voreinstellung ist), wobei das Argument `prob` die Ziehungswahrscheinlichkeiten für die einzelnen Elemente in `x` festlegt.

Das Modell der Bernoulli-Kette der Länge n zur Erfolgswahrscheinlichkeit p entspricht hierbei dem Ziehen einer Stichprobe des Umfangs `size = n` aus den 2 Elementen von `x = c( 0, 1)` mit `replace = TRUE` und `prob = c( 1-p, p)`. Beispiel:

```
> sample( x = c( 0, 1), size = 20, replace = TRUE, prob = c( 0.3, 0.7))
 [1] 1 1 1 1 1 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1
```

Bemerkung: Auf das n-fache Ziehen mit Wiederholungen aus einer Menge von $k \geq 3$ Elementen gehen wir in Abschnitt 9.5 im Zusammenhang mit der dort behandelten Multinomialverteilung ein. Das n-fache Ziehen ohne Wiederholungen aus einer Menge von $k \geq n$ Elementen (Stichwort: Hypergeometrische Verteilung) ist auch durch `sample()` realisierbar, wobei der Spezialfall $n = k$ eine zufällige Permutation der Elemente von `x` liefert. Für Details verweisen wir auf die Online-Hilfe (und zur Wiederholung der o. g. Modelle z. B. auf [43, Henze (2010)]).

9.2 Einstichprobenprobleme im Binomialmodell

Gelegentlich ist man daran interessiert, die in der Realität oft *unbekannte* Auftrittswahrscheinlichkeit p für ein gewisses Ereignis anhand einer Versuchsreihe unabhängiger Experimente, also mit Hilfe einer Bernoulli-Kette zu beurteilen. Es könnte dies, wie in dem weiter unten folgenden Spielbeispiel, die Chance auf „Rot“ im Roulette sein. Jedoch auch das Risiko einer Komplikation bei einem medizinischen Eingriff ist ein solches p. In beiden Fällen ist man z. B. an Konfidenzintervallen für das unbekannte p interessiert oder möchte eine ein- bzw. zweiseitige Hypothese über die „Erfolgs“-Wahrscheinlichkeit p in einem Binomialmodell zum Stichprobenumfang n testen.

9.2.1 Der exakte Test für die Auftrittswahrscheinlichkeit p: `binom.test()`

In diesem Abschnitt wollen wir uns mit dem exakten Einstichproben-Binomialtest beschäftigen und rekapitulieren zunächst die Testvorschrift:

<u>Annahmen:</u> $X \sim \mathrm{Bin}(n, p)$ mit bekanntem n und unbekanntem p.
<u>Zu testen:</u>

$$H_0 : p = p_0 \qquad \text{gegen} \qquad H_1 : p \neq p_0 \qquad \text{zum Signifikanzniveau } \alpha$$

bzw.

$$H_0' : p \overset{(\leq)}{\geq} p_0 \qquad \text{gegen} \qquad H_1' : p \overset{(>)}{<} p_0 \qquad \text{zum Signifikanzniveau } \alpha.$$

<u>Teststatistik:</u>

$$X \sim \mathrm{Bin}(n, p_0) \qquad \text{unter } H_0 \text{ bzw. unter } p = p_0, \text{ was der „Rand“ von } H_0' \text{ ist.}$$

<u>Entscheidungsregel</u> für ein *konkretes* Datum $x \in \{0, \ldots, n\}$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(')} \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha$$

Die Berechnung des p-Wertes für H_0 kann gemäß der “point probability”-Methode (wie z. B. in [32, Fleiss et al. (2003)], ch. 2.7.1 beschrieben) erfolgen:

$$p\text{-Wert} = \sum_{k \in K(x)} \mathbb{P}(\mathrm{Bin}(n, p_0) = k),$$

wobei $\quad K(x) = \{k \in \{0, \ldots, n\} : \mathbb{P}(\mathrm{Bin}(n, p_0) = k) \leq \mathbb{P}(\mathrm{Bin}(n, p_0) = x)\}$.
D. h., es wird über all diejenigen $0 \leq k \leq n$ summiert, deren Auftrittswahrscheinlichkeiten unter H_0 nicht größer sind als diejenige des beobachteten Wertes x. (Es gibt auch andere Verfahren, die ebenfalls in [32, Fleiss et al. (2003)] diskutiert werden: die “tail probability”-Methode in ch. 2.7.2 und die “likelihood ratio”-Methode in ch. 2.7.3.)

Berechnung des p-Wertes für H_0': $\qquad p\text{-Wert} := \mathbb{P}\left(\mathrm{Bin}(n, p_0) \overset{(\geq)}{\leq} x\right)$

Beispiel: In einem fairen Roulette-Spiel ist die Wahrscheinlichkeit für „Rot“ in jedem Durchgang konstant gleich $p_0 = 18/37$, da es 18 rote und 18 schwarze Zahlen sowie die grüne Null gibt. Angenommen, beim Besuch eines gewissen Casinos wurde in $n = 100$ Spielen 42-mal „Rot“ beobachtet. Um herauszufinden, ob in diesem Fall möglicherweise ein gezinktes Roulette-Rad verwendet wurde, ist (z. B.) die Hypothese $H_0 : p = 18/37$ gegen die Alternative $H_1 : p \neq 18/37$ zu testen. Dies kann in **R** mit der Funktion `binom.test()` wie folgt geschehen (die außerdem ein Konfidenzintervall für p liefert):

Der exakte Einstichprobentest für den Binomialparameter p	

`> binom.test( x = 42, n = 100, p = 18/37)`
`  Exact binomial test`
 	
`data:  42 and 100`
`number of successes = 42, number of trials = 100, p-value = 0.1943`
`alternative hypothesis: true probability of success is not equal to 0.4864865`
`95 percent confidence interval:`
`  0.3219855 0.5228808`
`sample estimates:`
`probability of success`
`  0.42`
	Exakter Einstichproben-Binomialtest für $H_0 : p = p_0$ bei `x` $= 42$ „Erfolgen“ unter `n` $= 100$ „Versuchen“, wobei an `p` das hypothetisierte p_0 übergeben wird, zu lesen als $H_0 : \mathtt{p} = p_0$; hier mit $p_0 = 18/37$.
	Resultate: p-Wert (`p-value`), Alternative (Voreinstellung: zweiseitig, $H_1 : p \neq p_0$), 95 % Clopper-Pearson-Konfidenzintervall für p (siehe hierzu aber Punkt 3 auf Seite 171 in §9.2.3), Schätzwert x/n für p.

`> binom.test( 42, 100)`
`  Exact binomial test`
 	
`data:  42 and 100`
`number of successes = 42, number of trials = 100, p-value = 0.1332`
`alternative hypothesis: true probability of success is not equal to 0.5`
`95 percent confidence interval:`
`  0.3219855 0.5228808`
`sample estimates:`
`probability of success`
`  0.42`
	Ohne Angabe von `p` wird $p_0 \equiv 0.5$ gesetzt.
	Resultate: Analog zu obigem.
	
	Mit `conf.level` ließe sich das Konfidenzniveau und mit `alternative` die Alternative spezifizieren: `alternative = "greater"` ($H_1' : p > p_0$) oder `alter = "less"` ($H_1' : p < p_0$).

9.2.2 Der approximative Test für p: `prop.test()`

Der Zentrale Grenzwertsatz (speziell der von de Moivre-Laplace) erlaubt für „hinreichend großes“ n die Approximation der $\mathrm{Bin}(n, p)$-Verteilung durch eine Normalverteilung, sodass auch ein approximativer Einstichproben-Binomialtest zur Verfügung steht:

<u>Annahmen</u> und <u>zu testende</u> Hypothesen wie im exakten Einstichproben-Binomialtest.
<u>Teststatistik:</u>

$$Z := \frac{X - np_0}{\sqrt{np_0(1-p_0)}} \overset{n\to\infty}{\longrightarrow} \mathcal{N}(0,1) \qquad \text{in Verteilung unter } H_0 \text{ bzw. unter } p = p_0\text{, was der „Rand“ von } H_0' \text{ ist.} \tag{17}$$

<u>Entscheidungsregel</u> für ein *konkretes* Datum x auf Basis des p-Wertes:

$$\text{Verwirf } H_0^{(\prime)} \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei

$$p\text{-Wert} = \begin{cases} 2\,(1 - \Phi(|z|)) & \text{für} \quad H_1 : p \neq p_0; \\ \Phi(z) & \text{"} \quad H_1' : p < p_0; \\ 1 - \Phi(z) & \text{"} \quad H_1' : p > p_0, \end{cases}$$

und Φ die Standardnormalverteilungsfunktion ist.

Bemerkungen: Als notwendig zu erfüllende Bedingung für die Zulässigkeit dieses approximativen Tests wird häufig $np_0(1-p_0) > 9$ angegeben. Unter gewissen Bedingungen, zumal bei zu kleinem Stichprobenumfang, wird eine „Stetigkeitskorrektur“ nach Yates für die an sich diskret verteilte Statistik Z empfohlen, um die Approximation durch die stetige (!) Normalverteilung zu verbessern. (Darauf gehen wir hier nicht ein, verweisen aber auf die Diskussion dieser Thematik im Zusammenhang mit Stetigkeitskorrekturen für Wilcoxons Vorzeichen-Rangsummenstatistik auf Seite 125. Nützliche Informationen dazu finden sich auch in ch. 2.4.1 von [32, Fleiss et al. (2003)].)

Das Quadrat der im vorliegenden Fall verwendeten asymptotisch normalverteilten Teststatistik Z stimmt mit der sogenannten X^2-Statistik von Pearson (im Einstichprobenfall) überein, die unter H_0 asymptotisch χ^2_1-verteilt ist. Sie ist ein Spezialfall der im Abschnitt 9.3 genauer beschriebenen Pearsonschen X^2-Statistik für k Stichproben und wird von der im Folgenden vorgestellten **R**-Funktion daher auch für $k = 1$ verwendet.

Den approximativen Test auf Basis der unter H_0 asymptotisch normalverteilten Statistik Z samt einem approximativen Konfidenzintervall für p liefert die Funktion `prop.test()`:

Der approximative Einstichprobentest für den Binomialparameter p	

`> prop.test( x = 42, n = 100)`
`  1-sample proportions test with`
`  continuity correction`
 	
`data: 42 out of 100,`
`null probability 0.5, X-squared = 2.25,`
`df = 1, p-value= 0.1336`
`alternative hypothesis: true P(success)`
`in Group 1 is not equal to 0.5`
`95 percent confidence interval:`
`  0.3233236 0.5228954`
`sample estimates:`
`  p`
`0.42`
	Approximativer Einstichprobentest im Binomialmodell für $H_0 : p = 0.5$ (datenabhängig mit oder ohne Stetigkeitskorrektur in der Teststatistik).
	Resultate: Hypothetisierter Wert p_0 (`null probability`, Voreinstellung: $p_0 \equiv 0.5$), Teststatistik Z^2 (`X-squared`, asymptotisch χ^2_1-verteilt), ihre Freiheitsgrade (`df`), p-Wert (`p-value`), Alternative (Voreinstellung: zweiseitig, $H_1 : p \neq 0.5$), approximatives 95 %-Konfidenzintervall für p, relative „Erfolgs“-Häufigkeit x/n als Schätzer für p.

`> prop.test( 42, 100, p = 18/37)`
`  1-sample proportions test with`
`  continuity correction`
 	
`data: 42 out of 100,`
`null probab. 18/37, X-squared = 1.513,`
`df = 1, p-value= 0.2186`
`alternative hypothesis: true P(success)`
`in Group 1 is not equal to 0.4864865`
. . . . (Rest wie oben)
	Bei Angabe von `p` wird p_0 auf dessen Wert gesetzt.
	Resultate: Analog zu obigem.
	
	Mit `conf.level` bzw. `alternative` lassen sich Konfidenzniveau und Alternative spezifizieren: `alternative = "greater"` ($H_1' : p > p_0$) oder `alter = "less"` ($H_1' : p < p_0$).

9.2.3 Konfidenzintervalle für p

Das Binomialmodell $X \sim \mathrm{Bin}(n, p)$ ist einerseits eines der einfachsten und andererseits eines der am häufigsten verwendeten Modelle in der Statistik. Erstaunlicherweise birgt die Schätzung des Parameters p durch Konfidenzintervalle eine erhebliche Komplexität. Dies liegt im diskreten Charakter des Problems begründet.

Mit der asymptotischen Normalverteiltheit der Statistik Z in (17) steht zwar ein approximatives

asymptotisches $(1-\alpha)$-Konfidenzintervall für p via

$$\hat{p} \pm u_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \qquad \text{— das } \underline{\text{Standard-Konfidenzintervall}} \tag{18}$$

zur Verfügung, wobei $\hat{p} = X/n$ ist. Dieses Konfidenzintervall zeigt jedoch *erhebliche* Schwächen, da

1. seine tatsächliche Überdeckungswahrscheinlichkeit für Werte von p nahe bei 0 oder nahe bei 1 das nominelle Niveau $1-\alpha$ weit unterschreiten kann, egal wie groß n ist;
2. selbst für Werte von p im „mittleren“ Bereich und große n die tatsächliche Überdeckungswahrscheinlichkeit klar unter dem nominellen Niveau $1-\alpha$ liegen kann;
3. die in der Anwendungsliteratur angegebenen Faustformeln bezüglich der Zulässigkeit des Standard-Konfidenzintervalls höchst zweifel- bis fehlerhaft sind.

Die Arbeit “Interval Estimation for a Binomial Proportion” von L. D. Brown, T. T. Cai und A. DasGupta [12, Brown et al. (2001)], kurz BCD, enthält eine hervorragende Darstellung und Diskussion der Problematik, deren Fazit nur sein kann:

„Finger weg vom Standard-Konfidenzintervall!“

Wir versuchen hier, die Ergebnisse und alternativen Empfehlungen von BCD ganz knapp zusammenzufassen:

1. In der Literatur wird eine Reihe von verschiedenen Bedingungen genannt, unter denen die Verwendung des Standard-Konfidenzintervalls (18) angeblich zulässig sein soll. Die folgenden sechs Bedingungen sind eine typische Auswahl:

   (a) $np \geq 5$ und $n(1-p) \geq 5$ (oder ≥ 10)
   (b) $np(1-p) \geq 5$ (oder ≥ 10)
   (c) $n\hat{p}$, $n(1-\hat{p}) \geq 5$ (oder ≥ 10)
   (d) $\hat{p} \pm 3\sqrt{\hat{p}(1-\hat{p})/n}$ enthält weder die 0 noch die 1.
   (e) n ziemlich groß
   (f) $n \geq 50$, sofern p nicht sehr klein ist.

   Sie sind jedoch höchst mangelhaft: (a) und (b) sind gar nicht überprüfbar, (c) und (d) sind datenabhängig und somit wieder mit einer (unbekannten!) Fehlerwahrscheinlichkeit behaftet, (e) ist schlicht nutzlos und (f) irreführend. Ihre Verwendung ist also riskant! (Siehe [12, BCD], p. 106.) Überdies hat das Standard-Konfidenzintervall auch die (oben unter 1. und 2.) erwähnten grundsätzlichen, mathematischen Defizite.

2. Laut BCD gibt es die folgenden, „empfehlenswerten“ Alternativen, wobei wir zur Abkürzung $\kappa := u_{1-\alpha/2}$ setzen:

   (a) Das <u>Wilson</u>-Intervall ([12, BCD], p. 107)

$$KI_W \quad : \quad \frac{n}{n+\kappa^2}\left\{\hat{p} + \frac{\kappa^2}{2n} \pm \kappa\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \left(\frac{\kappa}{2n}\right)^2}\right\} \tag{19}$$

   oder (nach leichter Umformung) auch

$$KI_W \quad : \quad \frac{n\hat{p} + \kappa^2/2}{n+\kappa^2} \pm \frac{\kappa}{n+\kappa^2}\sqrt{n\hat{p}(1-\hat{p}) + \frac{\kappa^2}{4}} \tag{20}$$

ist ein *approximatives* Konfidenzintervall für p zum Niveau $1-\alpha$. (Es ergibt sich durch Lösen der in p_0 quadratischen Gleichung $Z^2 = \Phi^2(1-\alpha/2)$ mit Z aus (17) und $n\hat{p} = X$. Eine Herleitung findet sich z. B. in [43, Henze (2010)], §27.7 oder in [32, Fleiss et al. (2003)], ch. 2.4.2, jeweils sogar für die mit Stetigkeitskorrektur versehene Version von Z.)

(b) Das Agresti-Coull-Intervall ([12, BCD], p. 108)

$$KI_{AC} \quad : \quad \tilde{p} \pm \kappa\sqrt{\frac{\tilde{p}(1-\tilde{p})}{n+\kappa^2}} \tag{21}$$

mit $\tilde{p} := \dfrac{X+\kappa^2/2}{n+\kappa^2}$ ist ein *approximatives* $(1-\alpha)$-KI für p.
Für $\alpha = 0.05$ und somit $\kappa = 1.96 \approx 2$ gibt es zu diesem KI eine „Eselsbrücke“: Nimm 2 zusätzliche Erfolge und 2 zusätzliche Misserfolge und dann das Standard-Konfidenzintervall.

(c) Das Jeffreys-Intervall ([12, BCD], p. 108)

$$KI_J \quad : \quad [l(X), u(X)] \tag{22}$$

mit $l(0) = 0$ und $u(n) = 1$ sowie

$$l(x) = \beta_{x+\frac{1}{2},\ n-x+\frac{1}{2};\ \alpha/2} \qquad \text{und} \qquad u(x) = \beta_{x+\frac{1}{2},\ n-x+\frac{1}{2};\ 1-\alpha/2}$$

ist ein *approximatives* $(1-\alpha)$-KI für p, wobei $\beta_{r,s;\gamma}$ das γ-Quantil der (zentralen) $\beta(r,s)$-Verteilung ist.

Die Empfehlung von BCD lautet auf das Wilson- oder das Jeffreys-Konfidenzintervall, falls $n \leq 40$. Falls $n > 40$ ist, seien alle drei (Wilson, Agresti-Coull, Jeffreys) vergleichbar gut.

Es gibt sogar noch weiter verbessernde Modifikationen der obigen Konfidenzintervalle, auf die wir hier jedoch nicht eingehen (vgl. [12, BCD], pp. 112 - 113).

3. Andere, häufig genannte Alternativen sind die Clopper-Pearson-, die Arcus-Sinus-, die Anscombe- und die Logit-Konfidenzintervalle, deren Diskussion wir uns hier sparen und dafür auf BCD verweisen. Als einzige Bewertung halten wir fest, dass die Clopper-Pearson-Konfidenzintervalle als viel zu konservativ (d. h. das Niveau $1-\alpha$ *über*erfüllend) gelten und die übrigen drei (Arcus-Sinus, Anscombe bzw. Logit) hinsichtlich anderer Kriterien (wie „mittlere absolute Überdeckungswahrscheinlichkeit“ oder „erwartete Konfidenzintervall-Länge“) unterliegen.

Außer dem *nicht* zu empfehlenden Clopper-Pearson-Konfidenzintervall in `binom.test()` (siehe Abschnitt 9.2.1) und dem ebenso *nicht* zu empfehlenden Standard-Konfidenzintervall (18) in `prop.test()` (vgl. §9.2.2), ist keines der obigen Konfidenzintervalle ohne Weiteres in “base **R**” verfügbar. Sie müssen entweder in Zusatz-Paketen wie z. B. `binom`, `exactci`, `ExactNumCI`, `PropCI` und evtl. weiteren gefunden werden oder sind eben selbst in **R** zu programmieren (siehe Übungen).

9.2.4 Hinweis zur Fallzahlschätzung für Konfidenzintervalle für p

In [32, Fleiss et al. (2003), §2.5.2] werden eine Formel und Erläuterungen zur Fallzahlschätzung für verschiedene Konfidenzintervalle aufgeführt, und zwar sowohl für das Standard-Konfidenzintervall (18) und für selbiges mit einer Stetigkeitskorrektur als auch für das Wilson-Konfidenzintervall (20), allerdings sogar ergänzt um eine Stetigkeitskorrektur.

9.3 Mehrstichprobentests im Binomialmodell

Häufig hat man mehrere, unabhängige Versuchsreihen, mit denen zum Beispiel die Wirksamkeiten, etwa im Sinne von Heilungserfolgen, verschiedener Behandlungen verglichen werden sollen. Dieses Problem läuft darauf hinaus, die Erfolgswahrscheinlichkeiten (möglicherweise) verschiedener Binomialmodelle zu vergleichen: In $k \geq 2$ Binomialmodellen zu beliebigen Stichprobenumfängen $n_1, \ldots, n_k$ lässt sich zum einen ein Test auf Gleichheit der (unspezifizierten) Erfolgsparameter $p_1, \ldots, p_k$ durchführen, wobei im Spezialfall $k = 2$ auch ein einseitiger Test möglich ist. Zum anderen ist ein Test der Erfolgsparameter $p_1, \ldots, p_k$ auf Gleichheit mit vollständig spezifizierten Werten $p_{01}, \ldots, p_{0k}$ möglich. Letzteren Test nennt man auch einen Anpassungstest (Engl.: "goodness-of-fit test").

9.3.1 Zur Theorie der approximativen k-Stichproben-Binomialtests (Pearsons X^2-Tests)

Annahmen: $X_1, \ldots, X_k$ seien unabhängig binomialverteilt mit $X_j \sim \mathrm{Bin}(n_j, p_j)$ für $j = 1, \ldots, k$ mit $k \geq 2$, wobei die n_j bekannt und die $p_j \in (0,1)$ unbekannt sind.
Zu testen: Die "goodness-of-fit"-Hypothese mit vollständig spezifizierten Werten $p_{0j} \in (0,1)$:

$$H_{0A} : p_j = p_{0j} \text{ für alle } 1 \leq j \leq k \qquad \text{gegen} \qquad H_{1A} : p_j \neq p_{0j} \text{ für ein } 1 \leq j \leq k$$

bzw. die Hypothese der Gleichheit der (unspezifizierten) $p_1, \ldots, p_k$:

$$H_{0B} : p_1 = \ldots = p_k \ (=: p) \qquad \text{gegen} \qquad H_{1B} : p_i \neq p_j \text{ für ein Paar } 1 \leq i \neq j \leq k.$$

Teststatistiken:

$$X_A^2 := \sum_{j=1}^{k} \frac{(X_j - n_j p_{0j})^2}{n_j p_{0j}(1 - p_{0j})} \longrightarrow \chi_k^2 \quad \text{in Verteilung unter } H_{0A}, \text{ falls } \min_{1 \leq j \leq k} n_j \to \infty$$

bzw.

$$X_B^2 := \sum_{j=1}^{k} \frac{\left(X_j - n_j \bar{X}_.\right)^2}{n_j \bar{X}_. \left(1 - \bar{X}_.\right)} \longrightarrow \chi_{k-1}^2 \quad \text{in Verteilung unter } H_{0B}, \text{ falls } \min_{1 \leq j \leq k} n_j \to \infty,$$

wobei $\bar{X}_. \equiv N^{-1} \sum_{j=1}^{k} X_j$ mit $N := \sum_{j=1}^{k} n_j$ und $n_j/N \longrightarrow \lambda_j \in (0,1)$ für $N \to \infty$.

Entscheidungsregel für konkrete Daten $x_1, \ldots, x_k$ auf Basis des p-Wertes:

$$\text{Verwirf } H_{0A} \iff p\text{-Wert für } H_{0A} \leq \alpha \quad \text{bzw.} \quad \text{verwirf } H_{0B} \iff p\text{-Wert für } H_{0B} \leq \alpha,$$

wobei $\quad p\text{-Wert für } H_{0A} = 1 - F_{\chi_k^2}(x_A^2) \quad$ bzw. $\quad p\text{-Wert für } H_{0B} = 1 - F_{\chi_{k-1}^2}(x_B^2)$
sowie $F_{\chi_\nu^2}$ die Verteilungsfunktion der χ^2-Verteilung mit ν Freiheitsgraden ist.

Bemerkungen:

- Für $k = 2$ ist auch der einseitige Test von $H'_{0B} : p_1 \overset{(\leq)}{\geq} p_2$ gegen $H'_{1B} : p_1 \overset{(>)}{<} p_2$ möglich.
  Teststatistik:

$$Z := \frac{X_1/n_1 - X_2/n_2}{\sqrt{\frac{n_1+n_2}{n_1 n_2} \bar{X}_. \left(1 - \bar{X}_.\right)}} \longrightarrow \mathcal{N}(0,1) \quad \begin{array}{l}\text{in Verteilung, falls } \min\{n_1, n_2\} \to \infty, \text{ un-} \\ \text{ter } p_1 = p_2, \text{ was der „Rand“ von } H'_{0B} \text{ ist,}\end{array} \tag{23}$$

  wobei $Z^2 = X_B^2$, wie sich (ausgehend von X_B^2) leicht zeigen lässt.
  Entscheidungsregel: $\quad$ Verwirf $H'_{0B} \iff p\text{-Wert} \leq \alpha, \qquad$ wobei

$$p\text{-Wert} = \begin{cases} \Phi(z) & \text{für} \quad H'_{1B} : p_1 < p_2; \\ 1 - \Phi(z) & \text{"} \quad H'_{1B} : p_1 > p_2. \end{cases}$$

- Die goodness-of-fit-Hypothese H_{0A} ist auch für $k = 1$ testbar, was identisch mit dem approximativen Einstichproben-Binomialtest von §9.2.2 ist. X_A^2 stimmt dann in der Tat mit dem Quadrat der dortigen Teststatistik Z überein (siehe (17) auf Seite 168).

- Sowohl X_A^2 als auch X_B^2 werden Pearsons X^2-Statistik genannt. Man begegnet ihr häufig in der suggestiven Form $X^2 = \sum_l (\text{observed}_l - \text{expected}_l)^2/\text{expected}_l$, die zu den obigen Darstellungen aufgrund der unterschiedlichen Nenner im Widerspruch zu stehen scheint. Dies klärt sich jedoch dadurch auf, dass in dieser suggestiven Form auch die „redundanten" Zufallsvariablen $n_j - X_j \sim \text{Bin}(n_j, 1 - p_j)$ in die Summation einbezogen werden: Für H_{0A} ist
$$X^2 = \sum_{l=1}^{k} \frac{(X_l - n_l p_{0l})^2}{n_l p_{0l}} + \sum_{l=1}^{k} \frac{(n_l - X_l - n_l(1 - p_{0l}))^2}{n_l(1 - p_{0l})},$$
was sich durch geeignetes Zusammenfassen zu X_A^2 vereinfachen lässt: Für jedes $1 \le l \le k$ und $\theta_l \in \{p_{0l}, \bar{X}.\}$ ist nämlich offenbar $(X_l - n_l\theta_l)^2/(n_l\theta_l) + (n_l - X_l - n_l(1 - \theta_l))^2/n_l(1 - \theta_l) = (X_l - n_l\theta_l)^2/(n_l\theta_l(1 - \theta_l))$.

  Für H_{0B} verhält es sich analog mit $\bar{X}.$ anstelle von p_{0j} in allen Summanden von X^2. (**R**-intern werden X_A^2 und X_B^2 in der Form von X^2 berechnet.)

- Wir skizzieren hier den Beweis der obigen Verteilungsaussage für X_A^2: Aufgrund des multivariaten Zentralen Grenzwertsatzes (ZGWS) und der Unabhängigkeit der X_j gilt:
$$\widetilde{\mathbf{X}}_{\mathbf{n}} := \left(\frac{X_1 - n_1 p_{01}}{\sqrt{n_1}}, \ldots, \frac{X_k - n_k p_{0k}}{\sqrt{n_k}}\right)' \longrightarrow \widetilde{\mathbf{X}} \sim \mathcal{N}_k(\mathbf{0}_k, \widetilde{\mathbf{\Sigma}}) \quad \text{in Verteilung unter } H_{0A}, \text{ falls } \mathbf{n} \to \infty,$$
wobei $\widetilde{\mathbf{\Sigma}} \equiv \text{diag}(p_{01}(1 - p_{01}), \ldots, p_{0k}(1 - p_{0k}))$ regulär (und somit invertierbar) ist, $\mathbf{n} \equiv (n_1, \ldots, n_k)'$ und $\mathbf{n} \to \infty$ eine Abkürzung für $\min_{1 \le j \le k} n_j \to \infty$ sein soll. Gemäß der Verteilungstheorie für quadratische Formen (siehe z. B. [58, Mathai & Provost (1992)]) folgt:
$$X_A^2 = \widetilde{\mathbf{X}}_{\mathbf{n}}' \widetilde{\mathbf{\Sigma}}^{-1} \widetilde{\mathbf{X}}_{\mathbf{n}} \longrightarrow \widetilde{\mathbf{X}} \widetilde{\mathbf{\Sigma}}^{-1} \widetilde{\mathbf{X}} \sim \chi_k^2 \quad \text{in Verteilung unter } H_{0A} \text{ für } \mathbf{n} \to \infty$$

- Die Beweisskizze der Verteilungsaussage für X_B^2: Der multivariate ZGWS und die Unabhängigkeit der X_j garantieren
$$\widetilde{\mathbf{X}}_{\mathbf{n}} := \left(\frac{X_1 - n_1 p}{\sqrt{n_1}}, \ldots, \frac{X_k - n_k p}{\sqrt{n_k}}\right)' \longrightarrow \widetilde{\mathbf{X}} \sim \mathcal{N}_k(\mathbf{0}_k, p(1 - p)\mathbf{I}_k) \quad \text{unter } H_{0B}$$
in Verteilung, falls $\mathbf{n} \to \infty$. „Etwas" Matrix-Arithmetik liefert ferner:
$$X_B^2 = \frac{\widetilde{\mathbf{X}}_{\mathbf{n}}' \mathbf{P}_{\mathbf{n}} \widetilde{\mathbf{X}}_{\mathbf{n}}}{\bar{X}.(1 - \bar{X}.)} \qquad \text{für } \mathbf{P}_{\mathbf{n}} = \mathbf{I}_k - \frac{1}{N}\sqrt{\mathbf{n}}(\sqrt{\mathbf{n}})',$$
wobei $\sqrt{\mathbf{n}} \equiv (\sqrt{n_1}, \ldots, \sqrt{n_k})'$ zu lesen sei und $\mathbf{P}_{\mathbf{n}}$ symmetrisch sowie idempotent ist. Nach Voraussetzung konvergiert jedes n_j/N gegen ein $0 < \lambda_j < 1$ für $\mathbf{n} \to \infty$, sodass $\mathbf{P}_{\mathbf{n}}$ gegen $\mathbf{P}_{\underline{\lambda}} = I_k - \sqrt{\underline{\lambda}}(\sqrt{\underline{\lambda}})'$ konvergiert mit $\underline{\lambda} \equiv (\lambda_1, \ldots, \lambda_k)'$ und $\underline{\lambda}' 1_k = 1$. Offenbar ist dann $\mathbf{P}_{\underline{\lambda}}$ ebenfalls symmetrisch und idempotent; außerdem ist $\text{Spur}(\mathbf{P}_{\underline{\lambda}}) = k - 1$. Die letzten drei Eigenschaften garantieren, dass auch $\text{Rang}(\mathbf{P}_{\underline{\lambda}}) = k - 1$ ist. Schließlich impliziert die Verteilungstheorie für quadratische Formen (vgl. [58, Mathai & Provost (1992)]) zusammen mit Cramér-Slutzky, dass
$$X_B^2 \longrightarrow \frac{\widetilde{\mathbf{X}}' \mathbf{P}_{\underline{\lambda}} \widetilde{\mathbf{X}}}{p(1 - p)} \sim \chi_{k-1}^2 \quad \text{in Verteilung unter } H_{0B} \text{ für } \mathbf{n} \to \infty$$

- Damit die Approximation durch die χ^2-Verteilung für endliche n_j akzeptabel ist, sollten **alle** erwarteten Häufigkeiten $n_j p_{0j}$ und $n_j(1 - p_{0j})$ bzw. geschätzten erwarteten Häufigkeiten $n_j \bar{X}.$ und $n_j(1 - \bar{X}.)$ mindestens 5 sein.

  Überdies wird auch hier unter gewissen Bedingungen eine „Stetigkeitskorrektur" für die an sich diskret verteilten Statistiken vorgenommen, um die Approximation durch die stetige (!) χ^2-Verteilung zu verbessern; wir verweisen hierzu auf die Literatur. `prop.test()` verwendet für $k > 2$ nie eine Stetigkeitskorrektur.

9.3.2 Zur Implementation der k-Stichproben-Binomialtests: `prop.test()`

Hier wenden wir uns der Umsetzung der obigen Theorie in **R** zu. Zunächst für den Fall zweier Stichproben, dann für mindestens drei Stichproben.

9.3.2.1 Der Fall $k = 2$ Stichproben

Dazu ziehen wir das **Beispiel** der sogenannten „Salk Polio-Impfstudie“ (aus den frühen 1950er Jahren) heran. In dieser randomisierten Studie erhielten eine Gruppe von 200745 Personen einen Polio-Impfstoff und eine andere Gruppe von 201229 Personen ein Placebo verabreicht. In der ersten Gruppe traten 57 Fälle von Polio auf, wohingegen es in der zweiten 142 Fälle waren. Folgende Tabelle fasst die Ergebnisse zusammen:

Behandlung	**Poliofälle**	**Gruppenstärke**
Impfstoff	57	200745
Placebo	142	201229

Unter der Annahme, dass es sich in den Gruppen jeweils um ein Binomialmodell mit „Erfolgs“-Parameter für Polio p_1 bzw. p_2 handelt, sind wir daran interessiert, herauszufinden, ob die beiden Parameter gleich sind, oder ob (aufgrund der Impfung) ein Unterschied zwischen den Wahrscheinlichkeiten besteht, Polio zu bekommen. Dies ist ein Test der Hypothese H_{0B}.

Es kann aber auch von Interesse sein, die beobachteten relativen Häufigkeiten mit vollständig spezifizierten „Erfolgs“-Wahrscheinlichkeiten zu vergleichen (deren Werte eventuell aus früheren Untersuchungen hervorgegangen sind oder von Medikamentenherstellern behauptet werden). D. h., p_1 und p_2 sind auf Gleichheit mit p_{01} bzw. p_{02} zu testen, beispielsweise für $p_{01} = 0.0002$ und $p_{02} = 0.0006$, was ein Test der Hypothese H_{0A} ist.

In beiden Fällen handelt es sich jeweils um einen Zweistichprobentest für Binomialmodelle, der mithilfe der Funktion `prop.test()` durchgeführt werden kann. Zunächst bereiten wir die Daten so auf, dass sie in die Testfunktion eingegeben werden können:

```
> erfolge <- c( 57, 142);     versuche <- c( 200745, 201229)
```

Approximative Zweistichprobentests für die Binomialparameter p_1, p_2	

`> prop.test( erfolge, versuche)`
 	
`  2-sample test for equality of pro-`
`  portions with continuity correction`
 	
`data:  erfolge out of versuche`
`X-squared = 35.2728, df = 1,`
`p-value = 2.866e-09`
`alternative hypothesis: two.sided`
`95 percent confidence interval:`
`  -0.0005641508 -0.0002792920`
`sample estimates:`
`  prop 1  prop 2`
`0.0002839423 0.0007056637`
	Approximativer Zweistichprobentest im Binomialmodell für $H_{0B} : p_1 = p_2$ (datenabhängig mit oder ohne Stetigkeitskorrektur in der Teststatistik).
	Resultate: Teststatistik X_B^2 (`X-squared`, asymptotisch χ_1^2-verteilt), ihre Freiheitsgrade (`df`), p-Wert (`p-value`), Alternative (Voreinstellung: zweiseitig, $H_{1B} : p_1 \neq p_2$), approximatives 95 %-Konfidenzintervall für $p_1 - p_2$, relative „Erfolgs“-Häufigkeiten $\hat{p}_1$ und $\hat{p}_2$ als Schätzer (`sample estimates`) für p_1 bzw. p_2. Mit `conf.level` bzw. `alternative` lassen sich Konfidenzniveau und Alternative wählen: `alternative = "greater"` ($H'_{1B} : p_1 > p_2$) oder `alter = "less"` ($H'_{1B} : p_1 < p_2$).

`> prop.test( erfolge, versuche,`
`+ p = c( 0.0002, 0.0006))`
 	
`  2-sample test for given propor-`
`  tions with continuity correction`
	Approximativer Zweistichproben-Anpassungstest im Binomialmodell für $H_{0A} : (p_1, p_2) = (p_{01}, p_{02})$ (datenabhängig mit oder ohne Stetigkeitskorrektur in der Teststatistik).

`data:  erfolge out of versuche,`
`null probabilities c(2e-04, 6e-04)`
`X-squared = 10.233, df = 2,`
`p-value = 0.005997`
`alternative hypothesis: two.sided`
`null values:`
`prop 1 prop 2`
` 2e-04  6e-04`
`sample estimates:`
`  prop 1  prop 2`
` 0.0002839423 0.0007056637`
	Resultate: Hypothetisierte Werte (p_{01}, p_{02}) (`null probabilities`), Teststatistik X_A^2 (`X-squared`, asymptotisch χ_2^2-verteilt), ihre Freiheitsgrade (`df`), p-Wert (`p-value`), Alternative (nur zweiseitig möglich: H_{1A} : $(p_1, p_2) \neq (p_{01}, p_{02})$), nochmals die hypothetisierten Werte der Gruppen (`null values`), relative „Erfolgs"-Häufigkeiten $\hat{p}_1$ und $\hat{p}_2$ als Schätzer (`sample estimates`) für p_1 bzw. p_2.

9.3.2.2 Der Fall $k \geq 3$ Stichproben

Hierfür verwenden wir ein **Beispiel** von vier Lungenkrebsstudien (aus [32, Fleiss et al. (2003)], p. 189). Jede Studie umfasste eine Anzahl an Patienten (zweite Spalte in folgender Tabelle), unter denen sich jeweils eine gewisse Zahl an Rauchern (dritte Spalte der Tabelle) befand:

Studie	**Anzahl an Patienten**	**Anzahl der Raucher darunter**
1	86	83
2	93	90
3	136	129
4	82	70

Wir sind daran interessiert, herauszufinden, ob die Raucher in den vier Studien (statistisch gesehen) gleich häufig repräsentiert waren. Mit anderen Worten, ob die Wahrscheinlichkeit dafür, dass ein beliebiger Patient ein Raucher ist, in jeder der vier Studien dieselbe ist, d. h., dass die Patienten aus einer bzgl. des Rauchens homogenen (Super-)Population stammen. Unter der Annahme, dass es sich in den vier Studien jeweils um ein Binomialmodell handelt, läuft es darauf hinaus, zu testen, ob die „Erfolgs"-Parameter für „Raucher" $p_1, \ldots, p_4$ identisch sind.

Die beobachteten relativen Häufigkeiten können aber auch mit vollständig spezifizierten „Erfolgs"-Wahrscheinlichkeiten $p_{01}, \ldots, p_{04}$ verglichen werden. Zum Beispiel, dass in den ersten drei Studien die Wahrscheinlichkeit für „Raucher" 0.95 und in der vierten gleich 0.90 ist.

In beiden Fällen handelt es sich um einen k-Stichprobentest (hier mit $k = 4$) für Binomialmodelle, der wieder mit Hilfe der Funktion `prop.test()` durchgeführt wird. Die Aufbereitung der Daten, sodass sie in die Testfunktion eingegeben werden können, folgt:

```
> raucher <- c( 83, 90, 129, 70);     patienten <- c( 86, 93, 136, 82)
```

Approx. k-Stichprobentests für die Binomialparameter p_j, $j = 1, \ldots, k (\geq 3)$	

`> prop.test( raucher, patienten)`
 	
`  4-sample test for equality of propor-`
`  tions without continuity correction`
 	
`data:  raucher out of patienten`
`X-squared = 12.6004, df = 3,`
`p-value = 0.005585`
`alternative hypothesis: two.sided`
`sample estimates:`
`  prop 1  prop 2  prop 3  prop 4`
`0.9651163 0.9677419 0.9485294 0.8536585`
	Approximativer k-Stichprobentest im Binomialmodell für H_{0B} : $p_1 = \ldots = p_k$ (stets ohne Stetigkeitskorrektur).
	Resultate: Teststatistik X_B^2 (`X-squared`, asymptotisch χ_{k-1}^2-verteilt), ihre Freiheitsgrade (`df`), p-Wert (`p-value`), Alternative (nur zweiseitig möglich: H_{1B} : $p_i \neq p_j$ für ein Paar $i \neq j$), relative „Erfolgs"-Häufigkeiten $\hat{p}_j$ als Schätzer für p_j für $j = 1, \ldots, k$. Beachte: Für $k \geq 3$ werden *keine* Konfidenzintervalle für die p_j oder ihre paarweisen Differenzen berechnet.

`> prop.test( raucher, patienten,`
`+ p = c( 0.95, 0.95, 0.95, 0.9))`
 	
`  4-sample test for given proportions`
`  without continuity correction`
 	
`data:  raucher out of patienten, null`
`probabilities c(0.95, 0.95, 0.95, 0.9)`
`X-squared = 2.9928, df = 4,`
`p-value = 0.559`
`alternative hypothesis: two.sided`
`null values:`
`prop 1 prop 2 prop 3 prop 4`
`  0.95  0.95  0.95  0.90`
`sample estimates:`
`  prop 1  prop 2  prop 3  prop 4`
`0.9651163 0.9677419 0.9485294 0.8536585`
 	
`Warning message:`
`Chi-squared approximation may be`
`incorrect in: prop.test(raucher,`
`patienten, p = c(0.95, 0.95, 0.95, 0.9))`
	Approximativer k-Stichproben-Anpassungstest im Binomialmodell für H_{0A} : $p_j = p_{0j}$ für alle $j = 1, \ldots, k$ (stets ohne Stetigkeitskorrektur).
	Resultate: Hypothetisierte Werte in `p` (`null probabilities`), Teststatistik X_A^2 (`X-squared`, asymptotisch χ_k^2-verteilt), ihre Freiheitsgrade (`df`), p-Wert (`p-value`), Alternative (nur zweiseitig möglich: H_{1A} : $p_j \neq p_{0j}$ für ein j), nochmals die hypothetisierten Werte der Gruppen (`null values`) und schließlich die relativen „Erfolgs"-Häufigkeiten $\hat{p}_j$ (`sample estimates`) als Schätzer für p_j für $j = 1, \ldots, k$.
	Die Warnung resultiert daraus, dass (mindestens) eine der hypothetisierten Wahrscheinlichkeiten zu (mindestens) einer erwarteten Häufigkeit führt, die kleiner als 5 ist (und zwar hier für die Eigenschaft *kein* Raucher zu sein). Die χ^2-Approximation ist dann möglicherweise nicht mehr gültig.

9.4 Testgüte und Fallzahlschätzung im Binomialmodell

Im Binomialmodell werden Berechnungen im Zusammenhang mit der Güte und den Fallzahlen im Ein- oder Zweistichprobenproblem zur Vereinfachung oft auf die Normalapproximation der Binomialverteilung gestützt (vgl. (17) und (23)). Daher verlaufen diese ähnlich zu denen der entsprechenden Testszenarien des Normalverteilungsmodells. Wir präsentieren zu Referenzzwecken in knapper Form nur die Resultate. (Siehe hierzu z. B. auch [32, Fleiss et al. (2003)], ch. 2.5.1 für den Ein- und ch. 4.2 für den Zweistichprobenfall.)

9.4.1 Einseitiger und zweiseitiger Einstichprobentest

Wir betrachten das Modell $X \sim \mathrm{Bin}(n, p)$ und zur Abkürzung sei im Folgenden stets $q := 1 - p$ sowie $q_0 := 1 - p_0$. Für den approximativen **einseitigen** Niveau-α-Test von $H_0' : p \overset{(\leq)}{\geq} p_0$ gegen $H_1' : p \overset{(>)}{<} p_0$ mithilfe von (17) ergibt sich die folgende Gütefunktion:

$$1 - \beta = \mathbb{P}_{H_1'}\left(\frac{X - np_0}{\sqrt{np_0q_0}} \overset{(\geq +)}{\leq -} u_{1-\alpha}\right) \approx \Phi\left(\frac{\sqrt{n}|p_0 - p| - u_{1-\alpha}\sqrt{p_0q_0}}{\sqrt{pq}}\right) \tag{24}$$

Beachte, dass die rechte Seite von (24) für beide einseitige Tests dieselbe ist. (Hinter dem „≈" steckt die Normalapproximation nach korrekter Zentrierung und Normierung von X durch np bzw. $\sqrt{npq}$.)

Auflösen von (24) nach der Fallzahl n liefert eine untere Schranke n' für n, deren Einhalten dazu führt, dass die Testgüte approximativ (!) $1 - \beta$ ist:

$$n \geq \left(\frac{u_{1-\alpha}\sqrt{p_0q_0} + u_{1-\beta}\sqrt{pq}}{|p_0 - p|}\right)^2 \quad =: n' \tag{25}$$

Auflösen von (24) nach $\Delta := p_0 - p$ (über eine quadratische Gleichung) erlaubt die Abschätzung der kleinsten zu entdeckenden Differenz, was wir uns hier sparen.

Für den approximativen **zweiseitigen** Niveau-α-Test von $H_0 : p = p_0$ gegen $H_1 : p \neq p_0$ ergeben sich Abschätzungen, die exakt baugleich mit (24) und (25) sind und lediglich $u_{1-\alpha/2}$ anstelle von $u_{1-\alpha}$ enthalten:

$$\begin{aligned} 1-\beta &= \mathbb{P}_{H_1}\left(\frac{|X-np_0|}{\sqrt{np_0q_0}} \geq u_{1-\alpha/2}\right) \\ &\approx \Phi\left(\frac{-\sqrt{n}(p_0-p)-u_{1-\alpha/2}\sqrt{p_0q_0}}{\sqrt{pq}}\right) + \Phi\left(\frac{\sqrt{n}(p_0-p)-u_{1-\alpha/2}\sqrt{p_0q_0}}{\sqrt{pq}}\right) \\ &\geq \Phi\left(\frac{\sqrt{n}|p_0-p|-u_{1-\alpha/2}\sqrt{p_0q_0}}{\sqrt{pq}}\right) \end{aligned} \tag{26}$$

und infolgedessen

$$n \geq \left(\frac{u_{1-\alpha/2}\sqrt{p_0q_0}+u_{1-\beta}\sqrt{pq}}{|p_0-p|}\right)^2 \quad =: n' \tag{27}$$

Sowohl im einseitigen als auch im zweiseitigen Fall lässt sich die Qualität der obigen Approximationen verbessern, indem in den Normalapproximationen in (24) bzw. (26) eine Stetigkeitskorrektur verwendet wird. (Dazu kommt dort im Zähler des Arguments von Φ jeweils ein $-1/(2\sqrt{n})$ hinzu; für Details siehe z. B. [32, Fleiss et al. (2003)], ch. 2.5.1.) Dies führt zu der modifizierten Fallzahlabschätzung

$$n \geq \frac{n'}{4}\left(1+\sqrt{1+\frac{2}{n'|p_0-p|}}\right)^2, \tag{28}$$

wobei n' jeweils aus der „Vorschätzung“ (25) bzw. (27) stammt.

Bemerkungen: Auch die so erhaltenen Fallzahlgrenzen garantieren *nicht*, dass Niveau und Güte exakt eingehalten werden. Vielmehr wird α unter- und $1-\beta$ in der Regel überschritten. Das erscheint zwar wünschenswert, aber im „Ernstfall“ ist eine Überprüfung der Fallzahl mit Hilfe des exakten Tests ratsam, weil die tatsächlich benötigte Fallzahl etwas kleiner sein könnte als durch (28) angegeben.
Für das Einstichprobenproblem ist – meines Wissens – keine „fertige“ Funktion in der “base distribution” von **R** implementiert, mit der die Güte oder die Fallzahl bestimmt werden kann.

9.4.2 Einseitiger und zweiseitiger Zweistichprobentest: `power.prop.test()`

Hier beschränken wir uns auf das Modell $X_1 \sim \mathrm{Bin}(n, p_1)$ und $X_2 \sim \mathrm{Bin}(n, p_2)$ mit unabhängigen X_1 und X_2, also auf das Zweistichproben-Binomialproblem mit gleich großen Stichprobenumfängen $n_1 = n_2 =: n$. Zur Abkürzung sei auch hier im Folgenden stets

$$q_1 := 1-p_1, \qquad q_2 := 1-p_2, \qquad \bar{p} := \frac{p_1+p_2}{2} \quad \text{und} \quad \bar{q} := 1-\bar{p} = 1-\frac{p_1+p_2}{2}$$

In diesem Modell ist die unter $p_1 = p_2$ approximativ normalverteilte Teststatistik aus (23)

$$\frac{X_1-X_2}{\sqrt{2n\bar{X}_.\left(1-\bar{X}_.\right)}} \qquad \text{mit} \quad \bar{X}_. = \frac{X_1+X_2}{2n} \tag{29}$$

Zum approximativen **einseitigen** Niveau-α-Test von $H_0' : p_1 \overset{(\leq)}{\geq} p_2$ gegen $H_1' : p_1 \overset{(>)}{<} p_2$ mittels obiger Teststatistik gehört die Gütefunktion

$$1-\beta = \mathbb{P}_{H_1'}\left(\frac{X_1-X_2}{\sqrt{2n\bar{X}_.\left(1-\bar{X}_.\right)}} \overset{(\geq +)}{\leq -} u_{1-\alpha}\right) \approx \Phi\left(\frac{\sqrt{n}|p_1-p_2|-u_{1-\alpha}\sqrt{2\bar{p}\bar{q}}}{\sqrt{p_1q_1+p_2q_2}}\right), \tag{30}$$

wobei die rechte Seite von (30) für beide einseitige Tests dieselbe ist. (Hinter dem „≈“ steckt die korrekte Zentrierung und Normierung von $X_1 - X_2$ durch $n(p_1 - p_2)$ bzw. $\sqrt{n(p_1q_1 + p_2q_2)}$ als auch der Ersatz von $\bar{X}.$ durch $\bar{p}$.)

Auflösen von (30) nach der Fallzahl n liefert eine untere Schranke n' für n, deren Einhalten die Testgüte approximativ (!) $1 - \beta$ sein lässt:

$$n \geq \left(\frac{u_{1-\alpha}\sqrt{2\bar{p}\bar{q}} + u_{1-\beta}\sqrt{p_1q_1 + p_2q_2}}{|p_1 - p_2|} \right)^2 \quad =: n' \tag{31}$$

Beachte, dass n' nicht nur von $|p_1 - p_2|$ abhängt, sondern auch von p_1 und p_2 selbst und damit von den „Gegenden“ in $(0, 1)$, in denen sich p_1 und p_2 befinden.
Wir ersparen uns das Auflösen von (30) nach $\Delta := p_1 - p_2$ (was die Abschätzung der kleinsten zu entdeckenden Differenz erlaubte).

Für den approximativen **zweiseitigen** Niveau-α-Test von $H_0 : p_1 = p_2$ gegen $H_1 : p_1 \neq p_2$ auf Basis der obigen Teststatistik erhält man zu (30) und (31) baugleiche Ergebnisse für die Abschätzungen, nur mit $u_{1-\alpha/2}$ anstelle von $u_{1-\alpha}$:

$$\begin{aligned} 1 - \beta &= \mathbb{P}_{H_1}\left(\frac{|X_1 - X_2|}{\sqrt{2n\bar{X}.\left(1 - \bar{X}.\right)}} \geq u_{1-\alpha/2} \right) \\ &\approx \Phi\left(\frac{\sqrt{n}(p_1 - p_2) - u_{1-\alpha/2}\sqrt{2\bar{p}\bar{q}}}{\sqrt{p_1q_1 + p_2q_2}} \right) + \Phi\left(- \frac{\sqrt{n}(p_1 - p_2) + u_{1-\alpha/2}\sqrt{2\bar{p}\bar{q}}}{\sqrt{p_1q_1 + p_2q_2}} \right) \qquad (32) \\ &\geq \Phi\left(\frac{\sqrt{n}|p_1 - p_2| - u_{1-\alpha/2}\sqrt{2\bar{p}\bar{q}}}{\sqrt{p_1q_1 + p_2q_2}} \right) \qquad (33) \end{aligned}$$

und in Konsequenz

$$n \geq \left(\frac{u_{1-\alpha/2}\sqrt{2\bar{p}\bar{q}} + u_{1-\beta}\sqrt{p_1q_1 + p_2q_2}}{|p_1 - p_2|} \right)^2 \quad =: n' \tag{34}$$

Auch im Zweistichprobenproblem lässt sich sowohl im ein- als auch im zweiseitigen Fall die Approximationsqualität durch Verwendung einer Stetigkeitskorrektur in (30) bzw. (32) verbessern. (In diesem Fall kommt in jenen Gleichungen im Zähler des Arguments von Φ jeweils ein $-1/\sqrt{n}$ hinzu; Details sind wieder in [32, Fleiss et al. (2003)], ch. 4.2 in Verbindung mit ch. 3.1 und 3.3 zu finden.) Die hier resultierende modifizierte Fallzahlabschätzung ist

$$n \geq \frac{n'}{4}\left(1 + \sqrt{1 + \frac{4}{n'|p_1 - p_2|}} \right)^2 \tag{35}$$

mit dem n' aus der jeweiligen „Vorschätzung“ (31) bzw. (34).

Obige Berechnungen sind in der “base distribution” von **R** in der Funktion `power.prop.test()` implementiert (aber eben nur für das Zweistichprobenproblem und dabei mit gleich großen Stichprobenumfängen). Sie arbeitet nach einem ähnlichen Schema wie `power.t.test()` aus Abschnitt 8.12 (weswegen wir hier nicht allzu detalliert auf ihre konkrete Syntax eingehen, sondern auf die Erläuterungen in der Online-Hilfe verweisen). Die Argumente von `power.prop.test()` und ihre Voreinstellungswerte nebst den von uns für sie benutzten Bezeichnungen lauten wie folgt:

n	p_1	p_2	α	$1 - \beta$
`n = NULL`	`p1 = NULL`	`p2 = NULL`	`sig.level = 0.05`	`power = NULL`

H_1	„Exakt“
`alternative = "two.sided"`	`strict = FALSE`

Der/die Benutzer/in muss dafür sorgen, dass beim Aufruf von `power.prop.test()` genau vier der Parameter `n`, `p1`, `p2`, `sig.level` und `power` mit einem von `NULL` verschiedenen Wert versehen werden, damit der „fehlende“ Parameter dann aus den Werten der anderen abgeleitet wird. Beachte, dass `sig.level` als Voreinstellung *nicht* den Wert `NULL` hat, sodass für ihn explizit `NULL` anzugeben ist, wenn er berechnet werden soll. Mit `alternative` lässt sich H_1 als `"two.sided"` oder `"one.sided"` spezifizieren. Wird `strict = TRUE` angegeben, führt `power.prop.test()` alle Berechnungen auf Basis der „exakten Approximation“ (32) durch und nicht unter Verwendung der vereinfachenden Abschätzung (33). Die jeweiligen Parameter und Berechnungsergebnisse werden schließlich zusammen in einer Liste zurückgegeben.

Bemerkung: Im Package `Hmisc` stehen drei Funktionen namens `bpower()`, `bsamsize()` und `bpower.sim()` zur Verfügung, die auch ungleiche Stichprobenumfänge zulassen. Es ist aber grundsätzlich angeraten, für Details die Literatur zu konsultieren, wie z. B. [32, Fleiss et al. (2003)], ch. 4, speziell ch. 4.4.

Warnung: Aufgrund der in §9.2.3 genannten Probleme mit der Normalapproximation der Binomialverteilung raten wir zur Vorsicht vor der unkritischen Verwendung von `power.prop.test()` und empfehlen die Kontrolle der erhaltenen Fallzahl und der damit erzielten Werte für Niveau und Güte wie in den Bemerkungen auf Seite 177 für den einseitigen Test!

9.5 Tests im Multinomialmodell

Ein in der Praxis häufig anzutreffendes Szenario ist die Durchführung von n gleichartigen, voneinander unabhängigen Experimenten, mit jeweils genau $k \geq 2$ möglichen und sich gegenseitig ausschließenden Ergebnissen, die wir abstrakt „Treffer 1. Art“ bis „Treffer k. Art“ nennen wollen. Bezeichnen wir mit p_j, für $j = 1, \ldots, k$, die (konstante!) Wahrscheinlichkeit für einen Treffer j-ter Art in einem jeden solchen Experiment, dann hat der k-dimensionale Vektor $(H_{n1}, \ldots, H_{nk})'$ der absoluten Häufigkeiten der verschiedenen Treffer in n Experimenten eine Multinomialverteilung $\mathcal{M}(n; p_1, \ldots, p_k)$, wobei natürlich jedes H_{nj} ganzzahlig und nicht-negativ ist sowie $\sum_{j=1}^{k} p_j = 1$. Ein Standardbeispiel ist der n-fache Würfelwurf mit $k = 6$. (Für $k = 2$ und $p_1 = p$ ist $p_2 = 1 - p$ und wir erhalten offenbar die Binomialverteilung $\mathrm{Bin}(n, p)$.)

9.5.1 Multinomial-Experimente mit `sample()`

Wie bereits in Abschnitt 9.1 erwähnt, lässt sich mithilfe der Funktion

```
sample( x, size, replace = FALSE, prob = NULL)
```

auch das n-fache Ziehen mit Wiederholungen aus einer Menge von $k \geq 3$ Elementen und somit die Multinomialverteilung realisieren. Und zwar indem eine Stichprobe des Umfangs `size` aus den Elementen von `x` mit Wiederholung (also mit `replace = TRUE`) gezogen wird, wobei das Argument `prob` die Ziehungswahrscheinlichkeiten für die einzelnen Elemente in `x` festlegt. Der Vektor der beobachteten absoluten Häufigkeiten eines jeden der Elemente von `x` – also auch der *nicht* gezogenen `x`-Elemente – stellt dann den multinomialverteilten Vektor dar.

Im folgenden Beispiel ist `table( Z)` eine Realisierung aus der $\mathcal{M}(20; 0.2, 0.3, 0.49, 0.01)$-Verteilung. (Beachte, dass `table( Z)` nur die tatsächlich in `Z` auftretenden Werte auflistet, falls `Z` kein Faktor ist. Deshalb wird hier `x = factor( letters[ 1:4])` und nicht nur `x = letters[ 1:4]` in `sample()` verwendet, denn bei letzterem würde `table(Z)` den Wert `d` gar nicht aufführen.)

```
> (Z <- sample( x = factor( letters[ 1:4]), size = 20, replace = TRUE,
+               prob = c( 0.2, 0.3, 0.49, 0.01)))
 [1] b c a b c b a b c c a c c a c c b c c c
Levels: a b c d
> table( Z)
Z
 a  b  c  d
 4  5 11  0
```

9.5.2 Der approximative χ^2-Test im Multinomialmodell: `chisq.test()`

Oft ist n bekannt, aber der Vektor der Trefferwahrscheinlichkeiten $\mathbf{p} := (p_1, \ldots, p_k)'$ nicht, und man ist daran interessiert, eine Hypothese über $\mathbf{p}$ zu testen. Diese könnte zum Beispiel auf Gleichheit der Wahrscheinlichkeiten $p_1, \ldots, p_k$ mit vollständig spezifizierten Werten $p_{01}, \ldots, p_{0k}$ lauten, wobei natürlich $\sum_{j=1}^k p_{0j} = 1$ ist. Hiermit wird auch die Hypothese der Gleichheit der Wahrscheinlichkeiten $p_1, \ldots, p_k$ abgedeckt, da sie äquivalent ist zu $p_{0j} \equiv 1/k$ für alle $j = 1, \ldots, k$. Offenbar handelt es sich hier wieder um einen Anpassungstest (Engl.: "goodness-of-fit test").

Annahmen: $(H_{n1}, \ldots, H_{nk})' \sim \mathcal{M}(n; p_1, \ldots, p_k)$, wobei n bekannt und die $p_j \in (0,1)$ unbekannt sind (aber $\sum_{j=1}^k p_j = 1$ erfüllen).
Zu testen:

$$H_0 : p_j = p_{0j} \text{ für alle } 1 \le j \le k \qquad \text{gegen} \qquad H_1 : p_j \ne p_{0j} \text{ für ein } 1 \le j \le k$$

zum Niveau α mit $p_{0j} \in (0,1)$ für alle $1 \le j \le k$ und $\sum_{j=1}^k p_{0j} = 1$.

Teststatistik:

$$X^2 := \sum_{j=1}^k \frac{(H_{nj} - np_{0j})^2}{np_{0j}} \longrightarrow \chi^2_{k-1} \quad \text{in Verteilung unter } H_0, \text{ falls } n \to \infty.$$

Entscheidungsregel für einen konkreten Datenvektor $(h_{n1}, \ldots, h_{nk})'$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0 \qquad \Longleftrightarrow \qquad p\text{-Wert } \le \alpha,$$

wobei p-Wert $= 1 - F_{\chi^2_{k-1}}(x^2)$ und $F_{\chi^2_{k-1}}$ die Verteilungsfunktion der χ^2_{k-1}-Verteilung ist.

Bemerkung: Hier die Beweisskizze der Verteilungsaussage für obiges, ebenfalls Pearsons X^2-Statistik genanntes X^2: Der multivariate ZGWS impliziert

$$\widetilde{\mathbf{X}}_n := \left(\frac{H_{n1} - np_{01}}{\sqrt{n}}, \ldots, \frac{H_{n,k-1} - np_{0,k-1}}{\sqrt{n}} \right)' \longrightarrow \widetilde{\mathbf{X}} \sim \mathcal{N}_{k-1}(\mathbf{0}_{k-1}, \widetilde{\mathbf{\Sigma}}) \quad \text{unter } H_0$$

in Verteilung für $n \to \infty$, wobei $\widetilde{\mathbf{\Sigma}} = \operatorname{diag}(p_{01}, \ldots, p_{0,k-1}) - (p_{01}, \ldots, p_{0,k-1})'(p_{01}, \ldots, p_{0,k-1})$. Des Weiteren lässt sich leicht nachrechnen, dass sowohl

$$X^2 = \widetilde{\mathbf{X}}_n' \mathbf{P}_{k-1} \widetilde{\mathbf{X}}_n \qquad \text{für } \mathbf{P}_{k-1} := \operatorname{diag}\left(\frac{1}{p_{01}}, \ldots, \frac{1}{p_{0,k-1}} \right) - \frac{\mathbf{1}_{k-1}\mathbf{1}_{k-1}'}{p_{0k}}$$

als auch $\mathbf{P}_{k-1} = \widetilde{\mathbf{\Sigma}}^{-1}$ und somit in Verteilung $X^2 \longrightarrow \widetilde{\mathbf{X}}' \widetilde{\mathbf{\Sigma}}^{-1} \widetilde{\mathbf{X}} \sim \chi^2_{k-1}$ unter H_0 für $n \to \infty$.

Nun am **Beispiel** des n-fachen Würfelwurfs die Implementation des obigen Tests durch die **R**-Funktion `chisq.test()`: Ein Würfel sei 200-mal geworfen worden, wobei sich die folgenden Häufigkeitsergebnisse eingestellt haben mögen:

Augenzahl	1	2	3	4	5	6	Σ
absolute Häufigkeit	32	35	41	38	28	26	200

Es soll entschieden werden, ob es sich um einen „fairen" Würfel handelt, d. h., ob die Wahrscheinlichkeit für eine jede Augenzahl dieselbe, also 1/6 ist.

Offenbar ist die Hypothese $H_0 : p_j = 1/6$ für alle $j = 1, \ldots, 6$ gegen $H_1 : p_j \ne 1/6$ für ein $1 \le j \le 6$ im Multinomialmodell $\mathcal{M}(n; p_1, \ldots, p_6)$ mit $n = 200$ zu testen. Dazu speichern wir die Daten zunächst in einem Vektor, um diesen dann an `chisq.test()` zu übergeben:

```
> absHKn <- c( 32, 35, 41, 38, 28, 26)
```

Der approximative χ^2-Test im Multinomialmodell	

`> chisq.test( absHKn)`
 	
`  Chi-squared test for given probabilities`
 	
`data:  absHKn`
`X-squared = 5.02, df = 5, p-value = 0.4134`
	Einen Vektor als alleiniges Argument interpretiert `chisq.test()` als Vektor absoluter Trefferhäufigkeiten in einem Multinomialmodell und führt den approximativen χ^2-Test auf Gleichheit der Trefferwahrscheinlichkeiten durch (stets ohne eine Stetigkeitskorrektur).
	Resultate: Teststatistik (`X-squared`, asymptot. χ^2_{k-1}-verteilt), ihre Freiheitsgrade (`df`), p-Wert (`p-value`).
	Mit dem Argument `p` kann ein Vektor an Trefferwahrscheinlichkeiten spezifiziert werden (nicht gezeigt).

9.6 Kontingenztafeln

Hat man es mit mindestens zwei endlich-diskreten und höchstens ordinal skalierten Zufallsvariablen (Faktoren in der **R**/**S**-Terminologie) zu tun, so ist man oft an deren gemeinsamer Verteilung interessiert oder an gewissen Eigenschaften dieser Verteilung. Dies führt auf die Analyse der Auftrittshäufigkeiten aller möglichen Kombinationen der Levels dieser Faktoren in einer Stichprobe von n Untersuchungseinheiten. Zur explorativen Darstellung bedient man sich der Häufigkeitstabellen, auch Kontingenztafeln genannt. Inferenzstatistisch ist man z. B. an der Unabhängigkeit der beteiligten Faktoren interessiert. Mit diesen Aspekten bei zwei Faktoren befassen wir uns in den folgenden Abschnitten.

9.6.1 χ^2-Test auf Unabhängigkeit zweier Faktoren und auf Homogenität

Die zwei folgenden Probleme, zur Abkürzung mit „U“ und „H“ benannt, sind eng verwandt und führen auf dieselbe Teststatistik:

U: Für zwei (nominal oder ordinal skalierte) Faktoren A und B mit $r \geq 2$ bzw. $s \geq 2$ möglichen Ausprägungen (genannt Levels oder Faktorstufen), die an derselben Untersuchungseinheit beobachtet werden, stellt sich häufig die Frage, ob die beiden Variablen in der zugrundeliegenden Population **u**nabhängig sind.

Ein Beispiel hierfür könnte die Frage sein, ob in der Population aller Menschen die Faktoren „Geschlecht“ (mit den $r = 2$ Levels männlich, weiblich) und „Blutgruppe“ (mit den $s = 4$ Levels A, B, AB, 0) unabhängig sind.

H: Für einen (nominal oder ordinal skalierten) Faktor B mit $s \geq 2$ möglichen Ausprägungen ist für $r \geq 2$ Populationen (einer zugrundeliegenden „Super“-Population) zu klären, ob die r Verteilungen der Variablen B gleich sind, d. h., ob die Populationen hinsichtlich der Verteilung von B **h**omogen sind.

Beispiel: Sind die Verteilungen des Faktors „Blutgruppe“ (mit seinen $s = 4$ Levels A, B, AB, 0) in $r \geq 2$ verschiedenen „Ethnien“ (ethnisch verschiedene Populationen der Super-Population Weltbevölkerung) gleich?

Als Daten hat man im Fall U die Häufigkeiten des gemeinsamen Auftretens zweier Faktorstufen a_i und b_j für $1 \leq i \leq r$ und $1 \leq j \leq s$ in einer Stichprobe von n Untersuchungseinheiten.
In H ist es genauso, nur dass der Faktor A die Populationszugehörigkeit markiert, also a_i die Zugehörigkeit zur Population a_i bedeutet. In beiden Fällen erfolgt die explorative Darstellung in

einer zweidimensionalen $(r \times s)$-Kontingenztafel (auch $(r \times s)$-Tafel oder Kreuztabelle genannt) wie sie z. B. durch `table()` oder `xtabs()` erzeugt wird.

Wir präsentieren hier die schematische Darstellung einer solchen $(r \times s)$-Kontingenztafel, auf deren Notation wir uns im Folgenden mehrfach beziehen werden: X_{ij} ist die in der Stichprobe beobachtete absolute Häufigkeit der Kombination (a_i, b_j) von Level a_i für Faktor A und Level b_j für Faktor B. Die X_{ij} sind im Folgenden also stets natürliche Zahlen einschließlich der 0 (Null).

	Faktor B					
Faktor A	b_1	$\dots$	b_j	$\dots$	b_s	Zeilensumme
a_1	X_{11}	$\dots$	X_{1j}	$\dots$	X_{1s}	$n_{1\cdot}$
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
a_i	X_{i1}	$\dots$	X_{ij}	$\dots$	X_{is}	$n_{i\cdot}$
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
a_r	X_{r1}	$\dots$	X_{rj}	$\dots$	X_{rs}	$n_{r\cdot}$
Spaltensumme	$n_{\cdot 1}$	$\dots$	$n_{\cdot j}$	$\dots$	$n_{\cdot s}$	$n \equiv n_{\cdot\cdot}$

Dabei werden die folgenden, üblichen Abkürzungen verwendet:

$$n_{i\cdot} \equiv \sum_{j=1}^{s} X_{ij}, \qquad n_{\cdot j} \equiv \sum_{i=1}^{r} X_{ij} \quad \text{und} \quad n_{\cdot\cdot} \equiv \sum_{i=1}^{r}\sum_{j=1}^{s} X_{ij} = \sum_{i=1}^{r} n_{i\cdot} = \sum_{j=1}^{s} n_{\cdot j}$$

9.6.1.1 Zum Fall der Unabhängigkeit

Offenbar hat der durch „Übereinanderstapeln“ der Spalten erhaltene $(r \cdot s)$-dimensionale Vektor

$$X_n := (X_{11}, \dots, X_{r1}, X_{12}, \dots, X_{r2}, \dots\dots, X_{1s}, \dots, X_{rs})'$$

eine Multinomialverteilung $\mathcal{M}(n; p_{11}, \dots, p_{r1}, p_{12}, \dots, p_{r2}, \dots\dots, p_{1s}, \dots, p_{rs})$ mit unbekannten „Zell“-Wahrscheinlichkeiten $0 < p_{ij} < 1$, für die natürlich $p_{\cdot\cdot} \equiv \sum_{i=1}^{r}\sum_{j=1}^{s} p_{ij} = 1$ ist. Die Darstellung der Parametrisierung dieser Multinomialverteilung in einer zur obigen Kontingenztafel analogen Tabelle hilft, die hiernach folgende Formulierung der Unabhängigkeitshypothese zu veranschaulichen:

	Faktor B					
Faktor A	b_1	$\dots$	b_j	$\dots$	b_s	Zeilensumme
a_1	p_{11}	$\dots$	p_{1j}	$\dots$	p_{1s}	$p_{1\cdot}$
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
a_i	p_{i1}	$\dots$	p_{ij}	$\dots$	p_{is}	$p_{i\cdot}$
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
a_r	p_{r1}	$\dots$	p_{rj}	$\dots$	p_{rs}	$p_{r\cdot}$
Spaltensumme	$p_{\cdot 1}$	$\dots$	$p_{\cdot j}$	$\dots$	$p_{\cdot s}$	$p_{\cdot\cdot} = 1$

Bekanntermaßen sind zwei Faktoren (diskrete Zufallsvariablen) A und B genau dann unabhängig, wenn mit $1 \le i \le r$ und $1 \le j \le s$ für alle Paare (i, j) gilt:

$$p_{ij} \equiv \mathbb{P}((A, B) = (a_i, b_j)) = \mathbb{P}(A = a_i)\mathbb{P}(B = b_j) \equiv p_{i\cdot} p_{\cdot j}$$

D. h., die Wahrscheinlichkeit p_{ij} einer jeden einzelnen Zelle (a_i, b_j) ist das Produkt der beiden dazugehörigen Randwahrscheinlichkeiten $p_{i\cdot}$ und $p_{\cdot j}$. Es ist daher naheliegend, dass ein Test

auf Unabhängigkeit der beiden Variablen auf dem Vergleich der beobachteten Häufigkeiten X_{ij} mit den unter der Unabhängigkeitsannahme erwarteten Häufigkeiten $np_{i\cdot}p_{\cdot j}$ hinausläuft, wobei letztere jedoch durch $n(n_{i\cdot}/n)(n_{\cdot j}/n) = n\bar{n}_{i\cdot}\bar{n}_{\cdot j}$ zu schätzen sind. (An dieser Stelle sei bemerkt, dass die Randsummen $n_{i\cdot}$ und $n_{\cdot j}$ selbst zufällig sind.)

9.6.1.2 Zum Fall der Homogenität

Der Faktor A bezeichne die Populationszugehörigkeit, wobei die Populationen möglicherweise unterschiedliche große Anteile an der Super-Population haben, nämlich $q_1, \ldots, q_r$ mit natürlich $\sum_{i=1}^{r} q_i = 1$. Wir sind an der Verteilung von A zwar nicht interessiert, weil nur die r Verteilungen des Faktors B auf Gleichheit untersucht werden sollen, aber es muss berücksichtigt werden, dass die Stichproben aus den r Populationen unterschiedlich große Anteile an der Gesamtstichprobe haben (können).

Die folgende Tabelle stellt die *gemeinsame* Verteilung von A und B in der Super-Population dar (und lässt natürlich das Szenario beliebiger Verteilungen in den r Populationen zu). Dies soll helfen, die Formulierung der Homogenitätshypothese zu begründen:

Population	Faktor B						Anteil an Super-Population
	b_1	$\ldots$	b_j	$\ldots$	b_s	Zeilensumme	
a_1	p_{11}	$\ldots$	p_{1j}	$\ldots$	p_{1s}	$p_{1\cdot}$	q_1
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$
a_i	p_{i1}	$\ldots$	p_{ij}	$\ldots$	p_{is}	$p_{i\cdot}$	q_i
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$
a_r	p_{r1}	$\ldots$	p_{rj}	$\ldots$	p_{rs}	$p_{r\cdot}$	q_r
						$p_{\cdot\cdot} = 1$	1

In der Super-Population liegt Homogenität der r Verteilungen des Faktors B vor, falls die nach der Population A bedingte Verteilung von B nicht von A abhängt, mit anderen Worten, wenn

$$\mathbb{P}(B = b_j | A = a_1) = \ldots = \mathbb{P}(B = b_j | A = a_r) =: p_j \quad \text{für jedes } j = 1, \ldots, s$$

Es folgt für die (unbedingte) Verteilung von B in der Super-Population, dass $\mathbb{P}(B = b_j) = p_j$ für jedes j ist. Deswegen ist für alle i und j

$$p_j = \mathbb{P}(B = b_j | A = a_i) = \frac{\mathbb{P}(B = b_j, A = a_i)}{\mathbb{P}(A = a_i)} \equiv \frac{p_{ij}}{q_i}$$

erfüllt, was äquivalent zu $q_i p_j = p_{ij}$ und damit äquivalent zur Unabhängigkeit von A und B in der Super-Population ist.

Offenbar läuft ein Test auf Homogenität auf den Vergleich von p_{ij} und $q_i p_j$ hinaus, wozu die beobachteten Häufigkeiten X_{ij} mit den Schätzwerten der unter der Homogenitätsannahme erwarteten Häufigkeiten $nq_i p_j$ verglichen werden. Dabei wird p_j unter Homogenität durch $n_{\cdot j}/n$ geschätzt (d. h., die Stichproben der r Populationen werden gepoolt und als eine Stichprobe aus der Super-Population aufgefasst) und q_i durch $n_{i\cdot}/n$ (d. h., durch den relativen Anteil des i-ten Stichprobenumfangs am Gesamtstichprobenumfang). Dies führt schließlich zu derselben Teststatistik wie in Szenario U (obgleich hier die Zeilensummen $n_{1\cdot}, \ldots, n_{r\cdot}$, sprich die Stichprobenumfänge, nicht notwendig zufällig sind, sondern auch fest vorgegeben sein können).

9.6.1.3 Der approximative χ^2-Test auf Unabhängigkeit und der approximative χ^2-Test auf Homogenität: `chisq.test() & Co.`

Annahmen: (A_k, B_k), $k = 1, \ldots, n$, seien unabhängig und identisch diskret verteilt mit Werten in $\{a_1, \ldots, a_r\} \times \{b_1, \ldots, b_s\}$ und unbekannten $p_{ij} \equiv \mathbb{P}((A_k, B_k) = (a_i, b_j)) \in (0, 1)$. Auch die Marginal-Wahrscheinlichkeiten $p_{i\cdot} \equiv \mathbb{P}(A_k = a_i)$ und $p_{\cdot j} \equiv \mathbb{P}(B_k = b_j)$ seien unbekannt (aber natürlich mit $\sum_{i=1}^r p_{i\cdot} = 1 = \sum_{j=1}^s p_{\cdot j}$).

U: Zu testen ist die Unabhängigkeitshypothese

$$\begin{aligned} H_{U0} &: p_{ij} = p_{i\cdot}p_{\cdot j} \text{ für alle } 1 \le i \le r \text{ und } 1 \le j \le s \qquad \text{gegen} \\ H_{U1} &: p_{ij} \ne p_{i\cdot}p_{\cdot j} \text{ für ein Paar } (i, j). \end{aligned}$$

H: Zu testen ist die Homogenitätshypothese

$$\begin{aligned} H_{H0} &: \mathbb{P}(B = b_j | A = a_1) = \ldots = \mathbb{P}(B = b_j | A = a_r) \text{ für alle } 1 \le j \le s \qquad \text{gegen} \\ H_{H1} &: \mathbb{P}(B = b_j | A = a_i) \ne \mathbb{P}(B = b_j | A = a_l) \text{ für mindestens ein } j \text{ und } i \ne l. \end{aligned}$$

Teststatistik (für beide Szenarien):

$$X^2 := \sum_{i=1}^r \sum_{j=1}^s \frac{(X_{ij} - n\bar{n}_{i\cdot}\bar{n}_{\cdot j})^2}{n\bar{n}_{i\cdot}\bar{n}_{\cdot j}} \longrightarrow \chi^2_{(r-1)(s-1)} \qquad \text{in Verteilung unter } H_{U0} \text{ bzw. } H_{H0}, \text{ falls } n \to \infty,$$

wobei $X_{ij} = \sum_{k=1}^n 1\{(A_k, B_k) = (a_i, b_j)\}$ und $\bar{n}_{i\cdot}$ sowie $\bar{n}_{\cdot j}$ wie üblich definiert sind.

Entscheidungsregel für einen konkreten Datenvektor $(x_{11}, \ldots, x_{rs})'$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0 \quad \Longleftrightarrow \quad p\text{-Wert } \le \alpha,$$

wobei p-Wert $= 1 - F_{\chi^2_{(r-1)(s-1)}}(x^2)$ und $F_{\chi^2_k}$ die VF der χ^2_k-Verteilung ist.

Bemerkung: Die Verteilungsaussage für die obige, ebenfalls Pearsons X^2-Statistik genannte Teststatistik X^2 folgt aus einem allgemeineren Resultat über χ^2-Tests für zusammengesetzte Nullhypothesen im Multinomialmodell, dessen Beweis für unsere Zwecke erheblich zu aufwändig ist. Wir sagen hier nur, dass für X^2 – unter Ausnutzung, dass $\bar{n}_{i\cdot}$ und $\bar{n}_{\cdot j}$ für alle i, j stark konsistente (Maximum-Likelihood-)Schätzer sind – eine approximative Darstellung als quadratische Form eines $(r \cdot s)$-dimensionalen, asymptotisch normalverteilten (Multinomial-) Vektors mit einer geeigneten $(rs \times rs)$-Matrix existiert, dass diese Matrix den Rang $(r-1)(s-1)$ hat und dass die Theorie über die Verteilung quadratischer Formen mit singulären Matrizen die obige Verteilungsaussage liefert. (Siehe z. B. [84, Witting & Müller-Funk (1995)], Kap. 6.1 & 6.2.)

Die Durchführung dieser Tests geschieht in **R** entweder direkt ebenfalls mit der Funktion `chisq.test()` oder indirekt mit „Hilfsfunktionen“ zur Tabellierung von Häufigkeiten. Ein **Beispiel** zum Vergleich zweier Therapieformen (aus [68, Sachs & Hedderich (2006)], S. 508 bzw. Martini (1953)) verdeutliche die Vorgehensweise: In einer Epidemie seien insgesamt 80 Personen behandelt worden. Eine Gruppe von 40 Kranken erhielt eine Standarddosis eines spezifischen Mittels. Die andere Gruppe von 40 Kranken sei nur symptomatisch behandelt worden (Behandlung der Krankheitserscheinungen, nicht aber ihrer Ursachen). Das Resultat der Behandlungen wird in Besetzungszahlen der drei Klassen „schnell geheilt“, „langsam geheilt“, „gestorben“ ausgedrückt:

Therapeutischer Erfolg	Therapie		Insgesamt
	symptomatisch	spezifisch	
Geheilt in x Wochen	14	22	36
Geheilt in $x + y$ Wochen	18	16	34
Gestorben	8	2	10
Insgesamt	40	40	80

Es soll die Hypothese H_0: „*Der therapeutische Erfolg ist von der Therapieform unabhängig*“ gegen die Alternative H_1: „*Er ist von der Therapieform nicht unabhängig*“ gestestet werden.

Für die geeignete Aufbereitung der Daten gibt es in **R** mehrere Möglichkeiten:

1. Liegen die Daten, wie im Beispiel, bereits in Form einer Kontingenztafel vor, so ist es das Einfachste, sie in Form einer Matrix zu speichern (und diese lediglich der besseren Lesbarkeit halber auch mit Zeilen- und Spalten- sowie Dimensionsnamen zu versehen):

```
> (Epidemie.mat <- matrix( c( 14, 18, 8, 22, 16, 2), ncol = 2,
+         dimnames = list( Erfolg = c( "Geheilt in x Wochen",
+                                      "Geheilt in x+y Wochen", "Gestorben"),
+                          Therapie = c( "symptomatisch", "spezifisch"))))
                        Therapie
Erfolg                   symptomatisch spezifisch
  Geheilt in x Wochen               14         22
  Geheilt in x+y Wochen             18         16
  Gestorben                          8          2
```

2. Die absoluten Häufigkeiten können auch in einem Data Frames wie folgt erfasst worden sein:

```
> Epi.df
  Therapie Erfolg  N
1     symp      x 14
2     spez      x 22
3     symp    x+y 18
4     spez    x+y 16
5     symp     ex  8
6     spez     ex  2
```

3. Hat man hingegen die Rohdaten als zwei Faktorvektoren vorliegen – separat oder als Komponenten eines Data Frames – deren Elemente für jeden Patienten jeweils angeben, welche Therapie er erhalten hat bzw. welches der therapeutische Erfolg war, so kann mit der Funktion `table()` im Fall separater Vektoren bzw. `xtabs()` im Fall eines Data Frames die Tabelle der absoluten Häufigkeiten angelegt werden (genauer ein `table`- bzw. `xtabs`-Objekt):

```
> therapie
 [1] symp spez spez symp symp
....
[76] spez spez spez symp symp
Levels: symp spez
> erfolg
 [1] x+y x   x   x   ex
....
[76] x+y x   x+y x+y x
Levels: x x+y ex

> table( erfolg, therapie)
      therapie
erfolg symp spez
   x     14   22
   x+y   18   16
   ex     8    2
```

```
> Epidemie.df
   Therapie Erfolg
1      symp    x+y
2      spez      x
3      spez      x
....
78     spez    x+y
79     symp    x+y
80     symp      x

> xtabs( ~ Erfolg + Therapie,
+         data = Epidemie.df)
      Therapie
Erfolg symp spez
   x     14   22
   x+y   18   16
   ex     8    2
```

Hinweis: Der Data Frame in Punkt 2 kann aus einem `table`-Objekt (und somit also auch aus den Rohdaten) mit Hilfe der `table`-Methode von `as.data.frame()` erzeugt werden:

```
> Epi.df <- as.data.frame.table( table( Therapie = therapie, Erfolg = erfolg),
                                 responseName = "N")
```

Doch nun zur Durchführung der Tests auf Unabhängigkeit bzw. Homogenität in **R**:

Der approximative χ^2-Test auf Unabhängigkeit bzw. Homogenität	

`> chisq.test( Epidemie.mat)`
 	
`  Pearson's Chi-squared test`
 	
`data:  Epidemie.mat`
`X-squared = 5.4954, df = 2,`
`p-value = 0.06407`
	χ^2-Unabhängigkeitstest (nur im (2×2)-Fall mit Yates' Stetigkeitskorrektur in der Teststatistik).
	
	Resultate: Teststatistik (`X-squared`, asymptotisch $\chi^2_{(r-1)(s-1)}$-verteilt), ihre $(r-1)(s-1)$ Freiheitsgrade (`df`), p-Wert (`p-value`).

`> chisq.test( erfolg, therapie)`
`....`
`> chisq.test( table( erfolg, therapie))`
`....`
`> chisq.test( xtabs( ~ Erfolg + Therapie,`
`+  data = Epidemie.df))`
`....`
`> chisq.test( xtabs( N ~ Erfolg + Therapie,`
`+  data = Epi.df))`
`....`
	Diese weiteren vier Aufrufe von `chisq.test()` liefern analoge Ausgaben und – bis auf die Information hinter `data:` – exakt dieselben Resultate wie der obige.

`> summary( table( erfolg, therapie))`
`Number of cases in table: 80`
`Number of factors: 2`
`Test for independence of all factors:`
`  Chisq = 5.495, df = 2, p-value = 0.06407`
	`summary()` von `table`- oder `xtabs`-Objekten liefert etwas umfangreichere Ausgaben, aber ebenfalls dieselben Resultate wie die vorherigen, expliziten Anwendungen von `chisq.test()`.

`> summary( xtabs( ~ Erfolg + Therapie,`
`+  data = Epidemie.df))`
`Call: xtabs(formula = ~Erfolg + Therapie,`
`  data = Epidemie.df)`
`Number of cases in table: 80`
`Number of factors: 2`
`Test for independence of all factors:`
`  Chisq = 5.495, df = 2, p-value = 0.06407`
	(Zur leistungsfähigen Funktionalität von `xtabs()` siehe auch §9.6.3.)

`> summary(  xtabs( N ~ Erfolg + Therapie,`
`+  data = Epi.df))`
`....`
	Dieselbe Ausgabe wie eben.

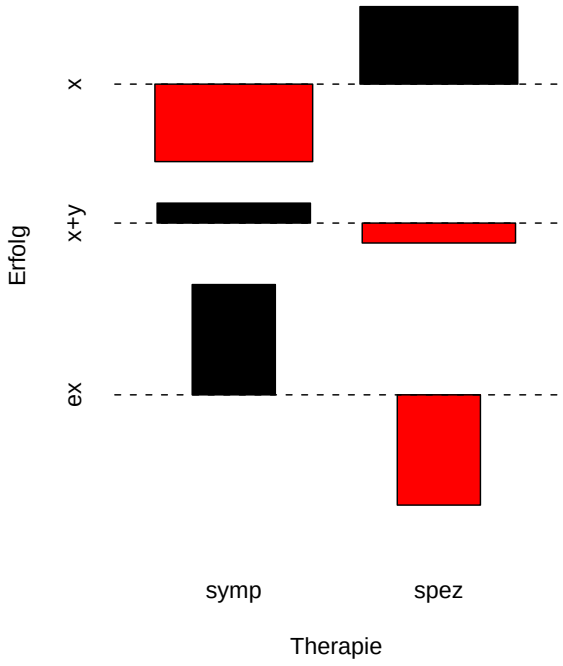
Eine grafische Darstellung der in den Zellen einer zweidimensionalen Kontingenztafel auftretenden Abweichungen von der hypothetisierten Unabhängigkeit der beiden Faktoren ist der auf der nächsten Seite rechts oben zu sehende, sogenannte Cohen-Friendly Assoziationsplot, der z. B. durch

```
> assocplot( xtabs( ~ Therapie + Erfolg, data = Epidemie.df))
```

generiert wird. (Beachte die andere Reihenfolge der Faktoren in der `xtabs`-Formel im Vergleich zur obigen Tabellierung; der erste Faktor in der Formel kommt in die Spalten der Grafik.)

Zur Interpretation: Bei einer zweidimensionalen Kontingenztafel ist der *signierte* Beitrag zu Pearsons X^2 aus Zelle (i,j) der Quotient $B_{ij} := (x_{ij} - \hat{e}_{ij})/\sqrt{\hat{e}_{ij}}$ mit beobachteter Häufigkeit x_{ij} und geschätzter erwarteter Häufigkeit $\hat{e}_{ij}$ in dieser Zelle. Im Cohen-Friendly-Plot wird jede

solche Zelle durch ein Rechteck repräsentiert, dessen Breite proportional zu $\sqrt{\hat{e}_{ij}}$ ist und dessen Höhe proportional zu B_{ij}, sodass seine Fläche proportional zur absoluten Abweichung $|x_{ij} - \hat{e}_{ij}|$ ist. Die Rechtecke sind relativ zu Bezugslinien platziert, die $B_{ij} = 0$, also Unabhängigkeit repräsentieren. Für $x_{ij} > \hat{e}_{ij}$ erstreckt sich das dann schwarze Rechteck oberhalb der Bezugslinie, für $x_{ij} < \hat{e}_{ij}$ ist es rot und unterhalb. So werden Zellen, sprich Faktorlevelkombinationen augenfällig, die unter der Unabhängigkeitshypothese zu viele oder zu wenige „Treffer“ haben, wobei eine gegebene absolute Abweichung (= Fläche) einen umso größeren Beitrag (= (quadrierte) Höhe) zur Ablehnung von H_0 liefert, je kleiner die erwartete Häufigkeit (= Breite) ist. D. h., dieselbe absolute Abweichung in einer erwartet schwach besetzten Zelle ist „gefährlicher“ für H_0 als die in einer erwartet stark besetzten Zelle. Visuell: Von zwei Rechtecken derselben Fläche ist das schmalere das „gefährlichere“ für H_0.



Es existieren weitere grafischen Darstellungsmethoden für diskrete Daten und insbesondere Kontingenztafeln. In **R** steht dafür u. a. das Package `vcd` zur Verfügung, das in [62, Meyer et al. (2006)] ausführlicher beschrieben wird und durch das Buch “Visualizing Categorical Data” von Michael Friendly [36] inspiriert wurde. Überhaupt scheint in diesem Zusammenhang die Website `http://www.datavis.ca` von Michael Friendly erwähnens- und besuchenswert, da dort das genannte Buch auch als Online-Version zur Verfügung steht. (Die o. g. Arbeit [62] steht im Package `vcd` auch als Vignette zur Verfügung; siehe `vignette("strucplot", package = "vcd")`. In diesem Zusammenhang evtl. auch interessant: `vignette("residual-shadings", package = "vcd")`.)

9.6.2 Fishers Exakter Test auf Unabhängigkeit zweier Faktoren

Ist in einer Zelle einer $(r \times s)$-Kontingenztafel die erwartete Häufigkeit unter der Unabhängigkeitshypothese kleiner als 5, so gibt die Funktion `chisq.test()` eine Warnung aus, dass die χ^2-Approximation der Teststatistik schlecht und das Testergebnis zweifelhaft ist. In einem solchen Fall wird als eine Alternative Fishers Exakter Test empfohlen.

Der Exakte Test nach Fisher geht von festen (!) Marginalhäufigkeiten aus. Zu diesen Marginalhäufigkeiten werden *alle* möglichen Belegungen einer Kontingenztafel sowie deren Auftrittswahrscheinlichkeiten unter der Unabhängigkeitshypothese ermittelt. Es wird also die durch Unabhängigkeit induzierte diskrete Verteilung auf der Menge *dieser* Kontingenztafeln exakt bestimmt. Im Falle einer (2×2)-Tafel wird dabei ausgenutzt, dass ihre gesamte Zellenbelegung bei gegebenen Marginalhäufigkeiten schon durch eine einzige Zelle, z. B. ihren Eintrag links oben, festgelegt ist. Die Verteilung der Werte dieser Zelle (und damit der ganzen Tafel) ist dann unter der Unabhängigkeitshypothese eine hypergeometrische Verteilung.

Als Memo ein (Urnen-)Modell der hypergeometrischen Verteilung: Aus einer $n_{..}$-elementigen Grundgesamtheit mit genau $n_{1.}$ Elementen der Eigenschaft A und $n_{2.}$ Elementen der Eigenschaft

$\overline{A}$ wird *ohne Zurücklegen* genau $n_{\cdot 1}$-mal zufällig gezogen. X sei die (zufällige) Anzahl gezogener A-Elemente. Für genau $n_{\cdot 1}$ der insgesamt $n_{\cdot\cdot}$ Elemente ist dann das Ereignis B gezogen zu werden eingetreten und für $n_{\cdot 2}$ eben nicht ($\hat{=}\overline{B}$). Die (2×2)-Tafel der vier möglichen Kombinationen aus Eigenschaft und Ziehungsstatus der $n_{\cdot\cdot}$ Elemente lautet dann:

		Ziehungsstatus		
		B	$\overline{B}$	
Eigen-	A	X	$n_{1\cdot} - X$	$n_{1\cdot}$
schaft	$\overline{A}$	$n_{\cdot 1} - X$	$n_{2\cdot} - n_{\cdot 1} + X$	$n_{2\cdot}$
		$n_{\cdot 1}$	$n_{\cdot 2}$	$n_{\cdot\cdot}$

Sind A und B unabhängig, so ist dann

$$\mathbb{P}\left(X = x | \text{Alle} = n_{\cdot\cdot}, \text{zu ziehende Elemente} = n_{\cdot 1}, A\text{-Elemente} = n_{1\cdot}\right) \underset{H_0}{=} \frac{\binom{n_{1\cdot}}{x}\binom{n_{\cdot\cdot} - n_{1\cdot}}{n_{\cdot 1} - x}}{\binom{n_{\cdot\cdot}}{n_{\cdot 1}}}$$

Assoziieren wir in obigem Epidemie-Beispiel das Erhalten einer spezifischen Therapie mit dem Gezogen-werden B (und das Erhalten einer symptomatischen Therapie also mit $\overline{B}$) sowie den – dichotomisierten – Therapieerfolg „geheilt“ (egal wann) mit A (und „nicht geheilt“ demnach mit $\overline{A}$), so wird klar, dass, wenn die Ereignisse A und B unabhängig sind, der Heilungserfolg oder -misserfolg unabhängig von der Therapieentscheidung ist, und somit quasi eben auch schon vor der Therapieentscheidung (= der Ziehung) festgestanden hat.

Im Fall einer $(r \times s)$-Tafel mit $r > 2$ oder $s > 2$ und mit gegebenen Marginalien hat ihre Zellenbelegung eine *multivariate* hypergeometrische Verteilung. Damit ist für jede mögliche Belegung dieser Kontingenztafel die Auftrittswahrscheinlichkeit berechenbar und alle Kontingenztafeln mit kleineren Auftrittswahrscheinlichkeiten als die der beobachteten Tafel sprechen „noch extremer“ gegen die Unabhängigkeitshypothese als die der vorliegenden Stichprobe. Die Summe dieser „extremen“ Wahrscheinlichkeiten samt derjenigen der beobachteten Tafel ist der p-Wert des Tests.

Aufgrund der Annahme fester Marginalhäufigkeiten handelt es sich um einen *bedingten* Test auf (damit ebenso nur bedingte) Unabhängigkeit. Deshalb hat der erhaltene p-Wert nur für solche Kontingenztafeln Gültigkeit, die dieselben Marginalhäufigkeiten aufweisen; er ist also auch eine bedingte Größe. Die statistische Schlussfolgerung darf für die betrachteten Variablen demnach *nicht* (so ohne Weiteres) auf Kontingenztafeln mit anderen Marginalhäufigkeiten übertragen werden. (Für die Theorie des exakten Testens in $(r \times s)$-Tafeln verweisen wir auf die Literatur, wie z. B. [2, Agresti (1996)], ch. 2.6.5, und etwas detaillierter in [1, Agresti (1990)], ch. 3.5.3., oder [32, Fleiss et al. (2003)]), ch. 3.2, aber nur für (2×2)-Tafeln.)

9.6.2.1 Die Implementation durch `fisher.test()`

In **R** ist Fishers Exakter Test auf Unabhängigkeit zweier Faktoren einer $(r \times s)$-Kontingenztafel durch die Funktion `fisher.test()` implementiert. Bei zu großen Tafeln bzw. bei Tafeln mit zu großen Einträgen kann die **R**-Implementation von `fisher.test()` (bzw. die intern verwendete C-Routine) den – exponentiell anwachsenden – kombinatorischen Aufwand nicht bewältigen und bricht mit einer entsprechenden Fehlermeldung ab. (Für mehr Details siehe die Online-Hilfe.) Für den Beispieldatensatz von Seite 184 unten funktioniert jedoch alles völlig problemlos. Als Formate für die Dateneingabe lassen sich dieselben wie für `chisq.test()` verwenden:

Fishers Exakter Test auf Unabhängigkeit	

`> fisher.test( Epidemie.mat)`
 	
`  Fisher's Exact Test for Count Data`
 	
`data:  Epidemie.mat`
`p-value = 0.06743`
`alternative hypothesis: two.sided`
	Fishers Exakter Test auf Unabhängigkeit.
	
	Resultate: p-Wert (`p-value`), Alternative: Für Tafeln größer als 2×2 immer zweiseitig (zur Erläuterung siehe unten).

`> fisher.test( erfolg, therapie)`
`....`
`> fisher.test( table( erfolg, therapie))`
`....`
`> fisher.test( xtabs( ~ Erfolg + Therapie,`
`+  data = Epidemie.df))`
`....`
`> fisher.test( xtabs( N ~ Erfolg + Therapie,`
`+  data = Epi.df))`
`....`
	Diese weiteren vier Aufrufe von `fisher.test()` liefern analoge Ausgaben und – bis auf die Information hinter `data:` – exakt dieselben Resultate wie der vorherige.

9.6.2.2 Der Spezialfall der (2×2)-Tafel: Die Odds Ratio

In der Situation zweier Faktoren A und B mit je zwei Levels a_1, a_2 bzw. b_1, b_2, also der sogenannten (2×2)-Tafel, besteht eine enge Beziehung zu zwei (vor allem im bio-medizinischen Kontext) häufig verwendeten Größen im Zusammenhang mit dem bedingten Ein- oder Nicht-Eintreten von Ereignissen. Identifizieren wir *zur Abkürzung* A mit $\{A = a_1\}$ und B mit $\{B = b_1\}$, so sind A und B nun zwei Ereignisse, deren Gegenereignisse mit $\overline{A}$ und $\overline{B}$ bezeichnet werden und deren Ein- oder Nicht-Eintreten nur davon abhängt, welchen Wert der jeweilig zugrundeliegende Faktor annimmt.

Für zwei Ereignisse A und B versteht man unter den <u>Odds für B unter A</u>, kurz $\Omega_{B|A}$, das Verhältnis der Wahrscheinlichkeit für B unter der Bedingung A zu der Wahrscheinlichkeit für $\overline{B}$ unter der Bedingung A. Die <u>Odds Ratio</u> $\omega_{B|A}$ bezeichnet den Quotienten der beiden Odds $\Omega_{B|A}$ und $\Omega_{B|\overline{A}}$, kurz:

$$\Omega_{B|A} := \frac{\mathbb{P}(B|A)}{\mathbb{P}(\overline{B}|A)} \qquad \text{und} \qquad \omega_{B|A} := \frac{\Omega_{B|A}}{\Omega_{B|\overline{A}}}$$

$\Omega_{B|A}$ gibt also an, „wie die Chancen für B (gegenüber $\overline{B}$) stehen“, falls A eingetreten ist. Eine bekannte Sprechweise in diesem Zusammenhang dürfte z. B. sein: „Falls A gilt, stehen die Chancen 2 zu 1 für B (gegenüber $\overline{B}$)“, was eben bedeutet, dass unter A die Wahrscheinlichkeit für B doppelt so hoch ist wie die für $\overline{B}$.

$\omega_{B|A}$ ist damit offenbar ein Chancen*verhältnis*, das angibt, um welchen Faktor die Chancen für B (gegenüber $\overline{B}$) im Fall A besser (oder schlechter) stehen als im Fall $\overline{A}$.

Die Odds Ratio steht in enger Beziehung zu der „theoretischen“ Tabelle von Seite 182 (unten) mit $r = 2 = s$, denn für

	B	$\overline{B}$
A	$\mathbb{P}(A \cap B)$	$\mathbb{P}(A \cap \overline{B})$
$\overline{A}$	$\mathbb{P}(\overline{A} \cap B)$	$\mathbb{P}(\overline{A} \cap \overline{B})$

ist $$\omega_{B|A} \equiv \frac{\mathbb{P}(B|A)/\mathbb{P}(\overline{B}|A)}{\mathbb{P}(B|\overline{A})/\mathbb{P}(\overline{B}|\overline{A})} = \frac{\mathbb{P}(A \cap B)\mathbb{P}(\overline{A} \cap \overline{B})}{\mathbb{P}(\overline{A} \cap B)\mathbb{P}(A \cap \overline{B})}.$$

Damit liegen auch Schätzer für die Odds und die Odds Ratio nahe: Basierend auf der beobachteten gemeinsamen Häufigkeitstabelle von A und B (vgl. Seite 182 oben)

	B	$\overline{B}$	
A	X_{11}	X_{12}	$n_{1\cdot}$
$\overline{A}$	X_{21}	X_{22}	$n_{2\cdot}$
	$n_{\cdot 1}$	$n_{\cdot 2}$	

sind $\widehat{\Omega}_{B|A} := \dfrac{X_{11}/n_{1\cdot}}{X_{12}/n_{1\cdot}}$, $\widehat{\Omega}_{B|\overline{A}} := \dfrac{X_{21}/n_{2\cdot}}{X_{22}/n_{2\cdot}}$ und $\widehat{\omega}_{B|A} := \dfrac{\widehat{\Omega}_{B|A}}{\widehat{\Omega}_{B|\overline{A}}}$

sogenannte "plug-in"-Schätzer für $\Omega_{B|A}$, $\Omega_{B|\overline{A}}$ bzw. $\omega_{B|A}$. Offenbar gilt $\widehat{\omega}_{B|A} = \dfrac{X_{11}X_{22}}{X_{12}X_{21}}$.

Doch was hat obiges mit der (bedingten) Unabhängigkeit zweier Faktoren im Fall einer (2×2)-Kontigenztafel zu tun? Antwort: Die Nullhypothese der (bedingten) Unabhängigkeit ist äquivalent zur Hypothese, dass die Odds Ratio der beiden Faktoren gleich 1 ist, denn wegen

$$\omega_{B|A} = \frac{\mathbb{P}(B|A)/(1-\mathbb{P}(B|A))}{\mathbb{P}(B|\overline{A})/(1-\mathbb{P}(B|\overline{A}))} = \frac{1/\mathbb{P}(B|\overline{A})-1}{1/\mathbb{P}(B|A)-1}$$

ist offenbar

$$\omega_{B|A} = 1 \iff \mathbb{P}(B|\overline{A}) = \mathbb{P}(B|A) \qquad (\iff \mathbb{P}(B|A) = \mathbb{P}(B)).$$

Bemerkung: Wegen der Beziehung $\omega_{B|A} = \omega_{\overline{B}|\overline{A}} = \dfrac{1}{\omega_{B|\overline{A}}} = \dfrac{1}{\omega_{\overline{B}|A}}$ ist die Anordnung der (2×2)-Tabelle im Hinblick auf einen Test auf Unabhängigkeit demnach irrelevant.

Für allgemeines $\omega \neq 1$ hat das erste Element der (2×2)-Tafel mit festen Marginalhäufigkeiten eine nicht-zentrale hypergeometrische Verteilung, deren Nicht-Zentralitätsparameter gerade ω ist. Die Bestimmung eines exakten p-Wertes für den Test der Nullhypothese $H_0 : \omega = \omega_0$ gegen eine ein- oder zweiseitige Alternative kann somit auf Basis dieser Verteilung geschehen. In `fisher.test()` steht für ω_0 das Argument `or` (wie odds ratio, mit der Voreinstellung 1) zur Verfügung. Es wird aber nur bei (2×2)-Tafeln berücksichtigt. (Für interessante Details zur Erzeugung und Auswertung von (2×2)-Tafeln siehe z. B. [32, Fleiss et al. (2003)], ch. 3, und zur Inferenz über die Odds Ratio ebenda in ch. 6 sowie speziell zu ihrer Interpretation dort in ch. 6.3.)

Als **Beispiel** für die Anwendung von `fisher.test()` auf eine (2×2)-Tafel benutzen wir Daten aus [2, Agresti (1996)], Table 2.4, p. 26, die sich bei einer (retrospektiven) Studie der Beziehung zwischen Herzinfarkt und Rauchgewohnheit ergaben. Es waren 262 Herzinfarktpatienten und 519 Kontrollpatienten (ohne Herzinfarkt) ausgewählt worden und nach ihren Rauchgewohnheiten klassifiziert worden. Dabei ergab sich die folgende Kontingenztafel:

Rauchgewohnheiten	Herzinfarkt Ja	Herzinfarkt Nein	
Jemals geraucht	172	173	345
Niemals geraucht	90	346	436
	262	519	

Die geschätzten Odds für „Herzinfarkt Ja“ (=: HI) in „Jemals geraucht“ und für HI in „Niemals geraucht“ ergeben sich zu

$$\widehat{\Omega}_{\text{HI}|\text{Je geraucht}} = \frac{172/345}{173/345} \doteq 0.9942 \qquad \text{und} \qquad \widehat{\Omega}_{\text{HI}|\text{Nie geraucht}} = \frac{90/436}{346/436} \doteq 0.2601$$

D. h., in der Rauchergruppe stehen die Chancen etwa 1:1 einen Herzinfarkt zu bekommen, sodass also jeder zweite einen Herzinfarkt erleidet, wohingegen unter den Nichtrauchern die Chancen etwa 1:4 stehen, also nur einer von fünf. Als Odds Ratio ergibt sich aus den obigen Zahlen

$$\widehat{\omega}_{\text{Herzinfarkt}|\text{Je geraucht}} \doteq 3.82,$$

was bedeutet, dass in der Rauchergruppe die Chancen einen Herzinfarkt zu erleiden, etwa 3.8-mal höher stehen als in der Nichtrauchergruppe (egal wie hoch die Chancen in der Raucher- bzw. Nichtrauchergruppe absolut sind).

