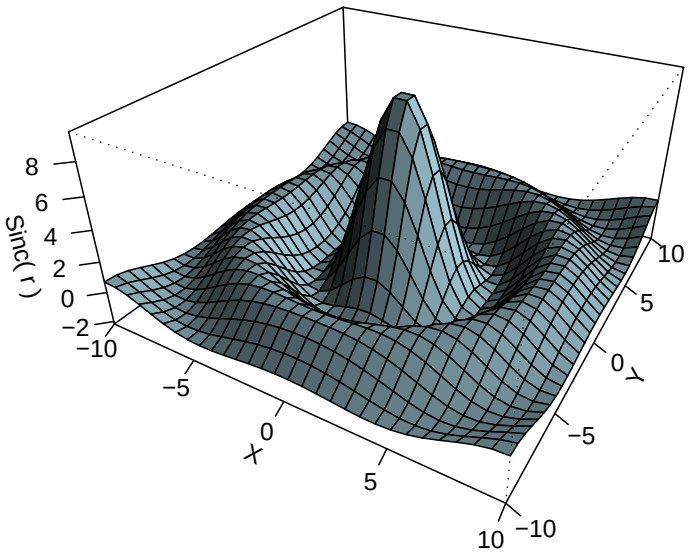
Skript zum Umgang und zur multivariaten Datenanalyse mit R
http://r-project.org

Grafiken und Statistik in R

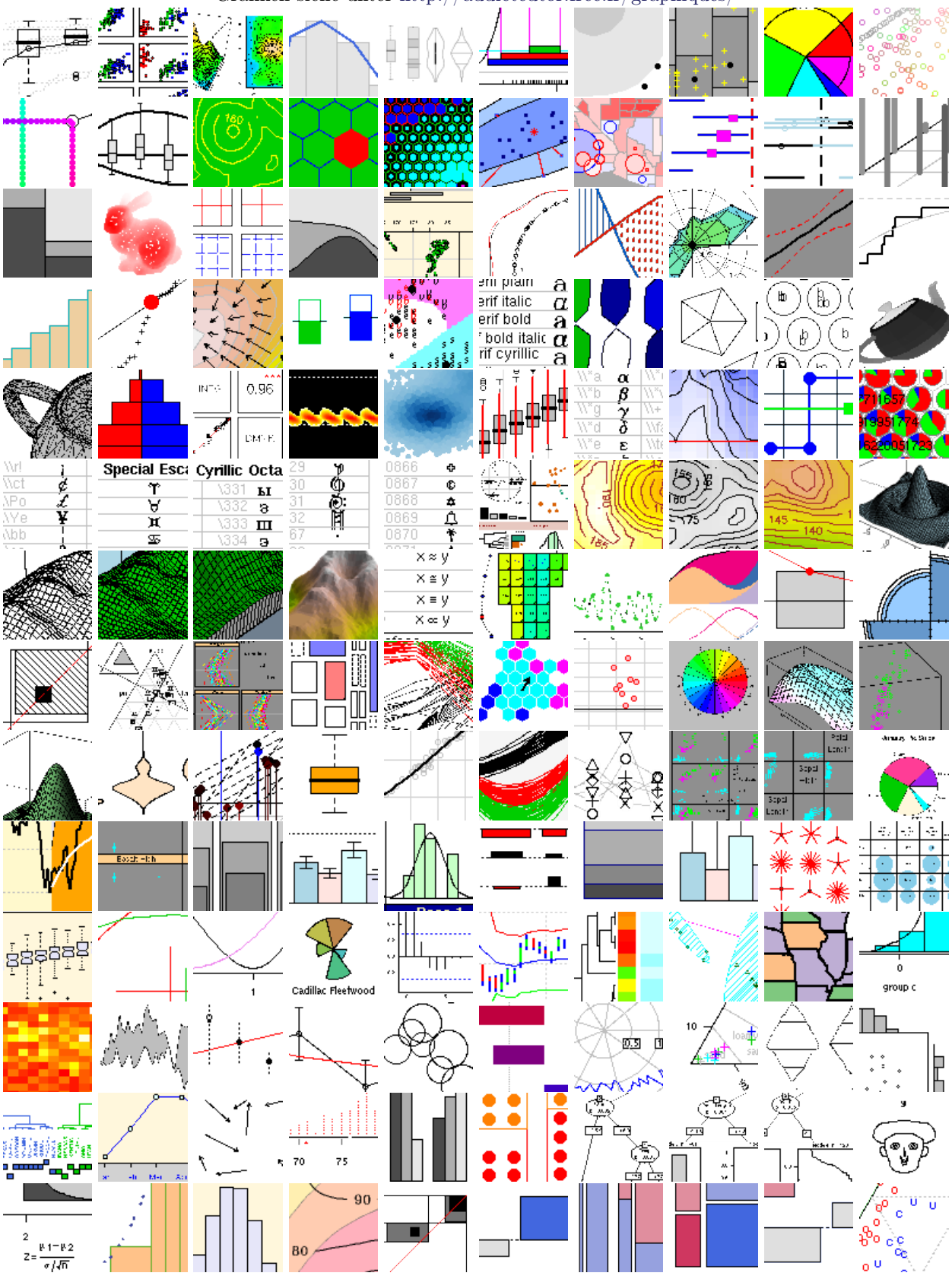
Dr. Andreas Plank (Biologe)

Version: 29. März 2010



FU Berlin
Inst. f. Geologische Wissenschaften
Fachbereich Palaeontologie
Malteser Str. 74 - 100
12249 Berlin
http://www.geo.fu-berlin.de/geol/fachrichtungen/pal/
http://www.chironomidaeproject.com

Grafiken siehe unter http://addictedtor.free.fr/graphiques/



Layout: LaTeX

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Funktionen

Benutzung des Skriptes

Dieses Skript ist als Hilfestellung für einen Kurs „Auswertung quantitativer Daten mittels statistischer und multivariater Verfahren“ entstanden und erhebt keinerlei Anspruch auf Vollständigkeit. Es soll sozusagen eine Referenz/Rezeptsammlung beim Umgang mit R darstellen. Der Benutzer sollte es bitte auch kritisch lesen, da es sicher den einen oder anderen Duckfehler enthält. Für Verbesserungsvorschläge und Hinweise auf Falsches bin ich immer dankbar. Veraltete Internetlinks können u.U. nicht aktualisiert werden; man benutze hierzu Archiv-Portale.

Am Anfang sei noch gesagt, daß die Beispiele i.d.R. aus diesem Skript kopiert werden können, es kann aber durchaus sein, daß z.B.: ein ’ nicht dem Akzentzeichen ' entspricht, das in R gebraucht wird und dann natürlich eine Fehlermeldung kommt.

Im Anhang auf Seite 199ff. findet sich auch eine 4 - seitige Kurz-Referenz (engl.) mit allen wichtigen Anweisungen von Tom Short (s. unter http://www.Rpad.org Version vom 2005-07-12) sowie ein Strukturbaum mit immer wieder häufig verwendeten Funktionen zum Zeichnen/Proportionieren von Grafiken.

Um sehr viel leichter mit R arbeiten zu können, dringend einen Editor mit Syntaxhervorhebung verwenden (Editoren s. auf Seite 135, z.B. Tinn-R).

1 Allgemeines

Eine gute, knappe (vielleicht besser lesbare?) Einführung kann man auch in der R-Kurzbeschreibung von Seyfang (2005) lesen. Ebenso ist es nützlich, die oft komlizierten statistischen Zusammenhänge zu visualisieren. Hierzu ist das Zusatzpaket `TeachingDemos` sehr zu empfehlen (Installation siehe auf Seite 10).

1.1 Hilfe

Hilfe

Die direkte Hilfe zu irgendwas erhält man durch ein vorangestelltes Fragezeichen `?` – für die Plotfunktion beispielsweise: `?plot` oder weitgefaßter mit `??plot`[1] zur Suche in allen vorhanden Paketen bzw. `help.search("plot")`. Das R-Hilfe-System arbeitet als HTML - Hilfe[2] die man im Menü bzw. unter Linux in: `/usr/lib/R/doc/html/index.html` findet. Mit den Pfeiltasten ↑ ↓ kann man schnell zu den letzten Befehlen zurückblättern und mit den Tasten Pos 1 und Ende läßt sich die Textmarke schnell an den Anfang der Zeile springen. So läßt sich flugs ein `?` vor die entsprechende Funktion setzen, egal wieviel gerade in der Konsolenzeile steht.

Beispiele

Visuelle Beispiele zu Funktionen, Grafiken, etc. lassen sich z.B. für 3D - plots so erzeugen:

```
example(plot) # Beispiele zur plot-Funktion ausführen
```

Jedoch sollte man R (manchmal) vorher dazu auffordern, sich die Bestätigung des Nutzers beim Neuzeichnen von Grafiken sicherheitshalber einzuholen, damit nicht alle Beispiele an einem vorbeirauschen. Dies geht mit der globalen Grafikeinstellung `par(...)` auf Seite 25: `par(ask=TRUE)`.