Zur Auswertung mit **R**: Wir speichern die Daten zunächst in Form einer Matrix (wobei wir der besseren Identifizierbarkeit halber nicht nur `dimnames`, sondern auch Komponentennamen für die `dimnames`-Liste der Matrix verwenden, was sich bei der Ausgabe von `HI` als nützlich erweist):

```
> (HI <- matrix( c( 172, 90, 173, 346), nrow = 2, dimnames = list(
+                Geraucht = c( "Ja", "Nein"), Herzinfarkt = c( "Ja", "Nein"))))
        Herzinfarkt
Geraucht  Ja Nein
    Ja   172  173
    Nein  90  346
```

Die Anwendung von Fishers Exaktem Test ergibt eine hochsignifikante Abweichung von der Nullhypothese, dass die Odds Ratio gleich eins ist, und spricht somit stark gegen die (bedingte) Unabhängigkeit. Darüber hinaus wird für ω ein approximatives zweiseitiges 95 %-Konfidenzintervall und ein Schätzwert angegeben:

```
> fisher.test( HI)

        Fisher's Exact Test for Count Data

data:  HI
p-value < 2.2e-16
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 2.760758 5.298034
sample estimates:
odds ratio
  3.815027
```

Die geschätzte Wert 3.815027 der Odds Ratio ist hier *nicht* der Wert des plug-in-Schätzers $\widehat{\omega}_{\text{Herzinfarkt}|\text{Je geraucht}}$ von Seite 190, sondern der eines sogenannten "conditional" Maximum-Likelihood-Schätzers $\widehat{\omega}^{(c)}$. Dieser maximiert die bedingte Verteilungsfunktion des ersten Elements der (2×2)-Tabelle (für feste Marginalhäufigkeiten), die ja die einer nicht-zentralen hypergeometrischen Verteilung ist, im Nichtzentralitätsparameter (vgl. [32, Fleiss et al. (2003)], ch. 6.4).

Mit `alternative = "greater"` wird ein einseitiger Test durchgeführt und für ω ein approximatives einseitiges 95 %-Konfidenzintervall präsentiert:

```
> fisher.test( HI, alternative = "greater")

        Fisher's Exact Test for Count Data

data:  HI
p-value < 2.2e-16
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:
 2.900966      Inf
sample estimates:
odds ratio
  3.815027
```

9.6.3 Kontingenztafeln für $k \geq 2$ Faktoren und ihre Darstellung: `xtabs()` & `ftable()`

Kontingenztafeln (oder Kreuztabellen) sind ein Werkzeug, um die k-dimensionale gemeinsame Häufigkeitsverteilung der Realisierungen von k (≥ 2) Faktoren darzustellen. Angenommen, Faktor j (für $j = 1, \ldots, k$) habe L_j (≥ 2) verschiedene Levels, die – der Einfachheit halber – schlicht $1, \ldots, L_j$ lauten mögen. Die Rohdaten einer Erhebung der Ausprägungen dieser Faktoren an einer Stichprobe von n Untersuchungseinheiten können grundsätzlich in zwei Formen vorliegen:

1. Für jede Faktorlevelkombination $(l_1, \ldots, l_k)$ mit $1 \leq l_j \leq L_j$ für $j = 1, \ldots, k$ der insgesamt $L := \prod_{j=1}^{k} L_j$ möglichen Kombinationen ist ihre absolute Häufigkeit $n_{l_1,\ldots,l_k} \geq 0$ gegeben, wobei $n = \sum_{i_1=1}^{L_1} \ldots \sum_{i_k=1}^{L_k} n_{i_1,\ldots,i_k}$ ist. D. h., es liegt eine „Liste“ von L absoluten Häufigkeiten vor.

2. Für jede Untersuchungseinheit $1 \leq i \leq n$ ist die an ihr beobachtete Faktorlevelkombination $(l_{i1}, \ldots, l_{ik})$ mit $1 \leq l_{ij} \leq L_j$ für $j = 1, \ldots, k$ notiert, sodass also eine „Liste“ von n explizit genannten Levelkombinationen gegeben ist.

Beide Datenformate kann die Funktion `xtabs()` verarbeiten, wie wir anhand von Beispielen mit zwei derartigen Datensätzen demonstrieren werden.

9.6.3.1 Der Fall bereits registrierter absoluter Häufigkeiten

Beispiel: Im Package `MASS` befindet sich der Data Frame `Insurance`, in dem Daten von Autoversicherungsfällen zusammengefasst sind: Seine Faktorkomponente `District` mit den vier Levels `1, 2, 3` und `4` enthält den Wohndistrikt der Versicherungsnehmer; der (geordnete) Faktor `Group` mit den Levels `<1l`, `1-1.5l`, `1.5-2l` und `>2l` (so aufsteigend) beschreibt die in vier Klassen eingeteilte Hubraumgröße der Fahrzeuge; der (geordnete) Faktor `Age` ist das in vier Klassen eingeteilte Alter der Fahrer mit den Levels `<25`, `25-29`, `30-35` und `>35`; die `integer`-Variable `Holders` nennt die Anzahl der durch `District`, `Group` und `Age` gegebenen Levelkombinationen und die `integer`-Variable `Claims` ist schließlich die Zahl der Versicherungsfälle in dieser Levelkombination. Hier ein Ausschnitt des Data Frames:

```
> data( Insurance, package = "MASS")
> Insurance
   District  Group   Age Holders Claims
1         1    <1l   <25     197     38
2         1    <1l 25-29     264     35
3         1    <1l 30-35     246     20
4         1    <1l   >35    1680    156
5         1 1-1.5l   <25     284     63
....
60        4 1.5-2l   >35     344     63
61        4    >2l   <25       3      0
62        4    >2l 25-29      16      6
63        4    >2l 30-35      25      8
64        4    >2l   >35     114     33
```

Beachte: Der Data Frame führt in seinen $4 \cdot 4 \cdot 4 = 64$ Zeilen alle *möglichen* Levelkombinationen auf, inclusive derer, die die `Claims`-Häufigkeit 0 hatten!

Wir ignorieren in diesem Beispiel die Variable `Holders` und beschränken uns auf die Beziehung zwischen der Schadenshäufigkeit `Claims` und den drei Faktoren `District`, `Group` und `Age`.

Zunächst wollen wir die Anzahl der Schäden nur nach Distrikt und Hubraumgruppe „kreuztabellieren“, was bedeutet, dass wir die absoluten Häufigkeiten aller $4 \cdot 4 = 16$ Levelkombinationen

von `District` und `Group` bestimmen wollen. Es soll also nicht nach den Levels von `Age` „aufgelöst“, sondern vielmehr über sie hinweg zusammengefasst (= summiert) werden.

Das geschieht, indem wir der Funktion `xtabs()` mit Hilfe einer Formel der Art „`HK ~ F1 + .... + Fk`“ als erstem Argument mitteilen, welche Variablen wie zu verwenden sind. Ihr zweites Argument `data` erhält den Data Frame, aus welchem diese Variablen kommen sollen. Die Formel ist wie folgt zu interpretieren: Der Variablenname links der Tilde `~` benennt den Vektor der bereits registrierten Auftrittshäufigkeiten. Rechts der Tilde befinden sich die durch `+` verknüpften Namen derjenigen Faktoren, die zur Kreuztabellierung zu verwenden sind.

Hier lautet die Formel `Claims ~ District + Group` und bedeutet demnach, dass die Schadenszahlen `Claims` nach `District` und `Group` zu klassifizieren sind. Das Ergebnis ist ein `xtabs`-Objekt und enthält als Resultat (lediglich) die zweidimensionale Häufigkeitsverteilung in Form einer Kontingenztafel mit so vielen Zeilen wie Levels des in der Formel auf der rechten Seite zuerst genannten Faktors (hier vier) und so vielen Spalten wie Levels des zweiten Faktors (hier ebenfalls vier):

```
> (HK1 <- xtabs( Claims ~ District + Group, data = Insurance))
       Group
District <1l 1-1.5l 1.5-2l >2l
       1 249 636    378    118
       2 150 415    242     84
       3  93 258    152     50
       4  47 141     91     47
```

Um den approximativen χ^2-Test von Seite 184 auf Unabhängigkeit der zwei Faktoren durchzuführen, gibt es eine spezielle Methode der Funktion `summary()` für `xtabs`-Objekte: Ihre Anwendung auf `HK1` zeigt als Ergebnis den Aufruf (`Call:`), der zur Erzeugung des `xtabs`-Objektes geführt hat, die Anzahl aller beobachteten Schäden, also den Stichprobenumfang n (`Number of cases in table: 3151`), wie viele Faktoren hinter der Kreuztabelle stecken (`Number of factors: 2`) und schließlich das Ergebnis des Unabhängigkeitstests:

```
> summary( HK1)
Call: xtabs(formula = Claims ~ District + Group, data = Insurance)
Number of cases in table: 3151
Number of factors: 2
Test for independence of all factors:
        Chisq = 12.599, df = 9, p-value = 0.1816
```

Nehmen wir als dritten Faktor neben `District` und `Group` die Altersklasse `Age` des Fahrers hinzu, wird von `xtabs()` die dreidimensionale Häufigkeitsverteilung der $4 \cdot 4 \cdot 4 = 64$ Faktorlevelkombinationen ermittelt. Für deren Darstellung wird die dritte Dimension (hier der Faktor `Age`, da drittes Element der rechten Seite der `xtabs`-Formel) „schichtenweise“ aufgelöst und für jeden der (hier vier) Levels des dritten Faktors eine zu den ersten beiden Faktoren passende Kontingenztafel präsentiert. Dies führt natürlich zu einer (gelegentlich unübersichtlich) umfangreichen Ausgabe:

```
> (HK2 <- xtabs( Claims ~ District + Group + Age, data = Insurance))
, , Age = <25

        Group
District <1l 1-1.5l 1.5-2l >2l
       1  38  63     19      4
```

```
       2  22  25     14      4
       3   5  10      8      3
       4   2   7      5      0

, , Age = 25-29

        Group
District <1l 1-1.5l 1.5-2l >2l
       1  35  84     52     18
       2  19  51     46     15
       3  11  24     19      2
       4   5  10      7      6

, , Age = 30-35

        Group
District <1l 1-1.5l 1.5-2l >2l
       1  20  89     74     19
       2  22  49     39     12
       3  10  37     24      8
       4   4  22     16      8

, , Age = >35

        Group
District <1l 1-1.5l 1.5-2l >2l
       1 156 400    233     77
       2  87 290    143     53
       3  67 187    101     37
       4  36 102     63     33
```

Auch hierfür liefert `summary()` das Ergebnis des approximativen χ^2-Tests auf Unabhängigkeit der drei Faktoren. (Eine Verallgemeinerung des Tests von Seite 184.) Dabei sollte die Warnung hinsichtlich der Verlässlichkeit der χ^2-Approximation beachtet werden: Die ohnehin nur approximativ geltende χ^2-Verteiltheit der verwendeten Teststatistik und somit auch der p-Wert sind aufgrund niedriger Besetzungszahlen einiger Zellen zweifelhaft:

```
> summary( HK2)
Call: xtabs(formula = Claims ~ District + Group + Age, data = Insurance)
Number of cases in table: 3151
Number of factors: 3
Test for independence of all factors:
        Chisq = 125.17, df = 54, p-value = 1.405e-07
        Chi-squared approximation may be incorrect
```

Eine hilfreiche Funktion für die Darstellung umfangreicher Kontingenztafeln ist `ftable()`, da sie „flache“ (Engl.: “flat”) und daher übersichtlichere Kontingenztafeln produziert:

```
> ftable( HK2)
                Age <25 25-29 30-35 >35
District Group
1        <1l         38    35    20 156
         1-1.5l      63    84    89 400
```

```
         1.5-2l      19    52    74 233
         >2l          4    18    19  77
2        <1l         22    19    22  87
         1-1.5l      25    51    49 290
         1.5-2l      14    46    39 143
         >2l          4    15    12  53
3        <1l          5    11    10  67
         1-1.5l      10    24    37 187
         1.5-2l       8    19    24 101
         >2l          3     2     8  37
4        <1l          2     5     4  36
         1-1.5l       7    10    22 102
         1.5-2l       5     7    16  63
         >2l          0     6     8  33
```

Bemerkung: `ftable()` erlaubt die explizite Angabe, welche Variablen in den Zeilen und welche in den Spalten gruppiert werden sollen; eine sehr nützliche Option (siehe ihre Online-Hilfe).

9.6.3.2 Der Fall explizit aufgeführter Levelkombinationen

Beispiel: Der im Package `rpart` zur Verfügung stehende Data Frame `solder` enthält Daten eines Experiments zum sogenannten "wave-soldering"- (= Wellenlöt-)Verfahren, das der Befestigung von Elektronikbauteilen auf Leiterplatinen dient. In ihm wurden fünf, für die Lötqualität relevante Einflussfaktoren variiert und als Ergebnis die Anzahl der sogenannten "solder skips" (= Lötlücken) registriert. Der Faktor `Opening` gibt den Freiraum um die Montageplatte durch die Levels `L, M` und `S` an. Der Faktor `Solder` quantifiziert durch `Thick` und `Thin` die Menge an verwendetem Lötzinn. Der Faktor `Mask` gibt an, welche von vier Lötmasken `A1.5`, `A3`, `B3` oder `B6` verwendet wurde (unterschieden sich in Material und Dicke). Der Faktor `PadType` ist der Montageplattentyp (variierte in Geometrie und Größe); seine zehn Levels sind `D4`, `D6`, `D7`, `L4`, `L6`, `L7`, `L8`, `L9`, `W4` und `W9`. Die `integer`-Variable `Panel` ist das Feld 1, 2 oder 3, in dem auf der Platte die "skips" gezählt wurden. Die `integer`-Variable `skips` schließlich enthält die Anzahl der sichtbaren Lötlücken auf der Leiterplatine. Hier ein Ausschnitt aus dem Data Frame:

```
> data( solder, package = "rpart")
> solder
    Opening Solder Mask PadType Panel skips
1         L  Thick A1.5      W4     1     0
2         L  Thick A1.5      W4     2     0
3         L  Thick A1.5      W4     3     0
4         L  Thick A1.5      D4     1     0
5         L  Thick A1.5      D4     2     0
6         L  Thick A1.5      D4     3     0
....
31        M  Thick A1.5      W4     1     0
32        M  Thick A1.5      W4     2     0
33        M  Thick A1.5      W4     3     0
....
91        L   Thin A1.5      W4     1     1
92        L   Thin A1.5      W4     2     1
93        L   Thin A1.5      W4     3     2
....
181       L  Thick   A3      W4     1     2
```

```
182       L  Thick   A3      W4     2     0
183       L  Thick   A3      W4     3     1
....
720       S   Thin   B6      L9     3    15
```

Uns interessiere hier jetzt nicht die tatsächliche Anzahl an "skips", sondern nur, ob welche vorliegen oder nicht. Daher leiten wir aus `skips` eine binäre (= „dichotome") Variable ab:

```
> solder$ok <- factor( solder$skips == 0)
```

Nun sollen `Opening`, `Mask` und `ok` kreuztabelliert werden, um letztendlich zu prüfen, ob diese drei Faktoren unabhängig sind oder nicht. Auch hier wird dies der Funktion `xtabs()` über eine Formel mitgeteilt. Allerdings besitzt diese Formel keine linke Seite, sondern nur ein rechte und lautet „`~ F1 + .... + Fk`". In Konsequenz ermittelt `xtabs()` „selbstständig" die absoluten Häufigkeiten aller beobachteten Fälle mit denselben Levelkombinationen der Variablen in dem angebenen Data Frame. Bei der Darstellung bedienen wir uns auch wieder der Hilfe von `ftable()`:

```
> ftable( SkipHK <- xtabs( ~ Opening + Mask + ok, data = solder))
             ok FALSE TRUE
Opening Mask
L       A1.5       21   39
        A3         20   40
        B3         36   24
        B6         37   23
M       A1.5       22   38
        A3         35   25
        B3         39   21
        B6         51    9
S       A1.5       49   11
        A3         48   12
        B3         60    0
        B6         60    0
```

`summary()` liefert wieder den approximativen χ^2-Test auf Unabhängigkeit der drei Faktoren sowie weitere Informationen:

```
> summary( SkipHK)
Call: xtabs(formula = ~Opening + Mask + ok, data = solder)
Number of cases in table: 720
Number of factors: 3
Test for independence of all factors:
        Chisq = 164.45, df = 17, p-value = 3.526e-26
```

10 Einführung in die lineare Regression

Das **klassische Modell der (multiplen) linearen (normalen) Regression** lautet in Vektor-Matrix-Notation bekanntermaßen

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

Dabei ist $\mathbf{X}$ die feste, bekannte $(n \times p)$-Designmatrix mit reellwertigen Elementen und vollem Spaltenrang, also $\text{Rang}(X) = p$, $\boldsymbol{\beta}$ der feste, unbekannte und zu schätzende p-dimensionale Regressionsparametervektor, $\boldsymbol{\varepsilon}$ ein zufälliger, unbekannter, n-dimensional zentriert multivariat normalverteilter Fehlervektor mit unkorrelierten (und damit unabhängigen) Komponenten sowie $\mathbf{Y}$ der (zwangsläufig auch zufällige) n-dimensionale Beobachtungs- (oder Response-)Vektor. Etwas detaillierter und formaler:

$$\mathbf{Y} = \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}, \ \mathbf{X} = \begin{pmatrix} x_{10} & x_{11} & \dots & x_{1,p-1} \\ x_{20} & x_{21} & \dots & x_{2,p-1} \\ \vdots & \vdots & \dots & \vdots \\ x_{n0} & x_{n1} & \dots & x_{n,p-1} \end{pmatrix}, \ \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{pmatrix} \ \text{und} \ \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix} \sim \mathcal{N}_n(\mathbf{0}, \sigma^2 \mathbf{I}_{n\times n})$$

Beachte:

- Es wird von *linearer* Regression gesprochen, weil der Parameter $\boldsymbol{\beta}$ (!) linear in das Modell eingeht. „Klassisch“ bezieht sich auf $\text{Cov}(\boldsymbol{\varepsilon}) = \sigma^2 \mathbf{I}_{n\times n}$, also darauf, dass die ε_i unkorreliert und homoskedastisch sind.
- Für $i = 1, \dots, n$ ist also $Y_i = \mathbf{x}_i'\boldsymbol{\beta} + \varepsilon_i \equiv \sum_{j=0}^{p-1} x_{ij}\beta_j + \varepsilon_i$, wobei der p-dimensionale Designvektor $\mathbf{x}_i' := (x_{i0}, x_{i1}, \dots, x_{i,p-1})$ die i-te Zeile der Designmatrix ist. Dieser geht durch eine geeignete, bekannte und für alle i gleiche Funktion aus einem Covariablenvektor $\mathbf{z}_i$ hervor.

  Beispiele: $\mathbf{z}_i' = (z_{i1}, z_{i2}) \in \mathbb{R}^2$ und $\mathbf{x}_i' = (1, z_{i1}, z_{i2})$ oder $\mathbf{x}_i' = (1, z_{i1}, z_{i1}^2, \log(|z_{i2}|))$.
- Die Spalten der Designmatrix werden auch (Modell- oder Covariablen-)Terme genannt. In der Regel enthält ihre erste Spalte nur Einsen: $x_{i0} \equiv 1$ für $i = 1, \dots, n$.
- Somit gilt: $Y_i \sim \mathcal{N}(\mathbf{x}_i'\boldsymbol{\beta}, \sigma^2)$ unabhängig für $i = 1, \dots, n$. Sind die Covariablen und damit die Designvektoren stochastisch, so werden die $(Y_i, \mathbf{x}_i)$ als unabhängig und identisch verteilt angenommen und die Y_i als bedingt $\mathbf{x}_i$ unabhängig normalverteilt mit $\mathbb{E}[Y_i|\mathbf{x}_i] = \mathbf{x}_i'\boldsymbol{\beta}$ und $\text{Var}(Y_i|\mathbf{x}_i) = \sigma^2$.

10.1 Einige Resultate aus der Theorie der normalen linearen Modelle

Zur Erinnerung und zur Referenz einige – zum Teil auch nochmal kurz begründete – Resultate aus der Theorie der normalen linearen Modelle (für deren ausführliche Beweise z. B. [46, Hocking (1996), §§3.1.1 und 7.3.1] oder [70, Searle (1971), Abschnitte 3.2 – 3.5] konsultiert werden könnten):

Der Kleinste-Quadrate-Schätzer $\hat{\boldsymbol{\beta}}$ für $\boldsymbol{\beta}$	minimiert $\|\|\mathbf{Y}-\mathbf{X}\mathbf{b}\|\| \equiv \sqrt{\sum_{i=1}^n (Y_i - \mathbf{x}_i'\mathbf{b})^2}$ in $\mathbf{b} \in \mathbb{R}^p$,
berechnet sich zu	$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$
und erfüllt (im Normalmodell)	$\hat{\boldsymbol{\beta}} \sim \mathcal{N}_p\left(\boldsymbol{\beta}, \sigma^2(\mathbf{X}'\mathbf{X})^{-1}\right)$.
Die gefitteten Werte sind	$\hat{\mathbf{Y}} := \mathbf{X}\hat{\boldsymbol{\beta}} = \underbrace{\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'}_{=:\,\mathbf{H}}\mathbf{Y}$.
Die $(n \times n)$-Projektionsmatrix	$\mathbf{H}$ ist symmetrisch, idempotent und erfüllt $\mathbf{H}(\mathbf{I}_{n\times n} - \mathbf{H}) = \mathbf{0}_{n\times n}$. Sie heißt auch “hat matrix”, da sie aus dem Beobachtungsvektor $\mathbf{Y}$ durch Aufsetzen eines „Hutes“ den Vektor $\hat{\mathbf{Y}}$ macht.

Für den Residuenvektor $\hat{\boldsymbol{\varepsilon}} := \mathbf{Y} - \hat{\mathbf{Y}} = (\mathbf{I}_{n\times n} - \mathbf{H})\mathbf{Y}$

gilt $\hat{\boldsymbol{\varepsilon}} \sim \mathcal{N}_n(\mathbf{0}, \sigma^2(\mathbf{I}_{n\times n} - \mathbf{H}))$ und $\hat{\boldsymbol{\varepsilon}} \perp \hat{\mathbf{Y}}$, denn $(\mathbf{Y} - \hat{\mathbf{Y}})'\hat{\mathbf{Y}} = \mathbf{Y}'(\mathbf{I}_{n\times n} - \mathbf{H})\mathbf{H}\mathbf{Y} = 0$.

Die Residuenquadratesumme $\mathrm{RSS} := \hat{\boldsymbol{\varepsilon}}'\hat{\boldsymbol{\varepsilon}} = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$

erfüllt $\mathbb{E}\left[\mathrm{RSS}/(n-p)\right] = \sigma^2$.

Des Weiteren gilt für $\hat{\sigma}^2 := \dfrac{\mathrm{RSS}}{n-p}$: $(n-p)\,\dfrac{\hat{\sigma}^2}{\sigma^2} \sim \chi^2_{n-p}$

und $\hat{\boldsymbol{\beta}}$ und $\hat{\sigma}^2$ sind unverzerrte, Minimum-Varianz-Schätzer für $\boldsymbol{\beta}$ bzw. für σ^2 sowie stochastisch unabhängig.

Außerdem ist (wg. $\mathbf{H}(\mathbf{I}_{n\times n} - \mathbf{H}) = \mathbf{0}_{n\times n}$) $\mathrm{Cov}(\hat{\mathbf{Y}}, \hat{\boldsymbol{\varepsilon}}) = \mathbf{0}_{n\times n}$,
also sind $\hat{\mathbf{Y}}$ und $\hat{\boldsymbol{\varepsilon}}$ stochastisch unabhängig.

Falls $\mathbf{X}$ eine Einsenspalte $\mathbf{1}_n$ hat, ist $\mathbf{H}\mathbf{1}_n = \mathbf{1}_n$ (denn $\mathbf{H}\mathbf{X} = \mathbf{X}$)

und es folgt die „orthogonale“ Quadratesummenzerlegung

$$\begin{aligned}
\mathrm{SS}_{Total} &:= \textstyle\sum_{i=1}^n (Y_i - \bar{Y}_.)^2 \\
&= \textstyle\sum_{i=1}^n (Y_i - \hat{Y}_i)^2 \;+\; \sum_{i=1}^n (\hat{Y}_i - \bar{Y}_.)^2 \\
&=: \mathrm{RSS} \;+\; \mathrm{SS}_{Regression}
\end{aligned}$$

(die sich ergibt aus

z. B. durch Teleskopieren mit $\hat{\mathbf{Y}}$ und Ausmultiplizieren zu

$$\begin{aligned}
\mathrm{SS}_{Total} &= (\mathbf{Y} - \mathbf{1}_n\bar{Y}_.)'(\mathbf{Y} - \mathbf{1}_n\bar{Y}_.) \\
&= (\mathbf{Y} - \hat{\mathbf{Y}})'(\mathbf{Y} - \hat{\mathbf{Y}}) + \\
&\quad (\hat{\mathbf{Y}} - \mathbf{1}_n\bar{Y}_.)'(\hat{\mathbf{Y}} - \mathbf{1}_n\bar{Y}_.) + \\
&\quad 2(\mathbf{Y} - \hat{\mathbf{Y}})'(\hat{\mathbf{Y}} - \mathbf{1}_n\bar{Y}_.) \\
&= \mathrm{RSS} \;+\; \mathrm{SS}_{Regression} \;+ \\
&\quad 2\big(\underbrace{(\mathbf{Y} - \hat{\mathbf{Y}})'\hat{\mathbf{Y}}}_{=\,0\ (\text{s. o.})} - \underbrace{(\mathbf{Y} - \hat{\mathbf{Y}})'\mathbf{1}_n}_{=\,0\ (\text{s. o.})}\bar{Y}_.\big)
\end{aligned}$$

oder nur durch Ausmultiplizieren und „Zusammensetzen“ aus

$$\begin{aligned}
\mathrm{SS}_{Total} &= (\mathbf{Y} - \mathbf{1}_n\bar{Y}_.)'(\mathbf{Y} - \mathbf{1}_n\bar{Y}_.) \\
&= \mathbf{Y}'\mathbf{Y} - \tfrac{1}{n}\mathbf{Y}'\mathbf{1}_n\mathbf{1}_n'\mathbf{Y}, \\
\mathrm{RSS} &= (\mathbf{Y} - \hat{\mathbf{Y}})'(\mathbf{Y} - \hat{\mathbf{Y}}) \\
&= \mathbf{Y}'\mathbf{Y} - \mathbf{Y}'\mathbf{H}\mathbf{Y} \quad \text{und} \\
\mathrm{SS}_{Regr.} &= (\hat{\mathbf{Y}} - \mathbf{1}_n\bar{Y}_.)'(\hat{\mathbf{Y}} - \mathbf{1}_n\bar{Y}_.) \\
&= \mathbf{Y}'\mathbf{H}\mathbf{Y} - \tfrac{1}{n}\mathbf{Y}'\mathbf{1}_n\mathbf{1}_n'\mathbf{Y}.\ )
\end{aligned}$$

Damit ist der multiple R^2-Wert

$$\begin{aligned}
R^2 &:= \frac{\mathrm{SS}_{Regression}}{\mathrm{SS}_{Total}} = \frac{\mathrm{SS}_{Total} - \mathrm{RSS}}{\mathrm{SS}_{Total}} \\
&= 1 - \frac{\mathrm{RSS}}{\mathrm{SS}_{Total}} \in [0,1]
\end{aligned}$$

ein Maß für die Güte der Anpassung (auch Bestimmtheitsmaß bzw. “coefficient of determination” oder “percentage of variation explained” genannt), da er den relativen Anteil an der Gesamtstreuung der Y_i misst, der durch das Regressionsmodell „erklärt“ wird. Statt R^2 wird (insbesondere bei kleinem n) wegen seiner Unverzerrtheit

der korrigierte (“adjusted”) R^2-Wert $\quad R_a^2 := 1 - \dfrac{\mathrm{RSS}/(n-p)}{\mathrm{SS}_{Total}/(n-1)}$

verwendet, denn $\mathrm{RSS}/(n-p)$ ist ein erwartungstreuer Schätzer für σ^2 und $\mathrm{SS}_{Total}/(n-1)$ (in „Abwesenheit“ jeglicher Covariablen) ein ebensolcher für die Varianz der Y_i.

Beachte: Hat $\mathbf{X}$ keine Einsenspalte, d. h. ist $\beta_0 = 0$ per Modell, so ist $\mathrm{SS}_{Total} := \sum_{i=1}^n Y_i^2$.

Der multiple Korrelationskoeffizient R ist der empirische Pearsonsche Korrelationskoeffizient der $(Y_i, \hat{Y}_i)$, und $\hat{\boldsymbol{\beta}}$ maximiert letzteren für $(Y_i, \tilde{Y}_i)$ in der Klasse $\{\tilde{\mathbf{Y}} = \mathbf{X}\mathbf{a} |\ \mathbf{a} \in \mathbb{R}^p\}$.

<u>t-Tests marginaler Hypothesen:</u> Wegen $\hat{\boldsymbol{\beta}} \sim \mathcal{N}_p\left(\boldsymbol{\beta}, \sigma^2(\mathbf{X}'\mathbf{X})^{-1}\right)$ gilt für $j = 1, \ldots, p$, dass $\hat{\beta}_{j-1} \sim \mathcal{N}_1\left(\beta_{j-1}, \sigma^2\left((\mathbf{X}'\mathbf{X})^{-1}\right)_{jj}\right)$ und daher

$$\frac{\hat{\beta}_{j-1} - c_0}{\hat{\sigma}\sqrt{\left((\mathbf{X}'\mathbf{X})^{-1}\right)_{jj}}} \sim t_{n-1} \qquad \text{unter } H_0 : \beta_{j-1} = c_0. \tag{36}$$

<u>F-Tests allgemeiner linearer Hypothesen:</u> Für eine $(r \times p)$-Matrix $\mathbf{C}$ mit $\text{Rang}(\mathbf{C}) = r$ und einen Vektor $\mathbf{c}_0 \in \mathbb{R}^r$ sei die

$$\text{Nullhypothese } H_0 : \mathbf{C}\boldsymbol{\beta} = \mathbf{c}_0 \qquad \text{gegen die} \qquad \text{Alternative } H_1 : \mathbf{C}\boldsymbol{\beta} \neq \mathbf{c}_0$$

zu testen. Wegen $\hat{\boldsymbol{\beta}} \sim \mathcal{N}_p\left(\boldsymbol{\beta}, \sigma^2(\mathbf{X}'\mathbf{X})^{-1}\right)$ gilt unter H_0, dass $\mathbf{C}\hat{\boldsymbol{\beta}} \sim \mathcal{N}_r\left(\mathbf{c}_0, \sigma^2\mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}'\right)$ und damit

$$\frac{1}{\sigma^2}(\mathbf{C}\hat{\boldsymbol{\beta}} - \mathbf{c}_0)'\left[\mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}'\right]^{-1}(\mathbf{C}\hat{\boldsymbol{\beta}} - \mathbf{c}_0) \sim \chi^2_r \tag{37}$$

gemäß der Verteilungstheorie quadratischer Formen normalverteilter Vektoren (siehe hierfür z. B. [58, Mathai & Provost (1992)] oder ein Buch zur Theorie linearer Modelle wie z. B. [46, Hocking (1996), part I]).

Aus $\text{RSS}/\sigma^2 \equiv (n-p)\,\hat{\sigma}^2/\sigma^2 \sim \chi^2_{n-p}$ sowie der Unabhängigkeit von $\hat{\boldsymbol{\beta}}$ und $\hat{\sigma}^2$ folgt

$$T(\mathbf{Y}) := \frac{(\mathbf{C}\hat{\boldsymbol{\beta}} - \mathbf{c}_0)'\left[\mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}'\right]^{-1}(\mathbf{C}\hat{\boldsymbol{\beta}} - \mathbf{c}_0)/r}{\text{RSS}/(n-p)} \sim F_{r,n-p} \quad \text{unter } H_0, \tag{38}$$

denn für zwei unabhängige Zufallsvariablen $Z \sim \chi^2_r$ und $N \sim \chi^2_s$ gilt $(Z/r)/(N/s) \sim F_{r,s}$. (Randmemo: $\mathbb{E}[F_{r,s}] = s/(s-2), s \geq 3 \Rightarrow \mathbb{E}[T(\mathbf{Y})] = (n-p)/(n-p-2), n-p \geq 3$, unter H_0.)

Zur Interpretation der F-Teststatistik $T(\mathbf{Y})$: Bezeichne RSS_{H_0} die Residuenquadratesumme des unter der linearen Nebenbedingung $H_0 : \mathbf{C}\boldsymbol{\beta} = \mathbf{c}_0$ gefitteten Modells. Dann ist

$$\begin{aligned} \text{RSS} &= \text{quadratischer Abstand von } \mathbf{Y} \text{ zu } \{\mathbf{X}\boldsymbol{\beta} : \boldsymbol{\beta} \in \mathbb{R}^p\} \text{ und} \\ \text{RSS}_{H_0} &= \text{quadratischer Abstand von } \mathbf{Y} \text{ zu } \{\mathbf{X}\boldsymbol{\beta} : \boldsymbol{\beta} \in \mathbb{R}^p \text{ mit } \mathbf{C}\boldsymbol{\beta} = \mathbf{c}_0\}, \end{aligned}$$

sodass sich $T(\mathbf{Y})$ wegen

$$T(\mathbf{Y}) \overset{(!)}{=} \frac{(\text{RSS}_{H_0} - \text{RSS})/r}{\text{RSS}/(n-p)} \tag{39}$$

- geometrisch deuten lässt als Unterschied zwischen den Abständen von $\mathbf{Y}$ zum Unterraum $\{\mathbf{X}\boldsymbol{\beta} : \boldsymbol{\beta} \in \mathbb{R}^p\}$ und $\mathbf{Y}$ zum Unterraum $\{\mathbf{X}\boldsymbol{\beta} : \boldsymbol{\beta} \in \mathbb{R}^p \text{ mit } \mathbf{C}\boldsymbol{\beta} = \mathbf{c}_0\}$ relativ zum Abstand von $\mathbf{Y}$ zum Unterraum $\{\mathbf{X}\boldsymbol{\beta} : \boldsymbol{\beta} \in \mathbb{R}^p\}$.
- oder als relativer <u>Zuwachs</u> an Reststreuung und somit Verschlechterung des Regressionsfits, wenn zum restriktiveren Modell mit $\mathbf{C}\boldsymbol{\beta} = \mathbf{c}_0$ übergegangen wird.

Beispiele: Der sogenannte „globale F-Test“ (Englisch auch “overall F-test”) der Hypothese $H_0 : \beta_1 = \ldots = \beta_{p-1} = 0$, also der Test auf *keine* Regressionsbeziehung zwischen den Beobachtungen und den Covariablen ist der Test der Hypothese $H_0 : \mathbf{C}\boldsymbol{\beta} = \mathbf{c}_0$ mit

$$\mathbf{C} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}_{(p-1)\times p} \qquad \text{und} \qquad \mathbf{c}_0 = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}_{p-1}, \qquad \text{wobei } \text{Rang}(\mathbf{C}) = p-1.$$

Die Hypothese $H_0 : \beta_1 = \beta_2 = \beta_3$, also auf Gleichheit der Regressionskoeffizienten der ersten drei Designterme ist als allgemeine lineare Hypothese $H_0 : \mathbf{C}\boldsymbol{\beta} = \mathbf{c}_0$ formulierbar mit z. B.

$$\mathbf{C} = \begin{pmatrix} 0 & 1 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & -1 & 0 & \cdots & 0 \end{pmatrix}_{2\times p} \quad \text{und} \quad \mathbf{c}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}_2, \qquad \text{wobei } \mathrm{Rang}(\mathbf{C}) = 2.$$

In **R** werden die Covariablenvektoren und der numerische Response-Vektor $\mathbf{Y}$ typischerweise gemeinsam in einem Data Frame zusammengefasst und die Designmatrix $\mathbf{X}$ durch eine Modellformel als Funktion der Covariablen beschrieben. Die Anpassung (= der Fit) des linearen Regressionsmodells wird durch die Funktion `lm()` (von "**l**inear **m**odel") bewerkstelligt. Ihre wichtigsten Argumente sind `formula` und `data`. Dabei spezifiziert `formula` (auf eine recht kompakte Weise) das Modell. Die Daten, sprich (Co-)Variablen werden aus dem `data` zugewiesenen Data Frame entnommen. Das Resultat der Funktion `lm()` ist ein sogenanntes `lm`-Objekt (der Klasse `lm`), das neben dem Kleinste-Quadrate-Schätzer (KQS) $\hat{\beta}$ für β noch weitere diagnostisch sowie inferenzstatistisch wichtige Größen enthält.

An dieser Stelle seien im Vorgriff auf Kommendes schon einmal ein paar mehr – aber auf jeden Fall unvollständige – Literaturhinweise gegeben:

- **R**-zentrierte und praktisch angelegte Bücher fast ohne Theorie, sondern mit zahlreichen bearbeiteten Beispielen sind [29, Faraway (2005)] und [30, Faraway (2006)] sowie [34, Fox (2002)].
- In [79, Venables & Ripley (2003)], [56, Maindonald & Braun (2010)], [81, Verzani (2005)], [28, Everitt & Hothorn (2010)] und [26, Dalgaard (2008)] werden mit starkem **R**-Bezug neben vielen anderen Themen auch lineare Modelle bzw. damit zu tun habende Verfahren in unterschiedlicher Breite und Tiefe behandelt.
- Praxis (aber ohne Bezug zu **R**) und Theorie sind kombiniert und umfangreich zu finden in [42, Harrell (2002)], [33, Fox (1997)] und [31, Fahrmeir et al. (2007)] (und zwar auf Deutsch).
- Mathematisch wird die Thematik in [46, Hocking (1996)] (oder [47, Hocking (2003)], aber in äußerst bescheidener Druckqualität!), [70, Searle (1971)] und [71, Seber (1977)] sowie [40, Graybill (1976)] intensiv entwickelt und ausgebreitet.

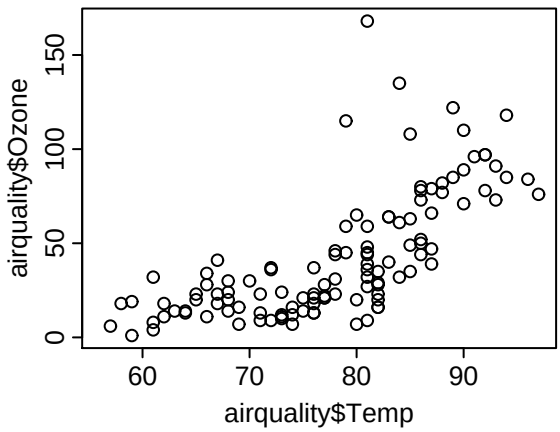
10.2 Die einfache lineare Regression: Modellanpassung & -zusammenfassung

Im Fall der einfachen linearen Regression besteht $\mathbf{X}$ nur aus zwei Spalten (d. h., $p = 2$) und die erste Spalte von $\mathbf{X}$ nur aus Einsen. Das Modell reduziert sich dann zu

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \quad \text{für } i = 1, \ldots, n$$

Zur Illustration verwenden wir den in **R** eingebauten Datensatz `airquality`, der ein Data Frame ist und in seiner Komponente `Ozone` Ozon-Konzentrationen enthält, die jeweils bei Temperaturen beobachtet wurden, die in der Komponente `Temp` verzeichnet sind. Hier ist ein Ausschnitt aus `airquality` und das Streudiagramm von `Ozone` gegen `Temp` gezeigt:

```
> airquality
    Ozone Solar.R Wind Temp Month Day
  1    41     190  7.4   67     5   1
  2    36     118  8.0   72     5   2
  3    12     149 12.6   74     5   3
  4    18     313 11.5   62     5   4
  5    NA      NA 14.3   56     5   5
  6    28      NA 14.9   66     5   6
  7    23     299  8.6   65     5   7
....
153    20     223 11.5   68     9  30
```



Wir wollen den Ozon-Gehalt in Abhängigkeit von der Temperatur durch eine einfache lineare Regression modellieren, d. h., es soll das Modell

$$\texttt{Ozone}_i = \beta_0 + \beta_1 \cdot \texttt{Temp}_i + \varepsilon_i \qquad \text{für } i = 1, \ldots, n$$

mit $\varepsilon_1, \ldots, \varepsilon_n$ i.i.d. $\sim \mathcal{N}(0, \sigma^2)$ „gefittet“, also an die Daten angepasst werden. In **R** wird das wie folgt realisiert:

Modellanpassung & -zusammenfassung

```
> (oz1.lm <- lm(formula = Ozone ~ Temp,
+ data = airquality))

Call: lm(formula = Ozone ~ Temp,
data = airquality)

Coefficients:
(Intercept)         Temp
   -146.995        2.429
```

Fittet eine einfache lineare Regression von `Ozone` an `Temp` (beide aus dem Data Frame `airquality`) und speichert sie im `lm`-Objekt `oz1.lm`. Die Tilde `~` im Argument `formula` bedeutet, dass die links von ihr stehende Variable von der rechts von ihr stehenden „abhängen“ soll. Ist eine dieser Variablen nicht in dem `data` zugewiesenen Data Frame vorhanden, wird sie unter den benutzerdefinierten Variablen im workspace gesucht.
Resultate: Der Funktionsaufruf (`Call`) und die Koeffizientenschätzer $\hat{\beta}_0$ (`(Intercept)`) und $\hat{\beta}_1$ (`Temp`).

```
> lm( Ozone ~ Temp, airquality)
```

.... (Argumentezuweisung per Position geht auch.)

```
> summary( oz1.lm)
Call: lm( formula = Ozone ~ Temp,
data = airquality)

Residuals:
   Min     1Q Median     3Q    Max
-40.73 -17.41  -0.59  11.31 118.27
```

Die Anwendung von `summary()` auf ein `lm`-Objekt liefert detaillierte Informationen über das gefittete Modell: Der `Call`, ...

die “summary statistics” der Residuen (`Residuals`) sowie ...

```
Coefficients:
             Estimate Std. Error
(Intercept) -146.9955    18.2872
Temp           2.4287     0.2331

    t value Pr(>|t|)
     -8.038 9.37e-13 ***
     10.418  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’
0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

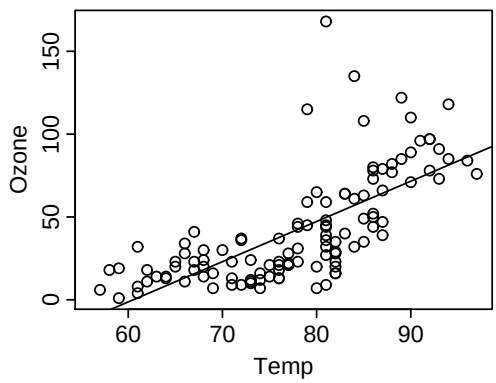
die `Coefficients`-„Tabelle“, in der zusätzlich zu den Koeffizientenschätzern (`Estimate`) deren geschätzte Standardabweichung (`Std. Error`) stehen, d. h. $\hat{\sigma}\sqrt{((\mathbf{X}'\mathbf{X})^{-1})_{jj}}$, und die Werte der t-Teststatistiken (`t value`) der zweiseitigen Hypothesentests $H_0 : \beta_{j-1} = 0$ (für $j = 1, 2$) samt deren p-Werten (`Pr(>|t|)`).
Die Markierungen durch Sternchen oder andere Symbole sind in der Legende `Signif. codes` darunter erläutert.

```
Residual standard error: 23.71 on
114 degrees of freedom
 (37 observations deleted due to
  missingness)
Multiple R-Squared: 0.4877,
Adjusted R-squared: 0.4832
F-statistic: 108.5 on 1 and 114 DF,
p-value: < 2.2e-16
```

Es folgen die Residuenstandardabweichung $\hat{\sigma} = \sqrt{\mathrm{RSS}/(n-p)}$ (`Residual standard error`) als Schätzer für σ mit ihren $n-p$ Freiheitsgraden, evtl. ein Hinweis bzgl. fehlender Werte, der multiple R^2-Wert (`Multiple R-Squared`, hier: Quadrat des empirischen Pearsonschen Korrelationskoeffizienten der (Y_i, x_i)), das korrigierte R_a^2 (`Adjusted R-Squared`), der Wert der F-Teststatistik (`F-statistic`) des globalen F-Tests $H_0 : \beta_1 = \ldots = \beta_{p-1} = 0$ mit ihren Freiheitsgraden $p-1$ und $n-p$ sowie dessen p-Wert.

Die Überlagerungung des Streudiagramms der Daten durch die gefittete Regressionsfunktion ist im Fall der einfachen linearen Regression sehr leicht, da es sich um eine *Gerade* (im $\mathbb{R}^2$) handelt: Die Funktion `abline()` kann aus einem `lm`-Objekt die beiden Koeffizienten extrahieren und die dazugehörige Gerade in den Plot einzeichnen:

```
> plot( airquality$Temp, airquality$Ozone)
> abline( oz1.lm)
```



Bemerkung: Anscheinend hat `lm()` keine Probleme damit, dass in `airquality` (mindestens) in der Komponente `Ozone` `NA`s auftreten, wie z. B. in Zeile 5. Begründung: Per Voreinstellung (in **R**) werden für den Modellfit diejenigen Zeilen des Data Frames eliminiert, die in den verwendeten Komponenten (hier `Ozone` und `Temp`) `NA`s enthalten. Daher ist nicht mit 153 Beobachtungen (so viele wie der Data Frame Zeilen hat) gerechnet worden, sondern nur mit den 116, für die gleichzeitig `Ozone` und `Temp` bekannt sind.

Der Ausschluss von Zeilen mit `NA`s geschieht – außer in der Ausgabe von `summary()` – ohne eine Mitteilung, sodass man bei Modellen mit verschiedenen Covariablen nicht davon ausgehen kann, dass sie auf demselben Datenumfang basieren. Um jedoch unterschiedlich komplexe Modelle miteinander vergleichen zu können, muss dies der Fall sein. Sicherheitshalber sollte man daher *vor* dem Beginn des „Modellbaus“ den Data Frame um alle `NA`-Zeilen bereinigen, die einem „in die Quere“ kommen könnten. Dies ist z. B. durch die Funktion `na.exclude()` möglich, die aus ihrem Argument alle `NA`-Zeilen ausschließt:

Wir eliminieren nun aus `airquality` die `NA`-Zeilen, speichern den bereinigten Data Frame unter dem Namen `air` (siehe rechts) und werden ab sofort nur noch mit diesem die Modellfits durchführen. Offenbar bleiben insgesamt nur $n = 111$ vollständige Zeilen übrig.

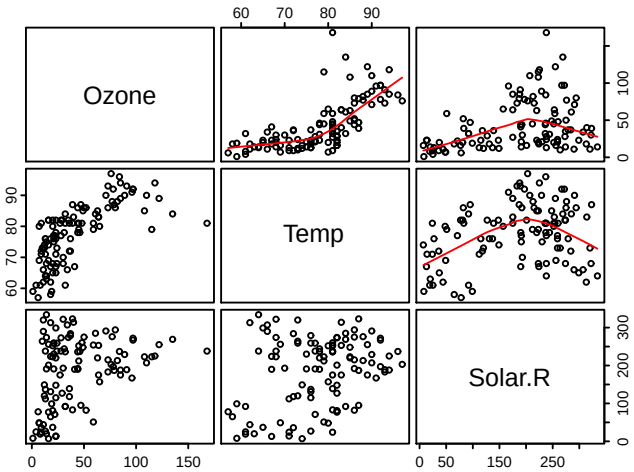
```
> (air <- na.exclude( airquality));  dim( air)
    Ozone Solar.R Wind Temp Month Day
1      41     190  7.4   67     5   1
2      36     118  8.0   72     5   2
3      12     149 12.6   74     5   3
4      18     313 11.5   62     5   4
7      23     299  8.6   65     5   7
....
153    20     223 11.5   68     9  30
[1] 111   6
```

10.3 Die multiple lineare Regression: Anpassung & -zusammenfassung

Der nebenstehende Plot aller paarweisen Streudiagramme (samt gewisser lowess-Kurven) von `Ozone`, `Temp` und `Solar.R` aus dem Data Frame `air`, erzeugt durch

```
> pairs( ~ Ozone + Temp + Solar.R,
+ data = air, cex = 0.6, gap = 0.3,
+ cex.labels = 1.2, cex.axis = 0.7,
+ upper.panel = panel.smooth)
```

deutet darauf hin, dass eventuell auch die Strahlung einen Einfluss auf die Ozon-Werte hat.



Daher soll ein Modell untersucht werden, in dem `Ozone` gemeinsam durch `Temp` und `Solar.R` beschrieben wird. Die Anpassung des multiplen linearen Regressionsmodells

$$\texttt{Ozone}_i = \beta_0 + \beta_1 \cdot \texttt{Temp}_i + \beta_2 \cdot \texttt{Solar.R}_i + \varepsilon_i \qquad \text{für } i = 1, \ldots, n$$

(also hier für $p = 3$) mit $\varepsilon_1, \ldots, \varepsilon_n$ i.i.d. $\sim \mathcal{N}(0, \sigma^2)$ ist genauso einfach wie der Fall der einfachen linearen Regression und wird anhand des folgenden Beispiels verdeutlicht:

Modellanpassung & -zusammenfassung

```
> summary( oz2.lm <- lm( Ozone ~ Temp +
+ Solar.R, air))
Call: lm(formula = Ozone ~ Temp + Solar.R,
data = air)

Residuals:
    Min      1Q  Median      3Q     Max
-36.610 -15.976  -2.928  12.371 115.555

Coefficients:
              Estimate Std. Error t value
(Intercept) -145.70316   18.44672  -7.899
Temp           2.27847    0.24600   9.262
Solar.R        0.05711    0.02572   2.221

    Pr(>|t|)
    2.53e-12 ***
    2.22e-15 ***
      0.0285 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’
0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 23.5 on 108
degrees of freedom
Multiple R-Squared: 0.5103,
Adjusted R-squared: 0.5012
F-statistic: 56.28 on 2 and 108 DF,
p-value: < 2.2e-16
```

Fittet eine multiple lineare Regression von `Ozone` an `Temp` und `Solar.R` (beide aus dem Data Frame `air`) und speichert sie als `lm`-Objekt `oz2.lm`.
Dabei soll die links der Tilde `~` in der Formel stehende Variable in einem multiplen linearen Modell **additiv** von den rechts von ihr stehenden Variablen abhängen. Dies wird durch das `+` zwischen den beiden unabhängigen Variablen erreicht.
(Es sind noch weitere, komplexere Verknüpfungsoperatoren für die unabhängigen Variablen zulässig, auf die wir in Abschnitt 10.4 eingehen.)

Resultate: Völlig analog zu denen im einfachen linearen Modell.

```
> summary( oz2.lm, correlation = TRUE)
... (Von Call bis p-value dieselbe Ausgabe wie
oben) ...

Correlation of Coefficients:
        (Intercept) Temp
Temp    -0.96
Solar.R  0.05       -0.29
```

Mit dem `summary()`-Argument `correlation = TRUE` wird zusätzlich das linke untere Dreieck (ohne Diagonale) der Matrix der geschätzten Korrelationen der Koeffizienten ausgegeben:

$$\left(\frac{((\mathbf{X}'\mathbf{X})^{-1})_{jk}}{\sqrt{((\mathbf{X}'\mathbf{X})^{-1})_{jj}}\sqrt{((\mathbf{X}'\mathbf{X})^{-1})_{kk}}} \right)_{1 \le j,k \le p}$$

Formulierung und Fit von komplizierteren Modellen, d. h., mit noch mehr Covariablen sind "straightforward" durch Erweiterung der Modellformel zu erreichen. Eine grafische Darstellung ist jedoch nur noch für zwei Covariablen mittels der dreidimensionalen „Response-Oberfläche" möglich, was wir in §10.9.1 „Schätzwerte für die Regressionsfunktion und grafische Darstellung" behandeln. Bei mehr als zwei Covariablen versagt unsere Anschauung.

10.3.1 Komponenten und Diagnoseplots eines linearen Modells

Um an die Koeffizientenschätzwerte $\hat{\boldsymbol{\beta}}' = (\hat{\beta}_0, \ldots, \hat{\beta}_{p-1})$, die Residuen $\hat{\varepsilon}_i = Y_i - \hat{Y}_i$, die gefitteten Werte $\hat{Y}_i$, $i = 1, \ldots, n$, oder die RSS eines Fits heranzukommen, stehen die folgenden, auf `lm`-Objekte anwendbaren Funktionen zur Verfügung:

Extraktion von Komponenten eines linearen Regressionsmodells:	

`> coef( oz2.lm)`
`(Intercept)  Temp  Solar.R`
`-145.7031551  2.2784668  0.0571096`
 	
`> coef( summary( oz2.lm))`
`  Estimate  Std. Error`
`(Intercept) -145.70315510 18.44671758`
`Temp  2.27846684  0.24599582`
`Solar.R  0.05710959  0.02571885`
 	
`  t value  Pr(>|t|)`
`  -7.898595 2.529334e-12`
`  9.262218 2.215559e-15`
`  2.220534 2.847063e-02`
	Die Funktion `coefficients()` (Abk. `coef()`) angewendet auf das `lm`-Objekt einer linearen Regression liefert die gefitteten Koeffizienten als Vektor $\hat{\boldsymbol{\beta}}' = \left(\hat{\beta}_0, \ldots, \hat{\beta}_{p-1}\right)$ zurück. `coef()` angewendet auf ein `summary.lm`-Objekt, also das Ergebnis von `summary()` für ein `lm`-Objekt, liefert die ganze `Coefficients`-Tabelle als Matrix.

`> resid( oz2.lm)`
`  1  2  3 ....`
`23.195054  10.914611  -19.412720 ....`
 	
`> fitted( oz2.lm)`
`  1  2  3 ....`
`17.80495  25.08539  31.41272 ....`
	Die Funktion `residuals()` (Abk. `resid()`) angewendet auf ein `lm`-Objekt liefert den Residuenvektor $\hat{\boldsymbol{\varepsilon}}' = (\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n)$ und die Funktion `fitted()` den Vektor der gefitteten Werte $\hat{\mathbf{Y}}' = (\hat{Y}_1, \ldots, \hat{Y}_n)$.

`> deviance( oz2.lm)` `[1] 59644.36`	Liefert die "deviance" des gefitteten Modells, was im Fall eines `lm`-Objektes die RSS ist.

Zur qualitativen Diagnose des Fits eines linearen Modells dienen die folgenden grafischen Darstellungen:

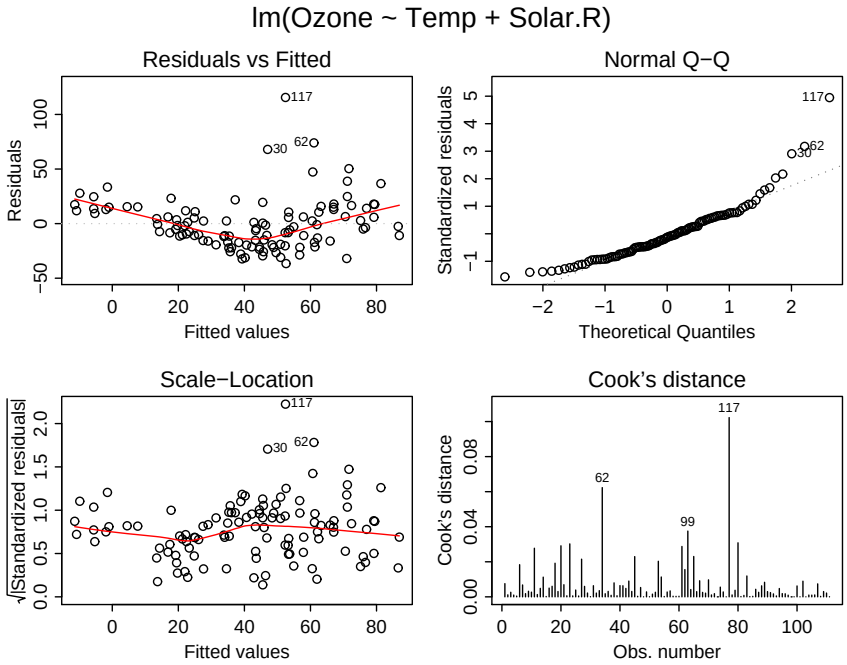
Ausgewählte Diagnoseplots für ein lineares Regressionsmodell:	
`> plot( oz2.lm,` `+ which = 1:4)`	Liefert für das in `oz2.lm` gespeicherte lineare Modell (wg. `which = 1:4`) die vier unten beschriebenen und auf der nächsten Seite stehenden Plots.

Im Bild auf der nächsten Seite von links nach rechts und von oben nach unten:

1. Die Residuen $\hat{\varepsilon}_i$ gegen die gefitteten Werte $\hat{Y}_i$. Voreinstellungsgemäß werden die drei extremsten Beobachtungen als potenzielle Ausreißer durch ihre Namen (bzw. Indexnummern, falls unbenannt) markiert. Es sollte kein Trend oder Muster in der Punktewolke erkennbar sein (wofür die theoretische Begründung auf Seite 211 geliefert wird).
2. Ein Normal-Q-Q-Plot der standardisierten Residuen $\hat{\varepsilon}_i/(\hat{\sigma}\sqrt{1-h_{ii}})$, um die Plausibilität der Normalverteilungsannahme der Fehler ε_i zu beurteilen. (Auf die standardisierten Residuen gehen wir in Abschnitt 10.8 „Modelldiagnose II: Ausreißer, Extrempunkte, einflussreiche Punkte und Residualanalyse“ näher ein. Aber schon jetzt weisen wird daraufhin, dass im Falle $h_{ii} = 1$ das dazugehörige Residuum – mit einer entsprechenden Warnmeldung – bei der Anfertigung des Q-Q-Plots einfach ausgeschlossen wird.) Wiederum werden die drei (verbliebenen) extremsten Beobachtungen durch ihre Namen bzw. Indexnummern markiert.

   Bemerkung: Im Paket `car` steht eine sehr nützliche Funktion namens `qqPlot()` zur Verfügung, die für ein `lm`-Objekt einen Q-Q-Plot liefert, der um ein Band punktweiser Konfidenzintervalle ergänzt ist, welches die Beurteilung der Adäquatheit der Verteilungsannahme erleichtert. (Zur Theorie desselben siehe §12.1.1 in [33, Fox (1997)].)

3. Ein sogenannter "scale-location"-Plot der Wurzel der absoluten standardisierten Residuen $\sqrt{|\hat{\varepsilon}_i/(\hat{\sigma}\sqrt{1-h_{ii}})|}$ gegen die gefitteten Werte.
4. Ein Plot der Cook-Abstände (auf die wir ebenfalls noch in Abschnitt 10.8 „Modelldiagnose II: Ausreißer, Extrempunkte, einflussreiche Punkte und Residualanalyse“ zu sprechen kommen) gegen ihre Indizes, um potenziell einflussreiche Beobachtungen zu identifizieren. Die drei einflussreichsten sind auch hier markiert.



10.4 Zur Syntax von Modellformeln

Das Konzept des Formelausdrucks (`formula`) in **S** und somit auch in **R** erlaubt es, eine Vielzahl von Modellen für die Beziehung zwischen Variablen auf sehr kompakte Weise zu spezifizieren und für verschiedene statistische Verfahren in einheitlicher Form zu verwenden (also nicht nur für lineare Regressionsmodelle). Wir konzentrieren uns zunächst auf ihre Verwendung für stetige Designvariablen in linearen Regressionsmodellen. Auf diskrete Designvariablen gehen wir im Abschnitt 10.11 „Faktorvariablen und Interaktionsterme im linearen Regressionsmodell“ ein.

In den Abschnitten 10.2 und 10.3 sind uns schon zwei einfache Versionen von Formelausdrücken begegnet. Mittels der Funktion `formula()` kann ein solcher Formelausdruck auch als ein eigenständiges Objekt der Klasse `formula` erzeugt und gespeichert werden.

Formelobjekte:	

`> oz.form1 <- formula( Ozone ~ Temp)`
`> oz.form1`
`Ozone ~ Temp`
	Erzeugt und speichert in `oz.form1` die Formel des einfachen linearen Modells
	$\texttt{Ozone}_i \;=\; \beta_0 + \beta_1\,\texttt{Temp}_i + \varepsilon_i$

Anhand von einigen Beispielformeln soll die Formelsyntax erläutert werden:

S-Formelausdruck:	Bedeutung/Bemerkungen:
`Ozone ~ 1`	Das sogenannte „Null-Modell“, das nur aus dem konstanten Term β_0 besteht: $\texttt{Ozone}_i = \beta_0 \;+\; \varepsilon_i$
`Ozone ~ Temp + Solar.R`	Das (bereits bekannte) Zwei-Variablen-Modell **mit** konstantem Term β_0: $\texttt{Ozone}_i \;=\; \beta_0 + \beta_1\,\texttt{Temp}_i + \beta_2\,\texttt{Solar.R}_i \;+\; \varepsilon_i$

`Ozone ~ -1 + Temp + Solar.R`	Obiges Modell **ohne** den konstanten Term β_0 (genannt „Regression durch den Ursprung“): $$\texttt{Ozone}_i \;=\; \beta_1\,\texttt{Temp}_i + \beta_2\,\texttt{Solar.R}_i \;+\; \varepsilon_i$$
`Ozone ~ Temp + Solar.R + Temp:Solar.R`	Zwei-Variablen-Modell mit Interaktion (`a:b` bedeutet die Interaktion zwischen den Variablen `a` und `b`; zur Erläuterung siehe Abschnitt 10.5): $$\texttt{Ozone}_i \;=\; \beta_0 + \beta_1\,\texttt{Temp}_i + \beta_2\,\texttt{Solar.R}_i + \beta_3\,\texttt{Temp}_i\,\texttt{Solar.R}_i \;+\; \varepsilon_i$$
`Ozone ~ Temp * Solar.R`	Ist äquivalent zum Zwei-Variablen-Interaktionsmodell von eben und lediglich eine Abkürzung.
`Ozone ~ Temp * Solar.R * Wind`	Drei-Variablen-Modell mit allen drei Zweifach-Interaktionen und der Dreifach-Interaktion. Es ist äquivalent zu `Ozone ~ Temp + Solar.R + Wind + Temp:Solar.R + Temp:Wind + Solar.R:Wind + Temp:Solar.R:Wind`
`Ozone ~ Temp * Solar.R * Wind - Temp:Solar.R:Wind`	Drei-Variablen-Modell mit allen drei Zweifach-Interaktionen, aber **ohne** die Dreifach-Interaktion; äquivalent zu `Ozone ~ Temp + Solar.R + Wind + Temp:Solar.R + Temp:Wind + Solar.R:Wind` (Mit dem Minuszeichen lässt sich *jeder* Term aus einer Modellformel wieder entfernen.)
`Ozone ~ (Temp + Solar.R + Wind)^2`	Eine dritte Notation für das Modell von eben mit allen Zweifach-Interaktionen, aber ohne die Dreifach-Interaktion. (Mit m an Stelle der 2 werden alle Terme bis zur „Interaktionsordnung“ m eingebaut.)
`Ozone ~ Temp^2`	*Keine* polynomiale Regression, sondern das Ein-Variablen-Interaktionsmodell mit allen Zweifach-Interaktionen von `Temp` „mit sich selbst“ und somit gleich `Ozone ~ Temp`. (Für polynomiale Regression siehe Abschnitt 10.10.)

Offenbar haben die Operatoren `+`, `-`, `:`, `*` und `^m` in Formeln eine spezielle, „nicht-arithmetische“ Bedeutung, wenn sie rechts der Tilde `~` auftreten. (Dies gilt auch für den noch nicht aufgeführten Operator `/`, auf dessen Sonderbedeutung wir erst in Abschnitt 10.11 „Faktorvariablen und Interaktionsterme im linearen Regressionsmodell“ eingehen.) Andererseits dürfen die rechts wie links vom `~` in einer Formel auftretenden Variablen durch alle Funktionen transformiert werden, deren Resultat wieder als eine Variable interpretierbar ist. Dies trifft insbesondere auf alle mathematischen Funktionen wie `log()`, `sqrt()` etc. zu (wie sie z. B. bei der Anwendung Varianz stabilisierender Transformationen (§10.6.2) oder linearisierender Transformationen (§10.6.3) zum Einsatz kommen).

Jedoch auch Funktionen, deren Ergebnis als *mehrere* Variablen aufgefasst werden können, sind zulässig. Ein Beispiel hierfür ist die Funktion `poly()`: Sie erzeugt Basen von Orthonormalpolynomen bis zu einem gewissen Grad, die im Zusammenhang mit der polynomialen Regression eine wichtige Rolle spielen und in Abschnitt 10.10 „Polynomiale Regression“ behandelt werden.

Die oben genannten Operatoren haben nur auf der „obersten Ebene“ einer Formel ihre spezielle Bedeutung, *innerhalb* eines Funktionsaufrufs in einer Formel bleibt es bei ihrer „arithmetischen“ Bedeutung. D. h., dass die zum Beispiel aus zwei Variablen u und v abgeleitete Variable $x := \log(u+v)$ in **S** als `log( u + v)` formuliert wird. Sollen die Operatoren jedoch auf der *obersten* Formelebene ihre arithmetische Bedeutung haben, so sind die betroffenen Terme in die Funktion `I()`, wie “**i**nhibit interpretation”, „einzupacken“. Damit also $x := u + v$ als Summe von u und v

eine einzelne Covariable darstellt, ist in einer **S**-Formel der Ausdruck `I( u + v)` zu verwenden, oder falls $x := u^n$, also die n-te Potenz von u eine Covariable sein soll, muss `I( u^n)` verwendet werden.

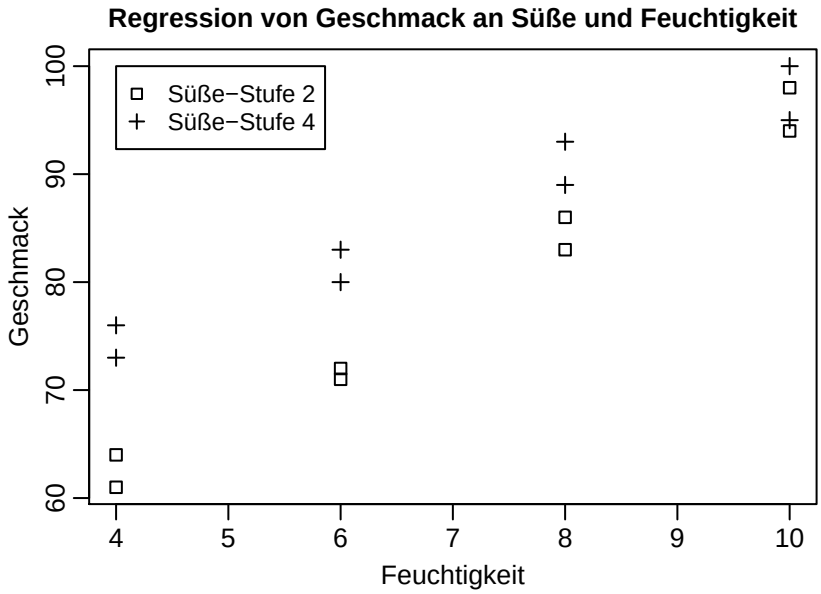
Es folgen einige Formelbeispiele mit transformierten Variablen:

S-Formelausdruck:	Bedeutung:
`log( Ozone) ~ I( Temp^2) + sqrt( Solar.R)`	Transformationen aller Variablen sind möglich; hier haben wir das Modell $\log(\texttt{Ozone}_i) = \beta_0 + \beta_1\ (\texttt{Temp}_i)^2 + \beta_2\ \sqrt{\texttt{Solar.R}_i} + \varepsilon_i$
`log( Ozone) ~ exp( Temp^2) + sqrt( (Solar.R/Wind)^3)`	*Innerhalb* einer Funktion agieren die arithmetischen Operatoren gemäß ihrer mathematischen Definition: $\log(\texttt{Ozone}_i) = \beta_0 + \beta_1\ \exp\left((\texttt{Temp}_i)^2\right) + \beta_2\ \sqrt{(\texttt{Solar.R}_i/\texttt{Wind}_i)^3} + \varepsilon_i$
`log( Ozone) ~ I( 1/Wind)`	Einsatz der Funktion `I()`, damit `/` die Bedeutung der Division hat: $\log(\texttt{Ozone}_i) = \beta_0 + \beta_1\ 1/\texttt{Wind}_i + \varepsilon_i$
`Ozone ~ I( (Temp + Solar.R + Wind)^2)`	Beachte den Unterschied zum Modell mit allen Interaktionen bis zur „Interaktionsordnung“ 2 auf der vorherigen Seite: $\texttt{Ozone}_i = \beta_0 + \beta_1\ (\texttt{Temp}_i + \texttt{Solar.R}_i + \texttt{Wind}_i)^2 + \varepsilon_i$

10.5 Zur Interaktion stetiger Covariablen

An einem Beispiel soll die Bedeutung einer Interaktion zwischen zwei stetigen Variablen veranschaulicht werden. In dem unten gezeigten Data Frame `marke.df` sind die Resultate eines kontrollierten Experimentes zur Beurteilung des Geschmacks (auf einer metrischen Skala zwischen 0 und 100) einer gewissen Süßware für verschiedene Feuchtigkeits- und Süße-Stufen dokumentiert und in dem nebenstehenden Streudiagramm veranschaulicht.

```
> marke.df
   Ge-     Feuch-
   schmack tigkeit Suesse
 1      64       4      2
 2      73       4      4
 3      61       4      2
 4      76       4      4
 5      72       6      2
 6      80       6      4
 7      71       6      2
 8      83       6      4
 9      83       8      2
10      89       8      4
11      86       8      2
12      93       8      4
13      98      10      2
14      95      10      4
15      94      10      2
16     100      10      4
```



Aufgrund der Markierung der Beobachtungen durch zwei verschiedene Symbole für die beiden Süße-Stufen ist in dem Streudiagramm deutlich zu erkennen, dass für verschiedene Süße-Stufen der Einfluss (!) der Feuchtigkeit auf die Geschmacksbeurteilung des Produktes ein anderer ist. D. h., der quantitative Einfluss der Covariablen „Feuchtigkeit“ auf die Response „Geschmack“ hängt vom Wert der Covariablen „Süße“ ab. Dies nennt man eine Interaktion, und um diesem Effekt Rechnung zu tragen, muss in das aufzustellende Zwei-Variablen-Modell ein geeigneter Interaktionsterm eingebaut werden.

Die einfachste Form der Interaktion stetiger Covariablen ist die multiplikative, weswegen in einem Zwei-Variablen-Modell zusätzlich zu den Haupteffekten β_1 und β_2 der beiden Covariablen ein Interaktionseffekt β_3 für das Produkt der beiden Covariablen eingebaut wird. Das Modell lautet damit

$$\texttt{Geschmack}_i = \beta_0 + \beta_1\ \texttt{Feuchtigkeit}_i + \beta_2\ \texttt{Suesse}_i + \beta_3\ \texttt{Feuchtigkeit}_i\ \texttt{Suesse}_i + \varepsilon_i$$

In kompakter **S**-Formelsyntax: `Geschmack ~ Feuchtigkeit * Suesse`

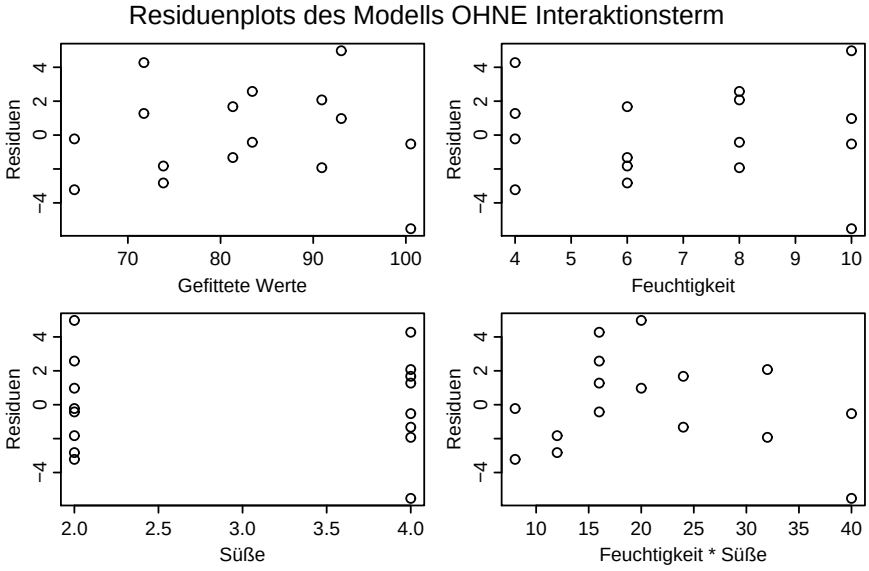
Zunächst erstellen wir einen Fit *ohne* Interaktionsterm:

```
> summary( marke.lm <- lm( Geschmack ~ Feuchtigkeit + Suesse, marke.df))

Call: lm(formula = Geschmack ~ Feuchtigkeit + Suesse, data = marke.df)

Residuals:
   Min     1Q Median     3Q    Max
-5.525 -1.850 -0.325  1.775  4.975

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   37.5250     3.3451  11.218 4.67e-08 ***
Feuchtigkeit   4.8000     0.3362  14.277 2.53e-09 ***
Suesse         3.7500     0.7518   4.988 0.000248 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.007 on 13 degrees of freedom
Multiple R-Squared: 0.9462,     Adjusted R-squared: 0.9379
F-statistic: 114.4 on 2 and 13 DF,  p-value: 5.611e-09
```

Die Plots der Residuen gegen die gefitteten Werte sowie gegen jede der Covariablen und gegen den *nicht* im Modell befindlichen Interaktionsterm mittels

```
> plot( fitted( marke.lm), resid( marke.lm), xlab = "Gefittete Werte",
+ ylab = "Residuen")
> plot( marke.df$Feuchtigkeit, resid( marke.lm), xlab = "Feuchtigkeit",
+ ylab = "Residuen")
> plot( marke.df$Suesse, resid( marke.lm), xlab = "Süße",
+ ylab = "Residuen")
> plot( marke.df$Feuchtigkeit * marke.df$Suesse, resid( marke.lm),
+ xlab = "Feuchtigkeit * Süße", ylab = "Residuen")
```

ergeben die Bilder auf der folgenden Seite oben. Dort zeigt der Plot der Residuen gegen den Interaktionsterm (rechts unten) eine leichte Krümmung in der Punktewolke, was den Schluss zulässt, dass eine Interaktion vorliegen könnte.



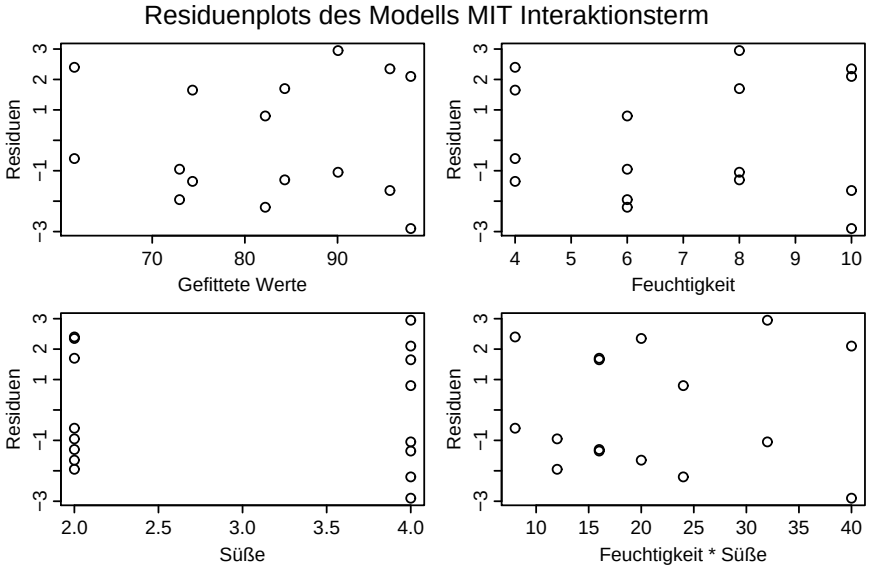
Daher fertigen wir nun den Fit *mit* Interaktionsterm an, und es zeigt sich, dass die Anpassung an Qualität gewinnt: Die Residuenstandardabweichung wird kleiner, die R^2-Werte werden größer und man beachte den signifikanten Beitrag des Interaktionsterms in diesem Modell:

```
> summary( marke2.lm <- lm( Geschmack ~ Feuchtigkeit * Suesse, marke.df))

Call: lm(formula = Geschmack ~ Feuchtigkeit * Suesse, data = marke.df)

Residuals:
   Min     1Q Median     3Q    Max
-2.900 -1.425 -0.775  1.800  2.950

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)          19.1500     5.6275   3.403 0.005241 **
Feuchtigkeit          7.4250     0.7658   9.696    5e-07 ***
Suesse                9.8750     1.7796   5.549 0.000126 ***
Feuchtigkeit:Suesse  -0.8750     0.2422  -3.613 0.003559 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.166 on 12 degrees of freedom
Multiple R-Squared: 0.9742,     Adjusted R-squared: 0.9678
F-statistic: 151.3 on 3 and 12 DF,  p-value: 8.47e-10
```

Auch die Residuenplots unterstützen dieses Modell, wie in den Grafiken auf der folgenden Seite oben zu sehen: Im Plot der Residuen gegen den Interaktionsterm (rechts unten) ist die leichte Krümmung in der Punktwolke verschwunden.

Bemerkung: Natürlich ist nicht gesagt, dass multiplikative Interaktionen die einzig möglichen sind. Eine allgemeine Interaktion zwischen Covariablen $x_1, \ldots, x_k$ wäre von der Form $f(x_1, \ldots, x_k)$, aber das wesentliche Problem dürfte darin bestehen, f richtig zu spezifizieren. In **S** steht im Rahmen der linearen Modelle für f bei stetigen Covariablen über den Operator `:` nur die Multiplikation zur Verfügung, was auch die einzige Form von Interaktion stetiger Covariablen im linearen Modell zu sein scheint, die in Lehrbüchern explizit behandelt wird. Unter der Abkürzung GAM für “Generalized Additive Models” werden entsprechende Verallgemeine-

rungen des linearen Modells in der Theorie behandelt und stehen in **R** z. B. in den Paketen `mgcv` und `gam` zur Verfügung.



10.6 Modelldiagnose I: Residualanalyse und Transformationen des Modells

Nach dem, was wir am Beginn von Kapitel 10 zur Erinnerung aufgeführt haben, sind die Residuen $\hat{\varepsilon}_i := Y_i - \hat{Y}_i$, $i = 1, \ldots, n$, unter den üblichen Modellannahmen zwar **normalverteilt, aber weder unabhängig noch identisch verteilt**, sondern mit

$$\mathrm{Var}(\hat{\varepsilon}_i) = \sigma^2\,(1 - h_{ii}) \qquad \text{und} \qquad \mathrm{Cov}(\hat{\varepsilon}_i, \hat{\varepsilon}_j) = -\sigma^2\, h_{ij} \quad \text{für } 1 \le i \ne j \le n,$$

wobei $(h_{ij})_{1 \le i,j \le n} = \mathbf{H}$ die schon eingeführte Projektionsmatrix ist.

Beachte: Im *einfachen linearen Regressionsmodell* mit $\hat{\varepsilon}_i := Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i$ für $i = 1, \ldots, n$, ist

$$h_{ij} = \frac{1}{n} + \frac{(x_i - \bar{x}_n)(x_j - \bar{x}_n)}{\sum_{l=1}^n (x_l - \bar{x}_n)^2} \qquad \text{für } 1 \le i, j \le n,$$

was für die Residuen

$$\mathrm{Var}(\hat{\varepsilon}_i) = \sigma^2\,(1 - h_{ii}) = \sigma^2\,\left(1 - \frac{1}{n} - \frac{(x_i - \bar{x}_n)^2}{\sum_{l=1}^n (x_l - \bar{x}_n)^2}\right)$$

liefert. Dies bedeutet, dass die Residuen solcher Designwerte, die weiter vom Mittelwert $\bar{x}_n$ entfernt sind, eine kleinere Varianz haben als solche, deren Designwerte näher bei $\bar{x}_n$ liegen. Damit können Residualplots (also $\hat{\varepsilon}_i$ gegen x_i oder gegen $\hat{y}_i$) im zentralen Bereich des Designs bei erfüllten Modellannahmen eine tendenziell größere Variabilität zeigen als in den Randbereichen (und daher eine Art leicht „linsenförmiges“ Aussehen haben dürfen)!

10.6.1 Grafische Residualanalyse

Trotz der eben erwähnten Abhängigkeit und der (meist leichten, aber grundsätzlich vorhandenen!) Varianz-Inhomogenität der Residuen sind Plots von $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$ gegen die Designvariable(n) nützlich, um aufgrund von hierbei eventuell zu beobachtenden systematischen Strukturen

- Nichtlinearität in den Designvariablen,
- Varianz-Inhomogenität (= Heteroskedastizität) in den Fehlern ε_i und/oder
- „Ausreißer“

entdecken zu können.

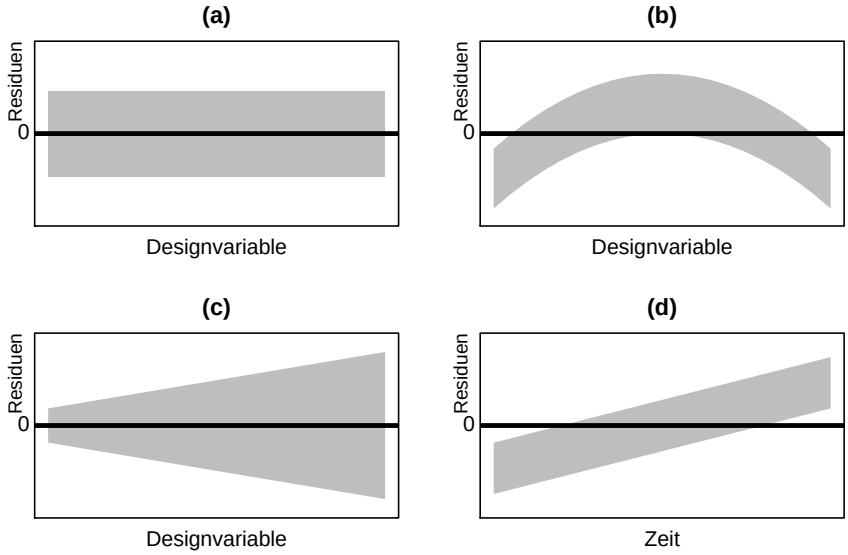
Schematische Skizzen einiger prototypischer Residualplots (s. u.) sollen dies veranschaulichen. Die grauen Bänder mögen dabei die Struktur der „Residuenwolke“ widerspiegeln. Plot ...

(a) zeigt einen idealtypischen Residualplot: Konstante Streuung (= Homoskedastizität) im gesamten Designwertebereich, keine Krümmung der Residuenwolke, keine Ausreißer.

(b) liefert durch die Krümmung ein Indiz für eine andere funktionale Beziehung als die modellierte zwischen der Response und der Designvariablen.

(c) dokumentiert eine Abhängigkeit der Residuenstreuung vom Wert der Designvariablen, also Varianz-Inhomogenität. Hier: Je größer x, desto stärker streut Y um die Regressionsfunktion.

(d) ist ein Plot der Residuen gegen ihren Index und, wenn die Daten in der Reihenfolge ihrer Indizes erhoben wurden, damit ein Plot gegen die *Zeit*. In diesem Fall impliziert er eine mögliche Abhängigkeit der Fehler voneinander.



Im einfachen linearen Regressionsmodell können äquivalent zu den obigen Plots der Residuen gegen eine Designvariable auch Plots der Residuen gegen die gefitteten Werte $\hat{Y}_1, \ldots, \hat{Y}_n$ betrachtet werden. Sie unterscheiden sich lediglich in der Skala der horizontalen Achse von den erstgenannten Plots, da die $\hat{Y}_i$ eine lineare Transformation der x_i sind, sodass der optische Eindruck der Plots derselbe ist. Außerdem sind diese Plots auch im *multiplen* Regressionsmodell (mit mehr als nur einer Designvariablen) anfertigbar und es kommt hinzu, dass die gefitteten Werte $\hat{Y}_i$ und die Residuen $\hat{\varepsilon}_i$ unter den Modellannahmen unkorreliert sind, denn:

In jedem linearen Regressionsmodell *mit* konstantem Term β_0 (also mit einer Einsenspalte in der Designmatrix $\mathbf{X}$) gilt $\mathbf{H}\mathbf{1}_n = \mathbf{1}_n$ (denn $\mathbf{H}\mathbf{X} = \mathbf{X}$), woraus mit Fakten von S. 197 unten folgt:

$$\mathbf{1}_n'\hat{\boldsymbol{\varepsilon}} = \mathbf{1}_n'(\mathbf{I}_{n\times n} - \mathbf{H})\mathbf{Y} = (\mathbf{1}_n' - \mathbf{1}_n'\mathbf{H})\mathbf{Y} = 0 \qquad \text{und} \qquad (\mathbf{I}_{n\times n} - \mathbf{H})\mathbf{1}_n = \mathbf{0}_n$$

In Konsequenz gilt für die empirische Kovarianz zwischen $\hat{Y}_1, \ldots, \hat{Y}_n$ und $\hat{\boldsymbol{\varepsilon}}_1, \ldots, \hat{\boldsymbol{\varepsilon}}_n$:

$$\begin{aligned}
(n-1)\widehat{\mathrm{cov}}(\hat{\mathbf{Y}}, \hat{\boldsymbol{\varepsilon}}) &\equiv \sum_{i=1}^{n}(\hat{Y}_i - \bar{\hat{Y}}_.)(\hat{\varepsilon}_i - \bar{\hat{\varepsilon}}_.) = (\hat{\mathbf{Y}} - n^{-1}\mathbf{1}_n\mathbf{1}_n'\hat{\mathbf{Y}})'\hat{\boldsymbol{\varepsilon}} \\
&= (\mathbf{H}\mathbf{Y} - n^{-1}\mathbf{1}_n\mathbf{1}_n'\mathbf{H}\mathbf{Y})'(\mathbf{I}_{n\times n} - \mathbf{H})\mathbf{Y} \\
&= \mathbf{Y}'(\mathbf{H} - n^{-1}\mathbf{1}_n\mathbf{1}_n'\mathbf{H})'(\mathbf{I}_{n\times n} - \mathbf{H})\mathbf{Y} \\
&= \mathbf{Y}'\left(\mathbf{H}(\mathbf{I}_{n\times n} - \mathbf{H}) - n^{-1}\mathbf{1}_n\mathbf{1}_n'(\mathbf{I}_{n\times n} - \mathbf{H})\right)\mathbf{Y} = 0,
\end{aligned}$$

sodass auch der empirische Pearsonsche Korrelationskoeffizient der $\hat{Y}_i$ und $\hat{\varepsilon}_i$ gleich Null ist (vgl. auch [46, Hocking (1996), §5.3.1]). Dies bedeutet, dass im Plot der Residuen gegen die gefitteten Werte insbesondere kein linearer Trend zu erkennen sein dürfte, wenn das Modell korrekt ist!

Falls die Diagnoseplots und die Modellannahmen im Widerspruch zueinander stehen, kann das, wie oben schon angesprochen, die folgenden möglichen Ursachen haben:

1. Die funktionale Beziehung zwischen Erwartungswert der Response und der (bzw. den) Designvariable(n), d. h. die Regressionsfunktion ist nicht korrekt formuliert (vgl. Plot (b) auf vorheriger Seite).

   Dies wiederum kann zwei Gründe haben:

   i. Die Designvariable(n) geht (gehen) nicht in der korrekten funktionalen Form in die Regressionsfunktion ein.

   ii. Die (Mess-Skala der) Response ist nicht in der adäquaten funktionalen Form für das Modell.

2. Über den Designbereich hinweg liegt Varianz-Inhomogenität der Fehler ε_i vor (s. Plot (c)).

3. Die ε_i sind nicht normalverteilt (was sich in einem Normal-Q-Q-Plot der Residuen zeigen sollte).

4. Die ε_i sind korreliert (vgl. Plot (d)).

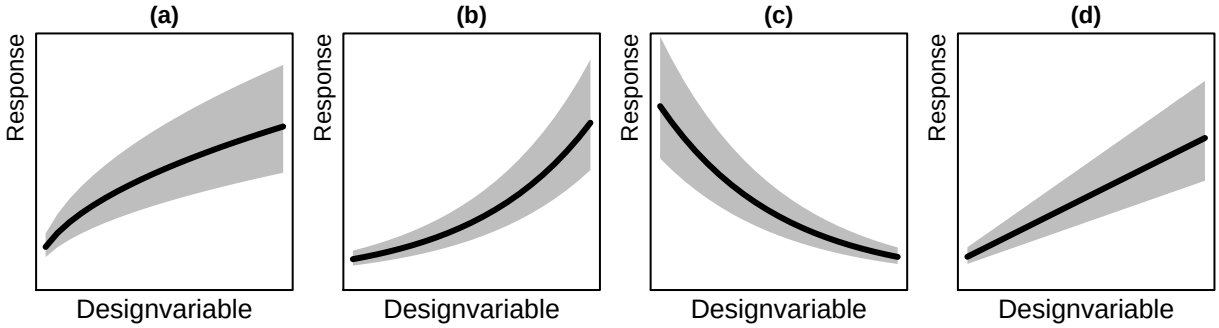
Für die jeweiligen Abweichungen von den Modellannahmen kann möglicherweise **Abhilfe** geschaffen werden, indem

- eine linearisierende Transformation auf die Designvariable(n) oder die Response angewendet wird,
- eine Varianz stabilisierende Transformation der Response vorgenommen wird,
- spezielle Covarianzmodelle für die Fehler ε_i angesetzt werden (worauf wir hier aber nicht eingehen).

10.6.2 Varianz stabilisierende Transformationen

Wenn – im Gegensatz zur Modellannahme der Homoskedastizität der ε_i – davon ausgegangen werden muss, dass die **Varianz eine Funktion der mittleren Response** (also des Wertes der Regressionsfunktion) ist, kann es nützlich sein, die Response zu transformieren. Wir beschränken uns hier auf eine *grafische* Methode zur Auswahl geeigneter Transformationen:

Es folgen vier prototypische *Regressions*plots samt Vorschlägen für **Response-Transformationen zur Varianzstabilisierung**. Wir geben die dazugehörige **S**-Formelsyntax unter Verwendung der Terminologie des Ozon-Beispiels an. Als willkommene „Nebenwirkung“ dieser Transformationen wird häufig eine Linearisierung der Regressionsbeziehung und eine „Normalisierung“ der Verteilung der Fehler beobachtet.



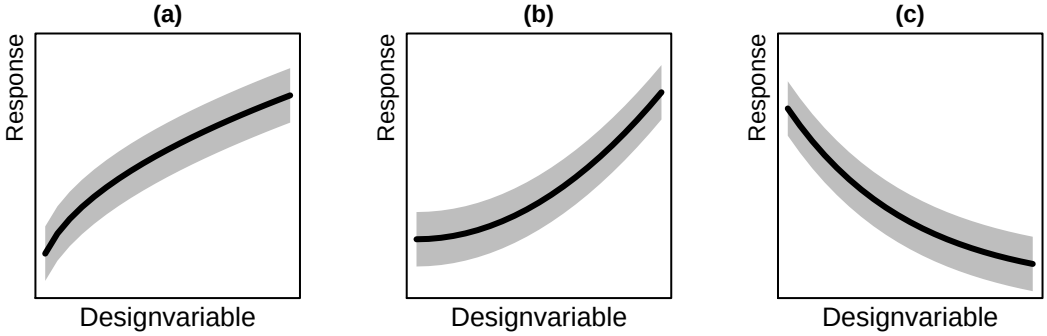
Mögliche Varianz stabilisierende Y-Transformationen:		
Transformation	**S**-Formel	Modell
$\sqrt{Y}$	`sqrt(Ozone) ~ Temp`	$\sqrt{\texttt{Ozone}_i} = \beta_0 + \beta_1\ \texttt{Temp}_i + \varepsilon_i$
$\log(Y)$	`log(Ozone) ~ Temp`	$\log(\texttt{Ozone}_i) = \beta_0 + \beta_1\ \texttt{Temp}_i + \varepsilon_i$
$1/Y$	`1/Ozone ~ Temp`	$1/\texttt{Ozone}_i = \beta_0 + \beta_1\ \texttt{Temp}_i + \varepsilon_i$

Zusätzlich zur Transformation der Response kann es durchaus hilfreich oder sogar notwendig sein, gleichzeitig eine Designtransformation (wie im nächsten Paragraphen beschrieben) durchzuführen, um den Modell-Fit zu verbessern. (Detaillierter wird hierauf z. B. in [9, Box, Hunter und Hunter (1978)], [46, Hocking (1996)] oder [66, Neter et al. (1990)] eingegangen.)

10.6.3 Linearisierende Transformationen

Wir gehen – in Übereinstimmung mit der Modellannahme der Homoskedastizität der ε_i – davon aus, dass die **Varianz konstant** ist, und geben auch hier eine *grafische* Methode zur Bestimmung der passenden linearisierenden Transformation an:

Die folgenden, prototypischen nichtlinearen Regressionsbeziehungen bei konstanter Varianz der Residuen legen die darunter angegebenen **Designtransformationen zur Linearisierung** der Beziehung zwischen Response und Design nahe. Für jede infrage kommende Transformation könnte ein eigener Modell-Fit durchgeführt werden, um die „beste“ Anpassung zu finden, wobei dazu die Verfahren aus Abschnitt 10.7 „Modifizierung eines linearen Regressionsmodells“ hilfreich sein können (und hierin speziell §10.7.3).



$\log(x), \quad \sqrt{x}$ $\qquad$ $x^2, \quad \exp(x)$ $\qquad$ $1/x, \quad \exp(-x)$

... sind mögliche Designtransformationen.

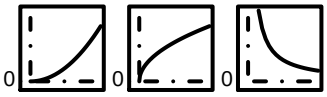
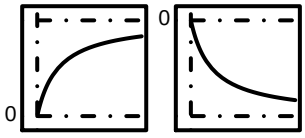
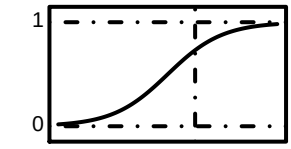
Es folgt tabelliert die **S**-Formelsyntax (wieder unter Verwendung des Ozon-Beispiels):

Linearisierende Transformationen:		
Transformation	**S**-Formel	Modell
$\log(x)$	`Ozone ~ log(Temp)`	$\texttt{Ozone}_i = \beta_0 + \beta_1 \log(\texttt{Temp}_i) + \varepsilon_i$
$\sqrt{x}$	`Ozone ~ sqrt(Temp)`	$\texttt{Ozone}_i = \beta_0 + \beta_1 \sqrt{\texttt{Temp}_i} + \varepsilon_i$
x^2	`Ozone ~ I(Temp^2)`	$\texttt{Ozone}_i = \beta_0 + \beta_1\ (\texttt{Temp}_i)^2 + \varepsilon_i$
(Polynomiale Regression wird als eigenes Thema in Abschnitt 10.10 behandelt!)		
$\exp(x)$	`Ozone ~ exp(Temp)`	$\texttt{Ozone}_i = \beta_0 + \beta_1 \exp(\texttt{Temp}_i) + \varepsilon_i$
$1/x$	`Ozone ~ I(1/Temp)`	$\texttt{Ozone}_i = \beta_0 + \beta_1\ 1/\texttt{Temp}_i + \varepsilon_i$
$\exp(-x)$	`Ozone ~ exp(-Temp)`	$\texttt{Ozone}_i = \beta_0 + \beta_1 \exp(-\texttt{Temp}_i) + \varepsilon_i$

10.6.4 Symmetrisierung des Designs und spezielle linearisierbare Regressionsfunktionen

Obige Designtransformationen können (zusätzlich zu ihrer linearisierenden Wirkung) im Fall schiefer Designverteilungen auch die Konsequenz haben, die Verteilung der Covariablen zu symmetrisieren, was deren Darstellung in Streudiagrammen verbessert und die Analyse erleichtert. Zum Beispiel können linkssteile Designverteilungen (wo sich also die Designwerte links „clustern“ und nach rechts ausdünnen) durch $\log(x)$ oder x^a mit $a < 1$ eventuell symmetrisiert werden, wohingegen dies bei rechtssteilen Verteilungen (rechts „clusternd“ und sich nach links ausdünnend) durch Potenzen x^a mit $a > 1$ geschieht.

In speziellen Anwendungen treten originär nichtlineare Regressionsbeziehungen auf, die sich jedoch durch geeignete Transfomationen in ein lineares Modell überführen lassen. Es folgt eine Auflistung solcher linearisierbarer Regressionsfunktionen samt ihrer zugehörigen Transformationen und **S**-Formelsyntax, in der `Y` jeweils der Response-Vektor und `x` die unabhängige Variable ist.

Spezielle linearisierbare Regressionsfunktionen:		
Linearisierbare Funktion	Transformation und Modell	Mögliche y-Graphen und **S**-Formel
$y = \theta_0 x^{\theta_1}$ (Cobb-Douglas-Funktion; Ökonomie)	$\log(y) = \log(\theta_0) + \theta_1 \log(x)$ $\log(y_i) = \beta_0 + \beta_1 \log(x_i) + \varepsilon_i$	 `log(Y) ~ log(x)`
$y = \dfrac{\theta_0 x}{\theta_1 + x}$ (Bioassay- oder Dosis-Response-Modell; Biologie)	$\dfrac{1}{y} = \dfrac{1}{\theta_0} + \dfrac{\theta_1}{\theta_0}\dfrac{1}{x}$ $\dfrac{1}{y_i} = \beta_0 + \beta_1 \dfrac{1}{x_i} + \varepsilon_i$	 `1/Y ~ I(1/x)`
$y = \dfrac{\exp(\theta_0 + \theta_1\ x)}{1 + \exp(\theta_0 + \theta_1\ x)}$ (Logistische Funktion)	$\log\left(\dfrac{y}{1-y}\right) = \theta_0 + \theta_1\ x$ $\log\left(\dfrac{y_i}{1-y_i}\right) = \beta_0 + \beta_1\ x_i + \varepsilon_i$	 `log(Y/(1-Y)) ~ x`

Beachte: Bei allen obigen Modellen muss die Annahme der additiven Fehler im *transformierten* Modell gerechtfertigt sein! Wenn dies nicht der Fall ist, können zwar die Parameterschätzwerte noch vernünftig sein, aber die inferenzstatistischen Verfahren der linearen Modelle (Hypothesentests, Konfidenzintervalle usw.) sind nicht mehr gültig.

Bemerkung: Ein quantitatives Verfahren zur objektiven Bestimmung einer adäquaten linearisierenden (und womöglich auch Varianz stabilisierenden) Transformation der Response Y (im Fall rein positiver Response-Werte) ist die Box-Cox-Methode. Sie wird z. B. in [46, Hocking (1996), ch. 6] unter dem Namen Box-Cox-Transformation ausführlich diskutiert und ist im **R**-Package `MASS` in der Funktion `boxcox()` implementiert. [34, Fox (2002), ch. 3.4] geht ebenfalls darauf ein.

10.7 Modifizierung eines linearen Regressionsmodells

Die Analyse eines bestehenden linearen Modells kann ergeben, dass nicht alle der Designvariablen (in der Form, in der sie im Modell auftauchen) einen signifikanten Einfluss haben. Das Entfernen eines Terms oder mehrerer Terme aus dem Modell könnte den Fit vereinfachen, ohne ihn wesentlich zu verschlechtern. Umgekehrt mag es naheliegen, weitere Terme zu einem bestehenden Modell hinzuzufügen, um den Fit zu verbessern. In beiden Fällen ist es interessant zu quantifizieren, ob das reduzierte bzw. erweiterte Modell im Vergleich zum Ausgangsmodell einen besseren Fit liefert.

Zur Modifikation eines bestehenden Modells, das in einem `lm`-Objekt gespeichert ist, stehen in **R** mehrere Funktionen zur Verfügung. Zum einen ist dies die Funktion `update()`, die im Wesentlichen Tipp- und Rechenarbeit ersparen kann, wenn ein Modell gezielt verändert werden soll. Zum anderen sind es die Funktion `drop1()`, die ausgehend von einem bestehenden Modell die Wirkung des Weglassens einzelner, im Modell befindlicher Terme bestimmt, und die Funktion `add1()`, die dasselbe für die einzelne Hinzunahme weiterer, nicht im Modell befindlicher Terme durchführt. Das Vergleichskriterium für Ausgangs- und modifiziertes Modell, das hier zum Einsatz kommt, ist "**A**n **I**nformation **C**riterion" (= AIC), welches auf [3, Akaike (1974)] zurückgeht und in §10.7.5 ausführlicher vorgestellt wird. Vorab geben wir hier die allgemeine Definition

$$\text{AIC} := -2 \cdot \{\text{maximierte log-Likelihood}\} + 2 \cdot \{\text{Anzahl der Parameter}\}, \tag{40}$$

die sich im Fall des linearen Regressionsmodells konkretisiert zu:

$$\text{AIC} = n\left(\log\left(\frac{\text{RSS}}{n}\right) + 1\right) + 2p \tag{41}$$

Aus letzterem wird deutlich, dass die Modellkomplexität in Form der Dimension p und die durch RSS quantifizierte Fit-Güte antagonistisch wirken: Ein kleinerer AIC-Wert ergibt sich nur dann, wenn bei einer Modellverkleinerung (= Dimensionsreduktion) der RSS-Wert nicht zu stark ansteigt, d. h., die Fit-Güte nicht zu sehr darunter leidet, oder wenn bei einer Modellvergrößerung (= Dimensionserhöhung) der RSS-Wert hinreichend stark zurückgeht, d. h., die Fit-Güte entsprechend zunimmt. Ziel ist es, einen möglichst niedrigen Wert für AIC zu erhalten. (Der Beitrag $2 \cdot \{\text{Anzahl der Parameter}\}$ zu AIC wird auch „Strafterm" (= "penalty term") genannt.)

Für das Verständnis der Funktionsweise von Akaikes Informationskriterium ist es hilfreich, AIC etwas genauer zu betrachten. Dazu folgt eine tabellarische Übersicht für AIC in den möglichen Szenarien „Ausgangsmodell", „reduziertes Modell" und „erweitertes Modell":

Modell	AIC
Ausgangsmodell mit Dimension p_0	$\text{AIC}_{p_0} = n(\log(\text{RSS}_0/n) + 1) + 2p_0$
Erweitertes Modell mit Dimension $p_+ > p_0$	$\text{AIC}_{p_+} = n(\log(\underbrace{\text{RSS}_+}_{\leq \text{RSS}_0}/n) + 1) + 2\underbrace{p_+}_{>p_0}$
Reduziertes Modell mit Dimension $p_- < p_0$	$\text{AIC}_{p_-} = n(\log(\underbrace{\text{RSS}_-}_{\geq \text{RSS}_0}/n) + 1) + 2\underbrace{p_-}_{<p_0}$

Die Berechnung der AIC-Statistik für Modelle, in denen sich p_- bzw. p_+ um genau 1 von p_0 unterscheiden, ist in den Funktionen `drop1()` bzw. `add1()` implementiert. Deren Verwendung wird in den beiden übernächsten Abschnitten erläutert. Zunächst jedoch zur Funktion `update()`, die hauptsächlich eine organisatorische Hilfe ist.

10.7.1 Die Funktion `update()`

Die Funktion `update()` erlaubt, ein neues Modell ausgehend von einem bestehenden zu kreieren, indem lediglich diejenigen Argumente angegeben werden, die zu ändern sind. Ihre zwei wesentlichen Argumente heißen `object` und `formula`, wobei `object` das Modellobjekt, wie z. B. ein `lm`-Objekt erwartet, welches modifiziert werden soll. Das Argument `formula` erwartet die modifizierte Modellformel, wobei ein Punkt (`.`) auf der linken oder rechten Seite der Tilde `~` durch die linke oder rechte Seite der ursprünglichen Modellformel aus `object` ersetzt wird. (Ein alleiniger Punkt links der Tilde kann weggelassen werden.)

Das in Abschnitt 10.3 erstellte Modell `oz2.lm` hätte zum Beispiel aus einem (auf Basis des um NAs bereinigten (!) Data Frames `air` generierten) `lm`-Objekt `oz1.lm` wie folgt erzeugt werden können:

```
> oz1.lm <- lm( Ozone ~ Temp, data = air)
> oz2.lm <- update( object = oz1.lm, formula = . ~ . + Solar.R)
```

Oder noch kürzer, indem die Argumentebenennungen und der alleinige Punkt links der Tilde weggelassen werden:

```
> oz2.lm <- update( oz1.lm, ~ . + Solar.R)
```

Eine (hier logarithmische) Transformation der Responsevariablen bei gleichzeitigem Ausschluss des konstanten Terms β_0 aus dem Modell erreicht man z. B. durch:

```
> oz2b.lm <- update( oz1.lm, log(.) ~ . - 1)
```

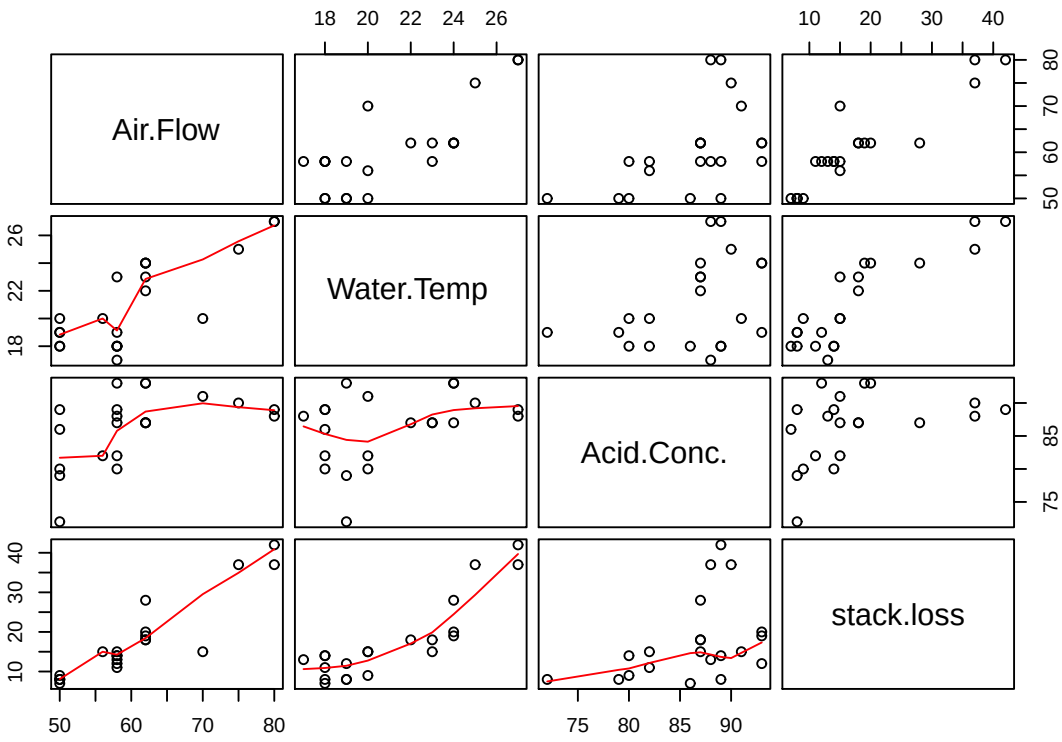
In den folgenden Abschnitten werden uns noch weitere Anwendungsbeispiele der Funktion `update()` begegnen.

10.7.2 Das Entfernen eines Terms: `drop1()`

Zur Illustration der Funktionsweise von `drop1()` verwenden wir den eingebauten Data Frame `stackloss`, der Informationen für einen gewissen Fabrikationsprozess von Nitrit (NHO_3) aus Ammoniak (NH_3) enthält. Seine Komponente `stack.loss` ist eine Maßzahl für die *nicht* in einem „Gegenstrom-Absorptionsturm“ aufgefangene Menge an Nitrit und hier die Responsevariable. Die übrigen drei Komponenten sind die Covariablen und beschreiben die jeweiligen Umgebungsbedingungen des Prozesses (siehe auch **R**sOnline-Hilfe dazu):

```
> stackloss
   Air.Flow Water.Temp Acid.Conc. stack.loss
1        80         27         89         42
2        80         27         88         37
3        75         25         90         37
....
21       70         20         91         15
```

Ein Aufruf der Art `pairs( stackloss, lower.panel = panel.smooth)` liefert die folgenden paarweisen Streudiagramme und demnach scheinen alle drei der Variablen `Air.Flow`, `Water.Temp` und `Acid.Conc.` einen Einfluss auf den `stack.loss` zu haben.



Wir fitten das Modell

$$\texttt{stack.loss}_i = \beta_0 + \beta_1\ \texttt{Air.Flow}_i + \beta_2\ \texttt{Water.Temp}_i + \beta_3\ \texttt{Acid.Conc.}_i + \varepsilon_i$$

folgendermaßen:

```
> summary( stack.lm <- lm( stack.loss ~ Air.Flow + Water.Temp +
+ Acid.Conc., data = stackloss))
Call: lm(formula = stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,
         data = stackloss)

Residuals:
    Min      1Q  Median      3Q     Max
-7.2377 -1.7117 -0.4551  2.3614  5.6978

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -39.9197    11.8960  -3.356  0.00375 **
Air.Flow      0.7156     0.1349   5.307  5.8e-05 ***
Water.Temp    1.2953     0.3680   3.520  0.00263 **
Acid.Conc.   -0.1521     0.1563  -0.973  0.34405
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.243 on 17 degrees of freedom
Multiple R-Squared: 0.9136,     Adjusted R-squared: 0.8983
F-statistic:  59.9 on 3 and 17 DF,  p-value: 3.016e-09
```

Obige Summary und die (nicht gezeigten) Diagnoseplots für `stack.lm` deuten eine gute Anpassung an, doch ist der Beitrag der Variable `Acid.Conc.` nicht signifikant (da der p-Wert des Tests $H_0 : \beta_3 = 0$ gleich 0.344 ist). Mittels der Funktion `drop1()` soll die Wirkung des Entfernens einer Variable aus `stack.lm` untersucht werden:

Entfernen von Modelltermen mittels `drop1()`:

```
> drop1( stack.lm)
Single term deletions

Model:
stack.loss ~ Air.Flow +
       Water.Temp + Acid.Conc.
           Df Sum of Sq    RSS
<none>                  178.83
Air.Flow    1    296.23 475.06
Water.Temp  1    130.31 309.14
Acid.Conc.  1      9.97 188.80

                  AIC
                52.98
                71.50
                62.47
                52.12
```

Ausgehend von dem gefitteten Modell `stack.lm` bestimmt `drop1()` für jeden einzelnen der Modellterme den Effekt seines Entfernens.
Resultat: Das Ausgangsmodell (`Model`), eine (Art) ANOVA-Tabelle, in deren Zeile `<none>` die Residuenquadratesumme (`RSS`) und die AIC-Statistik (`AIC`) des Ausgangsmodells angegeben werden. Die weiteren Zeilen enthalten für den jeweils genannten Term seine zugehörigen Freiheitsgrade (`Df`), seinen Anteil (`Sum of Sq`) an der Residuenquadratesumme (`RSS`) des um ihn reduzierten Modells sowie den zum reduzierten Modell gehörenden Wert der AIC-Statistik (`AIC`). (Es gilt also: „RSS(ohne Term) = RSS(`<none>`) + `Sum of Sq`(Term)“.)
Das (nicht gezeigte) Argument `test = "F"` würde die Ausgabe um marginale F-Tests für jeden der Modellterme ergänzen.

Das Entscheidungskriterium für eine mögliche Verbesserung des Modells durch Entfernung eines Terms lautet wie folgt: Tritt unter den AIC-Werten der reduzierten Modelle ein Wert auf, der kleiner als derjenige des Ausgangsmodells (in Zeile `<none>`) ist, so ist der Term mit dem kleinsten dieser `AIC`-Werte zu entfernen. (Ist der `AIC`-Wert des Ausgangsmodells der kleinste, so kann das Modell durch Entfernen eines Terms nicht verbessert werden.)

In obigem Beispiel bietet sich demzufolge `Acid.Conc.` zur Entfernung an, was mittels `update()` umgesetzt wird:

```
> summary( stack2.lm <- update( stack.lm, ~ . - Acid.Conc.))
Call: lm(formula = stack.loss ~ Air.Flow + Water.Temp, data = stackloss)

Residuals:
    Min      1Q  Median      3Q     Max
-7.5290 -1.7505  0.1894  2.1156  5.6588

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -50.3588     5.1383  -9.801 1.22e-08 ***
Air.Flow      0.6712     0.1267   5.298 4.90e-05 ***
Water.Temp    1.2954     0.3675   3.525  0.00242 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.239 on 18 degrees of freedom
Multiple R-Squared: 0.9088,     Adjusted R-squared: 0.8986
F-statistic: 89.64 on 2 and 18 DF,  p-value: 4.382e-10
```

Offenbar haben wir mit dem reduzierten Modell praktisch dieselbe Qualität des Fits erreicht wie mit dem komplizierteren Modell: Alle Designvariablen liefern einen signifikanten Beitrag und die Residuenstandardabweichung (`Residual standard error`) ist sogar leicht von 3.243 auf 3.239 gefallen, während der (multiple) R^2-Wert (`R-squared`) nur leicht von 0.9136 auf 0.9088 gefallen ist. Der adjustierte R^2-Wert R_a^2 (`Adjusted R-squared`) ist sogar – ganz leicht – von 0.8983 auf 0.8986 gestiegen. Auch die (nicht gezeigten) Diagnoseplots unterstützen das einfachere Modell.

10.7.3 Das Hinzufügen eines Terms: `add1()`

Hier betrachten wir erneut das Ozon-Problem aus Abschnitt 10.2. Dort hatten wir eine einfache lineare Regression von `Ozone` an `Temp` durchgeführt und das resultierende `lm`-Objekt mit `oz1.lm` bezeichnet. Der Data Frame `air` enthält aber noch die weiteren möglichen Designvariablen `Solar.R` und `Wind`, deren (hier nicht gezeigten) paarweisen Streudiagramme mit `Ozone` eine annähernd lineare Beziehung zu `Ozone` zeigen.

Wir wollen für jede der beiden Variablen den Effekt ihres Hinzufügens zum o. g. Ausgangsmodell bestimmen. Dies wird durch die Funktion `add1()` ermöglicht. Sie benötigt als erstes Argument das `lm`-Objekt mit dem gefitteten Ausgangsmodell, also `oz1.lm`. Das zweite Argument ist eine Formel, die den Umfang der möglichen Modellergänzungen spezifiziert. Die Formel braucht *keine* „linke Seite“, d. h. Response; diese wird automatisch aus dem Ausgangsmodell entnommen. Eine Formel der Art `~ . + Wind + Solar.R` gibt beispielsweise an, dass abwechselnd eine der Variablen `Wind` und `Solar.R` zum Modell hinzugefügt werden soll (wie in folgender Tabelle).

Hinzufügen von Modelltermen mittels `add1()`:

```
> add1( oz1.lm, ~ . +
+ Wind + Solar.R)
Single term additions

Model: Ozone ~ Temp
        Df Sum of Sq
<none>
Wind     1     11378
Solar.R  1      2723

           RSS    AIC
         62367    707
         50989    686
         59644    704
```

Für jeden der in der Formel `~ . + Wind + Solar.R` auftretenden Term wird der Effekt seines einzelnen Hinzufügens zu dem in `oz1.lm` bereits bestehenden und durch `.` symbolisierten Modell bestimmt.
Resultat: Das Ausgangsmodell (`Model`), eine (Art) ANOVA-Tabelle, in deren Zeile `<none>` die Residuenquadratsumme (`RSS`) und die AIC-Statistik (`AIC`) des Ausgangsmodells angegeben werden. Die weiteren Zeilen enthalten für den jeweils genannten Term die zu ihm gehörigen Freiheitsgrade (`df`), seine Reduktion (`Sum of Sq`) der Residuenquadratsumme (`RSS`) des Ausgangsmodells sowie den zum erweiterten Modell gehörenden Wert der AIC-Statistik. (Es gilt also: „RSS(mit Term) = RSS(`<none>`) − `Sum of Sq`(Term)“.)
Das (nicht gezeigte) Argument `test = "F"` würde die Ausgabe um marginale F-Tests für jeden neuen Term im *erweiterten* Modell ergänzen.

Das Entscheidungskriterium für eine mögliche Verbesserung des Modells durch Hinzufügen eines Terms ist analog zu dem zur Entfernung eines Terms: Tritt unter den AIC-Werten der erweiterten Modelle ein Wert auf, der kleiner als derjenige des Ausgangsmodells (in Zeile `<none>`) ist, so ist der Term mit dem kleinsten dieser `AIC`-Werte hinzuzufügen. (Ist der `AIC`-Wert des Ausgangsmodells der kleinste, so kann das Modell durch Hinzufügen eines Terms nicht verbessert werden.)

Im vorliegenden Beispiel könnte es sinnvoll sein, die Variable `Wind` hinzuzufügen (was wir hier aber nicht durchführen).

Zurückblickend erscheint die Wahl von `Temp` als Ausgangsvariable im einfachen linearen Modell als etwas willkürlich. Eine (rein statistisch) eventuell besser begründbare Vorgehensweise ist die folgende: Ausgehend von dem auch Null-Modell genannten Modell, das nur β_0 enthält (also nur den sogenannten “intercept term”), erlaubt uns die Funktion `add1()` mittels des AIC-Kriteriums die begründbare Auswahl einer Startvariablen:

```
> (oz0.lm <- lm( Ozone ~ 1, data = air))
Call: lm(formula = Ozone ~ 1, data = air)
```

```
Coefficients:
(Intercept)
       42.1

> add1( oz0.lm, ~ . + Temp + Wind + Solar.R)
Single term additions

Model: Ozone ~ 1
        Df Sum of Sq    RSS    AIC
<none>               121802    779
Temp     1     59434  62367    707
Wind     1     45694  76108    729
Solar.R  1     14780 107022    767
```

Und siehe da: `Temp` wäre auch auf diese Weise zur ersten Wahl geworden.

Eine typische Anwendung für `add1()` ist die Auswahl geeigneter Transformationen für Designvariablen, wie sie in §10.6.3 diskutiert wurden. Angenommen, für eine Covariable, die sich in ihrer „Reinform“ `x` bereits im Modell `some.model` befindet, werden mehrere Transformationen als Ersatz für `x` in Erwägung gezogen. Man kann sie alle mit Hilfe von `add1()` „durchprobieren“ lassen und die nach der AIC-Statistik (vgl. §10.7.5) am besten geeignete auswählen. Dazu wird `x` zunächst aus dem Modell `some.model` eliminiert und im reduzierten Modell mit den fraglichen Transformationen verglichen. Beispiel:

```
> some.model.minusx <- update( some.model, ~ . - x)
> add1( some.model.minusx, ~ . + x + I( x^2) + exp( x) + log( x) + I(1/x))
....
```

Beachte: Grundsätzlich sollten bei der Modellierung *fachspezifische* Überlegungen die entscheidende Rolle bei Auswahl und Transformation von Regressionsvariablen spielen!

Ferner sollte man sich vor halb- oder vollautomatischen Verfahren der Regressorselektion, wie z. B. “stepwise regression” (mit “forward” und “backward elimination”) oder sogenannter exhaustiver (= „erschöpfender“ = vollständiger) Suche hüten, denn bei ihrer Verwendung wird häufig das implizit ablaufende multiple Testen ignoriert, sodass die Validität des resultierenden Modells äußerst fragwürdig ist, wenn nicht Maßnahmen zur Kompensation der α-Fehler-Inflation ergriffen worden sind.

10.7.4 Beschränkung auf eine Teilmenge der Daten: das `lm`-Argument `subset`

Die Funktion `lm()` stellt ein optionales Argument `subset` zur Verfügung, über das auf verschiedene Arten (z. B. über einen `integer`-Indexvektor oder einen logischen Indexvektor als Ergebnis eines Ausdrucks) spezifiziert werden kann, welche Zeilen des Data Frames im Modell für dessen Anpassung verwendet werden sollen. Damit kann in manchen Fällen die Notwendigkeit entfallen, den Data Frame im Vorfeld „auszudünnen“. Details liefert die Online-Hilfe zu `lm()` bzw. zu `model.frame()`.

10.7.5 Exkurs: Akaikes Informationskriterium AIC

In diesem Exkurs wollen wir auf die Theorie eingehen, die AIC zugrunde liegt (ohne Bezug zu **R**).

Angenommen, der beobachtbare, n-dimensionale, zufällige Responsevektor $\mathbf{Y}$ habe die unbekannte, aber feste Verteilungsfunktion (VF) F in der Menge $\mathcal{M}$ aller n-dimensionalen VFn, von denen jede eindeutig identifizierbar (= wohldefiniert) sei. Gelegentlich lässt sich $\mathcal{M}$ aufgrund von Zusatzinformationen einschränken, z. B. auf die Menge der absolut-stetigen VFn oder weiter auf die Menge aller n-dimensionalen Normalverteilungen oder sogar auf solche mit einer gewissen Kovarianzstruktur. $\mathcal{M}$ wird auch "operating family" genannt (siehe z. B. Linhart und Zucchini (1986), an denen wir uns in den folgenden drei Paragraphen stark orientieren).

Zur Schätzung des unbekannten F's auf Basis von (endlich vielen!) Daten geht man häufig zu einer parametrischen Familie von VFn über, um F zu approximieren. In dieser „approximierende Modellfamilie" braucht F nicht enthalten zu sein; aus ihr wird lediglich das Modell (= die VF) gewählt, das an die „aus F stammenden" Daten angepasst werden soll. Es stellt sich die Frage, welche approximierende Modellfamilie zu wählen ist.

Eine Möglichkeit ist, die Familie zu wählen, von der man „schätzt", dass sie unter den gegebenen Umständen (wie Zusatzinformationen, Stichprobenumfang, Verwendungszweck des Modells etc.) die „am besten passende" ist. Dazu ist insbesondere zu spezifizieren, wie der Anpassungsfehler – die „Diskrepanz" – zwischen dem angepassten Modell und dem tatsächlich operierenden quantifiziert wird. Letztlich wäre dann diejenige approximierende Familie zu wählen, mit welcher die erwartete Diskrepanz minimiert wird, wobei jedoch nicht unbedingt davon ausgegangen werden kann, dass das operierende Modell überhaupt in der approximierende Familie liegt. Um diese Vorgehensweise umzusetzen, muss die erwartete Diskrepanz jedoch geschätzt werden, da sie von dem unbekannten operierenden Modell F abhängt.

10.7.5.1 Die Diskrepanz

Zunächst sind ein paar Formalisierungen zum Konzept der Diskrepanz notwendig (die im Bild auf der folgenden Seite grafisch etwas veranschaulicht werden):

- $\mathcal{M}$ sei die Menge (oder eine Teilmenge) aller n-dimensionalen VFn.
- Eine <u>Diskrepanz</u> $\Delta : \mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R}$ ist ein Funktional mit der Eigenschaft

  $$\Delta(G, H) \geq \Delta(H, H) \quad \text{für alle } G, H \in \mathcal{M}$$

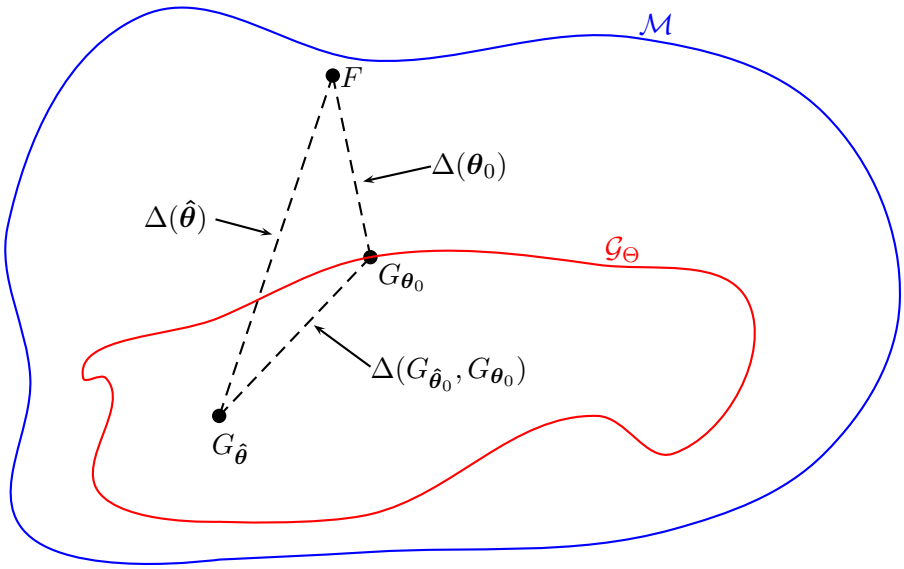
  $\Delta(G, H)$ sollte des Weiteren mit zunehmender „Unterschiedlichkeit" zwischen den zwei Modellen G und H wachsen. (Beachte: Δ braucht *keine* Metrik (= Abstand) zu sein; es wird z. B. weder Symmetrie noch die Dreiecksungleichung gefordert!)
- F ($\in \mathcal{M}$) sei das unbekannte <u>operierende Modell</u> und $\mathbf{Y}_1, \ldots, \mathbf{Y}_N$ iid $\sim F$.
- Eine <u>(approximierende)</u> <u>Modellfamilie</u> $\mathcal{G}_\Theta = \{G_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta\}$ ist eine Teilmenge von $\mathcal{M}$, deren jedes Element $G_{\boldsymbol{\theta}}$ durch seinen Parametervektor $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)'$ eineindeutig definiert ist.
- Ein <u>gefittetes Modell</u> $G_{\hat{\boldsymbol{\theta}}}$ ist ein Element von $\mathcal{G}_\Theta$, dessen Parametervektor $\hat{\boldsymbol{\theta}}$ aus den beobachtbaren Daten $\mathbf{Y}_1, \ldots, \mathbf{Y}_N$ geschätzt wird.
- Für ein approximierendes Modell $G_{\boldsymbol{\theta}}$ und das operierende Modell F sei ihre Diskrepanz abgekürzt

  $$\Delta(\boldsymbol{\theta}) := \Delta(G_{\boldsymbol{\theta}}, F)$$

- Die Approximationsdiskrepanz zwischen einer Modellfamilie $\mathcal{G}_\Theta$ und einem operierenden Modell F ist
$$\Delta(\boldsymbol{\theta}_0), \text{ wobei } \boldsymbol{\theta}_0 := \arg\inf\{\Delta(\boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}$$
und angenommen werde, dass genau ein $\boldsymbol{\theta}_0$ existiert. $G_{\boldsymbol{\theta}_0}$ heißt bestes approximierendes Modell der Familie $\mathcal{G}_\Theta$ zur Diskrepanz Δ.

- Die Schätzdiskrepanz ist die zufällige (!) Diskrepanz zwischen dem besten approximierenden und einem gefitteten Modell. Sie quantifiziert den Anpassungsfehler, der auf der Zufälligkeit der Daten beruht:
$$\Delta(G_{\hat{\boldsymbol{\theta}}}, G_{\boldsymbol{\theta}_0})$$

- Die Gesamtdiskrepanz ist die zufällige (!) Diskrepanz zwischen einem gefitteten Modell und dem operierenden Modell F:
$$\Delta(\hat{\boldsymbol{\theta}}) = \Delta(G_{\hat{\boldsymbol{\theta}}}, F)$$
Sie braucht *nicht* die Summe von Approximations- und Schätzdiskrepanz zu sein, die sich übrigens stets antagonistisch zueinander verhalten.



- Die unbekannte (!) erwartete (Gesamt-)Diskrepanz $\mathbb{E}_F[\Delta(\hat{\boldsymbol{\theta}})]$ wird durch das Paar ($\mathcal{G}_\Theta$, Schätzverfahren) festgelegt, welches Fitting-Prozedur genannt wird und wobei der Index F beim Erwartungswert bedeutet, dass dieser unter dem Modell F berechnet wird (d. h. unter der Annahme $\mathbf{Y}_j$ iid $\sim F$).

  Die Fitting-Prozedur besteht aus einer approximierenden Familie $\mathcal{G}_\Theta$, welche $\boldsymbol{\theta}_0$ und $\Delta(\boldsymbol{\theta}_0)$ festlegt, und einem Schätzverfahren zur Bestimmung von $\hat{\boldsymbol{\theta}}$ als Schätzer für $\boldsymbol{\theta}_0$. Typischerweise ist es das Ziel, *bei festgelegtem Schätzverfahren unter konkurrierenden approximierenden Familien diejenige zu wählen, in der die erwartete Diskrepanz minimiert wird.*

- Einen Schätzer für die erwartete Diskrepanz nennt man Kriterium.

- $\Delta(\boldsymbol{\theta}) \equiv \Delta(G_{\boldsymbol{\theta}}, F)$ ist offenbar für jedes $\boldsymbol{\theta}$ unbekannt (da F unbekannt ist). Einen konsistenten Schätzer $\Delta_N(\boldsymbol{\theta})$ für $\Delta(\boldsymbol{\theta})$ nennt man empirische Diskrepanz. Existiert
$$\hat{\boldsymbol{\theta}}_N := \arg\inf\{\Delta_N(\boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}$$
fast-sicher, so heißt $\hat{\boldsymbol{\theta}}_N$ der Minimum-Diskrepanz-Schätzer.

  Eine geeignete empirische Diskrepanz erhält man z. B., indem F in $\Delta(\boldsymbol{\theta}) \equiv \Delta(G_{\boldsymbol{\theta}}, F)$ durch die empirische VF F_N der $\mathbf{Y}_1, \ldots, \mathbf{Y}_N$ ersetzt wird: $\Delta_N(\boldsymbol{\theta}) := \Delta(G_{\boldsymbol{\theta}}, F_N)$.

Unter gewissen Regularitätsbedingungen (an Θ, an die Glattheit von Δ und Δ_N sowie an gewisse Konvergenzen von Δ_N, Δ'_N und Δ''_N; vgl. [55, Linhart und Zucchini (1986), S. 238]) kann man zeigen, dass ein Minimum-Diskrepanz-Schätzer $\hat{\boldsymbol{\theta}}_N$ fast-sicher und in Verteilung gegen $\boldsymbol{\theta}_0$ konvergiert, dass $\Delta_N(\hat{\boldsymbol{\theta}}_N) \to \Delta(\boldsymbol{\theta}_0)$ fast-sicher und dass für „große" N in Konsequenz die folgenden Approximationen gelten [55, Linhart und Zucchini (1986), S. 240-1]):

$$\mathbb{E}_F[\Delta(\hat{\boldsymbol{\theta}}_N)] \approx \mathbb{E}_F[\Delta_N(\hat{\boldsymbol{\theta}}_N)] + \mathrm{Spur}(\mathcal{H}^{-1}\boldsymbol{\Sigma})/N, \tag{42}$$

$$\mathrm{Var}_F(\Delta(\hat{\boldsymbol{\theta}}_N)) \approx \mathrm{Spur}(\mathcal{H}^{-1}\boldsymbol{\Sigma}\mathcal{H}^{-1}\boldsymbol{\Sigma})/(2N^2), \tag{43}$$

wobei

$$\mathcal{H} = \left(\frac{\partial^2\Delta}{\partial\theta_i\partial\theta_j}(\boldsymbol{\theta}_0)\right)_{1\le i,j\le p} \qquad \text{die } (p\times p)\text{-Hesse-Matrix von } \Delta(\boldsymbol{\theta}_0)$$

ist und $\boldsymbol{\Sigma}$ die asymptotische (für $N \to \infty$) Covarianzmatrix von $\sqrt{N}\ \mathrm{grad}(\Delta_N(\boldsymbol{\theta}_0))$.

Falls $F = G_{\boldsymbol{\theta}_0}$ ist (also die approximierende Familie $\mathcal{G}_\Theta$ das operierende Modell F enthält), hängt für gewisse Diskrepanzen die Spur von $\mathcal{H}^{-1}\Sigma$ nur noch von $p = \dim(\boldsymbol{\theta})$ ab und lässt sich explizit ausrechnen (sodass die im Allgemeinen unbekannten $\mathcal{H}$ und $\boldsymbol{\Sigma}$ *nicht* geschätzt zu werden brauchen). Es stehen damit Schätzer für die erwartete Diskrepanz $\mathbb{E}_F[\Delta(\hat{\boldsymbol{\theta}}_N)]$ und für $\mathrm{Var}_F(\Delta(\hat{\boldsymbol{\theta}}_N))$ zur Verfügung und also auch ein Kriterium. Speziell für die Kullback-Leibler-Diskrepanz (siehe nächster Abschnitt) ist unter $F = G_{\boldsymbol{\theta}_0}$ sogar $\mathcal{H} = \boldsymbol{\Sigma}$ und folglich $\mathrm{Spur}(\mathcal{H}^{-1}\boldsymbol{\Sigma}) = \mathrm{Spur}(\mathcal{H}^{-1}\boldsymbol{\Sigma}\mathcal{H}^{-1}\boldsymbol{\Sigma}) = p$, sodass wir aus (42) und (43)

$$\Delta_N(\hat{\boldsymbol{\theta}}_N) + p/N \quad \text{als Kriterium (d. h. Schätzer für } \mathbb{E}_F[\Delta(\hat{\boldsymbol{\theta}}_N)]) \text{ und} \tag{44}$$

$$p/(2N^2) \quad \text{als Schätzer für } \mathrm{Var}_F(\Delta(\hat{\boldsymbol{\theta}}_N)) \tag{45}$$

erhalten (indem $\mathbb{E}_F[\Delta_N(\hat{\boldsymbol{\theta}}_N)]$ in (42) durch $\Delta_N(\hat{\boldsymbol{\theta}}_N)$ „geschätzt" wird) .

Bemerkungen: Selbst für den Fall, dass das operierende Modell F zwar *nicht* in $\mathcal{G}_\Theta$ enthalten, aber auch nicht „allzu verschieden" von $G_{\boldsymbol{\theta}_0}$ ist, sind die Schätzer in (44) und (45) verwendbar. Aus (44) ergibt sich (im folgenden Abschnitt) schließlich Akaikes Informationskriterium AIC.

10.7.5.2 Die Kullback-Leibler-Diskrepanz und AIC

Die Kullback-Leibler-Diskrepanz wird von der analog benannten, sehr allgemein definierten Kullback-Leibler-Information abgeleitet: P und Q seien zwei Verteilungen und Y eine (beliebige) Zufallsvariable. Ferner sei

$$\mathrm{KL}(P,Q) := \begin{cases} \mathbb{E}_P\left[\log\left(\frac{dP}{dQ}(Y)\right)\right], & \text{falls } P << Q; \\ \infty & \text{sonst.} \end{cases} \tag{46}$$

Hierbei bedeutet $\mathbb{E}_P$, dass der Erwartungswert unter $Y \sim P$ berechnet wird.
Dann ist $\mathrm{KL}(P,Q)$ wohldefiniert und es gilt:

$$0 \le \mathrm{KL}(P,Q) \le \infty \qquad \text{und} \qquad \mathrm{KL}(P,Q) = 0 \Longleftrightarrow P = Q \tag{47}$$

Nun sei $\mathbf{Y}$ ein n-dimensionaler Zufallsvektor, für P und Q mögen n-dimensionale Lebesgue-Dichten $f = F'$ bzw. $g = G'$ existieren und es sei $\mathrm{KL}(F,G) := \mathrm{KL}(P,Q)$. Dann gilt (im Fall $f << g$):

$$0 \le \mathrm{KL}(F,G) = \mathbb{E}_F\left[\log\left(\frac{f(\mathbf{Y})}{g(\mathbf{Y})}\right)\right] = \int_{\mathbb{R}^n} \log\left(\frac{f(\mathbf{y})}{g(\mathbf{y})}\right) f(\mathbf{y})\ d\mathbf{y} \tag{48}$$

$$= \int_{\mathbb{R}^n} \log(f(\mathbf{y}))\ f(\mathbf{y})\ d\mathbf{y} \underbrace{- \int_{\mathbb{R}^n} \log(g(\mathbf{y}))\ f(\mathbf{y})\ d\mathbf{y}}_{=:\ \Delta_{KL}(G,F)} \tag{49}$$

Folglich ist $\Delta_{KL}(G,F) \geq \Delta_{KL}(F,F)$ mit „=“ $\iff g = f$ fast-überall, sodass es sich bei Δ_{KL} in der Tat um eine Diskrepanz handelt. Sie wird Kullback-Leibler-Diskrepanz genannt und offenbar ist

$$\Delta_{KL}(G,F) = -\mathbb{E}_F\left[\log(g(\mathbf{Y}))\right] \tag{50}$$

Die dazugehörige empirische Diskrepanz erhält man unter Verwendung der empirischen VF F_N (basierend auf $\mathbf{Y}_1,\ldots,\mathbf{Y}_N$ iid $\sim F$):

$$\Delta_{KL,N}(G,F) := -\frac{1}{N}\sum_{j=1}^{N}\log\left(g(\mathbf{Y}_j)\right) \tag{51}$$

Für die Dichte $g_{\boldsymbol{\theta}} = G'_{\boldsymbol{\theta}}$ einer VF aus einer approximierenden Familie $\mathcal{G}_\Theta$ lauten Diskrepanz und empirische Diskrepanz

$$\Delta_{KL}(\boldsymbol{\theta}) = -\mathbb{E}_F\left[\log(g_{\boldsymbol{\theta}}(\mathbf{Y}))\right] \qquad \text{bzw.} \qquad \Delta_{KL,N}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{j=1}^{N}\log\left(g_{\boldsymbol{\theta}}(\mathbf{Y}_j)\right) \tag{52}$$

Es sei $\hat{\boldsymbol{\theta}}_N$ der Minimum-Diskrepanz-Schätzer und da im Fall der Kullback-Leibler-Diskrepanz unter $F = G_{\boldsymbol{\theta}_0}$ die Identität $\mathcal{H} = \boldsymbol{\Sigma}$ gilt (vgl. [55, Linhart und Zucchini (1986), S. 245]), folgt $\mathrm{Spur}(\mathcal{H}^{-1}\boldsymbol{\Sigma}) = p$, und wir erhalten aus (52) als Kriterium (gemäß (44))

$$-\frac{1}{N}\sum_{j=1}^{N}\log\left(g_{\hat{\boldsymbol{\theta}}_N}(\mathbf{Y}_j)\right) + \frac{p}{N} \tag{53}$$

Die empirische Diskrepanz in (52) ist offenbar proportional zur log-Likelihood-Funktion $\log\left(\mathcal{L}(\boldsymbol{\theta};\mathbf{Y}_1,\ldots,\mathbf{Y}_n)\right) := \log\left(\prod_{j=1}^{N} g_{\boldsymbol{\theta}}(\mathbf{Y}_j)\right)$, sodass in diesem Szenario der Minimum-Diskrepanz-Schätzer $\hat{\boldsymbol{\theta}}_N$ gleich dem Maximum-Likelihood-Schätzer ist. Das Kriterium in (53) enthält demnach offenkundig als zentralen Bestandteil die *maximierte* log-Likelihood $\log(\mathcal{L}(\hat{\boldsymbol{\theta}}_N;\mathbf{Y}_1,\ldots,\mathbf{Y}_n))$. Aus gewissen Normierungsgründen wird das Kriterium mit $2N$ multipliziert – was bei der Suche nach der minimierenden approximierenden Familie jedoch keine Rolle spielt – und so schließlich Akaikes Informationskriterium definiert:

$$\mathrm{AIC} := -2\cdot\{\text{maximierte log-Likelihood}\} + 2\cdot\{\text{Anzahl der Parameter}\} \tag{54}$$

Bemerkungen: Da die log-Likelihood nur bis auf einen zwar datenabhängigen, aber nicht modellrelevanten, konstanten Summanden definiert ist, gilt dasselbe auch für AIC.
Oft (wie z. B. im folgenden Fall der linearen Regression) liegt nur *ein* beobachtbarer Responsevektor $\mathbf{Y}$ der Dimension n vor, also ist in (53) $N = 1$.

10.7.5.3 AIC im Modell der linearen Regression

Memo: Die Dichte der multivariaten Normalverteilung $\mathcal{N}_n(\boldsymbol{\mu},\boldsymbol{\Sigma})$ lautet

$$f_{\boldsymbol{\mu},\boldsymbol{\Sigma}}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\} \quad \text{für } \mathbf{x}\in\mathbb{R}^n \tag{55}$$

Angenommen, das operierende Modell ist F mit $Y_i = \mu_i + u_i$ für $i = 1,\ldots,n$, wobei die u_i unkorreliert und zentriert sind und die Varianz σ_i^2 haben; in vektorieller Notation:

$$\mathbf{Y} = \boldsymbol{\mu} + \mathbf{U} \sim F \qquad \text{mit } \mathbb{E}[\mathbf{U}] = \mathbf{0} \text{ und } \mathrm{Cov}(\mathbf{U}) = \mathrm{diag}(\sigma_1^2,\ldots,\sigma_n^2) \tag{56}$$

Die Annahmen des linearen Modells (siehe Seite 197) bedeuten, dass als approximierende Modelle solche der Form

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \qquad \text{mit } \boldsymbol{\varepsilon} \sim \mathcal{N}_n(\mathbf{0},\sigma^2\mathbf{I}_{n\times n}) \tag{57}$$

infrage kommen, mit bekanntem $\mathbf{X}$ und unbekanntem $(\boldsymbol{\beta}, \sigma^2) \in \Theta \subset \mathbb{R}^{p+1}$ (mit $\sigma^2 > 0$). Damit ist das approximierende Modell die VF $G_{\boldsymbol{\beta},\sigma^2}$ der $\mathcal{N}_n(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I}_{n\times n})$-Verteilung mit der Dichte

$$g_{\boldsymbol{\beta},\sigma^2}(\mathbf{z}) = (2\pi\sigma^2)^{-n/2} \exp\left\{-\|\mathbf{z} - \mathbf{X}\boldsymbol{\beta}\|^2/(2\sigma^2)\right\}, \qquad \mathbf{z} \in \mathbb{R}^n \tag{58}$$

Die Likelihood-Funktion für $\boldsymbol{\beta}$ und σ^2 und die log-Likelihood-Funktion (unter Ignoranz konstanter Terme) lauten dann demgemäß

$$\begin{aligned} \mathcal{L}(\boldsymbol{\beta}, \sigma^2; \mathbf{Y}) &= \left(2\pi\sigma^2\right)^{-n/2} \exp\left\{-\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2/(2\sigma^2)\right\} \qquad \text{bzw.} && (59) \\ l\left(\boldsymbol{\beta}, \sigma^2\right) &:= -\frac{n}{2}\log(\sigma^2) - \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2/(2\sigma^2) && (60) \end{aligned}$$

Die Maximierung der log-Likelihood in $\boldsymbol{\beta}$ und σ^2, also bei <u>unbekanntem</u> σ^2, liefert

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= \arg\min\{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2 : \boldsymbol{\beta} \in \Theta\} \quad (= (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}) \qquad \text{und} \\ \hat{\sigma}^2 &= \|\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}\|^2/n \;\equiv\; \mathrm{RSS}/n && (61) \end{aligned}$$

Ihr Maximum beträgt also $l(\hat{\boldsymbol{\beta}}, \hat{\sigma}^2) = -\frac{n}{2}\log(\mathrm{RSS}/n) - \frac{n}{2}$ und wird nun in Akaikes Informationskriterium eingesetzt:

$$\mathrm{AIC} := -2\,l(\hat{\boldsymbol{\beta}}, \hat{\sigma}^2) + 2(p+1) = n\left(\log\left(\frac{\mathrm{RSS}}{n}\right) + 1\right) + 2(p+1) \tag{62}$$

Bemerkung: σ^2 steht in den betrachteten approximierenden Modellen eigentlich gar nicht „zur Disposition“, sprich: wir wollen gar nicht in Erwägung ziehen, σ^2 aus dem Modell zu entfernen. Also ist p die Dimension des tatsächlich betrachteten Parameters und in der Regel wird der rechte Summand in (62) durch $2p$ ersetzt, weil er für die Suche des Modells mit minimalem AIC egal ist, wenn σ^2 in allen diesen Modellen vorkommt.

Bei <u>bekanntem</u> σ^2 ist die log-Likelihood nur eine Funktion in $\boldsymbol{\beta}$ und lautet $l(\boldsymbol{\beta}) := -\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2/(2\sigma^2)$. Ihre Maximierung ergibt dasselbe $\hat{\boldsymbol{\beta}}$ wie zuvor und ihr Maximum beträgt nun $l(\hat{\boldsymbol{\beta}}) = -\mathrm{RSS}/(2\sigma^2)$. Wenn dieses nun in Akaikes Informationskriterium AIC eingesetzt wird, erhalten wir

$$\mathrm{AIC} := -2\,l(\hat{\boldsymbol{\beta}}) + 2p = \frac{\mathrm{RSS}}{\sigma^2} + 2p \tag{63}$$

Konvention: Im Fall eines bekannten σ^2 wird statt dem obigen AIC in (63) Mallows $C_p := \mathrm{RSS}/\sigma^2 + 2p - n$ verwendet. Für mehr Informationen bzgl. Mallows C_p siehe z. B. [46, Hocking (1996), Sec. 7.5, p. 234 und Sec. 7.6, p. 250].

Bemerkung: Lediglich der Vollständigkeit halber und falls man einmal über die Beziehung zwischen AIC und R^2 bzw. R_a^2 nachdenken will, hier die Ergebnisse des Einsetzens von R^2 bzw. R_a^2 in obige Gleichungen nach ein paar einfachen Umformungen: Im Fall von (62) (mit p anstelle von $p+1$) folgt

$$\mathrm{AIC} = \begin{cases} n\left(\log\left(1 - R^2\right) + \log\left(\dfrac{n-1}{n}\right) + \log(\hat{\sigma}_Y^2) + 1\right) + 2p \\ n\left(\log\left(1 - R_a^2\right) + \log\left(\dfrac{n-p}{n}\right) + \log(\hat{\sigma}_Y^2) + 1\right) + 2p \end{cases}$$

und im Fall von (63) ist

$$\mathrm{AIC} = \begin{cases} (n-1)\dfrac{\hat{\sigma}_Y^2}{\sigma^2}\left(1 - R^2\right) + 2p \\ (n-p)\dfrac{\hat{\sigma}_Y^2}{\sigma^2}\left(1 - R_a^2\right) + 2p \end{cases}$$

wobei $\hat{\sigma}_Y^2 = (n-1)^{-1}\sum_{i=1}^2 (Y_i - \bar{Y}_\cdot)^2$.

(Ende des Exkurses.)

10.8 Modelldiagnose II: Ausreißer, Extrempunkte, einflussreiche Punkte und Residualanalyse

Wir liefern zunächst eine Beschreibung der in der Überschrift genannten Konzepte. Eine strenge Definition im mathematischen Sinn ist nicht existent, da es *das* Kriterium zur Bestimmung beispielsweise eines Extrempunktes nicht gibt.

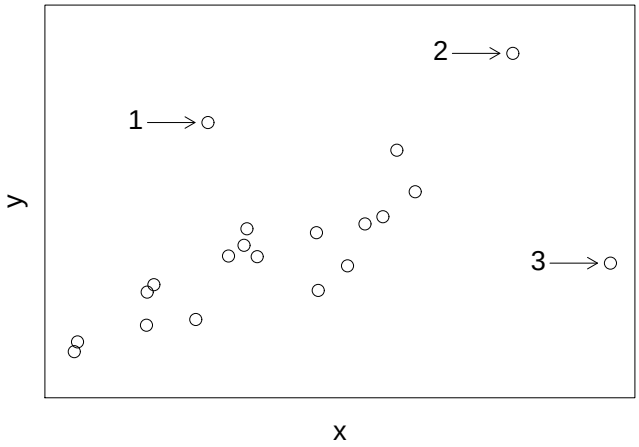
„Definition“: Ein Punkt $(\mathbf{x}_i, Y_i)$ ist ein ...

Ausreißer,	wenn sein Designwert $\mathbf{x}_i$ im „üblichen“ Bereich liegt (d. h. im Bereich der restlichen $\mathbf{x}_j$ für $j \neq i$), aber der Responsewert Y_i „zu groß“ oder „zu klein“ ist (relativ zu den Y-Werten von Designpunkten in der Umgebung von $\mathbf{x}_i$).
Extrempunkt,	wenn sein Designwert $\mathbf{x}_i$ deutlich verschieden vom Rest der $\mathbf{x}_j$, $j \neq i$ ist. (Bei der einfachen linearen Regression lässt sich das durch $\lvert x_i - \bar{x}_n \rvert \gg 0$ feststellen.) Er wird auch “leverage point” oder Hebelpunkt genannt.
einflussreicher Punkt,	wenn er ein Ausreißer oder Extrempunkt ist, dessen Einbeziehung in die Regression eine erhebliche Änderung der Parameterschätzwerte, der RSS oder der gefitteten Werte zur Folge hat.

Es stellt sich das Problem der Identifikation einflussreicher Punkte. Hier gibt es in beschränktem Maße nur für einfache Modelle grafisch-qualitative Verfahren und für komplexere Modelle nur quantitative Verfahren. Viele der in den folgenden Abschnitten aufgeführten quantitativen Verfahren findet man z. B. in [46, Hocking (1996)] beschrieben.

10.8.1 Grafische Identifizierung

Im Fall der einfachen linearen Regression sind obige Kriterien hinsichtlich ihres Erfülltseins oder Nicht-Erfülltseins anhand eines Streudiagramms der Y_i gegen die x_i möglicherweise noch gut beurteilbar, wie der folgende Plot veranschaulicht.



Der Punkt mit der Ziffer ...

1 ist offenbar ein Ausreißer bezüglich seines Y-Wertes, wohingegen sein x-Wert völlig im Rahmen der übrigen liegt.

   Es stellt sich die Frage, ob hier ein Fehler in der Messung (oder Übertragung oder Dateneingabe) eines *zu kleinen* x-Wertes vorliegt, der dazu führt, dass der Y-Wert hervorsticht, denn am rechten Rand der Punktwolke würde der Y-Wert nicht sonderlich auffallen. Oder haben wir es hier mit einem korrekten x-Wert, aber einem Fehler in der Y-Messung zu tun? In jedem Fall ist eine Datenkontrolle angeraten.

Nichtsdestotrotz wird der Einfluss dieses Datums auf die gefittete Regressionsgerade nicht allzu groß sein, da es noch einige weitere Beobachtungen mit einem ähnlichen x-Wert gibt, die die Regressionsgerade von diesem einen Datum nicht zu sehr beeinflusst sein lassen werden.

2 ist ungewöhnlich im Hinblick auf seinen x-Wert, aber nicht so sehr hinsichtlich seines Y-Wertes. Er ist zwar ein Extrempunkt, aber kein einflussreicher Punkt, da er mit der Tendenz der übrigen Beobachtungen in Einklang steht: Er liegt gewissermaßen in der „Verlängerung" der Punktewolke.

Dennoch sollte geklärt werden, wie es zu der Lücke zwischen dieser Beobachtung und den restlichen Punkten gekommen ist. Außerdem stellt sich die Frage nach der Adäquatheit des Modells im „leeren" Designbereich zwischen diesem x-Wert und den restlichen Designwerten.

3 ist offensichtlich ein Ausreißer und ein Extrempunkt. Darüber hinaus ist er äußerst einflussreich auf die Lage der Regressionsgeraden!

Es bleibt jedoch zu bedenken, dass es sich um eine korrekte Beobachtung handeln kann, die möglicherweise das Modell infrage stellt! Das Problem hierbei ist, dass zu wenige Daten in der Umgebung von Punkt 3 vorliegen, um eine Klärung der Frage der Modell-Adäquatheit zu erreichen.

Die in obigem Plot veranschaulichte grafische Analyse ist im einfachen linearen Regressionsmodell und auch noch in einem Modell mit zwei Designvariablen möglich, aber **für höherdimensionale Modelle ist sie viel schwieriger bzw. unmöglich**. Wie bereits erwähnt, existieren aber ebenfalls einige quantitative Hilfsmittel zur Identifizierung einflussreicher Punkte und zur Quantifizierung deren Einflusses. Diese Methoden stehen auch für komplexere Modelle zur Verfügung und sind Inhalt der nächsten Abschnitte.

10.8.2 Inferenzstatistische Residualanalyse

Wie bereits in Abschnitt 10.6 erwähnt, sind die Residuen $\hat{\varepsilon}_i = Y_i - \hat{Y}_i$ $(i = 1, \ldots, n)$ unter den üblichen Modellannahmen **normalverteilt, allerdings weder unabhängig noch identisch**, sondern mit

$$\mathrm{Var}(\hat{\varepsilon}_i) = \sigma^2\,(1 - h_{ii}) \qquad \text{und} \qquad \mathrm{Cov}(\hat{\varepsilon}_i, \hat{\varepsilon}_j) = -\sigma^2\, h_{ij} \quad \text{für } 1 \le i \ne j \le n.$$

Um die Residuen wenigstens approximativ auf die Varianz 1 zu skalieren und damit untereinander besser vergleichbar zu machen, wird die folgende Transformation vorgeschlagen:

Intern studentisierte Residuen (auch: standardisierte Residuen):

<u>Definition:</u> Die intern studentisierten Residuen werden definiert durch

$$\hat{\varepsilon}_{i,\ \text{int. stud.}} := \frac{\hat{\varepsilon}_i}{\hat{\sigma}\sqrt{1 - h_{ii}}} \qquad \text{für } i = 1, \ldots, n.$$

<u>Verteilung:</u> Unter der Modellannahme unabhängig und identisch $\mathcal{N}(0, \sigma^2)$-verteilter ε_i im linearen Regressionsmodell (der Dimension p) ist

$$(\hat{\varepsilon}_{i,\ \text{int. stud.}})^2 \sim (n - p)\ \beta\left(\frac{1}{2}, \frac{n - p - 1}{2}\right) \qquad \text{für } i = 1, \ldots, n.$$

Dabei steht $\beta(r, s)$ für die Beta-Verteilung mit r und s Freiheitsgraden (siehe z. B. [46, Hocking (1996), §9.4.2, S. 312]). Für großes n ist $\hat{\varepsilon}_{i,\ \text{int. stud.}}$ approximativ $\mathcal{N}(0, 1)$-verteilt.

Bemerkung: Ein schlecht gefittetes Datum Y_i bläht $\hat{\sigma}$ auf und reduziert $\hat{\varepsilon}_{i,\ \text{int. stud.}}$ dadurch. Man könnte sagen, Y_i versucht sich vor seiner Entdeckung zu schützen. Um diesen Effekt zu

kompensieren, wird eine alternative Betrachtung vorgeschlagen: Man bestimmt das Residuum zu Y_i für den Fit, den man erhält, wenn $(\mathbf{x}_i, Y_i)$ aus dem Datensatz *ausgeschlossen* wird. Dies sind sogenannte ...

Extern studentisierte Residuen:

Definition: Die extern studentisierten Residuen werden definiert durch

$$\hat{\varepsilon}_{i,\text{ ext. stud.}} := \frac{Y_i - \hat{Y}_i^{(-i)}}{\sqrt{\widehat{\text{Var}}(Y_i - \hat{Y}_i^{(-i)})}} \qquad \text{für } i = 1, \ldots, n.$$

Dabei ist $\hat{Y}_i^{(-j)}$ der Wert der auf der Basis der Daten **ohne** $(\mathbf{x}_j, Y_j)$ gefitteten Regressionsfunktion an der Stelle x_i und $\widehat{\text{Var}}(Y_i - \hat{Y}_i^{(-j)})$ ein geeigneter Schätzer für die Varianz der Differenz $Y_i - \hat{Y}_i^{(-j)}$. (Beachte, dass nur $j = i$ benötigt wird, und zur Berechnung von $\hat{\varepsilon}_{i,\text{ ext. stud.}}$ siehe „Berechnungsvereinfachung“ unten.)

Verteilung: Unter der Modellannahme unabhängig und identisch $\mathcal{N}(0, \sigma^2)$-verteilter ε_i im linearen Regressionsmodell (der Dimension p) ist (z. B. gemäß [46, Hocking (1996), §9.4.2, S. 311])

$$\hat{\varepsilon}_{i,\text{ ext. stud.}} \sim t_{n-p-1} \qquad \text{für } i = 1, \ldots, n.$$

Damit steht ein **Niveau-α-Test auf Ausreißereigenschaft** für ein *im Voraus spezifiziertes* Residuum zur Verfügung:

$$\text{Beobachtung } i \text{ ist ein Ausreißer} \quad \Longleftrightarrow \quad |\hat{\varepsilon}_{i,\text{ ext. stud.}}| > t_{n-p-1;1-\alpha/2}$$

Problem: Um zu entdecken, ob überhaupt *irgendein* Residuum einen Ausreißer darstellt, werden in Wirklichkeit *alle* Residuen gleichzeitig betrachtet und nicht ein einzelnes, voher bestimmtes. Das bedeutet, dass simultane Inferenz betrieben werden müsste. Erschwerend kommt hinzu, dass die Residuen nicht unabhängig sind.

Als **approximative Lösung** für dieses Problem wird eine Bonferroni-Methode vorgeschlagen ([82, Weisberg (1985)]):

$$\text{Beobachtung } i \text{ ist unter } n \text{ anderen ein Ausreißer} \quad \Longleftrightarrow \quad |\hat{\varepsilon}_{i,\text{ ext. stud.}}| > t_{n-p-1;1-\alpha/(2n)}$$

Als vereinfachender **Kompromiss** für ein Indiz der potenziellen Ausreißereigenschaft gilt:

$$\text{Beobachtung } i \text{ ist unter } n \text{ anderen ein Ausreißer} \quad \Longleftrightarrow \quad |\hat{\varepsilon}_{i,\text{ ext. stud.}}| > 3$$

Die gleiche Strategie wird für die intern studentisierten Residuen $\hat{\varepsilon}_{i,\text{ int. stud.}}$ verwendet.

Berechnungsvereinfachung: Man kann zeigen, dass zur Berechnung der extern studentisierten Residuen $\hat{\varepsilon}_{i,\text{ ext. stud.}}$ *nicht* für jeden der n jeweils um ein Datum reduzierten Datensätze eine Regressionsfunktion zu fitten ist. Es gilt vielmehr die Beziehung

$$\hat{\varepsilon}_{i,\text{ ext. stud.}} = \frac{\hat{\varepsilon}_i}{\hat{\sigma}^{(-i)}\sqrt{1 - h_{ii}}} = \frac{\hat{\sigma}}{\hat{\sigma}^{(-i)}}\, \hat{\varepsilon}_{i,\text{ int. stud.}},$$

wobei $\hat{\sigma}^{(-i)}$ der Schätzer für σ auf der Basis des um $(\mathbf{x}_i, Y_i)$ reduzierten Datensatzes ist und für den im linearen Regressionsmodell (der Dimension p) gilt:

$$(\hat{\sigma}^{(-i)})^2 = \frac{n-p}{n-p-1}\,\hat{\sigma}^2 - \frac{1}{n-p-1}\,\frac{\hat{\varepsilon}_i^2}{1-h_{ii}} = \frac{\hat{\sigma}^2}{n-p-1}\left(n - p - \hat{\varepsilon}_{i,\text{ int. stud.}}^2\right)$$

10.8.3 Quantitative Identifizierung einflussreicher Punkte und Quantifizierung ihres Einflusses

Die schon zu Beginn von Kapitel 10 angesprochene Projektionsmatrix $\mathbf{H} \equiv \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$ spielt im Folgenden eine entscheidende Rolle. Ihre Elemente werden mit h_{ij} für $1 \leq i, j \leq n$ bezeichnet und im Modell mit konstantem Term gilt für ihre Diagonalelemente

$$\frac{1}{n} \leq h_{ii} \leq 1, \quad i = 1, \ldots, n,$$

(ohne konstanten Term ist die untere Schranke 0) und für die gefitteten Werte

$$\hat{Y}_i = h_{ii} Y_i + \sum_{1 \leq j \neq i \leq n} h_{ij} Y_j, \quad i = 1, \ldots, n. \tag{64}$$

Aufgrund dieser Darstellung werden die Elemente der Projektionsmatrix $\mathbf{H}$ auch "leverage values" (= Hebelwerte) genannt, da einerseits die Diagonalelemente h_{ii} beeinflussen, wie sehr $\hat{Y}_i$ in die Nähe von Y_i „gehebelt“ wird, und andererseits die übrigen h_{ij} für $j \neq i$ mitbestimmen, wie weit $\hat{Y}_i$ von Y_i „weg-gehebelt“ wird.

Wir geben zwei sehr vereinfachende **Kriterien für die quantitative Identifizierung** einflussreicher Punkte an:

- Der Hebelwert h_{ii}: Im linearen Regressionsmodell (der Dimension p) gilt $(\mathbf{x}_i, Y_i)$ als ein potenziell einflussreicher Punkt, falls die Abschätzung

  $$h_{ii} > \frac{2p}{n}$$

  erfüllt ist. Memo: $\sum_{i=1}^{n} h_{ii} \equiv \text{Spur}(\mathbf{H}) = \text{Rang}(\mathbf{H}) = p$, da $\mathbf{H}$ symmetrisch und idempotent. D. h., als Schranke gilt das Doppelte des „durchschnittlichen“ Hebelwertes. (In **R** wird als Schranke sogar $3p/n$ verwendet.)

- Wegen (64) ist für $h_{ii} \approx 1$ auch $\hat{Y}_i$ nahe bei Y_i. Dies liegt daran, dass im Fall $h_{ii} \approx 1$ die übrigen h_{ij} für $j \neq i$ klein (im Sinne von nahe Null) sind. Y_i ist in diesem Fall ein Wert, der die Regressionsfunktion „in seine Nähe zwingt“, und $(\mathbf{x}_i, Y_i)$ somit ein potenziell einflussreicher Punkt.

Die **Quantifizierung des Einflusses eines Punktes** kann auf mehrere Arten erfolgen. Wir betrachten hier einige verschiedene Methoden, die quantifizieren, wie groß der Einfluss eines Punktes $(\mathbf{x}_i, Y_i)$ ist auf

1. die gefitteten Werte $\hat{Y}_1, \ldots, \hat{Y}_n$,
2. die Parameterschätzwerte $\hat{\beta}_k$ und/oder
3. den Schätzer $\hat{\sigma}^2$ für σ^2.

10.8.3.1 Einfluss eines Punktes auf $\hat{\mathbf{Y}} = (\hat{Y}_1, \ldots, \hat{Y}_n)'$

- **Cooks Abstand** D_i ist ein geeignet gewichteter Abstand zwischen dem *Vektor* $\hat{\mathbf{Y}}$ der gefitteten Werte $\hat{Y}_j$ beim Fit mit dem gesamten Datensatz und dem Vektor $\hat{\mathbf{Y}}^{(-i)}$ der gefitteten Werte $\hat{Y}_j^{(-i)}$ beim Fit ohne das Datum $(\mathbf{x}_i, Y_i)$:

  $$D_i \;:=\; \frac{(\hat{\mathbf{Y}} - \hat{\mathbf{Y}}^{(-i)})'(\hat{\mathbf{Y}} - \hat{\mathbf{Y}}^{(-i)})}{p\,\hat{\sigma}^2} \;=\; \frac{\sum_{j=1}^{n}(\hat{Y}_j - \hat{Y}_j^{(-i)})^2}{p\,\hat{\sigma}^2} \;=\; \frac{h_{ii}\,\hat{\varepsilon}_i^2}{p\,\hat{\sigma}^2\,(1-h_{ii})^2}$$

Beachte: D_i ist groß, falls $\hat{\varepsilon}_i \gg 0$ oder falls $h_{ii} \approx 1$ ist.

Es gibt mehrere Vorschläge, wie Cooks Abstand für die Entscheidung, ob $(\mathbf{x}_i, Y_i)$ ein einflussreicher Punkt ist, herangezogen werden soll: $(\mathbf{x}_i, Y_i)$ ist einflussreich, falls

$$D_i > \begin{cases} F_{p,n-p;1-\alpha} & \text{für } \alpha = 0.1 \text{ [23, Cook (1977)] bzw. für } \alpha = 0.5 \text{ ([82, Weisberg (1985)]);} \\ 1 & \text{für } n \text{ groß; entspricht ganz grob } \alpha = 0.5; \\ \text{Rest der Cook-Abstände} & \longleftarrow \text{Angeblich in der Praxis verwendet.} \end{cases}$$

Die Weisbergsche Version für $\alpha = 0.5$ ist in **R** implementiert.

- **DFFITS** (= **Dif**ferenzen zwischen den **FITS** (= gefittete Werte) an den Stellen $\mathbf{x}_i$ beim Fit mit und ohne Datum $(\mathbf{x}_i, Y_i)$, siehe z. B. [46, Hocking (1996), §9.4.5, S. 314]):

$$\text{DFFITS}_i := \frac{\hat{Y}_i - \hat{Y}_i^{(-i)}}{(\hat{\sigma}^{(-i)})^2\, h_{ii}} = \sqrt{p \frac{\hat{\sigma}^2}{(\hat{\sigma}^{(-i)})^2} D_i} = \sqrt{\frac{h_{ii}\, \hat{\varepsilon}_i^2}{(\hat{\sigma}^{(-i)})^2\, (1 - h_{ii})^2}},$$

wobei D_i Cooks Abstand ist und $(\hat{\sigma}^{(-i)})^2$ wie zuvor der Schätzer für σ^2 auf der Basis des um $(\mathbf{x}_i, Y_i)$ reduzierten Datensatzes. Damit sind hier die analogen Kriterien wie oben bei D_i anzulegen. In **R** wird als Kriterium jedoch $|\text{DFFITS}_i| > 3\sqrt{p/(n-p)}$ verwendet.

10.8.3.2 Einfluss eines Punktes auf $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \ldots, \hat{\beta}_{p-1})'$

- Ebenfalls **Cooks Abstand** D_i**:** Für die p-dimensionalen Vektoren $\hat{\boldsymbol{\beta}}$ und $\hat{\boldsymbol{\beta}}^{(-i)}$ gilt:

$$D_i = \frac{(\hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}^{(-i)})'\mathbf{X}'\mathbf{X}(\hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}^{(-i)})}{p\, \hat{\sigma}^2},$$

wobei $\mathbf{X}$ die Designmatrix ist und $\hat{\boldsymbol{\beta}}^{(-i)}$ analog zu $\hat{\mathbf{Y}}^{(-i)}$ zustande kommt. D_i ist somit ein Abstandsmaß für die Vektoren $\hat{\boldsymbol{\beta}}$ und $\hat{\boldsymbol{\beta}}^{(-i)}$ in der „$\mathbf{X}'\mathbf{X}$-Metrik“, was übrigens Cooks ursprüngliche Definition für D_i war (in [23, Cook (1977)]).

- **DFBETAS** (= **Dif**ferenz zwischen den **BETAS** (= geschätzte Parameter) beim Fit mit und ohne Datum $(\mathbf{x}_i, Y_i)$; vorgeschlagen wohl in [5, Belsley et al. (1980)]; siehe hierfür z. B. [24, Cook & Weisberg (1982)]):

$$\text{DFBETAS}_{ki} := \frac{\hat{\beta}_k - \hat{\beta}_k^{(-i)}}{\hat{\sigma}^{(-i)}\, \sqrt{c_{k+1,k+1}}} \qquad \text{für } k = 0, 1, \ldots, p-1,$$

wobei c_{ll} das l-te Diagonalelement von $(\mathbf{X}'\mathbf{X})^{-1}$ für $l = 1, \ldots, p$ ist.

Im einfachen linearen Modell ist speziell für $k = 0, 1$:

$$c_{k+1,k+1} = \begin{cases} \dfrac{\sum_{i=1}^n x_i^2}{n \sum_{i=1}^n (x_i - \bar{x}_n)^2} & \text{für } k = 0; \\ \dfrac{1}{\sum_{i=1}^n (x_i - \bar{x}_n)^2} & \text{für } k = 1. \end{cases}$$

Ein großer Wert für $|\text{DFBETAS}_{ki}|$ deutet auf einen großen Einfluss von Y_i auf $\hat{\beta}_k$ hin. Faustregel: $|\text{DFBETAS}_{ki}|$ gilt als „groß“, falls

$$|\text{DFBETAS}_{ki}| > \begin{cases} 1 & \text{für „kleines“ bis „mittleres“ } n \text{ (\textbf{R}-Implementation);} \\ \frac{2}{\sqrt{n}} & \text{für „großes“ } n. \end{cases}$$

10.8.3.3 Einfluss eines Punktes auf $\hat{\sigma}^2$

Für das Verhältnis des Varianzschätzers $(\hat{\sigma}^{(-i)})^2$ des reduzierten Datensatzes zu dem des vollen Datensatzes $\hat{\sigma}^2$ gilt:

$$\left(\frac{\hat{\sigma}^{(-i)}}{\hat{\sigma}}\right)^2 = \frac{n-p}{n-p-1}\left(1 - \frac{\hat{\varepsilon}^2_{i,\text{ int. stud.}}}{n-p}\right) \sim \frac{n-p}{n-p-1}\,\beta\left(\frac{n-p-1}{2}, \frac{1}{2}\right)$$

In **R** wird dieses Verhältnis anders skaliert verwendet, nämlich in Form der sogenannten COVRATIO. Sie ist proportional zum Verhältnis der Volumina der $(1-\alpha)$-Konfidenzellipsen (COV = "confidence volume") für $\boldsymbol{\beta}$ basierend auf $\hat{\boldsymbol{\beta}}$ bzw. $\hat{\boldsymbol{\beta}}^{(-i)}$:

$$\text{COVRATIO}_i := \frac{1}{1-h_{ii}}\left(\frac{\hat{\sigma}^{(-i)}}{\hat{\sigma}}\right)^{2p} \propto \left(\frac{F_{n,n-p;1-\alpha}}{F_{n,n-p-1;1-\alpha}}\right)^p * \left(\frac{\text{Vol}\left(\mathcal{E}\left(\hat{\boldsymbol{\beta}}^{(-i)}\right)\right)}{\text{Vol}\left(\mathcal{E}\left(\hat{\boldsymbol{\beta}}\right)\right)}\right)^2,$$

wobei $\mathcal{E}(\cdot)$ die auf ihrem Argument basierende Konfidenzellipse für $\boldsymbol{\beta}$ ist (vorgeschlagen wohl in [5, Belsley et al. (1980)]; siehe hierfür z. B. [24, Cook & Weisberg (1982)]).
Als potenziell einflussreicher Punkt gilt in **R** derjenige, welcher die Ungleichung

$$|1 - \text{COVRATIO}_i| > \frac{3p}{n-p}$$

erfüllt.

Bemerkung: Zu klären ist, was wohl – speziell in **R** – mit gewissen der obigen Einflussgrößen wie dem standardisierten Residuun, dem Cook-Abstand, den DFFITS und der COVRATIO geschieht, wenn $h_{ii} = 1$ ist!

10.8.4 Zusammenfassung und Umsetzung

Für einen konkreten Datensatz erscheint es sinnvoll, vorgenannte Einflussstatistiken zusammenfassend in einer Tabelle der folgenden Art zu präsentieren oder in Grafiken, in denen zusätzlich die jeweils kritischen (Schwellen-)Werte eingezeichnet sind:

Nr. i	Intern studentisierte Residuen	Extern studentisierte Residuen	$\lvert\text{DFBETAS}_{ki}\rvert$ $k=0$	…	$k=p-1$	DF-FITS$_i$	COV-RATIO$_i$	Cooks D_i	Hebel-werte h_{ii}
1	0.083	0.080	0.033	…	-0.030	0.037	1.377	0.001	0.175
2	-0.746	-0.735	-0.306	…	0.279	-0.338	1.284	0.059	0.175
⋮	⋮	⋮	…	⋮	⋮	⋮	⋮	⋮	⋮
$n-1$	-1.114	-1.123	0.374	…	-0.426	-0.516	1.173	0.131	0.175
n	-1.137	-1.148	0.382	…	-0.436	-0.528	1.165	0.137	0.175

10.8.4.1 Die Funktion `influence.measures()` und Co.

Die **R**-Funktionen, die die Berechnung der obigen Statistiken durchführen, heißen

- `rstandard()` für die intern studentisierten Residuen,
- `rstudent()` für die extern studentisierten Residuen,
- `cooks.distance()` für die Cook-Abstände,
- `dffits()` für die DFFITS,
- `hatvalues()` für die Hebelwerte,
- `dfbetas()` für die DFBETAS und

- `covratio()` für die skalierten Varianz- bzw. Konfindenzvolumenverhältnisse.

Sie können alle separat genutzt werden, aber bis auf die ersten beiden sind alle obigen Funktionen in `influence.measures()` gewissermaßen zusammengefasst und damit auch „auf einen Schlag“ aufrufbar. Die Funktion `influence.measures()` angewendet auf ein `lm`-Objekt liefert eine Liste mit drei Komponenten zurück, deren erste die $(n \times (p+4))$-Matrix der Werte der Einflussstatistiken (ohne die studentisierten Residuen) enthält. Diese Liste hat die Klasse `infl` und ihr Inhalt wird automatisch als Tabelle aufbereitet dargestellt (ergänzt durch eine weitere Spalte `inf`).

Zur Demonstration der Arbeitsweise der obigen Diagnosefunktionen betrachten wir als Beispiel die bereits aus Abschnitt 10.3 bekannten Ozon-Daten, die in dem Data Frame `air` gespeichert sind und für die wir ein multiples lineares Modell von `Ozone` an `Temp` und `Solar.R` fitteten.

Beispiel:

```
> influence.measures( oz2.lm)
Influence measures of
         lm(formula = Ozone ~ Temp + Solar.R, data = air) :

       dfb.1_  dfb.Temp  dfb.Sl.R    dffit cov.r   cook.d     hat inf
1    1.22e-01 -0.115726  0.039279  0.15034 1.023 7.53e-03 0.02218
2    3.12e-02 -0.018395 -0.025980  0.05837 1.038 1.14e-03 0.01542
3   -3.99e-02  0.023508  0.022946 -0.08831 1.020 2.61e-03 0.01121
4    3.68e-02 -0.042921  0.039252  0.05484 1.104 1.01e-03 0.06966   *
7    2.30e-02 -0.026909  0.025914  0.03609 1.084 4.38e-04 0.05241   *
....
153 -9.57e-03  0.009803 -0.006144 -0.01317 1.053 5.83e-05 0.02379
```

Die Spaltentitel sind zum Teil stark abgekürzt, insbesondere die der DFBETAS, weil darin auch noch die Namen der Modellterme (wie z. B. `1_` für den Intercept-Term und `Sl.R` für den Term `Solar.R`) untergebracht werden. Trotzdem sind sie noch gut zuzuordnen. Die letzte Spalte `inf` (= Abk. für “influential”) dient der „grafischen“ Identifizierung potenziell einflussreicher Punkte. Sie enthält genau dann einen Stern `*`, wenn mindestens einer der Einflussstatistikwerte der jeweiligen Zeile seine kritische Schwelle überschritten hat. Die korrespondierende Beobachtung könnte demzufolge als einflussreicher Punkt erachtet werden. (Die standardmäßig nicht gezeigte, zweite Komponente eines jeden `infl`-Objektes heißt `$is.infl` und enthält die logische $(n \times (p+4))$-Matrix, die elementweise für jeden der oben gezeigten Werte angibt, ob er oberhalb seiner kritische Schwelle liegt. Aus dieser logischen Matrix leitet **R** durch zeilenweise Betrachtung den Eintrag der Spalte `inf` ab.)

Die Ausgabe von `influence.measures()` kann etwas umfangreich ausfallen und falls man einzig an den potenziell einflussreichen Punkten interessiert ist, überwiegend Überflüssiges zeigen. Eine kompaktere Zusammenfassung liefert die `summary()`-Methode `summary.infl()`, die aktiviert wird, wenn `summary()` auf das Resultat von `influence.measures()` angewendet wird. Sie zeigt nur die Einflussstatistikwerte der potenziell einflussreichen Punkte, aber darüber hinaus sind dafür die „verantwortlichen“ Einflussstatistikwerte durch das Anhängsel `_*` markiert:

```
> summary( influence.measures( oz2.lm))
Potentially influential observations of
         lm(formula = Ozone ~ Temp + Solar.R, data = air) :

    dfb.1_ dfb.Temp dfb.Sl.R dffit   cov.r   cook.d hat
4    0.04  -0.04     0.04     0.05    1.10_*  0.00   0.07
```

```
7    0.02  -0.03     0.03     0.04    1.08_*  0.00   0.05
16  -0.04   0.05    -0.05    -0.06    1.10_*  0.00   0.07
19   0.01  -0.02     0.02     0.02    1.08_*  0.00   0.05
21   0.10  -0.07    -0.07     0.13    1.09_*  0.01   0.06
30   0.00   0.00     0.12     0.31    0.81_*  0.03   0.01
62  -0.16   0.13     0.24     0.45    0.78_*  0.06   0.02
117 -0.10   0.09     0.27     0.63_*  0.48_*  0.10   0.01
```

Die Funktionen `rstandard()` und `rstudent()` angewendet auf das `lm`-Objekt liefern die Vektoren der intern bzw. extern studentisierten Residuen.
Beispiel:

```
> rstandard( oz2.lm)
          1           2           3 ....         152         153
 0.99814545  0.46806987 -0.83073130 .... -0.72534464 -0.08475904
> rstudent( oz2.lm)
          1           2           3 ....         152         153
 0.99812817  0.46637112 -0.82953096 .... -0.72374378 -0.08436853
```

10.8.4.2 Die Funktion `summary.lm()`

Das Resultat der Funktion `summary()` angewendet auf ein `lm`-Objekt haben wir schon oft verwendet, aber eigentlich nur um den Fit eines Modells zu begutachten. Bei der Anwendung von `summary()` auf ein `lm`-Objekt kommt (von Benutzer und Benutzerin unbemerkt) jedoch automatisch die Funktion `summary.lm()` zum Einsatz, deren Ergebnis ein sogenanntes `summary.lm`-Objekt ist. Dies ist eine Liste mit zahlreichen Komponenten, auf die man natürlich zugreifen kann, um eventuell weitere Berechnungen durchzuführen. Es sind bis zu 13 Komponenten, deren wichtigste wir hier kurz erwähnen:

- `call` enthält den Aufruf, mit dem das Modell gebildet wurde.
- `residuals` ist der Vektor der üblichen Residuen $\hat{\varepsilon}_i$ (gewichtet, falls es eine gewichtete Regression war). Erhält man direkt durch die "Extraktor"-Funktion `resid()`.
- `coefficients` enthält die $(p \times 4)$-Matrix der Koeffizientenschätzwerte, deren Standardfehler, t-Teststatistikwerte und p-Werte. (Sie ist durch die Extraktor-Funktion `coef()` angewendet auf das `lm`-Objekt teilweise zu erhalten oder durch die Anwendung von `coef()` auf das `summary.lm`-Objekt vollständig.)
- `sigma` ist die Residuenstandardabweichung $\hat{\sigma} \equiv \sqrt{RSS/(n-p)}$.
- In `df` sind die Freiheitsgrade p und $n-p$ (sowie als drittes die Anzahl der sogenannten "non-aliased" Koeffizienten, worauf wir nicht eingehen).
- `r.squared` ist das multiple R^2. Kann durch `deviance()` direkt aus dem `lm`-Objekt extrahiert werden.
- `adj.r.squared` ist das multiple korrigierte ("adjusted") R^2.
- `fstatistic` ist der dreielementige Vektor, dessen Elemente den Wert der F-Teststatistik, ihre Zähler- und Nennerfreiheitsgrade enthalten.
- `cov.unscaled` ist $(\mathbf{X}'\mathbf{X})^{-1}$.
- In `correlation` steht die geschätzte $(p \times p)$-Korrelationsmatrix $(\hat{\rho}_{kl})_{1 \le k,l \le p}$ des Parameterschätzers $\hat{\boldsymbol{\beta}}$, wobei $\widehat{\mathrm{cor}}(\hat{\beta}_{k-1}, \hat{\beta}_{l-1}) \equiv \hat{\rho}_{kl} = c_{kl}/\sqrt{c_{kk}c_{ll}}$ und c_{kl} das (k,l)-Element von $(\mathbf{X}'\mathbf{X})^{-1}$ ist.

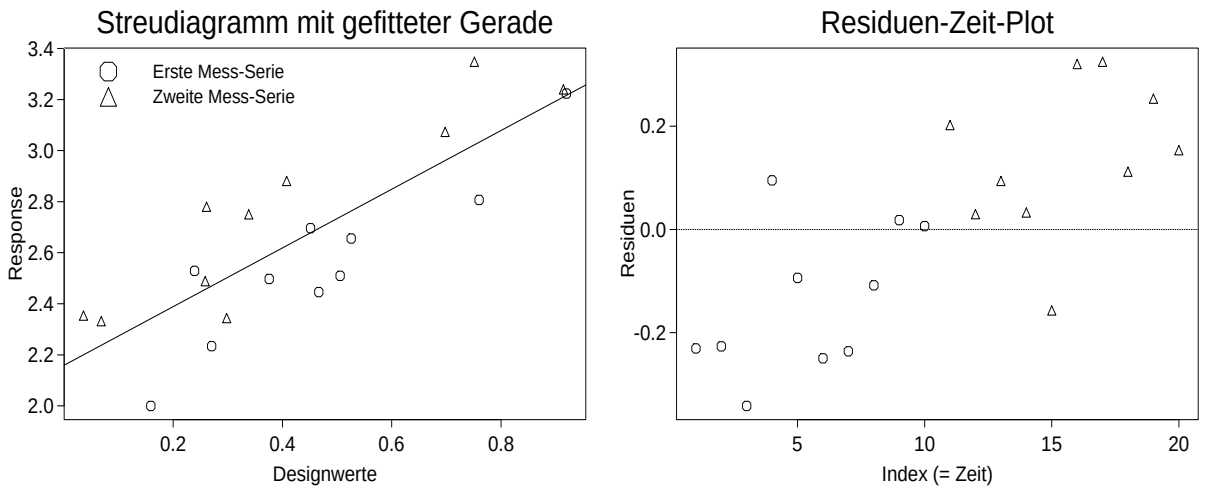
10.8.5 Zur Unabhängigkeitsannahme der Fehler

Die wesentliche **Grundannahme** in der linearen Regression ist, dass $\varepsilon_1, \ldots, \varepsilon_n$ unabhängig und identisch $\mathcal{N}(0, \sigma^2)$-verteilt sind. Auf ihr basieren alle inferenzstatistischen Aussagen über die Parameterschätzer und die gefitteten Werte. Man weiß, die Inferenz in der Regressionsanalyse ist

- relativ robust gegenüber moderaten Abweichungen von der Normalverteilungsannahme, **aber**
- empfindlich gegenüber Abweichungen von den Annahmen der Unabhängigkeit oder der Varianzhomogenität!

Die zeitlich sequenzielle Erhebung der Y_i kann zu einer Korrelation der jeweils assoziierten Fehler führen. Dies ist möglicherweise mittels einer grafischen Analyse durch einen Plot der **Residuen** $\hat{\varepsilon}_i$ gegen ihre Indizes i, also **gegen die Zeit**, aufdeckbar.

Ein synthetisches **Beispiel** soll das veranschaulichen: Angenommen, es wurden zwei Mess-Serien für denselben Produktionsprozess bei verschiedenen Designwerten zur Bestimmung seines Ertrages erstellt, und zwar eine frühe und eine späte. Bei der zweiten Mess-Serie war der Prozess schon „eingespielt“ und erzielte über den gesamten Designwertebereich hinweg tendenziell höhere Responsewerte. In dem einfachen linearen Regressionsmodell, das diesen zeitlichen Effekt nicht beachtet, wäre die im linken Plot präsentierte Regressionsgerade gefittet worden. Der Residuenplot gegen die Zeit deckt die dadurch entstandene zeitliche Korrelation der Fehler jedoch auf: Die Störungen sind positiv mit der Zeit korreliert, d. h., *nicht* unabhängig!



Natürlich sind auch andere zeitliche Abhängigkeitsstrukturen der Störungen möglich.

Bemerkungen zu Tests: Es existieren auch statistische Tests, die es ermöglichen, die Unabhängigkeitsannahme zu prüfen. Einige gehen dabei von sehr konkreten Modellen für die Abhängigkeitsstruktur der Störungen aus. Ein Beispiel hierfür ist der Durbin-Watson-Test auf Korrelation, der ein autoregressives Schema 1. Ordnung für die ε_i zugrunde legt (siehe z. B. [33, Fox (1997)] und die Funktion `durbin.watson()` im **R**-Paket `car` oder die Funktion `dwtest()` im **R**-Paket `lmtest`). Andere Tests sind nichtparametrischer Natur, wie beispielsweise der Runs-Test (siehe z. B. [16, Büning und Trenkler (1994)] und die Funktion `runs.test()` im **R**-Paket `lawstat` oder `runstest()` im **R**-Paket `tseries`). Wir gehen hier nicht weiter auf derlei Tests ein.

10.9 Schätz- und Prognosewerte sowie Konfidenz- und Toleranzintervalle im linearen Regressionsmodell

Im Rahmen eines linearen Modells lässt sich, wie gesehen, der Parametervektor $\boldsymbol{\beta}$ der Regressionsfunktion recht einfach schätzen. Zumindest innerhalb des Beobachtungsbereichs des Designs existiert damit auch für jede Stelle ein guter Schätzer für den zugehörigen Regressionsfunktionswert: Es ist der Wert der gefitteten Regressionsfunktion an eben dieser Stelle. Außerdem lassen sich, wie wir sehen werden, unter den gemachten (Normal-)Verteilungsannahmen **Konfidenzintervalle** für die Regressionsfunktionswerte angeben. Letzteres gilt auch für die Komponenten des Parametervektors $\boldsymbol{\beta}$.

Die Bedeutung der Regression liegt ferner darin, dass sie – vorausgesetzt das Modell ist und bleibt korrekt! – die Prognose *zukünftiger* Responses für gegebene Werte der Covariablen erlaubt: Es sind (auch dies) die Werte der gefitteten Regressionsfunktion an den zugehörigen Designstellen. Zu diesen Prognosewerten können unter der Bedingung, dass die Verteilungsannahmen auch zukünftig gelten, **Toleranzintervalle** berechnet werden.

Werden mehrere Designstellen betrachtet, so muss sowohl bei den Konfidenzintervallen als auch bei den Toleranzintervallen genau unterschieden werden, ob sie das gewählte Niveau jeweils **punktweise** (d. h. **einzeln**) oder **simultan** (d. h. **gemeinsam**) einhalten sollen. Zur Vollständigkeit listen wir die relevanten Intervallformeln im Modell der multiplen linearen Regression in den folgenden Abschnitten auf und gehen für das einfache lineare Regressionsmodell der Anschaulichkeit halber in §10.9.6 nochmal ganz explizit darauf ein.

Zur Wiederholung: Der Parametervektor $\boldsymbol{\beta}' = (\beta_0, \beta_1, \ldots, \beta_{p-1})$ ist p-dimensional und die Designmatrix $\mathbf{X}$ ist eine $(n \times p)$-Matrix mit $n \geq p$ und maximalem Rang p. Der beste lineare unverzerrte Schätzer (Engl.: "best linear unbiased estimator", kurz BLUE) der Regressionsfunktion an der Designstelle $\mathbf{x}$ ist $\mathbf{x}'\hat{\boldsymbol{\beta}}$ und der Schätzer für seine Standardabweichung $\sigma_{\mathbf{x}'\hat{\boldsymbol{\beta}}}$ (= "standard error of the fit") lautet $\hat{\sigma}\sqrt{\mathbf{x}'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}}$, wobei $\hat{\sigma}^2 = \mathrm{RSS}/(n-p)$ ist. (Zur Notation siehe nötigenfalls nochmal Anfang von Kapitel 10 und in Abschnitt 10.1.)

In **R** steht die Funktion `predict()` zur Verfügung, mit deren Hilfe wir eine gefittete Regressionsfunktion $\mathbf{x} \mapsto \mathbf{x}'\hat{\boldsymbol{\beta}}$ an beliebigen Designstellen auswerten können. Sie leistet auch die Berechnung punktweiser Konfidenz- oder Toleranzintervalle. Hingegen stehen simultane Konfidenzintervalle (gemäß der Bonferroni- oder der Scheffé-Methode) bzw. ein (Scheffé-)Konfidenzband für die gesamte Regressionsfunktion oder ein simultaner Konfidenzbereich für den Parametervektor $\boldsymbol{\beta}$ in "base **R**" (!) bisher anscheinend *nicht* zur Verfügung. Sie können aber mit geringem Aufwand selbst implementiert werden.
Allerdings: Das **R**-Paket `car` enthält eine Funktion `confidence.ellipse()`, die die zweidimensionale Konfidenz-Ellipse für zwei auszuwählende Komponenten von $\boldsymbol{\beta}$ zeichnet. Siehe die betreffende Online-Hilfe bzw. [33, Fox (1997)]. Das **R**-Paket `multcomp` von [11, Bretz et al. (2010)], welches in verschiedenen Kontexten sehr leistungsfähige Funktionen für Verfahren des multiplen Testens zur Verfügung stellt, erlaubt die Berechnung simultaner Konfidenzintervalle oder auch eines solchen Bandes im linearen Modell. Die Vorgehensweise wird in der „Vignette" `multcomp-examples` des Paketes beschrieben. (Das englische Wort "vignette" bedeutet hierbei soviel wie Kurzdarstellung.) An diese Vignette gelangt man, falls das Paket installiert ist, mit `vignette( "multcomp-examples", package = "multcomp")`. (Siehe auch `?vignette`.)

10.9.1 Schätzwerte für die Regressionsfunktion und grafische Darstellung

Wird der Funktion `predict()` als erstes Argument ein `lm`-Objekt übergeben, so wertet sie dessen geschätzte Regressionsfunktion aus. (Faktisch kommt dabei als Methode die Funktion `predict.lm()` zum Einsatz, weswegen bei Fragen also *deren* Online-Hilfe konsultiert werden sollte.) Als weiteres Argument kann ein Data Frame angegeben werden, der zeilenweise die

Werte der Covariablenvektoren enthält, für die die Auswertung erfolgen soll. Die Variablennamen (d. h. Spaltennamen) in diesem Data Frame müssen dieselben sein, die als Covariablen im `lm`-Objekt auftreten. (Er kann auch weitere Variablen enthalten.) Wird der Data Frame weggelassen, erhält man die gefitteten Werte $\hat{y}_1, \ldots, \hat{y}_n$ geliefert.

Wir werden im Folgenden zwei **Beispielmodelle** für den bereits aus vorherigen Abschnitten bekannten Ozon-Datensatz betrachten, um die Arbeitsweise von `predict()` zu beschreiben. Zur Übersicht und zu Referenzzwecken „erzeugen“ wir die beiden unterschiedlich komplexen Modelle hier und dokumentieren sie ausschnittsweise:

```
> summary( oz1.lm <- lm( Ozone ~ Temp + I( Temp^2), air))
....
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 292.95885  123.92019   2.364 0.019861 *
Temp         -9.22680    3.25493  -2.835 0.005476 **
I(Temp^2)     0.07602    0.02116   3.593 0.000494 ***
....

> summary( oz2.lm <- update( oz1.lm, ~ . + Solar.R + I( Solar.R^2)))
....
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.358e+02  1.215e+02   2.765 0.006725 **
Temp         -1.054e+01  3.200e+00  -3.293 0.001348 **
I(Temp^2)     8.265e-02  2.070e-02   3.993 0.000121 ***
Solar.R       1.763e-01  1.057e-01   1.667 0.098422 .
I(Solar.R^2) -3.296e-04  3.059e-04  -1.077 0.283779
....
```

Offenbar hängt das Modell `oz1.lm`, obwohl sein Parametervektor $\boldsymbol{\beta}$ dreidimensional ist, nur von *einer* Covariablen, nämlich der Temperatur ab, da die beiden Modellterme `Temp` und `I( Temp^2)` im Modell aus demselben Wert berechnet werden. (Zum Terminus „Modellterme“ siehe nötigenfalls nochmal den Beginn von Kapitel 10.) Das andere Modell, `oz2.lm`, ist fünfdimensional, hängt jedoch nur von *zwei* Covariablen ab, nämlich von Temperatur und Strahlung. Auch hier sind alle Modellterme aus diesen beiden Werten ableitbar.

Will man eine geschätzte Regressionsfunktion an, sagen wir, $k \geq 1$ verschiedenen Designstellen $\tilde{\mathbf{x}}_j = (\tilde{x}_{j0}, \tilde{x}_{j1}, \ldots, \tilde{x}_{j,p-1})'$ für $j = 1, \ldots, k$ auswerten, so braucht man also lediglich die den Modelltermen zugrundeliegenden Covariablenvektoren $\tilde{\mathbf{z}}_j$, $j = 1, \ldots, k$, um die Designvektoren $\tilde{\mathbf{x}}_j = \tilde{\mathbf{x}}(\tilde{\mathbf{z}}_j)$ und schließlich $\hat{y}(\tilde{\mathbf{x}}_j) = \tilde{\mathbf{x}}_j'\hat{\boldsymbol{\beta}}$ zu bestimmen. Dazu ist es in **R** notwendig, die Covariablenvektoren $\tilde{\mathbf{z}}_j$, $j = 1, \ldots, k$, als *Zeilen* erst in einen Data Frame zu „packen“. Seine Variablen, sprich Spaltennamen müssen so lauten wie die Spalten desjenigen Data Frames, der zur Erzeugung des `lm`-Objekts verwendet wurde.

Beispiele:

1. Wir wollen das Modell `oz2.lm` an den drei Temperatur-Strahlung-Wertepaaren $(60, 200)$, $(64, 250)$ und $(68, 300)$ auswerten, haben die (marginalen) Temperatur- und Strahlungswerte jedoch in separaten Vektoren gespeichert:

   ```
   > newtemp <- c( 60, 64, 68);     newrad <- c( 200, 250, 300)
   ```

   Dann müssen diese beiden Vektoren mit Hilfe von `data.frame()` in einem Data Frame so zusammengefasst werden, wie er von `predict()` für `oz2.lm` benötigt wird. Beachte die Wahl der Spaltennamen des Data Frames:

```
> data.frame( Temp = newtemp, Solar.R = newrad)
  Temp Solar.R
1   60     200
2   64     250
3   68     300
```

2. Will man ein Modell, das auf mindestens zwei Covariablen beruht, auf einem vollständigen (endlichen) *Gitter* auswerten und hat man die dabei zu durchlaufenden (Marginal-)Werte der Covariablen als separate Vektoren gespeichert, dann erzeugt die Funktion `expand.grid()` aus letzteren das entsprechende Gitter (Engl.: "grid"):

   Soll das Modell `oz2.lm` für alle Kombinationen der (marginalen) Temperaturwerte in $T = \{50, 55, \ldots, 105\}$ mit den (marginalen) Strahlungswerten in $S = \{0, 20, \ldots, 340\}$ ausgewertet werden, d. h. auf dem Gitter $T \times S$, und sind T und S als separate Vektoren à la

```
> newtemp2 <- seq( 50, 105, by = 5);   newrad2 <- seq( 0, 340, by = 20)
```

   gespeichert, dann erhalten wir das gewünschte Gitter als Data Frame mit geeignet benannten Spalten durch

```
> expand.grid( Temp = newtemp2, Solar.R = newrad2)
    Temp Solar.R
1     50       0
2     55       0
.... ....   ....
12   105       0
13    50      20
.... ....   ....
204  105     320
205   50     340
.... ....   ....
216  105     340
```

   Offenbar durchläuft die erste Komponente des resultierenden Data Frames ihren Wertebereich vollständig, bevor die zweite Komponente auf ihren nächsten Wert „springt". (Analog würde sich der Wert einer j-ten Komponente, falls vorhanden, erst auf den nächsten ändern, wenn sämtliche Kombinationen in den ersten $j-1$ Komponenten durchlaufen sind, usw.)

Mit den eben eingeführten vier „Hilfsvektoren" `newtemp`, `newrad`, `newtemp2` und `newrad2` wird `predict()` nun vorgestellt:

Schätzwerte für die Regressionsfunktion und grafische Darstellung	

`> (oz1pred <- predict( oz1.lm,`
`+ newdata = data.frame( Temp =`
`+ newtemp2)))`
`1 2 .... 12`
`21.67904 15.45770 .... 162.31067`
	`predict()` wertet das Modell `oz1.lm` mit dem `newdata`-Argument aus, welchem die 12 Werte in `newtemp2` unter dem Namen `Temp` in Form eines Data Frames zugewiesen wurden: Aus $\tilde{z}_j =$ `newtemp2[j]` wird intern $\tilde{\mathbf{x}}_j' = (1, \tilde{z}_j, \tilde{z}_j^2)$ und schließlich $\hat{y}(\tilde{\mathbf{x}}_j) = \tilde{\mathbf{x}}_j'\hat{\boldsymbol{\beta}}$ für $j = 1, \ldots, 12$.

`> with( air, plot( Temp, Ozone,`
`+ xlim = range( Temp, newtemp2),`
`+ ylim = range( Ozone, oz1pred),`
`+ xlab = "Temperature",`
`+ ylab = "Ozone", main = "oz1.lm"))`
`> lines( newtemp2, oz1pred)`
	Plottet das Streudiagramm der beobachteten `Temp`- und `Ozone`-Paare (z_i, y_i) so, dass sowohl diese als auch die „neuen" Temperaturwerte $\tilde{z}_j$ und die gefitteten Ozonwerte $\hat{y}(\tilde{\mathbf{x}}_j)$ in die Achsenlimits passen. `lines` fügt die gefittete Regressionsfunktion als Polygonzug durch $(\tilde{z}_j, \hat{y}(\tilde{\mathbf{x}}_j))$ hinzu. (Bild nächste Seite unten links.)

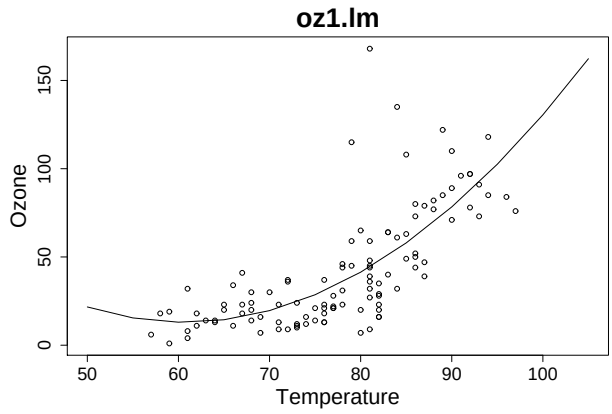
`> predict( oz2.lm, newdata =`
`+ data.frame( Temp = newtemp,`
`+ Solar.R = newrad))`
`1 2 3`
`23.01854 23.25397 24.48627`
	Für das Modell `oz2.lm` benötigt `predict()` im `newdata` zugewiesenen Data Frame *alle* Covariablen des Modells (hier `Temp` und `Solar.R`). Die diesen Variablen zugewiesenen Vektoren (hier `newtemp` und `newrad`) müssen gleich lang sein.

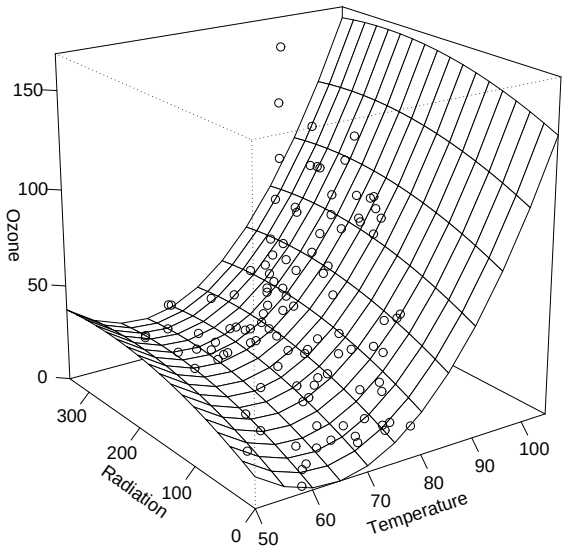
`> (oz2pred <- predict( oz2.lm,`
`+ newdata = expand.grid(`
`+ Temp = newtemp2,`
`+ Solar.R = newrad2)))`
`1 2 .... 216`
`15.42371 6.11790 .... 162.17885`
	Hier wird `oz2.lm` auf dem regulären Gitter `newtemp2` × `newrad2` ausgewertet, und zwar in der Reihenfolge, in der dieses Gitter in dem durch `expand.grid()` erzeugten Data Frame (mit den Spaltennamen `Temp` und `Solar.R`) durchlaufen wird.

`> zout <- persp( x = newtemp2,`
`+ y = newrad2,`
`+ z = matrix( oz2pred,`
`+ nrow = length( newtemp2)),`
`+ xlab = "Temperature",`
`+ ylab = "Radiation",`
`+ zlab = "Ozone",`
`+ zlim = range( oz2pred, air$Ozone),`
`+ ticktype = "detailed",`
`+ theta = -35, phi = 20, r = 9)`
	`persp()` generiert einen 3D-perspektivischen Plot von (linear interpolierten) Funktionswerten $z_{ij} = f(x_i, y_j)$ über einem 2D-Gitter (Bild unten rechts). Die Argumente `x` und `y` müssen die (aufsteigend sortierten!) Marginalien dieses Gitters sein. `z` erwartet eine (`length(x)` × `length(y)`)-Matrix, deren Element `z[ i, j]` den Funktionswert f(`x[ i]`, `y[ j]`) enthält. Beachte, wie der *Vektor* `oz2pred` für `z` verwendet wird! (Zur Bedeutung von `ticktype`, `theta`, `phi` und `r` siehe die Online-Hilfe!)

`> points( trans3d( x = air$Temp,`
`+ y = air$Solar.R,`
`+ z = air$Ozone,`
`+ pmat = zout))`
	Mit dem Resultat von `persp()` (hier `zout`) und der Funktion `trans3d()` werden weitere Punkte auf den 3D-Plot projiziert, wie im Bild unten rechts, wo die Originalbeobachtungen hinzugefügt sind.

`> (oz2hat <- predict( oz2.lm))`
`1 2 .... 153`
`22.24061 21.60098 .... 24.18454`
 	
`> plot( oz2hat, air$Ozone)`
`> abline( 0, 1, lty = 2)`
	Ohne eine Zuweisung an `newdata` werden die gefitteten Werte $\hat{y}_i = \hat{y}(\mathbf{x}_i)$ mit den Stellen $\mathbf{x}_i$ des *ursprünglichen* Designs geliefert. Es ist also dasselbe wie `fitted( oz2.lm)` und kann z. B. ebenfalls für den Plot von y_i gegen $\hat{y}_i$ zusammen mit der Identität als „Soll-Linie“ verwendet werden, um die Güte der Modellanpassung qualitativ zu begutachten. (Nicht gezeigt.)





`> predict( oz2.lm, newdata =`
`+ data.frame( Temp = newtemp,`
`+ Solar.R = newrad),`
`+ se.fit = TRUE)`
`$fit`
`1 2 3`
`23.01854 23.25397 24.48627`
`$se.fit`
`1 2 3`
`7.638859 5.217611 4.887100`
`$df`
`[1] 106`
`$residual.scale`
`[1] 22.06325`
	Das Argument `se.fit = TRUE` veranlasst `predict()` zur zusätzlichen Berechnung der geschätzten Standardabweichungen (Engl.: "standard errors") der Schätzwerte der Regressionsfunktion. Das Resultat von `predict()` ist jetzt eine Liste mit vier Komponenten. Die Komponente `fit` enthält den Vektor der Schätzwerte $\hat{y}(\tilde{\mathbf{x}}_j)$ und die Komponente `se.fit` den Vektor der dazugehörigen Standardabweichungen. D. h., das j-te Element in `se.fit` ist gleich $\hat{\sigma}\sqrt{\tilde{\mathbf{x}}_\mathbf{j}'(\mathbf{X}'\mathbf{X})^{-1}\tilde{\mathbf{x}}_\mathbf{j}}$.
	
	Die Komponente `residual.scale` enthält den Wert von $\hat{\sigma}$ und `df` die zugehörigen Residuenfreiheitsgrade $n-p$.

10.9.2 Punktweise Konfidenzintervalle für die Regressionsfunktion und für ihre Parameter

Das punktweise Konfidenzintervall zum Konfidenzniveau $1-\alpha$ für den Wert der Regressionsfunktion an der Stelle $\tilde{\mathbf{x}}$ lautet

$$\tilde{\mathbf{x}}'\hat{\boldsymbol{\beta}} \pm t_{n-p;1-\alpha/2}\,\hat{\sigma}\,\sqrt{\tilde{\mathbf{x}}'(\mathbf{X}'\mathbf{X})^{-1}\tilde{\mathbf{x}}} \qquad \text{mit } \hat{\sigma} = \sqrt{\frac{\text{RSS}}{n-p}}$$

Dabei ist $t_{n-p;1-\alpha/2}$ das $(1-\alpha/2)$-Quantil der t-Verteilung zu $n-p$ Freiheitsgraden.

Zur Erinnerung eine kurze Begründung (siehe 10.1): Wegen $\hat{\boldsymbol{\beta}} \sim \mathcal{N}_p\left(\boldsymbol{\beta}, \sigma^2(\mathbf{X}'\mathbf{X})^{-1}\right)$ gilt $\tilde{\mathbf{x}}'\hat{\boldsymbol{\beta}} \sim \mathcal{N}_1\left(\tilde{\mathbf{x}}'\boldsymbol{\beta}, \sigma^2\tilde{\mathbf{x}}'(\mathbf{X}'\mathbf{X})^{-1}\tilde{\mathbf{x}}\right)$, sodass zusammen mit der stochastischen Unabhängigkeit von $\hat{\boldsymbol{\beta}}$ und $\hat{\sigma}^2$ sowie $(n-p)\hat{\sigma}^2/\sigma^2 \sim \chi^2_{n-p}$ folgt:

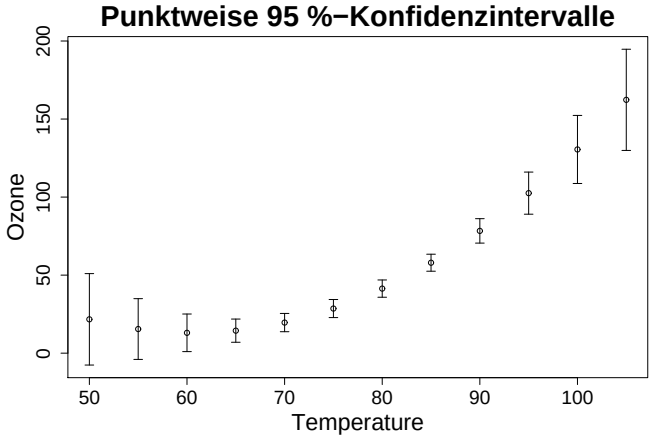
$$\frac{\tilde{\mathbf{x}}'\hat{\boldsymbol{\beta}} - \tilde{\mathbf{x}}'\boldsymbol{\beta}}{\hat{\sigma}\sqrt{\tilde{\mathbf{x}}'(\mathbf{X}'\mathbf{X})^{-1}\tilde{\mathbf{x}}}} \sim t_{n-p}$$

Zur Berechnung dieses Konfidenzintervalles ist `predict()` ebenfalls in der Lage. Eine grafische Darstellung ist allerdings nur für die Regression mit höchstens einer Covariablen möglich. Hierzu kann z. B. eine Funktion namens `errbar()` aus dem **R**-Paket `Hmisc` verwendet werden, wie im folgenden Beispiel gezeigt.

Punktweise Konfidenzintervalle für die Regressionsfunktion	

`> conf.level <- 0.95`
`> (oz1CIpw <- predict( oz1.lm,`
`+ newdata = data.frame(`
`+ Temp = newtemp2),`
`+ interval = "confidence",`
`+ level = conf.level))`
`fit lwr upr`
`1 21.6790 -7.6115 50.9696`
`2 15.4577 -4.0049 34.9203`
`.... .... .... ....`
`12 162.3107 129.8906 194.7307`
	`predict()` mit dem Argument `interval = "confidence"` bestimmt für die in `newdata` angegebenen Covariablenvektoren $\tilde{\mathbf{z}}_j$ die Schätzwerte $\hat{y}(\tilde{\mathbf{x}}_j)$ sowie die Ober- und Untergrenzen der zugehörigen, punktweisen Konfidenzintervalle zu dem durch `level` angegebenen Konfidenzniveau (Voreinstellung: 95 %). Resultat ist eine Matrix mit den suggestiven Spaltennamen `fit`, `lwr` und `upr`. (Ohne Wert für `newdata` erhielte man die gefitteten Werte $\hat{y}_i$ samt den Unter- und Obergrenzen ihrer zugehörigen Konfidenzintervalle.)

`> library( Hmisc)`
`> errbar( x = newtemp2,`
`+ y = oz1CIpw[, "fit"],`
`+ yplus = oz1CIpw[, "upr"],`
`+ yminus = oz1CIpw[, "lwr"],`
`+ xlab = "Temperature",`
`+ ylab = "Ozone",`
`+ main = paste( "Punktweise",`
`+ 100 * conf.level,`
`+ "%-Konfidenzintervalle"))`
	Einfügen des Paketes `Hmisc` in den Suchpfad. `errbar()` plottet an den in `x` angegebenen Stellen vertikale „Fehlerbalken“; hier also an den reellwertigen (!) Stellen $\tilde{z}_j$ in `newtemp2`. Die Werte in `y` (hier also die gefitteten Werte $\hat{y}(\tilde{\mathbf{x}}_j)$) werden markiert und die (absoluten) Unter- und Obergrenzen der vertikalen Balken werden von `yminus` bzw. `yplus` erwartet. Hier werden sie aus den Spalten der Matrix `oz1CIpw` genommen und liefern also punktweise Konfidenzintervalle (siehe folgendes Bild).



Mit dem j-ten p-dimensionalen Einheitsvektor $\tilde{\mathbf{x}}_j = (\overbrace{0,\ldots,0}^{j-1},1,0,\ldots,0)'$ für $j \in \{1,\ldots,p\}$ könnte mit obigem auch ein Konfidenzintervall für den Parameter β_{j-1} bestimmt werden. Dies erledigt allerdings etwas einfacher und suggestiver die Funktion `confint()`:

Punktweise Konfidenzintervalle für die Regressionsparameter	

`> confint( oz2.lm, level = 0.95)`
`  2.5 %  97.5 %`
`(Intercept)  9.496698e+01  5.765581e+02`
`Temp  -1.688436e+01 -4.193951e+00`
`I(Temp^2)  4.160941e-02  1.236857e-01`
`Solar.R  -3.334630e-02  3.859159e-01`
`I(Solar.R^2) -9.360809e-04  2.769364e-04`
	Punktweise Konfidenzintervalle für die Regressionsparameter eines `lm`-Objektes zum Konfidenzniveau `level` (Voreinstellung: 95 %).
	
	Beachte: Diese Konfidenzintervalle gelten *nicht* simultan, sondern einzeln!

10.9.3 Simultane Konfidenzintervalle für die Regressionsfunktion und simultane Konfidenzbereiche für ihren Parametervektor

Der beste lineare unverzerrte Schätzer der Regressionsfunktion an k Stellen $\tilde{\mathbf{x}}_1,\ldots,\tilde{\mathbf{x}}_k$ ist $\tilde{\mathbf{x}}_j'\hat{\boldsymbol{\beta}}$ für $j = 1,\ldots,k$ und zur Bestimmung von simultanen $(1-\alpha)$-Konfidenzintervallen für die Regressionsfunktionswerte $\tilde{\mathbf{x}}_1'\boldsymbol{\beta},\ldots,\tilde{\mathbf{x}}_k'\boldsymbol{\beta}$ gibt es (mindestens) zwei Methoden:

1. Die <u>Bonferroni-Methode</u>: Ihre simultanen Konfidenzintervalle lauten

$$\tilde{\mathbf{x}}_j'\hat{\boldsymbol{\beta}} \pm t_{n-p;1-\alpha/(2k)}\ \hat{\sigma}\ \sqrt{\tilde{\mathbf{x}}_\mathbf{j}'(\mathbf{X}'\mathbf{X})^{-1}\tilde{\mathbf{x}}_\mathbf{j}} \qquad \text{für } j = 1,\ldots,k.$$

   Beachte: Das $(1-\alpha/2)$-t-Quantil, wie es im Fall des punktweisen Konfidenzintervalles im vorherigen Paragraphen verwendet wird, ist hier durch das $(1-\alpha/(2k))$-t-Quantil ersetzt! Dies

kann in **R** realisiert werden, indem das punktweise Konfidenzniveau in `predict()` durch das Argument `level` „von Hand“ auf $1 - \alpha/k$ gesetzt wird.

2. Die Scheffé-Methode: Es sei d der Rang der $(p \times k)$-Matrix $(\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_k)$. D. h., d ist die Anzahl der *linear unabhängigen* Vektoren unter $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_k$ und es ist zwangsläufig $d \leq \min\{p, k\}$. Die simultanen Konfidenzintervalle lauten dann

$$\tilde{\mathbf{x}}_j'\hat{\boldsymbol{\beta}} \pm \sqrt{dF_{d,n-p;1-\alpha}}\ \hat{\sigma}\ \sqrt{\tilde{\mathbf{x}}_\mathbf{j}'(\mathbf{X}'\mathbf{X})^{-\mathbf{1}}\tilde{\mathbf{x}}_\mathbf{j}} \qquad \text{für } j = 1, \ldots, k.$$

Beachte: Hat man $k \geq p$ Designstellen, von denen p linear unabhängig sind, so ist $d = p$ und es spielt keine Rolle mehr, wie viele und welche Designstellen zusätzlich berücksichtigt werden. Insgesamt können es also auch (überabzählbar!) unendlich viele sein, sodass man z. B. gleich ein (stetiges) Konfidenz*band* für die Regressionsfunktion bestimmen kann, wie im nächsten §10.9.4 diskutiert.

Die Scheffé-Methode scheint in “base **R**” bisher *nicht* implementiert zu sein.

Bemerkungen:

- Mit beiden Methoden kann für $k = p$ und $\tilde{\mathbf{x}}_j = (\overbrace{0, \ldots, 0}^{j-1}, 1, 0, \ldots, 0)'$, dem j-ten p-dimensionalen Einheitsvektor für $j = 1, \ldots, p$, jeweils ein simultaner *Konfidenzbereich* für den (ganzen) Parameter*vektor* $\boldsymbol{\beta}$ angegeben werden. (Durch eine Auswahl von $k < p$ Einheitsvektoren geht dies auch für die entsprechenden Teile von $\boldsymbol{\beta}$. Dann ist zwangsläufig $d = k$.)

- Das Paket `ellipse` stellt durch seine Funktion `ellipse.lm()` eine sehr einfache Möglichkeit zur Verfügung, (zweidimensionale) Konfidenzellipsen für jede zweielementige Auswahl $(\beta_{j_1}, \beta_{j_2})$ mit $0 \leq j_1 \neq j_2 \leq p - 1$ aus $\boldsymbol{\beta}$ grafisch darstellen zu lassen.

- Die Intervall-Längen der Bonferroni-Methode und die der Scheffé-Methode sind unterschiedlich. Sinnvollerweise wählt man diejenige Methode, die die kürzeren Intervalle liefert, was in jeder gegebenen Situation auf einen Vergleich von $t_{n-p;1-\alpha/(2k)}$ und $\sqrt{dF_{d,n-p;1-\alpha}}$ hinausläuft.

Exkurs: Wie an obigen Beispielen bereits gesehen, laufen viele Fragestellungen an das lineare Modell $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ mit $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_{p-1})' \in \mathbb{R}^p$ auf die – gleichzeitige – Betrachtung von mehreren Linearkombinationen der Komponenten von $\boldsymbol{\beta}$ hinaus. D. h., man ist für ein geeignet gewähltes

$$\mathbf{A}^* := \begin{pmatrix} \mathbf{a}_1' \\ \vdots \\ \mathbf{a}_k' \end{pmatrix} \qquad \text{mit } \mathbf{a}_j \in \mathbb{R}^p \text{ für } j = 1, \ldots, k$$

an Eigenschaften von $\mathbf{A}^*\boldsymbol{\beta}$ interessiert. Offenbar ist $\mathbf{A}^*\boldsymbol{\beta}$ jedoch unbekannt, weswegen es durch $\mathbf{A}^*\hat{\boldsymbol{\beta}}$ und entsprechende Konfidenzbereiche zu schätzen ist. Dafür sind Verteilungseigenschaften von $\mathbf{A}^*\hat{\boldsymbol{\beta}}$ wichtig. Um diese näher zu bestimmen, seien o. B. d. A. $\mathbf{a}_1, \ldots, \mathbf{a}_r$ die linear unabhängigen Zeilen von $\mathbf{A}^*$ und der Rest, $\mathbf{a}_{r+1}, \ldots, \mathbf{a}_k$, sei linear abhängig von $\mathbf{a}_1, \ldots, \mathbf{a}_r$. Dann ist $r = \text{Rang}(\mathbf{A}^*) \leq \min\{k, p\}$. Für

$$\mathbf{A} := \begin{pmatrix} \mathbf{a}_1' \\ \vdots \\ \mathbf{a}_r' \end{pmatrix}$$

folgt dann (wie auch bereits in Abschnitt 10.1 berichtet) aus der Verteilungstheorie für quadratische Formen normalverteilter Vektoren mit $\hat{\sigma}^2 = \text{RSS}/(n-p)$:

$$T_\mathbf{A}(\mathbf{Y}) := \frac{\frac{1}{r}(\mathbf{A}\hat{\boldsymbol{\beta}} - \mathbf{A}\boldsymbol{\beta})' \left[\mathbf{A}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{A}'\right]^{-1} (\mathbf{A}\hat{\boldsymbol{\beta}} - \mathbf{A}\boldsymbol{\beta})}{\hat{\sigma}^2} \sim F_{r,n-p}$$

Es gilt also $\mathbb{P}\left(T_{\mathbf{A}}(\mathbf{Y}) \leq F_{r,n-p;1-\alpha}\right) = 1-\alpha$, sodass für festes $\mathbf{A}$ wegen

$$\begin{aligned} & T_{\mathbf{A}}(\mathbf{Y}) \leq F_{r,n-p;1-\alpha} \\ \iff & \mathbf{A}\boldsymbol{\beta} \in \mathcal{C}_{\mathbf{A}}(\mathbf{Y}) := \left\{\mathbf{z} \in \mathbb{R}^r : (\mathbf{A}\hat{\boldsymbol{\beta}} - \mathbf{z})' \left[\mathbf{A}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{A}'\right]^{-1} (\mathbf{A}\hat{\boldsymbol{\beta}} - \mathbf{z}) \leq \hat{\sigma}^2 r F_{r,n-p;1-\alpha}\right\} \end{aligned}$$

die Menge $\mathcal{C}_{\mathbf{A}}(\mathbf{Y})$ ein $(1-\alpha)$-Konfidenzbereich für $\mathbf{A}\boldsymbol{\beta}$ ist: $\mathbb{P}\left(\mathbf{A}\boldsymbol{\beta} \in \mathcal{C}_{\mathbf{A}}(\mathbf{Y})\right) = 1-\alpha$.

Zwei verschiedene Konfidenzbereiche für $\boldsymbol{\beta}$ durch spezielle Wahlen von $\mathbf{A}^*$:

1. Für $\mathbf{A}^* = \mathbf{I}_p$ ist $\mathbf{A} = \mathbf{A}^*$ und $r = k = p$ sowie

$$\mathcal{C}_{\text{Ellipsoid}}(\mathbf{Y}) := \left\{\mathbf{b} \in \mathbb{R}^p : (\hat{\boldsymbol{\beta}} - \mathbf{b})'\mathbf{X}'\mathbf{X}(\hat{\boldsymbol{\beta}} - \mathbf{b}) \leq \hat{\sigma}^2 p F_{p,n-p;1-\alpha}\right\}$$

   ein p-dimensionales $(1-\alpha)$-Konfidenz-Ellipsoid für $\boldsymbol{\beta}$ (zentriert in $\hat{\boldsymbol{\beta}} \in \mathbb{R}^p$), d. h., $1-\alpha = \mathbb{P}\left(\boldsymbol{\beta} \in \mathcal{C}_{\text{Ellipsoid}}(\mathbf{Y})\right)$.

2. Für $\mathbf{A}^* = \mathbf{e}_j' \equiv (\underbrace{0,\ldots,0}_{j-1},1,0,\ldots,0)$ ist ist $\mathbf{A} = \mathbf{e}_j'$ und $r = k = 1$ sowie

$$\begin{aligned} \mathcal{C}_j(\mathbf{Y}) &:= \left\{b \in \mathbb{R} : (\hat{\beta}_{j-1} - b)' \left[(\mathbf{X}'\mathbf{X})^{-1}_{jj}\right]^{-1} (\hat{\beta}_{j-1} - b) \leq \hat{\sigma}^2 F_{1,n-p;1-\alpha}\right\} \\ &= \left\{b \in \mathbb{R} : (\hat{\beta}_{j-1} - b)^2 \leq (\mathbf{X}'\mathbf{X})^{-1}_{jj}\ \hat{\sigma}^2 F_{1,n-p;1-\alpha}\right\} \\ &= \left[\hat{\beta}_{j-1} \pm \sqrt{(\mathbf{X}'\mathbf{X})^{-1}_{jj}}\ \hat{\sigma}\ t_{n-p;1-\alpha/2}\right], \text{ denn } \sqrt{F_{1,n-p;1-\alpha}} = t_{n-p;1-\alpha/2} \end{aligned}$$

   Damit sind $\mathcal{C}_1(\mathbf{Y}),\ldots,\mathcal{C}_p(\mathbf{Y})$ „punktweise“, also komponentenweise $(1-\alpha)$-Konfidenzintervalle für $\beta_0,\ldots,\beta_{p-1}$. Adjustiert (erhöht) man ihr Konfidenzniveau auf $1-\alpha/p$, so ist gemäß Bonferroni ihr kartesisches Produkt

$$\mathcal{C}_{\text{Quader}}(\mathbf{Y}) := \mathsf{X}_{j=1}^{p} \left[\hat{\beta}_{j-1} \pm \sqrt{(\mathbf{X}'\mathbf{X})^{-1}_{jj}}\ \hat{\sigma}\ t_{n-p;1-\alpha/(2p)}\right]$$

   ein p-dimensionaler $(1-\alpha)$-Konfidenz-Quader für $\boldsymbol{\beta}$ (zentriert in $\hat{\boldsymbol{\beta}} \in \mathbb{R}^p$). (Beachte, dass beim t-Quantil das α durch p dividiert wird und dass $\mathbb{P}\left(\boldsymbol{\beta} \in \mathcal{C}_{\text{Quader}}(\mathbf{Y})\right) \geq 1-\alpha$ ist.)

Die Ungleichung $T_{\mathbf{A}}(\mathbf{Y}) \leq F_{r,n-p;1-\alpha}$ lässt sich noch weiter äquivalent umformen:

$$\begin{aligned} T_{\mathbf{A}}(\mathbf{Y}) \leq F_{r,n-p;1-\alpha} \quad &\iff \quad (\hat{\boldsymbol{\phi}} - \boldsymbol{\phi})'\mathbf{L}^{-1}(\hat{\boldsymbol{\phi}} - \boldsymbol{\phi}) \leq \hat{\sigma}^2 r F_{r,n-p;1-\alpha} \\ &\qquad \text{mit } \hat{\boldsymbol{\phi}} := \mathbf{A}\hat{\boldsymbol{\beta}},\ \boldsymbol{\phi} := \mathbf{A}\boldsymbol{\beta} \text{ und } \mathbf{L} := \mathbf{A}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{A}' \\ &\iff \quad \sup_{\mathbf{h}\neq\mathbf{0}} \frac{\left(\mathbf{h}'(\hat{\boldsymbol{\phi}} - \boldsymbol{\phi})\right)^2}{\mathbf{h}'\mathbf{L}^{-1}\mathbf{h}} \leq \hat{\sigma}^2 r F_{r,n-p;1-\alpha}, \\ &\qquad \text{weil } \sup_{\mathbf{h}\neq\mathbf{0}} \frac{(\mathbf{h}'\mathbf{b})^2}{\mathbf{h}'\mathbf{L}\mathbf{h}} = \mathbf{b}'\mathbf{L}^{-1}\mathbf{b}, \text{ falls } \mathbf{L} \text{ positiv definit} \\ &\iff \quad \left(\mathbf{h}'(\hat{\boldsymbol{\phi}} - \boldsymbol{\phi})\right)^2 \leq \mathbf{h}'\mathbf{L}^{-1}\mathbf{h}\ \hat{\sigma}^2 r F_{r,n-p;1-\alpha} \text{ für alle(!) } \mathbf{h} \in \mathbb{R}^p \\ &\iff \quad \left|\mathbf{h}'\mathbf{A}\hat{\boldsymbol{\beta}} - \mathbf{h}'\mathbf{A}\boldsymbol{\beta}\right| \leq \sqrt{\mathbf{h}'\mathbf{L}^{-1}\mathbf{h}}\ \hat{\sigma}\sqrt{r F_{r,n-p;1-\alpha}} \text{ für alle } \mathbf{h} \in \mathbb{R}^p \end{aligned}$$

Folgerungen:

1. $\mathbf{h}'\mathbf{A}\hat{\boldsymbol{\beta}} \pm \sqrt{\mathbf{h}'\mathbf{A}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{A}'\mathbf{h}}\ \hat{\sigma}\sqrt{rF_{r,n-p;1-\alpha}}$ ist ein $(1-\alpha)$-Konfidenzintervall für $\mathbf{h}'\mathbf{A}\boldsymbol{\beta}$.

2. Die Aussage in Punkt 1 gilt simultan für *alle* $\mathbf{h} \in \mathbb{R}^p$:

   (a) Also insbesondere für $\mathbf{h} \in \{\mathbf{e}_1, \ldots, \mathbf{e}_r\}$, sodass $\mathbf{a}_j'\boldsymbol{\beta}$ für $j = 1, \ldots, r$ erfasst sind. Die übrigen $\mathbf{a}_j'\boldsymbol{\beta}$ für $j = r+1, \ldots, k$ erhält man durch geeignete Wahlen von $\mathbf{h}$, da diese $\mathbf{a}_j$'s Linearkombinationen von $\{\mathbf{a}_1, \ldots, \mathbf{a}_r\}$ sind, also als $\mathbf{h}'\mathbf{A}$ darstellbar sind.

   (b) Für $\mathbf{A}^* = \mathbf{I}_p$ ist $\mathbf{A} = \mathbf{A}^*$ und $r = k = p$, sodass die Aussage in Punkt 1 ein $(1-\alpha)$-Konfidenz<u>band</u> für $\mathbf{h} \mapsto \mathbf{h}'\boldsymbol{\beta}$ liefert (und damit auch für $\mathbf{x} \mapsto \mathbf{x}'\boldsymbol{\beta}$ mit $\mathbf{x} \in \{1\} \times \mathbb{R}^{p-1}$).

(**Ende des Exkurses**)

Die unten beginnende Funktion `lm.confint()` berechnet punktweise und simultane Bonferroni- bzw. Scheffé-Konfidenzintervalle für Werte von $y(\mathbf{x}) = \mathbf{x}'\boldsymbol{\beta}$ entweder

(a) an allen „alten“ Designstellen $\mathbf{x}$, falls ihr Argument `newdata = NULL` bleibt, oder

(b) an allen durch `newdata` „gewünschten“ Designstellen $\mathbf{x}$

sowie für den Parametervektor $\boldsymbol{\beta}$ (zum Vergleich mit dem Resultat von `confint()`).

`lm.confint()` erwartet für ihr Argument `lmfit` ein `lm`-Objekt und im Fall (b) für ihr Argument `newdata` einen Data Frame, der die Covariablenvektoren zu den Designstellen enthält, an denen die Regressionsfunktion samt ihrer Konfidenzintervalle bestimmt werden soll. Die Variablen in `newdata` müssen dieselben Namen haben wie diejenigen, welche bei der Erstellung des `lm`-Objektes verwendet wurden. Mit dem optionalen Argument `coverage` kann das Konfidenzniveau bestimmt werden; Voreinstellung für `coverage` ist 95 %.

Das Resultat ist eine Liste mit zwei Komponenten, die selbst wieder Listen sind:

i. Die erste Komponente namens `FitCI` enthält im Fall (a) die gefitteten Werte an allen „alten“ Designstellen und im Fall (b) an allen „gewünschten“ Designstellen mit den punktweisen und den simultanen Bonferroni- bzw. Scheffé-Konfidenzintervallen für die Regressionsfunktion. Außerdem die „Quantilfaktoren“ der punktweisen, der Bonferroni- und der Scheffé-Methode sowie die Freiheitsgrade des Scheffé-Quantils und das Konfidenzniveau.

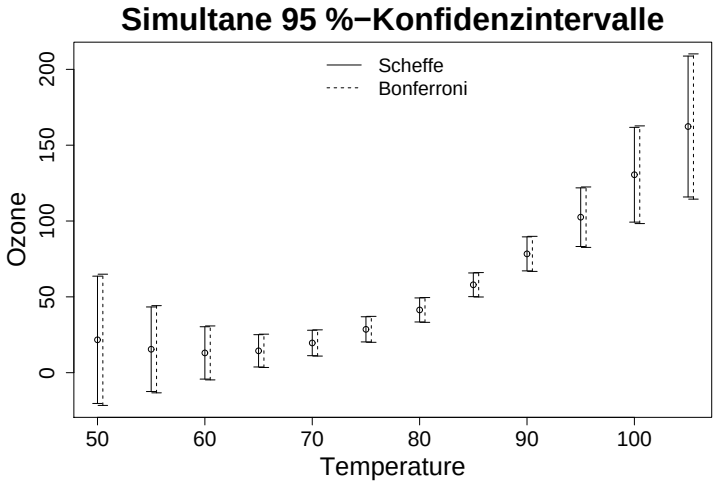
ii. Die zweite Liste namens `CoefCI` enthält (in beiden Fällen (a) und (b)) den gefitteten Koeffizientenvektor mit den punktweisen und den simultanen Bonferroni- bzw. Scheffé-Konfidenzintervallen für alle seine Komponenten; außerdem die „Quantilfaktoren“ der punktweisen, der Bonferroni- und der Scheffé-Methode sowie die Freiheitsgrade des Scheffé-Quantils und das Konfidenzniveau.

```
lm.confint <- function( lmfit, newdata = NULL, coverage = 0.95) {
  # 1. Gefittete Werte + punktweise & simultane Konfidenzintervalle ....
  #*********************************************************************
  if( !is.null( newdata)) {   # .... entweder an "gewuenschten" Designstellen:
    k <- nrow( newdata)

    # Berechnung des Rangs der zu newdata passenden neuen Designmatrix X:
    # (1) Der Rang ist "Abfallprodukt" der QR-Zerlegung von X: qr( X)$rank
    # (2) X wird passend zur "rechten Seite" der Modellformel des lm-Objektes fuer die
    #     "neuen" Designstellen bestimmt. Dazu wird erst ...
    #     a) die "rechte Seite" der Modellformel aus dem lm-Objekt extrahiert durch
    #        formula( delete.response( terms( lmfit))) und dann
    #     b) die neue Designmatrix X passend zu dieser rechten Seite der Modellformel
    #        unter Verwendung der Covariablen-Werte in newdata konstruiert durch
    #        model.matrix( "rechte Seite", data = newdata, xlev = lmfit$xlevels), wobei
```

```
  #        das Argument xlev fuer den (bisher noch nicht behandelten) Fall gebraucht
  #        wird, dass Faktorvariablen im Modell sind (vgl. Code von predict.lm()).
  #*****************************************************************************
  d <- qr( model.matrix( formula( delete.response( terms( lmfit))),
                         data = newdata, xlev = lmfit$xlevels))$rank

 } else {                      # .... oder an "alten" Designstellen:
  k <- length( fitted( lmfit))        # Stichprobenumfang
  d <- lmfit$rank                     # Rang der "alten" Designmatrix X
  }

df.res <- lmfit$df.residual         # Residuenfreiheitsgrade n - dim( beta)

PointwQuant <- qt( 1 - (1 - coverage)/2, df.res)
BonfQuant   <- qt( 1 - (1 - coverage)/(2 * k), df.res)
SchefQuant  <- sqrt( d * qf( coverage, d, df.res))
pred <- predict( lmfit, newdata = newdata, se.fit = TRUE)

PointwOffsets <- PointwQuant * pred$se.fit
BonfOffsets   <- BonfQuant   * pred$se.fit
SchefOffsets  <- SchefQuant  * pred$se.fit
FitCI <- list( CI = cbind( fit =        pred$fit,
                           pointw.lwr = pred$fit - PointwOffsets,
                           pointw.upr = pred$fit + PointwOffsets,
                           Bonf.lwr =   pred$fit - BonfOffsets,
                           Bonf.upr =   pred$fit + BonfOffsets,
                           Schef.lwr =  pred$fit - SchefOffsets,
                           Schef.upr =  pred$fit + SchefOffsets),
               Quant = c( Pointw = PointwQuant, Bonf = BonfQuant,
                          Schef  = SchefQuant),
               dof = c( num = d, den = df.res),
               coverage = coverage)

# 2. Koeffizienten + punktweise & simultane Konfidenzintervalle:
#*************************************************************
p <- length( beta <- coef( lmfit))
PointwQuant <- qt( 1 - (1 - coverage)/2, df.res)
BonfQuant   <- qt( 1 - (1 - coverage)/(2 * p), df.res)
SchefQuant  <- sqrt( p * qf( coverage, p, df.res))

se.coef <- coef( summary( lmfit, corr = FALSE))[ , "Std. Error"]
PointwOffsets <- PointwQuant * se.coef
BonfOffsets   <- BonfQuant   * se.coef
SchefOffsets  <- SchefQuant  * se.coef

CoefCI <- list( CI = rbind( coefficients = beta,
                            pointw.lwr =   beta - PointwOffsets,
                            pointw.upr =   beta + PointwOffsets,
                            Bonf.lwr =     beta - BonfOffsets,
                            Bonf.upr =     beta + BonfOffsets,
                            Schef.lwr =    beta - SchefOffsets,
                            Schef.upr =    beta + SchefOffsets),
                Quant = c( Pointw = PointwQuant, Bonf = BonfQuant,
                           Schef  = SchefQuant),
                dof = c( num = p, den = df.res),
                coverage = coverage)

return( list( FitCI = FitCI, CoefCI = CoefCI))
}
```

Beispiel: Berechnung sowohl simultaner Bonferroni- als auch simultaner Scheffé-Konfidenzintervalle mit `lm.confint()` für ausgewählte Werte der Regressionsfunktion von `oz1.lm` (siehe Bild unten):

```
> conf.level <- 0.95
> oz1CIsim <- lm.confint( oz1.lm, newdata = data.frame( Temp = newtemp2),
                          coverage = conf.level)
> attach( oz1CIsim$FitCI)
> errbar( x = newtemp2, y = CI[, "fit"], yplus = CI[, "Schef.upr"],
          yminus = CI[, "Schef.lwr"], xlab = "Temperature", ylab = "Ozone",
          main = paste( "Simultane", 100*conf.level, "%-Konfidenzintervalle"))
> errbar( x = newtemp2 + 0.5, y = CI[, "fit"], yplus = CI[, "Bonf.upr"],
          yminus = CI[, "Bonf.lwr"], add = TRUE, lty = 2)
> legend( x = "top", lty = 1:2, legend = c( "Scheffe", "Bonferroni"),
          cex = 1.5, bty = "n")
> detach( oz1CIsim$FitCI)
```



10.9.4 Ein Konfidenzband für die Regressionsfunktion

Aus der Scheffé-Methode erhält man, wie schon erwähnt, auch das (simultane!) $(1-\alpha)$-Konfidenzband für die gesamte Regressionsfunktion, indem *alle* Covariablenwerte (und damit alle Designstellen) betrachtet werden. Es ist dann $d = p$ und das Konfidenzband lautet

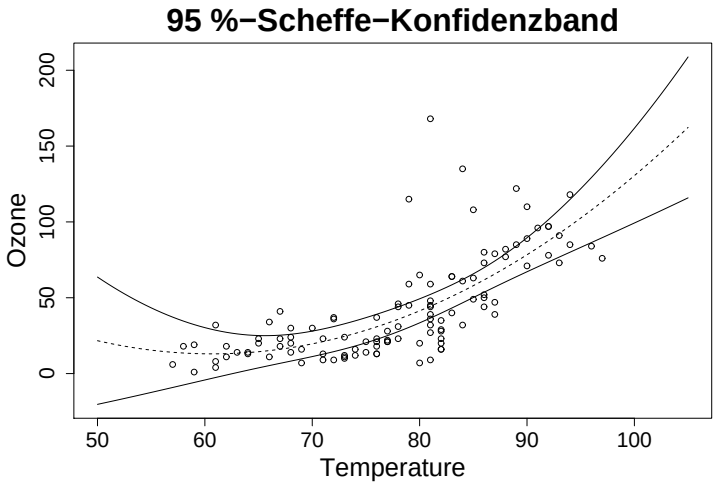
$$\mathbf{z} \longmapsto \mathbf{x}(\mathbf{z})'\hat{\boldsymbol{\beta}} \pm \sqrt{pF_{p,n-p;1-\alpha}}\,\hat{\sigma}\,\sqrt{\mathbf{x}(\mathbf{z})'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}(\mathbf{z})}$$

Konfidenzband für die Regressionsfunktion	

`> conf.level <- 0.95`
`> newtemp3 <- seq( 50, 105, by = 0.5)`
`> oz1CB <- lm.confint( oz1.lm, newdata =`
`+ data.frame( Temp = newtemp3),`
`+ coverage = conf.level)$FitCI`
	Wir bestimmen Regressionsschätzwerte und ihre simultanen 95 % Scheffé-Konfidenzintervalle für ein feines Gitter des Covariablenbereichs und damit das Konfidenzband. (Mit `coverage` ist ein anderes Konfidenzniveau wählbar; Voreinstellung: 95 %.)

`> with( oz1CB, matplot( newtemp3,`
`+ CI[, c( "fit", "Schef.upr", "Schef.lwr")],`
`+ type = "l", lty = c( 2,1,1), col = "black",`
`+ xlim = range( newtemp3, air$Temp),`
`+ ylim = range( CI[, c( "fit", "Schef.upr",`
`+ "Schef.lwr")], air$Ozone),`
`+ xlab = "Temperature", ylab = "Ozone",`
`+ main = paste( 100 * coverage,`
`+ "%-Scheffe-Konfidenzband"))`
`+ )`
	Zur grafischen Darstellung werden zunächst die gefittete Regressionsfunktion (gepunktet) sowie die Unter- bzw. Obergrenze des Konfidenzbandes (beide durchgezogen) geplottet. Zur Funktionsweise der sehr nützlichen Funktion `matplot()` und ihrer Argumente verweisen wir auf §7.5.6 (Seite 108) bzw. die Online-Hilfe.

`> points( air$Temp, air$Ozone)`	Schließlich werden mit `points()` die Originalbeobachtungen überlagert (siehe Grafik unten).



10.9.5 Punktweise und simultane Toleranzintervalle für zukünftige Response-Werte

Ein *zukünftiger* Response-Wert (Prognosewert) an der Stelle $\mathbf{x}^*$ enthält einen von „allem Bisherigen“ unabhängigen, neuen Fehler $\varepsilon^* \sim \mathcal{N}(0, \sigma^2)$ und lautet $Y^* = \mathbf{x}^{*\prime}\boldsymbol{\beta} + \varepsilon^*$. Für den Erwartungswert $\mathbf{x}^{*\prime}\boldsymbol{\beta}$ dieser zukünftigen Response ist $\mathbf{x}^{*\prime}\hat{\boldsymbol{\beta}}$ ein erwartungstreuer Schätzer. Offenbar ist bei Verwendung dieses Schätzers der Vorhersagefehler $\mathbf{x}^{*\prime}\hat{\boldsymbol{\beta}} - Y^* = \mathbf{x}^{*\prime}\hat{\boldsymbol{\beta}} - \mathbf{x}^{*\prime}\boldsymbol{\beta} - \varepsilon^* \sim \mathcal{N}(0, \sigma^2\mathbf{x}^{*\prime}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}^* + \sigma^2)$. Analog zum punktweisen Konfidenzintervall der Regressionsfunktion (in §10.9.2) ergibt sich also:

$$\frac{\mathbf{x}^{*\prime}\hat{\boldsymbol{\beta}} - Y^*}{\hat{\sigma}\sqrt{1 + \mathbf{x}^{*\prime}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}^*}} \sim t_{n-p}$$

Ein punktweises Toleranzintervall (auch Prognoseintervall genannt) zum Toleranzniveau $1 - \alpha$ für den zukünftigen Response-Wert hat daher die Gestalt

$$\mathbf{x}^{*\prime}\hat{\boldsymbol{\beta}} \pm t_{n-p;1-\alpha/2}\ \hat{\sigma}\ \sqrt{1 + \mathbf{x}^{*\prime}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}^*}$$

Beachte die 1 unter der Wurzel! Sie ist der Varianzbeitrag, der in der zukünftigen Response $Y^* = \mathbf{x}^{*\prime}\boldsymbol{\beta} + \varepsilon^*$ also auf das Konto der *neuen* Störung ε^* geht.

Bemerkungen:

- Soll ein punktweises Toleranzintervall für das *arithmetische Mittel* $\bar{Y}_m^*$ von m zukünftigen Responses $Y_1^*, \ldots, Y_m^*$ an *einer* Stelle $\mathbf{x}^*$ angegeben werde, so ist die 1 unter der Wurzel durch $1/m$ zu ersetzen. Dies ist *nicht* in “base **R**” implementiert.

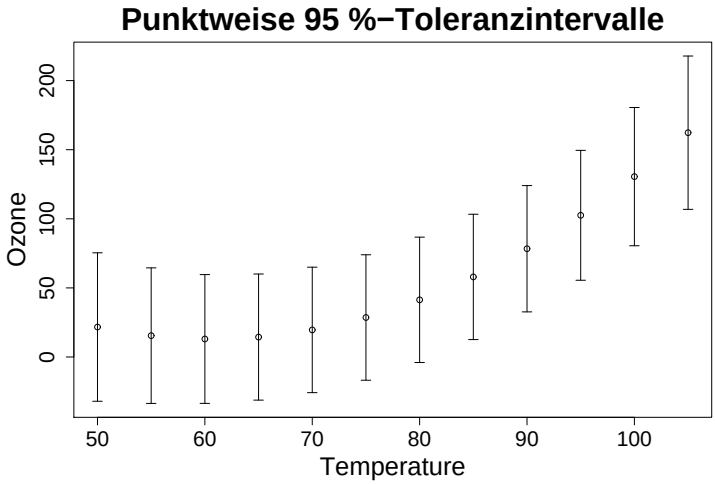
- Für simultane Toleranzintervalle für die zukünftigen Responses an k *verschiedenen* Stellen $\mathbf{x}_1^*, \ldots, \mathbf{x}_k^*$, von denen $d \leq \min\{k, p\}$ linear unabhängig sind, ist das Quantil $t_{n-p;1-\alpha/2}$ entweder durch $t_{n-p;1-\alpha/(2k)}$ (Bonferroni) oder durch $\sqrt{dF_{d,n-p;1-\alpha}}$ (Scheffé) zu ersetzen. Dies ist auch nicht in "base **R**" implementiert.

Punktweise Toleranzintervalle für zukünftige Response-Werte	

`> conf.level <- 0.95`
`> (oz1TIpw <- predict( oz1.lm,`
`+ newdata = data.frame( Temp =`
`+ newtemp2),`
`+ interval = "prediction",`
`+ level = conf.level))`
`fit lwr upr`
`1 21.6790 -32.0296 75.3876`
`2 15.4577 -33.5879 64.5033`
`.... .... .... ....`
`12 162.3107 106.8333 217.7880`
	`predict()` mit dem Argument `interval = "prediction"` bestimmt für die in `newdata` angegebenen Covariablenvektoren $\tilde{\mathbf{z}}_j^*$ die Prognosewerte $\hat{y}(\tilde{\mathbf{x}}_j^*)$ sowie die Ober- und Untergrenzen der zugehörigen, punktweisen Toleranzintervalle zu dem durch `level` angegebenen Toleranzniveau (Voreinstellung: 95 %). Resultat ist eine Matrix mit den suggestiven Spaltennamen `fit`, `lwr` und `upr`. (Ohne Wert für `newdata` erhielte man die gefitteten Werte $\hat{y}_i$ samt den Unter- und Obergrenzen ihrer zugehörigen Toleranzintervalle.)

`> library( Hmisc)`	Einfügen des Packages `Hmisc` in den Suchpfad.

`> errbar( x = newtemp2,`
`+ y = oz1TIpw[, "fit"],`
`+ yplus = oz1TIpw[, "upr"],`
`+ yminus = oz1TIpw[, "lwr"],`
`+ xlab = "Temperature",`
`+ ylab = "Ozone", main = paste(`
`+ "Punktweise", 100 * conf.level,`
`+ "%-Toleranzintervalle"))`
	Die grafische Darstellung funktioniert genauso wie für die punktweisen Konfidenzintervalle (siehe Seite 240 unten).
	
	Beachte in der Grafik unten die im Vergleich mit den Konfidenzintervallen auf Seite 240 oben erheblich größere und nahezu konstante Breite der Toleranzintervalle!



Die unten folgende Funktion `lm.tolint()` berechnet punktweise und simultane Bonferroni- bzw. Scheffé-Toleranzintervalle für zukünftige Response-Werte $Y^*(\mathbf{x}) = \mathbf{x}'\boldsymbol{\beta} + \varepsilon^*$ entweder

(a) an allen „alten" Designstellen $\mathbf{x}$, falls ihr Argument `newdata = NULL` bleibt, oder

(b) an allen durch `newdata` „gewünschten" Designstellen $\mathbf{x}$.

`lm.tolint()` erwartet für ihr Argument `lmfit` ein `lm`-Objekt und im Fall (b) für ihr Argument `newdata` einen Data Frame, der die Covariablenvektoren zu den Designstellen enthält, an denen die Prognosewerte samt ihrer Toleranzintervalle bestimmt werden sollen. Die Variablen

in `newdata` müssen dieselben Namen haben wie diejenigen, welche bei der Erstellung des `lm`-Objektes verwendet wurden. Mit dem optionalen Argument `coverage` kann das Toleranzniveau bestimmt werden; Voreinstellung ist 95 %.

Als Resultat wird eine Liste geliefert: Sie enthält im Fall (a) die geschätzten Prognosewerte an allen „alten“ Designstellen und im Fall (b) an allen „gewünschten“ Designstellen mit ihren punktweisen und ihren simultanen Bonferroni- bzw. Scheffé-Toleranzintervallen. Außerdem die „Quantilfaktoren“ der punktweisen, der Bonferroni- und der Scheffé-Methode sowie die Freiheitsgrade des Scheffé-Quantils und das Toleranzniveau:

```
lm.tolint <- function( lmfit, newdata = NULL, coverage = 0.95) {
  # Prognose-Werte + punktweise & simultane Toleranzintervalle ....
  #*****************************************************************
  if( !is.null( newdata)) {   # .... entweder an "gewuenschten" Designstellen:
    k <- nrow( newdata)
    d <- qr( model.matrix( formula( delete.response( terms( lmfit))),
                           data = newdata, xlev = lmfit$xlevels))$rank
                  # Berechnung des Rangs der zu newdata passenden
                  # neuen Designmatrix X genau wie in lm.confint().
   } else {                      # .... oder an "alten" Designstellen:
    k <- length( fitted( lmfit))        # Stichprobenumfang
    d <- lmfit$rank                     # Rang der "alten" Designmatrix X
   }

  df.res <- lmfit$df.residual         # Residuenfreiheitsgrade n - dim( beta)

  PointwQuant <- qt( 1 - (1 - coverage)/2, df.res)
  BonfQuant   <- qt( 1 - (1 - coverage)/(2 * k), df.res)
  SchefQuant  <- sqrt( d * qf( coverage, d, df.res))

  pred <- predict( lmfit, newdata = newdata, se.fit = TRUE)
  se.prog <- sqrt( pred$se.fit^2 + summary( lmfit)$sigma^2)

  PointwOffsets <- PointwQuant * se.prog
  BonfOffsets   <- BonfQuant   * se.prog
  SchefOffsets  <- SchefQuant  * se.prog

  ProgTI <- list( TI = cbind( fit =        pred$fit,
                              pointw.lwr = pred$fit - PointwOffsets,
                              pointw.upr = pred$fit + PointwOffsets,
                              Bonf.lwr =   pred$fit - BonfOffsets,
                              Bonf.upr =   pred$fit + BonfOffsets,
                              Schef.lwr =  pred$fit - SchefOffsets,
                              Schef.upr =  pred$fit + SchefOffsets),
                  Quant = c( Pointw = PointwQuant, Bonf = BonfQuant,
                             Schef  = SchefQuant),
                  dof = c( num = d, den = df.res),
                  coverage = coverage)
  return( ProgTI)
  }
```

(Für simultane Toleranzintervalle ist kein Bild gezeigt.)

10.9.6 Die Formeln im Spezialfall der einfachen linearen Regression

Obige Resultate für lineare Regressionsmodelle werden im Folgenden anhand des einfachen linearen Regressionsmodells

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \qquad \text{für } i = 1, \ldots, n$$

mit unabhängig und identisch $\mathcal{N}(0, \sigma^2)$-verteilten (i.i.d. $\sim \mathcal{N}(0, \sigma^2)$) Fehlern $\varepsilon_1, \ldots, \varepsilon_n$ und unbekannter Varianz σ^2 detaillierter aufgeführt. Hier ist die Dimension p des Modells also 2 und zur Abkürzung sei $y(x) := \beta_0 + \beta_1 x$.

10.9.6.1 Konfidenzintervalle für die Parameter der Regressionsgeraden

Konfidenzintervall zum Niveau $1 - \alpha$ ...

$$\ldots \text{ für } \beta_0: \qquad \hat{\beta}_0 \pm t_{n-2;1-\alpha/2}\, \hat{\sigma} \sqrt{\frac{\sum_{i=1}^n x_i^2}{n \sum_{i=1}^n (x_i - \bar{x}_n)^2}}$$

$$\ldots \text{ für } \beta_1: \qquad \hat{\beta}_1 \pm t_{n-2;1-\alpha/2}\, \hat{\sigma} \frac{1}{\sqrt{\sum_{i=1}^n (x_i - \bar{x}_n)^2}}$$

Bemerkung: Die angebenen Konfidenzintervalle halten das Niveau $1 - \alpha$ für jeden der beiden Parameter jeweils einzeln zwar exakt ein, aber die Komponenten des Vektors $\boldsymbol{\beta}' = (\beta_0, \beta_1)$ werden zum Niveau $1 - \alpha$ *nicht gleichzeitig* überdeckt! Um einen Konfidenzbereich für den Parametervektor als Ganzes anzugeben, muss eine der folgenden Methoden gewählt werden:

(Simultaner) Konfidenzbereich zum Niveau $1 - \alpha$ für (β_0, β_1) nach der ...

... Bonferroni-Methode:

$$\begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}: \quad \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{pmatrix} \pm t_{n-2;1-\alpha/4} \frac{\hat{\sigma}}{\sqrt{\sum_{i=1}^n (x_i - \bar{x}_n)^2}} \begin{pmatrix} \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \\ 1 \end{pmatrix}$$

... Scheffé-Methode:

$$\begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}: \quad \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{pmatrix} \pm \sqrt{2F_{2,n-2;1-\alpha}} \frac{\hat{\sigma}}{\sqrt{\sum_{i=1}^n (x_i - \bar{x}_n)^2}} \begin{pmatrix} \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \\ 1 \end{pmatrix}$$

Bemerkungen:

- Beachte den Unterschied zwischen den t-Quantilen der Bonferroni-Methode und denen der „nicht-simultanen“ Konfidenzintervalle: In ersteren steht $\alpha/4$ anstelle von $\alpha/2$ in letzteren.
- Die Intervall-Längen der Bonferroni-Methode und die der Scheffé-Methode sind, wie gesagt, unterschiedlich. Im vorliegenden Fall läuft die Entscheidung, welchen der Konfidenzbereiche man wählt, auf den Vergleich der Quantile $t_{n-2;1-\alpha/4}$ und $\sqrt{2F_{2,n-2;1-\alpha}}$ hinaus. (Memo: Im Package `car` gibt es die Funktion `confidence.ellipse()`, die *zwei*dimensionale Konfidenz-Ellipsen zeichnet. Siehe die betreffende Online-Hilfe bzw. [33, Fox (1997)].)
- Wir verweisen hier nochmal auf das Paket `ellipse`, das durch seine Funktion `ellipse.lm()` eine sehr einfache Möglichkeit zur Verfügung stellt, Konfidenzellipsen für (β_0, β_1) grafisch darstellen zu lassen.

10.9.6.2 Konfidenzintervalle für die Regressionsgerade

Die gefittete Regressionsfunktion an der Stelle x stellt gemäß Definition einen Schätzer für den Erwartungswert der Response zur „Dosis“ x dar. Bei der Angabe eines Konfidenzintervalles für diese Werte muss unterschieden werden, ob es sich um

- das Konfidenzintervall für den Response-Erwartungswert an *einem* x-Wert,
- k simultane Konfidenzintervalle für die Response-Erwartungswerte an k *verschiedenen* Werten $\tilde{x}_1, \ldots, \tilde{x}_k$
- oder ein Konfidenzband für *die Regressionsfunktion als Ganzes*

handeln soll. Hier die Formeln für jede der drei obigen Situationen:

Punktweises Konfidenzintervall zum Niveau $1-\alpha$ für $y(x) = \beta_0 + \beta_1\, x$:

$$\hat{\beta}_0 + \hat{\beta}_1\, x \;\pm t_{n-2;1-\alpha/2}\; \hat{\sigma}\, \sqrt{\frac{1}{n} + \frac{(x-\bar{x}_n)^2}{\sum_{i=1}^n (x_i - \bar{x}_n)^2}}$$

Bemerkung: Das Konfidenzintervall ist umso breiter, je weiter x von $\bar{x}_n$ entfernt ist.

Simultane Konfidenzintervalle zum Niveau $1-\alpha$ für $y(\tilde{x}_1), \ldots, y(\tilde{x}_k)$ gemäß der Bonferroni-Methode:

$$\hat{\beta}_0 + \hat{\beta}_1\, \tilde{x}_j \;\pm t_{n-2;1-\alpha/(2k)}\; \hat{\sigma}\, \sqrt{\frac{1}{n} + \frac{(\tilde{x}_j-\bar{x}_n)^2}{\sum_{i=1}^n (x_i - \bar{x}_n)^2}} \quad \text{für } j = 1, \ldots, k$$

Bemerkung: Die simultanen Konfidenzintervalle sind jeweils umso breiter, je weiter $\tilde{x}_j$ von $\bar{x}_n$ weg ist.

Konfidenzband zum Niveau $1-\alpha$ für die Regressionsgerade $x \longmapsto y(x)$:

$$x \longmapsto \hat{\beta}_0 + \hat{\beta}_1\, x \;\pm \sqrt{2F_{2,n-2;1-\alpha}}\; \hat{\sigma}\, \sqrt{\frac{1}{n} + \frac{(x-\bar{x}_n)^2}{\sum_{i=1}^n (x_i - \bar{x}_n)^2}}$$

Bemerkung: Die Breite des Bandes nimmt zu, je weiter sich x von $\bar{x}_n$ entfernt.

10.9.6.3 Toleranzintervalle zukünftiger Response-Werte

Die gefittete Regressionsfunktion an der Stelle x^* stellt auch einen Schätzer für den Erwartungswert einer *zukünftigen* Response Y^* zur Dosis x^* dar. Es können auch Toleranzintervalle für diese Responses angegeben werden. Hierbei muss aber unterschieden werden, ob es sich um

- das Toleranzintervall für *eine* zukünftige Response Y^* an der Stelle x^*,
- das Toleranzintervall für das arithmetische Mittel $\bar{Y}_m^*$ von m zukünftigen Responses $Y_1^*, \ldots, Y_m^*$ an *ein und derselben* Stelle x^* oder
- simultane Toleranzintervalle für die zukünftigen Responses $Y_1^*, \ldots, Y_k^*$ an $k \geq 2$ *verschiedenen* Stellen $x_1^*, \ldots, x_k^*$

handelt. Es folgen die Formeln für die drei genannten Szenarien:

Annahme: Der Fehler ε^* in der zukünftigen Response $Y^* = \beta_0 + \beta_1 x^* + \varepsilon^*$ ist unabhängig von $\varepsilon_1, \ldots, \varepsilon_n$ und ebenfalls $\mathcal{N}(0, \sigma^2)$-verteilt mit demselben σ^2.

Punktweises Toleranzintervall zum Niveau $1 - \alpha$ für Y^* an einer Stelle x^*:

$$\hat{\beta}_0 + \hat{\beta}_1\, x^* \;\pm t_{n-2;1-\alpha/2}\, \hat{\sigma} \sqrt{1 + \frac{1}{n} + \frac{(x^* - \bar{x}_n)^2}{\sum_{i=1}^n (x_i - \bar{x}_n)^2}}$$

Bemerkungen:

- Toleranzintervalle sind immer länger als Konfidenzintervalle, was an dem von der Prognosevarianz σ^2 (also von der Varianz von ε^* in Y^*) stammenden Summanden 1 unter der Wurzel liegt.
- Falls der Stichprobenumfang n hinreichend groß ist und x^* innerhalb des ursprünglich beobachteten Designbereichs liegt, dominiert die 1 unter der Wurzel (also die Prognosevarianz) die beiden anderen Terme, sodass die Länge der Toleranzintervalle *wesentlich* größer als die Länge der Konfidenzintervalle ist.
- Durch eine Erhöhung des Stichprobenumfangs n kann die Toleranzintervall-Länge zwar reduziert werden, aber sie ist immer von der Ordnung $t_{n-2;1-\alpha/2}\, \sigma$.
- Die fälschliche Verwendung der Konfidenzintervalle anstelle von Toleranzintervallen für zukünftige Responses liefert offenbar eine nicht zutreffende Präzision für die Prognose.

Annahme: Die Fehler $\varepsilon_1^*, \ldots, \varepsilon_m^*$ in den zukünftigen Responses $Y_j^* = \beta_0 + \beta_1 x^* + \varepsilon_j^*$ für $j = 1, \ldots, m$ sind unabhängig von $\varepsilon_1, \ldots, \varepsilon_n$ und ebenfalls (untereinander) i.i.d. $\sim \mathcal{N}(0, \sigma^2)$ mit demselben σ^2.

Punktweises Toleranzintervall zum Niveau $1 - \alpha$ für das arithmetische Mittel $\bar{Y}_m^*$ von m zukünftigen Responses $Y_1^*, \ldots, Y_m^*$ an *einer* Stelle x^*:

$$\hat{\beta}_0 + \hat{\beta}_1\, x^* \;\pm t_{n-2;1-\alpha/2}\, \hat{\sigma} \sqrt{\frac{1}{m} + \frac{1}{n} + \frac{(x^* - \bar{x}_n)^2}{\sum_{i=1}^n (x_i - \bar{x}_n)^2}}$$

Bemerkung: Es treffen alle Bemerkungen von oben zu, allerdings mit $1/m$ anstelle der 1 unter der Wurzel. Die Toleranzintervalle sind also abhängig von m etwas kürzer.

Annahme: Die Fehler $\varepsilon_1^*, \ldots, \varepsilon_k^*$ in den zukünftigen Responses $Y_j^* = \beta_0 + \beta_1 x_j^* + \varepsilon_j^*$ für $j = 1, \ldots, k$ sind unabhängig von $\varepsilon_1, \ldots, \varepsilon_n$ und ebenfalls (untereinander) i.i.d. $\sim \mathcal{N}(0, \sigma^2)$ mit demselben σ^2.

Simultane Toleranzintervalle zum Niveau $1 - \alpha$ für $Y_1^*, \ldots, Y_k^*$ an $k \geq 2$ verschiedenen Stellen $x_1^*, \ldots, x_k^*$:

$$\hat{\beta}_0 + \hat{\beta}_1\, x_j^* \;\pm \sqrt{2F_{2,n-2;1-\alpha}}\, \hat{\sigma} \sqrt{1 + \frac{1}{n} + \frac{(x_j^* - \bar{x}_n)^2}{\sum_{i=1}^n (x_i - \bar{x}_n)^2}} \quad \text{für } j = 1, \ldots, k$$

Bemerkungen:

- Auch simultane Toleranzintervalle sind immer länger als simultane Konfidenzintervalle, was wieder an dem von der Prognosevarianz σ^2 stammenden Summanden 1 unter der Wurzel liegt.

- Falls der Stichprobenumfang n hinreichend groß ist und die x_j^* innerhalb der ursprünglich beobachteten Designwerte liegen, dominiert die 1 unter der Wurzel die beiden anderen Terme, sodass die Länge der simultanen Toleranzintervalle *wesentlich* größer als die Länge der simultanen Konfidenzintervalle ist.
- Durch eine Erhöhung des Stichprobenumfangs n können die Toleranzintervall-Längen zwar reduziert werden, aber sie sind immer von der Ordnung $\sqrt{2F_{2,n-2;1-\alpha}}\,\sigma$.
- Auch hier liefert die fälschliche Verwendung von Konfidenzintervallen anstelle von Toleranzintervallen für zukünftige Responses eine nicht zutreffende Präzision für die Prognosen.

10.10 Polynomiale Regression

Wie gesehen, haben einige Operatoren in der Formelsyntax (wie `*`, `-`, `:` , `/` und `^m`) zwar eine spezielle Bedeutung, wenn sie rechts von `~` auf der obersten Ebene einer Formel auftreten, aber die Variablen in einer Formel dürfen durch alle Funktionen transformiert werden, deren Resultat wieder als eine Variable interpretierbar ist. Dies trifft insbesondere auf alle mathematischen Funktionen wie `log()`, `sqrt()` etc. zu (wie z. B. bei der Anwendung linearisierender Transformationen auf Seite 213). Aber auch Funktionen, deren Ergebnis als *mehrere* Variablen aufgefasst werden können, sind zulässig. Ein Beispiel hierfür ist die Funktion `poly()`: Sie erzeugt Basen von Orthonormalpolynomen bis zu einem gewissen Grad.

Bemerkung: Die eingangs genannten Operatoren haben *innerhalb* eines Funktionsaufrufs ihre „arithmetische“ Bedeutung. (Siehe auch nochmal Abschnitt 10.4.) D. h., die aus zwei Variablen u und v abgeleitete Variable $x := \log(u + v)$ wird in **S** als `log( u + v)` formuliert. Sollen die Operatoren auf der obersten Formelebene ihre arithmetische Bedeutung haben, so sind die betroffenen Terme in die Funktion `I()` „einzupacken“. Damit also $x := u + v$ eine einzelne Covariable darstellt, ist in einer **S**-Formel der Ausdruck `I( u + v)` zu verwenden.

Beispiele:

- Das auf drei Covariablen x_1, x_2, x_3 basierende (synthetische) Modell

$$\mathbf{Y} = \begin{pmatrix} 1 & x_{11} & x_{11}^2 & \log(x_{12}) & x_{13} \cdot 1_{\{x_{13} \leq 7\}}) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n1}^2 & \log(x_{n2}) & x_{n3} \cdot 1_{\{x_{n3} \leq 7\}}) \end{pmatrix}_{n \times 5} \cdot \boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

  mit seinen fünf Modelltermen 1, x_1, x_1^2, $\log(x_2)$ und $x_3 \cdot 1_{\{x_3 \leq 7\}}$ lautet in **S**:

```
Y ~ x1 + I( x1^2) + log( x2) + I( x3 * ( x3 <= 7))
```

- Ein Modell, in dem eine einzelne Covariable x_1 in Termen orthogonalpolynomialer Form bis zum Grad k auftritt, à la

$$\mathbf{Y} = \begin{pmatrix} 1 & p_1(x_{11}) & \dots & p_k(x_{11}) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & p_1(x_{n1}) & \dots & p_k(x_{n1}) \end{pmatrix}_{n \times (k+1)} \cdot \boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

  wobei $p_1, \dots, p_k$ Ortho*normal*polynome mit $\text{Grad}(p_j) = j$ für $j = 1, \dots, k$ sind, schreibt sich in **S** als `Y ~ poly( x1, degree = k)`.

Wir konzentrieren uns auf die Funktion `poly()`: Es seien `x` ein `numeric`-Vektor der Länge n und `k` $\in \mathbb{N}$. In der (von uns ausschließlich verwendeten) Form `poly( x, degree = k)` wird zu dem Vektor `x` $\equiv \mathbf{x} = (x_1, \dots, x_n)'$ eine Basis von **Orthonormalpolynomen** $p_1, \dots, p_k$ bis zum maximalen Grad k erzeugt, sodass $\text{Grad}(p_j) = j$. Die Orthogonalität besteht hierbei bezüglich der Stützstellen $x_1, \dots, x_n$, was bedeutet, dass für alle $1 \leq s, t \leq k$ gilt:

$$\begin{aligned} (p_s(\mathbf{x}))' \, p_t(\mathbf{x}) &\equiv (p_s(x_1), \dots, p_s(x_n)) \begin{pmatrix} p_t(x_1) \\ \vdots \\ p_t(x_n) \end{pmatrix} \\ &= \begin{cases} 0 & , \text{ falls } s \neq t \\ \parallel p_s \parallel^2 := \sum_{i=1}^n p_s(x_i)^2 = 1, & \text{ falls } s = t \end{cases} \end{aligned}$$

Damit eine solche Basis existiert, muss $k \leq n$ sein.

Diese k Polynome werden von `poly( x, degree = k)` erzeugt und ein jedes an den Stellen $x_1, \ldots, x_n$ ausgewertet. Die Resultate werden spaltenweise zu einer $(n \times k)$-Matrix

$$P_{(k)}(\mathbf{x}) := \begin{pmatrix} p_1(x_1) & p_2(x_1) & \ldots & p_k(x_1) \\ p_1(x_2) & p_2(x_2) & \ldots & p_k(x_2) \\ \vdots & \vdots & & \vdots \\ p_1(x_n) & p_2(x_n) & \ldots & p_k(x_n) \end{pmatrix}_{n \times k} \equiv (p_1(\mathbf{x}), \ldots, p_k(\mathbf{x}))$$

zusammengefasst. Sie erfüllt offenbar $\left(P_{(k)}(\mathbf{x})\right)' P_{(k)}(\mathbf{x}) = I_{k \times k}$.

Beispiel: Grafische Veranschaulichung der ersten fünf Orthonormalpolynome zu zwei verschiedenen, 20-elementigen Vektoren: Es sei $n = 20$ und

- für $\mathbf{x} = (x_1, \ldots, x_n)'$ seien die x_i äquidistant mit $x_i := i$ für $i = 1, \ldots, n$.
- In $\mathbf{y} = (y_1, \ldots, y_n)'$ seien die y_i *nicht* äquidistant, sondern es gelte $y_i := 1.2^{x_i}$.

In **R** generieren wir diese Vektoren und die dazugehörigen Orthonormalpolynombasen durch

```
> x   <- 1:20;                      y   <- 1.2^x
> P5x <- poly( x, degree = 5);      P5y <- poly( y, degree = 5)
```

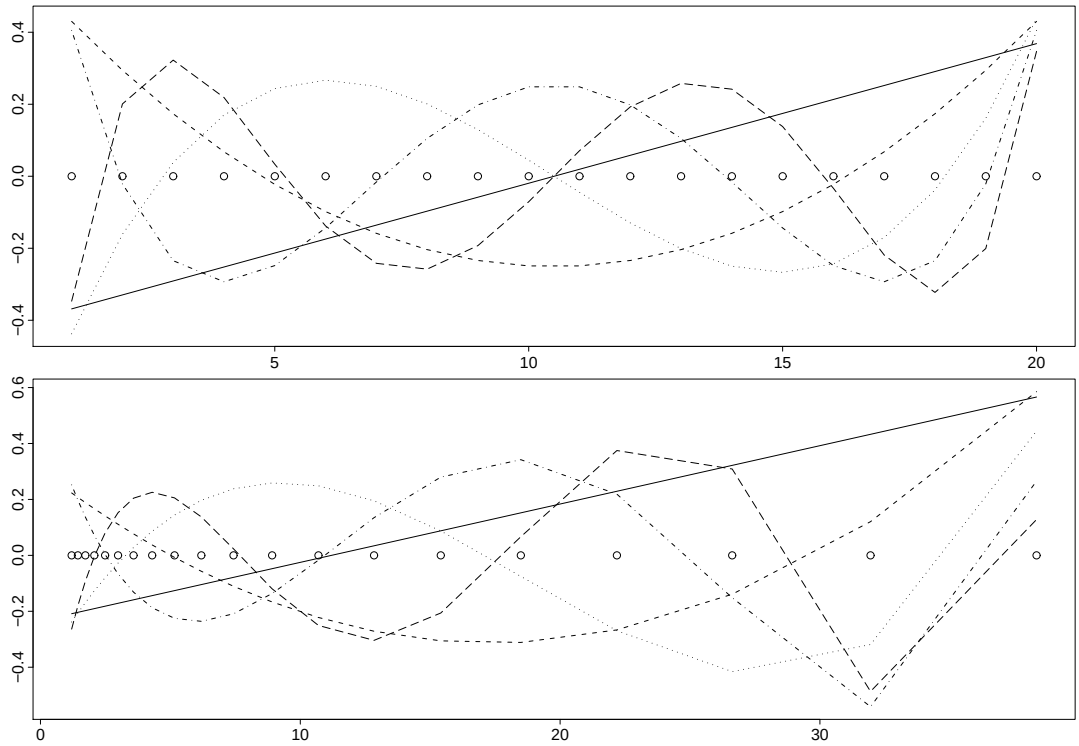
In `P5x` und `P5y` befinden sich nun die jeweiligen (20×5)-Matrizen (die mit einigen Attributen versehen sind, auf die wir hier aber nicht eingehen). Beispiel:

```
> P5x
                1           2           3           4           5
 [1,] -0.36839420  0.43019174 -0.43760939  0.40514757 -0.34694765
 [2,] -0.32961586  0.29434172 -0.16122451 -0.02132356  0.20086443
 [3,] -0.29083753  0.17358614  0.03838679 -0.23455912  0.32260045
....
[20,]  0.36839420  0.43019174  0.43760939  0.40514757  0.34694765
```

Dass die Spalten einer jeden der Matrizen orthogonal zueinander sind, zeigen wir hier jetzt auch nicht, aber eine grafische Darstellung, in der für jedes $j = 1, \ldots, k$ ein Polygonzug durch die Punkte $(x_i, p_j(x_i))$ geplottet wird, verdeutlicht, was hinter den obigen Zahlenkolonnen steckt. Die dazu verwendete Funktion `matplot()` (siehe auch §7.5.6) trägt jede Spalte einer n-zeiligen Matrix gegen jede Spalte einer anderen, ebenfalls n-zeiligen Matrix als Polygonzug ab, wobei jede Spaltenkombination einen eigenen Linientyp erhält. Jede der Matrizen kann auch ein n-elementiger Vektor sein. Zur Orientierung haben wir die x- bzw. y-Elemente als Punkte (auf der Nulllinie) eingetragen, um ihre äquidistante bzw. nicht äquidistante Lage abzubilden. (Siehe Plots auf der nächsten Seite oben.)

```
> matplot( x, P5x, type = "l", col = 1);   points( x, rep( 0, length( x)))
> matplot( y, P5y, type = "l", col = 1);   points( y, rep( 0, length( y)))
```

Wofür sind diese Orthonormalpolynombasen gut?
Sollen in einem linearen Modell Potenzen von metrisch-stetigen, also von `numeric`-Covariablen verwendet werden, wie z. B. geschehen im Ozon-Modell auf Seite 236 mit linearen sowie quadratischen Temperatur- und Strahlungstermen, so bekommt man es schnell mit einer „gefährlichen“ Eigenschaft der Monome 1, x, x^2, x^3 etc. zu tun:



Bei ihrer Verwendung werden die Spalten der Designmatrix, da sie Potenzen voneinander sind, sehr schnell stark korreliert, sprich „fast linear abhängig“ und die Designmatrix somit „fast-singulär“. Man spricht dann auch von (Multi-)Kollinearität (die natürlich auch bei Designmatrizen ohne polynomiale Modellterme auftreten kann). Diese kann sowohl zu numerischer Instabilität des Modellfits führen als auch sehr große Varianzen und Korrelationen für die Parameterschätzer liefern.

Bei der Verwendung der orthogonalen Polynome ist beides nicht der Fall. Dies kann man ganz deutlich an der Matrix der Korrelationen der Koeffizienten (`Correlation of Coefficients`) des Fits ablesen, in der die Korrelationen der Koeffizienten der orthogonalen Modellkomponenten Null sind, wie das folgende Beispiel dokumentiert.

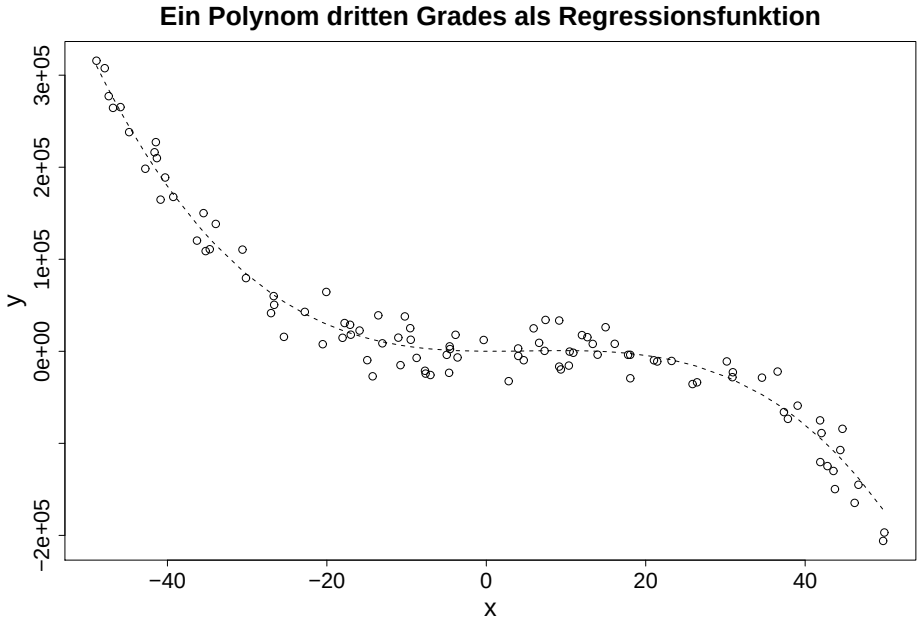
Anhand eines (künstlichen) **Beispiels** wird nun die Verwendung der Funktion `poly()` in einem linearen Modell erläutert. Dazu simulieren wir Realisierungen in dem Modell

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \varepsilon_i := 50 - 43x_i + 31x_i^2 - 2x_i^3 + \varepsilon_i, \tag{65}$$

indem wir erst die x_i als Design erzeugen, dann die Werte der Regressionsfunktion $m(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$ an diesen x_i berechnen und hieraus schließlich durch zufällige, additive „Störungen“ ε_i die Response-Werte y_i generieren:

```
> x <- runif( 100, -50, 50)
> m <- 50 - 43*x + 31*x^2 - 2*x^3
> y <- m + rnorm( length( m), sd = 20000)
```

Das zu diesen Daten gehörende Streudiagramm der (x_i, y_i), in dem die Regressionsfunktion $x \mapsto m(x)$ – eben ein Polynom dritten Grades – (fein) gepunktet eingezeichnet ist, findet sich auf der nächsten Seite oben.



Für den Fit eines orthogonalpolynomialen Modells dritten Grades gehen wir wie folgt vor:

```
> summary( xy.fit <- lm( y ~ poly( x, degree = 3)), corr = TRUE)
....
Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)              23041       1868   12.34   <2e-16 ***
poly(x, degree = 3)1   -907118      18675  -48.57   <2e-16 ***
poly(x, degree = 3)2    221733      18675   11.87   <2e-16 ***
poly(x, degree = 3)3   -403763      18675  -21.62   <2e-16 ***
....
Correlation of Coefficients:
                      (Intercept) poly(x, degree = 3)1 poly(x, degree = 3)2
poly(x, degree = 3)1 0.00
poly(x, degree = 3)2 0.00         0.00
poly(x, degree = 3)3 0.00         0.00                 0.00
```

Zur Interpretation der Resultate: Die in der `Coefficients`-Tabelle aufgeführten Terme `poly(x, degree = 3)1`, `poly(x, degree = 3)2` und `poly(x, degree = 3)3` sind die **S**-Bezeichnungen für die drei im Modell befindlichen Orthonormalpolynome p_1, p_2 und p_3, wobei die dem Term `poly(x, degree = 3)` angehängte Ziffer stets den Grad des jeweiligen Polynoms angibt.

Vorsicht: Die in der Spalte `Estimate` stehenden Koeffizientenwerte sind natürlich *nicht* die Schätzwerte für die Koeffizienten β_0, β_1, β_2 und β_3 unseres Ausgangsmodells (65)! Es handelt sich vielmehr um die Koeffizienten der drei, passend zu den Daten generierten Orthonormalpolynome p_1, p_2 und p_3 bzw. um den `(Intercept)`-Koeffizienten, der sich hierzu gehörend ergeben hat.

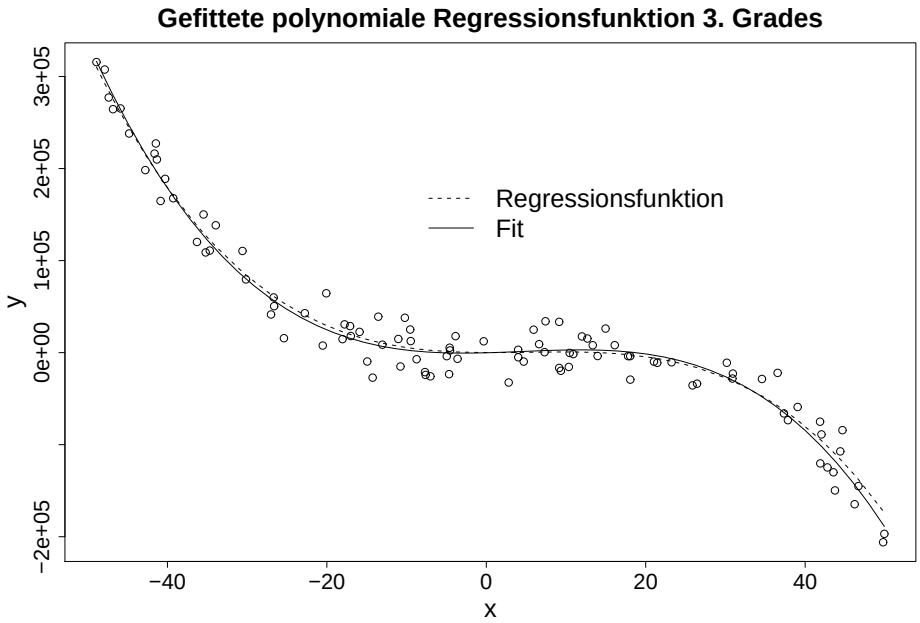
An der Tabelle `Correlation of Coefficients` ist deutlich zu erkennen, dass die Koeffizienten dieser orthogonalen Komponenten unkorreliert sind.

Ist man an den Schätzwerten für β_0, β_1, β_2 und β_3 interessiert, so müssen die für die Orthonormalpolynome erhaltenen Koeffizientenwerte also noch in eben jene umgeformt werden. Dies kann auf (mindestens) zwei Wegen geschehen: Zum einen, indem der Fit mit `poly( x, degree = 3, raw = TRUE)` anstelle von `poly( x, degree = 3)` wiederholt wird, sodass keine Orthogonalpolynome als Spalten der Designmatrix erzeugt werden, sondern (nicht-orthogonale) Monome. Zum anderen liefert die unten folgende (unkommentierte) Funktion `poly.transform()` mit der

verwendeten `poly()`-Funktion als erstem Argument und den Koeffizienten des Fits als zweitem Argument das Gewünschte:

```
> poly.transform <- function( polymat, coefs, intercept = TRUE) {
+  deg <- dim( polymat)[ 2]
+  deg1 <- deg + 1
+  if( !intercept) coefs <- c( 0, coefs)
+  if( length( coefs) != deg1) stop( "Wrong length for coefs!")
+  polycoefs <- attr( polymat, "coefs")
+  if( is.null( polycoefs)) stop( "Not enough information available!")
+
+  alpha <- polycoefs$alpha
+  norm2 <- polycoefs$norm2
+  polynoms <- vector( "list", deg1)
+  polynoms[[ 1]] <- c( 1, rep( 0, deg))
+
+  for( i in 1:deg) {
+    pi <- polynoms[[ i]]
+    temp <- c( 0, pi[ -deg1]) - alpha[ i] * pi
+    if( i > 1) temp <- temp - norm2[ i+1]/norm2[ i] * polynoms[[ i-1]]
+    polynoms[[ i+1]] <- temp
+    }
+
+  outmat <- coefs/c( 1, sqrt( norm2[ -1:-2])) * do.call( "rbind", polynoms)
+  ans <- drop( rep( 1, deg1) %*% outmat)
+  names( ans) <- paste( "x^", 0:deg, sep = "")
+  ans
+  }

> poly.transform( poly( x, 3), coef( xy.fit))
        x^0         x^1         x^2         x^3
-160.303658  227.714364   29.542089   -2.198264
```

Wie wir sehen, liegen die Koeffizientenschätzwerte teilweise recht gut, teilweise scheinen sie allerdings weit von ihren „Zielwerten“ entfernt zu sein. Aber ein Plot der gefitteten Regressionsfunktion zeigt, dass der Fit sehr gut ist, denn die Terme niedrigeren Grades sind gegenüber denen höheren Grades für größer werdendes x schnell vernachlässigbar:



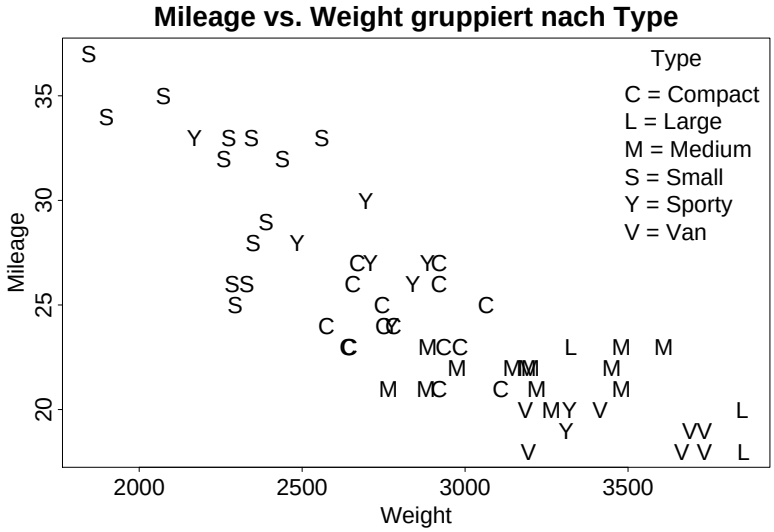
10.11 Faktorvariablen und Interaktionsterme im linearen Regressionsmodell

Zur Erinnerung: Variablen vom Typ `factor` bzw. `ordered factor` dienen in **R** der Repräsentierung von Daten des nominalen bzw. ordinalen Skalenniveaus. Wir sprechen zusammenfassend kurz von Faktoren. Sie können in Regressionsmodellen als Covariablen zur Gruppierung der Beobachtungen verwendet werden, um gewissermaßen „gruppenspezifische Submodelle“ zu fitten.

Anhand eines im **R**-Package `rpart` eingebauten Datensatzes namens `car.test.frame` sollen zunächst die Besonderheiten bei der Verwendung von **ungeordneten** Faktoren, also nominal skalierten Covariablen in Regressionsfunktionen erläutert werden. Hier zur Orientierung ein Ausschnitt aus `car.test.frame` und einige weitere Informationen:

```
> data( car.test.frame, package = "rpart");      car.test.frame
                Price Country Reliability Mileage  Type Weight Disp.  HP
Eagle Summit 4   8895     USA           4      33 Small   2560    97 113
Ford Escort   4  7402     USA           2      33 Small   2345   114  90
....             ....    ....        ....    ....  ....   ....  .... ....
Nissan Axxess 4 13949   Japan          NA      20   Van   3185   146 138
Nissan Van 4    14799   Japan          NA      19   Van   3690   146 106

> sapply( car.test.frame, class)
      Price     Country Reliability     Mileage        Type      Weight
  "integer"    "factor"   "integer"   "integer"    "factor"   "integer"
      Disp.          HP
  "integer"   "integer"

> table( car.test.frame$Type)
Compact   Large  Medium   Small  Sporty     Van
     15       3      13      13       9       7
```

Im vorliegenden Beispieldatensatz soll eine Regression der `Mileage` an `Weight` (metrisch) und `Type` (ungeordneter Faktor mit sechs Levels, also nominal mit sechs möglichen Ausprägungen) durchgeführt werden. In nebenstehendem Streudiagramm mit `Type`-spezifischen Symbolen wird deutlich, dass `Type` einen Einfluss auf die Beziehung zwischen `Mileage` und `Weight` haben könnte, also eine Interaktion zwischen `Type` und `Weight` vorläge.



Der Einbau von Faktoren in die Designformel eines Regressionsmodells geschieht genau wie bei metrischen (also `numeric`) Variablen. Allerdings ist zu beachten, dass die Levels eines Faktors *keine metrische* Bedeutung haben! D. h., selbst bei einer Codierung der Levels durch Zahlenwerte darf ein Faktor nicht wie eine metrisch skalierte Variable behandelt und einfach mit einem (multiplikativen) Koeffizienten versehen werden. Stattdessen wird für einen Faktor gleich eine ganze Gruppe von Koeffizienten ins Modell integriert, und zwar *für jedes seiner Levels ein eigener Koeffizient.*

Die folgenden Paragrafen 10.11.1 und 10.11.2 geben einen Überblick über drei Möglichkeiten, einen ungeordneten Faktor und eine metrische Variable als Designvariablen in einem Regressionsmodell zu kombinieren: Ohne Interaktion, d. h. mit rein additivem Effekt, und durch zwei mögliche Formen der Interaktion (faktoriell bzw. "crossed" oder hierarchisch bzw. "nested").

Die erste Methode (ohne Interaktion) liefert parallele Regressionsgeraden mit unterschiedlichen konstanten Termen. Die anderen beiden Methoden liefern für jedes Faktorlevel eine Regressionsgerade mit eigener Steigung und eigenem konstanten Term, allerdings in verschiedenen Parametrisierungen desselben Modells, die damit unterschiedliche Interpretationen der Beziehung zwischen den Designvariablen und der Response erlauben. Außerdem lassen sich mit diesen beiden Methoden auch Regressionsgeraden mit gleichem konstantem Term und unterschiedlichen Steigungen fitten.

Bemerkungen:

1. In vielen der folgenden Parametrisierungen sind die auftretenden Koeffizienten *nicht* eindeutig bestimmt (man sagt auch „nicht identifizierbar"), weswegen im Fall ihrer Schätzung besondere Vorkehrungen getroffen werden müssen, auf die wir aber erst in §10.11.4 eingehen.

2. Modelle mit einer Faktor- und einer metrischen Covariablen werden oft auch als ANCOVA-Modelle bezeichnet (ANCOVA = "analysis of covariance"). I. d. R. ist man dabei nur an der Wirkung des Faktors auf die Response interessiert; die metrische Covariable ist darin nur eine Störgröße, um deren Einfluss man den Faktoreinfluss „korrigiert" oder „adjustiert".

10.11.1 Ein ungeordneter Faktor und eine metrische Variable ohne Interaktion: „Parallele" Regression

Soll in unserem `car.test.frame`-Beispiel der Faktor `Type` ohne Interaktion mit der metrischen Covariablen `Weight` in das Modell eingehen, also durch seine Levels lediglich als ein additiver Effekt auf die `Mileage` wirken, so handelt es sich um das Modell

$$\texttt{Mileage}_{ij} = \beta_0 + \alpha_i + \beta_1\, \texttt{Weight}_{ij} + \varepsilon_{ij}, \quad j = 1, \ldots, n_i;\ i = 1, \ldots, I$$

Hier werden für die Covariablen ihre Haupteffekte gefittet: Für jedes Faktorlevel i von `Type` der Effekt α_i als Abweichung vom konstanten Term β_0 (hier: $i = 1, \ldots, I = 6$) und für `Weight` die für alle Faktorlevels „gemeinsame" Steigung β_1. (Insbesondere wird also *keine* Interaktion zwischen `Type` und `Weight` modelliert, da das Level von `Type` keinen Einfluss auf den Effekt, d. h. den Koeffizienten von `Weight` hat.)

In der **R**-Formelsyntax wird dies durch

```
Mileage ~ Type + Weight
```

erreicht, wobei diese Formel dasselbe wie `Mileage ~ 1 + Type + Weight` bedeutet, da der hier durch `1` explizit modellierte konstante Term gemäß Voreinstellung auch in das Modell eingebaut wird, wenn in dessen Formel die `1` fehlt.

10.11.2 Ein ungeordneter Faktor und eine metrische Variable mit Interaktionen

In Abschnitt 10.4 über die allgemeine Formelsyntax und anhand des Beispiels in Abschnitt 10.5 hatten wir das Konzept der Interaktion stetiger Designvariablen schon kennengelernt. Auch für die Kombination eines Faktors mit einer stetigen Designvariablen gibt es Interaktionen. Dabei wird für *jedes* Level des Faktors ein eigener Koeffizient für die stetige (!) Designvariable ermittelt. Die Parametrisierung dieses Modells kann auf zwei verschiedene Arten erfolgen, aber *hinsichtlich*

der Effekte der beteiligten Designvariablen auf die Response sind sie äquivalent, wenngleich sich auch die Notationen in der **R**-Formelsyntax unterscheiden.

10.11.2.1 Das faktorielle (= "crossed") Modell

Zusätzlich zu den Haupteffekten $\alpha_1, \ldots, \alpha_I$ für `Type` und β_1 für `Weight` werden Interaktionseffekte zwischen diesen beiden bestimmt: γ_i als Abweichung der `Weight`-Steigung von der „gemeinsamen" Steigung β_1 im Faktorlevel i von `Type`. Die mathematische Formulierung lautet

$$\texttt{Mileage}_{ij} = \beta_0 + \alpha_i + \beta_1\, \texttt{Weight}_{ij} + \gamma_i\, \texttt{Weight}_{ij} + \varepsilon_{ij}, \quad j = 1, \ldots, n_i;\ i = 1, \ldots, I$$

und die **R**-Formelsyntax:

```
Mileage ~ 1 + Type + Weight + Type:Weight
```

Auch hier kann die `1` weggelassen und der ganze Ausdruck durch `Mileage ~ Type * Weight` sogar noch weiter abgekürzt werden.

Die in diesem Modell naheliegende Zusammenfassung des konstanten Terms β_0 und der Faktor-Haupteffekte α_i zu der Parametrisierung

$$\texttt{Mileage}_{ij} = \alpha_i + \beta_1\, \texttt{Weight}_{ij} + \gamma_i\, \texttt{Weight}_{ij} + \varepsilon_{ij}, \quad j = 1, \ldots, n_i;\ i = 1, \ldots, I$$

wird in **R**-Formelsyntax durch

```
Mileage ~ -1 + Type + Weight + Type:Weight
```

oder kürzer durch `Mileage ~ Type * Weight - 1` erreicht. Dies ändert nicht die Effekte der beteiligten Designvariablen auf die Response, sondern nur die Parametrisierung des Modells.

10.11.2.2 Das hierarchische (= "nested") Modell

In diesem Modell werden die Haupteffekte α_i für den Faktor `Type` gefittet und „innerhalb" eines jeden Faktorlevels i eine spezifische `Weight`-Steigung. Mathematisch:

$$\texttt{Mileage}_{ij} = \beta_0 + \alpha_i + \beta_i\, \texttt{Weight}_{ij} + \varepsilon_{ij}, \quad j = 1, \ldots, n_i;\ i = 1, \ldots, I$$

In der **R**-Formelsyntax wird dies durch

```
Mileage ~ 1 + Type + Weight %in% Type
```

beschrieben, wobei auch wieder die `1` weggelassen werden kann und die Kurzform des Ausdrucks `Mileage ~ Type / Weight` lautet. Beachte dabei, dass der Formeloperator `/` *nicht* kommutativ ist (im Gegensatz zu `*`)! Beachte ferner, dass `%in%` als Formeloperator nichts mit der Funktion `%in%` zu tun hat, die die Abfrage „Element $\in$ Menge" implementiert (siehe die Online-Hilfe `?"%in%"`).

Auch hier sind konstanter Term β_0 und Faktor-Haupteffekte α_i zusammenfassbar zu

$$\texttt{Mileage}_{ij} = \alpha_i + \beta_i\, \texttt{Weight}_{ij} + \varepsilon_{ij}, \quad j = 1, \ldots, n_i;\ i = 1, \ldots, I,$$

was in **R**-Formelsyntax mit

```
Mileage ~ -1 + Type + Weight %in% Type
```

oder kürzer mit `Mileage ~ Type / Weight - 1` geschieht. Und wieder ändern sich nicht die Effekte der beteiligten Designvariablen auf die Response, sondern lediglich die Modellparametrisierung.

10.11.2.3 Modifikationen der beiden Interaktionsmodelle

Ein Modell aus Regressiongeraden mit gleichem konstantem Term und unterschiedlichen Steigungen ist sowohl im hierarchischen als auch im faktoriellen Modell erreichbar:

- Im faktoriellen Modell

$$\texttt{Mileage}_{ij} = \beta_0 + \beta_1\,\texttt{Weight}_{ij} + \gamma_i\,\texttt{Weight}_{ij} + \varepsilon_{ij}, \quad j = 1, \ldots, n_i;\ i = 1, \ldots, I$$

  durch `Mileage ~ 1 + Weight + Type:Weight`

  und kürzer durch `Mileage ~ Type * Weight - Type`

- Im hierarchischen Modell

$$\texttt{Mileage}_{ij} = \beta_0 + \beta_i\,\texttt{Weight}_{ij} + \varepsilon_{ij}, \quad j = 1, \ldots, n_i;\ i = 1, \ldots, I$$

  durch `Mileage ~ 1 + Weight %in% Type`

  und abgekürzt durch `Mileage ~ Type / Weight - Type`

10.11.3 Die Modelle im Überblick

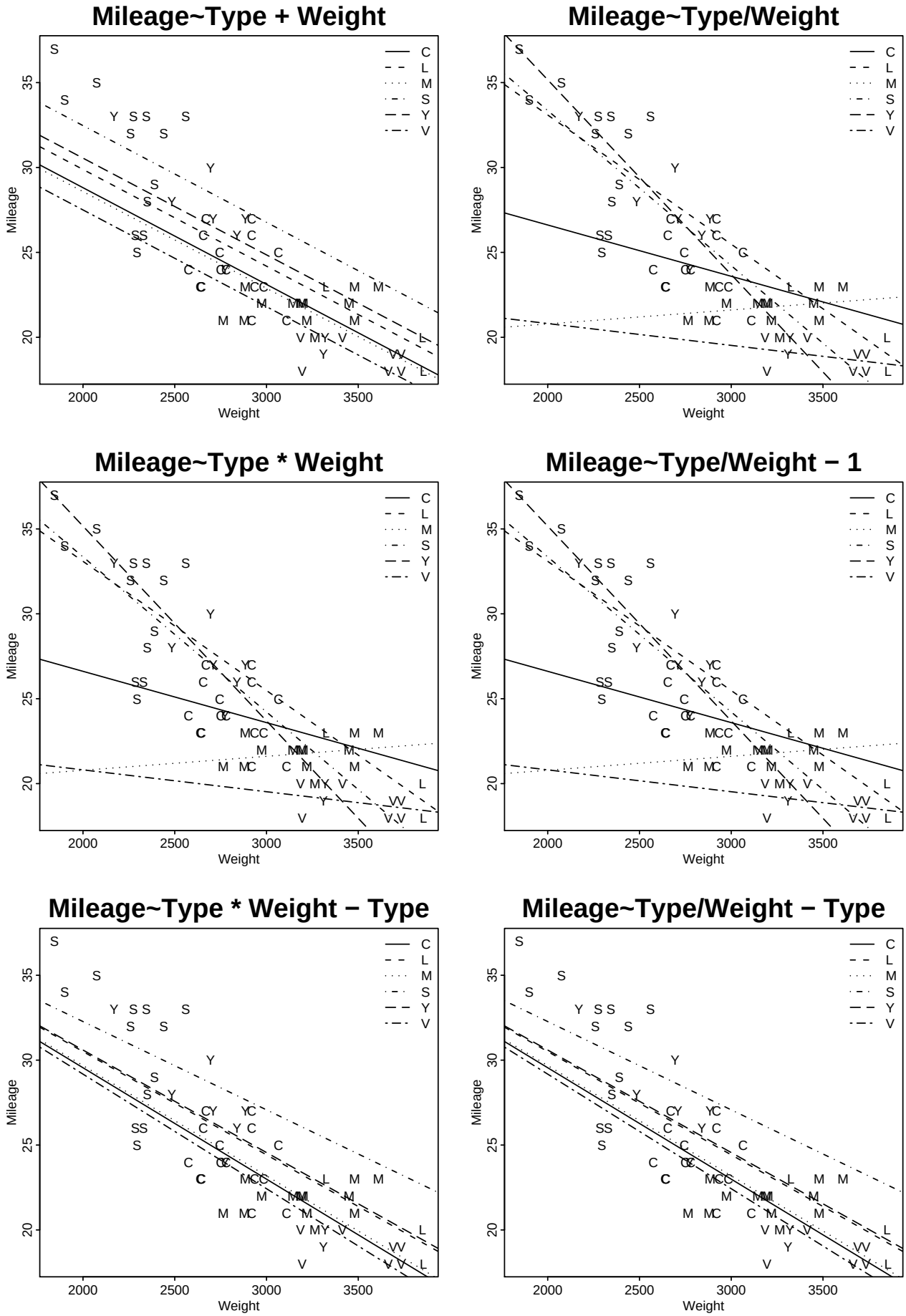
Es folgt eine tabellarische Zusammenfassung der mathematischen Parametrisierungen und der abgekürzten sowie „expandierten“ **R**-Formeln obiger Modelle. Auf der nächsten Seite befindet sich eine (entsprechend der Tabelle angeordnete) grafische Übersicht der gefitteten Regressionsgeraden.

Ohne Interaktion (parallele Geraden):
$\texttt{Mileage}_{ij} = \beta_0 + \alpha_i + \beta_1\,\texttt{Weight}_{ij} + \varepsilon_{ij}$
`Mileage ~ Type + Weight`
`Mileage ~ 1 + Type + Weight`
	Hierarchisch (levelspezifische Geraden):
	$\texttt{Mileage}_{ij} = \beta_0 + \alpha_i + \beta_i\,\texttt{Weight}_{ij} + \varepsilon_{ij}$
	`Mileage ~ Type / Weight`
	`Mileage ~ 1 + Type + Weight %in% Type`

Faktoriell (levelspezifische Geraden):
$\texttt{Mileage}_{ij} = \beta_0 + \alpha_i + \beta_1\,\texttt{Weight}_{ij} + \gamma_i\,\texttt{Weight}_{ij} + \varepsilon_{ij}$
`Mileage ~ Type * Weight`
`Mileage ~ 1 + Type + Weight`
`+ Type:Weight`
	Hierarchisch (levelspezifische Geraden):
	$\texttt{Mileage}_{ij} = \alpha_i + \beta_i\,\texttt{Weight}_{ij} + \varepsilon_{ij}$
	`Mileage ~ Type / Weight - 1`
	`Mileage ~ -1 + Type + Weight %in% Type`

Faktoriell (levelspezifische Steigungen):
$\texttt{Mileage}_{ij} = \beta_0 + \beta_1\,\texttt{Weight}_{ij} + \gamma_i\,\texttt{Weight}_{ij} + \varepsilon_{ij}$
`Mileage ~ Type * Weight - Type`
`Mileage ~ 1 + Weight + Type:Weight`
	Hierarchisch (levelspezifische Steigungen):
	$\texttt{Mileage}_{ij} = \beta_0 + \beta_i\,\texttt{Weight}_{ij} + \varepsilon_{ij}$
	`Mileage ~ Type / Weight - Type`
	`Mileage ~ 1 + Weight %in% Type`

Beachte: Die von **R** zunächst zurückgelieferten Koeffizienten eines Modellfits sind i. d. R. *nicht* diejenigen aus der mathematischen Parametrisierung. Näheres dazu in §10.11.4.


Mileage~Type + Weight
Mileage~Type/Weight
Mileage~Type * Weight
Mileage~Type/Weight − 1
Mileage~Type * Weight − Type
Mileage~Type/Weight − Type
Mileage
Weight
C
L
M
S
Y
V

10.11.4 Modellparametrisierung ungeordneter Faktoren durch Kontraste

Ein generelles Problem bei der Integration von Faktoren in ein Modell ist, dass sie normalerweise zu mehr Koeffizienten führen als in diesem Modell geschätzt werden können. Mit anderen Worten: Sie sind nicht „identifizierbar“, was auch *funktionale* Überparametrisierung genannt wird. Das folgende, einfache Beispiel eines linearen Modells, in dem als einzige Covariable ein Faktor auftritt (weswegen es sich faktisch um ein einfaktorielles-ANOVA-Modell handelt, worauf wir aber erst in Abschnitt 11.1 ausführlicher eingehen), soll das Problem klarmachen. Wir betrachten

$$Y_{ij} = \beta_0 + \alpha_i + \varepsilon_{ij}, \quad j = 1, \ldots, n_i; \quad i = 1, \ldots, I \quad \text{mit } n = \sum_{i=1}^{I} n_i, \tag{66}$$

wie z. B. `Mileage ~ Type` in **R**s Formelnotation. In Matrixformulierung lautet das

$$\mathbf{Y} = [\mathbf{1}_n | \mathbf{X}_a] \begin{pmatrix} \beta_0 \\ \boldsymbol{\alpha} \end{pmatrix} + \boldsymbol{\varepsilon},$$

wobei $\mathbf{1}_n$ der n-dimensionale Einsenvektor ist,

$$\mathbf{Y} = \begin{pmatrix} Y_{11} \\ \vdots \\ Y_{1n_1} \\ Y_{21} \\ \vdots \\ Y_{2n_2} \\ \vdots \\ \vdots \\ Y_{I1} \\ \vdots \\ Y_{In_I} \end{pmatrix}_{n\times 1}, \quad \mathbf{X}_a = \begin{pmatrix} 1 & 0 & \ldots & \ldots & 0 \\ \vdots & \vdots & & & \vdots \\ 1 & 0 & & & \\ 0 & 1 & & & \\ \vdots & \vdots & & & \\ & 1 & & & \\ & 0 & \ddots & & \vdots \\ & \vdots & & \ddots & 0 \\ & & & & 1 \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix}_{n\times I}, \quad \boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_I \end{pmatrix}_{I\times 1}$$

und $\boldsymbol{\varepsilon}$ analog zu $\mathbf{Y}$ indiziert ist.

Die $(n \times I)$-<u>Inzidenzmatrix</u> $\mathbf{X}_a$ hat offensichtlich den Rang I, aber die 1-Spalte zur Codierung des konstanten Terms β_0 in der Designmatrix $\mathbf{X} := [\mathbf{1}_n | \mathbf{X}_a]$ ist von den Spalten von $\mathbf{X}_a$ linear abhängig. In Konsequenz hat die $(n \times (I+1))$-Matrix $\mathbf{X}$ den Rang I, sodass die $((I+1) \times (I+1))$-Matrix $\mathbf{X}'\mathbf{X}$ singulär ist! D. h., das zu lösende Kleinste-Quadrate-Problem führt auf ein überbestimmtes Gleichungssystem hinaus, weil das Modell überparametrisiert ist bzw. die Parameter β_0, α_1, …, α_I nicht identifizierbar sind. (Zum Beispiel kann β_0 durch $\beta_0 + \delta$ mit einem beliebigen, konstanten δ ersetzt und dies durch $\alpha_i - \delta$ (für jedes $i = 1, \ldots, I$) kompensiert werden.)

Die Lösung dieses Problems ist eine geeignete Reparametrisierung des Modells: Ersetze $\boldsymbol{\alpha}$ durch $\mathbf{C}_a\boldsymbol{\alpha}^*$ mit einem <u>Kontraste</u> genannten Parametervektor $\boldsymbol{\alpha}^* = \left(\alpha_1^*, \ldots, \alpha_{I-1}^*\right)'$ und einer geeigneten $(I \times (I-1))$-<u>Kontrastmatrix</u> $\mathbf{C}_a$, sodass die $(n \times I)$-Matrix $\mathbf{X}^* := [\mathbf{1}_n | \mathbf{X}_a\mathbf{C}_a]$ den Rang I hat. Dies ist, wie man zeigen kann, z. B. erfüllt, falls $\text{Rang}([\mathbf{1}_I | \mathbf{C}_a]) = I$.

Man erhält dadurch ein „neues“ Modell

$$\mathbf{Y} = \mathbf{X}^* \begin{pmatrix} \beta_0 \\ \boldsymbol{\alpha}^* \end{pmatrix} + \boldsymbol{\varepsilon} \quad \text{mit invertierbarem } (\mathbf{X}^*)'\mathbf{X}^*$$

Bemerkung: Ist $\text{Rang}(\mathbf{C}_a) = I - 1$, so existiert zu $\mathbf{C}_a$ die eindeutig bestimmte Links-Inverse $\mathbf{C}_a^+ := (\mathbf{C}_a'\mathbf{C}_a)^{-1}\mathbf{C}_a'$, sodass wir aus $\boldsymbol{\alpha} = \mathbf{C}_a\boldsymbol{\alpha}^*$ gemäß $\boldsymbol{\alpha}^* = \mathbf{C}_a^+\boldsymbol{\alpha}$ die Beziehung der Reparametrisierung $\boldsymbol{\alpha}^*$ zur „ursprünglichen“ Parametrisierung $\boldsymbol{\alpha}$ erhalten.

10.11.4.1 "Treatment"-Kontraste: Definition und Eigenschaften

Natürlich stellt sich die Frage nach der Wahl von $\mathbf{C}_a$. Es gibt viele Möglichkeiten, eine Reparametrisierung vorzunehmen. **R** wählt bei einem ungeordneten Faktor für $\mathbf{C}_a$ gemäß Voreinstellung die sogenannten "treatment"-Kontraste, durch die der Parametervektor(-anteil) $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_I)'$ folgendermaßen als lineare Transformation eines Vektors $\boldsymbol{\alpha}^* = \left(\alpha_1^*, \ldots, \alpha_{I-1}^*\right)'$ ausgedrückt wird:

$$\boldsymbol{\alpha} = \mathbf{C}_a \boldsymbol{\alpha}^* \qquad \text{mit} \qquad \mathbf{C}_a = \begin{pmatrix} 0 & 0 & \ldots & 0 \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 \end{pmatrix}_{I \times (I-1)} \tag{67}$$

In **R** lässt sich mit der Funktion `contr.treatment()` zur Anzahl I der Levels eines Faktors die obige Matrix C_a generieren, indem ihr als Argument entweder I selbst oder die Levels des Faktors übergeben werden.

Einige **Eigenschaften der treatment-Kontraste**:

- Die Spalten von $\mathbf{C}_a$ sind orthogonal zueinander und $\text{Rang}([\mathbf{1}_I | \mathbf{C}_a]) = I$, sodass folgt: $\text{Rang}([\mathbf{1}_n | \mathbf{X}_a \mathbf{C}_a]) = I$.

- Die Links-Inverse $\mathbf{C}_a^+$ ergibt sich wegen $\mathbf{C}_a' \mathbf{C}_a = \text{diag}(\mathbf{1}_{I-1})$ sofort zu

$$\mathbf{C}_a^+ = (\mathbf{C}_a' \mathbf{C}_a)^{-1} \mathbf{C}_a' = \mathbf{C}_a' = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix}_{(I-1) \times I}$$

- Damit lautet die Beziehung zwischen den Kontrasten $\boldsymbol{\alpha}^*$ und den Effekten $\boldsymbol{\alpha}$:

$$\boldsymbol{\alpha}^* = \mathbf{C}_a^+ \boldsymbol{\alpha} = \begin{pmatrix} \alpha_2 \\ \vdots \\ \alpha_I \end{pmatrix}$$

  Das bedeutet faktisch, dass der Effekt α_1 aus dem Modell eliminiert (bzw. gleich Null gesetzt) wird und β_0 zur erwarteten Response des Faktorlevels 1 wird. Die Level-1-Beobachtungen werden so zu einer Art „Bezugsgruppe" (oder „Kontrollgruppe") interpretierbar. Der Kontrast α_i^* repräsentiert damit einen Vergleich zwischen dem Faktorlevel $i+1$ und dem Level 1; er quantifiziert demnach, wie sich die Behandlung (Engl.: "treatment") im Level $i+1$ relativ zur Kontrollgruppe auf die erwartete Response auswirkt.

- Die Rücktransformation von $\boldsymbol{\alpha}^*$ zu $\boldsymbol{\alpha}$ lautet:

$$\boldsymbol{\alpha} = \mathbf{C}_a \boldsymbol{\alpha}^* = \begin{pmatrix} 0 & 0 & \ldots & 0 \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 \end{pmatrix}_{I \times (I-1)} \begin{pmatrix} \alpha_1^* \\ \vdots \\ \alpha_{I-1}^* \end{pmatrix} = \begin{pmatrix} 0 \\ \alpha_2 \\ \vdots \\ \alpha_I \end{pmatrix}$$

10.11.4.2 Treatment-Kontraste im Beispiel der parallelen Regression

Für das Modell $Y_{ij} = \beta_0 + \alpha_i + \beta_1\, x_{ij} + \varepsilon_{ij}, \quad j = 1, \ldots, n_i;\ i = 1, \ldots, I$ aus Abschnitt 10.11.1 lautet die Modellformel in Matrixnotation

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{68}$$

wobei (mit $n = \sum_{i=1}^{I} n_i$)

$$\mathbf{Y} = \begin{pmatrix} Y_{11} \\ \vdots \\ Y_{1n_1} \\ Y_{21} \\ \vdots \\ Y_{2n_2} \\ \vdots \\ \vdots \\ Y_{I1} \\ \vdots \\ Y_{In_I} \end{pmatrix}_{n\times 1}, \quad \mathbf{X} = \begin{pmatrix} 1 & 1 & 0 & \ldots & \ldots & 0 & x_{11} \\ \vdots & \vdots & \vdots & & & \vdots & \vdots \\ & 1 & 0 & & & & x_{1n_1} \\ & 0 & 1 & & & & x_{21} \\ & \vdots & \vdots & & & & \vdots \\ & & 1 & & & & x_{2n_2} \\ & & 0 & \ddots & & \vdots & x_{31} \\ & & \vdots & & \ddots & 0 & \vdots \\ & & & & & 1 & x_{I1} \\ \vdots & \vdots & \vdots & & & \vdots & \vdots \\ 1 & 0 & 0 & \ldots & 0 & 1 & x_{In_I} \end{pmatrix}_{n\times(I+2)}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \alpha_1 \\ \vdots \\ \alpha_I \\ \beta_1 \end{pmatrix}_{(I+2)\times 1}$$

und $\boldsymbol{\varepsilon}$ analog zu $\mathbf{Y}$.

Wie auch schon im einführenden Beispiel (66), sind in dieser Codierung die zu β_0, α_1, …, α_I gehörenden Spalten der Designmatrix $\mathbf{X}$ linear abhängig, was die Matrix $\mathbf{X}'\mathbf{X}$ singulär werden lässt. Daher kann das obige Modell so nicht gefittet werden, sondern muss zunächst mit Hilfe einer Kontrastmatrix $\mathbf{C}_a$ reparametrisiert werden, um die Parameterzahl zu reduzieren. Dies läuft auf eine Transformation desjenigen Teils der Designmatrix hinaus, der die Codierung der Faktorlevel-Koeffizienten enthält, also hier der Spalten zwei bis $I + 1$.

Wird die treatment-Kontrastmatrix $\mathbf{C}_a$ aus (67) erweitert zu

$$\mathbf{A} := \begin{pmatrix} 1 & 0 & \ldots & \ldots & 0 \\ 0 & & & & \vdots \\ \vdots & & \mathbf{C}_a & & \vdots \\ \vdots & & & & 0 \\ 0 & \ldots & \ldots & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & \ldots & & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 0 & \vdots \\ \vdots & 1 & 0 & \ldots & 0 & \\ & 0 & 1 & \ddots & \vdots & \\ & \vdots & \ddots & \ddots & 0 & \vdots \\ \vdots & 0 & \ldots & 0 & 1 & 0 \\ 0 & \ldots & & \ldots & 0 & 1 \end{pmatrix}_{(I+2)\times(I+1)},$$

so ist $\boldsymbol{\beta} \equiv (\beta_0, \alpha_1, \ldots, \alpha_{I-1}, \alpha_I, \beta_1)' = \mathbf{A}\boldsymbol{\gamma}$ mit $\boldsymbol{\gamma} = (\beta_0, \alpha_1^*, \ldots, \alpha_{I-1}^*, \beta_1)' = (\beta_0, \alpha_2, \ldots, \alpha_I, \beta_1)'$ und β_0 sowie β_1 werden durch diese Reparametrisierung nicht beeinflusst. (Beachte, dass in Konsequenz $\alpha_1 = 0$ ist.)

Statt für Modell (68) wird nun für das reparametrisierte Modell

$$\mathbf{Y} = \mathbf{X}\mathbf{A}\boldsymbol{\gamma} + \boldsymbol{\varepsilon} =: \tilde{\mathbf{X}}\boldsymbol{\gamma} + \boldsymbol{\varepsilon} \tag{69}$$

der KQS $\hat{\boldsymbol{\gamma}}$ für $\boldsymbol{\gamma}$ ermittelt, wobei

$$\tilde{\mathbf{X}} \equiv \mathbf{XA} = \begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 & x_{11} \\ \vdots & \vdots & \vdots & & & \vdots & \vdots \\ & 0 & & & & & x_{1n_1} \\ & 1 & & & & & x_{21} \\ & \vdots & \vdots & & & & \vdots \\ & 1 & 0 & & & & x_{2n_2} \\ & 0 & 1 & & & & x_{31} \\ & \vdots & \vdots & & & & \vdots \\ & & 1 & & & & x_{3n_3} \\ & & 0 & \ddots & & \vdots & x_{41} \\ & & \vdots & & \ddots & 0 & \vdots \\ & & & & & 1 & x_{I1} \\ \vdots & \vdots & \vdots & & & \vdots & \vdots \\ 1 & 0 & 0 & \dots & 0 & 1 & x_{In_I} \end{pmatrix}_{n\times(I+1)}$$

Soll **R** ein lineares Modell mit ungeordneten Faktoren unter Verwendung der treatment-Kontraste fitten, so ist das übliche Vorgehen wie folgt:

```
> summary( fit1 <- lm( Mileage ~ Type + Weight, car.test.frame))
....
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 40.205298   3.738988  10.753 6.25e-15 ***
TypeLarge    1.074954   1.880900   0.572  0.57007
TypeMedium  -0.228946   1.029977  -0.222  0.82495
TypeSmall    3.657357   1.167511   3.133  0.00282 **
TypeSporty   1.740694   1.008523   1.726  0.09018 .
TypeVan     -1.310085   1.423336  -0.920  0.36152
Weight      -0.005697   0.001307  -4.358 6.05e-05 ***
....
```

Wir stellen fest: Die fünf (!) Zeilen für die Koeffizienten des *sechs* Levels besitzenden Faktors `Type` sind mit `TypeLarge` bis `TypeVan` bezeichnet. Insbesondere existiert kein `TypeCompact`-Koeffizient. Dies ist natürlich korrekt, da ja das reparametrisierte Modell $\mathbf{Y} = \tilde{\mathbf{X}}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}$ verwendet wurde, in dem nicht die Schätzwerte $\hat{\alpha}_1, \dots, \hat{\alpha}_6$ für die Faktorlevel-Koeffizienten $\alpha_1, \dots, \alpha_6$ bestimmt wurden, sondern die Schätzwerte $\hat{\alpha}_1^*, \dots, \hat{\alpha}_5^*$ für die treatment-Kontraste $\alpha_1^*, \dots, \alpha_5^*$, welche für die Levels `Large` bis `Van` die konstante Abweichung vom Level `Compact` darstellen. Die ausgegebenen Werte sind daher auch *nicht* die Schätzwerte $\hat{\alpha}_1, \dots, \hat{\alpha}_6$, sondern die Schätzwerte $\hat{\alpha}_1^*, \dots, \hat{\alpha}_5^*$. (Diese stehen dann via $\hat{\boldsymbol{\alpha}} = \mathbf{C}_a\hat{\boldsymbol{\alpha}}^*$ miteinander in Beziehung.)

Konkret beschreibt hier der konstante Term $\hat{\beta}_0 = 40.205298$ zusammen mit der Steigung $\hat{\beta}_1 = -0.005697$ die Regressionsgerade zum Level `Compact` und für die anderen fünf Levels `Large` bis `Van` sind die Koeffizienten `TypeLarge` bis `TypeVan` die konstanten *Abweichungen* der `Mileage` von dieser Regressionsgeraden.

Um an den Schätzwert für $\boldsymbol{\beta}$ im ursprünglichen Modell (68) zu kommen, muss die Reparametrisierung aus Modell (69) rückgängig gemacht werden, was gemäß

$$\tilde{\mathbf{X}}\boldsymbol{\gamma} \equiv \mathbf{XA}\boldsymbol{\gamma} = \mathbf{X}\boldsymbol{\beta}$$

durch $\hat{\boldsymbol{\beta}} = \mathbf{A}\hat{\gamma}$ erreicht wird. Diese Re-Reparametrisierung ist implementiert durch die Funktion `dummy.coef()`:

```
> dummy.coef( fit1)
Full coefficients are

(Intercept):     40.2053
Type:            Compact      Large     Medium      Small     Sporty        Van
               0.0000000  1.0749536 -0.2289456  3.6573575  1.7406940 -1.3100851
Weight:     -0.005697258
```

Das Resultat von `dummy.coef()` bestätigt, dass $\alpha_1 = 0$ gesetzt wurde und die Level-1-Beobachtungen als eine Art Kontroll-/Bezugs-/Referenzgruppe für die übrigen Levels erachtet werden können. (Das Ergebnis von `dummy.coef()` ist faktisch eine Liste, die jedoch nicht als solche, sondern – für meine Begriffe gelegentlich recht ungünstig – aufbereitet dargestellt wird. Mit `unclass( dummy.coef( fit1))` wird der Listencharakter deutlich und das Ergebnis meines Erachtens besser lesbar.)

Bemerkung: Die Faktorlevelkoeffizienten $\alpha_1, \ldots, \alpha_I$ werden von **R** automatisch (!) in dieser Reihenfolge den Faktorlevels zugeordnet, d. h., Koeffizient α_i korrespondiert zu Level i. Wichtig zu wissen ist dabei, dass die Levels eines ungeordneten Faktors, wenn von Benutzerin oder Benutzer nicht anders arrangiert, per Voreinstellung *alphanumerisch aufsteigend sortiert* sind.

10.11.4.3 Treatment-Kontraste im faktoriellen Modell

In Modellen mit Interaktionen tritt das Identifizierbarkeitsproblem sogar noch stärker zutage: In der Parametrisierung des faktoriellen Modells (siehe Seite 260) sind sowohl die Designmatrixspalten für $\beta_0, \alpha_1, \ldots, \alpha_I$ linear abhängig als auch diejenigen für $\beta_1, \gamma_1, \ldots, \gamma_I$. Auch hier wird intern eine (umfangreiche) Reparametrisierung vorgenommen. Die Umsetzung in **R** ist die folgende:

```
> summary( fit2 <- lm( Mileage ~ Type * Weight, car.test.frame))
....
Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)        32.657208   9.475664   3.446  0.00119 **
TypeLarge          15.653276  20.459640   0.765  0.44797
TypeMedium        -13.514109  12.040917  -1.122  0.26730
TypeSmall          18.964995  11.668288   1.625  0.11064
TypeSporty         25.451309  11.119165   2.289  0.02652 *
TypeVan            -9.272252  15.496060  -0.598  0.55241
Weight             -0.003022   0.003353  -0.901  0.37206
TypeLarge:Weight   -0.004588   0.005955  -0.770  0.44481
TypeMedium:Weight   0.003843   0.004076   0.943  0.35047
TypeSmall:Weight   -0.006113   0.004503  -1.358  0.18094
TypeSporty:Weight  -0.008450   0.003937  -2.146  0.03694 *
TypeVan:Weight      0.001734   0.004832   0.359  0.72123
....
```

Die präsentierten Koeffizientenschätzwerte sind auch hier *nicht* die, die wir in der Parametrisierung des faktoriellen Modells auf Seite 260 haben, sondern diejenigen, die in dem mittels der treatment-Kontraste reparametrisierten Modell auftreten. Also ist wieder eine Re-Reparametrisierung und damit ein Einsatz von `dummy.coef()` nötig, um die Parameter des faktoriellen Modells angeben zu können:

```
> dummy.coef( fit2)
Full coefficients are

(Intercept):    32.65721
Type:            Compact       Large      Medium       Small      Sporty         Van
                0.000000   15.653276  -13.514109   18.964995   25.451309   -9.272252
Weight:      -0.00302158
Type:Weight:     Compact       Large      Medium       Small      Sporty         Van
              0.00000000 -0.00458780  0.00384333 -0.00611262 -0.00845030  0.00173422
```

Beachte, wie $32.65721 - 0.003021579 * \texttt{Weight}$ im Level `Compact` die „Bezugsgerade“ darstellt, von der die Geraden der anderen Levels in konstantem Term und Steigung abweichen.

10.11.4.4 Treatment-Kontraste im hierarchischen Modell

In der Parametrisierung des hierarchischen Modells (siehe Seite 260) sind die Designmatrixspalten für $\beta_0, \alpha_1, \ldots, \alpha_I$ linear abhängig und wieder schafft eine interne Reparametrisierung Abhilfe. Das hierarchische Modell mit treatment-Kontrasten liefert in **R**:

```
> summary( fit3 <- lm( Mileage ~ Type / Weight, car.test.frame))
....
Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)         3.266e+01  9.476e+00   3.446  0.00119 **
TypeLarge           1.565e+01  2.046e+01   0.765  0.44797
TypeMedium         -1.351e+01  1.204e+01  -1.122  0.26730
TypeSmall           1.896e+01  1.167e+01   1.625  0.11064
TypeSporty          2.545e+01  1.112e+01   2.289  0.02652 *
TypeVan            -9.272e+00  1.550e+01  -0.598  0.55241
TypeCompact:Weight -3.022e-03  3.353e-03  -0.901  0.37206
TypeLarge:Weight   -7.609e-03  4.921e-03  -1.546  0.12858
TypeMedium:Weight   8.218e-04  2.318e-03   0.355  0.72445
TypeSmall:Weight   -9.134e-03  3.005e-03  -3.040  0.00382 **
TypeSporty:Weight  -1.147e-02  2.063e-03  -5.560 1.17e-06 ***
TypeVan:Weight     -1.287e-03  3.479e-03  -0.370  0.71296
....
```

Beachte, dass nur $\alpha_1, \ldots, \alpha_I$ reparametrisiert wurden. ($\beta_1, \ldots, \beta_I$ haben schließlich auch kein Problem dargestellt.) Die Re-Reparametrisierung versorgt uns mit den Parametern des hierarchischen Modells:

```
> dummy.coef( fit3)
Full coefficients are

(Intercept):    32.65721
Type:            Compact       Large      Medium       Small      Sporty         Van
                0.000000   15.653276  -13.514109   18.964995   25.451309   -9.272252
Type:Weight:     Compact       Large      Medium       Small      Sporty         Van
             -0.00302158 -0.00760938  0.00082175 -0.00913420 -0.01147188 -0.00128736
```

Beachte: Es gilt in der Tat die Äquivalenz obiger Modelle, derzufolge sich die Parameter gemäß

$$\hat{\beta}_i(\text{aus } \texttt{fit3}) = \hat{\beta}_1(\text{aus } \texttt{fit2}) + \hat{\gamma}_i(\text{aus } \texttt{fit2})$$

ineinander umrechnen lassen müssen. In **R**:

```
> dummy.coef( fit3)$"Type:Weight" -
+ (dummy.coef( fit2)$Weight + dummy.coef( fit2)$"Type:Weight")

     Compact        Large       Medium        Small       Sporty          Van
-4.16334e-17  0.00000e+00 -1.40946e-18 -1.73472e-17  3.46945e-18 -1.08420e-18
```

Offenbar sind die Differenzen $\hat{\beta}_i(\text{aus } \texttt{fit3}) - (\hat{\beta}_1(\text{aus } \texttt{fit2}) + \hat{\gamma}_i(\text{aus } \texttt{fit2}))$ im Rahmen der Rechengenauigkeit Null.

10.11.4.5 Helmert-Kontraste: Definition und Eigenschaften

Eine weitere Möglichkeit, bei Variablen vom Typ ungeordneter Faktor eine Reparametrisierung vorzunehmen, sind die sogenannten Helmert-Kontraste. (Sie sind etwas „gewöhnungsbedürftig“ und nicht so suggestiv wie die treatment-Kontraste. Unverständlicherweise sind sie in S-PLUS die Voreinstellung.) Ihre Kontrastmatrix $\mathbf{C}_a$ lautet

$$\mathbf{C}_a = \begin{pmatrix} -1 & -1 & \dots & -1 \\ 1 & -1 & \dots & -1 \\ 0 & 2 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & I-1 \end{pmatrix}_{I\times(I-1)}$$

In **R** lässt sich mit der Funktion `contr.helmert()` zur Anzahl I der Levels eines Faktors die obige Matrix C_a generieren, indem ihr als Argument entweder I selbst oder die Levels des Faktors übergeben werden.

Einige **Eigenschaften der Helmert-Kontraste**:

- Die Spalten von $\mathbf{C}_a$ sind orthogonal zueinander und $\text{Rang}([\mathbf{1}_I|\mathbf{C}_a]) = I$, sodass folgt: $\text{Rang}([\mathbf{1}_n|\mathbf{X}_a\mathbf{C}_a]) = I$.

- Die Links-Inverse $\mathbf{C}_a^+$ ist wegen

$$\mathbf{C}_a'\mathbf{C}_a = \text{diag}\left(\{s(s+1)\}_{s=1}^{I-1}\right) = \begin{pmatrix} 2 & & & & \\ & 6 & & & 0 \\ & & 12 & & \\ & 0 & & \ddots & \\ & & & & (I-1)I \end{pmatrix}$$

  leicht berechenbar zu

$$\mathbf{C}_a^+ = (\mathbf{C}_a'\mathbf{C}_a)^{-1}\mathbf{C}_a' = \begin{pmatrix} -\frac{1}{2} & \frac{1}{2} & 0 & \dots & & & 0 \\ -\frac{1}{6} & -\frac{1}{6} & \frac{2}{6} & 0 & \dots & & 0 \\ -\frac{1}{12} & -\frac{1}{12} & -\frac{1}{12} & \frac{3}{12} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ & & & & \ddots & \ddots & 0 \\ -\frac{1}{(I-1)I} & \dots & & & \dots & -\frac{1}{(I-1)I} & \frac{I-1}{(I-1)I} \end{pmatrix}_{(I-1)\times I}$$

- Damit lautet die Beziehung zwischen den Kontrasten $\boldsymbol{\alpha}^*$ und den Effekten $\boldsymbol{\alpha}$:

$$\boldsymbol{\alpha}^* = \mathbf{C}_a^+ \boldsymbol{\alpha} = \begin{pmatrix} \frac{1}{2}(\alpha_2 - \alpha_1) \\ \frac{1}{3}(\alpha_3 - \frac{1}{2}(\alpha_1 + \alpha_2)) \\ \frac{1}{4}(\alpha_4 - \frac{1}{3}(\alpha_1 + \alpha_2 + \alpha_3)) \\ \vdots \\ \frac{1}{I}(\alpha_I - \frac{1}{I-1}(\alpha_1 + \ldots + \alpha_{I-1})) \end{pmatrix}$$

D. h., der Kontrast α_i^* ist interpretierbar als (gewichtete) Differenz zwischen dem Effekt α_{i+1} des Faktorlevels $i+1$ und dem „Durchschnittseffekt“ der Levels 1 bis i.

- Die entsprechende Rücktransformation von $\boldsymbol{\alpha}^*$ zu $\boldsymbol{\alpha}$ lautet: $\boldsymbol{\alpha} = \mathbf{C}_a \boldsymbol{\alpha}^* =$

$$= \begin{pmatrix} -1 & -1 & -1 & \ldots & -1 & -1 \\ 1 & -1 & -1 & & \vdots & \vdots \\ 0 & 2 & -1 & & & \\ \vdots & 0 & 3 & \ddots & \vdots & \\ & \vdots & 0 & \ddots & -1 & \vdots \\ \vdots & \vdots & \vdots & \ddots & I-2 & -1 \\ 0 & 0 & 0 & \ldots & 0 & I-1 \end{pmatrix}_{I \times (I-1)} \begin{pmatrix} \alpha_1^* \\ \vdots \\ \alpha_{I-1}^* \end{pmatrix} = \begin{pmatrix} -\sum_{s=1}^{I-1} \alpha_s^* \\ \alpha_1^* - \sum_{s=2}^{I-1} \alpha_s^* \\ 2\alpha_2^* - \sum_{s=3}^{I-1} \alpha_s^* \\ 3\alpha_3^* - \sum_{s=4}^{I-1} \alpha_s^* \\ \vdots \\ (I-2)\alpha_{I-2}^* - \alpha_{I-1}^* \\ (I-1)\alpha_{I-1}^* \end{pmatrix}$$

- Eine weitere Konsequenz der Verwendung von Helmert-Kontrasten ist, dass sich die ursprünglichen Parameter (Haupteffekte) α_i zu Null addieren:

$$\sum_{s=1}^{I} \alpha_s = \mathbf{1}_I' \boldsymbol{\alpha} = \mathbf{1}_I' \mathbf{C}_a \, \boldsymbol{\alpha}^* = \mathbf{0}_{I-1}' \boldsymbol{\alpha}^* = 0$$

10.11.4.6 Helmert-Kontraste im Beispiel der parallelen Regression

Wird für das Modell $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ aus §10.11.1 (bzw. Modell (68) in §10.11.4.2, Seite 265) die Helmert-Kontrastmatrix $\mathbf{C}_a$ verwendet und diese erweitert zu

$$\mathbf{A} := \begin{pmatrix} 1 & 0 & \ldots & \ldots & 0 \\ 0 & & & & \vdots \\ \vdots & & \mathbf{C}_a & & \vdots \\ \vdots & & & & 0 \\ 0 & \ldots & \ldots & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & -1 & -1 & \ldots & -1 & 0 \\ 0 & 1 & -1 & \ldots & -1 & 0 \\ 0 & 0 & 2 & \ldots & -1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & I-1 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 1 \end{pmatrix}_{(I+2)\times(I+1)},$$

dann ist $\boldsymbol{\beta} \equiv (\beta_0, \alpha_1, \ldots, \alpha_{I-1}, \alpha_I, \beta_1)' = \mathbf{A}\boldsymbol{\gamma}$ mit $\boldsymbol{\gamma} = (\beta_0, \alpha_1^*, \ldots, \alpha_{I-1}^*, \beta_1)'$, sodass β_0 und β_1 auch durch diese Reparametrisierung unbeeinflusst gelassen werden.

Nun wird für das Modell $\mathbf{Y} = \mathbf{X}\mathbf{A}\boldsymbol{\gamma} + \boldsymbol{\varepsilon} =: \tilde{\mathbf{X}}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}$ der KQS $\hat{\boldsymbol{\gamma}}$ für $\boldsymbol{\gamma}$ ermittelt, wobei

$$\tilde{\mathbf{X}} \equiv \mathbf{X}\mathbf{A} = \begin{pmatrix} 1 & -1 & -1 & \dots & \dots & -1 & x_{11} \\ \vdots & \vdots & \vdots & & & \vdots & \vdots \\ & -1 & & & & & x_{1n_1} \\ & 1 & & & & & x_{21} \\ & \vdots & \vdots & & & & \vdots \\ & 1 & -1 & & & & x_{2n_2} \\ & 0 & 2 & & & & x_{31} \\ & \vdots & \vdots & & & & \vdots \\ & & 2 & & & & x_{3n_3} \\ & & 0 & \ddots & & \vdots & x_{41} \\ & & \vdots & & \ddots & -1 & \vdots \\ & & & & & I-1 & x_{I1} \\ \vdots & \vdots & \vdots & & & \vdots & \vdots \\ 1 & 0 & 0 & \dots & 0 & I-1 & x_{In_I} \end{pmatrix}_{n\times(I+1)}$$

Das spiegelt sich auch prompt in der folgenden Ausgabe von **R** wider. Beachte, dass zunächst die Voreinstellung der treatment-Kontraste geändert und die Verwendung der Helmert-Kontraste für ungeordnete Faktoren aktiviert werden muss, was durch den Befehl `options( contrasts= c( "contr.helmert", "contr.poly"))` geschieht:

```
> options( contrasts= c( "contr.helmert", "contr.poly"))
> summary( fit4 <- lm( Mileage ~ Type + Weight, car.test.frame))
....
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 41.027627   3.995954  10.267 3.32e-14 ***
Type1        0.537477   0.940450   0.572   0.5701
Type2       -0.255474   0.336024  -0.760   0.4505
Type3        0.843839   0.384218   2.196   0.0325 *
Type4        0.122971   0.189106   0.650   0.5183
Type5       -0.426483   0.205369  -2.077   0.0427 *
Weight      -0.005697   0.001307  -4.358 6.05e-05 ***
....
```

Zunächst stellen wir fest, dass die Zeilen für die Koeffizienten des Faktors `Type` nicht mit „Compact“, „Large“, „Medium“, „Small“, „Sporty“ und „Van“ bezeichnet werden, sondern mit `Type1` usw. (Das leuchtet ein, da die Helmert-Kontraste wenig mit den ursprünglichen Levels zu tun haben.) Des Weiteren existiert kein Koeffizient für das sechste Level von `Type`, weil ja das reparametrisierte Modell $\mathbf{Y} = \tilde{\mathbf{X}}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}$ verwendet wurde. Die ausgegebenen Werte in den Zeilen `Type1` bis `Type5` sind also *nicht* die Schätzwerte $\hat{\alpha}_1, \dots, \hat{\alpha}_6$ für die Faktorlevel-Koeffizienten, sondern die Schätzwerte $\hat{\alpha}_1^*, \dots, \hat{\alpha}_5^*$ für die Helmert-Kontraste (die ja bekanntlich via $\hat{\boldsymbol{\alpha}} = \mathbf{C}_a\hat{\boldsymbol{\alpha}}^*$ miteinander in Beziehung stehen).

Um an den Schätzwert für $\boldsymbol{\beta}$ im ursprünglichen Modell (68) zu kommen, muss die Reparametrisierung rückgängig gemacht werden, was durch $\hat{\boldsymbol{\beta}} = \mathbf{A}\hat{\boldsymbol{\gamma}}$ erreicht wird und in `dummy.coef()` implementiert ist:

```
> dummy.coef( fit4)
Full coefficients are

(Intercept):    41.02763
```

```
Type:            Compact      Large     Medium      Small     Sporty        Van
              -0.8223291  0.2526246 -1.0512747  2.8350284  0.9183649 -2.1324142
Weight:     -0.005697258
```

Nun ist also für alle sechs Levels (von `Compact` bis `Van`) des Faktors `Type` die konstante Abweichung (von -0.8223291 bis -2.1324142) der `Mileage` von der „mittleren“ Regressionsgeraden mit der Steigung $\hat{\beta}_1 = -0.005697258$ und dem konstanten Term $\hat{\beta}_0 = 41.02763$ quantifiziert.

Beachte, dass sich die `Type`-Effekte zu Null summieren.

10.11.4.7 Helmert-Kontraste im faktoriellen Modell

In der Parametrisierung des faktoriellen Modells (vgl. Seite 260) sind sowohl die Designmatrixspalten für $\beta_0, \alpha_1, \ldots, \alpha_I$ linear abhängig als auch diejenigen für $\beta_1, \gamma_1, \ldots, \gamma_I$. Das Resultat eines Fits in **R** bei Reparametrisierung mit Helmert-Kontrasten ist das folgende (wobei die erneute Aktivierung der Helmert-Kontraste mittels `options()` nicht notwendig ist, wenn an ihrer Einstellung nichts geändert und **R** auch nicht erneut gestartet wurde):

```
> options( contrasts= c( "contr.helmert", "contr.poly"))
> summary( fit5 <- lm( Mileage ~ Type * Weight, car.test.frame))
....
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)  38.8710777  4.4232354   8.788 1.46e-11 ***
Type1         7.8266381 10.2298202   0.765 0.447969
Type2        -7.1135825  4.2143236  -1.688 0.097908 .
Type3         4.5629848  2.4875232   1.834 0.072805 .
Type4         4.0350536  1.6291986   2.477 0.016836 *
Type5        -3.0972244  2.1889857  -1.415 0.163549
Weight       -0.0052838  0.0013562  -3.896 0.000302 ***
Type1:Weight -0.0022939  0.0029773  -0.770 0.444808
Type2:Weight  0.0020457  0.0012577   1.627 0.110362
Type3:Weight -0.0014661  0.0009207  -1.592 0.117876
Type4:Weight -0.0013472  0.0005431  -2.481 0.016669 *
Type5:Weight  0.0007993  0.0006295   1.270 0.210312
....
```

Auch die Bezeichnung der `Type:Weight`-Interaktionseffekte geschieht auf die für Helmert-Kontraste typische, wenig suggestive Art. Und natürlich ist wieder eine Re-Reparametrisierung nötig, um die Parameter des faktoriellen Modells identifizieren zu können:

```
> dummy.coef( fit5)
Full coefficients are

(Intercept):    38.87108
Type:            Compact       Large      Medium       Small      Sporty         Van
              -6.213870    9.439407  -19.727979   12.751125   19.237439  -15.486122
Weight:      -0.00528377
Type:Weight:     Compact       Large      Medium       Small      Sporty         Van
              0.00226219 -0.00232561  0.00610552 -0.00385042 -0.00618811  0.00399642
```

Beachte: Sowohl die sechs `Type`-Haupteffekte als auch die sechs `Type:Weight`-Interaktionseffekte addieren sich zu Null.

10.11.4.8 Helmert-Kontraste im hierarchischen Modell

Das hierarchische Modell (vgl. Seite 260) mit linear abhängigen Designmatrixspalten für $\beta_0, \alpha_1, \ldots, \alpha_I$ liefert in **R** für Helmert-Kontraste (wobei die Aktivierung der Helmert-Kontraste durch `options()` nur nötig ist, wenn ihre Einstellung geändert oder **R** erneut gestartet wurde):

```
> options( contrasts= c( "contr.helmert", "contr.poly"))
> summary( fit6 <- lm( Mileage ~ Type / Weight, car.test.frame))
....
Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)        38.8710777  4.4232354   8.788 1.46e-11 ***
Type1               7.8266381 10.2298202   0.765  0.44797
Type2              -7.1135825  4.2143236  -1.688  0.09791 .
Type3               4.5629848  2.4875232   1.834  0.07280 .
Type4               4.0350536  1.6291986   2.477  0.01684 *
Type5              -3.0972244  2.1889857  -1.415  0.16355
TypeCompact:Weight -0.0030216  0.0033534  -0.901  0.37206
TypeLarge:Weight   -0.0076094  0.0049207  -1.546  0.12858
TypeMedium:Weight   0.0008218  0.0023175   0.355  0.72445
TypeSmall:Weight   -0.0091342  0.0030046  -3.040  0.00382 **
TypeSporty:Weight  -0.0114719  0.0020633  -5.560 1.17e-06 ***
TypeVan:Weight     -0.0012874  0.0034787  -0.370  0.71296
....
```

Beachte, dass nur $\alpha_1, \ldots, \alpha_I$ reparametrisiert wurden und nicht $\beta_1, \ldots, \beta_I$, wie man auch an den Bezeichnungen der jeweiligen Effekte erkennen kann. Zur Re-Reparametrisierung dient wie bisher `dummy.coef()` und die `Type`-Haupteffekte summieren sich wieder zu Null:

```
> dummy.coef( fit6)
Full coefficients are

(Intercept):    38.87108
Type:            Compact       Large      Medium       Small      Sporty         Van
               -6.213870    9.439407  -19.727979   12.751125   19.237439  -15.486122
Type:Weight:     Compact       Large      Medium       Small      Sporty         Van
             -0.00302158 -0.00760938  0.00082175 -0.00913420 -0.01147188 -0.00128736
```

Beachte: Selbstverständlich gilt auch hier wieder die Äquivalenz der obigen Modelle, derzufolge sich die Parameter gemäß $\hat{\beta}_i(\text{aus } \texttt{fit6}) = \hat{\beta}_1(\text{aus } \texttt{fit5}) + \hat{\gamma}_i(\text{aus } \texttt{fit5})$ ineinander umrechnen lassen (in **R** freilich nur im Rahmen der Rechnergenauigkeit).

10.11.5 Modellparametrisierung geordneter Faktoren durch Polynom-Kontraste

Für die Parametrisierung im Fall von geordneten Faktoren, d. h. von ordinalskalierten Covariablen, verwendet **R** Kontrastmatrizen, die mit Hilfe von Orthonormalpolynomen erzeugt werden (eine gewöhnungsbedürftige Vorgehensweise).

Zur Erinnerung siehe Seite 263: Das Problem im Zusammenhang mit einem Faktor mit I Levels im Regressionsmodell ist, dass die Spalten seiner $(n \times I)$-Inzidenzmatrix $\mathbf{X}_a$ und die 1-Spalte $\mathbf{1}_n$ des konstanten Terms linear abhängig sind. Die Reparametrisierung mittels einer $(I \times (I-1))$-Kontrastmatrix $\mathbf{C}_a$ geschieht dergestalt, dass die Inzidenzmatrix ersetzt wird durch $\mathbf{X}_a\mathbf{C}_a$, sodass $\text{Rang}([\mathbf{1}_n|\mathbf{X}_a\mathbf{C}_a]) = I$ und das KQ-Problem somit lösbar ist. Dies ist z. B. dann garantiert, wenn die $I-1$ Spalten von $\mathbf{C}_a$ zueinander und zur 1-Spalte $\mathbf{1}_n$ orthogonal sind.

Im Falle eines geordneten Faktors wird letzteres in **R** durch Kontrastmatrizen erreicht, deren $I-1$ Spalten als Orthonormalpolynome der Grade 1 bis $I-1$ über einem Gitter von I äquidistanten Punkten interpretierbar sind. Diese zunächst etwas seltsam anmutende Strategie erlaubt es jedoch, bei der Interpretation der Kontraste und Koeffizienten des geordneten Faktors bis zu einem gewissen Grad den Charakter seiner Ordinalskalierung zu berücksichtigen.

Zur weiteren Erinnerung: Zu $z_1, z_2, \ldots, z_k \in \mathbb{R}$ seien $p_0, p_1, \ldots, p_{k-1}$ Orthonormalpolynome mit $\text{Grad}(p_s) = s$ für $s = 0, 1, \ldots, k-1$. Dann gilt für $\mathbf{z} := (z_1, \ldots, z_k)$ und alle $0 \leq s, t \leq k-1$:

$$\begin{aligned}(p_s(\mathbf{z}))' p_t(\mathbf{z}) &\equiv (p_s(z_1), \ldots, p_s(z_k)) \begin{pmatrix} p_t(z_1) \\ \vdots \\ p_t(z_k) \end{pmatrix} \\ &= \begin{cases} 0 & , s \neq t \\ \parallel p_s \parallel^2 := \sum_{l=1}^k p_s(z_l)^2 = 1, & s = t \end{cases} .\end{aligned}$$

R generiert die Orthonormalpolynome zur Codierung eines geordneten Faktors mit I Levels ähnlich wie jene, welche im Abschnitt 10.10 über polynomiale Regression im Zusammenhang mit der Funktion `poly()` besprochen wurden, und zwar per Voreinstellung wie folgt:

Definition und Eigenschaften der Kontrastmatrix eines geordneten Faktors mit I Levels: Durch $z_i := i$ (für $i = 1, \ldots, I$) wird jedem Faktorlevel i ein $z_i \in \mathbb{R}$ „zugeordnet“ und dadurch ein äquidistantes Gitter $z_1 < \ldots < z_I$ in $\mathbb{R}$ definiert. Die Kontrastmatrix sei

$$\mathbf{C}_a := \begin{pmatrix} p_1(z_1) & p_2(z_1) & \ldots & p_{I-1}(z_1) \\ p_1(z_2) & p_2(z_2) & \ldots & p_{I-1}(z_2) \\ \vdots & \vdots & & \vdots \\ p_1(z_I) & p_2(z_I) & \ldots & p_{I-1}(z_I) \end{pmatrix}_{I \times (I-1)}$$

Obige Matrix C_a lässt sich mit der **R**-Funktion `contr.poly()` zur Anzahl I der Levels eines (geordneten) Faktors generieren, indem ihr als Argument entweder I selbst oder die Levels des Faktors übergeben werden.

Es gilt:

- Es ist $\text{Rang}(\underbrace{[\mathbf{1}_I | \mathbf{C}_a]}_{I \times I}) = I$ und somit $\text{Rang}(\underbrace{[\mathbf{1}_n | \mathbf{X}_a \mathbf{C}_a]}_{n \times I}) = I$.

- Wegen $p_0 \equiv 1$ (denn $\text{Grad}(p_0) = 0$) garantiert die Orthogonalität von p_0 zu $p_1, \ldots, p_{I-1}$, dass

$$\sum_{l=1}^{I} p_s(z_l) = \sum_{l=1}^{I} p_0(z_l) p_s(z_l) = 0 \quad \text{für jedes } s = 1, \ldots, I-1$$

  und somit $\mathbf{1}_I' \mathbf{C}_a = \mathbf{0}_{I-1}'$.

- Die Orthonormalität der p_s hat zufolge, dass $\mathbf{C}_a' \mathbf{C}_a = \text{diag}(\{\parallel p_s \parallel^2\}_{s=1}^{I-1})$ die $((I-1) \times (I-1))$-Einheitsmatrix ist, denn $\parallel p_s \parallel^2 \equiv 1$ für alle $s = 1, \ldots, I-1$. Somit ist

$$\mathbf{C}_a^+ = (\mathbf{C}_a' \mathbf{C}_a)^{-1} \mathbf{C}_a' = \mathbf{C}_a'$$

- Für die Kontraste $\boldsymbol{\alpha}^*$ erhalten wir dann

$$\begin{aligned}
\boldsymbol{\alpha}^* = \mathbf{C}_a^+ \boldsymbol{\alpha} \quad &\equiv \quad \begin{pmatrix} p_1(z_1) & p_1(z_2) & \dots & p_1(z_I) \\ p_2(z_1) & p_2(z_2) & \dots & p_2(z_I) \\ \vdots & \vdots & & \vdots \\ p_{I-1}(z_1) & p_{I-1}(z_2) & \dots & p_{I-1}(z_I) \end{pmatrix}_{(I-1)\times I} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_I \end{pmatrix} \\
&= \begin{pmatrix} \sum_{l=1}^I p_1(z_l)\alpha_l \\ \sum_{l=1}^I p_2(z_l)\alpha_l \\ \vdots \\ \sum_{l=1}^I p_{I-1}(z_l)\alpha_l \end{pmatrix} \quad \begin{matrix} \longleftarrow \text{ linear in den } z_l \\ \longleftarrow \text{ quadratisch in den } z_l \\ \vdots \\ \longleftarrow \text{ Grad } I-1 \end{matrix}
\end{aligned}$$

- Die Re-Reparametrisierung lautet

$$\begin{aligned}
\boldsymbol{\alpha} = \mathbf{C}_a \boldsymbol{\alpha}^* \quad &\equiv \quad \begin{pmatrix} p_1(z_1) & p_2(z_1) & \dots & p_{I-1}(z_1) \\ p_1(z_2) & p_2(z_2) & \dots & p_{I-1}(z_2) \\ \vdots & \vdots & & \vdots \\ p_1(z_I) & p_2(z_I) & \dots & p_{I-1}(z_I) \\ \textit{linear} & \textit{quadrat.} & & \textit{Grad } I-1 \end{pmatrix}_{I\times(I-1)} \begin{pmatrix} \alpha_1^* \\ \vdots \\ \alpha_{I-1}^* \end{pmatrix} \\
&= \begin{pmatrix} \sum_{s=1}^{I-1} \alpha_s^* p_s(z_1) \\ \vdots \\ \sum_{s=1}^{I-1} \alpha_s^* p_s(z_I) \end{pmatrix},
\end{aligned}$$

  d. h., der Effekt α_i des Levels i ist eine Linearkombination von Orthogonalpolynomen der Grade 1 bis $I-1$ ausgewertet an z_i (wobei $z_1 < z_2 < \ldots < z_I$ äquidistant).

  Bedeutung: Der Einfluss der geordneten (!) Faktorlevels lässt sich polynomial modellieren (oder z. B. auch nur linear, wenn sich $\alpha_2^*, \ldots, \alpha_{I-1}^*$ alle als *nicht* signifikant verschieden von Null herausstellen sollten).

- Das reparametrisierte Modell lautet übrigens

$$Y_{ij} = \beta_0 + \alpha_1^* p_1(z_i) + \alpha_2^* p_2(z_i) + \ldots + \alpha_{I-1}^* p_{I-1}(z_i) + \varepsilon_{ij}$$

- Die Wahl der Kontraste garantiert wieder

$$\sum_{s=1}^I \alpha_s = \mathbf{1}_I' \mathbf{C}_a \, \boldsymbol{\alpha}^* = \mathbf{0}_{I-1}' \boldsymbol{\alpha}^* = 0$$

Ein (wenig sinnvolles) **Beispiel** für die Ausgabe von **R**: Wir wandeln den Fahrzeugtyp `Type` aus `car.test.frame` um in einen geordneten Faktor `oType` mit der Level-Ordnung `Small < Compact < Sporty < Medium < Large < Van` und fitten dann die `Mileage` an `oType` und `Weight`:

```
> oType <- ordered( car.test.frame$Type, levels = c( "Small", "Compact",
+                                                    "Sporty", "Medium", "Large", "Van"))
> summary( fit7 <- lm( Mileage ~ oType + Weight, data = car.test.frame))
....
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 41.027627   3.995954  10.267 3.32e-14 ***
oType.L     -2.818587   1.688737  -1.669   0.1010
```

```
oType.Q      0.503474   0.811241   0.621   0.5375
oType.C     -1.824880   0.952929  -1.915   0.0609 .
oType^4      0.405539   1.056330   0.384   0.7026
oType^5     -1.892254   0.815173  -2.321   0.0241 *
Weight      -0.005697   0.001307  -4.358 6.05e-05 ***
....
```

R hat für den geordneten Faktor `oType` mit sechs Levels automatisch (gemäß seiner Voreinstellung) fünf polynomiale Kontraste gewählt und sie mit `oType.L`, `oType.Q`, `oType.C`, `oType^4` und `oType^5` benannt. Dabei deuten `L`, `Q` und `C` die Koeffizienten für den linearen, quadratischen bzw. kubischen Term an und die Grade 4 und aufwärts werden durch ein angehängtes `^4` usw. abgekürzt.

Die Funktion `dummy.coef()` steht auch hier zur Verfügung, um $\boldsymbol{\alpha}$ zu liefern:

```
Full coefficients are

(Intercept):    41.02763
oType:             Small    Compact     Sporty     Medium      Large        Van
               2.8350284 -0.8223291  0.9183649 -1.0512747  0.2526246 -2.1324142
Weight:     -0.005697258
```

Nachrechnen bestätigt, dass sich die `oType`-Effekte, wie oben behauptet, zu Null addieren.

10.12 F-Tests gewisser linearer Hypothesen: `anova()`

Im Rahmen der multiplen linearen Modelle, die uns bisher begegnet sind, haben wir unterschiedlich komplizierte Modelle kennengelernt: Stetige Covariablen allein oder gemeinsam mit diskreten Faktor-Covariablen, ohne und mit Interaktionen. Des Weiteren sind uns inzwischen einige Möglichkeiten für die Modifikation von Modellen, sprich `lm`-Objekten bekannt (siehe z. B. Abschnitt 10.7). Allerdings können wir bei der Beurteilung, ob ein Covariablenterm einen signifikanten Einfluss auf die Response hat, bisher nur marginale Hypothesentests der Art $H_0 : \beta_j = 0$ und ihre zugehörigen p-Werte heranziehen (zu finden in der Ausgabe der Funktion `summary()`) oder Akaike's "information criterion" AIC verwenden (gleichbedeutend mit Mallow's C_p und aufgetaucht in den Ausgaben der „Modellbau"-Funktionen `drop1()` und `add1()` der Abschnitte 10.7.2 bzw. 10.7.3).

Von Interesse sind jedoch auch Hypothesen, mit denen mehrere Terme *gleichzeitig* auf signifikanten Einfluss getestet werden, z. B. wenn sogenannte hierarchische Modelle miteinander verglichen werden sollen. Dabei heißen zwei Modelle M_1 und M_2 <u>hierarchisch</u> (auch geschachtelt bzw. auf Englisch "nested" genannt), wenn der Parameterraum von Modell M_1 ein Unterraum des Parameterraums von M_2 ist, was wiederum insbesondere dann der Fall ist, wenn die Menge der Modellterme von M_1 eine Teilmenge der Modellterme von M_2 ist. Als Kurzschreibweise für diesen Sachverhalt wollen wir $M_1 \subset M_2$ verwenden und dann M_1 ein <u>Submodell</u> von M_2 nennen.

Um also zu prüfen, ob zwischen einem (Ober-)Modell M_2 und einem Submodell $M_1 \subset M_2$ ein signifikanter Unterschied in der Beschreibung der Regressionsbeziehung zwischen Response und Covariablentermen besteht, ist im Modell M_2 die Hypothese zu testen, dass die Regressionskoeffizienten derjenigen Covariablenterme von M_2, die in M_1 nicht auftreten, alle gleich Null sind: $H_0 : \beta_{j_1} = \ldots = \beta_{j_k} = 0$. (Zur Theorie des F-Tests allgemeiner linearer Hypothesen verweisen wir auf den Anfang von Kapitel 10, wo zur Wiederholung einige ihrer Aspekte aufgeführt worden waren (Seite 199 f).)

Ein paar spezielle Details bei der Betrachtung hierarchischer Modelle: Es seien $p_1 < p_2$ die Parameterdimensionen von M_1 bzw. M_2 und $k := p_2 - p_1$ die Dimensionendifferenz. Die zu testende Nullhypothese lautet in Form einer linearen Bedingung an den Parametervektor $\boldsymbol{\beta}$

$$H_0 : \mathbf{C}\boldsymbol{\beta} = \mathbf{0} \quad \text{mit der } (k \times p_2)\text{-Kontrastmatrix } \mathbf{C} = \begin{pmatrix} \mathbf{e}'_{j_1} \\ \vdots \\ \mathbf{e}'_{j_k} \end{pmatrix},$$

wobei $\mathbf{e}'_j = (\underbrace{0, \ldots, 0}_{j-1}, 1, 0, \ldots, 0)$ der j-te p_2-dimensionale Einheitsvektor ist. Offenbar ist $\text{Rang}(\mathbf{C}) = k \equiv p_2 - p_1$, also der Unterschied der Dimensionen der zwei betrachteten hierarchischen Modelle M_1 und M_2. Die zu H_0 gehörige F-Teststatistik lautet (wie schon in Abschnitt 10.1 auf Seite 199 in Erinnerung gerufen) wegen $\text{Rang}(\mathbf{C}) = p_2 - p_1 = n - p_1 - (n - p_2)$

$$T(\mathbf{Y}) = \frac{(\text{RSS}_{H_0} - \text{RSS})/\text{Rang}(\mathbf{C})}{\text{RSS}/(n-p)} \overset{\text{hier}}{=} \frac{(\text{RSS}_{M_1} - \text{RSS}_{M_2})/(n - p_1 - (n - p_2))}{\text{RSS}_{M_2}/(n - p_2)} \tag{70}$$

Dabei heißen $n - p_1$ und $n - p_2$ auch die Residuenfreiheitsgrade (= "residual degrees of freedom") von Modell M_1 bzw. M_2.

Ein Vergleich hierarchischer Modelle im Sinne dieser Hypothese kann mit der Funktion `anova()`, die eine geeignete Varianzanalyse (kurz: ANOVA = "analysis of variance") und den entsprechenden F-Test durchführt, bewerkstelligt werden. Anhand von Beispielen wird ihre praktischen Anwendung für den Vergleich hierarchischer Modelle in den nächsten Abschnitten erläutert.

10.12.1 Nur stetige Covariablen

Wir betrachten den NA-bereinigten Ozon-Datensatz (vom Ende von Abschnitt 10.2), lassen zunächst allerdings aus Gründen der besseren Lesbarkeit seine Variablenbenennungen automatisch verkürzen: Die Funktion `abbreviate()` ermittelt für die Elemente eines `character`-Vektors eindeutige Abkürzungen der (Mindest-)Länge `minlength`. Hier ist ihre Anwendung eine Art "overkill"; die explizite Angabe der einbuchstabigen Spaltennamen wäre „von Hand" kürzer gewesen ... (Beachte: `air` enthält nur 111 Zeilen, obwohl seine gezeigten Zeilen*namen* (nicht *-nummern*) etwas anderes suggerieren.)

```
> air <- na.exclude( airquality)
> names( air) <- abbreviate( names( air), minlength = 1);     air
     O   S    W  T M  D
1   41 190  7.4 67 5  1
2   36 118  8.0 72 5  2
3   12 149 12.6 74 5  3
4   18 313 11.5 62 5  4
7   23 299  8.6 65 5  7
....
153 20 223 11.5 68 9 30
```

Wir fitten das volle Modell mit allen zur Verfügung stehenden, durchweg stetigen Covariablen samt ihrer Interaktionen bis zur Ordnung drei, also

$$\mathbb{E}[\mathtt{O}] \;=\; \beta_0 + \beta_1\,\mathtt{T} + \beta_2\,\mathtt{W} + \beta_3\,\mathtt{S} + \beta_4\,\mathtt{T}\cdot\mathtt{W} + \beta_5\,\mathtt{T}\cdot\mathtt{S} + \beta_6\,\mathtt{W}\cdot\mathtt{S} + \beta_7\,\mathtt{T}\cdot\mathtt{W}\cdot\mathtt{S} \tag{71}$$

In **R**:

```
> summary( oz3I.lm <- lm( O ~ T * W * S, data = air))
....
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -7.139e+01  1.079e+02  -0.661    0.510
T            1.348e+00  1.476e+00   0.913    0.363
W            4.329e+00  8.888e+00   0.487    0.627
S           -6.647e-01  5.876e-01  -1.131    0.261
T:W         -7.262e-02  1.255e-01  -0.578    0.564
T:S          1.098e-02  7.790e-03   1.409    0.162
W:S          3.389e-02  5.184e-02   0.654    0.515
T:W:S       -5.604e-04  7.005e-04  -0.800    0.426
....
```

Wir stellen fest, dass in diesem Modell kein einziger Term einen signifikanten Einfluss zu haben scheint, denn die marginalen (!) p-Werte der Terme sind alle größer als 0.1. Es liegt nahe, die marginal nicht-signifikanten vier Interaktionsterme aus dem vollen Modell `oz3I.lm` zu entfernen, weil sie es unnötig zu „verkomplizieren" scheinen. Aus Anschauungsgründen wollen wir dies mit einem Zwischenschritt (über das Modell `oz2I.lm`) tun, in welchem lediglich die Dreifach-Interaktion fehlt:

```
> oz2I.lm <- update( oz3I.lm, ~ . - T:W:S)
> oz0I.lm <- update( oz2I.lm, ~ . - T:W - T:S - W:S)
```

Das heißt, dass `oz2I.lm` das Modell

$$\mathbb{E}[\mathtt{O}] \;=\; \beta_0 + \beta_1\,\mathtt{T} + \beta_2\,\mathtt{W} + \beta_3\,\mathtt{S} + \beta_4\,\mathtt{T}\cdot\mathtt{W} + \beta_5\,\mathtt{T}\cdot\mathtt{S} + \beta_6\,\mathtt{W}\cdot\mathtt{S} \tag{72}$$

enthält. Darin sind nun die Covariable `T` und der Interaktionsterm `T:W` marginal signifikant zum Niveau 1 % sowie die Covariable `W` und der Interaktionsterm `T:S` marginal signifikant zum Niveau 5 %:

```
> summary( oz2I.lm)
....
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.408e+02  6.419e+01  -2.193  0.03056 *
T            2.322e+00  8.330e-01   2.788  0.00631 **
W            1.055e+01  4.290e+00   2.460  0.01555 *
S           -2.260e-01  2.107e-01  -1.073  0.28591
T:W         -1.613e-01  5.896e-02  -2.735  0.00733 **
T:S          5.061e-03  2.445e-03   2.070  0.04089 *
W:S         -7.231e-03  6.688e-03  -1.081  0.28212
....
```

In oz0I.lm ist das Modell

$$\mathbb{E}[\mathtt{O}] \;=\; \beta_0 + \beta_1\,\mathtt{T} + \beta_2\,\mathtt{W} + \beta_3\,\mathtt{S} \tag{73}$$

gespeichert, wobei hier alle drei Covariablen marginal signifikant sind:

```
> summary( oz0I.lm)
....
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -64.34208   23.05472  -2.791  0.00623 **
T             1.65209    0.25353   6.516 2.42e-09 ***
W            -3.33359    0.65441  -5.094 1.52e-06 ***
S             0.05982    0.02319   2.580  0.01124 *
....
```

10.12.1.1 ANOVA für den Vergleich hierarchischer Modelle

Die Funktion anova() erlaubt den Vergleich mehrerer, sich nicht nur in einem Term (d. h. einer Parameterdimension) unterscheidender Submodelle. Dabei ist es üblich, die Modelle in der Reihenfolge zunehmender Parameterdimension aufzuführen. Hier werden in der hierarchischen Sequenz oz0I.lm $\subset$ oz2I.lm $\subset$ oz3I.lm der obigen drei Modelle die „nebeneinanderliegenden“ paarweise miteinander verglichen und die Ergebnisse in einer Varianzanalysetabelle (ANOVA-Tabelle) zusammengefasst:

```
> anova( oz0I.lm, oz2I.lm, oz3I.lm)
Analysis of Variance Table

Model 1: O ~ T + W + S
Model 2: O ~ T + W + S + T:W + T:S + W:S
Model 3: O ~ T * W * S
  Res.Df   RSS  Df Sum of Sq      F    Pr(>F)
1    107 48003
2    104 38205   3      9797 8.8592 2.809e-05 ***
3    103 37969   1       236 0.6401    0.4255
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Diese ANOVA-Tabelle hat für jedes Modell eine Zeile, die in ihrer ersten Spalte jeweils die Nummer des Modells enthält, das oberhalb der Tabelle beschrieben steht. Die Spalten Res.Df und RSS enthalten die Residuenfreiheitsgrade (wobei Df = “degrees of freedom”) bzw. die RSS des

Modells. Ab der zweiten Zeile stehen in den Spalten `Df`, `Sum of Sq`, `F` und `Pr(>F)` die Informationen über den Vergleich des Modells der jeweiligen Zeile mit dem Modell der Zeile darüber.

Interpretation der Ergebnisse:

- In Zeile `2` der ANOVA-Tabelle wird `Model 2` (= Modell `oz2I.lm` = (72)) mit `Model 1` (= Modell `oz0I.lm` = (73)) verglichen: Ihre Parameterräume unterscheiden sich durch die Koeffizienten der Terme `T:W`, `T:S` und `W:S`, was zu einer Dimensionsdifferenz von 3 führt (Spalte `Df`).

  Im Test der dazugehörigen Hypothese $H_0 : \beta_4 = \beta_5 = \beta_6 = 0$ (im Modell (72)) berechnet sich die Teststatistik (bemerkenswerterweise – siehe das „Beachte“ unten) gemäß

$$\begin{aligned} T(\mathbf{y}) &= \frac{(\mathrm{RSS}(\texttt{oz0I.lm}) - \mathrm{RSS}(\texttt{oz2I.lm}))/(n - \dim(\texttt{oz0I.lm}) - (n - \dim(\texttt{oz2I.lm})))}{\mathrm{RSS}(\texttt{oz3I.lm})/(n - \dim(\texttt{oz3I.lm}))} \\ &= \frac{(48003 - 38205)/(111 - 3 - (111 - 6))}{37969/(111 - 8)} \doteq 8.859 \end{aligned}$$

  Beachte: Beim paarweisen Vergleich der Modelle haben die verwendeten Teststatistiken alle *denselben* Nenner, nämlich den Varianzschätzer $\mathrm{RSS}/(n - \dim(\boldsymbol{\beta}))$ des „obersten“ (also größten) Modells in der Sequenz (hier also von von `oz3I.lm`). Dies ist nicht konform mit der in (70) aufgeführten Teststatistik für geschachtelte Hypothesen! Allerdings kann man mit einem gewissen Aufwand zeigen, dass in geschachtelten Modellen die F-Verteilungseigenschaft der obigen Teststatistik – mit dem zum größten Modell in der Sequenz gehörenden, sprich „falschen“ Nenner – auch gegeben ist. Dies folgt aus einer Verallgemeinerung von Cochrans Theorem (siehe z. B. [46, Hocking (1996), Thm. 2.7] oder [70, Searle (1971), p. 64 bzw. p. 60]), die die paarweise Unabhängigkeit und χ^2-Verteilung gewisser quadratischer Formen garantiert. Allerdings hat die Verwendung der obigen Teststatistik zur Konsequenz, dass das Verfahren eine etwas geringere Güte/Power hat als jenes, welches die Teststatistik mit variierendem Nenner verwendet, also mit (70) konform ist; siehe z. B. [41, Hagemann (2012)].

  Der Wert der Teststatistik liefert einen p-Wert von `Pr(>F)` $= 2.809 \times 10^{-05}$ und somit einen signifikanten Unterschied zwischen den beiden Modellen.

  Fazit: In Modell (72) können nicht alle der obigen drei βs gleichzeitig Null sein. Mit anderen Worten: Mindestens einer der drei Zweifach-Interaktionsterme hat einen signifikanten Einfluss auf die Response.

- In Zeile `3` steht das Ergebnis des Vergleichs von Modell (71) (= `oz3I.lm`) mit Modell (72) (= `oz2I.lm`): Ihre Parameterräume unterscheiden durch den Term `T:W:S`. Dies begründet die Differenz von 1 in den Parameterdimensionen (Spalte `Df`: 1).

  Der Test für diesen Term, d. h., der Test der Hypothese $H_0 : \beta_7 = 0$ (im Modell (71)) liefert bei einem Wert der F-Teststatistik von

$$\begin{aligned} T(\mathbf{y}) &= \frac{(\mathrm{RSS}(\texttt{oz2I.lm}) - \mathrm{RSS}(\texttt{oz3I.lm}))/(n - \dim(\texttt{oz2I.lm}) - (n - \dim(\texttt{oz3I.lm})))}{\mathrm{RSS}(\texttt{oz3I.lm})/(n - \dim(\texttt{oz3I.lm}))} \\ &= \frac{(38205 - 37969)/(111 - 7 - (111 - 8))}{37969/(111 - 8)} \doteq 0.6401 \end{aligned}$$

  kein signifikantes Ergebnis, denn: p-Wert `Pr(>F)` $= 0.4255$. (Dies ist hier natürlich dasselbe Resultat, wie beim t-Test der Hypothese H_0.)

  Fazit: Der Dreifach-Interaktionsterm `T:W:S` kann aus Modell (71) eliminiert werden (was wir aber schon durch den marginalen Test wussten).

Der direkte Vergleich des vollen Modells mit unserem bisher kleinsten Submodell (Dimensionsdifferenz = 4) ist natürlich ebenfalls möglich:

```
> anova( oz0I.lm, oz3I.lm)
Analysis of Variance Table

Model 1: O ~ T + W + S
Model 2: O ~ T * W * S
  Res.Df   RSS  Df Sum of Sq      F    Pr(>F)
1    107 48003
2    103 37969   4     10033 6.8045 6.701e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Fazit: Offenbar sind die beiden Modelle signifikant voneinander verschieden, aber an welchem (oder welchen) der vier Interaktionsterme es nun liegt, ist so nicht erkennbar.

10.12.1.2 Sequenzielle ANOVA für die Terme eines Modells

Wenn die Reihenfolge der Terme in der Modellformel eines `lm`-Objektes eine interpretierbare Rolle spielt (eventuell wie in unseren obigen Beispielen, in denen die Terme „von links nach rechts“ immer höhere Interaktionsordnungen darstellen), *kann* das folgende Vorgehen bei der Modellanalyse behilflich sein:

Wird die Funktion `anova()` mit nur *einem* `lm`-Objekt als Argument aufgerufen, so fittet sie eine Sequenz von aufsteigenden Submodellen, indem sie beim Null-Modell beginnt und in der Reihenfolge der Terme in der Modellformel des `lm`-Objektes sukzessive einen Term nach dem anderen hinzufügt. Für je zwei aufeinanderfolgende Submodelle führt sie dabei den F-Test auf Signifikanz des hinzugekommenen Terms durch. Die Resultate werden in einer ANOVA-Tabelle zusammengefasst ausgegeben. Im Fall unseres Modells in `oz3I.lm` erhalten wir das Folgende:

```
> anova( oz3I.lm)
Analysis of Variance Table

Response: O
           Df Sum Sq Mean Sq  F value    Pr(>F)
T           1  59434   59434 161.2288 < 2.2e-16 ***
W           1  11378   11378  30.8666 2.176e-07 ***
S           1   2986    2986   8.1006  0.005341 **
T:W         1   7227    7227  19.6048 2.377e-05 ***
T:S         1   2141    2141   5.8081  0.017726 *
W:S         1    429     429   1.1649  0.282976
T:W:S       1    236     236   0.6401  0.425510
Residuals 103  37969     369
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Wir erkennen, dass nach dem fünften Term (`T:S`) kein weiterer Term mehr einen (marginal) signifikanten Beitrag liefert. Der p-Wert des zuletzt hinzugekommenen Terms `T:W:S` ist natürlich gleich dem des marginalen t-Tests für diesen Term.

Beachte:

- Auch hier steht stets derselbe Nenner in den F-Teststatistiken, nämlich der des größten/vollen Modells, also RSS(`oz3I.lm`)/($n - \dim$(`oz3I.lm`)) (was im „Beachte“ auf Seite 280 begründet wird).

- Im Allgemeinen sind diese Resultate gegenüber einer Vertauschung der Reihenfolge der Terme in der Modellformel **nicht invariant**!

Obiges Resultat legt nahe, ein Modell zu fitten, in dem nur die Terme `T`, `W`, `S`, `T:W` und `T:S` auftreten, also das Modell

$$\mathbb{E}[\mathtt{O}] \;=\; \beta_0 + \beta_1\,\mathtt{T} + \beta_2\,\mathtt{W} + \beta_3\,\mathtt{S} + \beta_4\,\mathtt{T}\cdot\mathtt{W} + \beta_5\,\mathtt{T}\cdot\mathtt{S}, \tag{74}$$

z. B. durch:

```
> summary( oz2Ia.lm <- update( oz2I.lm, ~ . - W:S))
....
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.368e+02  6.414e+01  -2.133 0.035252 *
T            2.451e+00  8.250e-01   2.971 0.003678 **
W            1.115e+01  4.259e+00   2.617 0.010182 *
S           -3.531e-01  1.750e-01  -2.018 0.046184 *
T:W         -1.863e-01  5.425e-02  -3.434 0.000852 ***
T:S          5.717e-03  2.370e-03   2.412 0.017589 *
....
```

Wir stellen fest, dass hier alle Terme marginal signifikant sind und dass ferner der direkte Vergleich dieses Modells (74) mit dem vollen Modell (71) keinen signifikanten Unterschied zwischen den beiden liefert:

```
> anova( oz2Ia.lm, oz3I.lm)
Analysis of Variance Table

Model 1: O ~ T + W + S + T:W + T:S
Model 2: O ~ T * W * S
  Res.Df   RSS  Df Sum of Sq      F Pr(>F)
1    105 38635
2    103 37969   2       665 0.9025 0.4087
```

Fazit: Das im Vergleich zum vollen Modell (71) in `oz3I.lm` einfachere Modell (74) in `oz2Ia.lm` scheint eine statistisch adäquate Beschreibung der Regressionsbeziehung zwischen Ozon, Temperatur, Wind und Strahlung zu sein und die Bedingung „so einfach wie möglich und so komplex wie (statistisch) nötig“ zu erfüllen.

10.12.2 Stetige und Faktor-Covariablen

Im Fall eines Modells, das sowohl stetige Covariablen als auch Faktor-Covariablen enthält, ist die Vorgehensweise völlig analog zu der im vorherigen Abschnitt. Dies erlaubt die Beurteilung, ob eine Faktor-Covariable (sozusagen als Ganzes und nicht nur einer ihrer levelspezifischen Koeffizienten) einen signifikanten Beitrag liefert. Wir fitten als Anschauungsmaterial auf Vorrat gleich drei hierarchische Modelle für den bereits aus Abschnitt 10.11 bekannten Datensatz in `car.test.frame` (und zwar unter Verwendung der voreingestellten Treatment-Kontraste):

```
> data( car.test.frame, package = "rpart")
> miles.lm <- lm( Mileage ~ Weight, data = car.test.frame)
> miles1.lm <- update( miles.lm, ~ . + Type)
> summary( miles2.lm <- update( miles1.lm, ~ . + Type:Weight))
....
```

```
Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)        32.657208   9.475664   3.446  0.00119 **
Weight             -0.003022   0.003353  -0.901  0.37206
TypeLarge          15.653276  20.459640   0.765  0.44797
TypeMedium        -13.514109  12.040917  -1.122  0.26730
TypeSmall          18.964995  11.668288   1.625  0.11064
TypeSporty         25.451309  11.119165   2.289  0.02652 *
TypeVan            -9.272252  15.496060  -0.598  0.55241
Weight:TypeLarge   -0.004588   0.005955  -0.770  0.44481
Weight:TypeMedium   0.003843   0.004076   0.943  0.35047
Weight:TypeSmall   -0.006113   0.004503  -1.358  0.18094
Weight:TypeSporty  -0.008450   0.003937  -2.146  0.03694 *
Weight:TypeVan      0.001734   0.004832   0.359  0.72123
....
```

Die marginalen p-Werte bieten ein sehr heterogenes Bild: Zwar weicht zum Beispiel der Fahrzeugtyp `Sporty` signifikant von der `Mileage`-`Weight`-Beziehung der Bezugsgruppe `Compact` ab, aber keiner der anderen Typen. Ist der Interaktionsterm *als Ganzes* denn trotzdem notwendig?

Das Resultat der Funktion `anova()`, angewandt auf die Sequenz der hierarchischen Modelle, ermöglicht die Beurteilung der Beiträge der Modellterme:

```
> anova( miles.lm, miles1.lm, miles2.lm)
Analysis of Variance Table

Model 1: Mileage ~ Weight
Model 2: Mileage ~ Weight + Type
Model 3: Mileage ~ Weight + Type + Weight:Type
  Res.Df    RSS Df Sum of Sq      F   Pr(>F)
1     58 380.83
2     53 302.98  5     77.85 3.4663 0.009376 **
3     48 215.61  5     87.37 3.8899 0.004858 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Fazit: Jeder paarweise Vergleich liefert einen signifikanten Unterschied (und auch der hier nicht gezeigte, direkte Vergleich der Modelle `miles.lm` und `miles2.lm`). Damit wird deutlich, dass sowohl die `Type`-Koeffizienten als Ganzes als auch die `Type:Weight`-Interaktionsterme als Ganzes jeweils einen signifikanten Beitrag liefern und im Modell vertreten sein sollten.

Zu erkennen ist auch, dass die Faktor-Covariable `Type` im Modell `miles1.lm` wegen der Reparametrisierung ihrer sechs Levels für fünf (!) Parameterdimensionen verantwortlich ist (Spalte `Df`). Dasselbe gilt auch nochmal für den `Type`-`Weight`-Interaktionsterm `Type:Weight` im Modell `miles2.lm`.

Bemerkungen:

- Die gewählte Reparametrisierung, also die Art der Kontraste darf diese Testergebnisse natürlich nicht beeinflussen. Um dies zu überprüfen, führen Sie (zur Übung) die obige Analyse unter Verwendung der Helmert-Kontraste durch.
- Eine weitere Thematik, die im Zusammenhang mit dem F-Test zu besprechen sein könnte, ist die der Definition sowie der Gemeinsamkeiten und Unterschiede der sogenannten Typ

I-, II- und III-Tests, worauf wir hier aber nicht eingehen. Stattdessen sei z. B. auf [78, Venables (2000)], §8.6.2 in [33, Fox (1997)] oder ch. 4.2 in [34, Fox (2002)] verwiesen, worin die Thematik – z. T. sehr kritisch – diskutiert wird.

- Nützlich könnte auch ein Vergleich der oben besprochenen Funktion `anova()` mit der Funktion `Anova()` des **R**-Paketes `car` sein.
- Des Weiteren könnten die Pakete `lmtest` und `strucchange` (siehe auch [85, Zeileis et al. (2002)]) sehr interessant sein, um weitere und andere Tests im Rahmen linearer Modelle durchzuführen, als das wären z. B. diagnostische Tests auf Heteroskedastizität, serielle Korrelation der Fehler, Strukturbrüche in den Regressionskoeffizienten, Nichtlinearität, funktionelle Fehlspezifizierung oder fehlende Kovariablen.

11 Einführung in die Varianzanalyse

In der klassischen Varianzanalyse ("analysis of variance" = ANOVA) hat man es typischerweise mit Daten aus einem geplanten Experiment zu tun, in dem die Abhängigkeit einer metrischen Response von höchstens ordinal skalierten (also diskreten) Designvariablen untersucht wird. Genauer: Die Response wird als potenziell abhängig von den Levels bzw. Levelkombinationen der Designvariablen angesehen. Die Designvariablen werden in diesem Zusammenhang *Faktorvariablen* oder kurz *Faktoren* (auch *Behandlungen*, Englisch: "treatments") genannt. In erster Linie ist es das Ziel, zu untersuchen, ob die Levels der Faktoren (die Behandlungsstufen) einen signifikanten Einfluss auf die mittlere Response haben, d. h., welcher der Modellterme (als Ganzes) einen relevanten *Beitrag zur Erklärung der Gesamtvariation in der Response* leistet. Erst in zweiter Linie ist die mittlere Response, genauer ihr Erwartungswert für jeden Level oder jede Levelkombination zu schätzen.

11.1 Die einfaktorielle Varianzanalyse ("One-way-ANOVA")

Der einfachste Fall liegt vor, wenn nur ein Faktor als Einflussvariable für eine metrische Response betrachtet wird und auf jedem seiner, sagen wir, $L \geq 2$ Levels die Response an n_l unabhängigen Untersuchungseinheiten jeweils *genau einmal pro* Untersuchungseinheit gemessen wird, mit $l = 1, \ldots, L$. Ist jedes $n_l \geq 1$ und werden die Untersuchungseinheiten den Faktorlevels, also den Behandlungen zufällig (*randomisiert*) zugewiesen, haben wir es mit der einfaktoriellen Varianzanalyse (oder dem Ein-Faktor-Modell) für einen vollständigen, randomisierten Versuchsplan zu tun. Die n_l werden auch Zellhäufigkeiten genannt; sind sie alle gleich ($n_l \equiv n$), so handelt es sich um einen balancierten Versuchsplan, anderenfalls um einen unbalancierten Versuchsplan.

11.1.1 "cell means"-Modell, Inferenz und ANOVA-Tabelle

Formal lässt sich dies alles wie folgt als das sogenannte "cell means"-Modell schreiben:

$$Y_{li} = \mu_l + \varepsilon_{li} \quad \text{mit} \quad \varepsilon_{li} \text{ i.i.d.} \sim \mathcal{N}(0, \sigma^2) \quad \text{für } i = 1, \ldots, n_l \text{ und } l = 1, \ldots, L, \tag{75}$$

wobei Y_{li} die Response der Untersuchungseinheit i auf Faktorlevel l und μ_l der Erwartungswert der Response auf diesem Faktorlevel l ist sowie die „Fehler" ε_{li} die individuellen Abweichungen von μ_l sind. Insgesamt liegen also $N := \sum_{l=1}^{L} n_l$ Beobachtungen vor. Dieser Sachverhalt kann natürlich auch in der Matrixnotation der linearen Modelle formuliert werden:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\mu} + \boldsymbol{\varepsilon}, \qquad \text{wobei}$$

$$\mathbf{Y} = \begin{pmatrix} Y_{11} \\ \vdots \\ Y_{1n_1} \\ Y_{21} \\ \vdots \\ Y_{2n_2} \\ \vdots \\ Y_{L1} \\ \vdots \\ Y_{Ln_L} \end{pmatrix}, \; \mathbf{X} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}}_{N \times L}, \; \boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_L \end{pmatrix}, \; \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_{11} \\ \vdots \\ \varepsilon_{1n_1} \\ \varepsilon_{21} \\ \vdots \\ \varepsilon_{2n_2} \\ \vdots \\ \varepsilon_{L1} \\ \vdots \\ \varepsilon_{Ln_L} \end{pmatrix} \sim \mathcal{N}_N\left(\mathbf{0}, \sigma^2 \mathbf{I}_{N \times N}\right)$$

Bemerkung: Im Prinzip könnte die Analyse nun mittels der bereits bekannten Werkzeuge der linearen Regression durchgeführt werden. Allerdings stehen für die ANOVA „maßgeschneiderte" Verfahren und **R**-Funktionen zur Verfügung, die explorative Darstellungen der Daten sowie der

inferenzstatistischen Resultate liefern, welche für das Modell und die zentrale Fragestellung der Varianzzerlegung adäquater sind.

Parameterschätzung und Hypothesentest: Die Kleinste-Quadrate-Schätzer (KQS) $\hat{\mu}_1, \ldots, \hat{\mu}_L$ für $\mu_1, \ldots, \mu_L$ in obigem Modell (75) sind die Lösung des Problems

$$\boldsymbol{\varepsilon}'\boldsymbol{\varepsilon} \equiv \sum_{l=1}^{L}\sum_{i=1}^{n_l}(Y_{li} - \mu_l)^2 \stackrel{!}{=} \text{minimal in } \mu_1, \ldots, \mu_L$$

Dies liefert $\hat{\mu}_l = \frac{1}{n_l}\sum_{i=1}^{n_l} Y_{li} =: \bar{Y}_{l\cdot}$ für $l = 1, \ldots, L$ und $\hat{\varepsilon}_{li} := Y_{li} - \bar{Y}_{l\cdot}$ als Residuen. Die Residuenquadratesumme (= "residual sum of squares" = RSS) ergibt sich also zu

$$\text{RSS} = \sum_{l=1}^{L}\sum_{i=1}^{n_l}(Y_{li} - \bar{Y}_{l\cdot})^2 \tag{76}$$

Bemerkung: Diese Resultate erhält man natürlich auch, wenn $\hat{\boldsymbol{\mu}} = (\hat{\mu}_1, \ldots, \hat{\mu}_L)'$ gemäß $\hat{\boldsymbol{\mu}} := (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$ berechnet wird und die RSS gemäß RSS $:= \hat{\boldsymbol{\varepsilon}}'\hat{\boldsymbol{\varepsilon}}$ mit $\hat{\boldsymbol{\varepsilon}} := \mathbf{Y} - \hat{\mathbf{Y}}$ und $\hat{\mathbf{Y}} := \mathbf{X}\hat{\boldsymbol{\mu}}$ wie in Abschnitt 10.1 auf Seite 197.

Die Hypothese, dass der Faktor keinen Einfluss hat, lautet formal $H_0 : \mu_1 = \ldots = \mu_L$ und ist sowohl äquivalent zu $H_{0,1} : \mu_1 - \mu_L = \ldots = \mu_{L-1} - \mu_L = 0$ als auch zu $H_{0,2} : \mu_l - \bar{\mu}_\cdot = 0, l = 1, \ldots, L-1$. Insbesondere $H_{0,1}$ lässt sich recht übersichtlich als lineare Bedingung $\mathbf{C}\boldsymbol{\mu} = \mathbf{0}$ an den Parameter*vektor* $\boldsymbol{\mu}$ formulieren, und zwar mit

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & \cdots & 0 & -1 \\ 0 & 1 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & -1 \\ 0 & \cdots & 0 & 1 & -1 \end{pmatrix}_{(L-1)\times L}, \quad \text{wobei} \quad \text{Rang}(\mathbf{C}) = L - 1$$

Der Rang der Matrix $\mathbf{C}$ zu obiger Hypothese $H_{0,1} (= H_{0,2} = H_0)$ wird auch Anzahl der <u>Freiheitsgrade des Faktors</u> (bzw. <u>der Hypothese</u>) genannt.

Unter H_0 lautet das Modell offenbar $Y_{li} = \mu_0 + \varepsilon_{li}$ und der KQS $\hat{\mu}_0$ für μ_0 wird als Lösung von

$$\sum_{l=1}^{L}\sum_{i=1}^{n_l}(Y_{li} - \mu_0)^2 \stackrel{!}{=} \text{minimal in } \mu_0$$

ermittelt, was $\hat{\mu}_0 = \frac{1}{N}\sum_{l=1}^{L}\sum_{i=1}^{n_l} Y_{li} =: \bar{Y}_{\cdot\cdot}$ (= "overall mean") ergibt. Die Residuenquadratesumme unter H_0 (kurz und hier aus Bequemlichkeit ohne den Index 0: RSS_H) ist

$$\text{RSS}_H = \sum_{l=1}^{L}\sum_{i=1}^{n_l}(Y_{li} - \bar{Y}_{\cdot\cdot})^2 \tag{77}$$

(Auch hier gilt obige Bemerkung unter (76) mit dem Hinweis auf Abschnitt 10.1 auf Seite 197.)

Die Theorie der linearen Modelle liefert einen F-Test für H_0, denn (wie in Abschnitt 10.1 auf Seite 199 mit $\boldsymbol{\mu} = \boldsymbol{\beta}$, $\hat{\boldsymbol{\mu}} = \hat{\boldsymbol{\beta}}$ und $\mathbf{c}_0 = \mathbf{0}$):

$$T(\mathbf{Y}) = \frac{(\text{RSS}_H - \text{RSS})/\text{Rang}(\mathbf{C})}{\text{RSS}/(N - \dim(\boldsymbol{\mu}))} \sim F_{\text{Rang}(\mathbf{C}), N-\dim(\boldsymbol{\mu})} \quad \text{unter } H_0 \tag{78}$$

Bemerkung: Offenbar dürfen hierfür nicht alle $n_l \equiv n = 1$ sein, da sonst $N - \dim(\boldsymbol{\mu}) = Ln - L = 0$ ist. Es muss also mindestens ein $n_l \geq 2$ sein.

Zur Bestimmung von $\mathrm{RSS}_H - \mathrm{RSS}$ beachte, dass $Y_{li} - \bar{Y}_{..} = Y_{li} - \bar{Y}_{l.} + \bar{Y}_{l.} - \bar{Y}_{..}$ ist und sich nach Quadrieren dieser Gleichung beim Summieren über alle Indices l und i die gemischten Produkte eliminieren, sodass aus (77) mit (76) die orthogonale Zerlegung der RSS_H folgt:

$$\mathrm{RSS}_H = \sum_{l=1}^{L}\sum_{i=1}^{n_l} \hat{\varepsilon}_{li}^2 + \sum_{l=1}^{L}\sum_{i=1}^{n_l} (\bar{Y}_{l.} - \bar{Y}_{..})^2 = \mathrm{RSS} + \sum_{l=1}^{L} n_l (\bar{Y}_{l.} - \bar{Y}_{..})^2 \tag{79}$$

Dies liefert offenbar $\mathrm{RSS}_H - \mathrm{RSS}$ als Summe der gewichteten Abweichungsquadrate der Faktorlevel-Mittelwerte vom Gesamtmittel:

$$\mathrm{RSS}_H - \mathrm{RSS} = \sum_{l=1}^{L} n_l (\bar{Y}_{l.} - \bar{Y}_{..})^2 \tag{80}$$

(was sich auch wie in Abschnitt 10.1 auf Seite 199 mit $\boldsymbol{\mu} = \boldsymbol{\beta}$, $\hat{\boldsymbol{\mu}} = \hat{\boldsymbol{\beta}}$, geeignetem $\mathbf{C}$ und $\mathbf{c}_0 = \mathbf{0}$ aus der dortigen Definition von $T(\mathbf{Y})$ errechnet).

Die obigen Größen werden oft in einer ANOVA-Tabelle zusammengefasst dargestellt:

Quelle der Streuung (source of variation)	Freiheitsgrade (degrees of freedom)	Summe der Abweichungsquadrate (sums of squares)	Mittlere Abweichungsquadratesumme (mean squares)	F-Statistik ($\sim F_{L-1,N-L}$ unter H_0)
Zwischen den Faktorstufen (between treatments)	$L-1$ $= \mathrm{Rang}(\mathbf{C})$	$\mathrm{RSS}_H - \mathrm{RSS} = \sum_{l=1}^{L}\sum_{i=1}^{n_l}(\bar{Y}_{l.} - \bar{Y}_{..})^2$	$S_H^2 = \dfrac{\mathrm{RSS}_H - \mathrm{RSS}}{L-1}$	$F = \dfrac{S_H^2}{S^2}$
Innerhalb der Faktorstufen (within treatments, error, residuals)	$N-L$ $= \dim(\mathbf{Y}) - \dim(\boldsymbol{\mu})$	$\mathrm{RSS} = \sum_{l=1}^{L}\sum_{i=1}^{n_l}(Y_{li} - \bar{Y}_{l.})^2$	$S^2 = \dfrac{\mathrm{RSS}}{N-L}$	
Gesamtstreuung (total variation)	$N-1$	$\mathrm{RSS}_H = \sum_{l=1}^{L}\sum_{i=1}^{n_l}(Y_{li} - \bar{Y}_{..})^2$		

Bemerkung: Gemäß der Theorie der linearen Modelle ist mit $\mu^* = \sum_{k=1}^{L} \frac{n_k}{N}\mu_k$

$$\mathbb{E}\left[\frac{\mathrm{RSS}}{N-L}\right] = \sigma^2 \quad \text{und} \quad \mathbb{E}\left[\frac{\mathrm{RSS}_H - \mathrm{RSS}}{L-1}\right] = \sigma^2 + \frac{1}{L-1}\sum_{l=1}^{L} n_l(\mu_l - \mu^*)^2,$$

was die „Funktionsweise" der obigen Teststatistik S_H^2/S^2 noch etwas veranschaulicht: Sie ist der relative Zuwachs an Reststreuung, wenn man zum restriktiveren Modell unter H_0 übergeht, das die Daten in ein „Korsett" mit gleichen treatment/cell means „presst", ihnen also weniger Freiheiten lässt. Und wenn H_0 stimmt, hat diese Teststatistik S_H^2/S^2 eine $F_{L-1,N-L}$-Verteilung und somit den Erwartungswert $(N-L)/(N-L-2)$ (falls $N-L > 2$), der offenbar ≈ 1 ist.

11.1.2 Explorative Datenanalyse und konkrete Durchführung der Varianzanalyse mit `aov()`

Zur Illustration verwenden wir als **Beispiel** nebenstehenden Datensatz (leicht modifiziert aus Box, Hunter und Hunter (1978) entnommen), der Koagulationszeiten (= Blutgerinnungszeiten) in Abhängigkeit von vier verschiedenen Diäten A, B, C und D enthält.

Diät	Koagulationszeiten							
A	62	60	63	59				
B	63	67	71	64	65	66	66	
C	68	66	71	67	68	68		
D	56	62	60	61	63	64	63	59

Hier hat also der Faktor „Diät“ die vier Levels A, B, C und D und die metrische Response beinhaltet Zeitdauern. (Offenbar ist es ein unbalancierter Versuchsplan.)

Es ist in **R** – wie von den Regressionsmodellen her bereits bekannt – nicht nötig, die in obiger Matrixnotation verwendete Designmatrix $\mathbf{X}$ zu spezifizieren. Vielmehr müssen der Response-Vektor $\mathbf{Y}$ und der Vektor der zu den $\mathbf{Y}$-Elementen jeweils gehörigen Faktorlevels als Spalten in einem Data Frame zusammengestellt werden. Es ist also darauf zu achten, dass die im Experiment geltende Zuordnung von beobachteten Response-Werten und jeweiligem Faktorlevel in einer jeden Zeile des Data Frames korrekt wiedergegeben wird:

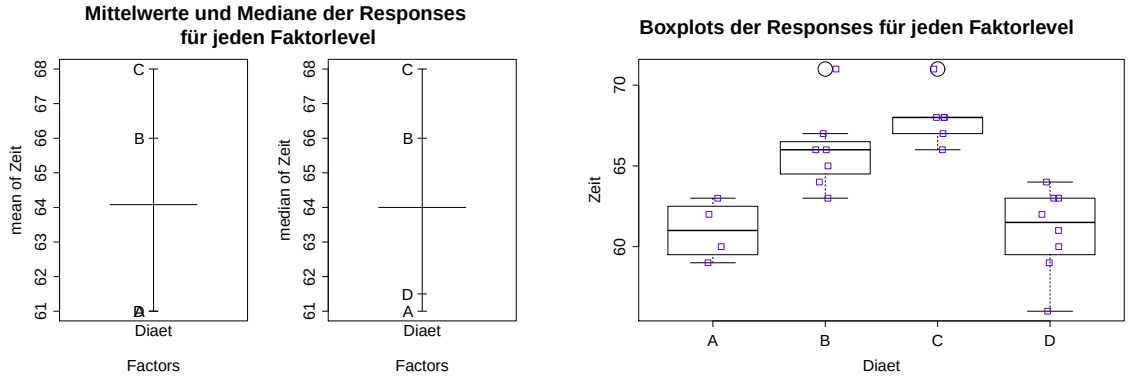
```
> (Koag.df <- data.frame( Diaet = factor( rep( LETTERS[ 1:4], c( 4, 7, 6, 8))),
+                         Zeit = c( 62, 60, 63, 59,
+                                   63, 67, 71, 64, 65, 66, 66,
+                                   68, 66, 71, 67, 68, 68,
+                                   56, 62, 60, 61, 63, 64, 63, 59)) )
   Diaet Zeit
 1     A   62
 2     A   60
 3     A   63
....
24     D   63
25     D   59
```

Bevor die eigentliche Varianzanalyse durchgeführt wird, sollte man sich die Daten erst einmal grafisch veranschaulichen und überprüfen, ob etwas gegen die Modellvoraussetzungen der Normalverteiltheit und Varianzkonstanz (= „Homoskedastizität“) der Fehler spricht. Dies kann mit den beiden im Folgenden beschriebenen Funktionen geschehen:

Einfaktorielle Varianzanalyse: Explorative Datenanalyse	
`> plot.design( Koag.df)`	Liefert einen Plot, in dem sowohl für jeden Faktorlevel (voreinstellungsgemäß) das arithmetische Mittel der zugehörigen Responses durch einen kurzen waagrechten Strich (mit dem Label des jeweiligen Levels) als auch das Gesamtmittel der Responses durch einen längeren waagrechten Strich markiert sind. Dies erlaubt eine erste Einschätzung, inwieweit die Levels die mittlere Response beeinflussen (linker Plot auf Seite 289).
`> plot.design( Koag.df,` `+ fun = median)`	Wie eben, aber das Argument `fun = median` veranlasst, dass Mediane als Lageparameter verwendet und markiert werden. Weicht dieser Plot von demjenigen mit den arithmetischen Mitteln stark ab, ist das ein Hinweis darauf, dass Asymmetrie in den Verteilungen der Responses bzw. Ausreißer in den Responses vorliegen (mittlerer Plot auf Seite 289).

`> plot( Zeit ~ Diaet,`
`+ data = Koag.df)`
`> stripchart( Zeit ~ Diaet,`
`+ data = Koag.df,`
`+ add = TRUE, col = "blue",`
`+ method = "jitter",`
`+ vertical = TRUE)`
	Erzeugt einen Plot, in dem für jeden Faktorlevel ein Boxplot der zugehörigen Responses gezeichnet ist. Dies ermöglicht eine Beurteilung sowohl der Normalverteiltheit und der Varianzhomogenität der ε_i über alle Levels hinweg als auch der Abhängigkeit der mittleren Responses von den Faktorlevels (rechter Plot auf Seite 289). Hier sind zusätzlich die Rohdaten (horizontal leicht verwackelt) überlagert.

Im vorliegenden Beispiel sprechen die Ergebnisse der EDA nicht gegen die Modellannahmen: Weder legt der Vergleich der Designplots links die Existenz von potenziellen Ausreißern noch Asymmetrie nahe, noch liefern die Faktor-Boxplots rechts deutliche Indizien gegen die Annahme der Varianzkonstanz zwischen den Faktorlevels oder gegen die Normalverteilung. (Eine belastbarere Modelldiagnose liefert §11.1.4.)



Die Varianzanalyse wird mit dem Aufruf `aov( formula, data)` durchgeführt (wobei `aov` für “analysis of variance” steht). Das Argument `formula` spezifiziert in der bereits bekannten Formelsyntax die Beziehung zwischen Response und Design und `data` den Data Frame, aus dem die Variablen entnommen werden. Das Resultat ist ein `aov`-Objekt. (Faktisch ist `aov()` allerdings nur eine “wrapper”-Funktion für einen geeigneten Aufruf von `lm()`, mit dem das entsprechende lineare Modell gefittet wird. Aus diesem Grund ist das Resultat von `aov()` genauer ein Objekt der Klasse `c("aov", "lm")`.) Die Erstellung der Ergebnissetabelle (ANOVA-Tabelle) wird durch die Anwendung von `summary()` auf das `aov`-Objekt veranlasst:

Einfaktorielle Varianzanalyse: `aov()`

```
> (Koag.aov <- aov( Zeit ~ Diaet,
+ data = Koag.df))
Call: aov(formula = Zeit ~ Diaet,
          data = Koag.df)

Terms:
                 Diaet Residuals
Sum of Squares  231.84    112.00
Deg. of Freedom      3        21

Residual standard error: 2.309401
Estimated effects may be unbalanced
```

Hier werden das einfaktorielle Modell von `Zeit` an `Diaet` (aus Data Frame `Koag.df`) gefittet, in `Koag.aov` abgelegt und einige Teilergebnisse ausgegeben. Die ANOVA-Tabelle und damit ausführlichere Informationen liefert die Funktion `summary()` (siehe unten), weswegen wir uns hier kurz fassen. Beachte die Meldung in der letzten Zeile der Ausgabe, die darauf hinweist, dass es sich um einen unbalancierten Versuchsplan handeln könnte, was hier auch der Fall ist (aber auch eine numerische Ursache aufgrund digitaler Rundungsprobleme haben kann). (Diese Warnung wird in der `summary()`-Ausgabe *nicht* geliefert.)

Die ANOVA-Tabelle

```
> summary( Koag.aov)     # oder: anova( Koag.aov)
            Df  Sum Sq Mean Sq F value    Pr(>F)
Diaet        3 231.840  77.280   14.49 2.435e-05 ***
Residuals   21 112.000   5.333
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Mit `summary( Koag.aov)` erhält man die ANOVA-Tabelle (`anova( Koag.aov)` liefert fast dieselbe Ausgabe). In deren Zeile `Diaet` stehen die $L-1$ Freiheitsgrade dieses Faktors (Spalte `Df`, “degrees of freedom”), die Summe der Abweichungsquadrate der Faktorlevel-Mittelwerte vom Gesamtmittel, also $\mathrm{RSS}_H - \mathrm{RSS}$ (Spalte `Sum Sq`), und deren mittlere Quadratesumme $(\mathrm{RSS}_H - \mathrm{RSS})/(L-1)$ (Spalte `Mean Sq`). In der Zeile `Residuals` stehen für die Residuen ihre $N-L$ Freiheitsgrade, ihre Quadratesumme RSS und ihre mittlere Quadratesumme $\mathrm{RSS}/(N-L)$. Außerdem wird der Wert der F-Teststatistik (Spalte `F value`) und der p-Wert (Spalte `Pr(>F)`) des Tests auf Gleichheit der mittleren Faktorlevel-Responses angegeben, also des Tests der Hypothese $H_0 : \mu_1 = \cdots = \mu_L$ (die hier offenbar verworfen werden kann).

11.1.3 Parametrisierung als Faktoreffekte-Modell und Parameterschätzer mit `model.tables()`

Zur Bestimmung der KQS $\hat{\mu}_1, \ldots, \hat{\mu}_L$ für $\mu_1, \ldots, \mu_L$ ist zu sagen, dass sich das in (75) auf Seite 285 formulierte cell means-Modell auch anders parametrisieren lässt, und zwar als Faktoreffekte-Modell. Darin wird ein Gesamtmittel (oder auch „Populationsmittel“, besser vielleicht: „globaler“ Erwartungswert) μ_0 für die Response angesetzt, von dem sich die Erwartungswerte der Responses der verschiedenen Faktorlevels durch additive Faktoreffekte α_l unterscheiden:

$$Y_{li} = \mu_0 + \alpha_l + \varepsilon_{li} \qquad \text{für } i = 1, \ldots, n_l \quad \text{und} \quad l = 1, \ldots, L, \tag{81}$$

wobei aus Identifizierbarkeitsgründen an diese Effekte die Bedingung $\sum_{l=1}^{L} n_l \alpha_l = 0$ gestellt wird. Offenbar ist Modell (81) ohne eine solche Nebenbedingung an die α_l überparametrisiert. α_l wird der Effekt des l-ten Faktorlevels genannt.

Bemerkungen: Im Prinzip ist jede lineare Nebenbedingung der Art $\sum_{l=1}^{L} d_l \alpha_l = 0$ mit $\sum_{l=1}^{L} d_l \neq 0$ möglich (vgl. z. B. Seber (1977), §9.1.6, S. 247). Jedoch garantiert $d_l = n_l$, dass die orthogonale Zerlegung (79) auf Seite 287 stets gültig ist (vgl. z. B. Seber (1977), §9.1.9, S. 250f). **R** verwendet in der Tat die Nebenbedingung $\sum_{l=1}^{L} n_l \alpha_l = 0$. Im balancierten Fall ($n_l \equiv n$) vereinfacht sich der gesamte Formalismus erheblich, was wir hier aber nicht ausführen.

Die Funktion `aov()` fittet das einfaktorielle Modell in der Faktoreffekte-Form (81), wobei es bei ungeordneten Faktoren aus Identifizierbarkeitsgründen – wie bei der linearen Regression – intern zu einer Reparametrisierung per Voreinstellung mittels der *Treatment-Kontraste* kommt (was man zwar nicht merkt, aber z. B. durch die Anwendung von `coef()` auf ein `aov`-Objekt bestätigt erhält; siehe nächsten Abschnitt 11.1.4). Die Schätzer $\hat{\alpha}_l$ für die Effekte im Faktoreffekte-Modell oder die Schätzer $\hat{\mu}_l$ für die Faktorlevel-Mittelwerte im cell means-Modell können dann je nach Bedarf mit Hilfe der Funktion `model.tables()` ermittelt werden:

Einfaktorielle Varianzanalyse: Die Parameterschätzer	

`> model.tables( Koag.aov)`
`Tables of effects`
 	
`  Diaet`
`  A  B  C  D`
`  -3.08 1.92 3.92 -3.08`
`rep  4.00 7.00 6.00  8.00`
	Liefert (voreinstellungsgemäß) die Schätzer $\hat{\alpha}_l$ für die Effekte im Faktoreffekte-Modell (81). Man entnimmt außer den Schätzwerten für die Effekte (= Abweichungen vom Gesamtmittel) aller Levels auch die Anzahl (`rep`) der Beobachtungen auf jedem Level.
	Beachte: $\sum_{l=1}^{L} n_l \hat{\alpha}_l = 0$ ist erfüllt!

`> model.tables( Koag.aov,`
`+ type = "means")`
`Tables of means`
`Grand mean`
`  64.08`
`  Diaet`
`  A  B  C  D`
`  61 66 68 61`
`rep  4  7  6  8`
	Das Argument `type = "means"` erwirkt, dass (anstelle der Effekteschätzer) sowohl der Schätzer $\hat{\mu}_0$ für das Gesamtmittel μ_0 als auch die Schätzer $\hat{\mu}_l$ für die Faktorlevel-Mittelwerte im cell means-Modell (75) sowie die Anzahl (`rep`) der Beobachtungen auf jedem Level ausgegeben werden.

11.1.4 Modelldiagnose

Da es sich bei einem `aov`-Objekt faktisch um ein lineares Modell und damit auch um ein `lm`-Objekt handelt, stehen die für lineare Modelle üblichen Diagnosefunktionen auch hier zur Verfügung, z. B. `coef()`, um an die Koeffizienten des zugrundeliegenden linearen Modells zu

kommen. Auf diese Art können (und sollten) nach dem Fit des Modells somit auch die Modellannahmen mit Hilfe der Residuen und der gefitteten Werte untersucht werden. Dies geschieht wie bei der linearen Regression auch z. B. durch Histogramme und Q-Q-Plots der Residuen zur Beurteilung der Normalverteilungsannahme bzw. Plots der Residuen gegen die gefitteten Werte, um die Varianzhomogenität zu prüfen, entweder „von Hand" oder durch Verwendung der für `lm`-Objekte zur Verfügung stehenden Methode von `plot()`.

Ferner liefert die explizit aufrufbare `lm`-Methode von `summary()` die für ein lineares Regressionsmodell übliche Ausgabe, wenn ihr ein `aov`-Objekt übergeben wird (s. u.):

Einfaktorielle Varianzanalyse: Modelldiagnose(-plots)

```
> coef( Koag.aov)
 (Intercept)        DiaetB
6.100000e+01  5.000000e+00

      DiaetC        DiaetD
7.000000e+00 -1.006208e-14
```

Die Koeffizientenschätzer des dem `aov`-Objekt zugrundeliegenden linearen Modells in der Parametrisierung durch Treatment-Kontraste. (Dass der Schätzwert für den Koeffizienten von `DiaetD` nicht exakt Null ist, liegt an der Maschinengenauigkeit, ist also ein digitales Rundungsproblem.)

```
> fitted( Koag.aov)
....
> resid( Koag.aov)
....
```

An die gefitteten Werte $\hat{Y}_{li} \equiv \hat{\mu}_l$ (= geschätzte mittlere Responses) und die Residuen $\hat{\varepsilon}_{li} = Y_{li} - \hat{Y}_{li}$ kommt man wieder durch die Funktionen `fitted()` bzw. `resid()`.

```
> plot( Koag.aov,
+ which = 1:2)
```

Zwei ausgewählte Diagnoseplots für das als `lm`-Objekt „interpretierte" `aov`-Objekt, wie sie aus der linearen Regression bekannt sind (siehe Plots auf der Seite 292 oben).

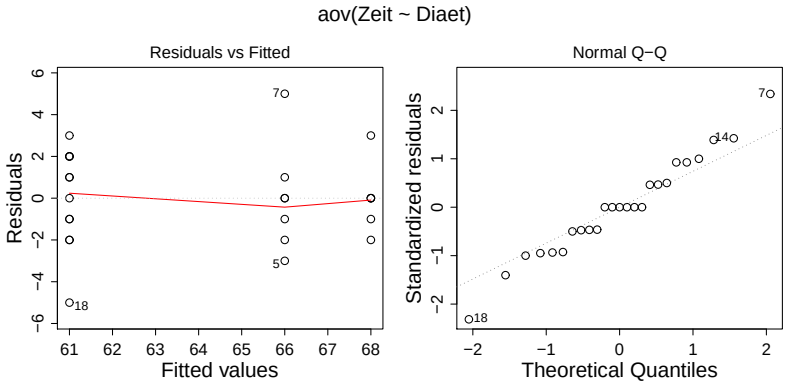
```
> plot( fitted( Koag.aov),
+ resid( Koag.aov))
> qqnorm( resid( Koag.aov))
```

Der Plot der Residuen gegen die gefitteten Werte, um die Varianzhomogenität zu prüfen, und Q-Q-Plot der Residuen zur Beurteilung der Normalverteilungsannahme der Fehler können auch „von Hand" angefertigt werden (nicht gezeigt). Natürlich wäre auch hier `qqPlot()` aus dem Package `car` auf das `aov`-Objekt anwendbar (und hilfreich).

```
> summary.lm( Koag.aov)
Call:
 aov(formula = Zeit ~ Diaet,
     data = Koag.df)
```

Das `aov`-Objekt wird durch die `lm`-Methode `summary.lm()` explizit als `lm`-Objekt ausgewertet und auch so dargestellt (weil eben `aov( Zeit ~ Diaet, ....)` inhaltlich dasselbe ist wie `lm( Zeit ~ Diaet, ....)`).

```
Residuals:
       Min         1Q     Median         3Q        Max
-5.000e+00 -1.000e+00  1.110e-16  1.000e+00  5.000e+00

Coefficients:
              Estimate Std. Error   t value Pr(>|t|)
(Intercept)  6.100e+01  1.155e+00    52.828  < 2e-16 ***
DiaetB       5.000e+00  1.447e+00     3.454 0.002375 **
DiaetC       7.000e+00  1.491e+00     4.696 0.000123 ***
DiaetD      -1.006e-14  1.414e+00 -7.11e-15 1.000000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.309 on 21 degrees of freedom
Multiple R-squared: 0.6743,     Adjusted R-squared: 0.6277
F-statistic: 14.49 on 3 and 21 DF,  p-value: 2.435e-05
```



11.1.5 Bartletts k-Stichprobentest auf Varianzhomogenität und Welchs k-Stichprobentest auf Erwartungswertegleichheit

Die Annahme der Gleichheit der Fehlervarianz – auch Homoskedastizität genannt – in den Faktorlevels (siehe Gleichung (75) lässt sich ebenfalls testen, und zwar mit dem Bartlett-Test auf Varianzhomogenität in $k \geq 2$ Stichproben aus normalverteilten Populationen. Seine Teststatistik hat unter der Hypothese der Varianzhomogenität für große Stichprobenumfänge eine *approximative* χ^2-Verteilung mit $k-1$ Freiheitsgraden. (Beachte, dass für $k = 2$ der F-Test auf Gleichheit zweier Varianzen ein exakter Test ist und dass Bartletts Test ziemlich empfindlich gegenüber Abweichungen von der Normalverteilungsannahme ist.) Wir gehen hier auf die Theorie des Bartlett-Tests nicht ein, sondern demonstrieren lediglich den Gebrauch seiner Implementierung durch die Funktion `bartlett.test()` und verweisen auf ihre Online-Hilfe bzw. die Literatur (z. B. [66, Neter, Wasserman, Kutner (1990)] oder [68, Sachs & Hedderich (2006), §7.5.1.3]):

```
> bartlett.test( Zeit ~ Diaet, data = Koag.df)

        Bartlett test of homogeneity of variances

data:  Zeit by Diaet
Bartlett's K-squared = 1.4092, df = 3, p-value = 0.7034
```

Hier liegt offenbar kein signifikantes Indiz gegen die Annahme der Varianzhomogenität vor.

Bemerkung: Ein gegenüber einer Abweichung von der Normalverteilungsannahme robusterer Test auf Homoskedastizität (= Varianzhomogenität = Varianzengleichheit) ist Levenes Test, der z. B. im **R**-Paket `car` in der Funktion `leveneTest()` implementiert ist (siehe z. B. [68, Sachs & Hedderich (2006), §7.5.1.4] oder die in der Online-Hilfe zitierte Literatur).

Ist die Annahme der Varianzhomogenität nicht zu halten, ist die „klassische“ ANOVA des vorherigen Abschnitts eigentlich unzulässig. In diesem Fall kann als *approximativer* Test Welchs Test auf Gleichheit der Erwartungswerte in $k \geq 2$ normalverteilten Populationen mit nicht notwendigerweise gleichen Varianzen eingesetzt werden. Er ist in der Funktion `oneway.test()` implementiert und wir verweisen auch hierzu auf die Online-Hilfe bzw. die dort erwähnte (Original-)Literatur. (Für den Spezialfall $k = 2$ siehe §8.5.3, Seite 130.):

```
> oneway.test( Zeit ~ Diaet, data = Koag.df)

        One-way analysis of means (not assuming equal variances)

data:  Zeit and Diaet
F = 16.9918, num df = 3.000, denom df = 10.431, p-value = 0.0002464
```

Das Resultat ist in Übereinstimmung mit der Schlussfolgerung für die im vorherigen Abschnitt durchgeführte, „klassische“ ANOVA mit der Annahme der Varianzhomogenität.

11.1.6 Testgüte und Fallzahlschätzung in der balancierten einfaktoriellen Varianzanalyse

Am Anfang von Abschnitt 8.12 wurde schon einmal auf das Konzept der Testgüte (oder "power") eingegangen, zunächst recht allgemein, aber dann in Ein- und Zweistichprobenproblemen für den Fall normalverteilter Zufallsvariablen im Detail, um ihre Anwendung zu demonstrieren, speziell für die Schätzung notwendiger Fallzahlen, um eine gewünschte Testgüte zu erzielen. Klar wurde, dass die Testgüte entscheidend vom Verhalten, sprich der Verteilung der Teststatistik unter der Alternative abhängt, und dies u. a. über den „Abstand" der Null- von der Alternativhypothese. Bei der einfaktoriellen Varianzanalyse ist dies ebenso, aber naheliegenderweise noch etwas komplizierter als bei (nur) zwei Stichproben.

Zur Klärung des Sachverhalts betrachten wir die Statistik $T(\mathbf{Y})$ für den Test einer linearen Hypothese $H_0 : \mathbf{C}\boldsymbol{\mu} = \mathbf{c}_0$ (mit einem $\mathbf{C}$ von vollem Zeilenrang $\text{Rang}(\mathbf{C}) = \dim(\mathbf{c}_0)$) im linearen Modell wie sie in Abschnitt 10.1 auf Seite 199 rekapituliert wurde nun noch einmal genauer, aber dabei in ihrer Notation bereits zugeschnitten auf die balancierte (!) einfaktorielle ANOVA (vgl. Gleichung (78) und beachte, dass im balancierten Fall mit n Untersuchungseinheiten pro Faktorlevel $N = n \cdot L$ ist):

$$T(\mathbf{Y}) = \frac{(\text{RSS}_H - \text{RSS})/\text{Rang}(\mathbf{C})}{\text{RSS}/(N - \dim(\boldsymbol{\mu}))} \tag{82}$$

$$= \frac{(\mathbf{C}\hat{\boldsymbol{\mu}} - \mathbf{c}_0)' \left[\mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}'\right]^{-1} (\mathbf{C}\hat{\boldsymbol{\mu}} - \mathbf{c}_0)/\text{Rang}(\mathbf{C})}{\text{RSS}/(nL - L)} \tag{83}$$

$$\sim F_{\text{Rang}(\mathbf{C}),L(n-1)} \quad \text{unter } H_0 \tag{84}$$

Die Testgüte $\mathbb{P}_{H_1}(H_0$ wird verworfen) ist demnach offenbar

$$\mathbb{P}_{H_1}\left(T(\mathbf{Y}) > F_{1-\alpha;\text{Rang}(\mathbf{C}),L(n-1)}\right), \tag{85}$$

was heißt, dass wir zu ihrer Berechnung die Verteilung von $T(\mathbf{Y})$ kennen müssen, wenn H_0 *nicht* erfüllt ist.

Exkurs: Mit einigem Aufwand (unter eleganter Verwendung des Matrixkalküls; siehe z. B. [46, Hocking (1996), §2 und §3] oder [70, Searle (1971)], §3.6) kann man in der Verteilungstheorie linearer und quadratischer Formen normalverteilter Vektoren nachweisen, dass $\text{RSS}_H - \text{RSS}$ im Zähler von $T(\mathbf{Y})$ eine mit σ^2 skalierte <u>nicht-zentrale χ^2-Verteilung</u> mit <u>Nicht-Zentralitätsparameter</u> λ besitzt, kurz:

$$\text{RSS}_H - \text{RSS} = (\mathbf{C}\hat{\boldsymbol{\mu}} - \mathbf{c}_0)' \left[\mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}'\right]^{-1} (\mathbf{C}\hat{\boldsymbol{\mu}} - \mathbf{c}_0) \sim \sigma^2 \chi^2_{\text{Rang}(\mathbf{C}),\lambda}, \tag{86}$$

wobei

$$\lambda = \frac{1}{\sigma^2}(\mathbf{C}\boldsymbol{\mu} - \mathbf{c}_0)' \left[\mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}'\right]^{-1} (\mathbf{C}\boldsymbol{\mu} - \mathbf{c}_0) \tag{87}$$

(natürlich unter der – im gegebenen Fall erfüllten – Bedingung, dass die Inversenbildung möglich ist).

Beachte:

- Offenbar ist $\lambda = 0$ unter H_0, sodass $\text{RSS}_H - \text{RSS}$ dann eine zentrale χ^2-Verteilung mit $\text{Rang}(\mathbf{C})$ Freiheitsgraden hat.
- Der Nicht-Zentralitätsparameter der χ^2-Verteilung wird in der Literatur nicht einheitlich definiert. In manchen Quellen wird er so wie in (87) verwendet, in anderen mit einem Vorfaktor 1/2, in wiederum anderen als Wurzel aus (87). In **R**s Implementation der nicht-zentralen χ^2-Verteilung kommt er in der Form von (87) zum Einsatz.

Hintergrundinformationen: Zu Referenzzwecken erinnern wir an ein Theorem, das das Zustandekommen der nicht-zentralen χ^2-Verteilung mit r Freiheitsgraden und Nicht-Zentralitätsparameter λ charakterisiert (siehe z. B. [46, Hocking (1996), §2.3.2] oder [70, Searle (1971)], §2.4.h): Es sei $\mathbf{Z} \sim \mathcal{N}_n(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ und $\mathbf{A}$ eine $(n \times n)$-Matrix. Dann ist $\mathbf{Z}'\mathbf{A}\mathbf{Z} \sim \chi^2_{r,\lambda}$ mit $r = \text{Rang}(\mathbf{A})$ und $\lambda = \boldsymbol{\mu}'\mathbf{A}\boldsymbol{\mu}$ genau dann, wenn $\mathbf{A}\boldsymbol{\Sigma}$ idempotent ist.

Und zur unmissverständlichen Parametrisierung der nicht-zentralen χ^2-Verteilung mit r Freiheitsgraden und Nicht-Zentralitätsparameter λ geben wir sicherheitshalber auch ihre Dichte explizit so an, wie sie in **R** implementiert ist. Sie lautet:

$$f_{r,\lambda}(x) = e^{-\lambda/2} \sum_{i=0}^{\infty} \frac{(\lambda/2)^i}{i!} f_{r+2i}(x) \quad \text{für } x > 0, \tag{88}$$

wobei die in dieser Reihendarstellung vorkommende Dichte f_k der zentralen χ^2-Verteilung mit k Freiheitsgraden

$$f_k(x) = \frac{x^{k/2-1} e^{-x/2}}{2^{k/2}\Gamma(k/2)} \quad \text{für } x > 0 \tag{89}$$

ist. Übrigens sind in obiger Parametrisierung $\mathbb{E}[\chi^2_{r,\lambda}] = r + \lambda$ und $\text{Var}(\chi^2_{r,\lambda}) = 2r + 4\lambda$. Für Details siehe z. B. [70, Searle (1971), §2.4.g - i].
(Ende der Hintergrundinformationen)

Bemerkung: In **R** ist in der Familie der Funktionen zur χ^2-Verteilung (`dchisq()`, `pchisq()`, `qchisq()` und `rchisq()`) mit deren Argument `ncp` der Nicht-Zentralitätsparameter λ in der obigen Form gemeint.