Demos

Alle zur Verfügung stehenden Demos zeigt man sich mit `demo()` an. Und nachfolgend z.B.:

```
demo() # alle geladenen/verfügbaren Demos anzeigen
demo(graphics) # R's Grafikmöglichkeiten
demo(image)    # bildähnlihe Grafiken
demo(persp)    # erweiterte persp() Beispiele (3D)
demo(plotmath) # mathematische Schreibweisen  auf Seite 41
```

`apropos()`

Hin und wieder sucht man nach einer Funktion, die bereits existiert, von der man aber den Namen nicht genau kennt oder wieder vergessen hat. Abhilfe schafft hier die Anweisung `apropos("")`, z.B.:

[1] seit Version +2.8
[2] Windows CHM-Hilfe Dateien sind in Version 2.10 leider nicht mehr verfügbar

```
apropos('plot') # sucht alles was '*plot*' enthält: Funktionen, benannte Variablen, Datenobjekte,...
# Suche mit regulären Ausdrücken
apropos('^plot') # sucht wie 'plot*', d.h. mit plot beginnend
apropos('plot$') # sucht wie '*plot', d.h. mit plot endend
```

1.2 Blitzstart

Abhängig vom Betriebssystem: Windows `RGui.exe` starten, unter Linux `R` in die Konsole tippen:

```
##############################
# Inhalt: wichtigste Anweisungen
# Datum:
# Zu tun:
##############################
options(prompt=" ") # beim Code kopieren: störendes '>' in ' ' verwandeln
setwd("Pfad/zu/meinem/Arbeitsverzeichnis") # Arbeitsverzeichnis festlegen
# require(...)     # Paket laden: gibt TRUE/FALSE zurück oder library(...)
# Daten einlesen:
  daten <- read.csv2(...) # Daten einlesen s. auf Seite 13
  # ---8<-----
  require(RODBC) # Zusatzpaket laden
  chExcel <- odbcConnectExcel("Exceldatei.xls") # Verbindung öffnen
  samples <- sqlFetch(chExcel, "Excel-Tabellenblatt") # Tabellenblatt holen
  odbcClose(chExcel) # Verbindung schließen
  # ---8<-----
  attach(samples)  # 'samples' in den Suchpfad aufnehmen: R findet dann auch Spalten- oder ↩
    Zeilennamen(!)
samples # Daten anschauen
str(daten)          # Datenstruktur eines Objektes 'daten': ganzzahlig, dezimal oder ↩
    Faktor-levels...
daten <- data.frame(zahlen= c(1,2,3)) # neues Datenobjekt
?data.frame()       # Hilfe aufrufen: Suche in geladenen Paketen
??data.frame()      # Hilfe aufrufen: Suche in ALLEN Paketen = HTML Such-Fkt.
fix(daten)          # Objekt namens 'daten' im Tabelleneditor bearbeiten
plot(daten)         # Objekt namens 'daten' zeichnen
summary(daten)      # Zusammenfassung
ls()                # alle Objekte der Session anzeigen
source("Funktion_Datei") # Anweisungen ausführen, Funktionen aus separater Datei lesen
# ...
save.image()        # Arbeitsdaten abspeichern als '.Rdata'
rm("daten")         # Objekte UNWIEDERRUFBAR löschen
# detach(package:ade4) # Paket ade4 wieder entfernen
# search()             # alle geladenen Pakete anzeigen bzw. searchpaths() alle Suchpfade
# q()                  # beenden
# dringend einen Editor mit Syntaxhervorhebung verwenden, und Funktionen aufheben.
# Editoren s. auf Seite 135
```

Am besten läßt sich arbeiten, wenn man eine zentrale Datei anlegt von der aus alle Aufgaben, Auswertungen abgefragt werden können. Durch einfaches Auskommentieren mit `#` kann man dann gewünschte Aufgaben rechnen lassen oder eben nicht. Die Auswertung wird übersichtlicher: man braucht nur die Quelldaten (*.xls, *.csv, *.mdb) und die entsprechenden R-Anweisungen in separaten Dateien. Ändern sich die Daten, läßt man flugs die Auswertung nochmal rechnen, indem die entsprechende verknüpfte Datei mit `source("/Pfad/zur/R_Datei.R")` von R ausgeführt wird. R liest dann die Datei `R_Datei.R` durch und macht das, was in der Datei `R_Datei.R` steht, doch werden alle „normalen“ Konsolenausgaben unterdrückt und müssen explizit angefordert werden (z.B. `print()` oder `cat()`). Optimalerweise: *Daten holen – zeichnen – Bild speichern*:

img: alle (generierten) Bilddateien

- **r:** alle R-Textdateien mit jeweils einem kompletten Ablauf nach Schema „F“ oder besser Schema „R“ ;-): *Daten holen – zeichnen – Bild speichern* (hier dann nach img)
  - **alleSkripte.R:** verwaltet alle R-Dateien durch `sourc(" Pfad/zur/Datei.R")` also z.B.: `sourc("species_secchi_depth_temp.R")`
  - **species_secchi_depth_temp.R:** enthält hier Secchi Tiefe, Tiefe und Temperatur z.B.
  - ...
- **tex:** hier liegt die LaTeX-R Vorlage für das Paket `Sweave` – hiermit läßt sich R-Code direkt in die LaTeX-Datei schreiben, dann mit `Sweave` prozessieren: Bilder, Abfragen, Tests etc. dann gleich in einer fertigen LaTeX-Datei ;-)
- **writer:** hier liegt die OO-Writer-R Vorlage für das Paket `odfWeave` – hiermit läßt sich R-Code direkt in die OO-Writer-Datei schreiben, dann mit `odfWeave` prozessieren: Bilder, Abfragen, Tests etc. dann gleich in einer fertigen OO-Writer-Datei ;-) (geht nicht mit MS-Word)

1.2.1 Vorlagendokumente

Nützlich ist auch ein **Vorlagendokument**, das wie folgt aussehen kann:

```
################################################
# Zeichenkodierung: utf8
# Datum: 10.9. 07
# Inhalt: ...
# diese Datei sollte mit source("Pfad/Dateiname.R") ausführbar sein
#   Hinweis: ausgeführt als source("Pfad/Dateiname.R") werden
#   alle Konsolenausgaben unterdrückt. Text zur Konsole dann mit
#   print() oder cat()
################################################
# Start R-Sitzung
# R                    # nur unter Linux
options(prompt=" ")    # Prompt-Zeichen auf ' ' setzen
# ?.. = Hilfestellung, z.B. ?plot
par(no.readonly=TRUE) -> paralt   # alte Grafikeinstellungen speichern
library(..)    # Paket laden
  (data <- read.table("clipboard")) # () = zusätzliches ausgeben
  str()        # Struktur ansehen
  attach()     # in den Suchpfad von R aufnehmen
  plot()       # zeichnen
  summary()    # Zusammenfassung
  #...         # Anweisungen
################################################
# Pfade+Namen abspeichern oder manuell über Menü
################################################
# Namen aller Verzeichnisse hier speichern
  basedir <- c(
    "/windows/D/Linux/Statistik/r/r_html/images/", # 1
    "../../../r_html/images/",                     # 2
    "/windows/D/Linux/Statistik/r/r_html/images/"  # 3
  )
################################################
# Breite Grafikfenster erzwingen
# siehe Benutzerfunktion grDeviceUserSize()  auf Seite 197
################################################
# Grafiken abspeichern s. auch auf Seite 9
# Größenangaben des Grafikfensters abfragen
  breite <- par("din")[1] # device in inch
  hoehe <- par("din")[2]  # device in inch
   # breite <- 12         # in inch
   # hoehe <- 7           # in inch
# Linux - PNG
```

```
dev.copy(bitmap,
  file=paste( # Dateiname hier angeben
    basedir[3],"Dateiname.png", # Pfad+Dateiname zusammenfügen
    sep="" # Kein Trennzeichen (' ' ist default)
  ),
  width=breite, # Breite explizit angeben
  height=hoehe, # Höhe explizit angeben
  type="png16m", # PNG als 16-Mio Farbenbild
  res=150 # Auflösung
)
dev.off() # Datei-'Schreibkanal' schließen, d.h. Datei schreiben
################################################
# Vektorgrafik EPS Linux, Windows, Apple
################################################
dev.copy(dev.copy2eps, # eine Grafiktyp Funktion, z.B.: pdf, png, jpeg, bmp
  file= 'Dateiname.eps' , # 1 Ebene hoch: '..\\filename.eps' oder '../filename.eps'
  width = par()$din[1],   # aktuelle Grafikbreite
  height = par()$din[2],  # aktuelle Grafikhöhe
  title = "Titel in EPS or PDF s"
)# Ende dev.copy()
dev.off() # Bild nun gespeichert
################################################
# Windows PNG Grafik als Kopie
################################################
#   dev.copy(png, # eine Grafiktyp Funktion
#     "..\\img\\H2O2006_pairs.png",
#     width  = par()$din[1], # aktuelle Breite Grafikfenster explizit angeben
#     height = par()$din[2]  # aktuelle Höhe Grafikfenster explizit angeben
#   )
#   # Grafik speichern Ende
#   dev.off()       # Grafikkanal/-device aus
################################################
# alte Grafikeinstellungen wieder auf Voreinstellungen
################################################
par(paralt)
################################################
# aufräumen
################################################
# detach(irgendetwasVorherAttachtes) # aus dem Suchpfad löschen
# rm(list=ls()) # löschen was die Funktion ls() liefert, d.h. ALLES !! - kein undo!
```

1.2.2 Übung: Erste Schritte

Starte eine R-Sitzung und gib folgendes – jeweils in schwarz – anfangs natürlich ohne das >-Promptzeichen ein (benutze auch die Tasten ↑ ↓ Pos 1 und Ende):

```
# R-Sitzung starten
> ls()  # anzeigen, welche Daten-Objekte geladen sind
 character(0) # also noch nichts
> options(prompt=" ") # Promptzeichen '>'zu Leerzeichen; besser f. Kopieren aus Konsole
 demo(persp)  # Demo zu 3-D Grafiken
# darauf achten was in der Konsole passiert!
ls()  # anzeigen, welche Daten-Objekte geladen sind
 [1] "fcol"      "fill"      "i1"        "i2"        "opar"      "rotsinc"
 [7] "sinc.exp"  "x"         "y"         "z"         "z0"        "zi"
# demo(persp) hat also eine ganze Reihe neuer Objekte erzeugt. Was könnte [1] und [7] bedeuten?
```

...Fortsetzung umseitig

```
ls()[1]  # Was ist der Unterschied zur Eingabe'ls()'?
 [1] "fcol"
ls()[2]  # Was ist der Unterschied zur Eingabe'ls()'?
 [1] "fill"
# aha - die eckige Klammer ist also eine Art Index-Klammer
rm(list=ls())  # rm(...) = remove - löscht UNWIDERRUFLICH Objekte
# 'ls()' liefert dabei das, was unwiderruflich gelöscht werden soll
?matrix  # direkte Hilfe zur Funktion 'matrix(...)' anschauen
matrix(1:12,3,4)  # Beispielmatrix
      [,1] [,2] [,3] [,4]
 [1,]    1    4    7   10
 [2,]    2    5    8   11
 [3,]    3    6    9   12
ls()  # anzeigen, welche Daten-Objekte geladen sind
 character(0) # keine Objekte da
m <- matrix(1:12,3,4)  # zurückblättern mit Taste ↑ und 'Pos 1' und 'm <-' hinzufügen
# Keine Ausgabe - warum?
ls()  # zurückblättern mit Taste ↑ und nochmal anzeigen, welche Daten-Objekte geladen sind
 [1] "m"   # aha - hier ist unsere Matrix
# die Ausgabe wurde also in 'm' hinein geschrieben und ist ab jetzt verfügbar!
# kleiner Tip: wenn ein Objekt mit 'name <- funktion(...)' erzeugt wird, kann
# man gleichzeitig eine Ausgabe bewirken, wenn man das Ganze nochmal klammert:
(m <- matrix(1:12,3,4))  # zurückblättern mit Taste ↑ und Klammer () dazu
# mit 'Pos 1' und 'Ende'-Taste
      [,1] [,2] [,3] [,4]
 [1,]    1    4    7   10
 [2,]    2    5    8   11
 [3,]    3    6    9   12
m  # Objekt anzeigen
      [,1] [,2] [,3] [,4]
 [1,]    1    4    7   10
 [2,]    2    5    8   11
 [3,]    3    6    9   12
m + 4  # 4 dazu addieren; Taste ↑ und '+ 4' eingeben
      [,1] [,2] [,3] [,4]
 [1,]    5    8   11   14
 [2,]    6    9   12   15
 [3,]    7   10   13   16
m  # Objekt anzeigen
      [,1] [,2] [,3] [,4]
 [1,]    1    4    7   10
 [2,]    2    5    8   11
 [3,]    3    6    9   12
# unsere Matrix ist also noch dieselbe geblieben!!!
m * 6  # 6 dazu multiplizieren
      [,1] [,2] [,3] [,4]
 [1,]    6   24   42   60
 [2,]   12   30   48   66
 [3,]   18   36   54   72
m  # Objekt anzeigen
      [,1] [,2] [,3] [,4]
 [1,]    1    4    7   10
 [2,]    2    5    8   11
 [3,]    3    6    9   12
# unsere Matrix ist also immer noch dieselbe geblieben!!!
```

...Fortsetzung umseitig

```
m <- m * 6  # 6 dazu multiplizieren + überschreiben!!
m  # Objekt anzeigen
     [,1] [,2] [,3] [,4]
[1,]    6   24   42   60
[2,]   12   30   48   66
[3,]   18   36   54   72
m : 6  # dividieren?
[1] 6
Warning message:
numerischer Ausdruck hat 12 Elemente: nur erstes wird genutzt in: m:6
# ':' hat wohl eine andere Bedeutung...
8:4  # ':' einfach mal ausprobieren
[1] 8 7 6 5 4
-8:4  # ':' einfach mal ausprobieren
[1] -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4
# aha eine Reihe wird also durch ': ' erzeugt
m / 8  # dividieren!
     [,1] [,2] [,3] [,4]
[1,] 0.75 3.00 5.25 7.50
[2,] 1.50 3.75 6.00 8.25
[3,] 2.25 4.50 6.75 9.00
m  # Objekt anzeigen
     [,1] [,2] [,3] [,4]
[1,]    6   24   42   60
[2,]   12   30   48   66
[3,]   18   36   54   72
# unsere Matrix ist also noch dieselbe seit dem Multiplizieren!!!
m / 8,5  # dividieren richtig?
Fehler: Syntaxfehler in Zeile "m/8,"
# Achtung Europäer: R ist aus Amerika .... also:
m / 8.5  # dividieren richtig!
          [,1]     [,2]     [,3]     [,4]
[1,] 0.7058824 2.823529 4.941176 7.058824
[2,] 1.4117647 3.529412 5.647059 7.764706
[3,] 2.1176471 4.235294 6.352941 8.470588
# Schaue mal aufmerksam die Ausgabe an, fällt da was auf? Ich meine die '[1,]'- und '[,4]'-Sachen?
# '[1,]' oder '[,4]' - könnte das was bedeuten?
m[1,]
[1]  6 24 42 60
m[,1]
[1]  6 12 18
m
     [,1] [,2] [,3] [,4]
[1,]    6   24   42   60
[2,]   12   30   48   66
[3,]   18   36   54   72
m[,2:3]
     [,1] [,2]
[1,]   24   42
[2,]   30   48
[3,]   36   54
# aha! Reihen und Spalten können so abgefragt werden
                                    einfacher Umgang mit Daten
data(airquality)  # R-eigene Daten laden: New York Air Quality Measurements
ls()  # anzeigen, welche Daten-Objekte geladen sind
[1] "airquality" "m"    # aha - unsere Matrix + Datensatz "airquality"
```

...Fortsetzung umseitig

```
str(airquality)  # Daten Struktur ansehen
 'data.frame':   153 obs. of  6 variables:
  $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
  $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
  $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
  $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
  $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
  $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
# aha: 6 Variablen, 153 Zeilen in einem Datenobjekt = data.frame()
# NA = not available (fehlende Werte)
# Was könnte 'int', 'num' bedeuten? Was heißt 'integer'?
# Könnte das $-Zeichen eine Bedeutung haben?
?airquality  # direkte Hilfe zum R-Datensatz anzeigen
                                    einzelne Variablen abfragen
airquality$Wind
  [1]  7.4  8.0 12.6 11.5 14.3 14.9  8.6 13.8 20.1  8.6  6.9  9.7  9.2 10.9 13.2
  [16] 11.5 12.0 18.4 11.5  9.7  9.7 16.6  9.7 12.0 16.6 14.9  8.0 12.0 14.9  5.7
  [31]  ....
# aha - also '$' greift auf Variablen zu oder untergeordnete Objekte
# Alternativen, um auf Unterobjekte, Spalten, Variablen in einem Objekt zuzugreifen sind:
airquality["Wind"]  # über [] eckige 'Index'-Klammern + Name in ""
# dasselbe, nur über die Spaltenzahl:
airquality[,3]  # über [,] eckige 'Index'-Klammern + Spaltenangabe nach ','
  [1]  7.4  8.0 12.6 11.5 14.3 14.9  8.6 13.8 20.1  8.6  6.9  9.7  9.2 10.9 13.2
  [16] 11.5 12.0 18.4 11.5  9.7  9.7 16.6  9.7 12.0 16.6 14.9  8.0 12.0 14.9  5.7
  [31]  ....
# allgemeiner: airquality[Reihe,Spalte]
# Wie gibt man Spalten 2 bis 4 an? Erinnern wir uns an ':'
airquality[,2:4]  # Spalten 2 bis 4; so einfach mittels ':'
     Solar.R Wind Temp
 1       190  7.4   67
 2       118  8.0   72
 3       149 12.6   74
 4       313 11.5   62
 5       ...  ...  ...
# Wie gibt man alle Spalten ohne die dritte an?
airquality[,-3]  # alle Spalten ohne dritte: einfach mittels '-' (minus-Zeichen)
     Ozone Solar.R Temp Month Day
 1      41     190   67     5   1
 2      36     118   72     5   2
 3      12     149   74     5   3
 4     ...     ...  ...    ..  ..
# Wie gibt man Spalten 1, 3 und 6 an?
airquality[,c(1, 3, 6)]  # Spalten 1, 3 und 6 durch Benutzen der Funktion c()
# 'c' steht für combine
          einfaches Zeichnen mit der Funktion plot() s.a. ‚‚Blasendiagramm‘‘-Bsp. auf Seite 54
plot(airquality)  # erzeugt eine pairs() - Grafik (s. auf Seite 53)
# Wie zeichnen wir nun einzelne Variablen? Erinnern wir uns an '$'
plot(airquality$Ozone~airquality$Temp)  # allgemein: plot(y~x) WICHTIG: Tilde ~
# übersichtlicher wird es nach ausführen von attach()
attach(airquality)  # Datensatz in den Suchpfad aufnehmen
par(no.readonly=TRUE) -> original  # alte Grafikeinstellungen speichern
layout(matrix(c(1, 1, 2, 3), 2, 2) -> m); m # spezielles layout:
      [,1] [,2]
 [1,]    1    2
 [2,]    1    3
```