Mit ebenfalls einigem Aufwand (sowie eleganter Verwendung des Matrixkalküls; vgl. [46, Hocking (1996), a. a. O.] oder [70, Searle (1971), a. a. O.]) ist ferner Folgendes belegbar: RSS/σ^2 ist stets mit $L(n-1)$ Freiheitsgraden zentral χ^2-verteilt und vom Zähler $\text{RSS}_H - \text{RSS}$ unabhängig, sodass $T(\mathbf{Y})$ eine nicht-zentrale F-Verteilung hat mit Zählerfreiheitsgrad $\text{Rang}(\mathbf{C})$ und Nennerfreiheitsgrad $L(n-1)$ sowie mit dem Nicht-Zentralitätsparameter λ aus (87), denn für zwei unabhängige Verteilungen $\chi^2_{r,\lambda}$ und χ^2_s ist $(\chi^2_{r,\lambda}/r)/(\chi^2_s/s) \sim F_{r,s,\lambda}$.

Um also die Testgüte in (85) für das einfaktorielle ANOVA-Design (in dem $\mathbf{X} = \mathbf{I}_L \otimes \mathbf{1}_n$ ist) konkret berechnen zu können oder auch ihre Abhängigkeit vom Stichprobenumfang n (pro Faktorlevel) und den Erwartungswerten $\mu_1, \ldots, \mu_L$ zu bestimmen, muss die rechte Seite in (87) „ausge-x-t“ werden: Aus $\mathbf{X} = \mathbf{I}_L \otimes \mathbf{1}_n$ folgt $(\mathbf{X}'\mathbf{X})^{-1} = \frac{1}{n}\mathbf{I}_L$ und $(\mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}')^{-1} = n(\mathbf{C}\mathbf{C}')^{-1}$. Dies liefert $\lambda = \frac{n}{\sigma^2}(\mathbf{C}\boldsymbol{\mu} - \mathbf{c}_0)'(\mathbf{C}\mathbf{C}')^{-1}(\mathbf{C}\boldsymbol{\mu} - \mathbf{c}_0)$, was sich für ein allgemeines $\mathbf{C}$ unter Einführung von $\mathbf{d}_0 := \mathbf{C}'(\mathbf{C}\mathbf{C}')^{-1}\mathbf{c}_0$ zu $\lambda = \frac{n}{\sigma^2}(\boldsymbol{\mu} - \mathbf{d}_0)'\mathbf{C}'(\mathbf{C}\mathbf{C}')^{-1}\mathbf{C}(\boldsymbol{\mu} - \mathbf{d}_0)$ umformen lässt und sich im spezielleren Fall $\mathbf{C} = (\mathbf{I}_{L-1} | -\mathbf{1}_{L-1})$ (was z. B. für die Hypothese der Gleichheit der μ_l benötigt würde) – auf *nicht* offensichtliche Weise – weiter konkretisiert zu $\lambda \stackrel{!}{=} \frac{n}{\sigma^2}(\boldsymbol{\mu} - \mathbf{d}_0)'(\mathbf{I}_L - \frac{1}{L}\mathbf{1}_L\mathbf{1}_L')(\boldsymbol{\mu} - \mathbf{d}_0) \stackrel{!}{=} \frac{n}{\sigma^2}\sum_{l=1}^{L}(\mu_l - \bar{\mu}_\cdot - d_{0,l})^2$ (denn $(\mathbf{C}\mathbf{C}')^{-1} = \mathbf{I}_{L-1} - \frac{1}{L}\mathbf{1}_{L-1}\mathbf{1}_{L-1}'$ (vgl. [46, Hocking (1996), Appendix A.I.10] oder auch [40, Graybill (1976), Thm. 8.3.4]) und weil $d_{0,\cdot} = 0$).
(Ende des Exkurses)

Den Exkurs zusammenfassend erhält man also im Fall $\mathbf{C} = (\mathbf{I}_{L-1} | -\mathbf{1}_{L-1})$ und $\mathbf{c}_0 = \mathbf{0}_{L-1}$:

$$T(\mathbf{Y}) \sim F_{\text{Rang}(\mathbf{C}),L(n-1),\lambda} \quad \text{mit} \tag{90}$$

$$\lambda = n(L-1)\frac{\frac{1}{L-1}\sum_{l=1}^{L}(\mu_l - \bar{\mu}_\cdot)^2}{\sigma^2}, \tag{91}$$

wobei unter $H_0 : \mathbf{C}\boldsymbol{\mu} = \mathbf{0}$, d. h. hier unter $H_0 : \mu_1 = \cdots = \mu_L$ offensichtlich $\lambda = 0$ ist (und dann eine zentrale F-Verteilung für $T(\mathbf{Y})$ resultiert).

Die Testgüte in (85) konkretisiert sich so zu

$$\text{Güte} = 1 - F_{\text{Rang}(\mathbf{C}),L(n-1),\lambda}\left(F_{1-\alpha;\text{Rang}(\mathbf{C}),L(n-1)}\right), \tag{92}$$

wodurch klar wird, dass sie abhängt vom Niveau α, vom levelspezifischen Stichprobenumfang n, der Anzahl der Levels L sowie – über λ! – vom Verhältnis der stets unbekannten „empirischen Varianz" (sinngemäß!) der L Erwartungswerte, auch "between treatment variability" genannt, und der i. d. R. ebenfalls unbekannten Varianz σ^2 der Residuen, auch "within treatment variability" genannt. (Außerdem steckt n auch noch einmal in λ.)

Konsequenzen:

- Bei einer konkreten Vorstellung über die μ_l ist die "between treatment variability" sofort berechenbar; benötigt wird dann aber auch noch die "within treatment variability" σ^2.
- Es reicht aber auch, „nur" das Verhältnis von "between" und "within treatment variability" zu spezifizieren.
- Offenbar können völlig verschiedene Szenarien für die L Erwartungswerte zum selben λ und somit zur selben Güte führen, insbesondere z. B. der Fall „alle μ_l paarweise 'ein bisschen' verschieden" und der Fall (o. B. d. A.) „$\mu_1 = \ldots = \mu_{L-1}$ und μ_L extrem abweichend".

Das Monotonieverhalten der Gütefunktion in (92) ist keineswegs offensichtlich. Man kann aber zeigen, dass sie bei Fixierung der jeweils übrigen Argumente ...

$$\begin{array}{llll} \text{in } \lambda & \text{monoton} & \text{wächst,} & \text{(Achtung, hier ist } \lambda = \lambda(n)\text{!)} \\ \text{"} \ L & \text{"} & \text{fällt,} & \\ \text{"} \ n & \text{"} & \text{wächst und} & \\ \text{"} \ \alpha & \text{"} & \text{wächst,} & \end{array} \tag{93}$$

womit auch klar ist, wie sich die "between" und die "within treatment variability" bei Fixierung der jeweils übrigen Argumente auswirken: die Güte ...

$$\begin{array}{lll} \text{wächst} & \text{monoton} & \text{in der "between treatment variability" und} \\ \text{fällt} & \text{"} & \text{in } \sigma. \end{array} \tag{94}$$

(Siehe auch [46, Hocking, 1996], §2.4.2, p. 59 oder direkt [38, Ghosh (1973)]. Vgl. außerdem auch die Aussagen des Abschnittes 8.12 zur Monotonie der Gütefunktion in Ein- oder Zweistichproben-Lokationsproblemen im Normalverteilungsmodell.)

Die **R**-Funktion `power.anova.test()` erlaubt die Berechnung der Güte oder die Bestimmung von Werten anderer Parameter, um eine angestrebte Güte zu erhalten. Ihre sechs Argumente und deren Voreinstellungswerte nebst den von uns für sie benutzten Bezeichnungen lauten

L	n	"between treatment variability"	σ^2
`groups = NULL`	`n = NULL`	`between.var = NULL`	`within.var = NULL`

α	$1-\beta$
`sig.level = 0.05`	`power = NULL`

(Vgl. auch den §8.12.2.2 zur Verwendung der Gütefunktion in Ein- oder Zweistichproben-Lokationsproblemen im Normalverteilungsmodell durch `power.t.test()`.)

Der/die Benutzer/in muss dafür sorgen, dass beim Aufruf von `power.anova.test()` genau fünf der sechs Argumente `groups`, `n`, `between.var`, `within.var`, `sig.level` und `power` mit einem von `NULL` verschiedenen Wert versehen werden, damit der Wert des „fehlenden" Arguments dann aus denen der anderen abgeleitet wird. Beachte, dass `sig.level` als Voreinstellung *nicht* den Wert `NULL` hat, sodass, wenn dieses berechnet werden soll, `NULL` hierfür explizit anzugeben ist.

11.2 Die zweifaktorielle Varianzanalyse ("Two-way-ANOVA")

Hier haben wir es mit einer metrischen Response zu tun, die für die Levelkombinationen *zweier* Faktoren A und B an unabhängigen Untersuchungseinheiten pro Untersuchungseinheit jeweils genau einmal gemessen wird. Wird jede Levelkombination mindestens einmal beobachtet und werden die Untersuchungseinheiten den Faktorlevelkombinationen zufällig (*randomisiert*) zugewiesen, handelt es sich um die zweifaktorielle Varianzanalyse (oder das Zwei-Faktoren-Modell) für einen vollständigen, randomisierten Versuchsplan.

11.2.1 "cell means"-Modell und Inferenz

Faktor A habe die Levels $j = 1, \ldots, J$ und Faktor B die Levels $l = 1, \ldots, L$ und die Response werde für jede Levelkombination (j, l) an $n_{jl} \geq 1$ unabhängigen Untersuchungseinheiten beobachtet. Dann lässt sich dies als cell means-Modell wie folgt schreiben:

$$\begin{aligned} Y_{jli} = \mu_{jl} + \varepsilon_{jli} \quad &\text{mit} \quad \varepsilon_{jli} \text{ i.i.d.} \sim \mathcal{N}(0, \sigma^2) \\ \text{für } i = 1, \ldots, n_{jl} \quad &\text{und} \quad j = 1, \ldots, J,\ l = 1, \ldots, L, \end{aligned} \tag{95}$$

wobei Y_{jli} die Response der Untersuchungseinheit i für die Faktorlevelkombination (j, l) und μ_{jl} die mittlere Response für diese Faktorlevelkombination ist sowie die „Fehler" ε_{jli} die individuellen Abweichungen von μ_{jl} sind. Insgesamt liegen $N := \sum_{j=1}^{J} \sum_{l=1}^{L} n_{jl}$ Beobachtungen vor. Die n_{jl} werden Zellhäufigkeiten genannt und es wird zwischen balancierten (falls $n_{jl} \equiv n$) und unbalancierten Versuchsplänen unterschieden.

Werden die Y_{jli} und ε_{jli} zueinander passend in N-dimensionale Vektoren $\mathbf{Y} = (Y_{111}, \ldots, Y_{JLn_{JL}})'$ bzw. $\boldsymbol{\varepsilon} = (\varepsilon_{111}, \ldots, \varepsilon_{JLn_{JL}})'$ „gestapelt" – wobei die tatsächliche Reihenfolge der Elemente keine Rolle spielt, falls sie nur konsistent ist – und die μ_{jl} geeignet in einem JL-dimensionalen Vektor $\boldsymbol{\mu} = (\mu_{11}, \ldots, \mu_{JL})'$ zusammengefasst, existiert eine $(N \times JL)$-Matrix $\mathbf{X}$ mit 0-1-Einträgen, sodass obiger Sachverhalt in Matrixnotation als $\mathbf{Y} = \mathbf{X}\boldsymbol{\mu} + \boldsymbol{\varepsilon}$ formuliert werden kann (analog zur Darstellung in §11.1.1). Details sparen wir uns hier jedoch.

Bemerkung zur Notation: Zur Abkürzung von im Folgenden häufig auftretenden Summen und arithmetischen Mitteln werden wir von üblicher Notation Gebrauch machen: Für ein mehrfach indiziertes Schema, wie z. B. x_{jli} mit $j = 1, \ldots, J$, $l = 1, \ldots, L$ und $i = 1, \ldots, n_{jl}$, bedeutet ein Punkt „·" anstelle eines Indexes, dass die marginale Summe der x_{jli} über diesen Index gebildet wurde. Beispiele:

$$x_{\cdot li} = \sum_{j=1}^{J} x_{jli} \qquad \text{oder} \qquad x_{\cdot l \cdot} = \sum_{j=1}^{J} \sum_{i=1}^{n_{jl}} x_{jli} \qquad \text{oder} \qquad x_{\cdots} = \sum_{j=1}^{J} \sum_{l=1}^{L} \sum_{i=1}^{n_{jl}} x_{jli}$$

Analog werden (marginale) arithmetische Mittel bezeichnet:

$$\bar{x}_{\cdot li} = \frac{1}{J} \sum_{j=1}^{J} x_{jli} \equiv \frac{x_{\cdot li}}{J} \qquad \text{oder} \qquad \bar{x}_{\cdot l \cdot} = \frac{1}{n_{\cdot l}} \sum_{j=1}^{J} \sum_{i=1}^{n_{jl}} x_{jli} \equiv \frac{x_{\cdot l \cdot}}{n_{\cdot l}}$$

$$\text{oder} \qquad \bar{x}_{\cdots} = \frac{1}{n_{\cdot\cdot}} \sum_{j=1}^{J} \sum_{l=1}^{L} \sum_{i=1}^{n_{jl}} x_{jli} \equiv \frac{x_{\cdots}}{n_{\cdot\cdot}}$$

Parameterschätzung: Die Kleinste-Quadrate-Schätzer (KQS) $\hat{\mu}_{jl}$ für μ_{jl} (für $j = 1, \ldots, J$ und $l = 1, \ldots, L$) in obigem Modell (95) sind die Lösung von

$$\boldsymbol{\varepsilon}'\boldsymbol{\varepsilon} \equiv \sum_{j=1}^{J} \sum_{l=1}^{L} \sum_{i=1}^{n_{jl}} (Y_{jli} - \mu_{jl})^2 \stackrel{!}{=} \text{minimal in allen } \mu_{jl}$$

Dies liefert $\hat{\mu}_{jl} = \bar{Y}_{jl\cdot}$ und $\hat{\varepsilon}_{jli} := Y_{jli} - \bar{Y}_{jl\cdot}$ als Residuen. Die Residuenquadratesumme (RSS) ergibt sich also zu

$$\mathrm{RSS} = \sum_{j=1}^{J} \sum_{l=1}^{L} \sum_{i=1}^{n_{jl}} \left(Y_{jli} - \bar{Y}_{jl\cdot}\right)^2 \tag{96}$$

Hierfür gilt dieselbe Bemerkung wie die unter Gleichung (76) auf Seite 286 zur Berechnung von $\hat{\boldsymbol{\mu}}$ und der RSS.

11.2.1.1 „Interessante" Hypothesen

Die **Hypothese der Gleichheit aller Erwartungswerte** (cell means) lautet

$$H_0 : \mu_{11} = \ldots = \mu_{JL} \tag{97}$$

und bedeutet, dass *beide* Faktoren keinen Einfluss haben. Sie ist äquivalent zu der Form $H_0^{(1)}$: $\mu_{jl} - \mu_{JL} = 0 \;\forall\; (j,l) \neq (J,L)$ mit $j = 1,\ldots,J, l = 1,\ldots,L$, die sich mit der $((JL-1) \times JL)$-Matrix $\mathbf{C} = [\mathbf{I}_{JL-1} | -\mathbf{1}_{JL-1}]$ vom Rang $JL-1$ als eine lineare Bedingung $\mathbf{C}\boldsymbol{\mu} = \mathbf{0}$ an den Parameter*vektor* $\boldsymbol{\mu}$ formulieren lässt. Unter H_0 vereinfacht sich das Modell zu $Y_{jli} = \mu_0 + \varepsilon_{jli}$ und der KQS $\hat{\mu}_0$ für μ_0 wird als Lösung von

$$\sum_{j=1}^{J} \sum_{l=1}^{L} \sum_{i=1}^{n_{jl}} (Y_{jli} - \mu_0)^2 \stackrel{!}{=} \text{minimal in } \mu_0$$

zu $\hat{\mu}_0 = \bar{Y}_{\cdots}$ ermittelt. Die Residuenquadratesumme unter H_0 ist demnach

$$\mathrm{RSS}_H = \sum_{j=1}^{J} \sum_{l=1}^{L} \sum_{i=1}^{n_{jl}} \left(Y_{jli} - \bar{Y}_{\cdots}\right)^2$$

(Siehe auch die Bemerkung unter Gleichung (76) auf Seite 286.) Völlig analog zum einfaktoriellen Modell erhalten wir (durch Teleskopieren gemäß $Y_{jli} - \bar{Y}_{\cdots} = Y_{jli} - \bar{Y}_{jl\cdot} + \bar{Y}_{jl\cdot} - \bar{Y}_{\cdots}$ und anschließendes Quadrieren sowie Summieren) die orthogonale Zerlegung

$$\mathrm{RSS}_H = \mathrm{RSS} + \sum_{j=1}^{J} \sum_{l=1}^{L} n_{jl} \left(\bar{Y}_{jl\cdot} - \bar{Y}_{\cdots}\right)^2 \tag{98}$$

Damit ist H_0 unter Verwendung von (96) und (98) mit Hilfe der F-Teststatistik in (78) testbar, wobei $\dim(\boldsymbol{\mu}) = JL$ und $\mathrm{Rang}(\mathbf{C}) = JL - 1$ ist. Ferner gilt die Bemerkung unter (78) bzgl. der Anforderung an die Zellhäufigkeiten auch hier.

Wird H_0 nicht verworfen, sind weitere inferenzstatistische Vergleiche der cell means faktisch überflüssig. Anderenfalls ist es in der Regel interessant, genauer zu klären, wie die Faktoren A und B wirken.

Für die Formulierung weiterer, typischer Hypothesen, die die Einflüsse der verschiedenen Levels der beiden Faktoren A und B separat bzw. die (noch näher zu spezifizierende) Interaktion zwischen A und B betreffen, ist als „Anschauungsmaterial" die auf der nächsten Seite folgende, aus [46, Hocking (1996), S. 437] stammende tabellarische Darstellung der cell means für den Fall eines Faktors A mit $J = 3$ Levels und eines Faktors B mit $L = 4$ Levels hilfreich. Die Tabelle zeigt neben den cell means μ_{jl} jeder Faktorlevelkombination (j,l) auch deren Zeilenmittelwerte $\bar{\mu}_{j\cdot}$ bzw. Spaltenmittelwerte $\bar{\mu}_{\cdot l}$ in ihren Rändern. Diese werden daher die marginalen Erwartungswerte von A bzw. B genannt.

		Faktor B				
		1	2	3	4	
	1	μ_{11}	μ_{12}	μ_{13}	μ_{14}	$\bar{\mu}_{1\cdot}$
Faktor A	2	μ_{21}	μ_{22}	μ_{23}	μ_{24}	$\bar{\mu}_{2\cdot}$
	3	μ_{31}	μ_{32}	μ_{33}	μ_{34}	$\bar{\mu}_{3\cdot}$
		$\bar{\mu}_{\cdot 1}$	$\bar{\mu}_{\cdot 2}$	$\bar{\mu}_{\cdot 3}$	$\bar{\mu}_{\cdot 4}$	$\bar{\mu}_{\cdot\cdot}$

Haupteffekt-Hypothesen:

Eine Frage kann sein, ob A allein betrachtet, also ohne Berücksichtigung der Existenz verschiedener Levels von B, einen Einfluss auf die cell means hat. Man nennt diesen potenziellen Einfluss Haupteffekt von A oder – angesichts der tabellarischen Darstellung – auch Zeileneffekt und meint damit (potenzielle) Unterschiede zwischen den marginalen Erwartungswerten $\bar{\mu}_{1\cdot}, \ldots, \bar{\mu}_{J\cdot}$ von A, weswegen Haupteffekt-Hypothesen für A stets Hypothesen über die marginalen Erwartungswerte von A sind. D. h., der Einfluss der Levels von A wird gemittelt über alle Levels von B analysiert. Die (wesentlichste) **Haupteffekt-Hypothese für A** lautet

$$H_A : \bar{\mu}_{1\cdot} = \ldots = \bar{\mu}_{J\cdot} \tag{99}$$

Zur Interpretation: Es kann geschehen,

- dass H_A nicht verworfen wird, obwohl Unterschiede zwischen den A-Levels existieren, wenn diese Unterschiede jedoch von den B-Levels abhängen und sich „herausmitteln“, oder
- dass umgekehrt H_A allein aufgrund eines (hinreichend großen) Einflusses von A innerhalb *eines einzelnen* B-Levels verworfen wird, obwohl A in den anderen B-Levels keinen Einfluss hat.

Einerseits sollte das Ergebnis des Tests von H_A daher gewissermaßen als vorläufig angesehen werden, denn die detaillierte Klärung der Ursache für die Testentscheidung bedarf weitergehender Untersuchungen (und führt in natürlicher Weise zum Thema der multiplen Erwartungswertvergleiche).

Andererseits kann H_A auch schon für sich genommen von abschließendem Interesse sein: Falls eigentlich ein über alle B-Levels hinweg gleichmäßig „bestes“ A-Level gesucht ist, dieses aber nicht existiert, so ist (eventuell) dasjenige A-Level akzeptabel, welches immerhin *im Durchschnitt über alle B-Levels* am besten abschneidet. Dann ist ein Vergleich der marginalen Erwartungswerte von A adäquat und schon ausreichend.

Für den Faktor B gelten die obigen Ausführungen analog. Die (wesentlichste) **Haupteffekt-Hypothese für B** lautet

$$H_B : \bar{\mu}_{\cdot 1} = \ldots = \bar{\mu}_{\cdot L} \tag{100}$$

Bemerkungen: Die Hypothesen H_A und H_B sind jeweils äquivalent zu

$$H_A^{(2)} : \bar{\mu}_{j\cdot} = \bar{\mu}_{\cdot\cdot} \;\forall\; 1 \leq j \leq J-1 \qquad \text{bzw.} \qquad H_B^{(2)} : \bar{\mu}_{\cdot l} = \bar{\mu}_{\cdot\cdot} \;\forall\; 1 \leq l \leq L-1 \tag{101}$$

Die Bestimmung der zu H_A und H_B gehörenden F-Teststatistiken, deren allgemeine Form in (78) steht, kann daher z. B. auch mit Hilfe der Kontrastmatrizen für $H_A^{(2)}$ und $H_B^{(2)}$ geschehen. Wir sparen uns hier aber weitere Details.

Die Differenz $\bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot\cdot}$ heißt Haupteffekt von A für Level j und $\bar{\mu}_{\cdot l} - \bar{\mu}_{\cdot\cdot}$ heißt Haupteffekt von B für Level l.

Die Hypothese keiner Interaktion:

Wie sich zeigen wird, gestaltet sich die Interpretation des Testergebnisses für H_A abhängig davon, ob die Unterschiede zwischen den A-Levels von den B-Levels abhängen oder nicht. Daher betrachten wir diesen Sachverhalt nun etwas näher: Falls für zwei A-Levels $j_1 \neq j_2$ und zwei B-Levels $l_1 \neq l_2$ gilt

$$\mu_{j_1,l_1} - \mu_{j_2,l_1} = \mu_{j_1,l_2} - \mu_{j_2,l_2}, \tag{102}$$

so ist offensichtlich der Unterschied zwischen den beiden A-Levels j_1 und j_2 im B-Level l_1 derselbe wie im B-Level l_2. Zu (102) ist

$$\mu_{j_1,l_1} - \mu_{j_1,l_2} = \mu_{j_2,l_1} - \mu_{j_2,l_2} \tag{103}$$

äquivalent, sodass also auch der Unterschied zwischen den B-Levels l_1 und l_2 unverändert bleibt, wenn von A-Level j_1 zu j_2 gewechselt wird. Dies ist die Definition dafür, dass die A-Levels j_1 und j_2 mit den B-Levels l_1 und l_2 nicht interagieren.

Ist die Beziehung (102) für *alle* A-Level-Paare und B-Level-Paare erfüllt, so sagt man, dass zwischen A und B keine Interaktion besteht. Die entsprechende **Hypothese keiner Interaktion** – kurz, aber etwas widersinnig auch **Interaktionshypothese** genannt – lautet also

$$H_{AB} : \mu_{j_1,l_1} - \mu_{j_1,l_2} - (\mu_{j_2,l_1} - \mu_{j_2,l_2}) = 0 \quad \forall\, 1 \leq j_1 \neq j_2 \leq J,\ 1 \leq l_1 \neq l_2 \leq L \tag{104}$$

Bemerkungen: Die Hypothese H_{AB} ist äquivalent zu

$$H_{AB}^{(2)} : \mu_{jl} - \bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot l} + \bar{\mu}_{\cdot\cdot} = 0 \quad \forall\, 1 \leq j \leq J-1,\ 1 \leq l \leq L-1 \tag{105}$$

Offenbar ist $\mu_{jl} - \bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot l} + \bar{\mu}_{\cdot\cdot} = \mu_{jl} - (\bar{\mu}_{\cdot\cdot} + \bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot\cdot} + \bar{\mu}_{\cdot l} - \bar{\mu}_{\cdot\cdot})$ und stellt somit die Differenz zwischen μ_{jl} und dem Wert dar, der erwartet würde, wenn die Faktoren A und B rein additiv über ihre Haupteffekte $\bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot\cdot}$ und $\bar{\mu}_{\cdot l} - \bar{\mu}_{\cdot\cdot}$ wirkten. Diese Differenz heißt Interaktionseffekt des A-Levels j mit dem B-Level l. Trifft die Interaktionshypothese *nicht* zu, sagt man, dass Interaktionseffekte vorliegen.

11.2.1.2 Interaktion oder nicht? Grafische Darstellung

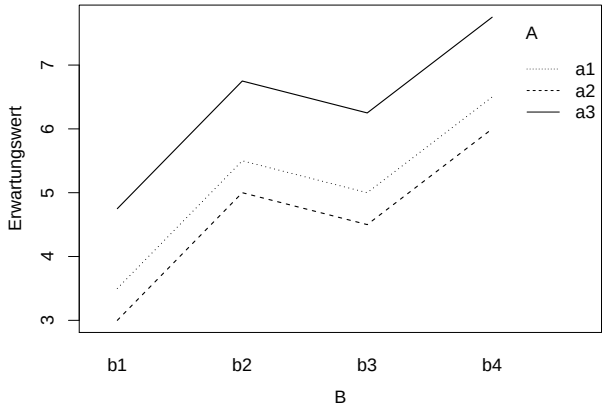
Für die Darstellung von Interaktionseffekten steht als grafisches Werkzeug der sogenannte Interaktionsplot zur Verfügung. Zu seiner Motivation ist zu beachten, dass unter der Hypothese H_{AB} keiner Interaktion für feste $l_1 \neq l_2$ und beliebiges j (bzw. für feste $j_1 \neq j_2$ und beliebiges l) aus (104) folgt:

$$\mu_{jl_1} - \mu_{jl_2} = \bar{\mu}_{\cdot l_1} - \bar{\mu}_{\cdot l_2} \qquad \text{(bzw.} \qquad \mu_{j_1 l} - \mu_{j_2 l} = \bar{\mu}_{j_1\cdot} - \bar{\mu}_{j_2\cdot}\text{)}$$

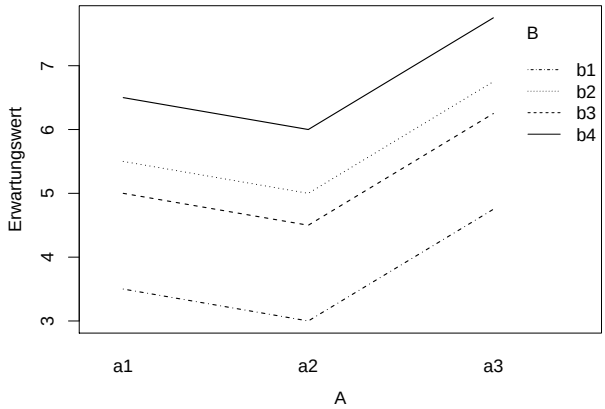
Expressis verbis: Der Wirkungs*unterschied* zwischen den B-Faktorlevels l_1 und l_2 (also der Abstand von μ_{jl_1} zu μ_{jl_2}) ist entlang der „j-Achse" (also für verschiedene A-Levels) konstant, nämlich gleich $\bar{\mu}_{\cdot l_1} - \bar{\mu}_{\cdot l_2}$. Die (Wirkungs-)Profile $(\mu_{1l_1}, \ldots, \mu_{Jl_1})$ und $(\mu_{1l_2}, \ldots, \mu_{Jl_2})$ des Faktors A für zwei verschiedene B-Levels l_1 und l_2 sind also „parallel". Da l_1 und l_2 zwar fest, aber beliebig waren, sind folglich *alle* L Profile von Faktor A „parallel". Eben dies gilt analog für die $j = 1, \ldots, J$ Profile $(\mu_{j1}, \ldots, \mu_{jL})$ des Faktors B.

Der beschriebene Sachverhalt kann in einer Grafik veranschaulicht werden, in der für jedes Level $l = 1, \ldots, L$ des Faktors B die Elemente μ_{jl} des Profils $(\mu_{1l}, \ldots, \mu_{Jl})$ des Faktors A gegen $j = 1, \ldots, J$ gezeichnet werden. Indem jedes der L A-Profile durch einen eigenen Polygonzug repräsentiert wird, ist dann unter H_{AB} Parallelität dieser Polygonzüge zu beobachten. Völlig analog gilt dies für eine entsprechende Grafik der J B-Profile. Beides wird in den drei folgenden Grafiken beispielhaft dargestellt.

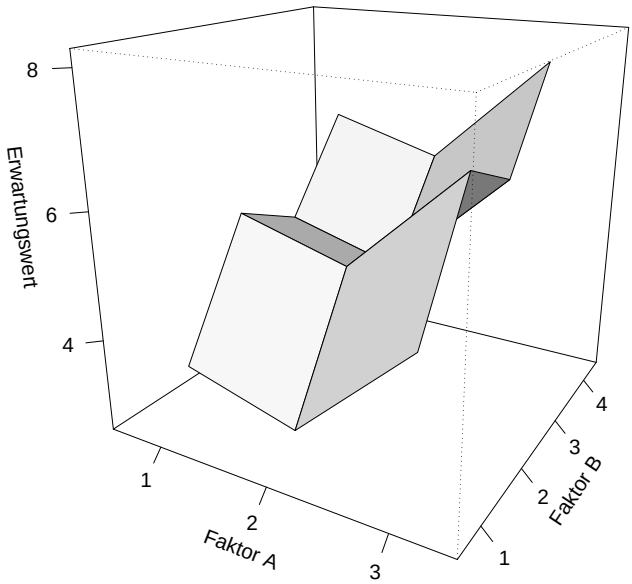
Interaktionsplots mit parallelen Profilen im Fall *keiner* Interaktion zwischen A und B:
Die B-Profile $(\mu_{j1}, \ldots, \mu_{jL})$ für $j = 1, \ldots, J = 3$:



Die A-Profile $(\mu_{1l}, \ldots, \mu_{Jl})$ für $l = 1, \ldots, L = 4$:

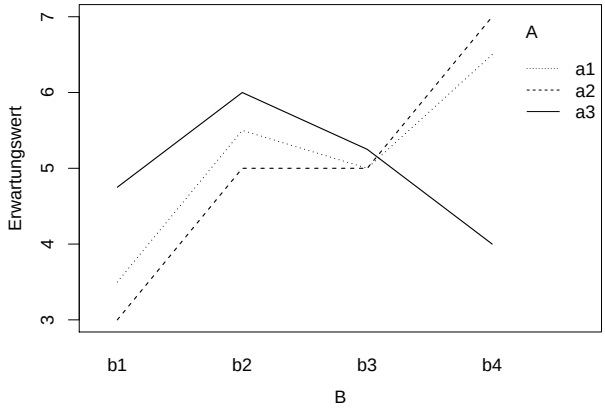


Eine die Vorstellung eventuell noch etwas weiter unterstützende Veranschaulichung des Konzeptes von Interaktion ist die perspektivische Darstellung der dreidimensionalen „Erwartungswerte-Oberfläche“ μ_{jl} über (j, l), wie sie in der folgenden Grafik für das Szenario zu sehen ist, welches den obigen beiden Interaktionsplots für zwei Faktoren *ohne* Interaktion zugrundeliegt:

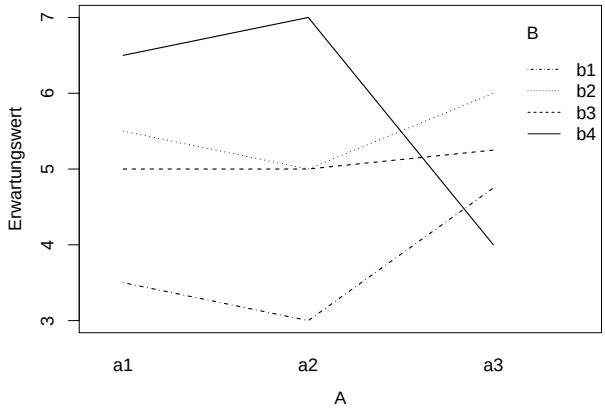


Es ist einigermaßen gut zu erkennen, wie das A-Profil beim Wechsel von einem B-Level zu einem anderen quasi als Ganzes parallel nach oben oder unten (bzgl. der vertikalen Achse) „wandert“. Analog verhält es sich mit den B-Profilen beim Wechseln entlang der A-Level-Achse.

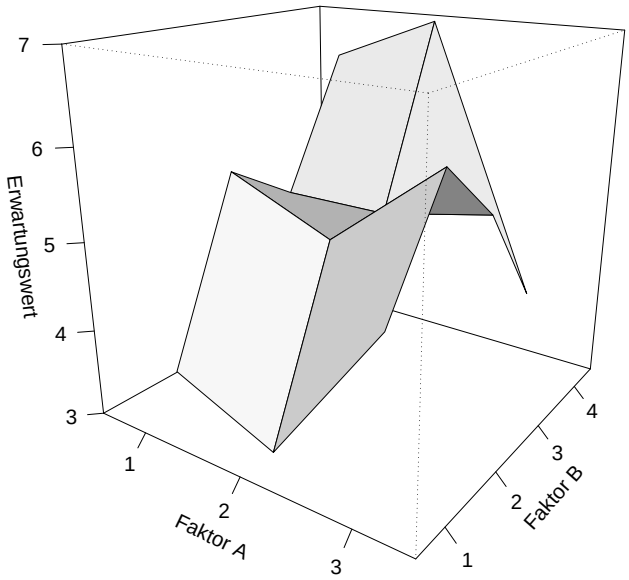
Im Gegensatz zu obigem zeigen wir nun beispielhaft zwei Interaktionsplots mit *nicht* parallelen Profilen in einem Szenario *mit* Interaktionseffekten zwischen A und B:
Die B-Profile $(\mu_{j1}, \ldots, \mu_{jL})$ für $j = 1, \ldots, J = 3$:



Die A-Profile $(\mu_{1l}, \ldots, \mu_{Jl})$ für $l = 1, \ldots, L = 4$:



Dreidimensionale „Erwartungswerte-Oberfläche" μ_{jl} über (j, l) für zwei Faktoren **mit** Interaktion:



Die „Oberfläche" zeigt einen erheblich irregulläreren Verlauf als im Beispiel ohne Interaktion.

11.2.1.3 Zur Interpretation der Testergebnisse

Die Konsequenz aus dem Erfülltsein von H_{AB} ist, dass in diesem Fall die Haupteffekt-Hypothesen H_A und H_B jeweils sogar äquivalent zu den „stärkeren“ Hypothesen

$$H_A^{(0)} : \mu_{1l} = \ldots = \mu_{Jl} \ \forall\ 1 \leq l \leq L \quad \text{bzw.} \quad H_B^{(0)} : \mu_{j1} = \ldots = \mu_{jL} \ \forall\ 1 \leq j \leq J \tag{106}$$

sind, wie sich mit Hilfe von $H_{AB}^{(2)}$ in der Form $\mu_{jl} - \bar{\mu}_{\cdot l} = \bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot\cdot}\ \forall\ 1 \leq j \leq J-1,\ 1 \leq l \leq L-1$ nachweisen lässt.

Für die korrekte Interpretation der Ergebnisse der Tests von H_A oder H_B muss also bekannt sein, ob zwischen Faktor A und B eine Interaktion vorliegt. Denn:

- **Ohne** eine Interaktion zwischen A und B bedeutet H_A also, dass innerhalb eines jeden B-Levels kein Unterschied zwischen den A-Levels existiert, und analog für H_B, dass innerhalb eines jeden A-Levels kein Unterschied zwischen den B-Levels ist.
- **Mit** einer A-B-Interaktion bedeutet H_A dann lediglich, dass der Faktor A *im Durchschnitt über alle B-Levels* – also im marginalen Sinne – keinen Einfluss hat. (Entsprechendes gilt für H_B.)

Kommen wir daher noch einmal zurück zu den vier „interessanten“ Hypothesen:

- H_0 bedeutet: Kein Einfluss der Faktoren A und B.
- H_{AB} bedeutet: Keine A-B-Interaktion, d. h., der Einfluss des einen Faktors hängt nicht vom anderen Faktor ab.
- H_A bedeutet: Falls H_{AB} erfüllt ist, ist Faktor A ohne jeglichen Einfluss; falls H_{AB} nicht erfüllt ist, hat Faktor A nur im Durchschnitt über alle B-Levels keinen Einfluss (ist also ohne marginalen Einfluss).
- H_B bedeutet: Falls H_{AB} erfüllt ist, ist Faktor B ohne jeglichen Einfluss; falls H_{AB} nicht erfüllt ist, hat Faktor B nur im Durchschnitt über alle A-Levels keinen Einfluss (ist also ohne marginalen Einfluss).

Obschon sich jede der vier Hypothesen bei angemessener Interpretation separat betrachten lässt, ist es also naheliegend, erst H_0, dann H_{AB} und schließlich H_A oder/und H_B zu testen: Kann H_0 nicht verworfen werden, brauchen die übrigen gar nicht in Betracht gezogen zu werden; wird H_{AB} verworfen, müssen H_A und H_B im marginalen Sinne interpretiert werden; falls H_{AB} nicht verworfen werden kann, sind H_A und H_B äquivalent zu den „stärkeren“ Hypothesen $H_A^{(0)}$ bzw. $H_B^{(0)}$ und die jeweiligen Testergebnisse entsprechend zu interpretieren.

Die konkrete Durchführung der Hypothesentests erfordert die Berechnung der jeweiligen Teststatistiken, womit wir uns in den folgenden Abschnitten befassen. Im Zwei-Faktoren-Modell unterscheiden sich balancierter und unbalancierter Fall allerdings in ihrer formalen Komplexität stärker als im Ein-Faktor-Modell, sodass wir uns der Einfachheit halber (zunächst) auf den balancierten Versuchsplan beschränken. Außerdem gestaltet sich die Darstellung der Tests der bisher im cell means-Modell (95) beschriebenen Hypothesen in der Parametrisierung als Faktoreffekte-Modell etwas anschaulicher, weswegen wir im Folgenden gleich zu jener Parametrisierung übergehen.

11.2.2 Der balancierte Versuchsplan

Im balancierten Versuchsplan ist $n_{jl} \equiv n$, sodass sich Modell (95) vereinfacht zu

$$Y_{jli} = \mu_{jl} + \varepsilon_{jli} \qquad \text{für } i = 1, \ldots, n,\ j = 1, \ldots, J,\ l = 1, \ldots, L, \tag{107}$$

mithin ist $N = JLn$. Es sei $n > 1$, da Inferenz sonst nicht ohne Weiteres möglich ist.

11.2.2.1 Faktoreffekte-Parametrisierung und ihre Schätzer

Das cell means-Modell (107) lässt sich auch als Faktoreffekte-Modell mit Interaktionsterm wie folgt parametrisieren:

$$Y_{jli} = \mu_0 + \alpha_j + \beta_l + (\alpha\beta)_{jl} + \varepsilon_{jli} \quad \text{für } i = 1, \ldots, n,\ j = 1, \ldots, J,\ l = 1, \ldots, L, \tag{108}$$

wobei μ_0 das Gesamtmittel ist, α_j der Effekt des j-ten Levels von Faktor A, β_l der Effekt des l-ten Levels von Faktor B und $(\alpha\beta)_{jl}$ der Interaktionseffekt beim j-ten Level von Faktor A und l-ten Level von Faktor B. (Dabei hat die Notation „$(\alpha\beta)_{jl}$" nichts mit einer Multiplikation zu tun, sondern soll nur darauf hinweisen, zu welchen Faktoren dieser Interaktionsterm gehört; er könnte stattdessen auch γ_{jl} heißen.)

Modell (108) ist überparametrisiert und um die Identifizierbarkeit der Parameter zu gewährleisten, müssen sie (z. B.) die folgende Bedingung erfüllen:

$$\alpha_{\cdot} = \beta_{\cdot} = (\alpha\beta)_{\cdot l} = (\alpha\beta)_{j\cdot} = 0 \quad \text{für alle } l = 1, \ldots, L \text{ und } j = 1, \ldots, J \tag{109}$$

Bemerkungen:

1. Die Beziehungen zwischen den beiden Parametrisierungen (107) und (108) unter der Identifizierungsbedingung (109) lassen sich ableiten, indem die Erwartungswerte der rechten Seiten von (107) und (108) (also ohne ε_{jli}) gleichgesetzt und geeignet summiert werden. Zum Beispiel ist
$$\mu_{\cdot\cdot} = LJ\mu_0 \qquad \text{und somit} \qquad \mu_0 = \bar{\mu}_{\cdot\cdot}$$
oder für ein beliebiges $1 \leq j \leq J$ ist
$$\mu_{j\cdot} = L\mu_0 + L\alpha_j \qquad \text{und somit} \qquad \alpha_j = \bar{\mu}_{j\cdot} - \mu_0 = \bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot\cdot} \tag{110}$$
Analog folgt $\beta_l = \bar{\mu}_{\cdot l} - \bar{\mu}_{\cdot\cdot}$ und schließlich $(\alpha\beta)_{jl} = \mu_{jl} - \bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot l} + \bar{\mu}_{\cdot\cdot}$

2. Die Faktoreffekte-KQS $\hat{\mu}_0$, $\hat{\alpha}_j$, $\hat{\beta}_l$ und $\widehat{(\alpha\beta)}_{jl}$ für alle j und l ergeben sich durch Lösen der jeweiligen Minimierungsprobleme unter Beachtung der Identifizierungsbedingung (109). Es zeigt sich, dass man sie durch Einsetzen der cell means-KQS $\hat{\mu}_{jl} = \bar{Y}_{jl\cdot}$ in die Beziehungen zwischen den Parametrisierungen (107) und (108) erhält. Die folgende Tabelle enthält diese Beziehungen sowie die entsprechenden KQS:

Parameterbeziehungen	Kleinste-Quadrate-Schätzer
$\mu_0 = \bar{\mu}_{\cdot\cdot}$	$\hat{\mu}_0 = \bar{Y}_{\cdot\cdot\cdot}$
$\alpha_j = \bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot\cdot}$	$\hat{\alpha}_j = \bar{Y}_{j\cdot\cdot} - \bar{Y}_{\cdot\cdot\cdot}$
$\beta_l = \bar{\mu}_{\cdot l} - \bar{\mu}_{\cdot\cdot}$	$\hat{\beta}_l = \bar{Y}_{\cdot l\cdot} - \bar{Y}_{\cdot\cdot\cdot}$
$(\alpha\beta)_{jl} = \mu_{jl} - \bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot l} + \bar{\mu}_{\cdot\cdot}$	$\widehat{(\alpha\beta)}_{jl} = \bar{Y}_{jl\cdot} - \bar{Y}_{j\cdot\cdot} - \bar{Y}_{\cdot l\cdot} + \bar{Y}_{\cdot\cdot\cdot}$
$= \mu_{jl} - (\bar{\mu}_{\cdot\cdot} + \bar{\mu}_{j\cdot} - \bar{\mu}_{\cdot\cdot} + \bar{\mu}_{\cdot l} - \bar{\mu}_{\cdot\cdot})$	$= \bar{Y}_{jl\cdot} - (\bar{Y}_{\cdot\cdot\cdot} + \bar{Y}_{j\cdot\cdot} - \bar{Y}_{\cdot\cdot\cdot} + \bar{Y}_{\cdot l\cdot} - \bar{Y}_{\cdot\cdot\cdot})$

11.2.2.2 Die „interessanten“ Hypothesen

Aufgrund der in (110) und der in der Zeile direkt darunter angegebenen Beziehung zwischen den beiden Parametrisierungen gelten für die in §11.2.1.1 besprochenen „interessanten“ Hypothesen die folgenden Äquivalenzen:

- $H_{AB}^{(2)} \Longleftrightarrow (\alpha\beta)_{jl} = 0$ für alle $j = 1, \ldots, J$ und $l = 1, \ldots, L$.
  Bedeutung: Keine A-B-Interaktion, d. h., der Einfluss des einen Faktors hängt nicht vom anderen Faktor ab.

- $H_A^{(2)} \Longleftrightarrow \alpha_1 = \ldots = \alpha_J = 0$.
  Bedeutung: Falls H_{AB} erfüllt: Faktor A ohne Einfluss. Falls H_{AB} nicht erfüllt: Faktor A im Durchschnitt über alle B-Levels ohne Einfluss (= ohne marginalen Einfluss).

- $H_B^{(2)} \Longleftrightarrow \beta_1 = \ldots = \beta_L = 0$.
  Bedeutung: Analog zu $H_A^{(2)}$.

- $H_0 \Longleftrightarrow \alpha_j = 0$ und $\beta_l = 0$ und $(\alpha\beta)_{jl} = 0$ für alle $j = 1, \ldots, J$ und $l = 1, \ldots, L$.
  Bedeutung: Weder A noch B haben einen Einfluss.

Jede der vier Hypothesen H_0, H_{AB}, H_A und H_B lässt sich unter Verwendung einer geeigneten Hypothesenmatrix $\mathbf{C}$ als eine lineare Bedingung an den cell means-Vektor $\boldsymbol{\mu} = (\mu_{11}, \ldots, \mu_{JL})'$ schreiben. Um eine solche Hypothese $H_* : \mathbf{C}\boldsymbol{\mu} = \mathbf{0}$ zu testen, müssen – wie üblich – die Residuenquadratesumme RSS_{H_*} unter dieser Hypothese H_* und der Rang von $\mathbf{C}$ bestimmt werden, damit die übliche F-Teststatistik der Art (78) zur Anwendung kommen kann (mit der RSS aus (96)).

Der Rang der Matrix $\mathbf{C}$ wird je nach Hypothese Freiheitsgrad des Faktors bzw. des Interaktionseffektes (oder kurz: Freiheitsgrad der Hypothese) genannt.

Wir tabellieren die für die betrachteten vier Hypothesen notwendigen Ergebnisse – für den balancierten Fall – ohne Herleitung (eine solche findet sich z. B. in [71, Seber (1977)], §9.2.2):

Hypothese	SS_* $:= \mathrm{RSS}_{H_*} - \mathrm{RSS}$	Rang($\mathbf{C}$) (= Freiheitsgrade)
H_0	$\mathrm{SS}_0 := n \sum_{j=1}^{J} \sum_{l=1}^{L} (\bar{Y}_{jl\cdot} - \bar{Y}_{\ldots})^2$	$JL - 1$
H_{AB}	$\mathrm{SS}_{AB} := n \sum_{j=1}^{J} \sum_{l=1}^{L} \widehat{(\alpha\beta)}_{jl}^2$	$(J-1)(L-1)$
H_A	$\mathrm{SS}_A := nL \sum_{j=1}^{J} \hat{\alpha}_j^2$	$(J-1)$
H_B	$\mathrm{SS}_B := nJ \sum_{l=1}^{L} \hat{\beta}_l^2$	$(L-1)$

11.2.2.3 Orthogonale Varianzzerlegung und ANOVA-Tabelle

Die Interpretation der in der obigen Tabelle auftretenden Quadratsummen SS_* wird durch die folgenden Überlegungen erleichtert: Die Abweichung einer Beobachtung Y_{jli} vom Gesamtmittelwert $\bar{Y}_{\ldots}$ (also vom KQS im Modell ohne jegliche Faktoreinflüsse $Y_{jli} = \mu_0 + \varepsilon_{jli}$) kann wie folgt in die Abweichung des Faktorstufen-Mittelwertes vom Gesamtmittelwert und die Abweichung der Beobachtung vom Faktorstufen-Mittelwert zerlegt werden:

$$Y_{jli} - \bar{Y}_{\ldots} \quad = \quad \bar{Y}_{jl\cdot} - \bar{Y}_{\ldots} + \underbrace{Y_{jli} - \bar{Y}_{jl\cdot}}_{\equiv\ \hat{\varepsilon}_{jli}}$$

Weiter wird $\bar{Y}_{jl\cdot} - \bar{Y}_{\cdots}$ (= Abweichung des Faktorstufen-Mittelwertes vom Gesamtmittelwert) in die Anteile der Faktoreffekte und des Interaktionseffekts zerlegt:

$$\bar{Y}_{jl\cdot} - \bar{Y}_{\cdots} = \underbrace{\bar{Y}_{j\cdot\cdot} - \bar{Y}_{\cdots}}_{\text{A-Effekt}} + \underbrace{\bar{Y}_{\cdot l\cdot} - \bar{Y}_{\cdots}}_{\text{B-Effekt}} + \underbrace{\bar{Y}_{jl\cdot} - \bar{Y}_{j\cdot\cdot} - \bar{Y}_{\cdot l\cdot} + \bar{Y}_{\cdots}}_{\text{AB-Interaktionseffekt}} \equiv \hat{\alpha}_j + \hat{\beta}_l + \widehat{(\alpha\beta)}_{jl}$$

Entsprechend der obigen Zerlegungen kann die Gesamtstreuung in den Beobachtungen

$$\mathrm{SS}_{Total} = \sum_{j=1}^{J}\sum_{l=1}^{L}\sum_{i=1}^{n}(Y_{jli} - \bar{Y}_{\cdots})^2 \tag{111}$$

in Anteile der (Haupt-)Effekte, des Interaktionseffekts und der (durch das Modell nicht weiter erklärbaren) Residuen zerlegt werden, da sich beim Summieren der quadrierten Zerlegung die gemischten Produkte eliminieren. **Dies gilt jedoch nur im balancierten Fall; im Allgemeinen ist es falsch!** Man erhält die orthogonale Zerlegung der Gesamtstreuung:

$$\mathrm{SS}_{Total} = \underbrace{nL\sum_{j=1}^{J}\hat{\alpha}_j^2}_{\equiv\ \mathrm{SS}_A} + \underbrace{nJ\sum_{l=1}^{L}\hat{\beta}_l^2}_{\equiv\ \mathrm{SS}_B} + \underbrace{n\sum_{j=1}^{J}\sum_{l=1}^{L}\widehat{(\alpha\beta)}_{jl}^2}_{\equiv\ \mathrm{SS}_{AB}} + \underbrace{\sum_{j=1}^{J}\sum_{l=1}^{L}\sum_{i=1}^{n}\hat{\varepsilon}_{jli}^2}_{\equiv\ \mathrm{RSS}} \tag{112}$$

und offenbar ist $\mathrm{SS}_0 = \mathrm{SS}_A + \mathrm{SS}_B + \mathrm{SS}_{AB}$.

Diese "sums of squares" werden üblicherweise in einer ANOVA-Tabelle dokumentiert:

Streuungsquelle (source of variation)	Freiheitsgrade (degrees of freedom)	Summe der Abweichungsquadrate (sums of squares)	Mittlere Abweichungsquadratesumme (mean squares)	F-Stat. (H_*)
Zwischen den Faktorstufen (between treatments)	$JL-1$	$\mathrm{SS}_0 = n\sum_{j=1}^{J}\sum_{l=1}^{L}(\bar{Y}_{jl\cdot} - \bar{Y}_{\cdots})^2$	$S_0^2 = \frac{\mathrm{SS}_0}{JL-1}$	$\frac{S_0^2}{S^2}$ (H_0)
Faktor-A-Haupteffekte (A main effects)	$J-1$	$\mathrm{SS}_A = nL\sum_{j=1}^{J}\hat{\alpha}_j^2$	$S_A^2 = \frac{\mathrm{SS}_A}{J-1}$	$\frac{S_A^2}{S^2}$ (H_A)
Faktor-B-Haupteffekte (B main effects)	$L-1$	$\mathrm{SS}_B = nJ\sum_{l=1}^{L}\hat{\beta}_l^2$	$S_B^2 = \frac{\mathrm{SS}_B}{L-1}$	$\frac{S_B^2}{S^2}$ (H_B)
A-B-Interaktionseffekte (AB interactions)	$(J-1)(L-1)$	$\mathrm{SS}_{AB} = n\sum_{j=1}^{J}\sum_{l=1}^{L}\widehat{(\alpha\beta)}_{jl}^2$	$S_{AB}^2 = \frac{\mathrm{SS}_{AB}}{(J-1)(L-1)}$	$\frac{S_{AB}^2}{S^2}$ (H_{AB})
Innerhalb der Faktorstufen (residuals, ...)	$JL(n-1)$	$\mathrm{RSS} = \sum_{j=1}^{J}\sum_{l=1}^{L}\sum_{i=1}^{n}(Y_{jli} - \bar{Y}_{jl\cdot})^2$	$S^2 = \frac{\mathrm{RSS}}{JL(n-1)}$	
Gesamtstreuung (total variation)	$JLn-1$	$\mathrm{SS}_{Total} = \mathrm{RSS}_{H_0} = \sum_{j=1}^{J}\sum_{l=1}^{L}\sum_{i=1}^{n}(Y_{jli} - \bar{Y}_{\cdots})^2$		

11.2.2.4 Aufbereitung der Daten und explorative Analyse

Wir verwenden als **Beispiel** wieder einen Datensatz aus Box, Hunter und Hunter (1978): Nebenstehend sind Überlebenszeiten (in Einheiten zu zehn Stunden) angegeben, die im Rahmen eines Tierexperiments für die Kombination von drei Giftstoffen I, II und III und vier verschiedenen Behandlungsstoffen A, B, C und D ermittelt wurden. Für jede Kombination aus Giftstoff und Behandlungsstoff wurden vier Tiere verwendet, also vier Wiederholungen durchgeführt. (Offenbar ist es ein balancierter Versuchsplan.)

	Behandlung			
Gift	A	B	C	D
I	0.31	0.82	0.43	0.45
	0.45	1.10	0.45	0.71
	0.46	0.99	0.63	0.66
	0.43	0.72	0.76	0.62
II	0.36	0.92	0.44	0.56
	0.29	0.61	0.35	1.02
	0.40	0.49	0.31	0.71
	0.23	1.24	0.40	0.38
III	0.22	0.30	0.23	0.30
	0.21	0.37	0.25	0.36
	0.18	0.38	0.24	0.31
	0.23	0.29	0.22	0.33

Der Vektor **Y** aller Responses und die zwei Vektoren der dazugehörigen Faktorlevels sind als Spalten in einem Data Frame zusammenzufassen, und zwar dergestalt, dass jede Zeile des Data Frames eine beobachtete Response und die hierfür gültige Faktorlevel-Kombination enthält. Hierbei ist die Funktion `expand.grid()` behilflich, die wir bereits auf Seite 51 in Abschnitt 10.9.1 kennengelernt haben. Sie erzeugt aus den an sie übergebenen Vektoren einen Data Frame aller Kombinationen der Elemente dieser Vektoren, wobei nicht-numerische Vektoren zu Faktoren konvertiert werden.

Auf diese Weise generieren wir, wie unten zu sehen, aus den beiden `character`-Vektoren `LETTERS[ 1:4]` und `c( "I", "II", "III")` zunächst einen Data Frame `Gift.df` mit den zwei Faktoren `Behandlung` und `Gift` als Komponenten. Er enthält alle $JL = 4 \cdot 3 = 12$ Levelkombinationen in seinen Zeilen und stellt somit das Grunddesign des Versuchsplans dar. Dieses Grunddesign muss nun n-mal vollständig repliziert werden (hier $n = 4$), um die (balancierte!) n-fache Wiederholung einer jeden Levelkombination zu erreichen. Dazu wird der Inhalt des 12-zeiligen (!) Data Frames `Gift.df` in die bis zu diesem Zeitpunkt *noch nicht* existierenden Zeilen $JL + 1$ bis JLn (also 13 bis 48) eben dieses selben Data Frames geschrieben, wodurch diese Zeilen „entstehen“ und mit dem (zyklisch replizierten) Grunddesign gefüllt werden. Danach hat `Gift.df` $JLn = 48$ Zeilen:

```
> Gift.df <- expand.grid( Behandlung = LETTERS[ 1:4],
+                         Gift = c( "I", "II", "III"))
> JL <- nrow( Gift.df);       n <- 4
> Gift.df[ (JL + 1):(n * JL),] <- Gift.df;     Gift.df
```

```
   Behandlung Gift
 1          A    I
 2          B    I
 3          C    I
 4          D    I
 5          A   II
.... ....
12          D  III
13          A    I
.... ....
48          D  III
```

Eine Tabellierung des Inhalts von `Gift.df` zur Kontrolle bestätigt die Balanciertheit des Versuchsplans:

```
> table( Gift.df)
           Gift
Behandlung I II III
         A 4  4   4
         B 4  4   4
         C 4  4   4
         D 4  4   4
```

Hinweis: Zur Generierung regelmäßig strukturierter Faktoren für balancierte Versuchspläne steht in **R** auch die Funktion `gl()` (für “*g*enerate *l*evels”) zur Verfügung. Sie erwartet für ihr Argument `n` die Anzahl der Levels des Faktors, für `k` die Anzahl der Wiederholungen eines jeden

Levels und für `length` die Gesamtlänge des Ergebnisses. Durch das Argument `labels` können die Faktorlevels beliebige Benennungen erhalten. Damit kann obiges Gesamtdesign auch wie folgt und vor allem einfacher erzeugt werden:

```
> Gift.df <- data.frame(
+  Behandlung = gl( n = 4, k = 1, length = 48, labels = LETTERS[ 1:4]),
+  Gift =       gl( n = 3, k = 4, length = 48, labels = c( "I", "II", "III")) )
```

Nun muss noch der Vektor der Response-Werte (als Komponente `Survival`) zum Data Frame des Versuchsplans hinzugefügt werden. Dabei ist darauf zu achten, dass die Reihenfolge der Vektorelemente zum Arrangement des Versuchsplans in `Gift.df` passt:

```
> Gift.df$Survival <- c(
+ 0.31, 0.82, 0.43, 0.45,   0.36, 0.92, 0.44, 0.56,   0.22, 0.30, 0.23, 0.30,
....
+ 0.43, 0.72, 0.76, 0.62,   0.23, 1.24, 0.40, 0.38,   0.23, 0.29, 0.22, 0.33)
> Gift.df
   Behandlung Gift Survival
 1          A    I     0.31
 2          B    I     0.82
 3          C    I     0.43
 4          D    I     0.45
 5          A   II     0.36
....
 8          D   II     0.56
 9          A  III     0.22
....
12          D  III     0.30
13          A    I     0.45
.... ....
48          D  III     0.33
```

Wie in §11.1.2 werden für die explorative Datenanalyse wieder `plot.design()` und `plot()` (mit einer geeigneten Modellformel) verwendet, aber nun kommt auch noch eine Methode zur Entdeckung möglicher Interaktionen zum Einsatz:

- `plot.design()` erlaubt eine erste Beurteilung, wie die Levels des einen Faktors auf die über die Levels des anderen Faktors gemittelte Response wirken. Konkret werden dazu die marginalen Mittelwerte $\bar{Y}_{j\cdot\cdot}$ für $j = 1, \ldots, J$ auf einer vertikalen Skala markiert, ebenso $\bar{Y}_{\cdot l\cdot}$ für $l = 1, \ldots, L$ sowie der Gesamtmittelwert $\bar{Y}_{\cdots}$. Dies geschieht in einem gemeinsamen Koordinatensystem (Plot auf Seite 308 links oben). (Dabei ist $\bar{Y}_{j\cdot\cdot}$ der KQS für $\bar{\mu}_{j\cdot}$, $\bar{Y}_{\cdot l\cdot}$ für $\bar{\mu}_{\cdot l}$ und $\bar{Y}_{\cdots}$ für $\bar{\mu}_{\cdot\cdot} = \mu_0$.)

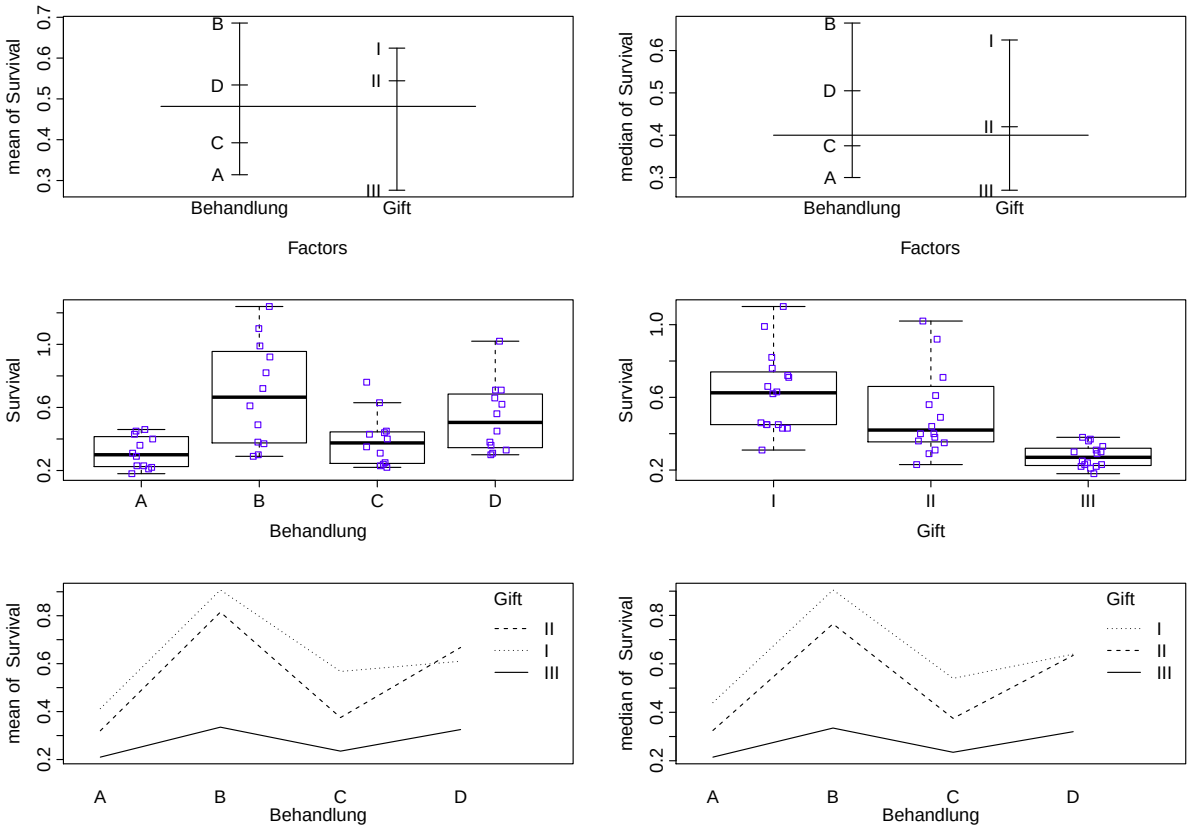
- `plot()` mit Verwendung der entsprechenden Modellformel erlaubt darüber hinaus die Beurteilung der Homoskedastizitätsannahme, da für jedes Level des einen Faktors ein Boxplot aller dazugehörigen (über die Levels des anderen Faktors gepoolten) Response-Werte geplottet wird. Konkret wird für jedes $j = 1, \ldots, J$ ein Boxplot für $\{Y_{jli} : l = 1, \ldots, L,\ i = 1, \ldots, n\}$ gezeichnet, wobei diese J Boxplots in ein gemeinsames Koordinatensystem kommen; ebensolches geschieht für jedes $l = 1, \ldots, L$ für $\{Y_{jli} : j = 1, \ldots, J,\ i = 1, \ldots, n\}$ (mittlere Plots auf Seite 308).

  Mithin lassen sich durch die beiden obigen Methoden die Faktoreffekte separat begutachten, allerdings im marginalen Sinne.

- Für die Darstellung potenzieller Interaktionseffekte steht ebenfalls ein exploratives Werkzeug zur Verfügung: Die empirische Variante des bereits in §11.2.1.2 vorgestellten Interaktionsplots. Zur Erinnerung: Unter der Hypothese keiner Interaktion, also unter H_{AB}, sind die Wirkungsprofile $(\mu_{1l}, \ldots, \mu_{Jl})$ und $(\mu_{1l'}, \ldots, \mu_{Jl'})$ des Faktors A „parallel" für zwei verschiedene B-Levels, und damit *alle* Profile von Faktor A. Eben dies gilt analog für die Profile $(\mu_{j1}, \ldots, \mu_{jL})$ des Faktors B.

Dieser Sachverhalt sollte sich in einem Plot widerspiegeln, in dem für jedes Level $l = 1, \ldots, L$ des Faktors B die *empirischen* Profile $(\bar{Y}_{1l\cdot}, \ldots, \bar{Y}_{Jl\cdot})$ des Faktors A gegen $j = 1, \ldots, J$ gezeichnet werden. (Es kommen also die KQS $\bar{Y}_{jl\cdot}$ für μ_{jl} zum Einsatz.) Indem jedes der L A-Profile durch einen eigenen Polygonzug repräsentiert wird, sollte unter H_{AB} – in etwa – Parallelität dieser Polygonzüge zu beobachten sein. Völlig analog gilt dies für einen Plot der J B-Profile. (Siehe die beiden unteren Plots auf dieser Seite.)

Diese empirischen Interaktionsplots können durch die Funktion `interaction.plot()` erzeugt werden. Als ihr erstes Argument erwartet sie den Vektor des Faktors, dessen Profile gezeichnet werden sollen, d. h., dessen Levels auf der waagrechten Achse abzutragen sind, als zweites den Vektor des Faktors, für dessen Levels die Profile des ersten Faktors gebildet werden, und als drittes den Vektor der Werte, die auf der senkrechten Achse abgetragen werden. (`interaction.plot()` kann mehr, wie man in der Online-Hilfe und auch noch in einem der folgenden Abschnitte erfährt.)



Die folgende Tabelle demonstriert im vorliegenden Gift-Beispiel die Erstellung der drei oben beschriebenen und abgebildeten, explorativen Plot-Typen:

Zweifaktorielle Varianzanalyse: Explorative Datenanalyse	
`> plot.design( Gift.df)` `> plot.design( Gift.df, fun = median)`	Liefert den mittelwertbasierten Designplot (linker oberer Plot) bzw. medianbasierten Designplot (rechter oberer Plot).
`> plot( Survival ~ Behandlung + Gift,` `+ data = Gift.df)`	Erstellt die zwei Faktorplots (Mitte), aber *ohne* die Rohdaten; dazu müsste jeder Plot separat angefertigt und sie überlagert werden.

`> with( Gift.df, {`
`+ interaction.plot( Behandlung, Gift,`
`+  Survival)`
`+ interaction.plot( Behandlung, Gift,`
`+  Survival, fun = median) })`
	Der Interaktionsplot links unten zeigt die zuvor beschriebenen, mittelwertbasierten Profile; derjenige rechts unten medianbasierte, was durch `fun = median` erreicht wird. (`with()` entlastet nur die Notation.)

Das qualitative Fazit der explorativen Datenanalyse lautet:

- Die Designplots (obere Plots auf vorheriger Seite) zeigen für die Behandlung-Levels A und C kürzere und für D und B längere mittlere bzw. mediane Überlebenszeiten. Die Gift-Levels I, II und III verringern in dieser Reihenfolge die mittleren bzw. medianen Überlebenszeiten.

  Der Vergleich von "mean"- und "median"-Designplots zeigt keine starken Unterschiede. Lediglich im Gift-Level II deutet der Unterschied zwischen arithmetischem Mittel und Median auf eine schiefe Verteilung oder (mindestens) einen potenziellen Ausreißer hin.

- Die Boxplots (mittlere Plots auf vorheriger Seite) deuten klar auf eine größere Response-Variabilität bei höherem Response-Median hin (und somit auf eine mögliche Varianzinhomogenität).

- Die nicht parallelen Verläufe in den Interaktionsplots (untere Plots auf vorheriger Seite) zeigen stärkere Effekte der Behandlung bei den Giften I und II (welche diejenigen mit den höheren mittleren Überlebensdauern bzw. deren Medianen sind) als bei Gift III, was auf eine Interaktion der beiden Faktoren hindeutet.

11.2.2.5 Test auf Varianzhomogenität

Wünschenswert ist es, auch hier einen Test auf Homoskedastizität, wie z. B. Bartletts Test (vgl. §11.1.5) einsetzen zu können. Dabei ist zu beachten, dass die zu testende Modellannahme lautet, dass die Varianz der Fehler in *jeder* Zelle (j, l), also über alle Levelkombinationen hinweg, dieselbe ist. Einen Test einfach für jeden Faktor getrennt anzuwenden, wie in `bartlett.test( Survival ~ Gift, ....)` und in `bartlett.test( Survival ~ Behandlung, ....)`, wäre falsch, weil damit nur die „marginalen" Varianzen verglichen würden. Selbst die Modellformeln `Survival ~ Gift + Behandlung` und `Survival ~ Gift * Behandlung` erzielen *nicht* das eigentlich Gewünschte, weil sie durch `bartlett.test()` nicht entsprechend aufgelöst werden, sondern nur der jeweils erste Term auf der rechten Seite der Modellformel berücksichtigt wird (hier also `Gift`).

Zielführend ist hingegen der Einsatz der Funktion `interaction()`, die zu gegebenen Faktorvektoren den Vektor ihrer „Interaktionen" bestimmt. Zum Beispiel im Fall zweier Faktorvektoren `A` und `B` gleicher Länge k mit Levels `a1` bis `aJ` bzw. `b1` bis `bL` geschieht dies, indem ein Faktorvektor der Länge k gebildet wird, der alle $J \cdot L$ (potenziellen) Kombinationen der Levels der beiden Faktoren als Levels `a1.b1`, `a1.b2`, ..., `a1.bL`, `a2.b1`, ..., `aJ.bL` hat und dessen i-tes Element $(1 \leq i \leq k)$ durch `paste( A[i], B[i], sep = ".")` entsteht. (Bei mehr als zwei Faktoren wird analog verfahren. Sind die Längen der Faktorvektoren nicht gleich, so wird der kürzere, bzw. werden die kürzeren zyklisch repliziert.)

Der angestrebte Test auf Homoskedastizität kann also mit `interaction( Behandlung, Gift)` auf der rechten Seite der Modellformel realisiert werden:

```
> bartlett.test( Survival ~ interaction( Behandlung, Gift), data = Gift.df)
```

11.2.2.6 Durchführung der Varianzanalyse: `aov()`

Das Zwei-Faktoren-Modell wird als Modell (108) natürlich auch durch die Funktion `aov()` gefittet, wobei es bei ungeordneten Faktoren intern (wie stets) zu einer Reparametrisierung mittels der *Treatment-Kontraste* kommt, allerdings ohne, dass es sich für uns bemerkbar macht. `summary()` auf das resultierende `aov`-Objekt angewendet liefert eine ANOVA-Tabelle. Auf das `aov`-Objekt gehen wir hier nicht nochmal näher ein, sondern verweisen auf §11.1.2.

Zweifaktorielle Varianzanalyse: Die ANOVA-Tabelle

```
> summary( Gift.aov <- aov( Survival ~ Gift * Behandlung, data = Gift.df))
                Df  Sum Sq Mean Sq F value    Pr(>F)
Gift             2 1.06390 0.53195 23.6482 2.768e-07 ***
Behandlung       3 0.96537 0.32179 14.3053 2.695e-06 ***
Gift:Behandlung  6 0.26600 0.04433  1.9708   0.09576 .
Residuals       36 0.80980 0.02249
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Es wird das zweifaktorielle Modell mit Interaktion (108) von `Survival` an `Gift` und `Behandlung` (aus `Gift.df`) gefittet und in `Gift.aov` abgelegt. Mit `summary( Gift.aov)` erhält man eine ANOVA-Tabelle, in der, wie an der Zeilenbenennung zu erkennen, die Zerlegung (112) dargestellt ist (aber ohne SS_0):

- In der Spalte `Df` stehen die Freiheitsgrade der zwei Faktoren (also $J-1$ bzw. $L-1$), der Interaktion ($(J-1)(L-1)$) und der RSS ($N-JL$).
- Unter `Sum Sq` stehen die Summen der Abweichungsquadrate aus (112) ($= \mathrm{SS}_* \equiv \mathrm{RSS}_{H_*} - \mathrm{RSS}$ bzw. RSS).
- Daneben, in Spalte `Mean Sq`, befinden sich die mittleren Summen dieser Quadrate (d. h. $S_*^2 = \mathrm{SS}_*/\text{Freiheitsgrade}$ bzw. $S^2 = \mathrm{RSS}/(N-JL)$).
- Es folgen die Werte der F-Teststatistiken S_*^2/S^2 (`F value`) der Hypothesen „Kein Effekt“ (also H_A sowie H_B) bzw. „Keine Interaktion“ (H_{AB}) und deren p-Werte (`Pr(>F)`).
- Zur Interpretation der Testergebnisse siehe §11.2.1.1.

Bemerkungen:

- Mit `anova( Gift.aov)` erhält man dieselbe Ausgabe.
- Die erhaltenen Resultate sind unabhängig von der Reihenfolge der Modellterme, weil dieses Beispiel ein *balanciertes* Design hat und hier sog. Typ-I-Tests durchgeführt werden. (Typ-I-Test werden auch sequentielle Test genannt, auf deren Konzept und Vergleich mit Typ-II- und Typ-III-Tests wir später noch einmal kurz eingehen.)

  In balancierten Designs werden die Quadratesummen orthogonal zerlegt, was induziert, dass die darauf basierenden F-Teststatistiken invariant gegenüber der Reihenfolge der Modellterme sind. D. h., `aov( Survival ~ Behandlung * Gift, data = Gift.df)` würde hier dieselben Testergebnisse liefern.
- Der Test von H_0 wird in obiger Ausgabe von `summary( Gift.aov)` offenbar nicht dokumentiert. Allerdings ist er faktisch der globale F-Test im zugrundeliegenden linearen Modell und man erhält ihn daher z. B. durch den Aufruf der `lm`-Methode von `summary()` angewendet auf das interessierende `aov`-Objekt, hier also durch `summary.lm( Gift.aov)`. Ein anderer Weg ist der direkte Vergleich des Null- mit dem vollen Modell durch `anova( Gift0.aov, Gift.aov)`, wobei `Gift0.aov <- aov( Survival ~ 1, data = Gift.df)` ist.

11.2.2.7 Parameterschätzer mit `model.tables()`

Zur expliziten Ermittlung von $\hat{\alpha}_j$, $\hat{\beta}_l$ und $\widehat{(\alpha\beta)}_{jl}$ als Schätzwerte für die Effekte dient schließlich wieder `model.tables()`:

Zweifaktorielle Varianzanalyse: Die Parameterschätzer

```
> model.tables( Gift.aov)
Tables of effects

 Gift
Gift
       I       II      III
 0.14271  0.06271 -0.20542

 Behandlung
Behandlung
        A        B        C        D
 -0.16750  0.20417 -0.08917  0.05250

 Gift:Behandlung
     Behandlung
Gift  A        B        C        D
  I   -0.04437  0.07896  0.03229 -0.06688
  II  -0.05688  0.06646 -0.08021  0.07063
  III  0.10125 -0.14542  0.04792 -0.00375
```

Liefert voreinstellungsgemäß die KQS für die Effekte im Faktoreffekte-Modell (108):

$\hat{\alpha}_1, \ldots, \hat{\alpha}_J$

$\hat{\beta}_1, \ldots, \hat{\beta}_L$

$\widehat{(\alpha\beta)}_{11} \quad \ldots \quad \widehat{(\alpha\beta)}_{1L}$
$\vdots \qquad\qquad \vdots$
$\widehat{(\alpha\beta)}_{J1} \quad \ldots \quad \widehat{(\alpha\beta)}_{JL}$

Wie man schnell nachrechnet, ist die Identifizierungsbedingung (109) bis auf Rundungsfehler erfüllt.

```
> model.tables( Gift.aov, type = "means")
Tables of means
Grand mean

0.4816667

 Gift
Gift
     I     II    III
0.6244 0.5444 0.2763

 Behandlung
Behandlung
     A      B      C      D
0.3142 0.6858 0.3925 0.5342

 Gift:Behandlung
     Behandlung
Gift  A      B      C      D
  I   0.4125 0.9075 0.5675 0.6100
  II  0.3200 0.8150 0.3750 0.6675
  III 0.2100 0.3350 0.2350 0.3250
```

Das Argument `type = "means"` erwirkt die Ausgabe von KQS für das cell means-Modell (107):

$\bar{Y}_{\cdot\cdot\cdot} (= \hat{\mu}_0)$ für das Gesamtmittel $\bar{\mu}_{\cdot\cdot}$ $(= \mu_0$; "Grand mean"),

$\bar{Y}_{1\cdot\cdot}, \ldots, \bar{Y}_{J\cdot\cdot}$ für die marginalen (!) Faktorlevel-Mittelwerte $\bar{\mu}_{1\cdot}, \ldots, \bar{\mu}_{J\cdot}$ (wobei $\bar{\mu}_{j\cdot} = \mu_0 + \alpha_j$),

$\bar{Y}_{\cdot 1\cdot}, \ldots, \bar{Y}_{\cdot L\cdot}$ für die marginalen (!) Faktorlevel-Mittelwerte $\bar{\mu}_{\cdot 1}, \ldots, \bar{\mu}_{\cdot L}$ (wobei $\bar{\mu}_{\cdot l} = \mu_0 + \beta_l$) sowie

$\bar{Y}_{11\cdot} \quad \ldots \quad \bar{Y}_{1L\cdot}$
$\vdots \qquad\qquad \vdots$
$\bar{Y}_{J1\cdot} \quad \ldots \quad \bar{Y}_{JL\cdot}$

für die cell means $\mu_{jl} = \mu_0 + \alpha_j + \beta_l + (\alpha\beta)_{jl}$.

11.2.2.8 Modelldiagnose

Erläuterungen zur Modelldiagnose finden sich in §11.1.4. Beachte auch hier den Einsatz von `plot()` für das `aov`-Objekt! Sie ist faktisch die Diagnosefunktion für `lm`-Objekte und liefert so den Normal-Q-Q-Plot und den Plot der Residuen gegen die gefitteten Werte (sodass sie nicht „von Hand“ angefertigt zu werden brauchen):

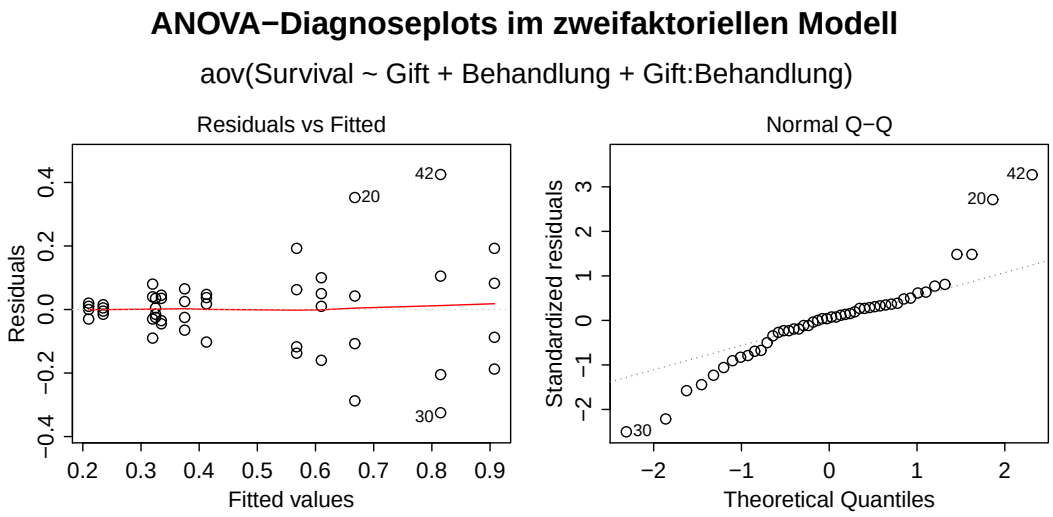
Zweifaktorielle Varianzanalyse: Modelldiagnose(-plots)

`> fitted( Gift.aov)`
`> resid( Gift.aov)`
 	
`> plot( Gift.aov,`
`+ which = 1:2)`
	Gefittete Werte und die Residuen gibt es wieder mit `fitted()` bzw. `resid()`. Einen Plot der Residuen gegen die gefitteten Werte zur Prüfung der Varianzhomogenität sowie einen Normal-Q-Q-Plot der Residuen (zur Beurteilung der Normalverteilungsannahme der Fehler) liefert die Plotfunktion für `lm`-Objekte (siehe Grafiken unten).

```
> signif( coef( Gift.aov), 2)
       (Intercept)              GiftII             GiftIII         BehandlungB
           4.1e-01            -9.3e-02           -2.0e-01             4.9e-01
       BehandlungC         BehandlungD GiftII:BehandlungB GiftIII:BehandlungB
           1.5e-01             2.0e-01            1.2e-16            -3.7e-01
GiftII:BehandlungC GiftIII:BehandlungC GiftII:BehandlungD GiftIII:BehandlungD
          -1.0e-01            -1.3e-01            1.5e-01            -8.2e-02
```

Die Koeffizientenschätzer des dem `aov`-Objekt zugrundeliegenden linearen Modells in der Parametrisierung durch Treatment-Kontraste. Sie erhält man zusammen mit weiteren Informationen auch durch die Anwendung von `summary.lm()` (vgl. auch §11.1.4). Beachte wieder das Rundungsproblem (aufgrund der Maschinengenauigkeit) beim Schätzwert für den Koeffizienten von `GiftII:BehandlungB`!

Fazit: Der Plot der Residuen gegen die gefitteten Werte (siehe linke Grafik unten) offenbart eine deutliche Varianz**in**homogenität, was die inferenzstatistischen Ergebnisse zweifelhaft erscheinen lässt. Hier wäre eine varianzstabilisierende wie z. B. evtl. eine log-Transformation angezeigt (Stichwort „Box-Cox-Transformation“), worauf wir aber nicht eingehen, sondern dazu auf die Literatur verweisen (wie z. B. auf [71, Seber (1977)], ch. 6.7, auf [66, Neter, Wasserman und Kuttner (1990)], pp. 149-150, auf [34, Fox (2002)], ch. 3.4 oder die Originalarbeit “An analysis of transformations” von [8, Box und Cox (1964)]. Die Box-Cox-Transformation ist z. B. in der Funktion `boxcox()` des Paketes `MASS` implementiert, wie auch schon in der Bemerkung auf Seite 214 erwähnt.



11.2.3 Genau eine Beobachtung pro Levelkombination

Wir gehen hier nicht ins Detail, sondern machen nur einige Bemerkungen zum Fall $n = 1$ und liefern ein paar Beispiele:

- Im Modell *mit* Interaktionsterm (`Y ~ A * B`) ist keine Inferenz möglich, weil RSS = 0. Dies ist der Fall, da $Y_{jl} = \mu_{jl} + \varepsilon_{jl}$, sodass die μ_{jl} perfekt an die Y_{jl} angepasst werden können und quasi keine Reststreuung für die ε_{jl} „übrig" bleibt.

  Es existiert allerdings ein approximatives Modell, innerhalb dessen ein exakter Test auf „Keine Interaktion" möglich ist (vgl. [46, Hocking (1996)], §13.1.5) und somit geklärt werden kann, ob für vorliegende Daten nicht doch ein rein additives Modell (also `Y ~ A + B`) ausreichend ist.

  Die Funktion `aov()` fittet zwar auch im Fall $n = 1$ das zweifaktorielle Modell mit Interaktion ohne zu murren, aber in der ANOVA-Tabelle als Resultat von `summary()` finden sich korrekterweise keine Inferenzresultate:

  Beispiel: Das Argument `subset` der Funktion `aov()` erlaubt es, eine Teilmenge der Zeilen des an `data` übergebenen Data Frames zu spezifizieren, um alle Berechnungen nur auf Basis der Daten jener Zeilen durchzuführen. Im folgenden Beispiel werden nur die ersten 12 Zeilen von `Gift.df` ausgewählt und demnach ein Versuchsplan mit genau einer Beobachtung für jede Faktorlevel-Kombination. Dafür wird dann ein Modell mit Interaktion gefittet:

```
> summary( aov( Survival ~ Gift * Behandlung, data = Gift.df, subset = 1:12))
                Df   Sum Sq  Mean Sq
Gift             2 0.208950 0.104475
Behandlung       3 0.250300 0.083433
Gift:Behandlung  6 0.084850 0.014142
```

- Im Modell *ohne* Interaktionsterm (`Y ~ A + B`), also dem rein additiven Modell, ist Inferenz bezüglich der Faktoreffekte möglich und die `summary` des `aov`-Objektes enthält auch die jeweiligen p-Werte:

```
> summary( aov( Survival ~ Gift + Behandlung, data = Gift.df, subset = 1:12))
            Df   Sum Sq  Mean Sq F value  Pr(>F)
Gift         2 0.208950 0.104475  7.3877 0.02409 *
Behandlung   3 0.250300 0.083433  5.8998 0.03193 *
Residuals    6 0.084850 0.014142
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

- Die Tatsache, nur eine Beobachtung pro Faktorlevel-Kombination zu haben und deswegen keinen Interaktionstest durchführen zu können, rechtfertigt *nicht* die Annahme des Fehlens einer Interaktion! Diese Annahme muss durch „außer"-statistische, sachliche Überlegungen begründet sein. Die Interaktionsplots erlauben aber wenigstens eine qualitative Einschätzung des Vorhandenseins oder Fehlens von Interaktion.
  Im Modell des nächsten Abschnitts wird implizit davon ausgegangen, dass keine Interaktion vorhanden ist.

11.2.4 Das einfache, randomisierte Blockexperiment

Bei den bisherigen Betrachtungen sind wir davon ausgegangen, dass die in den Analysen betrachtete Population an Untersuchungseinheiten homogen ist und sich Unterschiede, wenn überhaupt, dann nur aufgrund der Behandlungsstufen manifestieren. Gelegentlich werden die Behandlungen

aber auf bekanntermaßen **in**homogene Mengen von Untersuchungseinheiten angewendet. Und wie überall in der Statistik ist ein (Behandlungs-)Effekt schwer zu entdecken, wenn die Variabilität der Response relativ zur Größe des Effektes zu groß ist; man sagt, dass der Effekt „maskiert" wird. Konkret im Fall der Versuchspläne kann eine große Varianz innerhalb der Faktorlevels (= Behandlungsstufen) den Einfluss der Faktoren maskieren. Ist jedoch eine Störgröße bekannt, die diese Variabilität mitverantwortet, aber selbst nur von nachrangigem Interesse ist, kann dies wie folgt zu einer Verbesserung der Analyse genutzt werden. Wir beschränken uns hier auf den Fall *eines* interessierenden Faktors B mit L Levels und einer Störgröße A mit J verschiedenen Ausprägungen:

Angenommen, man hat $J \cdot L$ Untersuchungseinheiten, sodass für jede Störgrößen-Ausprägung genau L Untersuchungseinheiten vorliegen. Dann teilt man die Untersuchungseinheiten in J Gruppen, genannt Blöcke, gleicher – oder zumindest sehr ähnlicher – Störgrößen-Ausprägung ein. (Daher wird die Störgröße auch Blockbildungsfaktor ("blocking factor") genannt.) *Innerhalb* eines jeden Blocks kann nun jeder der L B-Faktorlevels an genau einer Untersuchungseinheit zur Anwendung kommen. Werden die Untersuchungseinheiten innerhalb der Blöcke zufällig (also randomisiert) den Faktorlevels zugewiesen, so wird dies als einfaches, randomisiertes Blockexperiment bezeichnet. (Beachte: Eine randomisierte Zuweisung von Untersuchungseinheiten zu den Blöcken ist *nicht* möglich, da Störgrößen-Ausprägungen unveränderliche Eigenschaft der Untersuchungseinheiten sind!)

Kann man davon ausgehen, dass B und A nicht interagieren, was typischerweise angenommen wird, lässt sich das einfache Blockexperiment als ein Zwei-Faktoren-Modell ohne Interaktion mit einer Beobachtung pro Levelkombination parametrisieren:

$$Y_{jl} = \mu_0 + \alpha_j + \beta_l + \varepsilon_{jl} \qquad \text{für } j = 1, \ldots, J \quad \text{und} \quad l = 1, \ldots, L, \tag{113}$$

wobei μ_0 wieder das Gesamtmittel ist, α_j der Effekt des j-ten Levels des Blockbildungsfaktors A und β_l der Effekt des l-ten Levels des Behandlungsfaktors B. Die Effekte müssen auch hier wieder Identifizierbarkeitsbedingungen genügen, wie

$$\alpha_{\cdot} = 0 \qquad \text{und} \qquad \beta_{\cdot} = 0$$

Die Inferenz bezüglich eines Einflusses des Behandlungsfaktors läuft in diesem Modell also auf den Test der Hypothese $H_B : \beta_1 = \ldots = \beta_L = 0$ im zweifaktoriellen Modell ohne Interaktion hinaus.

Es zeigt sich ferner, dass Parameterbeziehungen und dazugehörige KQS von Modell (113) analog zu denen sind, die in der Tabelle auf Seite 303 unten aufgelistet sind, jedoch ohne Interaktionsterme und ohne dritten Index. Daraus folgt, dass die Abweichung einer Beobachtung Y_{jl} vom Gesamtmittelwert $\bar{Y}_{\cdot\cdot}$ (also vom KQS im Modell ohne jeglichen Faktor- oder Blockeinfluss) wie folgt zerlegt werden kann:

$$Y_{jl} - \bar{Y}_{\cdot\cdot} \;=\; \underbrace{\bar{Y}_{j\cdot} - \bar{Y}_{\cdot\cdot}}_{\text{Blockeffekt}} + \underbrace{\bar{Y}_{\cdot l} - \bar{Y}_{\cdot\cdot}}_{\text{Behandlungseffekt}} + \underbrace{Y_{jl} - \bar{Y}_{j\cdot} - \bar{Y}_{\cdot l} + \bar{Y}_{\cdot\cdot}}_{\text{RSS}} \equiv \hat{\alpha}_j + \hat{\beta}_l + \hat{\varepsilon}_{jl}$$

Entsprechend zerlegt sich die Gesamtstreuung der Beobachtungen in Anteile der Block- und Faktoreffekte und der (durch das Modell nicht erklärten) Residuen, da sich beim Summieren der quadrierten Gleichung die gemischten Produkte (aufgrund der hier zwangsläufigen Balanciertheit des Designs) eliminieren. Man erhält:

$$\mathrm{SS}_{Total} = \sum_{j=1}^{J}\sum_{l=1}^{L}(Y_{jl} - \bar{Y}_{\cdot\cdot})^2 = \underbrace{L\sum_{j=1}^{J}\hat{\alpha}_j^2}_{\equiv\ \mathrm{SS}_{Block}} + \underbrace{J\sum_{l=1}^{L}\hat{\beta}_l^2}_{\equiv\ \mathrm{SS}_{Beh.}} + \underbrace{\sum_{j=1}^{J}\sum_{l=1}^{L}\hat{\varepsilon}_{jl}^2}_{\equiv\ \mathrm{RSS}} \tag{114}$$

Diese "sums of squares" werden in der ANOVA-Tabelle des einfachen Blockexperiments dokumentiert. (Offenbar sind SS_{AB} und RSS aus (112) zur RSS in (114) „verschmolzen".) Die in der Tabelle auf Seite 303 unten aufgeführten "sums of squares" SS_A und SS_B nebst ihrer Freiheitsgrade kommen wieder in der F-Teststatistik (78) zum Einsatz.

Als **Beispiel** dient uns erneut ein Datensatz aus [9, Box, Hunter und Hunter (1978)]: Er dokumentiert den Ertrag eines gewissen Penicillin-Produktionsprozesses für vier verschiedene Verfahrenstypen (= Faktorlevels, Behandlungsstufen) A, B, C und D. Als eine weitere Einflussgröße wurde die Sorte des verwendeten Rohstoffs "corn steep liquor" ausgemacht, wovon es fünf verschiedene Sorten gab. (Dies ist eine Nährstoffquelle für Mikroorganismen, siehe z. B. Liggett & Koffler: *Corn steep liquor in Microbiology.* Bacteriol Rev. 1948 December; 12(4): 297 – 311.)

Hier stellt sich der Faktor „Sorte" als Störvariable dar, da man in erster Linie am Einfluss des Faktors „Verfahrenstyp" auf den Penicillin-Ertrag interessiert ist. Die Sorte dient somit zur Blockbildung für die registrierten Ertragsmengen.

Von jeder der fünf verschiedenen Sorten "corn steep liquor" wurde eine Ladung in vier Teile unterteilt, die randomisiert den Verfahrenstypen zugewiesen wurden. Es konnte dann jeweils genau ein Durchlauf des Produktionsprozesses durchgeführt und der Ertrag bestimmt werden. Wir haben es also mit einem einfachen, randomisierten Blockexperiment zu tun, dessen Ergebnisse in der folgenden Tabelle zu sehen sind:

		Sorten (= Blöcke)				
		1	2	3	4	5
Verfahrenstyp	A	89	84	81	87	79
	B	88	77	87	92	81
	C	97	92	87	89	80
	D	94	79	85	84	88

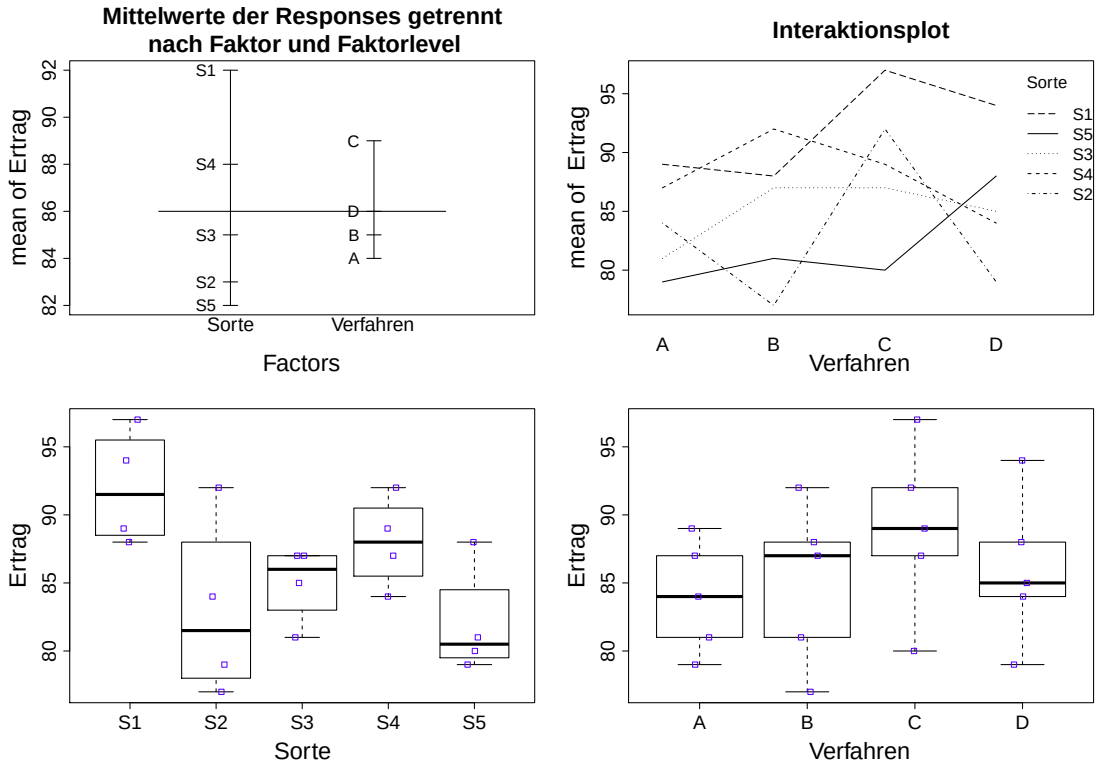
Zunächst sind (wie in §11.2.2.4 auf Seite 306) das Design des Experiments und die Daten so in einem Data Frame zusammenzustellen, dass die gewünschten Levelkombinationen der beiden Faktoren „Sorte" und „Verfahren" in ihrer aufgetretenen Häufigkeit sowie die beobachteten Erträge repräsentiert sind:

```
> ( Pen.df <- data.frame(
+       Sorte = gl( n = 5, k = 1, length = 20,
+                   labels = paste( "S", 1:5, sep = "")),
+   Verfahren = gl( n = 4, k = 5, labels = LETTERS[ 1:4]),
+      Ertrag = c( 89, 84, 81, 87, 79,   88, 77, 87, 92, 81,
+                  97, 92, 87, 89, 80,   94, 79, 85, 84, 88)) )
   Sorte Verfahren Ertrag
1     S1         A     89
2     S2         A     84
3     S3         A     81
4     S4         A     87
5     S5         A     79
6     S1         B     88
....
20    S5         D     88
```

Als zweites sollte sich eine explorative Analyse der Daten anschließen. Die Erklärungen auf Seite 307 treffen auch hier zu (nur eben mit $n = 1$ und ohne dritten Index):

Einfaches Blockexperiment: Explorative Datenanalyse	

`> plot.design( Pen.df)`
`> with( Pen.df,`
`+  interaction.plot( Verfahren, Sorte,`
`+  Ertrag))`
`> plot( Ertrag ~ Verfahren + Sorte,`
`+  data = Pen.df)`
	Liefert die folgenden Design-, und Interaktions- sowie darunter die Faktorplots (wie schon für das zweifaktorielle Modell auf S. 307 beschrieben), aber letzere *ohne* die Rohdaten; dazu müsste jeder Plot separat angefertigt und sie überlagert werden.



Qualitatives Fazit:

- Der Vergleich der beiden Faktorplots (untere „Plot-Reihe“) zeigt, dass die Streuung der Response innerhalb der Verfahren-Levels im rechten Plot (wo über die Sorte-Blöcke gepoolt ist) größer ist als die Response-Streuung innerhalb der Sorte-Blöcke (bis auf Block „S2“). Dies ist ein Indiz dafür, dass die Störvariable Sorte eine größere Variabilität induziert als die interessierende Variable Verfahren.

- Denselben Eindruck vermittelt der Designplot (Plot links oben), in dem die mittleren Erträge zwischen den Sorten stärker variieren als zwischen den Verfahren.

- Keine Interaktion – hier zwischen dem Blockbildungsfaktor Sorte und dem Behandlungsfaktor Verfahren – läge vor, wenn die Verfahren-Profile (Polygonzüge) im Interaktionsplot (rechts oben) idealisierterweise parallel verliefen, d. h. der „profildefinierende Faktor“ (hier: Sorte) einen rein additiven Effekt hätte. Dies scheint hier nicht der Fall zu sein! Allerdings kann dieser Sachverhalt nicht berücksichtigt werden, da qua Design zu wenige Beobachtungen vorliegen (vgl. §11.2.3).

  Außerdem ist im einfachen Blockexperiment die Interpretation eines Interaktionsplots mit Vorsicht vorzunehmen, da hinter den Knoten eines jeden Polygonzuges jeweils nur *eine einzige* Beobachtung steckt. Ein Interaktionseffekt kann somit durch die Streuung der Daten leicht suggeriert, aber auch genauso gut maskiert werden.