...Fortsetzung umseitig

```
# Grafikfläche hat jetzt 3 Grafiken: li eine, re zwei übereinander s. auf Seite 29
plot(Ozone~Temp, # airquality$Ozone ist jetzt nicht mehr nötig, R weiß wo zu suchen ist
  col=rainbow(5)[Month-4], # 5 Regenbogenfarben zeichnen bezüglich Monat
  pch=16,  # Punkttyp gefüllt s. auf Seite 28
   # main="Luftdaten New York \n 1973", # Titelei - \n ist gleich Zeilenumbruch
  xlab="Temperatur (°F)",  # x-Achsenbeschriftung
  ylab="Ozon ppb" # y-Achsenbeschriftung
) # Funktion plot() hier zu Ende
rug(Ozone, side=2)  # gibt Anzahl der Beobachtungen als Randplot aus
```

☞ Zur Farbe: `rainbow(5)` liefert an sich nur eine Liste mit 5 Farben à la `"#FF0000FF" "#CCFF00FF"` usw, die intern fortlaufend indiziert werden. `rainbow(5)[1]` ist also die erste Farbe: `"#FF0000FF"` , `rainbow(5)[2]` zweite Farbe: `"#CCFF00FF"` usw. Damit die richtige Farbe zum richtigen Monat kommt (Monate 5, 6, 7, 8, 9), wird einfach in der eckigen Index-Klammer direkt nach `rainbow(5)` `'Month-4'` gerechnet. So haben wir jetzt nicht mehr eine Liste `5 5 5 5 .. 6 6 6 .. 7 7 7 ... 8 .. 9 9`, sondern `1 1 1 1 .. 2 2 2 .. 3 3 3 ... 4 .. 5 5` – 1 bis 5 – und in der eckigen Index-Klammer `[Month-4]` von `rainbow(5)` wird also für Monat 5 die Zahl 1 zurückgegeben – und damit die erste Farbe, für Monat 6 die Zahl 2 – und damit die zweite Farbe – usw.

```
                Daten übersichtlicher mit Funktion boxplot(...) s. auf Seite 48
par(mar=c(0,4.1,4.1,2.1))
# Rand reduzieren bottom, left, top, right; s. S. 26
boxplot(Ozone~Month, # Month ist hier die Variable nach der gruppiert wird
  col=rainbow(5),  # 5 Regenbogenfarben zeichnen bezüglich Monat
  xlab="",  # x-Achsenbeschriftung
  ylab="Ozon ppb",  # y-Achsenbeschriftung
   # main="Luftdaten New York \n 1973", # Titelei - \n ist gleich Zeilenumbruch
  notch=TRUE,  # Vertrauensintervall des Median
  xaxt = "n" # x-Achse ohne 'ticks' und 'labels'
) # Funktion boxplot() hier zu Ende
par(mar=c(5.1,4.1,0,2.1))  # Rand reduzieren s. auf Seite 26
boxplot(Temp~Month, # Month ist hier die Variable nach der gruppiert wird
  col=rainbow(5),  # 5 Regenbogenfarben zeichnen bezüglich Monat
  xlab="Monat",  # x-Achsenbeschriftung
  ylab="Temperatur (°F)",  # y-Achsenbeschriftung
   # main="Luftdaten New York \n 1973", # Titelei - \n ist gleich Zeilenumbruch
  notch=TRUE  # Vertrauensintervall des Median
) # Funktion boxplot() hier zu Ende
par(original)  # Grafikeinstellungen zurücksetzen oder Grafikfenster schließen und neuzeichnen
title("Luftdaten New York \n 1973")  # Titelei - \n ist gleich Zeilenumbruch
# Welche Grafiken sind informativer?
```

☞ Farben würde ich beim Boxplot weglassen, da sie an sich hier keine Funktion haben, denn die Gruppen stehen ja drunter.

1.2.3 Zeichen Werte Schalter

`<-` `->` Die Zuordnung von Variablen oder Objekten wird einfach mit `<-` oder `->` bewerkstelligt. So lassen sich folgende Daten der Variable `durchmesser` zuordnen:

```
durchmesser <- c(3.4, 3.6, 7.4, 8.7) # c() steht für combine
(durchmesser <- c(3.4, 3.6, 7.4, 8.7)) # (durchmesser...) bewirkt gleich eine Ausgabe
```

Die „Richtung“ ist dabei egal. Desweiteren gibt es noch die Zuweisungen mit `<<-` und `->>`, die hingegen globale sind, die also auch außerhalb von programmierten Funktionen funktionieren.

`.` `,` Der Unterschied zwischen `.` und `,` im Beispiel ist dabei zu beachten! Punkt für Dezimaltrennzeichen und Komma, um Argumente zu trennen.

`#` Kommentare kann man durch `#` hinzufügen. Alles was dahinter steht wird von R ignoriert.

`:` Will man die Zahlen 1 bis 4 eingeben, kann man dies verkürzt durch `1:4` machen.

`$` Geschachtelte Variablen lassen sich mit `$` ansprechen.

```
Variable$Untervariable
Objekt@Unterobjekt
```

Geschachtelte Datensätze lassen sich auch mit `[[Zahl]]` ansprechen (s. auch durch Eingabe von `?Extract`). `[[]]`

```
x <- cbind(a = 3, b = c(4:1, 2:5)) # 2 Spalten verbinden 3, 3, 3... & 4, 3, 2, ... 4, 5
dimnames(x)[[1]] <- letters[1:8]   # Buchstaben als Zeilennamen zuweisen
x # Daten anschauen
```

☞ `cbind()` steht für column bind

Für viele Funktionen gibt es Schalter, die alle mit `TRUE` an- und mit `FALSE` ausgestellt werden können. Kürzer geht das auch mit `T` oder `F`, z.B.: `TRUE` `FALSE`

```
read.table(meineDatei.txt, header = FALSE) # oder kurz
read.table(meineDatei.txt, header = F)
```

Spezielle Werte werden in R folgendermaßen ausgedrückt: `NA` undefiniert (missing data – not available), `Inf` Unendlich, `NaN` Not a number. `NA` `∞ Inf` `NaN`

NA Daten entfernen

```
data(airquality)  # R-eigene Daten laden
?airquality       # Hilfe anzeigen
names(airquality) # Variablen anschauen
o3temp <- cbind(airquality$Ozone, airquality$Temp) # Spalten zs.fassen
na.omit(o3temp)   # zeigt Daten ohne 'NA' an
```

NA durch 0 ersetzen

```
(temp <-matrix(runif(25), 5, 5)) # Zufallszahlen (0 bis 1) in einer Matrix
temp[temp < 0.1] <- NA # alles kleiner 0.1 wird NA
temp # Ausgabe erzeugen
temp[is.na(temp)] <- 0 # falls Wert gleich NA, dann 0 überschreiben
temp # Ausgabe erzeugen
```

1.3 Grafiken abspeichern

Unter Windows nur entsprechendes Menü anklicken und Dateityp auswählen oder mit rechter Maustaste (=Kontextmenü) über Grafik gehen und in Zwischenablage kopieren. Es werden auch automatisch die Grafikparameter übernommen. Für Windowsbenutzer ist EPS oder Metafile zu empfehlen, da das ein Vektorformat ist.

Um Grafiken plattformübergreifend abzuspeichen und zu nutzen empfiehlt sich das (speicherintensive) Vektorformat EPS. Im folgenden Beispiel ist die Benutzerfunktion `asking()` mit im Spiel:

Speichern mit Abfrage zum Verzeichnis.

```
paste("../pfad/zum/Verzeichnis/" -> dirToSave ,
  "Dateiname.eps", sep="") -> fileName
if(!file.exists(dirToSave)){ # Verzeichnis vorhanden?
  if(exists("asking",mode="function")){# falls Nutzer-Fkt vorhanden
    asking(
      paste("Verzeichnis '",dirToSave,"' erstellen?", sep=""),
      "Stop hier.",
      "Verzeichnis erstellt...")
    dir.create(dirToSave) # erstellt Verzeichnis
  } else
  stop("Falsches Verzeichnis angegeben! Vermutete unter: ",dirToSave)
}
dev.copy(dev.copy2eps, # muß Funktionsname sein
  file=fileName,        # Dateiname
  width = par()$din[1], # aktuelle Breite Grafikfenster
  height = par()$din[2],# aktuelle Höhe Grafikfenster
```

```
  title = "Titelei für PDF/EPS"
)
dev.off() # Grafik nun gespeichert
cat(fileName, "gespeichert...\n") # Info zur Konsole
```

Andere Bild-Formate sind auch über Funktionen speicherbar (z.B. UNIX/Linux):

```
##### PDF abspeichern
pdf("contour.pdf") # Dateityp und -name festlegen
##### Beispiel Konturenplot
library(graphics) # Paket laden
  data(volcano)   # R-eigene Daten laden
?volcano # Hilfe des Datensatzes anzeigen
contour(outer(x, x), # Konturenplot
  method = "edge",
  vfont = c("sans serif", "plain")
)
breite <- par("din")[1] # Breite des aktuellen Grafikfensters abfragen
hoehe <- par("din")[2]  # Höhe des aktuellen Grafikfensters abfragen
  # übernimmt aktuelle Einstellung d. Grafikfensters und speichert in 'breite' bzw. 'hoehe'
  # din = device in iches
##### Linux
dev.copy(bitmap,
  type="pdfwrite",        # Typen: "png16m", "jpg",
  res=150,                # Auflösung
  './pfad/zur/Datei.pdf') # aktuelles Unterverzeichnis './'
# breite <- 6; hoehe <- 4/3*breite # Verhältnis festlegen: inch oder px je nach Grafik
# dev.copy(postscript, '../../zwei/drüber.ps') # 2 Verzeichnisse hoch
# dev.copy(bitmap, type="png16m", 'Datei.png') # 16 Mio Farben
# dev.copy(bitmap, type="jpg", 'Datei.jpg')
# dev.copy(bitmap, type="pdfwrite", res=150, width=breite, hight=hoehe, 'Datei.pdf')
dev.off() # Device ("Laufwerk") wieder ausschalten: jetzt erst Grafik gespeichert
##### Windows
dev.copy(png,"..\\img\\H2O2006_pairs.png",
  width = par()$din[1], # aktuelle Breite Grafikfenster explizit angegeben
  height = par()$din[2] # aktuelle Höhe Grafikfenster explizit angegeben
)
# Grafik speichern Ende
dev.off() # Device ("Laufwerk") wieder ausschalten: jetzt erst Grafik gespeichert
detach(package:graphics) # Paket wieder entfernen
```

⚠ Vorsicht beim abspeichern: Dateien werden ohne Nachfrage überschrieben. Die Variante `dev.copy(bitmap, ...)` arbeitet wohl laut R-FAQ besser, dabei wird im Hintergrund Ghost Script verwendet.

1.4 Pakete laden, herunterladen, deinstallieren

Oftmals benötigt man ein neues statistisches Paket, um eine Prozedur zu rechnen. Dieses muß man entweder in R laden[3] im Menü mit: **Packages|Load packages...** oder neu installieren. Letzteres geht schnell durch einen Menü-Click in **Packages|Install package(s) from CRAN...**. Eher unter Linux oder für R-Skripte:

```
# chooseCRANmirror() # Server wählen
# Paket installieren
  install.packages("ade4", dep=TRUE) # versucht automatisches herunterladen incl. abh. Pakete
  install.packages("Simple",contriburl="http://www.math.csi.cuny.edu/Statistics/R/simpleR/")
  install.packages( # von lokaler Datei
    "/tmp/Rtmp5V10oS/downloaded_packages/rgdal_0.5-24.tar.gz",
    repos=NULL)
```

[3] Man kann auch `library(packagename)` eingeben, um es zu laden.

```
  # gespeichertes Paket unter Linux: als root außerhalb R mittels Konsole
  R CMD INSTALL /home/plankandreas/Desktop/tmp/R/Design_2.0-10.tar.gz
# Paket deinstallieren
  remove.packages("Rcmdr", lib="C:/Programme/R/rw2001/library")
# Paket aktualisieren
  update.packages(
    lib.loc="analogue",
    dependencies =TRUE # inclusive Abhängigkeiten
  )
```

Ein Paket laden kann man mit `library(paket)` oder `require(paket)` jedoch gibt es in einigen Fällen beim Laden sehr vieler Pakete Überschneidungen mit anderen Funktionen in anderen Paketen, so daß geladene Pakete wieder aus dem Suchpfad entfernt werden müssen mit `detach(package:name)`:

```
library(vegan) # Paket 'vegan' für ökologische Datenanalyse laden
# require(vegan) # alternatives Laden
  #... viele R-Anweisungen
detach(package:vegan) # Paket aus dem Suchpfad entfernen
```

Braucht man jedoch trotzdem zwei gleiche Funktionsnamen aus verschiedenen Paketen, kann man `paket::` eine Zuordnung herbeiführen:

```
stats::anova(...) # explizit die Funktion anova aus dem Paket 'stats' benutzen
```

1.5 Einfache Berechnungen

Mit R läßt sich natürlich auch schlichtweg als Taschenrechner nutzen (`?Arithmetic`) z.B:

```
pi/4*40⇒ π/4 · 40; sqrt(400)⇒ ∛400; 354^2 ⇒ 354²;log(100,10)⇒ log₁₀ 100; 3+5; 3*3 etc.
x = 13; y = 6
x+y  # 19
x-y  #  7
x*y  # 78
x/y  #  2.166667
x^y  # 4826809
x %% y # 1: ist gleich dem Rest beim Teilen (Modulo)
x %/% y # 2: ganzzahlig x/y
```

☞ Auch lassen sich so Objekte und Variablen in R verändern, z.B.: `eigeneDaten*3`. Desweiteren werden die Vergleichsoperatoren verwendet:

```
!  # Negation
&  # Und
|  # Oder
== # Gleich
!= # Ungleich
<  # Kleiner
>  # Größer
<= # Kleiner gleich
>= # Größer gleich
```

Siehe auch in der Hilfe mittels `?Syntax`.

1.6 Mathematische Funktionen

Eine ausführliche Liste an verfügbaren Funktionen ist:

```
  min(...) # Minimum
  max(...) # Maximum
  mean(...) # arithmetisches Mittel
  median(...) # Median
  sum(...), prod(...) # Summe, Produkt über Vektorelemente
  cumsum(...), cumprod(...) # kumulierte Summe, Produkt
  log(...) # natürlicher Logarithmus
  log10(...) # 10er Logarithmus
  exp(...) # Exponentialfunktion
  sqrt(...) # Wurzel
  abs(...) # Absolutbetrag
##trigonometrische Funktionen:
  sin(...), cos(...), tan(...), asin(...), acos(...), atan(...)
##hyperbolische Funktionen:
  sinh(...), cosh(...), tanh(...), asinh(...), acosh(...), atanh(...)
  gamma(...) # Gammafunktion
```

☞ siehe auch Kapitel Transformieren auf Seite 23.

2 Daten

2.1 Allgemeines

Daten verwalten ist das A und O. Deshalb bei großen Daten (immer) eine Datenbank anlegen, in der Tabellen über eindeutige Nummern, sogenannte ID's verknüpft sind. Freie Datenbanken: http://de.openoffice.org/ BASE, MySQL (mit Server http://www.apachefriends.org/de/xampp.html).

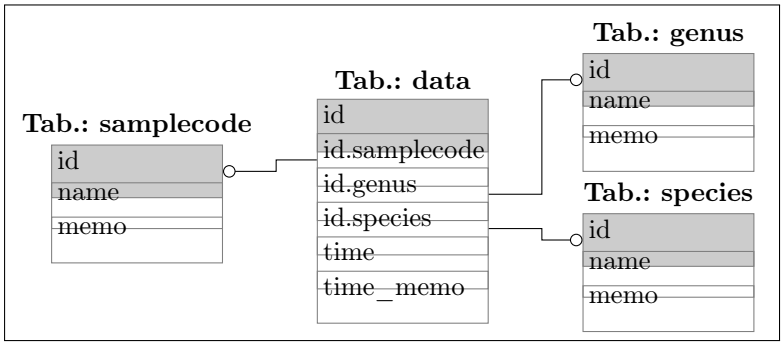


Abbildung 1: Datenbankschema mit verknüpften Tabellen, die über eindeutige, numerische `id`'s verknüpft sind. Abfragen wie: Zeige mir Arten, pH-Wert mit den und den Proben an, sind leicht und flexibel möglich. Abfragen dieser Art allgemein mit: `SELECT spalte1, spalte2 ... FROM tabelle WHERE tabelle.spalte=bedingung`

Falls der Datenumfang klein ist, dann immer(!) eine fortlaufende Tabelle anlegen mit Spalten als Variablen und neuen Daten als Zeilen. Diese Daten lassen sich dann mit Hilfe des Datenpilotes (in http://de.openoffice.org/ Calc) oder dem PIVOT Tabellen - Assistenten (Excel) schnell und flexibel(!) auswerten.

Für die Datenbankverbindung benutze man `RODBC` oder `RMySQL` (s. auf der nächsten Seite) und um Textdokumente mit R zu verbinden benutze man das Paket `odfWeave` oder die Funktion `Sweave()` aus dem Paket `utils` (S.133).

2.2 Grundlegendes in R

`setwd("")` `getwd()`

Arbeitsverzeichnis festlegen mit `setwd("")`. Anzeigen läßt es sich mit `getwd()`

```
setwd("C:/Media/Eigene Dateien/Dokumente/Praktikum/R")
# oder
setwd("C:\\Media\\Eigene Dateien\\Dokumente\\Praktikum\\R")
```

Um sich alle Dateien des Arbeitsverzeichnisses anzuzeigen gibt man den Befehl `dir()` ein. `dir()`

Löschen kann man ein Objekt mit `rm("")` für remove. Umgekehrt lassen sich alle Objekte durch `ls()` anzeigen. `rm("")`

```
ls() # alle Objekte anzeigen
rm(list=ls()[42:38]) # Objekte 38 bis 42 löschen
rm(list=ls()) # löscht alles - VORSICHT!
```

Die Struktur der Daten (ob numerische oder kategoriale Variablen) läßt sich durch `str(Objekt)` anzeigen. `str(...)`

Die Namen eines Objektes lassen sich mit `names(" Objekt")` ausgeben. `names("")`

2.3 Datenumgang

2.3.1 Eigene Daten/Funktionen einlesen

Daten können aus der Zwischenablage, aus diversen Programm-Dateien oder Datenbanken geladen werden. Bei Programm-Dateien kommen häufig folgende Funktionen zum Einsatz: `read.table(...)`, `read.csv2(...)`, `read.delim2(...)` oder `read.fwf(...)`[4] für das Eingelesen von `ASCII` (`*.txt`), Komma-getrennte Dateien (`*.csv`). Die `csv2`- und `delim2`- Varianten beziehen sich auf Komma-Dezimalzeichen und nicht auf Punkt-Dezimalzeichen, wie im englischsprachigen Bereich.