Um obiges Modell (113) zu fitten, wird wieder die Funktion `aov()` verwendet, wobei es bei den ungeordneten Faktoren intern (wie zuvor) zu einer Reparametrisierung mittels der *Treatment-Kontraste* kommt:

Einfaches Blockexperiment: Die ANOVA-Tabelle

```
> summary( Pen.aov <- aov( Ertrag ~ Sorte + Verfahren, data = Pen.df))
            Df  Sum Sq Mean Sq F value  Pr(>F)
Sorte        4 264.000  66.000  3.5044 0.04075 *
Verfahren    3  70.000  23.333  1.2389 0.33866
Residuals   12 226.000  18.833
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Es wird das additive zweifaktorielle Modell von `Ertrag` an `Sorte` und `Verfahren` (aus `Pen.df`) gefittet und in `Pen.aov` abgelegt. `summary()` liefert dafür die ANOVA-Tabelle: In der Zeile für den Blockbildungsfaktor `Sorte` stehen die Freiheitsgrade (`Df`) $J-1$, die Summe SS_{Block} der Abweichungsquadrate der Blockmittelwerte vom Gesamtmittel (`Sum of Sq`), die mittlere Blockquadratesumme $\mathrm{SS}_{Block}/(J-1)$ (`Mean Sq`) sowie der F-Test (`F Value`) samt p-Wert (`Pr(>F)`) auf Einfluss des Blockbildungsfaktors. In der Zeile für den Behandlungsfaktor `Verfahren` stehen die analogen Größen, insbesondere die Summe $\mathrm{SS}_{Beh.}$ der Abweichungsquadrate der Behandlungsmittelwerte vom Gesamtmittel und der F-Test samt p-Wert auf Einfluss des Behandlungsfaktors. In der Zeile `Residuals` stehen die Residuenquadratesumme RSS mit ihren $(J-1)(L-1)$ Freiheitsgraden und die mittlere Residuenquadratesumme $\mathrm{RSS}/(J-1)(L-1)$.

Zur Interpretation der Ergebnisse siehe §11.2.1.1, wobei hier faktisch nur das Ergebnis für `Verfahren` interessiert.

Zur expliziten Ermittlung von $\hat{\alpha}_j$ und $\hat{\beta}_l$ als Schätzwerte für die Effekte dient natürlich auch hier die Funktion `model.tables()`:

Einfaches Blockexperiment: Die Parameterschätzer

`> model.tables( Pen.aov)`
`Tables of effects`
 	
`Sorte`
`Sorte`
`S1 S2 S3 S4 S5`
`6 -3 -1 2 -4`
	Liefert (voreinstellungsgemäß) die KQS für die Effekte in (113):
	
	$\hat{\alpha}_1, \ldots, \hat{\alpha}_J$

`Verfahren`
`Verfahren`
`A B C D`
`-2 -1 3 0`
	(Identifizierungsbedingung ist, evtl. bis auf Rundungsfehler, erfüllt.)
	$\hat{\beta}_1, \ldots, \hat{\beta}_L$

`> model.tables( Pen.aov, type = "means")`
`Tables of means`
`Grand mean`
`86`
	Wegen `type = "means"` werden die KQS für das cell means-Modell $Y_{jl} = \mu_{jl} + \varepsilon_{jl}$ ausgegeben:
	$\bar{Y}_{..}(= \hat{\mu}_0)$ für das Gesamtmittel $\bar{\mu}_{..}$ $(= \mu_0$; "Grand mean"),

`Sorte`
`Sorte`
`S1 S2 S3 S4 S5`
`92 83 85 88 82`
	$\bar{Y}_{1.}, \ldots, \bar{Y}_{J.}$ für die marginalen (!) Blocklevel-Mittelwerte $\bar{\mu}_{1.}, \ldots, \bar{\mu}_{J.}$ (wobei $\bar{\mu}_{j.} = \mu_0 + \alpha_j$),

`Verfahren`
`Verfahren`
`A B C D`
`84 85 89 86`
	$\bar{Y}_{\cdot 1}, \ldots, \bar{Y}_{\cdot L}$ für die marginalen (!) Faktorlevel-Mittelwerte $\bar{\mu}_{\cdot 1}, \ldots, \bar{\mu}_{\cdot L}$ (wobei $\bar{\mu}_{\cdot l} = \mu_0 + \beta_l$).

Die Modelldiagnose ist identisch zu der in §11.2.2.8:

Einfaches Blockexperiment: Modelldiagnose(-plots)	

`> fitted( Pen.aov)`
`> resid( Pen.aov)`
 	
`> plot( Pen.aov,`
`+ which = 1:2)`
	An die gefitteten Werte (= geschätzte mittlere Responses für jede Levelkombination) und die Residuen kommt man wie stets mit `fitted()` bzw. `resid()`. Die üblichen Residualplots zur Prüfung der Varianzhomogenität und der Normalverteilungsannahme liefert die Plotfunktion für `lm`-Objekte. (Plots nicht gezeigt.)

11.2.5 Hinweise zu unbalancierten Versuchsplänen

(Dieser Abschnitt ist noch unvollständig.)

Auf das Szenario unbalancierter Versuchspläne im zweifaktoriellen Design gehen wir nicht ein, sondern verweisen auf die Literatur wie z. B. [46, Hocking (1996), §13.2]. Im Prinzip funktionieren Theorie und Formalismus im unbalancierten Fall genauso wie im balancierten Fall (vgl. §11.2.2), solange alle $n_{ij} > 0$ sind, außer dass die Formeln komplizierter werden. Schwieriger wird es, wenn einige der Faktorlevelkombinationen gar nicht beobachtet wurden, also einige $n_{ij} = 0$ sind.

Es muss allerdings darauf geachtet werden, dass die Funktion `aov()` laut Online-Hilfe für den balancierten Fall entworfen wurde und ihre Resultate im unbalancierten Fall schwierig zu interpretieren sein könnten ... (worauf sich das auch immer bezieht). Daher erscheint es im unbalancierten Fall angeraten, die o. g. Literatur zu konsultieren, um zu erkennen, worauf sich diese Bemerkung bezieht.

11.3 Einige nicht-parametrische Mehrstichprobentests

Für die Inferenzstatistik der ein- oder mehrfaktoriellen Modelle ist die Normalverteiltheit der Fehler eine zentrale Eigenschaft. Wenn die Normalverteilungsannahme *nicht* gerechtfertigt erscheint, wohl aber von stetiger Verteilung der Daten ausgegangen werden kann, stehen analog zu den Zweistichproben-Szenarien auch hier verschiedene nicht-parametrische (Rang-)Verfahren zur Verfügung.

Bemerkung: Einige – zum Teil sehr – empfehlenswerte Bücher zur nicht-parametrischen Statistik sind auf Deutsch [16, Büning & Trenkler (1994)] und [7, Bortz et al. (2000)] (mit wenig Mathematik, aber zahlreichen, in – etwas viel – Prosa beschriebenen Verfahren und jeweils durchgerechneten Beispielen) sowie auf Englisch [48, Hollander & Wolfe (1973)] (eher anwendungsorientiert) und [44, Hettmansperger (1984)] (eher mathematisch ausgerichtet). In [15, Brunner & Munzel (2013)] wird für unverbundene Stichproben und in [14, Brunner & Langer (1999)] für longitudinale Daten (also für verbundene Stichproben) sogar auf die Forderung stetiger Verteilungen verzichtet; beide Male in hervorragender (deutschsprachiger) Darstellung.

11.3.1 Lokationsprobleme bei unabhängigen Stichproben

Es mögen $L > 2$ unabhängige Stichproben unabhängiger, stetig verteilter Zufallsvariablen X_{li} ($i = 1, \ldots, n_l$, $l = 1, \ldots, L$) vorliegen und zu testen sei, ob ihre L Verteilungsfunktionen $F_1, \ldots, F_L$ gleich sind. Für zweiseitige (= „ungerichtete“ oder „ungeordnete“) Alternativen ist der im folgenden Abschnitt vorgestellte Kruskal-Wallis-Test – als Verallgemeinerung von Wilcoxons zweiseitigem Rangsummentest für den Fall $L = 2$ – das nicht-parametrische Pendant zur einfaktoriellen Varianzanalyse. Für „geordnete“ Alternativen, m. a. W. für Trends in den Lokationen der Verteilungen steht der Jonckheere-Terpstra-Test zur Verfügung, den wir in §11.3.1.2 kurz vorstellen.

11.3.1.1 Der Kruskal-Wallis-Test: `kruskal.test()`

Unter der Hypothese gleicher Verteilungsfunktionen stammen alle $N := \sum_{l=1}^{L} n_l$ Zufallsvariablen X_{li} aus derselben stetigen Verteilung und der N-dimensionale Vektor $(R_{11}, \ldots, R_{1n_1}, \ldots, R_{l1}, \ldots, R_{Ln_L})$ der Ränge R_{li} der *gepoolten* Stichproben ist uniform auf der Menge Σ_N der N-Permutationen verteilt. Daher sollte jede der L Stichproben-Rangsummen $R_{l\cdot} := \sum_{i=1}^{n_l} R_{li}$ einen zu ihrem Stichprobenumfang n_l in etwa proportionalen Anteil an der konstanten Gesamtsumme $\sum_{l=1}^{L} R_{l\cdot} = N(N+1)/2$ haben, also $R_{l\cdot} \approx n_l/N * N(N+1)/2 = n_l(N+1)/2$. Es ist daher intuitiv nachvollziehbar, die (quadratischen) „Abstände“ der Stichproben-Rangsummen von ihren „erwarteten“ Werten zu messen. (Zur etwas formaleren Vorgehensweise siehe unten.)

SP	Zufallsvariablen	Vert.-fkt.		Ränge im SP-**Pool**	Zeilen-Σ
1	$X_{11}, \ldots, X_{1n_1}$	$\overset{\text{u.i.v.}}{\sim} F_1$		$R_{11}, \ldots, R_{1n_1}$	$R_{1\cdot}$
$\vdots$		$\vdots$			$\vdots$
l	$X_{l1}, \ldots \ldots \ldots, X_{ln_l}$	$\overset{\text{u.i.v.}}{\sim} F_l$	$\Rightarrow$	$R_{l1}, \ldots \ldots \ldots, R_{ln_l}$	$R_{l\cdot}$
$\vdots$		$\vdots$			$\vdots$
L	$X_{1L}, \ldots \ldots, X_{Ln_L}$	$\overset{\text{u.i.v.}}{\sim} F_L$		$R_{1L}, \ldots \ldots, R_{Ln_L}$	$R_{L\cdot}$
				Gesamtsumme:	$N(N+1)/2$

<u>Annahmen:</u> Die X_{li} sind für $i = 1, \ldots, n_l$ und $l = 1, \ldots, L$ unabhängig und für $l = 1, \ldots, L$ sind $X_{l1}, \ldots, X_{ln_l}$ u. i. v. $\sim F_l \equiv F(\cdot - \theta_l)$ mit stetiger Verteilungsfunktion F und unbekanntem θ_l.
<u>Zu testen</u> zum Signifikanzniveau α:

$$H_0 : \theta_1 = \ldots = \theta_L \qquad \text{gegen} \qquad H_1 : \theta_l \neq \theta_k \text{ für mindestens ein Paar } l \neq k.$$

Teststatistik:

$$H_N \equiv H_{n_1,\ldots,n_L} := \frac{12}{N(N+1)} \sum_{l=1}^{L} \frac{1}{n_l} \left(R_{l\cdot} - \frac{n_l(N+1)}{2} \right)^2,$$

wobei N und $R_{l\cdot}$ wie eingangs definiert. Die Verteilung der Kruskal-Wallis-Statistik H_N unter H_0 ist für kleine Werte von n_l und L bekannt (siehe z. B. in [48, Hollander & Wolfe (1973)]). Kombinatorische Überlegungen zeigen, dass für jedes l gilt:

$$\mathbb{E}[R_{l\cdot}] = \frac{n_l(N+1)}{2} \qquad \text{und} \qquad \mathrm{Var}(R_{l\cdot}) = \frac{n_l(N+1)(N-n_l)}{12} \qquad \text{unter } H_0$$

Des Weiteren ist jedes $R_{l\cdot}$ asymptotisch normalverteilt:

$$Z_{n_l} := \frac{R_{l\cdot} - \mathbb{E}[R_{l\cdot}]}{\sqrt{\mathrm{Var}(R_{l\cdot})}} \overset{n_l\to\infty}{\longrightarrow} \mathcal{N}(0,1) \quad \text{in Verteilung unter } H_0, \text{ falls } \frac{n_l}{N} \to \lambda_l > 0,$$

woraus folgt: $Z_{n_l}^2 \overset{n_l\to\infty}{\longrightarrow} \chi_1^2$ in Verteilung. Allerdings sind die $R_{l\cdot}$ und damit die $Z_{n_l}^2$ *nicht* unabhängig, denn $\sum_{l=1}^{L} R_{l\cdot} = N(N+1)/2$, weswegen für die (geeignet gewichtete) Summe der $Z_{n_l}^2$ nicht L, sondern $L-1$ Freiheitsgrade für die asymptotische χ^2-Verteilung resultieren:

$$X_N^2 := \sum_{l=1}^{L} \frac{N-n_l}{N} Z_{n_l}^2 \overset{N\to\infty}{\longrightarrow} \chi_{L-1}^2 \text{ in Verteilung unter } H_0, \text{ falls } \frac{n_l}{N} \overset{N\to\infty}{\longrightarrow} \lambda_l > 0 \text{ für } l = 1,\ldots,L$$

Die **R**-Funktion `kruskal.test()` verwendet stets diese χ^2-Approximation. Treten Bindungen zwischen den X_{li} *verschiedener* Stichproben (also verschiedener l-Werte) auf, so wird die Methode der Durchschnittsränge verwendet, was den Erwartungswert von X_N^2 nicht, wohl aber ihre Varianz ändert. (Bindungen *innerhalb* von Stichproben, also für dasselbe l, spielen keine Rolle, da diese die Rangsummen $R_{l\cdot}$ nicht beeinflussen.) In diesem Fall wird eine durch einen Faktor modifizierte Statistik $X_N^{2,mod}$ anstelle von X_N^2 verwendet. $X_N^{2,mod}$ hat dieselbe χ^2-Asymptotik wie X_N^2. (Wir gehen auf diesen Sachverhalt hier nicht ein, sondern verweisen z. B. auf [16, Büning & Trenkler (1994)].)

Entscheidungsregel für konkrete Daten $x_{11},\ldots,x_{Ln_L}$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0 \quad \Longleftrightarrow \quad p\text{-Wert} \le \alpha,$$

wobei für die Berechnung des p-Wertes die χ^2-Approximation zur Anwendung kommt: Ist x_N^2 der realisierte Wert von X_N^2 (bzw. von $X_N^{2,mod}$, wenn Bindungen vorliegen), dann ist

$$p\text{-Wert} = 1 - F_{\chi_{L-1}^2}\left(x_N^2\right)$$

Beispiel anhand des Datensatzes der Koagulationszeiten auf Seite 287 in Abschnitt 11.1: Die Funktion `kruskal.test()` erwartet als erstes Argument den Vektor der X-Werte und als zweites Argument einen genauso langen gruppierenden Vektor – typischerweise ein Faktor – der die Stichprobenzugehörigkeit charakterisiert:

```
> kruskal.test( Koag.df$Zeit, Koag.df$Diaet)

        Kruskal-Wallis rank sum test

data:  Koag.df$Zeit and Koag.df$Diaet
Kruskal-Wallis chi-squared = 17.0154, df = 3, p-value = 0.0007016
```

Die Ausgabe ist vollständig selbsterklärend und die Formelvariante

```
> kruskal.test( Zeit ~ Diaet, data = Koag.df)
```

liefert exakt dasselbe.

11.3.1.2 Der Jonckheere-Terpstra-Test auf Trend (= geordnete Alternativen)

Der Jonckheere-Terpstra-Test ist eine Verallgemeinerung von Wilcoxons Rangsummentest (bzw. des U-Tests von Mann und Whitney) aus §8.5.4. Hier nur knapp ein paar Fakten (für Details siehe z. B. [16, Büning & Trenkler (1994), S. 194 f] oder [48, Hollander & Wolfe (1973), p. 121 ff]):

Annahmen: wie beim Kruskal-Wallis-Test (vgl. §11.3.1.1).
Zu testen zum Signifikanzniveau α:

$$H_0 : \theta_1 = \ldots = \theta_L \quad \text{gegen} \quad H_1 : \theta_1 \leq \ldots \leq \theta_L \text{ mit } \theta_l < \theta_k \text{ für mindestens ein Paar } l \neq k.$$

Beachte: H_1 ist äquivalent zu $F_1 \geq \ldots \geq F_L$ mit mindestens einem „>“.

Teststatistik:

$$J_N \equiv J_{n_1,\ldots,n_L} := \sum_{l<k}^{L} U_{lk} \qquad \text{mit} \qquad U_{lk} = \sum_{i=1}^{n_l} \sum_{j=1}^{n_k} 1_{\{X_{li} < X_{kj}\}},$$

wobei U_{lk} die Wilcoxon-Rangsummen- bzw. Mann-Whitney-Statistik W'_{n_l,n_k} aus §8.5.4 für das Stichprobenpaar l und k ist. Genauer gilt: $U_{lk} = \sum_{j=1}^{n_k} R(X_{kj}) - n_k(n_k+1)/2 = W'_{n_l,n_k} = n_l n_k - W_{n_l,n_k}$, worin $R(X_{kj})$ der Rang von X_{kj} im *gepoolten* Sample des Stichprobenpaares l und k ist. Offenbar stellt J_N eine Verallgemeinerung von Wilcoxons Rangsummenstatistik dar.

Zu J_N's Funktionsweise: U_{lk} nimmt für $l < k$ tendenziell große Werte an, wenn die Werte in Stichprobe l kleiner sind als in Stichprobe k. Und unter der Alternative ist zu erwarten, dass die Werte in Stichprobe 1 tendenziell kleiner sind als die in den Stichproben 2 bis L, die in Stichprobe 2 kleiner als die in den Stichproben 3 bis L usw. Damit wird J_N unter H_1 tendenziell groß.

Die Verteilung der Jonckheere-Terpstra-Statistik J_N unter H_0 ist für kleine Werte von L und $n_1, \ldots, n_L$ bekannt (siehe z. B. in [48, Hollander & Wolfe (1973)]). Kombinatorische Überlegungen zeigen, dass mit $N = \sum_{l=1}^{L} n_l$ gilt:

$$\mathbb{E}[J_N] = \frac{N^2 - \sum_{l=1}^{L} n_l}{4} \quad \text{und} \quad \mathrm{Var}(J_N) = \frac{N^2(2N+3) - \sum_{l=1}^{L} n_l^2(2n_l+3)}{72} \qquad \text{unter } H_0$$

Des Weiteren folgt:

$$Z_N := \frac{J_N - \mathbb{E}[J_N]}{\sqrt{\mathrm{Var}(J_N)}} \overset{N\to\infty}{\longrightarrow} \mathcal{N}(0,1) \text{ in Verteilung unter } H_0, \text{ falls } \frac{n_l}{N} \overset{N\to\infty}{\longrightarrow} \lambda_l > 0 \text{ für } l = 1, \ldots, L$$

Entscheidungsregel für konkrete Daten $x_{11}, \ldots, x_{Ln_L}$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0 \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei für die Berechnung des p-Wertes die Normalverteilungsaproximation zur Anwendung kommt: Ist z_N der realisierte Wert von Z_N (bzw. von Z_N^{mod} mit $1_{\{X_{li} < X_{kj}\}} + \frac{1}{2} \cdot 1_{\{X_{li} = X_{kj}\}}$ anstelle von $1_{\{X_{li} < X_{kj}\}}$ in J_N, wenn Bindungen vorliegen), dann ist

$$p\text{-Wert} = 1 - \Phi(z_N)$$

Bemerkungen: Mir ist bisher keine in einem Paket oder etwa in der Base-Distribution von **R** implementierte Funktion für den Jonckheere-Terpstra-Test bekannt. Wohl aber ist eine Implementation unter `http://tolstoy.newcastle.edu.au/R/help/06/06/30112.html` zu finden. Im relativ neuen Paket `clinfun` scheint eine Permutationsvariante des Tests implementiert zu sein und in der E-Mail `http://finzi.psych.upenn.edu/Rhelp10/2010-May/237833.html` wird eine weitere solche Implementation beschrieben. In all diesen Funktionen ist vor ihrer Verwendung jedoch bestimmt etwas „Code-Recherche“ ratsam.

11.3.2 Lokationsprobleme bei verbundenen Stichproben

Das einfache Blockexperiment, welches ein geeignet interpretiertes zweifaktorielles Modell ist und den Fall von $L > 2$ verbundenen Stichproben dadurch widerspiegelt, dass die Verbindungen innerhalb der Stichproben durch die Blockbildung modelliert werden, kann man nichtparametrisch mit Hilfe des im folgenden Abschnitt vorgestellten Friedman-Tests untersuchen; er testet auf zweiseitige (= „ungerichtete“ oder „ungeordnete“) Alternativen. Für geordnete Alternativen, also für Trends in den Lokationen der Verteilungen der verbundenen Stichproben stellen wir in §11.3.2.2 kurz den Page-Test vor.

Zur Beschreibung des Szenarios des einfachen Blockexperiments erinnern wir an die Ausführungen zu Beginn von §11.2.4: Wir wollen $L \geq 3$ verschiedene Behandlungen untersuchen, deren Effekte durch eine Störgröße mit $J \geq 2$ Ausprägungen maskiert werden könnten, und haben insgesamt $J \cdot L$ Untersuchungseinheiten so zur Verfügung, dass in jeder Behandlungsgruppe zu jeder Störgrößenausprägung genau eine Untersuchungseinheit existiert. Die Zufallsvariable X_{jl} ist dann die Beobachtung in Behandlung l im (bezüglich der Störgröße homogenen) Block j und habe die stetige Verteilungsfunktion F_{jl}. Es soll getestet werden, ob $F_{j1} = \ldots = F_{jL}$ für jedes $j = 1, \ldots, J$ gilt.

11.3.2.1 Der Friedman-Test: `friedman.test()`

Unter der Hypothese $F_{j1} = \ldots = F_{jL} (=: F_j)$ für jedes $j = 1, \ldots, J$ stammen die L unabhängigen Zufallsvariablen $X_{j1}, \ldots, X_{jL}$ aus derselben stetigen Verteilung F_j und der L-dimensionale Vektor $(R_{j1}, \ldots, R_{jL})$ der Ränge R_{jl} im Block j ist uniform auf der Menge Σ_L der L-Permutationen verteilt. Jede Blockrangsumme $R_{j\cdot}$ ist außerdem stets gleich $L(L+1)/2$ und die Summe $R_{\cdot\cdot}$ über alle Ränge der J Stichproben daher gleich $JL(L+1)/2$. Daher sollten unter der genannten Hypothese alle L Behandlungsrangsummen $R_{\cdot l} := \sum_{j=1}^{J} R_{jl}$ etwa denselben Wert haben und dieser in etwa gleich $R_{\cdot\cdot}/L = J(L+1)/2$ sein. Es ist daher sinnvoll, die (quadratischen) „Abstände“ der Behandlungsrangsummen von ihren „erwarteten“ Werten zu messen. (Die formalere Vorgehensweise folgt unten.)

	Zufallsvariablen			
	Behandlung			
Block	1	2	…	L
1	X_{11}	X_{12}	…	X_{1L}
⋮	⋮	⋮	⋮	⋮
j	X_{j1}	X_{j2}	…	X_{jL}
⋮	⋮	⋮	⋮	⋮
J	X_{J1}	X_{J2}	…	X_{JL}

$\Rightarrow$

Ränge **pro Block**				
Behandlung				Block-
1	2	…	L	Rang-Σ
R_{11}	R_{12}	…	R_{1L}	$L(L+1)/2$
⋮	⋮	⋮	⋮	⋮
R_{j1}	R_{j2}	…	R_{jL}	$L(L+1)/2$
⋮	⋮	⋮	⋮	⋮
R_{J1}	R_{J2}	…	R_{JL}	$L(L+1)/2$
$R_{\cdot 1}$	$R_{\cdot 2}$	…	$R_{\cdot L}$	$JL(L+1)/2$
Behandlungsrang-Σ				

<u>Annahmen:</u> Die X_{jl} sind für $j = 1, \ldots, J$ und $l = 1, \ldots, L$ unabhängig und $X_{jl} \sim F_{jl} \equiv F(\cdot - \alpha_j - \theta_l)$ mit stetigem F sowie unbekannten α_j und θ_l.
<u>Zu testen</u> zum Signifikanzniveau α:

$$H_0 : \theta_1 = \ldots = \theta_L \qquad \text{gegen} \qquad H_1 : \theta_l \neq \theta_k \text{ für mindestens ein Paar } l \neq k.$$

<u>Teststatistik:</u>

$$F_J := \sum_{l=1}^{L} \left(R_{\cdot l} - \frac{J(L+1)}{2} \right)^2,$$

wobei $R_{\cdot l}$ wie eingangs definiert. Die Verteilung der <u>Friedman-Statistik</u> F_J unter H_0 ist für kleine Werte von J und L bekannt (siehe z. B. in [48, Hollander & Wolfe (1973)] oder in [16, Büning

& Trenkler (1994)]). Kombinatorische Überlegungen ergeben, dass für jedes j und l gilt:

$$\mathbb{E}[R_{jl}] = \frac{L+1}{2}, \qquad \mathrm{Var}(R_{jl}) = \frac{L^2-1}{12} \qquad \text{und} \qquad \mathrm{Cov}(R_{jl}, R_{jk}) = -\frac{L+1}{12} \text{ für } l \neq k$$

sowie

$$\mathbb{E}[F_J] = L-1 \qquad \text{und} \qquad \mathrm{Var}(F_J) = \frac{2(L-1)(J-1)}{J} \qquad \text{unter } H_0$$

Des Weiteren ist unter H_0 jedes $R_{\cdot l}$ asymptotisch normalverteilt:

$$Z_l := \frac{R_{\cdot l} - \mathbb{E}[R_{\cdot l}]}{\sqrt{\mathrm{Var}(R_{\cdot l})}} \overset{J\to\infty}{\longrightarrow} \mathcal{N}(0,1) \quad \text{in Verteilung unter } H_0,$$

wobei

$$\mathbb{E}[R_{\cdot l}] = \frac{J(L+1)}{2} \qquad \text{und} \qquad \mathrm{Var}(R_{\cdot l}) = \frac{J(L^2-1)}{12}$$

Daraus folgt: $Z_l^2 \overset{J\to\infty}{\longrightarrow} \chi_1^2$ in Verteilung. Allerdings sind die $R_{\cdot l}$ und damit die Z_l^2 *nicht* unabhängig, denn $\sum_{l=1}^{L} R_{\cdot l} = JL(L+1)/2$, weswegen für die (geeignet gewichtete) Summe der Z_l^2 nicht L, sondern $L-1$ Freiheitsgrade für die asymptotische χ^2-Verteilung resultieren:

$$X_J^2 = \sum_{l=1}^{L} \frac{L-1}{L} Z_l^2 \overset{J\to\infty}{\longrightarrow} \chi_{L-1}^2 \quad \text{in Verteilung unter } H_0$$

Die **R**-Funktion `friedman.test()` verwendet stets diese χ^2-Approximation. Treten Bindungen zwischen den X_{jl} *innerhalb* eines Blocks (also für verschiedene l-Werte bei gleichem j) auf, so wird die Methode der Durchschnittsränge verwendet, was den Erwartungswert von X_J^2 nicht, wohl aber ihre Varianz beeinflusst. In diesem Fall wird eine durch einen Faktor modifizierte Statistik $X_J^{2,mod}$ anstelle von X_J^2 verwendet. $X_J^{2,mod}$ hat dieselbe χ^2-Asymptotik wie X_J^2. (Wir gehen auf diesen Sachverhalt hier nicht ein, sondern verweisen z. B. auf [16, Büning & Trenkler (1994)].)

Entscheidungsregel für konkrete Daten $x_{11}, \ldots, x_{JL}$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0 \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei für die Berechnung des p-Wertes die χ^2-Approximation zur Anwendung kommt: Ist x_J^2 der realisierte Wert von X_J^2 (bzw. von $X_J^{2,mod}$, wenn Bindungen vorliegen), dann ist

$$p\text{-Wert} = 1 - F_{\chi_{L-1}^2}\left(x_J^2\right)$$

Beispiel anhand des Penicillin-Datensatzes (siehe Seite 315 in §11.2.4): Auch die Funktion `friedman.test()` erwartet als erstes Argument den Vektor der X-Werte und als zweites Argument einen Faktorvektor, der die Behandlung charakterisiert, aber als drittes Argument noch einen Faktor, der die Blockzugehörigkeit angibt:

```
> with( Pen.df, friedman.test( Ertrag, Verfahren, Sorte))

        Friedman rank sum test

data:  Ertrag and Verfahren and Sorte
Friedman chi-square = 3.4898, df = 3, p-value = 0.3221
```

Auch diese Ausgabe ist vollständig selbsterklärend und die Formelvariante lautet

```
> friedman.test( Ertrag ~ Verfahren | Sorte, data = Pen.df)
```

Bemerkungen: Für die Anwendung des Friedman-Tests kann die Voraussetzung der Unabhängigkeit der X_{jl} abgeschwächt werden zur Bedingung der "compound symmetry" an ihre Kovarianzstruktur. Deren Vorliegen ist aber insbesondere für Longitudinalbeobachtungen im Allgemeinen schwer zu rechtfertigen. Für Details verweisen wir auf §7.2.1 in [14, Brunner & Langer (1999)]. In [7, Bortz et al. (2000)] wird ein weiterer Test für abhängige Stichproben beschrieben, der in gewissen Situationen leistungsfähiger als der Friedman-Test sein soll (hier aber nicht näher betrachtet wird): Der (Spannweitenrang-)Test von Quade. Er ist in **R** in der Funktion `quade.test()` implementiert.

11.3.2.2 Der Test von Page auf Trend (= geordnete Alternativen)

Auch hier nur ganz knapp ein paar Fakten (Details stehen wieder z. B. in [16, Büning & Trenkler (1994), S. 212 f] oder in [48, Hollander & Wolfe (1973), p. 147 ff]):

Annahmen: wie beim Friedman-Test (vgl. §11.3.2.1).
Zu testen zum Signifikanzniveau α:

$$H_0 : \theta_1 = \ldots = \theta_L \quad \text{gegen} \quad H_1 : \theta_1 \leq \ldots \leq \theta_L \text{ mit } \theta_l < \theta_k \text{ für mindestens ein Paar } l \neq k.$$

Beachte: H_1 ist äquivalent zu $F(\cdot - \alpha_j - \theta_1) \geq \ldots \geq F(\cdot - \alpha_j - \theta_L)$ für $j = 1, \ldots, J$ und mit mindestens einem „$>$".

Teststatistik:

$$P_J := \sum_{l=1}^{L} l R_{\cdot l},$$

wobei $R_{\cdot l} := \sum_{j=1}^{J} R_{jl}$ ist sowie R_{jl} der Rang von X_{jl} in Block j, also unter $X_{j1}, \ldots, X_{jL}$ ist.

Zu P_J's Funktionsweise: Unter der Alternative ist zu erwarten, dass für $l_1 < l_2$ die Ränge $R_{1l_1}, \ldots, R_{Jl_1}$ tendenziell kleinere Werte annehmen als die Ränge $R_{1l_2}, \ldots, R_{Jl_2}$, sodass auch $R_{\cdot l_1}$ tendenziell kleiner als $R_{\cdot l_2}$ erwartet wird (im Gegensatz zu H_0, worunter deren Erwartungswerte gleich sind). Durch die Gewichtung von $R_{\cdot l_1}$ mit dem kleineren l_1 und von $R_{\cdot l_2}$ mit dem größeren l_2, wird P_J unter H_1 somit tendenziell groß.

Die Verteilung der Page-Statistik P_J unter H_0 ist für kleine Werte von J und L bekannt (siehe z. B. [48, Hollander & Wolfe (1973)]). Kombinatorische Überlegungen liefern:

$$\mathbb{E}[P_J] = \frac{JL(L+1)^2}{4} \quad \text{und} \quad \mathrm{Var}(P_J) = \frac{JL^2(L+1)^2(L-1)}{144} \qquad \text{unter } H_0$$

Des Weiteren folgt:

$$Z_J := \frac{P_J - \mathbb{E}[P_J]}{\sqrt{\mathrm{Var}(P_J)}} \overset{J\to\infty}{\longrightarrow} \mathcal{N}(0,1) \quad \text{in Verteilung unter } H_0$$

Entscheidungsregel für konkrete Daten $x_{11}, \ldots, x_{JL}$ auf Basis des p-Wertes:

$$\text{Verwirf } H_0 \quad \Longleftrightarrow \quad p\text{-Wert} \leq \alpha,$$

wobei für die Berechnung des p-Wertes die Normalverteilungsaproximation zur Anwendung kommt: Ist z_J der realisierte Wert von Z_J, dann ist

$$p\text{-Wert} = 1 - \Phi(z_J)$$

Bemerkungen: Mir ist bisher keine in einem Paket oder etwa in der Base-Distribution von **R** direkte Implementation des Page-Tests bekannt. Allerdings scheint im Paket `coin` eine Permutationsvariante des Page-Tests durch die Funktion `friedman_test()` (!) zur Verfügung zu stehen, wenn auf der rechten Seite in deren Modellformel ein geordneter Faktor steht. Darüberhinaus ist der von `friedman_test()` durchgeführte Page-Test ein *zwei*seitiger Test, also ein Test auf

irgendeinen Trend. Vor ihrer Verwendung ist also weitere Recherche z. B. in der Online-Hilfe der genannten Funktion ratsam.

Beachtenswert in diesem Zusammenhang ist, dass eine direkte Beziehung zwischen der Page-Teststatistik und dem Spearmanschen Rangkorrelationskoeffizienten besteht, die in [48, Hollander & Wolfe (1973)] auf Seite 149 in "Comment 13" beschrieben wird.

11.3.3 Hinweise zu Skalenproblemen

Es existieren auch nicht-parametrische Mehrstichprobentests für Skalenprobleme: Für zwei unabhängige Stichproben stehen **R**-Implementationen des Ansari- und des Mood-Tests auf Skalenunterschiede (auch Varianzinhomogenität oder Variabilitätsalternativen genannt) in den Funktionen `ansari.test()` bzw. `mood.test()` zur Verfügung sowie in `fligner.test()` der Fligner-Killeen-Test auf Varianzinhomogenität in mehreren unabhängigen Stichproben. Für Details verweisen wir auf die Online-Hilfe der jeweiligen Funktionen und auf die dort zitierte Literatur.

Für abhängige Stichproben sind z. B. in [7, Bortz et al. (2000)] der Siegel-Tukey- und der Meyer-Bahlburg-Test beschrieben; in **R** scheinen sie wohl noch nicht implementiert zu sein.

11.3.4 Hinweis zur nicht-parametrischen Analyse in mehrfaktoriellen Versuchsplänen

Für die nicht-parametrischen Analyse in zwei- oder mehrfaktoriellen Designs verweisen wir nachdrücklich auf und empfehlen wärmstens [15, Brunner & Munzel (2013)].

11.4 Multiple Mittelwertvergleiche

Achtung: Dieser Abschnitt befindet sich noch in einem einigermaßen provisorischen Zustand!

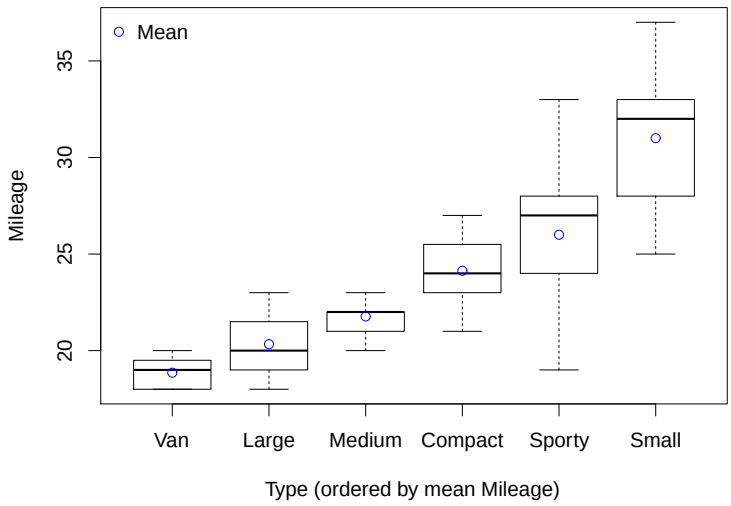
Häufig ist man im Zusammenhang mit einer ANOVA nicht nur daran interessiert, herauszufinden, *ob* die Levels des betrachteten Faktors einen signifikanten Einfluss auf die mittlere Response haben, sondern auch vielmehr *welche* Levels sich signifikant voneinander unterscheiden, und im nächsten Schritt, wie stark (quantifiziert durch Konfidenzintervalle). Dies läuft auf *multiple* Mittelwertvergleiche (Engl.: "multiple comparisons", kurz: MC) hinaus, wofür zahlreiche Verfahren entwickelt wurden, alldieweil es sehr unterschiedliche Möglichkeiten gibt, solche Vergleiche durchzuführen.

Wir beschränken uns hier auf die Vorstellung einiger Standardverfahren für die *ein*faktorielle ANOVA, die in **R** z. B. in den Funktionen des Paketes `multcomp` realisiert sind. (Diese Funktionen sind allerdings im Stande, erheblich mehr zu leisten, als das, worauf wir hier eingehen. [11, Bretz et al. (2010)] liefern weitergehende Informationen.) Sie produzieren nicht nur die Signifikanzergebnisse, sprich die p-Werte, die in einer „Familie" von multiplen Vergleichen im Einzelnen erzielt wurden, sondern überdies *simultane* Konfidenzintervalle für die betrachteten Mittelwertedifferenzen (bzw. allgemeiner für die den Vergleichen zugrundeliegenden linearen Kontraste) samt entsprechenden Grafiken. Diese Konfidenzintervalle liefern erheblich mehr interpretierbare Informationen, als die alleinige Dokumentation von „Signifikant"- oder „Nicht signifikant"-Aussagen für einzelne Vergleiche.

Wir verzichten hier auf jegliche Darstellung des theoretischen Hintergrunds, sondern verweisen auf die Literatur. Zum Beispiel bieten [11, Bretz et al. (2010)] (eine Mischung aus etwas Theorie und **R**-spezifischem Anwendungsbezug), [45, Hochberg & Tamhane (1987)], [50, Hsu (1996)] (beide mathematisch ausgerichtet) und [49, Horn & Vollandt (1995)] (deutschsprachig und rein anwendungsbezogen) gute Einführungen in und umfangreiche Überblicke über das Thema. Wir belassen es hier bei Anwendungsbeispielen.

Der Data Frame `car.test.frame` des Paketes `rpart` enthält in seiner Komponente `Mileage` die Fahrleistung zahlreicher Automobile in Meilen pro Gallone. Die Faktorkomponente `Type` kategorisiert die Autos in verschiedene Fahrzeugtypen. Nach `Type` gruppierte Boxplots der `Mileage` deuten auf Unterschiede in den mittleren Fahrleistungen zwischen einigen der Faktorlevels hin:

```
> data( car.test.frame, package = "rpart")
> Type.by.MeanMileage <- with( car.test.frame, reorder( Type, Mileage, mean))
> plot( Mileage ~ Type.by.MeanMileage, data = car.test.frame,
+       xlab = "Type (ordered by mean Mileage)")
> points( 1:nlevels( Type.by.MeanMileage), col = "blue",
+         tapply( car.test.frame$Mileage, Type.by.MeanMileage, mean))
> legend( "topleft", legend = "Mean", pch = 1, col = "blue", bty = "n")
```



Eine einfaktorielle ANOVA dient der Klärung:

```
> summary( Mileage.aov <- aov( Mileage ~ Type.by.MeanMileage,
+                              data = car.test.frame))
                    Df Sum Sq Mean Sq F value    Pr(>F)
Type.by.MeanMileage  5 943.02  188.60  24.746 7.213e-13 ***
Residuals           54 411.56    7.62
....
```

Offenbar liegt ein signifikanter Einfluss von `Type` auf die mittlere `Mileage` vor. Doch welche *paarweisen* Unterschiede zwischen den Fahrzeugtypen sind dafür verantwortlich bzw. sind signifikant? Und wie groß sind diese Unterschiede? Solche Fragen erfordern die Analyse einer „Familie“ paarweiser Differenzen. Zur ersten Orientierung hier das der ANOVA zugrundeliegende lineare Modell:

```
> summary( lm( Mileage ~ Type.by.MeanMileage, data = car.test.frame))
....
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                 18.857      1.043  18.072  < 2e-16 ***
Type.by.MeanMileageLarge     1.476      1.905   0.775 0.441795
Type.by.MeanMileageMedium    2.912      1.294   2.250 0.028544 *
Type.by.MeanMileageCompact   5.276      1.264   4.175 0.000109 ***
Type.by.MeanMileageSporty    7.143      1.391   5.134 3.98e-06 ***
Type.by.MeanMileageSmall    12.143      1.294   9.382 6.22e-13 ***
....
```

Die marginalen p-Werte lassen in den Einzelvergleichen des (ersten und damit Bezugs-)Levels `Van` mit den fünf anderen vier der Mittelwertedifferenzen auf dem (lokalen, sprich auf den einzelnen Test bezogenen) Niveau 5 % signifikant verschieden von Null erscheinen. Aber – Vorsicht! – dies geschieht eben ohne Korrektur der Multiplizität, die auch schon bei einer Familie von „nur“ fünf Tests das sogenannte globale Niveau (also die Wahrscheinlichkeit für *mindestens* einen Fehler 1. Art in der betrachteten Familie) auf gut 22.6 % und damit deutlich über die lokal gültigen 5 % hinaus befördert!

Das Paket `multcomp` (siehe auch [11, Bretz et al. (2010)]) stellt leistungsfähige Funktionen für die adäquate Durchführung multipler Tests zur Verfügung. Wesentlich darunter ist `glht()` (von “general linear hypothesis”):

```
glht( model, linfct, alternative = c( "two.sided", "less", "greater"),
      rhs = 0, ...)
```

Zur Bedeutung der Argumente:

- `model`: Ein gefittetes parametrisches Modell wie z. B. ein `aov`-, `lm`- oder `glm`-Objekt.
- `linfct`: Erhält die Spezifikation der linken Seite der zu testenden linearen Globalhypothese $H_0 : \mathbf{C}\boldsymbol{\mu} = \mathbf{c}_0$, sprich die Spezifikation der zu testenden Hypothesenfamilie, und zwar entweder als Kontrastematrix $\mathbf{C}$ oder als eine symbolische Beschreibung einer oder mehrerer linearer Hypothesen (entsprechen den Zeilen von $\mathbf{C}$) oder als Ergebnis der Funktion `mcp()`, die multiple Vergleiche in AN(C)OVA-Modellen spezifiert (indem sie geeignete $\mathbf{C}$s konstruiert).
- `alternative`: Legt fest, ob zwei- oder einseitige (obere oder untere) Alternativen betrachtet werden.
- `rhs`: Rechte Seite $\mathbf{c}_0$ in der linearen Globalhypothese $H_0 : \mathbf{C}\boldsymbol{\mu} = \mathbf{c}_0$. Voreinstellung ist $\mathbf{0}$.
- `...`: Weitere Argumente.

11.4.1 Multiple Vergleiche mit einer Kontrolle

Unter multiplen Vergleichen mit einer Kontrolle (Engl.: "multiple comparisons with a control", kurz: MCC) wird typischerweise der Vergleich verschiedener Gruppen (Treatments) mit einer ausgewählten (Bezugs-)Gruppe wie z. B. einer „Standard-Behandlung" oder einer „Kontrolle" verstanden. Die linearen Kontraste, die diese für einen Faktor mit k Levels $k-1$ paarweisen Vergleiche beschreiben, werden auch Dunnett-Kontraste genannt.

11.4.1.1 Zweiseitige Vergleiche und Konfidenzintervalle

Wir bedienen uns des Fahrleistungsdatensatzes und werden darin für den (willkürlich ausgewählten) Faktor `Type.by.MeanMileage` alle paarweisen Vergleiche mit einer Kontrolle durchführen. Der Funktion `glht()` teilen wir dies durch `linfct = mcp( Type.by.MeanMileage= "Dunnett")` mit, das dafür sorgt, dass als Hypothesenmatrix $\mathbf{C}$ die Dunnett-Kontraste der zu den Levels von `Type.by.MeanMileage` gehörenden `Mileage`-Mittelwerten gebildet werden. Per Voreinstellung wird der erste Level des Faktors – hier `Van` gemäß der Levelsortierung von `Type.by.MeanMileage` – automatisch zur Kontrolle (was durch eine andere Levelsortierung geändert werden kann):

```
> (Mileage.2smcc <- glht( Mileage.aov,
+                         linfct = mcp( Type.by.MeanMileage = "Dunnett")))

        General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts

Linear Hypotheses:
                   Estimate
Large - Van == 0      1.476
Medium - Van == 0     2.912
Compact - Van == 0    5.276
Sporty - Van == 0     7.143
Small - Van == 0     12.143
```

Das `glht`-Objekt `Mileage.2smcc` enthält jetzt alles zur Durchführung eines Tests Notwendige, wie z. B. in seiner Komponente `Mileage.2smcc$linfct` die Kontrastematrix, die aus der in **R** voreingestellten Treatment-Kontraste-Parametrisierung des gefitteten linearen Modells die für die Hypothese interessierenden (also durch $\mathbf{C}$ definierten) Differenzen bildet. Den eigentlichen zweiseitigen Dunnett-Test, d. h. den Test von $H_0 : \mathbf{C}\boldsymbol{\mu} = \mathbf{0}$ mit der Dunnett-Kontrastematrix $\mathbf{C}$ gegen $H_0 : \mathbf{C}\boldsymbol{\mu} \neq \mathbf{0}$ unter Verwendung der sogenannten "single-step"-Methode (auf der Basis eines sogenannten max-t-Tests und parametrischer multivariater t-Verteilungseigenschaften) führt `summary()` angewendet auf das `glht`-Objekt aus:

```
> summary( Mileage.2smcc)

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts

Fit: aov(formula = Mileage ~ Type.by.MeanMileage, data = car.test.frame)

Linear Hypotheses:
                   Estimate Std. Error t value p value
Large - Van == 0      1.476      1.905   0.775   0.884
Medium - Van == 0     2.912      1.294   2.250   0.103
```

```
Compact - Van == 0    5.276      1.264   4.175  <0.001 ***
Sporty - Van == 0     7.143      1.391   5.134  <0.001 ***
Small - Van == 0     12.143      1.294   9.382  <0.001 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

Laut letzter Zeile der Ausgabe sind die angegebenen p-Werte adjustiert, und zwar gemäß der "single-step"-Methode (für den Dunnett-Test der Dunnett-Kontraste $\mathbf{C}\boldsymbol{\mu}$).

Folgerung: Auf einem Signifikanzniveau von (sogar!) 0.1 % gilt die folgende „Global"-Aussage über die (hier fünf) betrachteten Vergleiche der Levels `Large`, `Medium`, `Compact`, `Sporty` und `Small` mit der Kontrollgruppe `Van`: „Es besteht hinsichtlich der `Mileage`-Mittelwerte weder ein signifikanter Unterschied zwischen `Large` und `Van` noch zwischen `Medium` und `Van`, aber es bestehen signifikante Unterschiede zwischen den drei anderen (`Compact`, `Sporty`, `Small`) und `Van`."

Zweiseitige simultane Konfidenzintervalle erhält man durch `confint()` angewendet auf das obige `glht`-Objekt:

```
> (MileageSCI.2smcc <- confint( Mileage.2smcc, level = 0.95))

         Simultaneous Confidence Intervals

Multiple Comparisons of Means: Dunnett Contrasts

Fit: aov(formula = Mileage ~ Type.by.MeanMileage, data = car.test.frame)

Estimated Quantile = 2.5653
95% family-wise confidence level
Linear Hypotheses:
                   Estimate lwr     upr
Large - Van == 0    1.4762  -3.4109  6.3632
Medium - Van == 0   2.9121  -0.4080  6.2322
Compact - Van == 0  5.2762   2.0345  8.5179
Sporty - Van == 0   7.1429   3.5739 10.7119
Small - Van == 0   12.1429   8.8228 15.4630
```

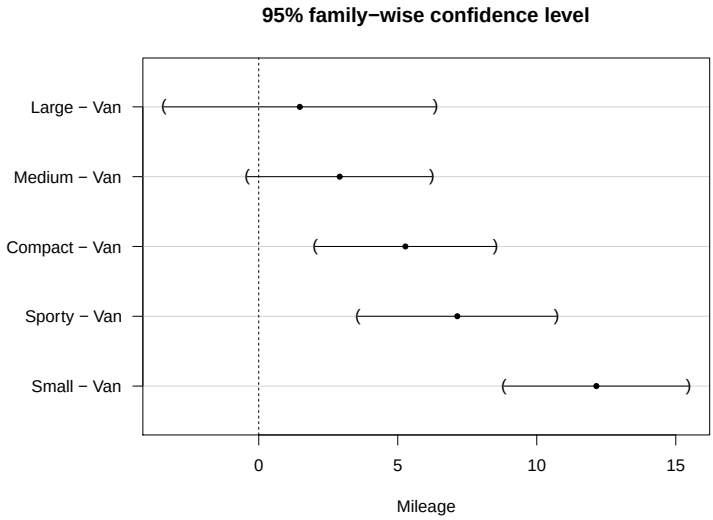
Die Ausgabe dokumentiert, dass simultane 95 %-Konfidenzintervalle für die Dunnett-Kontraste, d. h. für die Differenzen zwischen dem `Mileage`-Mittelwert des `Type`-Levels `Van` und den anderen Levels berechnet wurden, und zwar – ohne dass es hier explizit gesagt wird – mittels der Invertierung der "single-step"-Methode für den Dunnett-Test. (Das `Estimated quantile` ist das für die Konfidenzintervalle der Differenzen verwendete geschätzte (!) 95 %-Quantil der Verteilung der betraglichen Maximumskomponente einer multivariaten (hier 5-dimensionalen) t-Verteilung, deren Korrelationsstruktur aus der des multivariat normalverteilten Parameterschätzers $\hat{\boldsymbol{\mu}}$ abgeleitet und geschätzt wurde.) Die Konfidenzintervalle, die nicht die Null enthalten, identifizieren die dazugehörigen Mittelwerte als signifikant voneinander verschieden.

Folgerung: Auf einem Konfidenzniveau von 95 % gilt die folgende „Global"-Aussage über die (hier fünf) betrachteten Vergleiche: „Es besteht bzgl. der `Mileage`-Mittelwerte weder ein signifikanter Unterschied zwischen `Large` und `Van` noch zwischen `Medium` und `Van`, aber es bestehen signifikante Unterschiede zwischen den drei anderen (`Compact`, `Sporty`, `Small`) und `Van`, deren Größenordnungen – gemeint sind natürlich die Differenzen zwischen den levelspezifischen

`Mileage`-Mittelwerten – simultan durch die Grenzen der jeweiligen Konfidenzintervalle zum Niveau 95 % nach oben und nach unten abgeschätzt werden können.“

`plot()` angewendet auf das Ergebnis von `confint()` produziert die folgende grafische Darstellung dieser zweiseitigen simultanen Konfidenzintervalle:

```
> plot( MileageSCI.2smcc, xlab = "Mileage")
```



11.4.1.2 Einseitige Vergleiche und Konfidenzschranken

Häufig ist man an einseitigen Hypothesentests interessiert und es ist auch nicht notwendig, beidseitig Konfidenzgrenzen, also Konfidenz*intervalle* anzugeben, sondern einseitige (obere oder untere) Konfidenz*schranken* reichen aus. Dies ist z. B. der Fall, wenn die Überlegenheit (oder Unterlegenheit) verschiedener Treatmens im Vergleich mit einem/r Standard/Placebo/Kontrolle untersucht werden soll. Damit sind einseitige multiple Vergleiche und ihre simultanen Konfidenzschranken gefragt.

Anhand des Beispiels des vorherigen Abschnitts wollen wir demonstrieren, wie die Funktion `glht()` das Gewünschte liefert: Der einzige Unterschied ist die Verwendung des Arguments `alternative = "greater"` in `glht()`. Das führt durch `summary()` schließlich zur Durchführung des einseitigen Dunnett-Tests (mittels der “single-step”-Methode) und zur Berechnung – hier – *unterer* Schranken für die Dunnett-Kontraste, d. h. für die Differenzen der `Mileage`-Mittelwerte der Levels `Large`, `Medium`, `Compact`, `Sporty` und `Small` zu demjenigen im Level `Van`:

```
> Mileage.1smcc <- glht( Mileage.aov, alternative = "greater",
+                        linfct = mcp( Type.by.MeanMileage = "Dunnett"))
> summary( Mileage.1smcc)

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts
....
                   Estimate Std. Error t value p value
Large - Van <= 0      1.476      1.905   0.775  0.4981
Medium - Van <= 0     2.912      1.294   2.250  0.0514 .
Compact - Van <= 0    5.276      1.264   4.175  <0.001 ***
Sporty - Van <= 0     7.143      1.391   5.134  <0.001 ***
```

```
Small - Van <= 0     12.143      1.294   9.382  <0.001 ***
....
(Adjusted p values reported -- single-step method)
```

(Beachte, dass die p-Werte einseitiger Tests – wie immer – kleiner sind als diejenigen der korrespondierenden zweiseitigen Tests!)

Folgerung: Auf einem Signifikanzniveau von (sogar!) 0.1 % gilt die folgende „Global"-Aussage über die (hier fünf) betrachteten Vergleiche: „Die Levels `Large` und `Medium` haben keine signifikant größere mittlere `Mileage` als `Van`, aber die Levels `Compact`, `Sporty` und `Small` haben signifikant höhere mittlere `Mileage`-Werte als `Van`."

Die dazugehörigen einseitigen simultane Konfidenzintervalle, sprich -schranken (ermittelt durch Invertierung der "single-step"-Methode für den einseitigen Dunnett-Test):

```
> (MileageSCI.1smcc <- confint( Mileage.1smcc, level = 0.95))

         Simultaneous Confidence Intervals

Multiple Comparisons of Means: Dunnett Contrasts

Fit: aov(formula = Mileage ~ Type.by.MeanMileage, data = car.test.frame)

Estimated Quantile = 2.261
95% family-wise confidence level
Linear Hypotheses:
                   Estimate lwr      upr
Large - Van <= 0    1.47619 -2.83111      Inf
Medium - Van <= 0   2.91209 -0.01415      Inf
Compact - Van <= 0  5.27619  2.41905      Inf
Sporty - Van <= 0   7.14286  3.99725      Inf
Small - Van <= 0   12.14286  9.21662      Inf
```

(Beachte, dass die unteren Konfidenzschranken größer sind als die Untergrenzen der zweiseitigen Konfidenzintervalle zum selben Niveau (wie auch obere Konfidenzschranken kleiner wären als die Obergrenzen zweiseitiger Konfidenzintervalle).)

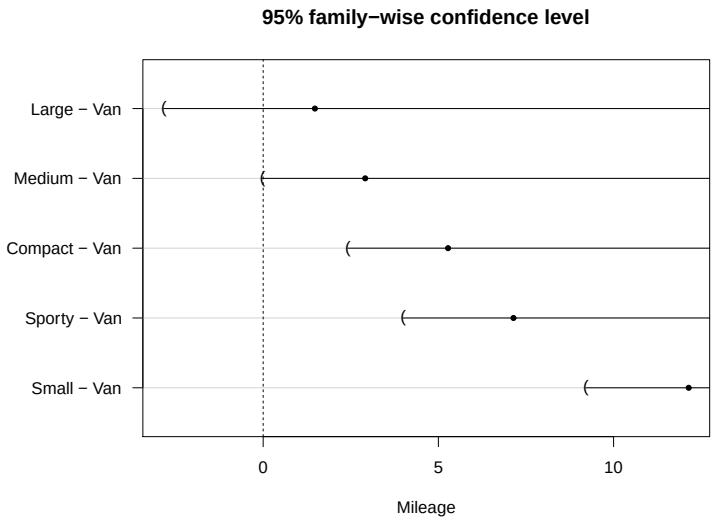
Ist die Null außerhalb eines Konfidenzbereichs, so ist – hier – seine untere Schranke größer als Null und somit seine zugehörige Mittelwertedifferenz signifikant größer als Null. Dies bedeutet, dass der `Mileage`-Mittelwert des entsprechenden Levels signifikant größer als der `Mileage`-Mittelwert in `Van` ist.

Folgerung: Auf einem Konfidenzniveau von 95 % gilt die folgende „Global"-Aussage über die (hier fünf) betrachteten Vergleiche: „Die Levels `Large` und `Medium` haben keine signifikant größere mittlere `Mileage` als `Van`, aber die Levels `Compact`, `Sporty` und `Small` haben signifikant höhere mittlere `Mileage`-Werte als `Van`, deren Abstände (zum mittleren `Mileage`-Wert in `Van`) durch die jeweiligen Konfidenzschranken zum Niveau 95 % nach unten abgeschätzt werden können."

Die durch

```
> plot( MileageSCI.1smcc, xlab = "Mileage")
```

erzeugte grafische Darstellung (auf der nächsten Seite) zeigt nun Halbstrahlen, die an der jeweilgen Konfidenzschranke beginnen.



11.4.1.3 Weitere Möglichkeiten, C zu spezifizieren

Die die Nullhypothese beschreibende (Kontraste-)Matrix **C** kann nicht nur durch die mit `"Dunnett"` oder `"Tukey"` (siehe §11.4.2) benannten Varianten (und noch weiteren) spezifiziert werden, sondern ist durch ihre explizite Angabe nahezu frei nach den eigenen Anforderungen spezifizierbar. Es stehen mindestens zwei alternative Methoden für die Kontrastspezifikation zur Verfügung, die wir anhand von zwei Beispielen ganz knapp vorstellen:

1. Mittels symbolischer Beschreibung, in der die pro Kontrast (= Zeile von **C**) zu verwendenden Modellparameter durch die Namen der zu ihnen gehörenden Faktorlevels zu identifizieren sind und die „Seitigkeit“ der Nullhypothese durch `==`, `<=` oder eben `>=` (das Argument `alternative` ist dann nicht nötig):

```
> glht( Mileage.aov, linfct = mcp( Type.by.MeanMileage =
+  c( "Large  - Van <= 0", "Medium - Van <= 0", "Compact - Van <= 0",
+     "Sporty - Van <= 0", "Small  - Van <= 0")))

         General Linear Hypotheses

Multiple Comparisons of Means: User-defined Contrasts

Linear Hypotheses:
                   Estimate
Large - Van <= 0      1.476
Medium - Van <= 0     2.912
Compact - Van <= 0    5.276
Sporty - Van <= 0     7.143
Small - Van <= 0     12.143
```

2. Durch explizite Angabe von **C**, deren Zeilen beschreibend benannt werden können (aber nicht müssen). Die „Einseitigkeit“ der Nullhypothese muss dann im Bedarfsfalls durch `alternative` mitgeteilt werden:

```
> Contrast1 <- rbind( "Large   - Van" = c( -1, 1, 0, 0, 0, 0),
+                     "Medium  - Van" = c( -1, 0, 1, 0, 0, 0),
+                     "Compact - Van" = c( -1, 0, 0, 1, 0, 0),
+                     "Sporty  - Van" = c( -1, 0, 0, 0, 1, 0),
```

```
+                     "Small   - Van" = c( -1, 0, 0, 0, 0, 1))
> glht( Mileage.aov, alternative = "greater",
+  linfct = mcp( Type.by.MeanMileage = Contrast1))

         General Linear Hypotheses

Multiple Comparisons of Means: User-defined Contrasts

Linear Hypotheses:
                    Estimate
Large   - Van <= 0    1.476
Medium  - Van <= 0    2.912
Compact - Van <= 0    5.276
Sporty  - Van <= 0    7.143
Small   - Van <= 0   12.143
```

Beispiel: Eine gelegentlich nützliche Anwendung für die obige Freiheit sind multiple Vergleiche mit *mehreren* Kontrollen. Wir wählen willkürlich `Large` und `Van` als die zwei „Kontrollen“ und beschreiben die Nullhypothese (also **C**) zunächst symbolisch:

```
> glht( Mileage.aov, linfct = mcp( Type.by.MeanMileage =
+  c( "Medium - Large <= 0", "Compact - Large <= 0", "Sporty - Large <= 0",
+     "Small  - Large <= 0",
+     "Medium - Van <= 0",   "Compact - Van <= 0",   "Sporty - Van <= 0",
+     "Small  - Van <= 0")))

         General Linear Hypotheses

Multiple Comparisons of Means: User-defined Contrasts

Linear Hypotheses:
                     Estimate
Medium - Large <= 0     1.436
Compact - Large <= 0    3.800
Sporty - Large <= 0     5.667
Small - Large <= 0     10.667
Medium - Van <= 0       2.912
Compact - Van <= 0      5.276
Sporty - Van <= 0       7.143
Small - Van <= 0       12.143
```

Dasselbe kann erzielt werden durch die Angabe einer Kontrastmatrix (mit benannten Zeilen) wie folgt:

```
> Contrast2 <- rbind( Contrast1[ -1,],
+                     "Medium  - Large" = c( 0, -1, 1, 0, 0, 0),
+                     "Compact - Large" = c( 0, -1, 0, 1, 0, 0),
+                     "Sporty  - Large" = c( 0, -1, 0, 0, 1, 0),
+                     "Small   - Large" = c( 0, -1, 0, 0, 0, 1))
> summary( glht( Mileage.aov, alternative = "greater",
+                linfct = mcp( Type.by.MeanMileage = Contrast2)) )
```

```
         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: User-defined Contrasts
....
                     Estimate Std. Error t value p value
Medium  - Van <= 0      2.912      1.294   2.250 0.07177 .
Compact - Van <= 0      5.276      1.264   4.175 < 0.001 ***
Sporty  - Van <= 0      7.143      1.391   5.134 < 0.001 ***
Small   - Van <= 0     12.143      1.294   9.382 < 0.001 ***
Medium  - Large <= 0    1.436      1.768   0.812 0.60006
Compact - Large <= 0    3.800      1.746   2.176 0.08351 .
Sporty  - Large <= 0    5.667      1.840   3.079 0.00976 **
Small   - Large <= 0   10.667      1.768   6.032 < 0.001 ***
....
(Adjusted p values reported -- single-step method)
```

11.4.1.4 Weitere Prozeduren für multiple Vergleich mit einer Kontrolle

- Die "step-down"-Methode des Dunnett-Tests (= "Step-down"-Algorithmus für die Abschlusstestprozedur unter der "free combination condition" für die Elementarhypothesen der Dunnett-Kontraste; ebenfalls unter Verwendung von Verteilungsaussagen für max-t-Tests unter parametrischen Annahmen):

```
> summary( Mileage.2smcc, test = adjusted( type = "free"))

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts
....
                     Estimate Std. Error t value  p value
Large - Van == 0      1.476      1.905   0.775  0.44180
Medium - Van == 0     2.912      1.294   2.250  0.05333 .
Compact - Van == 0    5.276      1.264   4.175  0.00026 ***
Sporty - Van == 0     7.143      1.391   5.134 1.38e-05 ***
Small - Van == 0     12.143      1.294   9.382 1.04e-12 ***
....
(Adjusted p values reported -- free method)
```

- Holms Methode (= "Step-down"-Algorithmus mit Bonferroni-Tests in der Abschlusstestprozedur für die Elementarhypothesen der Dunnett-Kontraste; ohne parametrische Verteilungsannahmen):

```
> summary( Mileage.2smcc, test = adjusted( type = "holm"))

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts
....
                     Estimate Std. Error t value  p value
Large - Van == 0      1.476      1.905   0.775 0.441795
Medium - Van == 0     2.912      1.294   2.250 0.057088 .
Compact - Van == 0    5.276      1.264   4.175 0.000327 ***
```

```
Sporty - Van == 0     7.143      1.391   5.134 1.59e-05 ***
Small - Van == 0     12.143      1.294   9.382 3.11e-12 ***
....
(Adjusted p values reported -- holm method)
```

- Bonferronis Methode (= "Single-step"-Methode mit einem Bonferroni-Test für die Dunnett-Kontraste, also insbesondere ohne Eigenschaften der gemeinsamen t-Verteilung der einzelnen Teststatistiken für die Elementarhypothesen der Dunnett-Kontraste zu nutzen):

```
> summary( Mileage.2smcc, test = adjusted( type = "bonferroni"))

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts
....
                   Estimate Std. Error t value  p value
Large - Van == 0      1.476      1.905   0.775 1.000000
Medium - Van == 0     2.912      1.294   2.250 0.142720
Compact - Van == 0    5.276      1.264   4.175 0.000546 ***
Sporty - Van == 0     7.143      1.391   5.134 1.99e-05 ***
Small - Van == 0     12.143      1.294   9.382 3.11e-12 ***
....
(Adjusted p values reported -- bonferroni method)
```

- „Multiple“ Vergleiche mit einer Kontrolle *ohne* Korrekturmaßnahmen für die Multiplizität:

```
> summary( Mileage.2smcc, test = adjusted( type = "none"))
+                                          # Dasselbe wie test = univariate()

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts
....
                   Estimate Std. Error t value  p value
Large - Van == 0      1.476      1.905   0.775 0.441795
Medium - Van == 0     2.912      1.294   2.250 0.028544 *
Compact - Van == 0    5.276      1.264   4.175 0.000109 ***
Sporty - Van == 0     7.143      1.391   5.134 3.98e-06 ***
Small - Van == 0     12.143      1.294   9.382 6.22e-13 ***
....
(Adjusted p values reported -- none method)


> summary( Mileage.2smcc, test = Ftest()) # F-Test der globalen Nullhypothese

         General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts
....
Global Test:
      F DF1 DF2    Pr(>F)
1 24.75   5  54 7.213e-13
```

```
> summary( Mileage.2smcc, test = Chisqtest()) # Wald-Test der globalen
....                                          # Nullhypothese

Global Test:
  Chisq DF Pr(>Chisq)
1 123.7  5  5.086e-25
```

11.4.2 Alle paarweisen Vergleiche

Ist man an *allen* paarweisen Vergleichen (Engl.: "all-pairwise multiple comparisons", kurz: MCA) interessiert, so erfordert dies für einen Faktor mit k Levels $k \cdot (k-1)/2$ paarweise Vergleiche. Die sie beschreibenden linearen Kontraste heißen auch Tukey-Kontraste. Auch im Folgenden betrachten wir die Fahrleistungsdaten und darin den (willkürlich ausgewählten) Faktor `Type.by.MeanMileage`.

11.4.2.1 Zweiseitige Vergleiche und Konfidenzintervalle

Durch das Argument `linfct = mcp( Type.by.MeanMileage = "Tukey")` teilen wir der Funktion `glht()` mit, dass die Tukey-Kontraste der zu den Levels von `Type.by.MeanMileage` gehörenden `Mileage`-Mittelwerten als geeignete Hypothesenmatrix $\mathbf{C}$ zu bilden sind:

```
> (Mileage.2smca <- glht( Mileage.aov,
                          linfct = mcp( Type.by.MeanMileage = "Tukey")))

         General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts

Linear Hypotheses:
                      Estimate
Large - Van == 0         1.476
Medium - Van == 0        2.912
Compact - Van == 0       5.276
Sporty - Van == 0        7.143
Small - Van == 0        12.143
Medium - Large == 0      1.436
Compact - Large == 0     3.800
Sporty - Large == 0      5.667
Small - Large == 0      10.667
Compact - Medium == 0    2.364
Sporty - Medium == 0     4.231
Small - Medium == 0      9.231
Sporty - Compact == 0    1.867
Small - Compact == 0     6.867
Small - Sporty == 0      5.000
```

Das `glht`-Objekt `Mileage.2smca` enthält jetzt alles zur Durchführung eines Tests Notwendige, wie z. B. in seiner Komponente `Mileage.2smca$linfct` die Kontrastematrix, die aus der in **R** üblichen Treatment-Kontraste-Parametrisierung des gefitteten linearen Modells die für die Hypothese interessierenden (also durch $\mathbf{C}$ definierten) Differenzen bildet. Den eigentlichen zweiseitigen Tukey-Test, d. h. den Test von $H_0 : \mathbf{C}\boldsymbol{\mu} = \mathbf{0}$ mit der Tukey-Kontrastematrix $\mathbf{C}$ gegen

$H_1 : \mathbf{C}\boldsymbol{\mu} \neq \mathbf{0}$ unter Verwendung der "single-step"-Methode für den Tukey-Test (auf der Basis eines sogenannten max-t-Tests und parametrischer multivariater t-Verteilungseigenschaften) führt `summary()` angewendet auf das `glht`-Objekt aus:

```
> summary( Mileage.2smca)

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts
....
                       Estimate Std. Error t value p value
Large - Van == 0          1.476      1.905   0.775 0.96911
Medium - Van == 0         2.912      1.294   2.250 0.22413
Compact - Van == 0        5.276      1.264   4.175 0.00139 **
Sporty - Van == 0         7.143      1.391   5.134 < 0.001 ***
Small - Van == 0         12.143      1.294   9.382 < 0.001 ***
Medium - Large == 0       1.436      1.768   0.812 0.96229
Compact - Large == 0      3.800      1.746   2.176 0.25631
Sporty - Large == 0       5.667      1.840   3.079 0.03443 *
Small - Large == 0       10.667      1.768   6.032 < 0.001 ***
Compact - Medium == 0     2.364      1.046   2.260 0.22004
Sporty - Medium == 0      4.231      1.197   3.534 0.00971 **
Small - Medium == 0       9.231      1.083   8.525 < 0.001 ***
Sporty - Compact == 0     1.867      1.164   1.604 0.58919
Small - Compact == 0      6.867      1.046   6.564 < 0.001 ***
Small - Sporty == 0       5.000      1.197   4.177 0.00134 **
....
(Adjusted p values reported -- single-step method)
```

Laut letzter Zeile der Ausgabe sind die angegebenen p-Werte adjustiert, und zwar gemäß der "single-step"-Methode (für den Tukey-Test der Tukey-Kontraste $\mathbf{C}\boldsymbol{\mu}$).

Folgerung: Auf einem Signifikanzniveau von 5 % gilt die folgende „Global"-Aussage über die (hier 15) betrachteten Vergleiche der Levels `Large`, `Medium`, `Compact`, `Sporty`, `Small` und `Van` miteinander: „Es besteht hinsichtlich der `Mileage`-Mittelwerte kein signifikanter Unterschied in den obigen sechs Vergleichen Nummer 1, 2, 6, 7, 10 und 13, wohl aber bestehen signifikante Unterschiede in den neun anderen Vergleichen 3, 4, 5, 8, 9, 11, 12, 14 und 15."

Bemerkung: Dies ist dasselbe Ergebnis wie für Tukeys HSD-Methode (= Methode der "honestly significant differences"), in **R** durch `TukeyHSD( Mileage.aov)` zu erhalten, *falls* das Design balanciert oder höchstens geringfügig unbalanciert ist.

Zweiseitige simultane Konfidenzintervalle erhält man durch `confint()` angewendet auf das obige `glht`-Objekt:

```
> (MileageSCI.2smca <- confint( Mileage.2smca, level = 0.95))

         Simultaneous Confidence Intervals

Multiple Comparisons of Means: Tukey Contrasts

Fit: aov(formula = Mileage ~ Type.by.MeanMileage, data = car.test.frame)
```

```
Estimated Quantile = 2.9331
95% family-wise confidence level
Linear Hypotheses:
                      Estimate lwr     upr
Large - Van == 0       1.4762  -4.1115  7.0639
Medium - Van == 0      2.9121  -0.8840  6.7082
Compact - Van == 0     5.2762   1.5697  8.9827
Sporty - Van == 0      7.1429   3.0622 11.2235
Small - Van == 0      12.1429   8.3468 15.9390
Medium - Large == 0    1.4359  -3.7506  6.6224
Compact - Large == 0   3.8000  -1.3212  8.9212
Sporty - Large == 0    5.6667   0.2684 11.0649
Small - Large == 0    10.6667   5.4802 15.8531
Compact - Medium == 0  2.3641  -0.7042  5.4325
Sporty - Medium == 0   4.2308   0.7195  7.7420
Small - Medium == 0    9.2308   6.0547 12.4068
Sporty - Compact == 0  1.8667  -1.5475  5.2808
Small - Compact == 0   6.8667   3.7983  9.9350
Small - Sporty == 0    5.0000   1.4888  8.5112
```

Die Ausgabe dokumentiert, dass simultane 95 %-Konfidenzintervalle für die Tukey-Kontraste, d. h. für die Differenzen zwischen den `Mileage`-Mittelwerten aller `Type`-Levels berechnet wurden, und zwar – ohne dass es hier explizit gesagt wird – mittels der Invertierung der "single-step"-Methode für den Tukey-Test. (Das `Estimated quantile` ist das für die Konfidenzintervalle der Differenzen verwendete geschätzte (!) 95 %-Quantil der Verteilung der betraglichen Maximumskomponente einer multivariaten (hier 15-dimensionalen!) t-Verteilung, deren Korrelationsstruktur aus der des multivariat normalverteilten Parameterschätzers $\hat{\boldsymbol{\mu}}$ abgeleitet und geschätzt wurde.) Die Konfidenzintervalle, die nicht die Null enthalten, identifizieren die dazugehörigen Mittelwerte als signifikant voneinander verschieden.

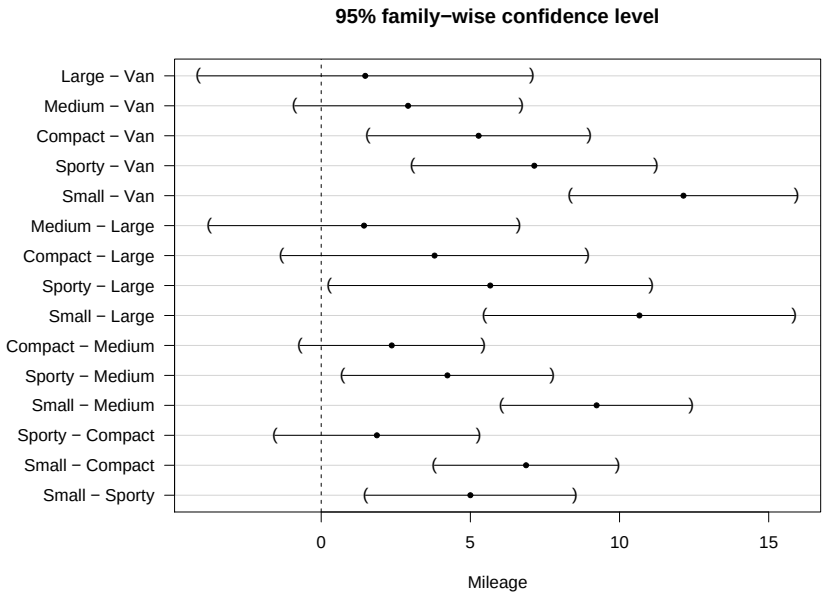
Folgerung: Auf einem Konfidenzniveau von 95 % gilt die folgende „Global"-Aussage über die (hier 15) betrachteten Vergleiche aller Levels: „Es besteht bzgl. der `Mileage`-Mittelwerte kein signifikanter Unterschied in den sechs Vergleichen Nummer 1, 2, 6, 7, 10 und 13, wohl aber bestehen signifikante Unterschiede in den neun anderen Vergleichen 3, 4, 5, 8, 9, 11, 12, 14 und 15, deren Größenordnungen – gemeint sind natürlich die Differenzen zwischen den levelspezifischen `Mileage`-Mittelwerten – durch die Grenzen der jeweiligen Konfidenzintervalle zum Niveau 95 % nach oben und nach unten abgeschätzt werden können."

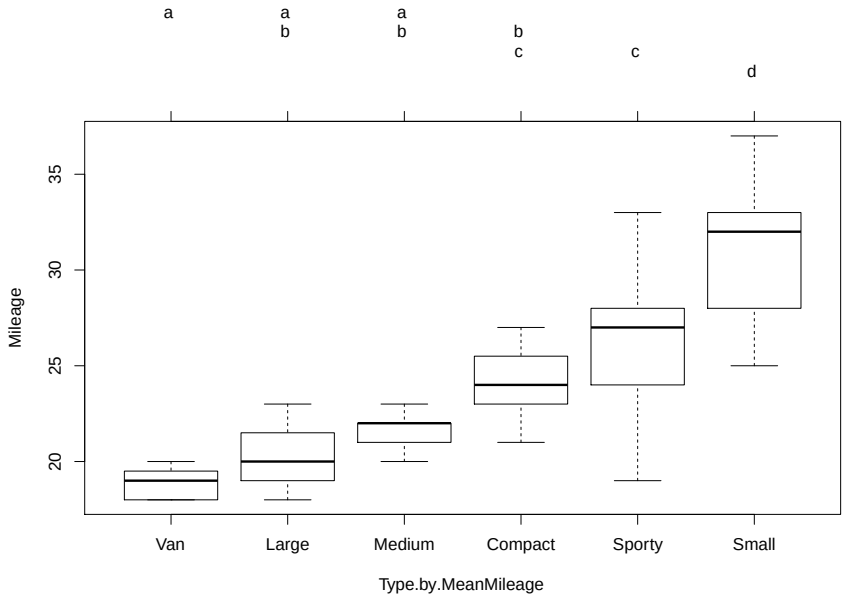
`plot()` angewendet auf das Ergebnis von `confint()` produziert die auf der folgenden Seite oben gezeigte grafische Darstellung dieser zweiseitigen simultanen Konfidenzintervalle:

```
> plot( MileageSCI.2smca, xlab = "Mileage")
```

Oft übersichtlicher oder gewünscht ist ein "compact letter display", das mit Hilfe der Funktion `cld()` des `multcomp`-Pakets vorbereitet wird und deren Ergebnis durch `plot()` zur Grafik (auf der folgenden Seite in der Mitte) führt. (Gibt es nur für alle paarweisen Vergleiche.)

```
> plot( cld( Mileage.2smca))
```





Bemerkung: Wie das oben gezeigte "compact letter display" genau zustandegekommen ist, ist mir unklar. Der in [67, Piepho (2004)] beschriebene Algorithmus ist laut `cld()`s Online-Hilfe die Grundlage dafür.

11.4.2.2 Einseitige Vergleiche und Konfidenzschranken

Häufig ist man an einseitigen Hypothesentests interessiert und es ist auch nicht notwendig, beidseitig Konfidenzgrenzen, also Konfidenz*intervalle* anzugeben, sondern einseitige (obere oder untere) Konfidenz*schranken* sind ausreichend. Damit sind einseitige multiple Vergleiche und ihre simultanen Konfidenzschranken gefragt.

Am Beispiel des vorherigen Abschnitts demonstrieren wir, wie die `glht()` das Gewünschte liefert: Der einzige Unterschied ist die Verwendung des Arguments `alternative = "greater"` in `glht()`. Das führt durch `summary()` schließlich zur Durchführung des einseitigen Tukey-Tests (mittels der "single-step"-Methode) und zur Berechnung – hier – *unterer* Schranken für die Tukey-Kontraste, d. h. für die Differenzen der `Mileage`-Mittelwerte aller `Type`-Levels:

```
> Mileage.1smca <- glht( Mileage.aov, alternative = "greater",
+                        linfct = mcp( Type.by.MeanMileage = "Tukey"))
> summary( Mileage.1smca)

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts

Fit: aov(formula = Mileage ~ Type.by.MeanMileage, data = car.test.frame)

Linear Hypotheses:
                      Estimate Std. Error t value p value
Large - Van <= 0         1.476      1.905   0.775 0.86955
Medium - Van <= 0        2.912      1.294   2.250 0.13674
Compact - Van <= 0       5.276      1.264   4.175 < 0.001 ***
Sporty - Van <= 0        7.143      1.391   5.134 < 0.001 ***
Small - Van <= 0        12.143      1.294   9.382 < 0.001 ***
Medium - Large <= 0      1.436      1.768   0.812 0.85454
Compact - Large <= 0     3.800      1.746   2.176 0.15800
Sporty - Large <= 0      5.667      1.840   3.079 0.01919 *
Small - Large <= 0      10.667      1.768   6.032 < 0.001 ***
Compact - Medium <= 0    2.364      1.046   2.260 0.13378
Sporty - Medium <= 0     4.231      1.197   3.534 0.00513 **
Small - Medium <= 0      9.231      1.083   8.525 < 0.001 ***
Sporty - Compact <= 0    1.867      1.164   1.604 0.41222
Small - Compact <= 0     6.867      1.046   6.564 < 0.001 ***
Small - Sporty <= 0      5.000      1.197   4.177 < 0.001 ***
....
(Adjusted p values reported -- single-step method)
```

(Beachte, dass die p-Werte einseitiger Tests – wie immer – kleiner sind als diejenigen der korrespondierenden zweiseitigen Tests!)

Folgerung: Auf einem Signifikanzniveau von 5 % gilt die folgende „Global"-Aussage über die (hier 15) betrachteten Vergleiche der `Mileage`-Mittelwerte aller `Type`-Levels: „In den sechs Vergleichen Nummer 1, 2, 6, 7, 10 und 13 liegen keine signifikanten Ergebnisse vor, wohl aber in den neun anderen Vergleichen 3, 4, 5, 8, 9, 11, 12, 14 und 15."

Die dazugehörigen einseitigen simultane Konfidenzintervalle, sprich -schranken (ermittelt durch Invertierung der "single-step"-Methode für den einseitigen Tukey-Test):

```
> (MileageSCI.1smca <- confint( Mileage.1smca, level = 0.95))

         Simultaneous Confidence Intervals

Multiple Comparisons of Means: Tukey Contrasts

Fit: aov(formula = Mileage ~ Type.by.MeanMileage, data = car.test.frame)

Estimated Quantile = 2.7003
95% family-wise confidence level
Linear Hypotheses:
                      Estimate lwr     upr
```

```
Large - Van <= 0       1.4762  -3.6681     Inf
Medium - Van <= 0      2.9121  -0.5828     Inf
Compact - Van <= 0     5.2762   1.8639     Inf
Sporty - Van <= 0      7.1429   3.3860     Inf
Small - Van <= 0      12.1429   8.6480     Inf
Medium - Large <= 0    1.4359  -3.3390     Inf
Compact - Large <= 0   3.8000  -0.9148     Inf
Sporty - Large <= 0    5.6667   0.6968     Inf
Small - Large <= 0    10.6667   5.8918     Inf
Compact - Medium <= 0  2.3641  -0.4607     Inf
Sporty - Medium <= 0   4.2308   0.9982     Inf
Small - Medium <= 0    9.2308   6.3068     Inf
Sporty - Compact <= 0  1.8667  -1.2765     Inf
Small - Compact <= 0   6.8667   4.0418     Inf
Small - Sporty <= 0    5.0000   1.7674     Inf
```

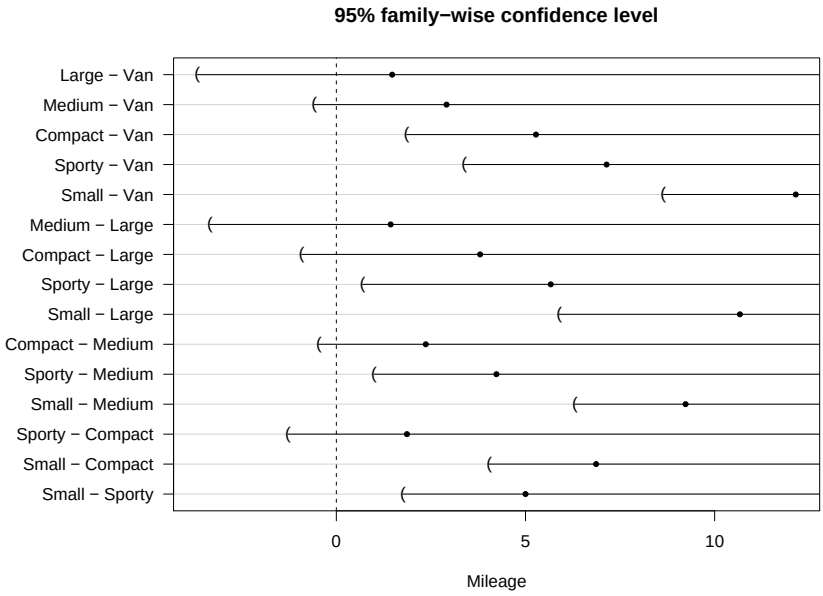
(Beachte, dass die unteren Konfidenzschranken größer sind als die Untergrenzen der zweiseitigen Konfidenzintervalle zum selben Niveau (wie auch obere Konfidenzschranken kleiner wären als die Obergrenzen zweiseitiger Konfidenzintervalle).)

Ist die Null außerhalb eines Konfidenzbereichs, so ist – hier – seine untere Schranke größer als Null und somit seine zugehörige Mittelwertedifferenz signifikant größer als Null. Dies bedeutet, dass der `Mileage`-Mittelwert des jeweils erstgenannten Levels signifikant größer als der des zweitgenannten ist.

Folgerung: Auf einem Konfidenzniveau von 95 % gilt die folgende „Global"-Aussage über die (hier 15) betrachteten Vergleiche der `Mileage`-Mittelwerte aller: „In den sechs Vergleichen Nummer 1, 2, 6, 7, 10 und 13 liegen keine signifikanten Ergebnisse vor, wohl aber in den neun anderen Vergleichen 3, 4, 5, 8, 9, 11, 12, 14 und 15, deren Abstände durch die jeweiligen Konfidenzschranken zum Niveau 95 % nach unten abgeschätzt werden können."

Die grafische Darstellung:

```
> plot( MileageSCI.1smca, xlab = "Mileage")
```



11.4.2.3 Weitere Prozeduren für alle paarweisen Vergleich

- Westfalls "step-down"-Methode des Tukey-Tests (= "Step-down"-Algorithmus, in dem die vollständige Abschlusstestprozedur unter Ausnutzung von logischen Einschränkungen für die Elementarhypothesen der Tukey-Kontraste trunkiert („gestutzt") wird):

```
> summary( Mileage.2smca, test = adjusted( type = "Westfall"))

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts
....
                      Estimate Std. Error t value p value
Large - Van == 0         1.476      1.905   0.775 0.44180
Medium - Van == 0        2.912      1.294   2.250 0.09592 .
Compact - Van == 0       5.276      1.264   4.175 < 0.001 ***
Sporty - Van == 0        7.143      1.391   5.134 < 0.001 ***
Small - Van == 0        12.143      1.294   9.382 < 0.001 ***
Medium - Large == 0      1.436      1.768   0.812 0.42034
Compact - Large == 0     3.800      1.746   2.176 0.09592 .
Sporty - Large == 0      5.667      1.840   3.079 0.01182 *
Small - Large == 0      10.667      1.768   6.032 < 0.001 ***
Compact - Medium == 0    2.364      1.046   2.260 0.06715 .
Sporty - Medium == 0     4.231      1.197   3.534 0.00423 **
Small - Medium == 0      9.231      1.083   8.525 < 0.001 ***
Sporty - Compact == 0    1.867      1.164   1.604 0.21489
Small - Compact == 0     6.867      1.046   6.564 < 0.001 ***
Small - Sporty == 0      5.000      1.197   4.177 < 0.001 ***
....
(Adjusted p values reported -- Westfall method)
```

- Shaffers Methode „S2" (= "Step-down"-Algorithmus mit Bonferroni-Tests, in dem die vollständige Abschlusstestprozedur wie oben bei Westfall trunkiert („gestutzt") wird):

```
> summary( Mileage.2smca, test = adjusted( type = "Shaffer"))

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts
....
                      Estimate Std. Error t value  p value
Large - Van == 0         1.476      1.905   0.775  0.44180
Medium - Van == 0        2.912      1.294   2.250  0.11418
Compact - Van == 0       5.276      1.264   4.175  0.00076 ***
Sporty - Van == 0        7.143      1.391   5.134 3.98e-05 ***
Small - Van == 0        12.143      1.294   9.382 9.33e-12 ***
Medium - Large == 0      1.436      1.768   0.812  0.42034
Compact - Large == 0     3.800      1.746   2.176  0.11418
Sporty - Large == 0      5.667      1.840   3.079  0.01305 *
Small - Large == 0      10.667      1.768   6.032 1.06e-06 ***
Compact - Medium == 0    2.364      1.046   2.260  0.08366 .
Sporty - Medium == 0     4.231      1.197   3.534  0.00509 **
Small - Medium == 0      9.231      1.083   8.525 1.42e-10 ***
```

```
Sporty - Compact == 0    1.867      1.164   1.604  0.22926
Small - Compact == 0     6.867      1.046   6.564 1.47e-07 ***
Small - Sporty == 0      5.000      1.197   4.177  0.00076 ***
....
(Adjusted p values reported -- Shaffer method)
```

- Bonferronis Methode (= "Single-step"-Methode mit einem Bonferroni-Test für die Tukey-Kontraste, also insbesondere ohne Eigenschaften der gemeinsamen t-Verteilung der einzelnen Teststatistiken für die Elementarhypothesen der Tukey-Kontraste zu nutzen):

```
> summary( Mileage.2smca, test = adjusted( type = "bonferroni"))

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts
....
                      Estimate Std. Error t value  p value
Large - Van == 0         1.476      1.905   0.775  1.00000
Medium - Van == 0        2.912      1.294   2.250  0.42816
Compact - Van == 0       5.276      1.264   4.175  0.00164 **
Sporty - Van == 0        7.143      1.391   5.134 5.97e-05 ***
Small - Van == 0        12.143      1.294   9.382 9.33e-12 ***
Medium - Large == 0      1.436      1.768   0.812  1.00000
Compact - Large == 0     3.800      1.746   2.176  0.50882
Sporty - Large == 0      5.667      1.840   3.079  0.04894 *
Small - Large == 0      10.667      1.768   6.032 2.27e-06 ***
Compact - Medium == 0    2.364      1.046   2.260  0.41829
Sporty - Medium == 0     4.231      1.197   3.534  0.01272 *
Small - Medium == 0      9.231      1.083   8.525 2.13e-10 ***
Sporty - Compact == 0    1.867      1.164   1.604  1.00000
Small - Compact == 0     6.867      1.046   6.564 3.15e-07 ***
Small - Sporty == 0      5.000      1.197   4.177  0.00163 **
....
(Adjusted p values reported -- bonferroni method)
```

- „Alle paarweisen“ Vergleiche *ohne* Korrekturmaßnahmen für die Multiplizität:

```
> summary( Mileage.2smca, test = adjusted( type = "none"))
+                                          # Dasselbe wie test = univariate()

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts
....
                      Estimate Std. Error t value  p value
Large - Van == 0         1.476      1.905   0.775 0.441795
Medium - Van == 0        2.912      1.294   2.250 0.028544 *
Compact - Van == 0       5.276      1.264   4.175 0.000109 ***
Sporty - Van == 0        7.143      1.391   5.134 3.98e-06 ***
Small - Van == 0        12.143      1.294   9.382 6.22e-13 ***
Medium - Large == 0      1.436      1.768   0.812 0.420338
Compact - Large == 0     3.800      1.746   2.176 0.033921 *
Sporty - Large == 0      5.667      1.840   3.079 0.003263 **
```

```
Small - Large == 0      10.667      1.768   6.032 1.51e-07 ***
Compact - Medium == 0    2.364      1.046   2.260 0.027886 *
Sporty - Medium == 0     4.231      1.197   3.534 0.000848 ***
Small - Medium == 0      9.231      1.083   8.525 1.42e-11 ***
Sporty - Compact == 0    1.867      1.164   1.604 0.114628
Small - Compact == 0     6.867      1.046   6.564 2.10e-08 ***
Small - Sporty == 0      5.000      1.197   4.177 0.000109 ***
....
(Adjusted p values reported -- none method)


> summary( Mileage.2smca, test = Ftest()) # F-Test der globalen Nullhypothese

         General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts
....
Global Test:
      F DF1 DF2    Pr(>F)
1 24.75   5  54 7.213e-13


> summary( Mileage.2smca, test = Chisqtest()) # Wald-Test der globalen
....                                          # Nullhypothese

Global Test:
  Chisq DF Pr(>Chisq)
1 123.7  5  5.086e-25
```

11.4.3 Zu nichtparametrischen multiplen Vergleichen

Hierzu nur ein paar kurze Hinweise:

- Für alle paarweisen Vergleiche auf Lokationsshifts zwischen als stetig angenommenen Verteilungen (in den Levels eines gruppierenden Faktors) mittels Wilcoxon-Rangsummentests mit p-Werte-Adjustierung für multiples Testen steht die Funktion `pairwise.wilcox.test()` zur Verfügung. Es sind mehrere Adjustierungsverfahren („Bonferroni“, „Holm“ u. a.) zur Auswahl. Die Online-Hilfe zu `pairwise.wilcox.test()` liefert Details.
- Das Paket `npmc` enthält simultane Rangtestverfahren (vom Behrens-Fisher- und vom Steel-Typ) für alle paarweisen Vergleiche sowie für multiple Vergleiche mit einer Kontrolle im einfaktoriellen Design, ohne (!) dass Annahmen über die zugrundeliegenden Verteilungsfunktionen gemacht werden. (Quelle: [63, Munzel und Hothorn (2001)].)
- Mathematischer Hintergrund wird z. B. in [48, Hollander und Wolfe (1973)] geliefert.

12 Exkurs: Einführung in die Lebensdaueranalyse ("Survival Analysis")

Die Analyse von „Lebensdauern" spielt eine wichtige Rolle in medizinischen und biologischen Studien. Sie kann beispielsweise der Beurteilung der Wirksamkeit einer Therapie bei einer bestimmten Krebsart dienen oder der Beurteilung der Toxizität von Schadstoffen auf Organismen. Auch in ingenieurswissenschaftlichen Untersuchungen zur Funktionsdauer von mechanischen oder elektronischen Komponenten findet sie Anwendung. Der Begriff „Lebensdauer" (= "survival time") ist dabei als *terminus technicus* zu verstehen, da es sich im Allgemeinen nicht um eine Lebensdauer im biologischen oder funktionstechnischen Sinn zu handeln braucht. Vielmehr ist damit ein Datum gemeint, das eine Zeitspanne von einem wohldefinierten Startzeitpunkt bis zum Eintritt eines gewissen Ereignisses (dem Endzeitpunkt) ist. Dennoch werden diese Daten einheitlich als Lebensdauern bezeichnet. Als Beispiel aus der Ökonomie möge die Dauer ab Eintritt in Arbeitslosigkeit bis zur Wiederaufnahme einer Beschäftigung dienen.

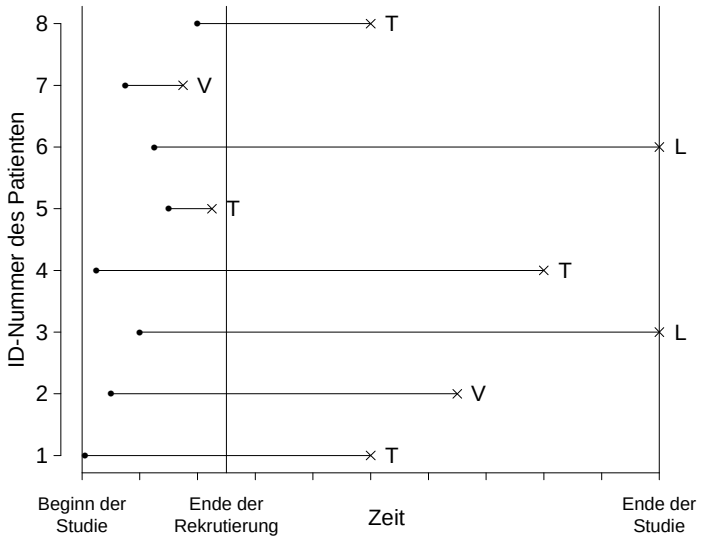
Es kann vorkommen, dass die interessierende Lebensdauer – wie die Zeit vom Beginn der Therapie für einen Krebspatienten bis zu seinem Tod oder die Zeit vom Beginn der Verabreichung eines Giftstoffes bis zum Auftreten von Krankheitssymptomen bei einem Versuchstier oder die Dauer des Einsatzes eines bestimmten Maschinenbauteils bis zu seinem Ausfall – *nicht* beobachtet werden kann. Das geschieht z. B. im Fall einer Krebsstudie, wenn bei einem Patienten die Lebensdauer aufgrund anderer Ereignisse nicht beobachtbar ist. Letzteres kann wegen eines Umzuges und dem damit verbundenen Ausscheiden aus der Studie oder wegen eines Unfalltodes (statt Tod durch die in der Studie untersuchte Krebsart) oder schlicht aufgrund der Beendigung der Studie selbst geschehen. Wir sprechen dann von einer Zensierung der Lebensdauer.

Eine – *äußerst* unvollständige – Auswahl an Literaturquellen, in denen für das Folgende unterschiedlich umfangreich recherchiert wurde, ist (in alphabetischer Reihung) [22, Collett (2015)], [61, Meeker und Escobar (1998)], [73, Stute (2004)] und [74, Therneau und Grambsch (2001)].

12.1 Das Modell der zufälligen Rechtszensierung

Es gibt mehrere verschiedene Arten der Zensierung. Wir beschränken uns auf die Rechtszensierung im "random-censorship"-Modell, wie sie in folgender Grafik anhand eines möglichen Verlaufs einer Survival-Studie mit acht Patienten veranschaulicht wird:

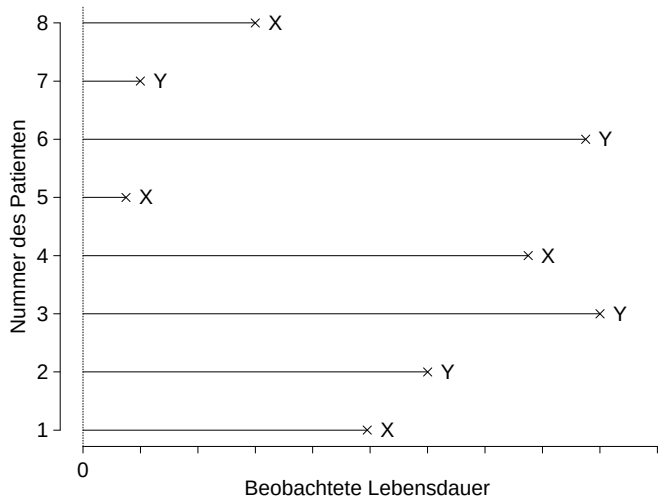
Typischerweise treten die Teilnehmer nicht alle gleichzeitig in die Studie ein, sondern während der sogenannten „Rekrutierungsphase" zu verschiedenen Zeitpunkten (markiert durch •). In unserem Beispiel gehen bis zum Ende der Studie die Patienten 2 und 7 während der Beobachtungsphase „verloren"; sie sind mit einem V markiert. Die Individuen 1, 4, 5 und 8 versterben im Verlauf der Studie an der untersuchten Krankheit (markiert durch ein T), wohingegen die Personen 3 und 6 am Ende der Studie noch leben (L).



Da wir an den individuellen Lebens**dauern** der Patienten interessiert sind, d. h. am zeitlichen

Abstand zwischen Behandlungsbeginn und Todeszeitpunkt, verschieben wir die Startpunkte der Beobachtungszeiträume in einen gemeinsamen zeitlichen Nullpunkt:

Die Todeszeitpunkte der Patienten 2, 3, 6 und 7 konnten (aufgrund des „Verlusts" des Patienten (V) oder des Überlebens des Studienendes (L)) nicht beobachtet werden. Die jeweiligen Lebensdauern können also nicht berechnet werden; sie sind rechtszensiert und in nebenstehendem Bild mit einem Z markiert. Die wirklich beobachteten Lebensdauern sind mit einem T versehen.

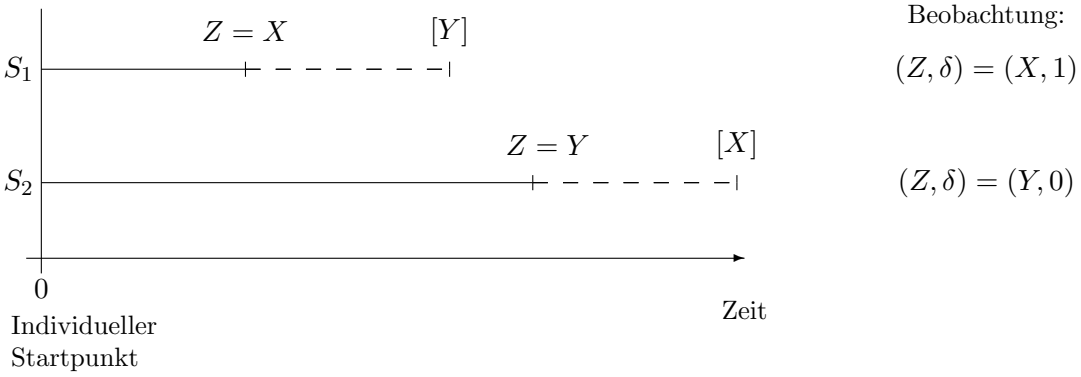


Zur mathematischen Modellierung sei $X_1, \ldots, X_n$ eine Folge von unabhängig und identisch nach einer Verteilungsfunktion F verteilten Zufallsvariablen mit nicht-negativen, reellen Werten, die die **wahren Lebensdauern** sind. Diese X_i sind dem Risiko ausgesetzt, von rechts zensiert zu werden. Zur Formalisierung dieses Sachverhalts sei $Y_1, \ldots, Y_n$ eine von den X_i unabhängige Folge von unabhängig und identisch nach einer Verteilungsfunktion G verteilten nicht-negativen, reellwertigen Zufallsvariablen, den **zensierenden Zeitdauern**. Beobachtbar sind für $i = 1, \ldots, n$ jedoch nur

$$\begin{aligned} Z_i &= \min\{X_i, Y_i\} && \text{— die } \textbf{beobachtete Lebensdauer} \\ \delta_i &= 1_{\{X_i \le Y_i\}} && \text{— der zugehörige } \textbf{Zensierungsindikator} \\ &= \begin{cases} 1 \iff X_i \le Y_i \iff \text{Wahre Lebensdauer beobachtet, d. h. } Z_i = X_i \\ 0 \iff X_i > Y_i \iff \text{Zensierungszeit beobachtet, d. h. } Z_i = Y_i < X_i \end{cases} \end{aligned}$$

Im Fall $\delta_i = 0$ ist also nur bekannt, dass der Patient oder die Patientin *mindestens* so lange überlebt hat wie durch $Z_i = Y_i$ verzeichnet. Aber es wäre ein Verlust an Information, wenn diese Beobachtung gänzlich ignoriert würde.

Grafisch veranschaulicht ist für ein Individuum genau eines der zwei folgenden Szenarien S_1 und S_2 möglich, wobei die Größe in eckigen Klammern jeweils unbeobachtet bleibt:



Das Ziel der Survival Analysis ist, etwas über die Verteilungsfunktion F der wahren Lebensdauern zu erfahren. Aus Darstellungsgründen, die weiter unten klar werden, betrachtet man an Stelle von $F(x)$ ($\equiv \mathbb{P}(X \le x) =$ Wahrscheinlichkeit, vor der Zeit x zu versterben) die sogenannte <u>Survival-Funktion</u> $1 - F(x) = \mathbb{P}(X > x)$, die die Wahrscheinlichkeit angibt, die Zeit x zu überleben.

12.2 Nicht-Parametrik: Der Kaplan-Meier-Schätzer für die Survival-Funktion

Ist man ohne Informationen über die Klasse der Verteilungsfunktionen, zu der F gehört – was der Regelfall ist – so ist es nahe liegend, einen nicht-parametrischen Schätzer zu verwenden. Im „klassischen“ Fall vollständig beobachtbarer X_i ist die empirische Verteilungsfunktion $t \longmapsto F_n(t) = n^{-1}\sum_{i=1}^n 1_{\{X_i \le t\}}$ ein bekanntermaßen hervorragender nicht-parametrischer Schätzer für F, welcher hier aber offensichtlich *nicht geeignet* ist, da F_n für zensierte X_i gar nicht berechenbar ist. Die empirische Verteilungsfunktion $t \longmapsto H_n(t) := n^{-1}\sum_{i=1}^n 1_{\{Z_i \le t\}}$ der Beobachtungen Z_i ist natürlich auch *nicht geeignet*, um die Verteilungsfunktion F der X_i zu schätzen, denn schließlich ist H_n ein Schätzer der Verteilungsfunktion H der Z_i!

Um zu einem Schätzer für F auf der Basis der beobachtbaren (Z_i, δ_i) zu gelangen, muss zunächst eine Darstellung für F gefunden werden, welche aus Größen besteht, die aus den Daten (Z_i, δ_i) geschätzt werden können. Im folgenden Abschnitt wird eine solche Darstellung angegeben und einer der bekanntesten nicht-parametrischen Schätzer für die Survival-Funktion vorgestellt.

12.2.1 Motivation, Definition und Berechnung

Der Einfachheit halber nehmen wir zunächst an, dass wir **im absolutstetigen Fall** sind und F die Dichte f besitzt.

Eine wichtige Rolle in der Survival Analysis spielt die sogenannte Hazard-Funktion (oder auch Hazard-Rate) λ definiert für $t \ge 0$ durch

$$\begin{aligned} \lambda(t) &:= \lim_{\delta\to 0} \frac{\mathbb{P}(t \le X < t+\delta | X \ge t)}{\delta} = \lim_{\delta\to 0} \frac{\mathbb{P}(t \le X < t+\delta)}{\delta\, \mathbb{P}(X \ge t)} \\ &= \frac{1}{1-F(t-)} \lim_{\delta\to 0} \frac{F(t+\delta)-F(t)}{\delta} = \frac{f(t)}{1-F(t)} \end{aligned} \tag{115}$$

Sie ist ein Maß für das Risiko, im nächsten Augenblick (besser: im infinitesimal kurzen Intervall $[t, t+\delta)$) zu sterben, falls man bis zur Zeit t überlebt hat. Aus (115) folgt also im absolutstetigen Fall

$$\lambda(t) = -\frac{d}{dt}\log(1-F(t)) \qquad \text{und somit} \qquad 1-F(t) = \exp\left\{-\int_0^t \lambda(u)\, du\right\} \tag{116}$$

Im **Fall einer allgemeinen Verteilungsfunktion** F definiert man für jedes $t \ge 0$ die kumulative Hazard-Funktion Λ durch

$$\Lambda(t) := \int_0^t \frac{F(du)}{1-F(u-)} \tag{117}$$

Demnach gilt für absolutstetiges F: $\Lambda(t) = \int_0^t \lambda(u)du$ und also $1-F(t) = \exp\{-\Lambda(t)\}$.

Die zwei entscheidenden Aspekte bei der Schätzung von F liegen in einer Verallgemeinerung der rechten Gleichung in (116) auf obiges Λ und einer geeigneten, nämlich schätzbaren Darstellung von Λ. Dies wird im Folgenden ausgeführt:

i) Zum einen lässt sich Λ additiv in einen stetigen Anteil Λ_c und einen diskreten Anteil zerlegen:

$$\Lambda_c(t) = \Lambda(t) - \sum_{a_i \in A, a_i \le t} \Lambda\{a_i\},$$

wobei $\Lambda\{a_i\} = \Lambda(a_i) - \Lambda(a_i-)$ und A die abzählbare Menge der F-Atome ist. Man kann beweisen, dass gilt:

$$1-F(t) = \exp\{-\Lambda_c(t)\} \cdot \prod_{a_i \in A, a_i \le t} (1-\Lambda\{a_i\}), \tag{118}$$

woraus im absolutstetigen Fall (in dem $\Lambda_c \equiv \Lambda$ und $A = \emptyset$ ist) die rechte Gleichung in (116) folgt und im Fall eines rein diskreten Fs (in dem $\Lambda \equiv 0$ ist) die Beziehung $1 - F(t) = \prod_{\substack{a_i \in A \\ a_i \le t}} (1 - \Lambda\{a_i\})$.

ii) Zum anderen lässt sich aus der Unabhängigkeit von X und Y für die (Sub-)Verteilungsfunktionen

$$\begin{aligned} H(z) &:= \mathbb{P}(Z \le z) \qquad \text{und} \\ H^{(1)}(z) &:= \mathbb{P}(Z \le z, \delta = 1) = \mathbb{P}(\min\{X, Y\} \le z, X \le Y), \end{aligned} \tag{119}$$

folgern, dass gilt:

$$\begin{aligned} 1 - H(z-) &= (1 - F(z-))(1 - G(z-)) \qquad \text{und} \\ H^{(1)}(z) &= \mathbb{P}(X \le z, X \le Y) \ = \int_0^z \mathbb{P}(Y \ge x) \ F(dx) \\ &= \int_0^z (1 - G(x-)) \ F(dx) \end{aligned}$$

Wir erhalten mit ii) in Fortsetzung von (117)

$$\Lambda(t) = \int_0^t \frac{(1 - G(u-)) \ F(du)}{(1 - G(u-))(1 - F(u-))} \ = \int_0^t \frac{H^{(1)}(du)}{1 - H(u-)}, \tag{120}$$

und Einsetzen der empirischen Analoga $H_n = n^{-1} \sum_{i=1}^n 1_{\{Z_i \le \cdot\}}$ für H und $H_n^{(1)} = n^{-1} \sum_{i=1}^n 1_{\{Z_i \le \cdot, \delta_i = 1\}}$ für $H^{(1)}$ führt zu einem Schätzer für Λ:

$$\Lambda_n(t) := \int_0^t \frac{H_n^{(1)}(du)}{1 - H_n(u-)} \tag{121}$$

Für diesen rein diskreten, sogenannten Nelson-Aalen-Schätzer Λ_n ist sein stetiger Anteil $\Lambda_{n,c} \equiv 0$ (per definitionem) und die Menge A_n der Atome von Λ_n besteht offenbar nur aus denjenigen Z_i, für die $\delta_i = 1$ ist. Wir erhalten somit aus (118) einen Schätzer für F, der nur aus dem Produktterm besteht. Es ist der Kaplan-Meier-Schätzer der Survival-Funktion (im Folgenden kurz KM-Schätzer):

$$1 - \hat{F}_n(t) := \prod_{i: Z_i \le t, \delta_i = 1} (1 - \Lambda_n\{Z_i\}) \tag{122}$$

Ein anderer Name für $1 - \hat{F}_n$ ist auch Produkt-Limes-Schätzer.