Für Programme, wie Excel oder Access existieren spezielle Pakete. SPSS Dateien lassen sich ebenfalls einlesen – benutze Package `foreign` mit der Funktion `read.spss(...)`. Am einfachsten ist das einlesen über die Zwischenablage aus Calc oder Excel. Dabei kann man bei den Einlesefunktionen wohl generell(?) die Option `"clipboard"` angeben, so daß man via kopieren & einfügen die Daten in R verfügbar hat[5].

Direktes Einlesen aus Excel oder Access via ODBC Treiber[6], ist mit dem Paket `RODBC` möglich. Das Paket `gdata` kann ebenfalls Excel Dateien lesen. Für MySQL gibt es das Paket `RMySQL`.

Aus Zwischenablage von OO-Calc oder Excel heraus

☞ darauf achten, was *gerade* in der Zwischenablage ist!!

```
# erst Daten markieren, Str+C drücken (=kopieren)
# nur für 1 Spalte
(daten <- read.csv2("clipboard")) # extra () bewirkt gleichzeitig eine Bildschirmausgabe
# für mehr als 2 Spalten
(daten <- read.delim2("clipboard"))
```

Komma getrennt – read.table(...)

```
arten <- read.table("arten.csv", sep=";", header=TRUE, dec=".", row.names=1)
```

Optionen `read.table(...)`: `sep=";"` Spaltentrenner, `header=T` Spaltenkopf: ja, `dec="."` Dezimaltrennzeichen; ☜ `row.names=1`, erste Spalte als Reihennamen benutzen (wenn das Dezimaltrennzeichen das Komma ist, dann muß `dec=","` angegeben werden – R akzeptiert nur den Punkt . für Zahlen.); `na.strings = "-"` fehlende Werte („not available“) in den Originaldaten: „-“ werden zu `NA` übersetzt.

Feste Breite – fixed width file: read.fwf(...)

```
arten <- read.table("arten.csv", sep=";", header=TRUE, dec=".", row.names=1)
  ff <- tempfile() # temporäre Datei erstellen
    read.fwf(ff, width=c(1,2,3))      # 1 23 456 \9 87 654
    read.fwf(ff, width=c(1,-2,3))     # 1 456 \9 654
    unlink(ff) # Datei löschen
    cat(file=ff, "123", "987654", sep="\n")
    read.fwf(ff, width=c(1,0, 2,3))      # 1 NA 23 NA \9 NA 87 654
```

[4] **fwf** steht für **f**ixed **w**idth **f**ormatted

[5] Unter Linux gibt es standardmäßig zwei Zwischenablagen: eine von der Maus-Auswahl und eine über Strg + C . `"clipboard"` scheint die Zwischenablage der Maus-Auswahl zu nutzen.

[6] muß vorher installiert sein; unter Linux funktioniert unixODBC nicht mit Excel oder Access

```
    unlink(ff) # Datei löschen
    cat(file=ff, "123456", "987654", sep="\n")
    read.fwf(ff, width=list(c(1,0, 2,3), c(2,2,2)))     # 1 NA 23 456 98 76 54
    unlink(ff) # Datei löschen
```

☞ `\n` bedeutet allgemein Zeilenumbruch.

Aus Excel/Access o.ä. Datenbanken

```
# Zugriff auf Daten in Excel
require(RODBC) # Zusatzpaket laden
chExcel <- odbcConnectExcel("Abiotik_China.xls") # Verbindung öffnen
  samples <- sqlFetch(chExcel, "samples") # Tabellenblatt 'samples' holen
odbcClose(chExcel) # Verbindung schließen
samples # Daten anschauen
# Zugriff auf Daten in Access
chAccess <- odbcConnectAccess("Daten.mdb") # Verbindung öffnen
  code <- sqlFetch(chAccess, "samplecode") # Abfrage/Tabelle 'samplecode' holen
odbcClose(chAccess) # Verbindung schließen
code # Daten anschauen
# detach(package:RODBC) # Paket eventuell wieder entfernen
```

☞ Analog können auch andere Datenquellen erschlossen werden, wie z.B. MySQL. (Optionen s. Hilfe). Vielleicht häufiger verwendet für Benutzer u. Passwort: `uid="benutzer"` sowie `pwd="topsecret"`.

Aus Excel mit `library(gdata)`

```
# Zugriff auf Daten in Excel
require(gdata) # Paket laden
read.xls("../src/Diatom-core-Leblanc2004JR01.xls",
  sheet = 2,       # welches Arbeitsblatt?
  header = TRUE,    # header vorhanden: Ja
  skip   =2         # 2 Zeilen am Anfang weglassen
)
detach(package:gdata) # Paket eventuell wieder entfernen
```

☞ `read.xls()` versteht auch die Argumente von `read.csv(...)` s. auf der vorherigen Seite.

MySQL – Verbindung

```
library(RMySQL) # Paket laden
verbindung <- dbConnect(MySQL(),
   user="root",
   password="secret",
   dbname="Daten",
   host="localhost")
dbListTables(verbindung)  # Tabellen auflisten
dbListFields(verbindung, "tab\_lake") # Tabellen Felder
sql=c(
  "SELECT * FROM `data`", # alles von Tab `data` auswählen
  "UPDATE `data` SET `counted` = 1 WHERE `id`=23" # Feld `counted` auf 1 setzen
)
abfragemysql <- dbSendQuery(verbindung, sql[1]) # alles von Tab. `data`
  data <- fetch(abfragemysql, n=64) # Daten in ein data.frame() schreiben
  str(data)                  # Struktur ansehen
dbSendQuery(verbindung, sql[2]) # UPDATE ausführen
dbDisconnect(verbindung) # Verbindung trennen
detach(package:RMySQL)    # Paket entfernen
```

Eigene Funktionen einlesen

```
# Funktion vorher in separate Datei speichern
source("../functions/plot.depth.en.R") # die Funktion "plot.depth.en" einlesen
```
☞ alle Konsolenausgaben werden unterdrückt und müssen mit `cat(...)` oder `print(...)` angefordert werden.

Es empfiehlt sich allgemein immer eingelesene Objekt mit `attach(...)` in den Suchpfad von R aufzunehmen, ☜ da die einzelnen Spalten/Variablen sich dadurch direkt mit ihrem Namen ansprechen lassen. Sonst müßte man zusätzlich in den Funktionen angeben: `Funktion(..., data="datenobjekt")` oder mit `$`-Zeichen darauf zugreifen.

In einem Beispiel heißt mein eingelesenes Datenobjekt `diatomeen` mit 72 Spalten. Ohne `attach(diatomeen)` müßte ich die Spalte `depth` mit `diatomeen$depth` ansprechen. Lasse ich hingegen vorher `attach(diatomeen)` durchführen, brauche ich *nur* `depth` schreiben, um etwas mit der Spalte `depth` zu rechnen, zeichnen usw. – erspart also Schreibarbeit. Wieder entfernen geht mit `detach(diatomeen)`.

2.3.2 Daten von R einlesen

In R enthalten sind auch Beispieldatensätze.

```
data(package = .packages(all.available = TRUE)) # Datensätze aller Pakete anzeigen
data() # Datensätze nur der geladenen Pakete
data(volcano) # Daten "volcano" laden
```

Hat man R-Anweisungen wie z. B. Funktionen separat in einer Datei, kann man sie mit `source(..)` auslesen. Ausgaben zur Konsole werden aber unterdrückt und müssen explizit mit `cat("Infotext zur Konsole")` oder `print(..)` angefordert werden.

```
source("/Pfad/zur/Datei.r") # Funktionen einlesen
```

Ähnlich arbeitet die Funktion `dget(..)` nur liest sie mit `dput(..)` geschriebene ASCII-R-Objekte ein.

```
dget("/Pfad/zur/Datei.txt") # ASCII-R-Objekt einlesen
```

2.3.3 Speichern/Auslesen

Es gibt viele Möglichkeiten Daten abzuspeichen (s. ausführliche Dokumentation „R Data Import/Export“ in der HTML-Hilfe). Hierbei gibt es oft Pakete, die Nicht-R-Daten lesen und meist auch schreiben können, wie das Extra-Paket `library(foreign)` z. B. auch DBF-Dateinen schreiben kann. Klassischerweise lassen sich Daten mit `write.table(...)`in eine `ASCII`-Datei auslesen, z.B.: als (`*.txt`) oder Komma-Getrennte Datei (`*.csv`).

```
write.table("arten" , file="output.csv", sep=";" , dec="," )
```
☞ Aufpassen mit Dezimaltrennzeichen; die Funktion schreibt direkt ins aktuelle Arbeitsverzeichnis

R-Objekte lassen sich natürlich in Form einer Sitzungs-Speicherung (workspace) sichern. Einzelne R-Daten-Objekte aber auch zusätzlich mit `dput()`

```
dput("meinRObjekt" , file="meinRObjekt.ascii", ) # einzelnes R-Objekt als ASCII Datei abspeichern
```

2.3.4 Eingeben/Erzeugen

Geht über Tabellenblatt oder direkte Eingabe.

Einlesen durch direkte Eingabe in Konsole

```
datensatz = scan() # R schaltet auf Datenscannen ein
4 6 46 9 27 # Daten eingeben auch mit Enter möglich
datensatz    # nochmal anzeigen
```

Eingabe durch Tabellenblatt

```
länge=2; breite=3
data.entry(länge, breite) # startet Eingabe über ein Tabellenblatt mit Startwerten 2 & 3
# Beispieldaten eingeben
# quit
# daten in Datenobjekt zusammenführen
(studie <- data.frame(länge, breite)) # Beispieldaten:
  länge breite
1     2      3
2    23     10
3    45      3
4     6      4
```

☞ Große Datenmengen lassen sich hingegen mit `data.frame()` zusammenfügen

Einen Daten Frame anhand von Variablen erzeugen illustriert folgendes Beispiel:

```
zeilen <- c("cond","d_samp","T_Jul_webgis","pH" ,"area_H2O" ,"P_Jan_webgis")
  nZeilen <- length(zeilen)  # Anzahl Zeilen
spalten <- c("R2","RMSE")# Namen Spalten
  nSpalten <- length(spalten)# Anzahl Spalten
daten <- data.frame(
  matrix(2, # der data.frame soll nur Zahlen haben hier "2"
    nZeilen,  # Anzahl Zeilen
    nSpalten, # Anzahl Spalten
    dimnames=list(
      zeilen, # rownames
      spalten # colnames
    )
  )
)
daten # Anzeigen
#              R2 RMSE
# cond          2    2
# d_samp        2    2
# T_Jul_webgis  2    2
# pH            2    2
# area_H2O      2    2
# P_Jan_webgis  2    2
```

. . . mit `for()` Schleifen (siehe z.B. auf Seite 20) lassen sich dann einzelne Werte als Ergebnisse in unser Daten Objekt `daten` schreiben. Zum Beispiel:

```
for(i in 1:nZeilen){
  daten[i,1] <- i # tue dies und das
}
daten
#              R2 RMSE
# cond          1    2
# d_samp        2    2
# T_Jul_webgis  3    2
# pH            4    2
# area_H2O      5    2
# P_Jan_webgis  6    2
```

. . . ist natürlich beliebig änderbar in `daten[indexZeile, IndexSpalte]` durch z.B. speichern der Ergebnisse aus Modellrechnungen.

Das Kombinieren von Daten in allen möglichen Varianten bewerkstelligt die Funktion `expand.grid()`:

```
expand.grid(
```

```
  "einsZwei"=c(1,2), # Zahlen
  "dreiVier"=c(3,4), # Zahlen
  "Faktoren"=c("height","low") # Faktoren
)
# einsZwei dreiVier Faktoren
#        1        3   height
#        2        3   height
#        1        4   height
#        2        4   height
#        1        3      low
#        2        3      low
#        1        4      low
#        2        4      low
```

2.3.5 Anzeigen

Gezieltes Zugreifen durch Angabe in eckigen Klammern

```
Spalten, Reihen
datensatz[,'laenge']  # zeigt von 'datensatz' nur Spalte 'laenge' an
datensatz[,2:4]  # zeigt nur Spalten 2-4 an
datensatz[2:5,'laenge']  # zeigt Spalte 'laenge' an und Zeilen 2-5
Level gezielt anzeigen
data(InsectSprays)  # R-eigene Daten laden
?InsectSprays  # Hilfe dazu anschauen
InsectSprays$count[levels(spray)=="B"]
```

2.3.6 Verändern

Spalten und Reihen eines Objektes lassen sich allgemein ansprechen und ändern durch: `Objekt[Reihe,Spalte]`.

Siehe auch Beispiel wie man `NA`-Werte in 0 wandelt 9

```
Reihe ändern
# die extra-runde Klammer bewirkt eine gleichzeitige Ausgabe
(Objekt <- matrix(1:36, 3,9)) # Matrize mit 3 Reihen & 9 Spalten
(Objektneu <- Objekt[-1,])  # ohne erste Reihe
(Objektneu <- Objekt[-1:3,])  # ohne erste bis dritte Reihe
☞ ändern/überschreiben
(Objektneu <- Objekt[,-1])  # ohne erste Spalte
(Objekt <- Objekt[,-1])  # überschreiben ohne erste Spalte
(Objektneu <- Objekt[,-1:3])  # ohne erste bis dritte Spalte
(Objektneu <- Objekt[,c(1,3,7:9)])  # nur erste, dritte, 7. - 9. Spalte
```

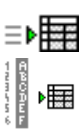
Spalten und Reihen + Funktionen anwenden dazu gibt es eine Ganze Reihe Funktionen, die alle irgendwie `..apply()` heißen. Gib auch ein `apropos("apply")`. Hier ein Beispiel um Spalten Mediane von Datenspalten abzuziehen. Es lassen sich bei allen `..apply()`–Funktionen natürlich auch andere Funktionen nutzen.

```
require(stats) # for median
?attitude # Hilfe, Info zu Datensatzt
# Umfrage unter 35 Angestellten einer großen Finanzorganisation
#    rating complaints privileges learning raises critical advance
# 1      43         51         30       39     61       92      45
# 2      63         64         51       54     63       73      47
```

```
# ....
med.att <- apply(
  X=attitude,
  MARGIN=2, # 1-Reihen, 2-Spalten
  FUN=median # Funktion median() anwenden
)# end apply()
sweep(data.matrix(attitude), 2, med.att)
# subtrahiert Spaltenmediane für jede Spalte
#      rating complaints privileges learning raises critical advance
#  [1,]  -22.5        -14      -21.5    -17.5   -2.5     14.5       4
#  [2,]   -2.5         -1       -0.5     -2.5   -0.5     -4.5       6
#  ....
```

Ein Beispiel um anhand einer Gruppierung Daten mit beliebigen Funktionen auszuwerten ist mit Hilfe von `tapply()` möglich:

```
my.df <- data.frame(# Daten erzeugen
  x=rnorm(20, 50, 10),
  group=factor(sort(rep(c("A", "B"), 10)))
)
my.df # anzeigen
#           x group
# 1  55.40921     A
# 2  45.83548     A
# 3  39.05827     A
# 4  44.04649     A
# ...
tapply(# eigene Funktion anwenden
  X     = my.df$x, # Werte
  INDEX = my.df$group, # Gruppierung
  FUN   = function(x){(x-mean(x))/sd(x)}
  # (Wert - arithmetisches Mittel) durch Standardabweichung
)
# $A
#  [1]  1.44944823 -0.31454498 -0.57431450  0.28354849 -1.04435583  1.20555464
#  [7] -0.45117046 -1.57519830 -0.05509169  1.07612441
#
# $B
#  [1]  1.01688021  1.00858347 -0.67579515  0.17505645  0.40726846 -2.24757792
#  [7]  0.82086939 -0.73193067  0.17913644  0.04750932
# oder einfach mit mean
tapply(# R Funktion anwenden
  X     = my.df$x, # Werte
  INDEX = my.df$group, # Gruppierung
  FUN   = mean # Mittelwert
)
#        A        B
# 49.58716 51.79856
```



Spalten-/Reihennamen zuweisen lassen sich u.a. mit den Funktionen `rownames(...)` oder mit der Option `row.names=1` beim Daten einlesen in `read.csv2(...)` oder dgl.:

rownames(...)/colnames(...)

```
arten <- data.frame( # Datenobjekt erzeugen
   spezies= paste(rep("Art",10), 1:10),  # 10x Art 1, Art 2, ...
   anzahl=sample(10) # 10 Zufallsdaten
) # data.frame() Ende
```

...Fortsetzung umseitig

```
arten  # anschauen
rownames(arten) <- arten[,1]  # 1. Spalte weist Artnamen als Reihennamen zu
(m <- matrix(1:10,10,1)) # neue Matrix -> matrix(Inhalt, Reihe, Spalte)
# und anzeigen durch extra Klammer ( )
rownames(m) <- rownames(m, do.NULL = FALSE, prefix = "Obs."); m  # Zeilenname mit Präfix
```

☞ analog funktioniert `colnames(...)`

```
colnames(geodaten)[1] <- "orte"   # ändert Spalte 1
names(geodaten)[1] <- "orte"   # ändert auch Spalte 1
rownames(geodaten)[15] <- "Deutschland"   # ändert Reihenname 15
m2 <- cbind(1,1:4)  # Daten generieren
m2  # Daten anschauen
colnames(m2, do.NULL = FALSE)  # Standardspaltennamen zuweisen
colnames(m2) <- c("x","Y")  # Namen zuweisen
m2  # Daten nocheinmal anschauen
```

dimnames(daten)[[1]]

```
daten <- cbind(a = 3, b = c(4:1, 2:5))  # Daten in 2 Spalten generieren
dimnames(daten)[[1]] <- letters[1:8]  # Buchstaben als Zeilennamen zuweisen
daten  # Daten anschauen
```

☞ die `[[1]]` steht für die Zeilennamen, `[[2]]` stünde für die Reihennamen

matrix(Inhalt, Reihen, Spalten) - mit Rehen & Spaltennamen

```
matrix(2,3,3, dimnames=list(c("R1", "R2","R3"), c("Sp1", "Sp2","Sp3")))
```

zuweisen beim Einlesen - read.table("...", ...)

```
arten <- read.table("arten.csv", sep=";", header=TRUE, dec=".", row.names=1)
```

☞ erste Spalte wird beim Dateneinlesen für die Benennung der Reihen verwendet. Die Reihennamen dürfen nicht doppelt vorkommen

Ausfüllen in Reihen/Spalten kann man auf vielerleiweise mit den Funktionen `c(...)` - combine, `rep(...)` - replicate, `letters[1:8]` - Buchstaben, `factor(...)` - Faktoren, `gl(...)` - generiere labels. Diese Funktionen lassen sich natürlich auch für Spalten mit `cbind(...)` und für Reihen mit `rbind(...)` verbinden.