Zur näheren Bestimmung der Struktur des KM-Schätzers leiten wir aus (121) ab, dass

$$\Lambda_n\{Z_i\} = \frac{\frac{1}{n}\delta_i}{1 - H_n(Z_i-)} = \frac{\delta_i}{n - R(Z_i) + 1} \tag{123}$$

ist, wobei $R(Z_i)$ der Rang von Z_i unter $Z_1, \ldots, Z_n$ ist. Beim Einsetzen in (122) kann wegen des δ_i's im Zähler von $\Lambda_n\{Z_i\}$ auf die Bedingung $\delta_i = 1$ für den Laufindex i verzichtet und demnach über alle Z_i multipliziert werden. Das liefert

$$1 - \hat{F}_n(t) = \prod_{Z_i \le t} \left(1 - \frac{\delta_i}{n - R(Z_i) + 1}\right) = \prod_{Z_i \le t} \left(\frac{n - R(Z_i)}{n - R(Z_i) + 1}\right)^{\delta_i} = \prod_{Z_{i:n} \le t} \left(\frac{n - i}{n - i + 1}\right)^{\delta_{[i:n]}}, \tag{124}$$

wobei $Z_{i:n}$ die i-te Ordnungsstatistik der $Z_1, \ldots, Z_n$ ist und $\delta_{[i:n]}$ die zu $Z_{i:n}$ gehörende Konkomitante der Zensierungsindikatoren $\delta_1, \ldots, \delta_n$. In einer der obigen oder ähnlichen Produktdarstellungen trifft man den KM-Schätzer in der Literatur meistens an.

Doch es lässt sich auch eine Summendarstellung für ihn angeben: Aus (124) ist relativ leicht zu bestimmen, welches Gewicht W_{ni} der KM-Schätzer jedem $Z_{i:n}$ zuordnet. Konkret ist

$$W_{ni} = \frac{\delta_{[i:n]}}{n-i+1} \prod_{j=1}^{i-1} \left(\frac{n-j}{n-j+1} \right)^{\delta_{[j:n]}} \qquad \text{für } i = 1, \ldots, n \tag{125}$$

Damit hat $\hat{F}_n$ die Form

$$\hat{F}_n(t) = \sum_{i=1}^{n} W_{ni} \; 1_{\{Z_{i:n} \leq t\}} \qquad \text{für jedes } t \geq 0 \tag{126}$$

Anders als im Fall vollständig beobachtbarer Daten, wo die empirische Verteilungsfunktion jedem Datum das Gewicht $1/n$ zuteilt, sind die Gewichte W_{ni} offenbar von der Zensierung der Ordnungsstatistiken $Z_{1:n}$ bis $Z_{i:n}$ abhängig und damit insbesondere zufällig. Hinzukommt, dass die W_{ni} *nicht* unabhängig sind.
Bemerkenswert ist ferner, dass ein Datum $Z_{i:n}$, wenn es zensiert ist, d. h. wenn $\delta_{[i:n]} = 0$ gilt, unter $\hat{F}_n$ die empirische Masse 0 zugewiesen bekommt. (Betrachte dazu (125).)

Die eigenartig erscheinenden Gewichte W_{ni} werden gut interpretierbar, wenn man sich vorstellt, dass sie gemäß des folgenden Algorithmus' zur Verteilung der Gesamtwahrscheinlichkeitsmasse 1 auf die $Z_{i:n}$ zustandekommen:

- Zunächst erhält jedes $Z_{i:n}$ die Masse $\frac{1}{n}$.
- Dann wird $Z_{1:n}$ betrachtet: Ist $\delta_{[1:n]} = 1$, wird $W_{n1} := \frac{1}{n}$ gesetzt; ist $\delta_{[1:n]} = 0$, wird $W_{n1} := 0$ gesetzt und die $Z_{1:n}$ „weggenommene" Masse zu gleichen Teilen auf die übrigen $Z_{i:n}$, also auf $Z_{2:n}, \ldots, Z_{n:n}$ verteilt. In diesem Stadium des Algorithmus' haben $Z_{2:n}, \ldots, Z_{n:n}$ demnach jeweils die Masse $\frac{1}{n} + \frac{1}{n} \cdot \frac{1}{n-1}$.
- Analog zu $Z_{1:n}$ wird nun sukzessive mit $Z_{2:n}, \ldots, Z_{n:n}$ verfahren: Einem zensierten $Z_{i:n}$ wird seine gesamte Masse weggenommen und zu gleichen Teilen auf die größeren Ordnungsstatistiken $Z_{i+1:n}, \ldots, Z_{n:n}$ verteilt.
  Ist $Z_{n:n}$ zensiert, geht seine Masse vollständig verloren, weswegen $\hat{F}_n$ in diesem Fall *nicht* auf 1 springt.

12.2.2 Eigenschaften des Kaplan-Meier-Schätzers; speziell Konfidenzintervalle

Wir geben hier einige wichtige **Eigenschaften** des KM-Schätzers an (aber ohne Beweise; einen guten Überblick und zahlreiche weitere Verweise liefert [73, Stute (2004)]): $\hat{F}_n$ ist ...

- ... eine Sprungfunktion mit *zufälligen* Sprunghöhen an den Beobachtungen Z_i, wobei an zensierten Z_i keine Sprünge stattfinden;
- ... eine Subverteilungsfunktion, wenn die größte Ordnungsstatistik $Z_{n:n}$ zensiert ist;
- ... <u>nicht</u> erwartungstreu, sondern nach unten verzerrt (was gewissermaßen der Preis ist, der für den Informationsverlust durch Zensierung zu zahlen ist): $\mathbb{E}[\hat{F}_n(t)] \leq F(t)$;
- ... stark konsistent (punktweise und gleichmäßig auf Kompakta links von $\tau_H := \inf\{x : H(x) = 1\}$): $\hat{F}_n(t) \xrightarrow{f.s.} F(t)$ und $\sup_{t \in [0,T]} |\hat{F}_n(t) - F(t)| \xrightarrow{f.s.} 0$ für jedes feste $T < \tau_H$;
- ... punktweise asymptotisch normalverteilt gemäß

$$\hat{\alpha}_n(t) := \sqrt{n} \left(\hat{F}_n(t) - F(t) \right) \longrightarrow \mathcal{N}(0, \sigma^2(t)) \qquad \text{für } t < \tau_H,$$

wobei für die asymptotische Varianz $\sigma^2(t)$ gilt:

$$\sigma^2(t) = (1-F(t))^2 \int_0^t \frac{H^{(1)}(du)}{(1-H(u-))^2} \tag{127}$$

$$= (1-F(t))^2 \int_0^t \frac{F(du)}{(1-G(u-))(1-F(u-))^2} \tag{128}$$

- ... asymptotisch korreliert gemäß

$$\mathrm{Cov}\,(\hat{\alpha}_n(t), \hat{\alpha}_n(u)) \longrightarrow (1-F(t))(1-F(u)) \int_0^{t\wedge u} \frac{H^{(1)}(du)}{1-H(u-))^2} \tag{129}$$

für $t, u < \tau_H$.

Interpretation und Schätzung der Varianz von $F(t)$:
Die Darstellung (128) ermöglicht eine gute Interpretation der Wirkung der Zensierung auf die asymptotische Varianz an der Stelle t in Abhängigkeit von der Zensierungsverteilung G bei festem F: Je „früher" G Masse gewinnt, d. h., je stärker die Zensierung ist, desto größer ist die punktweise asymptotische Varianz des KM-Schätzers $\hat{F}_n$.

Die Gleichung (127) impliziert einen konsistenten Schätzer für die asymptotische Varianz von $\sqrt{n}(\hat{F}_n(t) - F(t))$, und zwar Greenwoods Formel:

$$\hat{\sigma}_n^2(t) := (1-\hat{F}_n(t))^2 \int_0^t \frac{H_n^{(1)}(du)}{(1-H_n(u-))^2} = (1-\hat{F}_n(t))^2 \, \frac{1}{n} \sum_{i=1}^n \frac{1\{Z_i \le t, \delta_i = 1\}}{(1-H_n(Z_i-))^2}$$

Dies wiederum legt für endliches n als Schätzer für die Varianz von $\hat{F}_n(t)$ die Größe $\hat{\sigma}_n^2(t)/n$ nahe.

Konfidenzintervalle für $F(t)$:

1. Greenwoods Formel motiviert die Konstruktion eines ersten approximativen asymptotischen $(1-\alpha)$-Konfidenzintervalls für $F(t)$:

$$\hat{F}_n(t) \pm u_{1-\alpha/2} \, \frac{\hat{\sigma}_n(t)}{\sqrt{n}}, \tag{130}$$

   wobei u_γ das γ-Quantil der Standardnormalverteilung ist.

   Bei diesem Konfidenzintervall (Konfidenzintervall) ist unschön, dass seine Obergrenze größer als 1 und seine Untergrenze kleiner als 0 werden kann! Eine Ad-hoc-Lösung für dieses Problem ist die Beschränkung („Truncierung") der Intervalle nach oben durch 1 und nach unten durch 0.

2. Eine weitere mögliche „Lösung" wird in folgendem Vorgehen gesehen:
   Es werden punktweise Konfidenzintervalle für die kumulative Hazardfunktion Λ ermittelt und diese durch geeignete Transformationen in punktweise Konfidenzintervalle für F überführt. Als Motivation dient dabei die *nur im stetigen (!) Fall* gültige Beziehung $\ln(1-F(t)) = -\Lambda(t)$ (siehe bei (117)) und die Tatsache, dass auch die empirische kumulative Hazardfunktion Λ_n punktweise asymptotisch normalverteilt ist: Man kann nämlich zeigen, dass gilt:

$$\sqrt{n}\,(\Lambda_n(t) - \Lambda(t)) \longrightarrow \mathcal{N}\left(0, \gamma^2(t)\right) \qquad \text{mit} \qquad \gamma^2(t) = \int_0^t \frac{H^{(1)}(du)}{(1-H(u-))^2}, \tag{131}$$

   was ein approximatives asymptotisches $(1-\alpha)$-Konfidenzintervall für $-\Lambda(t)$ liefert, nämlich

$$-\Lambda_n(t) \pm u_{1-\alpha/2} \, \frac{\hat{\gamma}_n(t)}{\sqrt{n}} \qquad \text{mit} \qquad \hat{\gamma}_n^2(t) := \int_0^t \frac{H_n^{(1)}(du)}{(1-H_n(u-))^2}$$

Durch Exponentiation erhält man daraus ein Konfidenzintervall für $\exp\{-\Lambda(t)\}$. Nun wird zur Begründung von $-\Lambda_n(t) \approx \ln(1-\hat{F}_n(t))$ die für $|x| < 1$ geltende Approximation $\ln(1-x) \approx -x$ (mit $x = \Lambda_n\{Z_{i:n}\}$) zusammen mit (122) verwendet und schließlich

$$\exp\left\{\ln\left(1-\hat{F}_n(t)\right) \pm u_{1-\alpha/2}\,\frac{\hat{\gamma}_n(t)}{\sqrt{n}}\right\} \tag{132}$$

als ein zweites approximatives asymptotisches $(1-\alpha)$-Konfidenzintervall für $1-F(t)$ vorgeschlagen. Es hat gegenüber (130) den Vorteil, stets eine nicht-negative Untergrenze zu haben, wenngleich die Obergrenze immer noch größer als 1 werden kann.

3. Als die dritte Vorgehensweise wird vorgeschlagen, die asymptotische Normalität von $\ln\Lambda_n(t)$ aus derjenigen von $\Lambda_n(t)$ in (131) mit der δ-Methode herzuleiten und die zugehörige asymptotische Varianz zu schätzen: Wegen

$$\sqrt{n}\,(\ln\Lambda_n(t) - \ln\Lambda(t)) \longrightarrow \mathcal{N}\left(0,\tau^2(t)\right) \quad \text{mit} \quad \tau^2(t) = \frac{\gamma^2(t)}{\Lambda^2(t)} \quad \text{und} \quad \hat{\tau}_n^2(t) := \frac{\hat{\gamma}_n^2(t)}{\Lambda_n^2(t)}$$

begründet $\ln\Lambda_n(t) \pm u_{1-\alpha/2}\,\hat{\tau}_n(t)/\sqrt{n}$ ein approximatives asymptotisches $(1-\alpha)$-Konfidenzintervall für $\ln\Lambda(t)$. Exponentiation liefert daraus ein Konfidenzintervall für $\Lambda(t)$ und für $\exp\{-\Lambda(t)\}$ ist seinerseits $\exp\left[-\exp\left\{\ln\Lambda_n(t) \pm u_{1-\alpha/2}\,\hat{\tau}_n(t)/\sqrt{n}\right\}\right]$ ein Konfidenzintervall. Wie in der Herleitung von (132) wird $\Lambda_n(t)$ durch $-\ln(1-\hat{F}_n(t))$ approximiert und nun

$$\exp\left[-\exp\left\{\ln\left(-\ln\left(1-\hat{F}_n(t)\right)\right) \pm u_{1-\alpha/2}\,\frac{\hat{\tau}_n(t)}{\sqrt{n}}\right\}\right] \tag{133}$$

als ein drittes approximatives asymptotisches $(1-\alpha)$-Konfidenzintervall für $1-F(t)$ vorgeschlagen. Dies nun hat stets eine Untergrenze größer als 0 und eine Obergrenze kleiner als 1.

12.2.3 Schätzung der mittleren Survival-Zeit

Als Schätzer für den Erwartungswert $\mathbb{E}[X] = \int_0^\infty x F(dx)$, also für die mittlere Lebensdauer, liegt $\int_0^\infty x\hat{F}_n(dx)$ nahe. Dieser ist für endliches n jedoch nach unten verzerrt und hat demnach einen negativen Bias:

$$\text{Bias}\left[\int_0^\infty x\hat{F}_n(dx)\right] \equiv \mathbb{E}\left[\int_0^\infty x\hat{F}_n(dx)\right] - \mathbb{E}[X] \le 0$$

Dieser Bias verschwindet zwar für $n \to \infty$, d. h., der Schätzer ist immerhin asymptotisch unverzerrt, kann aber im endlichen Fall erheblich sein. Dennoch wird er vielerorts verwendet (obwohl es inzwischen einen trickreich modifizierten Schätzer mit deutlich geringerem Bias gibt; für Details und weitere Verweise siehe [73, Stute (2004)]).

Um für die mittlere Lebensdauer z. B. ein (approximatives) asymptotisches Konfidenzintervall auf Basis von $\int_0^\infty x\hat{F}_n(dx)$ angeben zu können, wird auch seine asymptotische Varianz benötigt (und ein Schätzer für sie). Da $\hat{F}_n(x)$ auf $\mathbb{R}^+$ konzentriert ist, gilt

$$\int_0^\infty x\hat{F}_n(dx) \approx \int_0^\infty (1-\hat{F}_n(x))\,dx$$

(mit „=“ nur, falls die größte Beobachtung $Z_{n:n}$ nicht zensiert ist). Hieraus kann mit Hilfe von (127) und (129) abgeleitet werden, dass im Stetigkeitsfall für $n \to \infty$

$$n \cdot \text{Var}\left(\int_0^\infty x\hat{F}_n(dx)\right) \longrightarrow \int_0^\infty \left(\int_v^\infty (1-F(v))\,dv\right)^2 \frac{H^{(1)}(dv)}{(1-H(v))^2} \tag{134}$$

Die rechte Seite der obigen Gleichung enthält durch “plug-in” schätzbare Größen, sodass ein Schätzer für die asymptotische Varianz auf der Hand liegt.

12.2.4 Der Kaplan-Meier-Schätzer in R: `survfit()`

Doch nun von der Theorie zur Praxis in **R**: Die Funktion `survfit()` des **R**-Paketes `survival` berechnet für einen Datensatz von beobachteten Überlebensdauern Z_i und zugehörigen Zensierungsindikatoren δ_i den KM-Schätzer $1 - \hat{F}_n$ für die Survival-Funktion $1 - F$ und weitere damit in Zusammenhang stehende Größen. `survfit()` wird allerdings nicht „direkt" auf die Vektoren der Z_i und δ_i angewendet, sondern die Z_i und δ_i müssen zunächst geeignet in einer Formel „zusammengefasst" werden, was mit Hilfe einer Funktion `Surv()` geschieht. Das Resultat von `survfit()`, genauer von ihrer `formula`-Methode, ist ein `survfit`-Objekt, für das z. B. die `survfit`-Methode der Funktion `summary()` einige wichtige Ergebnisse des Fits liefert und z. B. die `survfit`-Methode der Funktion `plot()` in der Lage ist, die geschätzte Survival-Kurve samt punktweisen (approximativen asymptotischen) Konfidenzintervallen zu zeichnen.

Anhand des folgenden **Beispieldatensatzes** aus [22, Collett (2015)] (bzw. den Vorgängerversionen von 1996 und 2003), der aus beobachteten Überlebensdauern Z_i (in Monaten) und den dazugehörigen Zensierungsindikatoren δ_i von $n = 48$ Patienten und Patientinnen besteht, die an einem *multiplen Myelom* litten, soll die Wirkung der genannten Funktionen veranschaulicht werden.

Nr. i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Z_i	13	52	6	40	10	7	66	10	10	14	16	4	65	5	11	10
δ_i	1	0	1	1	1	0	1	0	1	1	1	1	1	1	0	1
Nr. i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Z_i	15	5	76	56	88	24	51	4	40	8	18	5	16	50	40	1
δ_i	0	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1
Nr. i	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Z_i	36	5	10	91	18	1	18	6	1	23	15	18	12	12	17	3
δ_i	1	1	1	1	0	1	0	1	1	1	1	1	0	1	1	0

Angenommen, die Daten liegen in einem Data Frame vor und das Paket `survival` wird in den Suchpfad eingehängt:

```
> myeloma <- data.frame( Z =     c( 13, 52, 6, ...., 12, 17, 3),
+                        delta = c(  1,  0, 1, ....,  1,  1, 0))
> library( survival)
```

Dann berechnet der folgende Aufruf den KM-Schätzer, speichert verschiedene Resultate hierzu im `survfit`-Objekt `sf` und liefert die Ausgabe einiger dieser Resultate (die weiter unten erläutert werden):

```
> (sf <- survfit( Surv( Z, delta) ~ 1, data = myeloma))
Call: survfit(formula = Surv(Z, delta) ~ 1, data = myeloma)

 records   n.max n.start  events  median 0.95LCL 0.95UCL
      48      48      48      36      17      13      40
```

Berechnung und Ausgabe eines Schätzers für die mittlere Survival-Zeit erreicht man über die `survfit`-Methode von `print()` bei Spezifizierung ihres Argumentes `rmean` (wie "restricted mean"):

```
> print( sf, rmean = 60)   # "common", "individual", (a numeric constant)
Call: survfit(formula = Surv(Z, delta) ~ 1, data = myeloma)

  records  n.max  n.start  events  *rmean *se(rmean)  median  0.95LCL  0.95UCL
    48.00  48.00    48.00   36.00   26.87       3.39   17.00    13.00    40.00
  * restricted mean with upper limit =  60
```

Darin ist

- `records` der Stichprobenumfang n;
- `n.max` die maximale Anzahl an Untersuchungseinheiten, die während der Laufzeit in der „Risikomenge" der waren (im "random-right-censorship"-Modell stets n);
- `n.start` die Anzahl an Untersuchungseinheiten, die zu Beginn der Laufzeit in der „Risikomenge" der waren (im "random-right-censorship"-Modell stets n);
- `events` die Anzahl der *nicht* zensierten Beobachtungen;
- `*rmean` der „Schätzer" $\int_0^T x\hat{F}_n(dx)$ für die mittlere Lebensdauer $\int_0^\infty xF(dx)$. Dabei kann die Einschränkung des Integrationsbereichs (Grund für Bezeichnung "restricted mean") bis zum "upper limit" T verschieden gewählt werden: Wenn `rmean` eine numerische Konstante zugewiesen wird, ist T gleich deren Wert. Wird im - hier vorliegenden - Einstichprobenfall `rmean = "common"` oder `rmean = "individual"` gewählt, wird $T = Z_{n:n}$ gesetzt, ist also datenabhängig! (Für einen Gegenvorschlag zu diesen fragwürdigen Modifikationen siehe [73, Stute (2004)].)
- `*se(mean)` der aus (134) durch "plug-in" der empirischen Größen erhaltene Schätzer der Standardabweichung des obigen Schätzers der "restricted" mittleren Lebensdauer;
- `median` der Stichprobenmedian, definiert als $\hat{F}_n^{-1}(1/2)$;
- `0.95LCL` und `0.95UCL` die untere bzw. obere 95 %-Konfidenzintervall-Grenze für den Median, definiert durch die Schnittpunkte der 95 %-Konfidenzintervall-Grenzen (vermutlich aus (132)) für $\hat{F}_n$ mit dem 0.5-Niveau.

Die Funktion `summary()` liefert noch wesentlich ausführlichere Informationen über den Survival-Schätzer $1 - \hat{F}_n$:

```
> summary( sf)
Call: survfit(formula = Surv(Z, delta) ~ 1, data = myeloma)

 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    1     48       3   0.9375  0.0349       0.8715        1.000
    4     44       2   0.8949  0.0445       0.8118        0.986
    5     42       4   0.8097  0.0571       0.7051        0.930
....
   88      2       1   0.0674  0.0576       0.0126        0.359
   91      1       1   0.0000     NaN           NA           NA
```

Die Spalten dieser Ergebnistabelle enthalten das Folgende:

`time`	Die beobachteten **unzensierten** Überlebenszeiten $Z_{i:n}$ ohne Duplikate (d. h., bei Bindungen oder Zensierungen in den Z_i sind es weniger als n).
`n.risk`	Anzahl der kurz vor der jeweiligen (unzensierten) Zeit $Z_{i:n}$ noch in der Studie befindlichen Individuen (= „Risikomenge"), also $n(1-H_n(Z_{i:n}-))$.
`n.event`	Anzahl der zur jeweiligen (unzensierten) Zeit $Z_{i:n}$ beobachteten Tode (also unzensierten Z_i), also $nH_n^{(1)}\{Z_{i:n}\}$.
`survival`	Wert des Survival-Schätzers zur Zeit $Z_{i:n}$, also $1 - \hat{F}_n(Z_{i:n})$.
`std.err`	Schätzwert der Standardabweichung von $1 - \hat{F}_n(Z_{i:n})$ gemäß Greenwoods Formel (130).
`lower 95% CI`	Untergrenze des approximativen asymptotischen punktweisen 95 %-Konfidenzintervalls in (132) für $1 - F$ an der Stelle $Z_{i:n}$.
`upper 95% CI`	Obergrenze dieses Konfidenzintervalls.

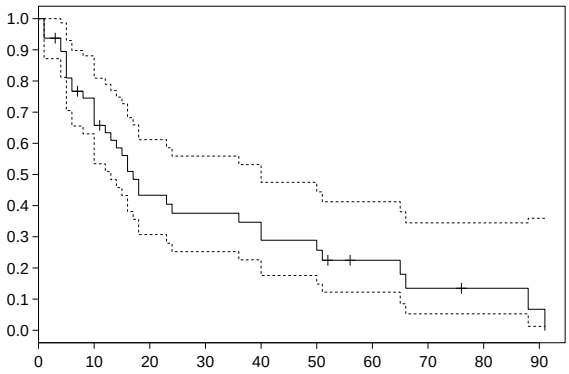
Die Einträge `NaN` und `NA` in der letzten Zeile bei `std.err` bzw. bei `lower 95% CI` und `upper 95% CI` resultieren aus der Unmöglichkeit für $1 - \hat{F}_n(t) = 0$ eine Standardabweichung und damit ein Konfidenzintervall zu berechnen.

Da sowohl ein `survfit`-Objekt (wie hier `sf`) als auch ein `summary.survfit`-Objekt (wie hier das Resultat von `summary( sf)`) speziellen Listen sind, kann auf die Komponenten auch durch Listenkomponentenindizierung (wie z. B. `sf$`*komponente*) zugegriffen werden, sodass weitere Berechnungen möglich sind. Für zusätzliche Informationen und Details hierzu siehe die Online-Hilfe zu `summary.object` und zu `summary.survfit()`.

Die Funktion `plot()` hat die Methode `plot.survfit()`, die ein `survfit`-Objekt direkt verarbeiten kann: Die Wirkung von

```
> plot( sf)
```

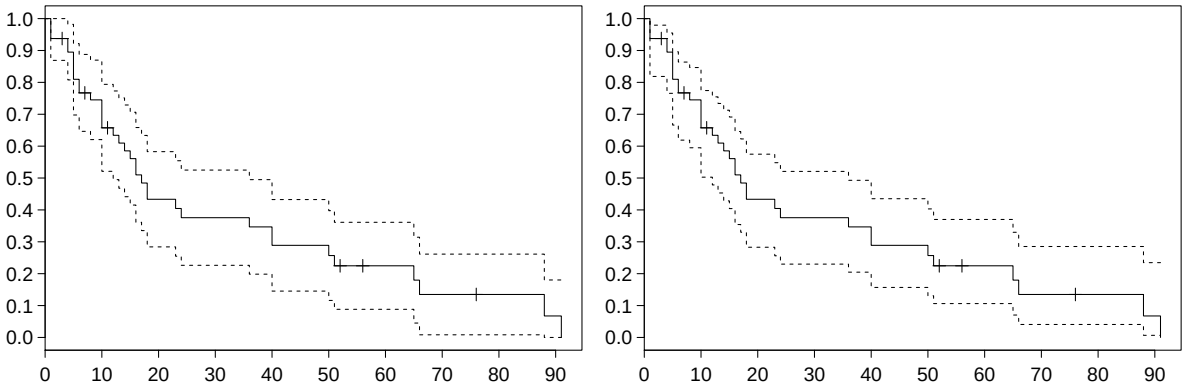
ist aus dem unten stehenden Plot zu ersehen. Die durchgezogene „Treppenfunktion" darin ist der Verlauf des KM-Schätzer $1 - \hat{F}_n$. Die zwei unterbrochenen Linien stellen die Ober- bzw. Untergrenze der approximativen asymptotischen *punktweisen* 95 %-Konfidenzintervalle für $1 - F$ gemäß (132) dar. An den Stellen Z_i, die eine zensierte Beobachtung sind, markiert ein kleines „+" in der Treppenfunktion $1 - \hat{F}_n$ den „fehlenden" Sprung.



Auch die beiden anderen Typen von approximativen asymptotischen *punktweisen* Konfidenzintervallen sind in der Funktion `survfit` implementiert. Sie besitzt dazu das Argument `conf.type`, das `"plain"`, `"log"` (Voreinstellung) sowie `"log-log"` als mögliche Werte akzeptiert und dementsprechend die Konfidenzintervalle gemäß (130), (132) (Voreinstellung) bzw. (133) liefert:

```
> sf2 <- survfit( Surv( Z, delta) ~ 1, data = myeloma, conf.type = "plain")
> sf3 <- survfit( Surv( Z, delta) ~ 1, data = myeloma, conf.type = "log-log")
```

Links ist der Plot des KM-Schätzers mit approximativen asymptotischen *punktweisen* Konfidenzintervallen vom Typ `"plain"`, rechts der mit den Konfidenzintervallen vom Typ `"log-log"`:



Bemerkung: Es gibt zwei Alternativen zum KM-Schätzer, und zwar den Fleming-Harrington-Schätzer und den (modifizierten) Nelson-Schätzer für F. Sie beruhen beide auf dem Nelson-Aalen-Schätzer (121) der kumulativen Hazardfunktion und der (nur für stetige VFn!) gültigen Beziehung $1 - F(t) = \exp\{-\Lambda(t)\}$ (siehe hierzu nochmal (117)). Beide sind in `survfit()` realisiert und mit dem Argument `type` zu erhalten:

- Der Fleming-Harrington-Schätzer setzt (unter Ignoranz der Unstetigkeit von Λ_n) einfach

  $$1 - F_{FH}(t) := \exp\{-\Lambda_n(t)\}$$

  und wird von `survfit()` bei Angabe von `type = "fleming-harrington"` berechnet.

- Der (modifizierte) Nelson-Schätzer basiert auf der gleichen Formel, allerdings wird Λ_n zuvor gemäß eines von Nelson (bzw. auch von Fleming und Harrington) gemachten Vorschlags modifiziert, um einen stark Bias-generierenden Einfluss von Bindungen in den Daten zu reduzieren. Diese Version erhält man von `survfit()` bei Angabe des Arguments `type = "fh2"`.

12.2.5 Der Kaplan-Meier-Schätzer für gruppierte Survival-Daten

Hat man stratifizierte (= gruppierte) Survival-Daten, wie z. B. Daten für unterschiedlich behandelte PatientInnen-Gruppen, so kann diese Gruppierung im ersten Argument von `survfit()` unter Verwendung der (aus den linearen Modellen bzw. der Varianzanalyse, speziell z. B. Abschnitt 11.1) bekannten Formelsyntax mittels Faktoren modelliert werden.

Die Geschlechtszugehörigkeit in obigem Datensatz ist eine solche stratifizierende Variable und sei in dem Faktor `MW` durch `m` = männlich und `w` = weiblich kodiert:

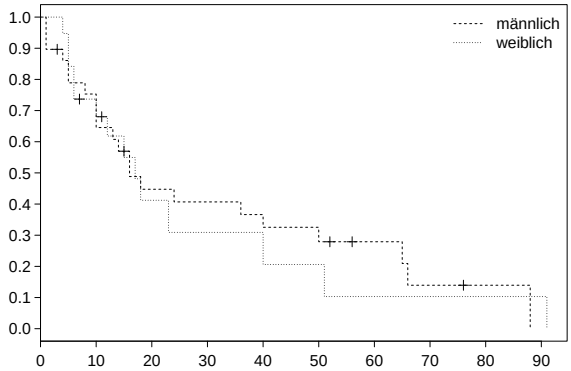
```
> myeloma$MW <- factor( c( "m", "m", "w", "m", "m", "m", "w", "w", "w",
+ .... "w", "w", "w", "m"))
```

Die Bestimmung des stratifizierten KM-Schätzers, die unterschiedlich detaillierte Ausgabe der Resultate (wie sie analog zu den Ausgaben für einen nicht stratifizierten KM-Schätzer auf den vorherigen zwei Seiten zu sehen sind) und die Erzeugung eines Plots der geschätzten Survival-Kurven kann wie folgt bewerkstelligt werden:

```
> (sf4 <- survfit( Surv( Z, delta) ~ MW, data = myeloma))
Call: survfit(formula = Surv(Z, delta) ~ MW, data = myeloma)

      records n.max n.start events median 0.95LCL 0.95UCL
 MW=m      29    29      29     22     16      10      65
 MW=w      19    19      19     14     17      10      NA
```

```
> summary( sf4)
Call: survfit(formula = Surv(Z, delta) ~ MW, data = myeloma)

                MW=m
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    1     29       3    0.897  0.0566       0.7923        1.000
    4     25       1    0.861  0.0647       0.7428        0.997
....
   66      3       1    0.139  0.0831       0.0434        0.448
   88      1       1    0.000     NaN           NA           NA

                MW=w
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    4     19       1    0.947  0.0512       0.8521        1.000
    5     18       2    0.842  0.0837       0.6931        1.000
....
   51      2       1    0.103  0.0944       0.0171        0.621
   91      1       1    0.000     NaN           NA           NA

> plot( sf4, lty = 2:3)
> legend( "topright", c( "maennlich", "weiblich"), lty = 2:3, bty = "n")
```

Die im Plot gezeigten beiden Treppenfunktionen stellen die Verläufe der KM-Schätzer (ohne Konfidenzintervalle) in den zwei Strata dar, hier also in den jeweiligen Geschlechtsgruppen.

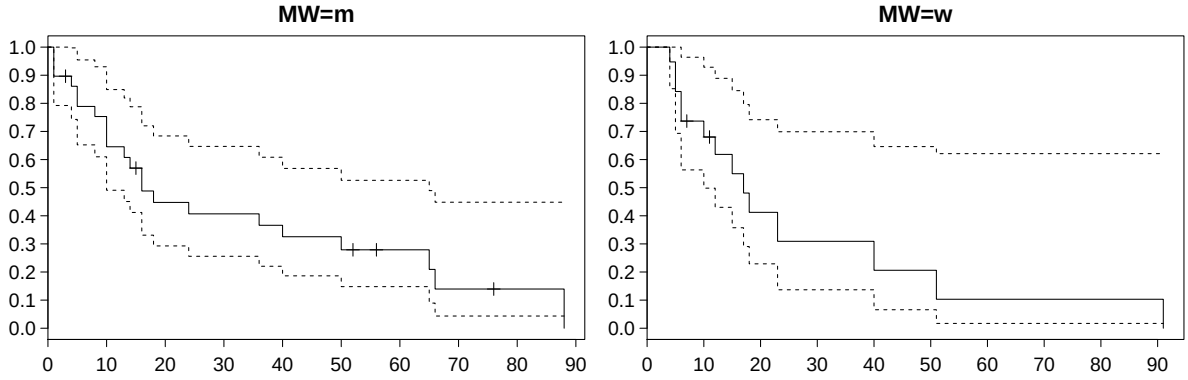


Durch die Verwendung des Parameters `lty` in `plot()` wird für jede der Kurven ein eigener Linientyp verwendet. Die korrekte Zuordnung der Linientypen zu den beiden Gruppen ist in `legend()` aber nur möglich, wenn man weiß, in welcher Reihenfolge die Strata im Resultat von `survfit()` abgelegt worden sind. Dies ist aus der Ausgabe des `survfit`-Objektes ersichtlich: Das erste Stratum ist `MW = m` und das zweite `MW = w`. (Sie sind also in der Reihenfolge der Faktorlevel-Ordnung sortiert.)

Auf die gemäß der Stratifizierung separaten KM-Schätzer in einem `survfit`-Objekt kann mittels einer suggestiven Indizierung zugegriffen werden: `sf4[ 1]` z. B. ist der KM-Schätzer für das erste Stratum und entsprechend wird auf die Resultate der anderen Strata zugegriffen. Dies ermöglicht eine detailliertere Darstellung oder die weitere Verarbeitung der Strata, wie z. B. das getrennte Plotten der Verläufe der beiden KM-Schätzer in `sf4`: In dem folgenden Beispiel wird auf jedes Stratum separat zugegriffen und durch `conf.int = TRUE` erreicht, dass die Kurven mit (approximativen asymptotischen *punktweisen*) Konfidenzintervallen versehen werden (siehe Plot auf der nächsten Seite oben). Durch `main = names( sf4$strata[ i])` erhält der jeweilige Plot als Überschrift den Inhalt der Komponente `strata` des `survfit`-Objektes `sf4[ i]`:

```
> plot( sf4[ 1], conf.int = TRUE, main = names( sf4$strata)[ 1])
```

```
> plot( sf4[ 2], conf.int = TRUE, main = names( sf4$strata)[ 2])
```



12.2.6 Vergleich gruppierter Survival-Daten und die Funktion `survdiff()`

Für Survival-Daten, die bzgl. einer Covariablen (wie Geschlecht oder Therapieform u. Ä.) gruppiert werden können, ist es häufig von Interesse herauszufinden, ob sich die den Gruppen zugehörigen Survival-Funktionen (bzw. die Lebensdauerverteilungen) signifikant voneinander unterscheiden. Das **R**-Paket `survival` stellt hierzu die Funktion `survdiff()` zur Verfügung, mit der mehrere (verallgemeinerte lineare) Rangtests für zensierte Daten durchführbar sind. `survdiff()` erwartet in der Grundversion dieselben zwei Argumente wie die Funktion `survfit()` für gruppierte Daten, doch dazu später mehr.

Zur theoretischen Motivation der Tests beschränken wir uns auf den Fall zweier Gruppen von Survival-Daten. Es seien

$$\begin{aligned}
&X_1^{(1)}, \ldots, X_n^{(1)} \text{ i. i. d. } \sim F^{(1)}, \quad Y_1^{(1)}, \ldots, Y_n^{(1)} \text{ i. i. d. } \sim G^{(1)},\\
&X_1^{(2)}, \ldots, X_m^{(2)} \text{ i. i. d. } \sim F^{(2)}, \quad Y_1^{(2)}, \ldots, Y_m^{(2)} \text{ i. i. d. } \sim G^{(2)}\\
\text{und} \quad &Z_i^{(j)} := \min\{X_i^{(j)}, Y_i^{(j)}\} \quad \text{sowie} \quad \delta_i^{(j)} := 1_{\{X_i^{(j)} \le Y_i^{(j)}\}}\\
&\text{für } i = 1, \ldots, n, \text{ falls } j = 1, \quad \text{und} \quad \text{für } i = 1, \ldots, m, \text{ falls } j = 2.
\end{aligned}$$

Zu testen ist die Hypothese

$$H_0 : F^{(1)} = F^{(2)} \quad \text{gegen} \quad H_1 : F^{(1)} \neq F^{(2)}.$$

Wir nehmen an, dass unter den $Z_i^{(j)}$ **keine Bindungen** auftreten, und mit $N := n + m$ seien

$Z_1, \ldots, Z_N$	das gepoolte Sample aller $Z_i^{(j)}$,
$\delta_1, \ldots, \delta_N$	die zugehörigen Zensierungsindikatoren,
$\gamma_1, \ldots, \gamma_N$	die Gruppenzugehörigkeitsindikatoren, also

$$\gamma_i := \begin{cases} 1, & \text{falls } (Z_i, \delta_i) \text{ aus Gruppe 1;} \\ 0, & \text{sonst.} \end{cases}$$

Wie üblich seien ferner $Z_{1:N}, \ldots, Z_{N:N}$ die OSn der Z_i und $\delta_{[1:N]}, \ldots, \delta_{[N:N]}$ sowie $\gamma_{[1:N]}, \ldots, \gamma_{[N:N]}$ die Konkomitanten der δ_i bzw. γ_i. Außerdem sei $w_{N1}, \ldots, w_{NN}$ eine Folge von Gewichten. Zur Abkürzung setzen wir

$$n_i := \sum_{j=1}^{N} 1_{\{Z_j \ge Z_{i:N}\}} = \begin{cases} \text{Anzahl der Individuen, die zum Zeitpunkt } Z_{i:N} \text{ noch} \\ \text{in der „Risikogruppe“ sind, d. h. mindestens bis } Z_{i:N} \\ \text{überlebt haben } (= N - i + 1); \end{cases}$$

$$n_i^{(1)} := \sum_{j=1}^{N} \gamma_j \, 1_{\{Z_j \ge Z_{i:N}\}} = \begin{cases} \text{Anzahl der Individuen in Gruppe 1, die zum Zeitpunkt} \\ Z_{i:N} \text{ noch in der „Risikomenge“ sind.} \end{cases}$$

Als Teststatistik dient

$$S_N := \sum_{i=1}^{N} w_{Ni}\ \delta_{[i:N]} \left(\gamma_{[i:N]} - \frac{n_i^{(1)}}{n_i} \right), \tag{135}$$

die, wenn entsprechend standardisiert, asymptotisch standardnormalverteilt ist, wie man zeigen kann (wenn $N \to \infty$ so, dass $n/N \to \lambda$, wobei $0 < \lambda < 1$).

Um eine Vorstellung davon zu bekommen, was S_N misst, betrachten wir den Fall $w_{Ni} \equiv 1$ für alle $i = 1, \ldots, N$ und stellen fest, dass aufgrund des Faktors $\delta_{[i:N]}$ in jedem Summanden von S_N nur über die unzensierten $Z_{i:N}$, d. h. die beobachteten wahren Lebensdauern unter allen $Z_{i:N}$ summiert wird. Zur Vereinfachung seien daher

$$\begin{aligned}
\{T_1, \ldots, T_r\} &:= \{Z_i^{(1)} : \delta_i^{(1)} = 1\} \cup \{Z_i^{(2)} : \delta_i^{(2)} = 1\} \\
&= \text{die Menge aller beobachteten wahren Lebensdauern im gepoolten} \\
&\quad\ \text{Sample, also aller unzensierten } Z_i^{(1)} \text{ und } Z_i^{(2)}, \\
\tilde{\gamma}_l &\quad\ \text{der Gruppenzugehörigkeitsindikator für die } T_l, \\
\tilde{n}_l &\quad\ \text{die Größe der Risikogruppe zum Zeitpunkt } T_l \text{ und} \\
\tilde{n}_l^{(1)} &\quad\ \text{die Größe der Risikogruppe zum Zeitpunkt } T_l \text{ in Gruppe 1}
\end{aligned}$$

jeweils für $l = 1, \ldots, r$. Wir halten fest, dass sich S_N aus (135) dann vereinfacht zu

$$S_N = \sum_{l=1}^{r} \tilde{\gamma}_l - \frac{\tilde{n}_l^{(1)}}{\tilde{n}_l}\ . \tag{136}$$

Zur heuristischen Motivation von S_N erstellen wir für jede beobachtete wahre Lebensdauer T_l die folgende (2×2)-Tafel:

Gruppe	Anzahl der Tode z. Z. T_l	Anzahl der nach T_l Überlebenden	Größe der Risikogruppe kurz vor T_l
1	$d_l^{(1)}$	$\tilde{n}_l^{(1)} - d_l^{(1)}$	$\tilde{n}_l^{(1)}$
2	$d_l^{(2)}$	$\tilde{n}_l^{(2)} - d_l^{(2)}$	$\tilde{n}_l^{(2)}$
Summe	d_l	$\tilde{n}_l - d_l$	$\tilde{n}_l$

Unter der Annahme (!), dass die Marginalhäufigkeiten d_l, $\tilde{n}_l^{(1)}$ und $\tilde{n}_l$ **fest** sind, ist die Zellenbelegung der Tafel schon allein durch $d_l^{(1)}$ festgelegt, sodass es reicht, $d_l^{(1)}$ zu betrachten. Des Weiteren sind unter H_0 die Lebensdauerverteilungen in Gruppe 1 und 2 gleich. Damit ist insbesondere auch für alle Individuen in der gepoolten Risikogruppe die Wahrscheinlichkeit, zur Zeit T_l zu sterben, gleich groß, unabhängig von ihrer Gruppenzugehörigkeit. D. h., die ZV $d_l^{(1)}$ hat eine *hypergeometrische Verteilung* (zum Stichprobenumfang d_l und „Erfolgsparameter" $\tilde{n}_l^{(1)}/\tilde{n}_l$) und somit den Erwartungswert

$$e_l^{(1)} := \mathbb{E}\left[d_l^{(1)}\right] = \frac{d_l \tilde{n}_l^{(1)}}{\tilde{n}_l}$$

In unserem Fall stetiger Verteilungen, also ohne Bindungen, ist die zum Zeitpunkt T_l beobachtete Anzahl $d_l^{(1)}$ wahrer Lebensdauern in Gruppe 1 entweder 1 oder 0. Demzufolge ist die Anzahl der an T_l insgesamt beobachteten wahren Lebensdauern $d_l \equiv 1$. Also ist $d_l^{(1)} = \tilde{\gamma}_l$ und $e_l^{(1)} = \tilde{n}_l^{(1)}/\tilde{n}_l$, sodass ein Summand $\tilde{\gamma}_l - \tilde{n}_l^{(1)}/\tilde{n}_l$ von S_N in (136) die beobachtete und die erwartete Häufigkeit

zur Zeit T_l miteinander vergleicht.

Durch Summation über alle Lebensdauern $T_1, \ldots, T_r$ erhalten wir

$$\begin{aligned} S_N &= \sum_{l=1}^{r} d_l^{(1)} - e_l^{(1)} = \sum_{l=1}^{r} d_l^{(1)} - \sum_{l=1}^{r} e_l^{(1)} \\ &= \begin{cases} \text{Differenz der in Gruppe 1 insgesamt beobachte-} \\ \text{ten und insgesamt erwarteten Tode.} \end{cases} \end{aligned}$$

Aufgrund der Unabhängigkeit der wahren Lebensdauern T_l sind die (2×2)-Tafeln und damit die $d_l^{(1)}$ unabhängig voneinander und es ist

$$\sigma_{S_N}^2 := \mathrm{Var}(S_N) = \sum_{l=1}^{r} \mathrm{Var}(d_l^{(1)}) = \sum_{l=1}^{r} \frac{\tilde{n}_l^{(1)} \tilde{n}_l^{(2)} d_l (\tilde{n}_l - d_l)}{\tilde{n}_l^2 (\tilde{n}_l - 1)}$$

(gemäß der Varianzformel für die hypergeometrische Verteilung). Damit haben wir heuristisch motiviert, dass

$$\frac{S_N}{\sigma_{S_N}} \sim \mathcal{N}(0,1) \qquad \text{bzw.} \qquad \frac{S_N^2}{\sigma_{S_N}^2} \sim \chi_1^2 \qquad \text{approximativ unter } H_0.$$

Angeblich gilt dies auch im allgemeinen Fall nicht notwendig stetiger Verteilungen.

Je nach Wahl der Gewichte w_{Ni} erhält man spezielle Tests:

Gewichte w_{Ni}	Name des Tests
1	Log-Rank-Test (= Mantel-Haenszel- oder Mantel-Cox- oder Peto-Mantel-Haenszel-Test)
$\frac{N-i+1}{N} = \frac{n_i}{N}$	Wilcoxon-Test
$\frac{i}{N}$	Gehans Wilcoxon-Test
$1 - \hat{F}_N(Z_{i:N})$	Peto-Peto-Modifikation des Wilcoxon-Tests (mit dem Kaplan-Meier-Schätzer $\hat{F}_N$ des gepoolten Samples).

(Siehe auch [52, Klein & Moeschberger (2003)], Abschnitt 7.3, sowie das R-Paket `survival`.)

In **R** ist in `survdiff()`

$$w_{Ni} = (1 - \hat{F}_N(Z_{i:N}))^{\rho} \quad \text{für } \rho \in \mathbb{R}$$

implementiert, wobei ρ im Aufruf von `survdiff()` über den Parameter `rho` gesteuert wird. Die Voreinstellung ist $\rho = 0$, d. h., es wird standardmäßig der Log-Rank-Test duchgeführt, der im Fall der Proportional-Hazards-Alternative (siehe hierzu den nächsten Abschnitt 12.3) die beste Güte hat. Für $\rho = 1$ erhalten wir die Peto-Peto-Modifikation des Wilcoxon-Tests. Allgemein gilt: Für $\rho > 0$ ist der Test empfindlich gegenüber Unterschieden zwischen den Survival-Funktionen am frühen Ende (d. h. „nahe der Null“).

Im **Beispiel** unserer Daten über die Krebspatientinnen und -patienten erhalten wir bei Gruppierung nach Männern und Frauen das Ergebnis des Log-Rank-Tests auf Unterschied zwischen den Gruppen wie folgt (beachte, dass dabei dieselbe Formelsyntax zum Einsatz kommt wie bei `survfit()` auf Seite 355):

```
> (sdiff <- survdiff( Surv( Z, delta) ~ MW, data = myeloma))
Call: survdiff(formula = Surv(Z, delta) ~ MW, data = myeloma)
      N Observed Expected (O-E)^2/E (O-E)^2/V
MW=m 29       22     22.5     0.012    0.0358
MW=w 19       14     13.5     0.020    0.0358

 Chisq= 0  on 1 degrees of freedom, p= 0.85
```

Bemerkung: Es ist zu bemängeln, dass **R** hier per Voreinstellung rabiat rundet. Daher ist es ratsam, obigen Befehl in eine `print`-Anweisung mit explizit angegebener, hinreichend hoher Stellenzahl zu packen, wie z. B. in

```
> print( sdiff, digits = 6)
....
      N Observed Expected (O-E)^2/E (O-E)^2/V
MW=m 29       22  22.5196 0.0119871 0.0358454
MW=w 19       14  13.4804 0.0200250 0.0358454

 Chisq= 0  on 1 degrees of freedom, p= 0.849835
```

Nun zur Bedeutung der Ausgabe: In Zeile `MW=m` steht in der Spalte `Observed` die Anzahl insgesamt beobachteter Tode $\sum_{l=1}^{r} d_l^{(\mathtt{m})}$ und unter `Expected` steht die Anzahl insgesamt erwarteter Tode $\sum_{l=1}^{r} e_l^{(\mathtt{m})}$. Auf beide wird in den beiden rechten Spaltenüberschriften durch die Buchstaben `O` und `E` Bezug genommen, sodass in der letzten Spalte namens `(O-E)^2/V` also der Wert der Statistik $S_N^2/\sigma_{S_N}^2$ für die Gruppe `MW=m` steht.

In der Zeile `MW=w` stehen die analogen Resultate für die zweite Gruppe, und natürlich muss sich für die Teststatistik dieser Gruppe derselbe Wert wie in der ersten ergeben.

Beachte: Der Wert dieser Teststatistik wird in der Ausgabe des Resultats von `survdiff()` auch nochmal als `Chisq` präsentiert. Dort allerdings rundet **R** – trotz der `digits`-Angabe im `print`-Befehl – unverständlicherweise automatisch (und unveränderbar) auf eine (!) Nachkommastelle (gemeint ist `Chisq= 0`). Um an den exakten Wert zu kommen, muss man auf die Komponente `$chisq` des Resultats von `survdiff()` zugreifen:

```
> sdiff$chisq
[1] 0.03584544
```

Auch andere Tests (für $\rho = 1$, d. h. für die Peto-Peto-Modifikation des Wilcoxon-Tests, oder für $\rho = -1$) liefern hier kein Indiz für einen signifikanten Unterschied zwischen den Gruppen hinsichtlich ihrer Lebensdauerverteilungen (natürlich sollte *vor* Durchführung eines Tests unveränderlich feststehen, was und wie getestet wird):

```
> print( survdiff( Surv( Z, delta) ~ MW, data = myeloma, rho = 1), digits = 6)
Call: survdiff(formula = Surv(Z, delta) ~ MW, data = myeloma, rho = 1)
      N Observed Expected   (O-E)^2/E  (O-E)^2/V
MW=m 29 13.53431 13.62888 0.000656231 0.00248419
MW=w 19  8.43291  8.33834 0.001072599 0.00248419

 Chisq= 0  on 1 degrees of freedom, p= 0.960249

> print( survdiff( Surv( Z, delta) ~ MW, data = myeloma, rho = -1), digits = 6)
Call: survdiff(formula = Surv(Z, delta) ~ MW, data = myeloma, rho = -1)
      N Observed Expected (O-E)^2/E (O-E)^2/V
```

```
MW=m 29  49.3900  48.3331 0.0231122 0.0245772
MW=w 19  38.2891  39.3460 0.0283913 0.0245772

 Chisq= 0  on 1 degrees of freedom, p= 0.875425
```

Bemerkungen:

- Es lassen sich auch g (> 2) Gruppen von Survival-Daten vergleichen. Die Motivation des Tests fußt auf der Erweiterung der obigen Herleitung: Anstelle von (2×2)-Tafeln sind nun $(g \times 2)$-Tafeln zu betrachten und die schließlich erhaltene Teststatistik hat eine asymptotische χ^2-Verteilung mit $g-1$ Freiheitsgraden.

  Für jede beobachtete Lebensdauer T_l ergibt sich (in analoger Notation) die folgende $(g \times 2)$-Tafel:

Gruppe	Anzahl der Tode z. Z. T_l	Anzahl der nach T_l Überlebenden	Größe der Risikogruppe kurz vor T_l
1	$d_l^{(1)}$	$\tilde{n}_l^{(1)} - d_l^{(1)}$	$\tilde{n}_l^{(1)}$
2	$d_l^{(2)}$	$\tilde{n}_l^{(2)} - d_l^{(2)}$	$\tilde{n}_l^{(2)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
g	$d_l^{(g)}$	$\tilde{n}_l^{(g)} - d_l^{(g)}$	$\tilde{n}_l^{(g)}$
Summe	d_l	$\tilde{n}_l - d_l$	$\tilde{n}_l$

  Unter der Annahme (!), dass die Marginalhäufigkeiten $d_l, \tilde{n}_l^{(1)}, \ldots, \tilde{n}_l^{(g-1)}$ und $\tilde{n}_l$ **fest** sind, ist die Zellenbelegung der $(g \times 2)$-Tafel schon allein durch die $g-1$ Größen $d_l^{(1)}, \ldots, d_l^{(g-1)}$ festgelegt:

  Analog zum Zweigruppenfall hat der $(g-1)$-dimensionale Zufalls*vektor* $O_l := \left(d_l^{(1)}, \ldots, d_l^{(g-1)}\right)'$ der beobachteten ("observed") Häufigkeiten unter H_0 eine *multivariate hypergeometrische Verteilung* zum Stichprobenumfang d_l und zum „Erfolgsparametervektor" $\left(\tilde{n}_l^{(1)}/\tilde{n}_l, \ldots, \tilde{n}_l^{(g-1)}/\tilde{n}_l\right)'$. Der Vektor der erwarteten ("expected") Häufigkeiten $E_l := \mathbb{E}[O_l]$ lautet demnach

$$E_l \equiv \left(e_l^{(1)}, \ldots, e_l^{(g-1)}\right)' \equiv \mathbb{E}\left[\left(d_l^{(1)}, \ldots, d_l^{(g-1)}\right)'\right] = d_l \left(\frac{\tilde{n}_l^{(1)}}{\tilde{n}_l}, \ldots, \frac{\tilde{n}_l^{(g-1)}}{\tilde{n}_l}\right)'$$

  Auf Basis der Unabhängigkeit der wahren Lebensdauern T_l wird wieder argumentiert, dass die $(g \times 2)$-Tafeln und damit die $(g-1)$-dimensionalen Vektoren O_l unabhängig sind. Daher gilt:

$$\sum_{l=1}^{r} O_l - E_l \sim \mathcal{N}_{g-1}\left(0_{g-1}, \Sigma_{g-1}\right) \qquad \text{approximativ unter } H_0,$$

  wobei $\Sigma_{g-1} := \mathrm{Cov}\left(\sum_{l=1}^{r} O_l - E_l\right)$ eine $((g-1) \times (g-1))$-Kovarianzmatrix ist, die wegen der Unabhängigkeit der O_l gleich $\sum_{l=1}^{r} \mathrm{Cov}(O_l - E_l) = \sum_{l=1}^{r} \mathrm{Cov}(O_l)$ ist und somit als Summe von Kovarianzmatrizen von multivariaten hypergeometrischen Verteilungen berechnet wird.

  Schließlich erhalten wir aus all dem die unter H_0 approximativ χ^2-verteilte Teststatistik:

$$\left(\sum_{l=1}^{r} O_l - E_l\right)' \Sigma_{g-1}^{-1} \left(\sum_{l=1}^{r} O_l - E_l\right) \sim \chi^2_{g-1} \qquad \text{approximativ unter } H_0 \tag{137}$$

Obige Argumentation trifft natürlich auf jede Auswahl von $g-1$ der g Gruppen zu. Die Funktion `survdiff()` berechnet die Teststatistik auch nicht auf Basis der Gruppen 1 bis $g-1$, sondern der Gruppen 2 bis g.

Fortsetzung des **Myeloma-Beispiels**: Zusätzlich zum Geschlecht als gruppierendem Faktor in der Variablen `MW` (siehe Seite 355) ist noch ein weiterer Faktor namens `BJ` bekannt (zur dessen Bedeutung auf Seite 367 etwas gesagt wird). Ein Vergleich der durch die Kombination der beiden Faktoren definierten Gruppen wird unter Verwendung der bekannten Formelsyntax erzielt:

```
> myeloma$BJ <- factor( c( "y", "n", "y", "y", "n", "n", "y", "y",
+ .... "n", "n", "y"))
> print( sdiff <- survdiff( Surv( Z, delta) ~ MW + BJ, data = myeloma),
+  digits = 6)
Call: survdiff(formula = Surv(Z, delta) ~ MW + BJ, data = myeloma)
            N Observed Expected (O-E)^2/E (O-E)^2/V
MW=m, BJ=n 19       15 12.58518  0.463351  0.770452
MW=m, BJ=y 10        7  9.93438  0.866747  1.311558
MW=w, BJ=n 14        9  7.51889  0.291755  0.421878
MW=w, BJ=y  5        5  5.96154  0.155088  0.235099

 Chisq= 2.1  on 3 degrees of freedom, p= 0.556189
```

Wir bestätigen die Berechnung der Teststatistik `Chisq` „von Hand": Dazu ist zu wissen, dass in der Komponente `$var` eines `survdiff`-Objekts die singuläre (!) Kovarianzmatrix des g-Vektors $\left(d_l^{(1)}, \ldots, d_l^{(g)}\right)'$ steckt, aus der man durch Streichen der j-ten Zeile und Spalte die reguläre Kovarianzmatrix des $(g-1)$-Vektors $\left(d_l^{(1)}, \ldots, d_l^{(j-1)}, d_l^{(j+1)}, \ldots, d_l^{(g)}\right)'$ erhält. Diese dient dann der Umsetzung von (137).

(Beachte zuvor: Für einen n-Vektor x und eine invertierbare $(n \times n)$-Matrix A ist die Berechnung von $x'A^{-1}x$ in **R** zwar einerseits durch `t( x) %*% solve( A) %*% x` (bzw. auch ohne explizite Transponierung von `x`) zu erreichen, aber `sum( solve( A, x) * x)` liefert dasselbe, nur deutlich effizienter!)

```
> d <- sdiff$obs - sdiff$exp;                    v <- sdiff$var
> sum( solve( v[ -1, -1], d[-1]) * d[-1]);       sdiff$chisq
> [1] 2.078923
> [1] 2.078923
```

- Zur Bedeutung des Begriffs „stratifizieren": Im vorherigen Abschnitt wurde er synonym zu „gruppieren" verwendet. Häufig ist er jedoch mit einer anderen Bedeutung versehen, die wir anhand eines Beispiels beschreiben wollen:

  Angenommen, im Rahmen einer „multi-zentrischen" klinischen Studie wurden Daten zum Vergleich zweier Krebstherapien erhoben. Darunter für alle PatientInnen die Information, in welchem „Zentrum" (sprich Krankenhaus) sie behandelt wurden, und weitere Covariablen, die einen Einfluss auf die Lebensdauerverteilung haben können, wie das Alter der PatientInnen zur Zeit ihres jeweiligen Therapiebeginns.

  Wir sind aber allein an einem Therapieunterschied interessiert und nicht daran, ob der Ort der Behandlung oder das Alter der behandelten Person eine Rolle spielt. Man spricht dann davon, dass die Daten nach Kliniken (oder Alter) stratifiziert sind, und kann sich das so vorstellen, das die stratifizierende Covariable einer Art Blockbildung dient, wie sie im

einfachen randomisierten Blockexperiment durchgeführt wird. Für einen solchen Fall gibt es *stratifizierte Versionen obiger Tests*, was im Wesentlichen bedeutet, dass die Teststatistiken für jedes Stratum separat berechnet und geeignet zusammengefasst werden. (Wir gehen hier nicht näher darauf ein und verweisen z. B. auf [22, Collett (2015)], Abschnitt 2.8 (bzw. in seinen Vorgängerversionen von 1996 und 2003).)

12.3 Semi-Parametrik: Das Cox Proportional Hazards Modell

Der nicht-parametrische Kaplan-Meier-Schätzer, wie wir ihn in den beiden vorigen Abschnitten betrachtet haben, lässt die Analyse eines homogenen Samples an Survival-Daten oder den Vergleich von zwei oder mehr (in sich homogenen) Gruppen zu. Von noch größerem Interesse ist jedoch oft die *Modellierung* der Survival-Funktion in Abhängigkeit von mehreren, insbesondere auch stetigen Einflussgrößen (ähnlich wie in der linearen Regression). Dazu wird die Hazard-Rate (!) λ als Funktion der Zeit und eines Vektors $\mathbf{x}' = (x_1, \ldots, x_p)$ an Covariablen geschrieben: $\lambda(t; \mathbf{x})$. Dies induziert vermöge (116) bzw. (118) natürlich sofort eine entsprechende Abhängigkeit der Survival-Funktion von diesen Größen. Für die Modellierung dieser Abhängigkeit gibt es verschiedene Möglichkeiten, aber das in der Survival Analysis wohl am weitesten verbreitete Regressionsmodell ist das sogenannte Proportional Hazards Modell:

12.3.1 Definition des Proportional Hazards Modells

Im Proportional Hazards Modell wird angenommen, dass die Risiken (Engl.: "hazards") zweier Individuen mit verschiedenen Covariablenvektoren proportional zueinander sind, wobei die Proportionalitätskonstante eine Funktion eines „Abstandes" der beiden Covariablenvektoren ist. Dies führt für die Hazard-Rate eines Individuums mit Covariablenvektor $\mathbf{x}$ zu dem Modellansatz

$$\lambda(t; \mathbf{x}) = \lambda_0(t)\psi(\mathbf{x}),$$

wobei λ_0 die sogenannte Baseline Hazard-Funktion ist und $\psi(\mathbf{x})$ die (zeit**un**abhängige!) Proportionalitätskonstante, die häufig relatives Risiko, "risk ratio" oder auch "hazard ratio" genannt wird.

Da sinnigerweise weder eine Hazard-Rate noch eine risk ratio negativ sein kann, wird $\psi(\mathbf{x}) := \exp\{\mathbf{x}'\boldsymbol{\beta}\}$ mit einem (unbekannten und daher aus Daten zu schätzenden) Parametervektor $\boldsymbol{\beta}' = (\beta_1, \ldots, \beta_p)$ angesetzt, sodass sich

$$\lambda(t; \mathbf{x}) = \lambda_0(t) \exp\{\mathbf{x}'\boldsymbol{\beta}\} \quad \text{— das Cox Proportional Hazards Modell} \tag{138}$$

ergibt. Der Term $\mathbf{x}'\boldsymbol{\beta}$ ist offenbar eine parametrische Komponente des Modells, wohingegen über λ_0 keinerlei parametrischen Annahmen gemacht werden. Es handelt sich somit um ein sogenanntes *semi-parametrisches* Modell für die Hazard-Rate. Darüber hinaus ist dieser Ansatz wegen

$$\log\left\{\frac{\lambda(t; \mathbf{x})}{\lambda_0(t)}\right\} = \mathbf{x}'\boldsymbol{\beta}$$

ein *log-lineares* Modell für die hazard ratio.

Die Größe $\mathbf{x}'\boldsymbol{\beta}$ wird die lineare Komponente genannt, aber häufiger Risiko-Score oder prognostischer Index; letzteres typischerweise in bio-medizinischen Anwendungen. Offenbar ist die Baseline Hazard-Funktion λ_0 die Hazard-Rate eines (möglicherweise fiktiven) Individuums, dessen prognostischer Index den Wert 0 hat, was beispielsweise der Fall ist, wenn alle Covariablen gleich 0 sind. Zu bemerken ist, dass der Covariablenvektor $\mathbf{x}$ (anders als bei der linearen Regression) keine Komponente mit dem konstanten Wert 1 hat, d. h., dass im prognostischen Index kein konstanter Term, sagen wir β_0, steht. Dieser ist auch nicht nötig, da er durch Reparametrisierung in die Baseline Hazard-Funktion integriert werden kann, was o. B. d. A. grundsätzlich als geschehen angenommen wird.

Als Konsequenz des obigen Ansatzes (138) folgt aus

$$\frac{\lambda(t; \mathbf{x})}{\lambda(t; \mathbf{y})} = \frac{\lambda_0(t) \exp\{\mathbf{x}'\boldsymbol{\beta}\}}{\lambda_0(t) \exp\{\mathbf{y}'\boldsymbol{\beta}\}} = \exp\{(\mathbf{x} - \mathbf{y})'\boldsymbol{\beta}\},$$

dass die Proportionalitätskonstante der Risiken zweier Individuen mit Covariablenvektoren $\mathbf{x}$ und $\mathbf{y}$ gleich $\exp\{(\mathbf{x}-\mathbf{y})'\boldsymbol{\beta}\}$ ist. Dies wiederum bedeutet, dass e^{β_j} der Faktor ist, um den die Hazard-Rate (= das Risiko) im Fall $\beta_j > 0$ steigt bzw. im Fall $\beta_j < 0$ sinkt, wenn sich der Wert der Covariablen x_j um *eine* Einheit erhöht. Man könnte die Größen $e^{\beta_j}, j = 1, \ldots, p$, daher auch covariablenspezifische hazard ratios nennen.

Auf der Ebene der Survival-Funktion erhalten wir im Proportional Hazards Modell *im Fall einer stetigen Verteilungsfunktion* F aus der Beziehung (116), dass gilt:

$$\begin{aligned} 1 - F(t;\mathbf{x}) &= \exp\{-\Lambda(t;\mathbf{x})\} \equiv \exp\left\{-\int_0^t \lambda(u;\mathbf{x})\, du\right\} \\ &= \exp\left\{-\int_0^t \lambda_0(u)\, du\, \exp\{\mathbf{x}'\boldsymbol{\beta}\}\right\} = \left[\exp\left\{-\int_0^t \lambda_0(u)\, du\right\}\right]^{\exp\{\mathbf{x}'\boldsymbol{\beta}\}} \\ &=: [1 - F_0(t)]^{\exp\{\mathbf{x}'\boldsymbol{\beta}\}} \end{aligned} \tag{139}$$

Die Größe $1 - F_0$ wird Baseline Survival-Funktion genannt. Da stets $1 - F_0(t) \in [0,1]$ ist und $1 - F(t;\mathbf{x})$ durch Potenzierung der Baseline Survival-Funktion entsteht, folgt, dass sich die Survival-Kurven für verschiedene x-Werte im Inneren des Trägers von F_0 (also auf $\{t : F_0(t) \in (0,1)\}$) *nie* schneiden, sondern gewissermaßen „parallel“ zueinander (besser „zwiebelschalenförmig“) verlaufen. (Stichwort: Lehmann Alternativen.)

12.3.2 Parameterschätzung und Inferenz

Wie bereits erwähnt, sind die unbekannten Größen im Cox Proportional Hazards Modell die Baseline Hazard-Funktion λ_0 und der Parametervektor $\boldsymbol{\beta}$. Das Ziel ist es, einen Schätzer für das unbekannte $\boldsymbol{\beta}$ zu bestimmen, da seine Komponenten den Einfluss der Covariablen quantifizieren. λ_0 selbst ist zunächst nicht von Interesse. Auf den ersten Blick scheint jedoch die Kenntnis der Baseline Hazard-Funktion vonnöten zu sein, was aber günstigerweise *nicht* der Fall ist. Durch Verwendung und Maximierung einer sogenannten partiellen Likelihood-Funktion erhält man einen Maximum-Likelihood-Schätzer für $\boldsymbol{\beta}$, ohne den “nuisance parameter” λ_0 kennen zu müssen.

Wir skizzieren hier ganz grob die Vorgehensweise und die für die Maximum-Likelihood-Schätzer sowie einige Teststatistiken folgenden Verteilungsaussagen. (Siehe auch [22, Collett (2015)], Abschnitt 3.3 ff. Näheres zur Theorie der Maximum-Likelihood-Schätzung und zu Likelihood-Quotienten-Tests findet sich z. B. in [25, Cox und Hinkley (1974)].)

Zu einem Satz an Survival-Daten mit Covariablen $(Z_i, \delta_i, \mathbf{x}_i), i = 1, \ldots, n$, lautet die partielle Likelihood-Funktion (im Cox Proportional Hazards Modell)

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \left(\frac{\exp\{\mathbf{x}_i'\boldsymbol{\beta}\}}{\sum_{j=1}^{n} 1_{\{Z_j \geq Z_i\}} \exp\{\mathbf{x}_j'\boldsymbol{\beta}\}} \right)^{\delta_i} \tag{140}$$

und für die log-Likelihood-Funktion gilt:

$$\ln L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \delta_i \left(\mathbf{x}_i'\boldsymbol{\beta} - \ln\left\{ \sum_{j=1}^{n} 1_{\{Z_j \geq Z_i\}} \exp\{\mathbf{x}_j'\boldsymbol{\beta}\} \right\} \right)$$

(L heißt „partielle“ Likelihood-Funktion, weil sie keine Likelihood-Funktion „im üblichen Sinn“ ist, da die beobachteten zensierten und unzensierten Z_i nicht *direkt* in sie eingehen.)

Die Maximierung von $\ln L(\boldsymbol{\beta})$ in $\boldsymbol{\beta}$ liefert den Maximum-Likelihood-Schätzer $\hat{\boldsymbol{\beta}}$, der für den Vektor der „effizienten Scores"

$$\mathbf{u}(\boldsymbol{\beta}) := \left(\frac{\partial \ln L(\boldsymbol{\beta})}{\partial \beta_1}, \ldots, \frac{\partial \ln L(\boldsymbol{\beta})}{\partial \beta_p} \right)',$$

also den Gradientenvektor von $\ln L$, notwendigerweise $\mathbf{u}(\hat{\boldsymbol{\beta}}) = 0$ erfüllt.

Eine Reihe von Argumenten stützt die Aussage, dass der Vektor $\hat{\boldsymbol{\beta}}$ asymptotisch p-dimensional normalverteilt ist. Zur Approximation der Kovarianzmatrix $\mathbf{D}(\hat{\boldsymbol{\beta}})$ von $\hat{\boldsymbol{\beta}}$ dient die <u>beobachtete Informationsmatrix</u>

$$\mathbf{I}(\boldsymbol{\beta}) := -\left(\frac{\partial^2 \ln L(\boldsymbol{\beta})}{\partial \beta_j \partial \beta_k} \right)_{1 \le j,k \le p},$$

die die negative *Hesse-Matrix* der log-Likelihood-Funktion ist. Es gilt:

$$\mathbf{D}(\hat{\boldsymbol{\beta}}) \approx \mathbf{I}^{-1}(\hat{\boldsymbol{\beta}})$$

Unter Verwendung weiterer Argumente erhält man die folgenden drei, unter $H_0 : \boldsymbol{\beta} = \mathbf{0}$ approximativ χ^2_p-verteilten Statistiken für Tests der Hypothese H_0:

- die Likelihood-Quotienten-Statistik

$$2\left[\ln L(\hat{\boldsymbol{\beta}}) - \ln L(\mathbf{0})\right] \tag{141}$$

- die Wald-Statistik

$$\hat{\boldsymbol{\beta}}' \mathbf{I}(\hat{\boldsymbol{\beta}}) \hat{\boldsymbol{\beta}} \tag{142}$$

- die "efficient scores"-Statistik

$$\mathbf{u}(\mathbf{0})' \mathbf{I}^{-1}(\hat{\boldsymbol{\beta}}) \mathbf{u}(\mathbf{0}) \tag{143}$$

Eine spezielle Folgerung für die einzelnen Komponenten $\hat{\beta}_j$ von $\hat{\boldsymbol{\beta}}$ lautet:

$$\frac{\hat{\beta}_j}{\hat{\sigma}_{\hat{\beta}_j}} \text{ ist approximativ } \mathcal{N}(0,1)\text{-verteilt unter } H_0 : \beta_j = 0, \tag{144}$$

wobei $\hat{\sigma}^2_{\hat{\beta}_j}$ das j-te Diagonalelement von $\mathbf{I}^{-1}(\hat{\boldsymbol{\beta}})$ ist.
(Äquivalent dazu ist, dass $\hat{\beta}_j^2 / \hat{\sigma}^2_{\hat{\beta}_j}$ unter $H_0 : \beta_j = 0$ approximativ χ^2_1-verteilt ist.)

Beachte: Die obigen drei Teststatistiken (141), (142) und (143) werden für den Test von $H_0 : \boldsymbol{\beta} = \mathbf{0}$ ($\Leftrightarrow$ *gar keine* Regressionsbeziehung) verwendet. Der komponentenweise Test in (144) lässt nur Schlüsse über $H_0 : \beta_j = 0$ zu *in Anwesenheit* der übrigen Komponenten $\beta_i, i \neq j$, im Modell (Stichwort: marginaler p-Wert).

Zum Vergleich zweier Cox Proportional Hazard Modelle M_1 und M_2, von denen das eine (M_2) eine echte Erweiterung des anderen (M_1) ist, lässt sich ebenfalls die Likelihood-Quotienten-Statistik verwenden. Angenommen, die Hazard-Funktionen λ_1 und λ_2 der beiden Modelle M_1 und M_2 lauten

$$\lambda_1(t; (x_1, \ldots, x_p)) = \lambda_0(t) \exp\{\beta_1 x_1 + \cdots + \beta_p x_p\} \equiv \lambda_0(t) \exp\{\mathbf{x}'_{(1)} \boldsymbol{\beta}_{(1)}\}$$

bzw.

$$\begin{aligned} &\lambda_2(t; (x_1, \ldots, x_p, x_{p+1}, \ldots, x_{p+q})) \\ &= \lambda_0(t) \exp\{\beta_1 x_1 + \cdots + \beta_p x_p + \beta_{p+1} x_{p+1} + \cdots + \beta_{p+q} x_{p+q}\} \\ &\equiv \lambda_0(t) \exp\{\mathbf{x}'_{(2)} \boldsymbol{\beta}_{(2)}\} \end{aligned}$$

Man sagt auch, Modell 1 ist ein Submodell von Modell 2 (oder auf Englisch: "model 1 is *parametrically nested within* model 2") und schreibt kurz $M_1 \subset M_2$. Die zu testende Hypothese möge lauten:

$$H_0 : \beta_{p+1} = \cdots = \beta_{p+q} = 0$$

Dann gilt unter diesem H_0 approximativ:

$$2\left[\ln L(\hat{\boldsymbol{\beta}}_{(2)}) - \ln L(\hat{\boldsymbol{\beta}}_{(1)})\right] \sim \chi^2_q, \tag{145}$$

sodass mit Hilfe dieser Likelihood-Quotienten-Statistik ein (approximativer) Test obiger Hypothese durchgeführt werden kann.

12.3.3 Das Cox Proportional Hazards Modell in R: `coxph()`

Nun zur Implementation des Cox Proportional Hazard Modells in **R**: Den Modell-Fit (und noch einiges mehr) erledigt die Funktion `coxph()`. In ihrer Grundversion benötigt sie (ähnlich der Funktion `survfit()`) zwei Argumente: Eine Formel, die in der üblichen Syntax (vgl. lineare Modelle) die mittels der Funktion `Surv()` geeignet zusammengefassten Überlebensdauern Z_i und Zensierungsindikatoren δ_i an den Covariablen „modelliert", sowie einen Data Frame, in dem sich alle verwendeten Variablen befinden. Als Ergebnis erhält man ein `coxph`-Objekt.

Die Funktionsweise von `coxph()` soll an dem Krebsdatenbeispiel aus [22, Collett (2015) oder Vorgängerversionen] erläutert werden. In diesem Beispiel wurde für jedes Individuum der Studie über die eigentlichen Survival-Daten hinaus zusätzlich das Alter und ein Satz medizinischer Covariablen bestimmt: BUN ("blood urea nitrogen"), CA (Serum-Calciumgehalt), HB (Serum-Hämoglobingehalt), PC (Prozentsatz Plasmazellen im Knochenmark) und BJ (Indikator, der das Vorhandensein des Bence-Jones-Proteins im Urin angibt; Codierung: 1 = vorhanden, 0 = nicht vorhanden). Wir wollen ein Cox Proportional Hazard Modell fitten, das **nur die stetigen** Covariablen Alter, BUN, CA, HB und PC enthält. D. h., wir setzen für die Survival-Funktion $1 - F$ an:

$$1 - F(t; \mathbf{x}) = [1 - F_0(t)]^{\exp\{\beta_1 \text{Alter} + \beta_2 \text{BUN} + \beta_3 \text{CA} + \beta_4 \text{HB} + \beta_5 \text{PC}\}} \tag{146}$$

mit $\mathbf{x}' = (\text{Alter, BUN, CA, HB, PC})$ sowie unbekannten F_0 und $\boldsymbol{\beta}' = (\beta_1, \ldots, \beta_5)$.

Angenommen, die Daten sind in dem Data Frame `myeloma` erfasst:

```
> myeloma
    Z delta MW Alter BUN CA   HB  PC BJ
 1 13     1  m    66  25 10 14.6  18  1
 2 52     0  m    66  13 11 12.0 100  0
 3  6     1  w    53  15 13 11.4  33  1
....
46 12     1  w    60   6 10  5.5  25  0
47 17     1  w    65  28  8  7.5   8  0
48  3     0  m    59  90 10 10.2   6  1
```

Dann ist in **R** die Vorgehensweise folgendermaßen:

```
> summary( myel.coxph <- coxph( Surv( Z, delta) ~ Alter + BUN + CA + HB + PC,
+  data = myeloma))
Call: coxph(formula = Surv(Z, delta) ~ Alter + BUN + CA + HB + PC,
   data = myeloma)

 n= 48, number of events= 36
           coef exp(coef)  se(coef)      z Pr(>|z|)
```

```
Alter -0.011747  0.988322  0.027578 -0.426 0.670155
BUN    0.020255  1.020461  0.005842  3.467 0.000527 ***
CA     0.010135  1.010187  0.127945  0.079 0.936861
HB    -0.140869  0.868603  0.065041 -2.166 0.030322 *
PC     0.002457  1.002460  0.006245  0.393 0.693986
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

      exp(coef) exp(-coef) lower .95 upper .95
Alter    0.9883     1.0118    0.9363    1.0432
BUN      1.0205     0.9799    1.0088    1.0322
CA       1.0102     0.9899    0.7861    1.2981
HB       0.8686     1.1513    0.7646    0.9867
PC       1.0025     0.9975    0.9903    1.0148

Rsquare= 0.259   (max possible= 0.989 )
Likelihood ratio test= 14.39  on 5 df,   p=0.01333
Wald test            = 16.88  on 5 df,   p=0.004741
Score (logrank) test = 21.29  on 5 df,   p=0.0007133
```

Bemerkung: Die Ausgabe eines `coxph`-Objektes zeigt eine echte Teilmenge der Informationen, die die Anwendung der Funktion `summary()` auf ein solches Objekt liefert. Daher beschränken wir unsere Beschreibung hier auf letzteres.

Zur Interpretation der obigen Ausgabe: Zunächst wird der Funktionsaufruf protokolliert, wie groß der Stichprobenumfang n ist und wie viele "`events`", sprich unzensierte Beobachtungen es gegegeben hat. Die dann folgende Tabelle enthält für jede Covariable von Modell (146) in der entsprechend benannten Zeile die folgenden Infos:

Spalte	Bedeutung
`coef`	Schätzwert $\hat{\beta}_j$ für jeweiliges β_j
`exp(coef)`	Schätzwert $\exp\{\hat{\beta}_j\}$ für die covariablenspezifischen hazard ratio $\exp\{\beta_j\}$ der Covariablen x_j
`se(coef)`	Schätzer $\hat{\sigma}_{\hat{\beta}_j}$ für die Standardabweichung $\sigma_{\hat{\beta}_j}$ von $\hat{\beta}_j$
`z`	Wert der Teststatistik $\hat{\beta}_j/\hat{\sigma}_{\hat{\beta}_j}$ aus (144) für den Test der Hypothese $H_0 : \beta_j = 0$
`Pr(>|z|)`	p-Wert des Tests von $H_0 : \beta_j = 0$
`exp(-coef)`	$\exp\{-\hat{\beta}_j\}$
`lower .95`	Untergrenze eines approximativen 95 %-Konfidenzintervalls für $\exp\{\beta_j\}$ (Niveau durch Argument `conf.int` in `summary()` variierbar)
`upper .95`	Obergrenze dieses Konfidenzintervalls

Unter den Koeffizienteninformationen wird ein R^2-Wert (`Rsquare`) nebst seinem maximal möglichen Wert R^2_{max} (`max possible`) angegeben. Weder Herkunft noch Interpretation sind für mich bisher erklärbar, aber ihre Definitionen lauten:

$$R^2 = 1 - \exp\left\{\frac{2}{n}\left[\ln L(\mathbf{0}) - \ln L(\hat{\boldsymbol{\beta}})\right]\right\} \qquad \text{und} \qquad R^2_{max} = 1 - \exp\left\{\frac{2}{n}\ln L(\mathbf{0})\right\}$$

(R^2_{max} ist offenbar eine obere Schranke von R^2, da stets $-\ln L(\hat{\boldsymbol{\beta}}) \geq 0$, weil $\ln L(\boldsymbol{\beta}) \leq 1$, vgl. 140.)

Schlussendlich werden auch noch die Werte, zugehörigen Freiheitsgrade und p-Werte des Hypothesentests $H_0 : \boldsymbol{\beta} = \mathbf{0}$ für die Likelihood-Quotienten-Statistik (141), die Wald-Statistik (142) und die "efficient score"-Statistik (143) ausgedruckt.

Die aus dem gefitteten Cox Proportional Hazards Modell resultierenden Survival-Kurven können sowohl numerisch ausgegeben als auch geplottet werden. Dazu muss jedoch zunächst die Funktion `survfit()` auf das `coxph`-Objekt angewendet werden und das Resultat entweder an die Funktion `summary()` für eine numerische oder an die Funktion `plot()` für eine grafische Darstellung der Kurve übergeben werden.

Das Resultat ist *voreinstellungsgemäß* die Cox Survival-Kurve für ein fiktives Individuum, dessen Covariablenwerte die arithmetischen Mittel der im Fit verwendeten Covariablenwerte sind! In unserem Beispiel wäre das eine Person mit den folgenden Covariablenwerten:

```
> sapply( myeloma[ c("Alter", "BUN", "CA", "HB", "PC") ], mean)
    Alter      BUN     CA       HB      PC
 62.89583 33.91667 9.9375 10.25208 42.9375
```

Es wird also ein Schätzer für die Kurve

$$1 - F(t; \mathbf{x}' = (62.89583, 33.91667, 9.9375, 10.25208, 42.9375)) = [1 - F_0(t)]^{\exp\{\mathbf{x}'\boldsymbol{\beta}\}}$$

geliefert:

```
> summary( survfit( myel.coxph))
Call: survfit.coxph(object = myel.coxph)

 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    1     48       3   0.9630  0.0245     0.916066        1.000
    4     44       2   0.9301  0.0365     0.861242        1.000
    5     42       4   0.8389  0.0553     0.737171        0.955
    6     38       2   0.7898  0.0620     0.677100        0.921
....
   88      2       1   0.0379  0.0404     0.004691        0.306
   91      1       1   0.0109  0.0186     0.000387        0.308
```

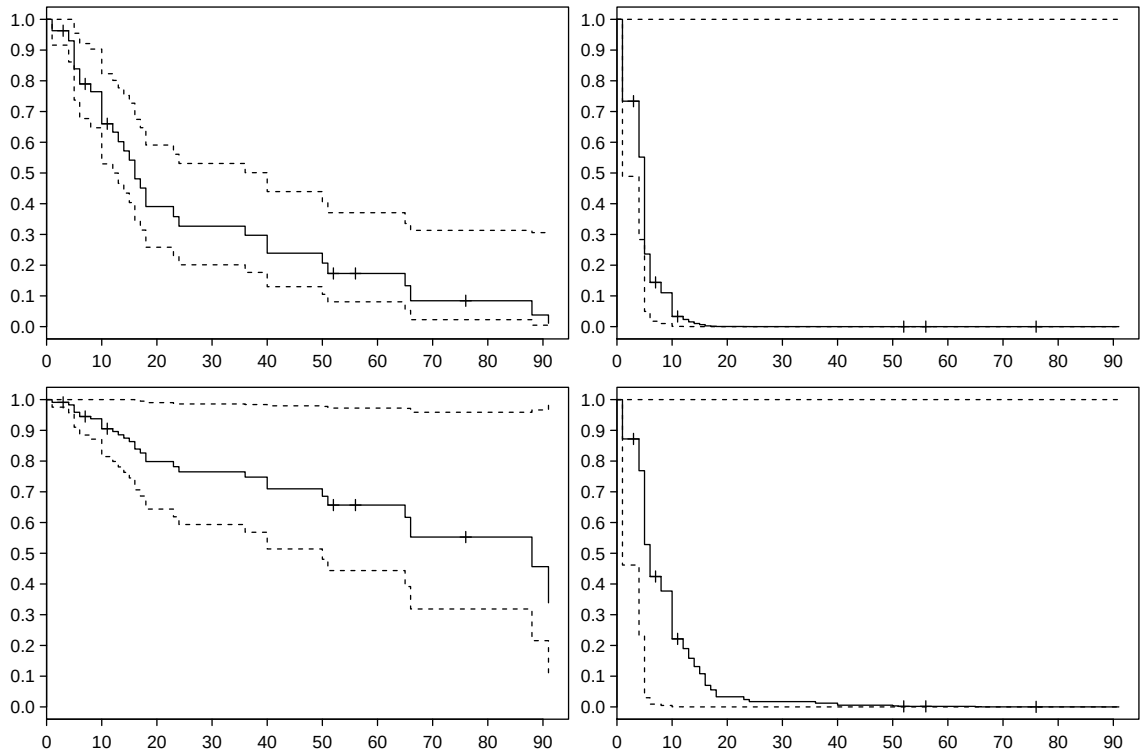
Einen Plot der Kurve (im Bild links oben auf der nächsten Seite) erhält man durch:

```
> plot( survfit( myel.coxph))
```

Um die Survival-Kurven für andere Covariablenvektorwerte zu erhalten, ist ein weiteres Argument der Funktion `survfit()` zu verwenden, und zwar das Argument `newdata`. Ihm muss ein Data Frame mit vollständigen Vektoren an Covariablenwerten zugewiesen werden.

Die folgenden drei Befehle lieferten (in dieser Reihenfolge) den rechten oberen, den linken unteren und den rechten unteren Plot in der Grafik oben auf der nächsten Seite. Dabei ist die Survival-Kurve im rechten unteren Plot für $x = 0$ gezeichnet, also die Baseline Survival-Funktion $1 - F_0$.

```
> plot( survfit( myel.coxph, newdata = data.frame( Alter = 60, BUN = 100,
+ CA = 10, HB = 5, PC = 40)) )
> plot( survfit( myel.coxph, newdata = data.frame( Alter = 60, BUN = 2,
+ CA = 10, HB = 16, PC = 40)) )
> plot( survfit( myel.coxph, newdata = data.frame( Alter = 0, BUN = 0,
+ CA = 0, HB = 0, PC = 0)) )
```



Bemerkung: Hängt der Schätzer für die Baseline Survival-Funktion mit dem (nicht-parametrischen) Kaplan-Meier-Schätzer zusammen? Die theoretische Beziehung $1 - F(\cdot;\mathbf{x}) = [1 - F_0]^{\exp\{\mathbf{x}'\boldsymbol{\beta}\}}$ zwischen der „individuellen" und der Baseline Survival-Funktion (s. (139) auf S. 365 oben) begründet den (in obigen Grafiken dargestellten) Schätzer

$$1 - \hat{F}(t;\mathbf{x}) := \exp\left\{-\hat{\Lambda}_0(t)\ \exp\{\mathbf{x}'\hat{\boldsymbol{\beta}}\}\right\},$$

wobei, wie man zeigen kann, der Schätzer $\hat{\Lambda}_0$ von Breslow für Λ_0 folgendermaßen lautet:

$$\hat{\Lambda}_0(t) = \sum_{i=1}^{n} \frac{1\{Z_{i:n} \le t, \delta_{[i:n]} = 1\}}{\sum_{j=1}^{n} 1\{Z_{j:n} \ge Z_{i:n}\} \exp\{\mathbf{x}'_{[j:n]}\hat{\boldsymbol{\beta}}\}} = \sum_{i=1}^{n} \frac{\delta_{[i:n]} 1\{Z_{i:n} \le t\}}{\sum_{j=i}^{n} \exp\{\mathbf{x}'_{[j:n]}\hat{\boldsymbol{\beta}}\}}$$

Dies führt wegen $1 - \hat{F}_0(t) = 1 - \hat{F}(t;\mathbf{0}) = \exp\left\{-\hat{\Lambda}_0(t)\right\}$ mit $\hat{e}_j := \exp\{\mathbf{x}'_{[j:n]}\hat{\boldsymbol{\beta}}\}$ und unter Ausnutzung der Approximation $\ln(1-u) \approx -u$ für $|u| < 1$ zu

$$\begin{aligned} 1 - \hat{F}_0(t) &= \exp\left\{-\sum_{Z_{i:n} \le t} \frac{\delta_{[i:n]}}{\sum_{j=i}^{n} \hat{e}_j}\right\} = \prod_{Z_{i:n} \le t} \left(\exp\left\{\frac{-1}{\sum_{j=i}^{n} \hat{e}_j}\right\}\right)^{\delta_{[i:n]}} \\ &\approx \prod_{Z_{i:n} \le t} \left(\frac{\sum_{j=i}^{n} \hat{e}_j - 1}{\sum_{j=i}^{n} \hat{e}_j}\right)^{\delta_{[i:n]}} \end{aligned}$$

Für den Kaplan-Meier-Schätzer hat man gemäß (124) (siehe Seite 348), dass

$$1 - \hat{F}_n(t) = \prod_{Z_{i:n} \le t} \left(1 - \frac{1}{\sum_{j=1}^{n} 1\{Z_{j:n} \ge Z_{i:n}\}}\right)^{\delta_{[i:n]}} = \prod_{Z_{i:n} \le t} \left(\frac{n-i}{n-i+1}\right)^{\delta_{[i:n]}},$$

was ein gewisse Ähnlichkeit zwischen den beiden zeigt. Dennoch ist klar festzustellen, dass es es sich **nicht** um dieselben Survival-Funktionen handelt.

(Müsste eigentlich fortgesetzt werden ...)

Literatur

[1] Agresti, A.: *Categorical Data Analysis.* John Wiley, New York, 1990. (2nd ed., 2002: ∼ 135 €)

[2] Agresti, A.: *An Introduction to Categorical Data Analysis.* John Wiley, New York, 1996. (2nd ed., 2007: ∼ 71 €)

[3] Akaike, H.: *A new look at statistical model identification.* IEEE Transactions on Automatic Control, Vol. AC-19, No. 6, 1974, pp. 716 - 723.

[4] Bates, D. M. : *lme4: Mixed-effects modeling with R.* Unveröffentlicher und unfertiger Entwurf, 2010. URL http://lme4.r-forge.r-project.org/book/

[5] Belsley, D. A., Kuh, E., Welsch, R. E.: *Regression Diagnostics.* John Wiley, New York, 1980. (∼ 28 €, gebraucht)

[6] Bock, J.: *Bestimmung des Stichprobenumfangs für biologische Experimente und kontrollierte klinische Studien.* R. Oldenbourg Verlag, München, 1998. (Evtl. vergriffen.)

[7] Bortz, J., Lienert, G. A., Boehnke, K.: *Verteilungsfreie Methoden in der Biostatistik.* 2., korrig. und aktualis. Auflage, Springer-Verlag, Berlin, 2000. (oder 3. Aufl. 2008). (Beides Taschenbücher á ∼ 50 €)

[8] Box, Cox: *An analysis of transformations.* Journal of the Royal Statistical Society, Series B, Vol. 26, 1964

[9] Box, G. E. P., Hunter, W. G., Hunter, J. S.: *Statistics for Experimenters: An Introduction to Design, Data Analysis and Model Building.* John Wiley, New York, 1978. (∼ 79 €; 2nd ed., 2005: ∼ 125 €)

[10] Braun, W. J., Murdoch, D. J.: *A First Course in Statistical Programming with R.* Cambridge University Press, 2007. (Taschenbuchausgabe aus 2011: ∼ 37 €)

[11] Bretz, F., Hothorn, T., Westfall, P.: *Multiple Comparisons Using R.* Chapman & Hall/CRC Press, Boca Raton/Florida, 2010. (∼ 67 €)

[12] Brown, L. D., Cai, T. T., DasGupta, A.: *Interval Estimation for a Binomial Proportion.* Statistical Sciences, 2001, Vol. 16, No. 2, pp. 101 - 133.

[13] Brunner, E., Domhof, S., Langer, F.: *Nonparametric Analysis of Longitudinal Data in Factorial Experiments.* John Wiley, 2002. (Vermutlich vergriffen.)

[14] Brunner, E., Langer, F.: *Nichtparametrische Analyse longitudinaler Daten.* Oldenbourg-Verlag, 1999. (Gebraucht ab ∼ 357 (!) €)

[15] Brunner, E., Munzel, U.: *Nicht-parametrische Datenanalyse. Unverbundene Stichproben.* 2. Auflage, Springer-Verlag, Berlin, 2013. (∼ 45 €, Taschenbuch)

[16] Büning, H., Trenkler, G.: *Nichtparametrische statistische Methoden.* 2., völlig neu überarb. Ausg., Walter-de-Gruyter, Berlin, 1994. (∼ 45 €, Taschenbuch)

[17] Chambers, J.: *Software for Data Analysis: Programming with R.* Corr. 2nd printing, Springer-Verlag, Berlin, 2008. (Taschenbuchausgabe aus 2010: ∼ 33 €)

[18] Chambers, J. M., Cleveland, W. S., Kleiner, B., Tukey, P. A.: *Graphical Methods for Data Analysis.* Wadsworth Pub. Co., Belmont/Californien, 1983. (Gebraucht ab 21 €)

[19] Cleveland, W. S.: *LOWESS: A program for smoothing scatterplots by robust locally weighted regression.* The American Statistician, 35, p. 54.

[20] Cleveland, W. S.: *The Elements of Graphing Data.* Wadsworth Advanced Books and Software, Monterey/Californien, 1985. (Hobart Pr., revised ed., 1994: Gebraucht ab 57 €)

[21] Cleveland, W. S., Grosse, E., Shyu, W. M.: *Local regression models.* Chapter 8 of *Statistical Models in S*, eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole, 1992. (Gebraucht ab 62 €)

[22] Collett, D.: *Modelling Survival Data in Medical Research.* 3rd ed., Chapman & Hall, London, 2015. (∼ 70 €)

[23] Cook, R. D.: *Detection of influential observations in linear regression.* Technometrics, 19 (1), pp. 15-18, 1977.

[24] Cook, R. D., Weisberg, S.: *Residuals and Influence in Regression.* Chapman & Hall, New York, 1982. (82 €)

[25] Cox, D. R., Hinkley, D. V.: *Theoretical Statistics.* Chapman & Hall, London, 1974. (∼ 60 €)

[26] Dalgaard, P.: *Introductory Statistics with R.* 2nd ed., Springer-Verlag, 2008. (∼ 50 €, paperback)

[27] Dilba, G., Schaarschmidt, F., Hothorn,L. A.: *Inferences for Ratios of Normal Means.* In: R News Vol. 7(1), April 2007, pp. 20 - 23. URL `http://cran.r-project.org/doc/Rnews/Rnews_2007-1.pdf`

[28] Everitt, B. S., Hothorn, T.: *A Handbook of Statistical Analyses Using R.* 2nd ed., Chapman & Hall/CRC, Boca Raton, 2010. (∼ 47 €, paperback)

[29] Faraway, J. J.: *Linear Models with R.* Chapman & Hall/CRC, Boca Raton, 2005. (58 €)

[30] Faraway, J. J.: *Extending the Linear Model with R.* Chapman & Hall/CRC, Boca Raton, 2006. (59 €)

[31] Fahrmeir, L., Kneib, T., Lang, S.: *Regression. Modelle, Methoden und Anwendungen.* 2. Aufl., Springer-Verlag, Berlin, 2009. (∼ 33 €, Taschenbuch)

[32] Fleiss, J. L., Levin, B., Paik, M. C.: *Statistical Methods for Rates and Proportions.* 3rd ed., John Wiley, New York, 2003. (∼ 133 €)

[33] Fox, J.: *Applied Regression Analysis, Linear Models, and Related Methods.* Sage Publications, Thousand Oaks, 1997. (99 €)

[34] Fox, J.: *An R and S-PLUS Companion to Applied Regression.* Sage Publications, Thousand Oaks, 2002. (43 €, paperpack)

[35] Friendly, M.: *Mosaic displays for multi-way contingency tables.* Journal of the American Statistical Association, 89, pp. 190 - 200.

[36] Friendly, M.: *Visualizing Categorical Data.* SAS Insitute, Carey, NC, 2000. URL `http://www.math.yorku.ca/SCS/vcd` (Gebraucht ab ∼ 157 €)

[37] Gardner, M. J., Altman, D. G.: *Confidence intervals rather than P values: estimation rather than hypothesis testing.* British Medical Journal 1986, Vol. 292, pp. 746 - 750.

[38] Ghosh, B. K.: *Some monotonicity theorems for χ^2, F and t distributions with appplications.* Journal of the Royal Statistical Society, Series B, 1973, Vol. 35, pp. 480 - 492.

[39] Goldberg, D.: *What Every Computer Scientist Should Know about Floating-Point Arithmetic.* ACM Computing Surveys, Vol. 23(1), 1991. Available at `http://www.validlab.com/goldberg/paper.ps`, extended version at `http://www.validlab.com/goldberg/paper.pdf`

[40] Graybill, F. A.: *Theory and Application of the Linear Model.* Duxbury Press, Wadworth Publishing Comp., Inc., North Scituate, 1976. (Gebunden gebraucht ab ∼ 93 €, Taschenbuch (2000) ∼ 117 €)

[41] Hagemann, S.: *Geschachtelte Hypothesensequenzen in linearen Modellen.* Bachelorthesis, Math. Inst., JLU Giessen, 2012.

[42] Harrell, Jr., F. E.: *Regression Modeling Strategies. With Applications to Linear Models, Logistic Regression, and Survival Analysis.* Corr. 2nd Printing, Springer-Verlag, New York, 2002. (87 €)

[43] Henze, N.: *Stochastik für Einsteiger. Eine Einführung in die faszinierende Welt des Zufalls.* 8., erw. Aufl., Vieweg + Teubner Verlag, 2010. (∼ 25 €; 9., erw. Aufl. 2011 ∼ 25 €, Taschenbuch)

[44] Hettmansperger, T. P.: *Statistical inference based on ranks.* John Wiley, New York, 1984. (Gebraucht ab ∼ 68 €)

[45] Hochberg, Y., Tamhane, A. C.: *Multiple Comparison Procedures.* John Wiley, New York, 1987. (∼ 187 €)

[46] Hocking, R. R.: *Methods and Applications of Linear Models: Regression and the Analysis of Variance.* John Wiley, New York, 1996. (108 €)

[47] Hocking, R. R.: *Methods and Applications of Linear Models: Regression and the Analysis of Variance.* 2nd ed., John Wiley, New York, 2003. (108 €)

[48] Hollander, M., Wolfe, D. G.: *Nonparametric statistical methods.* John Wiley, New York, 1974. (2nd ed., 1999; gebraucht ab ∼ 55 €)

[49] Horn, M., Vollandt, R.: *Multiple Tests und Auswahlverfahren.* Spektrum Akademischer Verlag, 1995. (∼ 21 €; ehemals im Gustav Fischer Verlag, Stuttgart erschienen)

[50] Hsu, J. C.: *Multiple Comparisons.* Chapman & Hall/CRC, London, 1996. (91 €)

[51] ICH E9: *Statistical Principles for Clinical Trials.* London, UK: International Conference on Harmonisation, 1998. Adopted by CPMP March 1998 (CPMP/ICH/363/96). URL `http://www.ich.org`

[52] Klein, J. P., Moeschberger, M. L.: *Survival Analysis. Techniques for Censored and Truncated Data.* 2nd ed., 2003 (corrected 3rd printing, 2005), Springer, New York, 2003. (∼ 92 €)

[53] Ligges, U.: *R Help Desk: Accessing the sources.* In: R News Vol. 6(4), October 2006, pp. 43 - 45. URL `http://cran.r-project.org/doc/Rnews/Rnews_2006-4.pdf`

[54] Ligges, U.: *Programmieren mit R.* 3., überarb. & aktualis. Auflage, Springer-Verlag, 2008. (∼ 33 €, Taschenbuch)

[55] Linhart, Zucchini: *?*, 1986.

[56] Maindonald, J., Braun, J.: *Data Analysis and Graphics Using R. An Example-based Approach.* 3rd ed., Cambridge University Press, 2010. (∼ 66 €).

[57] Mardia, K. V., Kent, J. T., Bibby, J. M.: *Multivariate Analysis.* Academic Press, London, 1979. (Evtl. vergriffen)

[58] Mathai, A. M., Provost, S. B.: *Quadratic Forms in Random Variables: Theory and Applications.* Marcel Dekker, Inc., New York, 1992. (Scheint vergriffen.)

[59] Matsumoto, M., Nishimura, T.: *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator.* ACM Transactions on Modeling and Computer Simulation, Vol. 8, 1998, pp. 3 - 30.

[60] Matloff, N.: *The Art of R Programming. A Tour of Statistical Software Design.* No Starch Press, Inc., San Francisco/California, 2011. (∼ 27 €, paperback)

[61] Meeker, W. Q., Escobar, L. A.: *Statistical Methods for Reliability Data.* John Wiley, New York, 1998. (∼ 124 €)

[62] Meyer, D., Zeileis, A., Hornik, K.: *The Strucplot Framework: Visualizing Multi-way Contingency Tables with vcd.* Journal of Statistical Software, Vol. 17 (3), 2006, pp. 1 - 48. URL `http://www.jstatsoft.org/v17/i03`

[63] Munzel, U., Hothorn, L. A.: *A Unified Approach to Simultaneous Rank Test Procedures in the Unbalanced One-way Layout.* Biometrical Journal, Vol. 43, No. 5, 2001, pp. 553 569.

[64] Murrell, P.: *R Graphics.* Chapman & Hall/CRC Press, Boca Raton/Florida, 2005. (Gebraucht ab ∼ 41 €)

[65] Murrell, P.: *R Graphics.* 2nd ed. Chapman & Hall/CRC Press, Boca Raton/Florida, 2011. (∼ 61 €)

[66] Neter, J., Wasserman, W., Kutner, M. H.: *Applied Linear Statistical Models: Regression, Analysis of Variance, and Experimental Designs.* 3rd ed., Richard D. Irwin, Inc., 1990. (Gebraucht ab 24 €) ODER: Kutner, M. H., Neter, J., Nachtsheim, C. J., Wasserman, W.: *Applied Linear Statistical Models.* 5th revised ed., McGraw Hill Higher Education, 2004. (Gebraucht ab 41 €, paperback)

[67] Piepho, H.-P.: *An Algorithm for a Letter-Based Representation of All-Pairwise Comparisons.* Journal of Computational and Graphical Statistics, Vol. 13, No. 2., 2004, pp. 456 - 466.

[68] Sachs, L., Hedderich, J.: *Angewandte Statistik. Methodensammlung mit R.* 12., vollst. neu bearb. Aufl., Springer-Verlag, 2006. (oder 14. Aufl., 2012, beide ∼ 47 €, Taschenbuch)

[69] Sarkar, D.: *Lattice: Multivariate Data Visualization with R.* Springer-Verlag, Berlin, 2008. (∼ 65 €, paperback)

[70] Searle, S. R.: *Linear Models.* John Wiley & Sons, New York, 1971.

[71] Seber, G. A. F.: *Linear Regression Analysis.* John Wiley & Sons, New York, 1977. (Gebraucht ab ca. 115 €)

[72] Snedecor, G. W., Cochran, W. G.: *Statistical Methods.* 8th ed., Iowa State University Press, Ames, Iowa, 1989; Blackwell Publishers. (∼ 88 €)

[73] Stute, W.: *Kaplan-Meier Integrals.* Handbook of Statistics, Vol. 23, Elsevier, 2004.

[74] Therneau, T. M., Grambsch, P. M.: *Modeling Survival Data: Extending the Cox Model.* 2nd printing, Springer-Verlag, New York, 2001. (Gebunden ∼ 108 €; Taschenbuch (2010): ∼ 97 €)

[75] Timischl, W.: *Biostatistik. Eine Einführung für Biologen.* Springer-Verlag, Wien, New York, 1990. (Nachfolger: *Biostatistik. Eine Einführung für Biologen und Mediziner.* 2., neu bearb. Aufl., 2000. (∼ 40 €, Taschenbuch))

[76] Tong, Y. L.: *The multivariate normal distribution.* Springer-Verlag, New York, 1990. (Scheint vergriffen.)

[77] Tukey, J. W.: *Exploratory Data Analysis* Addison-Wesley, Reading/Massachusetts, 1977. (∼ 73 €, paperback)

[78] Venables, W. N.: *Exegeses on linear models.* A paper presented to the S-PLUS User's Conference in Washington, DC/USA, 8 - 9th Oct. 1998. Version of May 13, 2000. URL `http://www.stats.ox.ac.uk/pub/MASS3/Exegeses.pdf`

[79] Venables, W. N., Ripley, B. D.: *Modern Applied Statistics with S.* 4th ed., Corr. 2nd printing, Springer-Verlag, New York, 2003. (∼ 82 €, paperback)

[80] Venables, W. N., Ripley, B. D.: *S Programming.* Corr. 3rd printing, Springer-Verlag, New York, 2004. (∼ 97 €) ODER: Corr. 2nd printing, 2001. (∼ 68 €, Taschenbuch)

[81] Verzani, J.: *Using R for Introductory Statistics.* Chapman & Hall/CRC Press, Boca Raton/Florida, 2005. (∼ 43 €)

[82] Weisberg, S.: *Applied Linear Regression.* 3rd ed., John Wiley, New York, 2005. (82 €)

[83] Wickham, H.: *Reshaping data with the reshape Package.* Journal of Statistical Software, 2007, Vol. 21, No. 12, pp. 1 - 20.

[84] Witting, H., Müller-Funk, U.: *Mathematische Statistik II. Asymptotische Statistik: Parametrische Modelle und nichtparametrische Funktionale.* Teubner, Stuttgart, 1995. (Scheint vergriffen.)

[85] Zeileis, A., Leisch, F., Hornik, K., Kleiber, C.: *strucchange: An R package for testing for structural change in linear regression models.* Journal of Statistical Software, 7(2):1 - 38, 2002. URL `http://www.jstatsoft.org/v07/i02/`.

[86] Zhao, Y. D., Rahardja, D., Qu, Y.: *Sample size calculation for the Wilcoxon-Mann-Whitney test adjusting for ties.* Statistics in Medicine 2008, Vol. 27, pp. 462 - 468.

[87] Zuur, A. F., Ieno, E. N., Elphick, C. S.: *A protocol for data exploration to avoid common statistical problems.* Methods in Ecology & Evolution 2009, pp. 1 - 12.

Bemerkung: Circa-Preise lt. `Amazon.de` zwischen November 2012 und Februar 2015.