Faktoren generieren `factor(...)`

```
factor(1:20, label="letter") # letter1, letter2, ...
factor(1:20, label="letter", levels=2:5) # <NA> , letter1, letter2 - nur levels 2-5
```

Vordefinierte Variablen: `LETTERS`, `letters`, `month.abb`, `month.name`

```
LETTERS[1:8] # A, B, C, ...
letters # a, b, c, ...
month.abb # Jan, Feb, Apr, ...
month.name # January, February, March, ...
format(ISOdate(2000, 1:12, 1), "%B") #  Januar, Februar, ...
```

Faktoren generieren: `gl(levels, Wdh., Anz./Namen)`

```
gl(2, 8, label = c("Control", "Treat")) # je 8 Control, Control, ..., Treat, Treat, ...
gl(2, 1, 20) # je 1 2 1 2...
gl(2, 2, 20) # je 1 1 2 2...
```

Dinge replizieren: `rep(x, wievielmal, ...)`

```
rep(1:4, 2) # 1, 2, 3, 4, 1, 2, 3, 4
rep(1:4, each = 2) # alle 2. -> Erhöhung: 1 1 2 2 3 3 ...
rep(1:4, c(2,2,2,2)) # dasselbe
rep(1:4, c(2,1,2,1)) # 1 1 2 3 3 4
rep(1:4, each = 2, len = 4)   # nur erste 4
rep(1:4, each = 2, len = 10)  # 8 ganzzahlige plus 2 einer
rep(1:4, each = 2, times = 3) # Länge 24, 3 komplette Wiederholungen: 1 1 2 2 3 3 4 4 1 1
# Liste replizieren
```

```
Liste <- list(Einstufung = 1:10, name = "Skala")
  rep(Liste, 2)
```

Sequenzen erzeugen: `seq(von, bis, ...)`

```
seq(0,1, length=11) # 11 Werte von 0 - 1
seq(1,9, by = 2)    # um 2 erhöht
seq(1,9, by = pi)    # um PI erhöht
seq(17) # dasselbe wie 1:17
```

Reihen/Spalten zusammenführen geht mit `cbind(...)` für Spalten, mit `rbind(...)` für Reihen und mit `merge(...)` um Datensätze zusammenzuführen:

```
tmp <- cbind( # Zahlenmaterial
  1:10, # Spalte 1
  c(     # Spalte 2
    seq(1, 1, length=5),
    seq(2, 2, length=5)
  )
)
daten <- merge("Candona candida",tmp) # fusionieren: zusätzliche Spalte 'Candona candida'
colnames(daten) <- c("spec","nr", "gruppe") # Spalten beschriften
daten # Objekt "daten" anschauen.
#               spec nr gruppe
#1  Candona candida  1      1
#2  Candona candida  2      1
#...
```

ifelse(...) - Anweisung

☞ Beispiel etwas komplexer (kann kopiert werden): es kommen die Funktionen '`for(var in seq) expr`' und '`ifelse(test, yes, no)`' vor (s.a. `?Control`). Das Beispiel an sich ist trivial, doch lassen sich mit `ifelse(...)` komplexer Anweisungen vornehmen, für die man sonst alles nochmal per Hand eintippen müßte.

```
for(zahl in 2:7)
{
   ifelse(zahl == 2,
      {  # Ausdruck für Bedingung wenn zahl = 2
      salin <- rnorm(20,zahl)  # 20 Zufallszahlen generieren mit dem Mittelwert 'zahl'
      tmp <- cbind(salin, zahl)  # Zahlenmaterial als Spalten verbinden
      daten <- merge("Candona candida",tmp)  # zusammenschmeißen
      colnames(daten) <- c("spec","sal","nr")  # Spalten beschriften
      cand.cand <- daten  # Daten in Objekt 'cand.cand' schreiben
      },
      {  # Ausdruck für Bedingung wenn zahl ≠ 2
      salin <- rnorm(20, zahl)  # 20 Zufallszahlen generieren mit dem Mittelwert 'zahl'
      tmp <- cbind(salin, zahl)  # Zahlenmaterial als Spalten verbinden
      daten <- merge("Candona candida",tmp)  # zusammenschmeißen
      colnames(daten) <- c("spec","sal","nr")  # Spalten beschriften
      cand.cand <- rbind(cand.cand, daten)  # Daten in Objekt 'cand.cand' schreiben, aber Reihen verbinden
      }
   )
}

boxplot(cand.cand$sal~cand.cand$nr,  # Daten
col="lightblue1",  # Farbe
notch=TRUE,  # 5%-Vertrauensintervall Median zeigen
ylab=expression(Salinität ~ ~group("[", g%.%l^-1, "]"))
# '~'gibt zusätzlich Platz; mathematische Ausdrücke: S. 41
) # Ende boxplot()

# Daten anschauen
str(cand.cand)  # Datenstruktur anschauen
```

Daten Splitten geht mit `split(...)`

```
daten.split <- split(daten, daten$artname)
```

☞ die Daten werden anhand der Untervariable `artname` gesplittet; in dem Objekt `daten.split` lassen sich alle Unterdatensätze mit z.B.: `daten.split$art.xy` ansprechen oder allgemein mit `daten.split`[[2]]. s. auch Boxplot - Beispiel auf Seite 49

Eine andere Funktion ist `aggregate(daten, gruppierung, FUN)`. Diese wird z.B. zum berechnen eines maximalen Fehlers (Bias) von einigen sogenannten Transfer-Modellen in der Paläontologie verwendet. Entlang eines Umweltgradiienten werden dazu 10 Intervalle gebildet. Das Intervall mit der größten Abweichung ist *maximum bias*:

Gruppierung mit `aggregate(daten, gruppierung, FUN)`

```
require(analogue) # Extra Paket für transfer-functions s.Frey und Deevey (1998)
# install.packages('analogue', dependencies=TRUE) # installieren falls nicht da
data(swapdiat) # swap Daten aus analogue package
?swapdiat # Hilfe zu Datensatz
data(swappH)   # swap Daten aus analogue package
# swap=Surface Waters Acidifcation Project
# Transfer Modell Weigthed Averaging
(modWA <- wa(swapdiat, swappH, deshrink = "inverse"))#
resid <- residuals(modWA) # Abweichungen/Residuen des Modells
gruppen <- cut(swappH, breaks = 10) # neue Gruppen für Datensatz
  aggregate(as.vector(resid), list(group = gruppen), mean)
# anhand von 'gruppen' arithmetisches Mittel berechnen: mean()
#          group           x
# 1  (4.33,4.62]  0.06637257
# 2  (4.62,4.91]  0.04999767
# 3   (4.91,5.2] -0.01928001
# 4    (5.2,5.5] -0.01472779
# 5   (5.5,5.79] -0.14858938
# 6  (5.79,6.08] -0.12676709
# 7  (6.08,6.38] -0.06348923
# 8  (6.38,6.67]  0.13828467
# 9  (6.67,6.96]  0.02071210
# 10 (6.96,7.25]  0.04416457
maxBias(resid, swappH) # -0.1485894
# in den 10 Intervallen ist -0.1485894 der maximale Bias
mean(resid) # 'average bias'
detach(package:analogue)# Paket wieder entfernen
```

Daten neu organisieren `stack(...)` und umordnen geht mit `unstack(...)`

```
data(PlantGrowth) # R- eigenen Datensatz laden
?PlantGrowth # Hilfe anzeigen lassen
PlantGrowth   # daten anschauen
  weight group
1 4.17   ctrl
2 5.58   ctrl
3 5.18   ctrl
... ... ...
unstack(PlantGrowth) # neu ordnen
  ctrl trt1 trt2
1 4.17 4.81 6.31
2 5.58 4.17 5.12
. ... .... ....
stack(PlantGrowth)
   values ind
1 4.17 weight
2 5.58 weight
3 .... .....
```

Daten als **Kreuztabellen** ausgeben kann mit den Funktionen `table()`, `ftable()`, `xtabs()` und `tapply()` bewerkstelligt werden.

```
warpbreaks   # Daten Wollesorten + Webfehler
    breaks wool tension replicate
 1      26    A       L         1
 2      30    A       L         2
 3      54    A       L         3
 4      25    A       L         4
xtabs(breaks ~  wool + tension, data = warpbreaks)
     tension
 wool   L   M   H
    A 401 216 221
    B 254 259 169
                              aus Liste Kreuztabelle berechnen - tapply()
(
  daten <- data.frame( # Daten erzeugen
    'zahlen'=c(1, 2, 2, 0.5, 2.5),
    'abc'=c("A", "A", "B", "B", "C"),
    'test'=c("Test 1", "Test 1", "Test 1", "Test 2", "Test 2"))
)
   zahlen abc   test
 1    1.0   A Test 1
 2    2.0   A Test 1
 3    2.0   B Test 1
 4    0.5   B Test 2
 5    2.5   C Test 2
attach(daten)  # in den Suchpfad aufnehmen
(tab <- tapply(zahlen, list(abc, test), sum))
# daten ausgeben durch zusätzliche Klammer (...)
   Test 1 Test 2
 A      3     NA
 B      2    0.5
 C     NA    2.5
 tab[is.na(tab)] <- 0 # NA Werte in 0 wandeln
tab # anschauen
   Test 1 Test 2
 A      3    0.0
 B      2    0.5
 C      0    2.5
detach(daten);rm(c(daten,tab))   # aus dem Suchpfad entfernen + löschen
```

Daten untereinander versetzten Mit der Funktion `lag(...)` lassen sich Daten untereinander verschieben, dies kann sinnvoll bei Zeitreihen angewendet werden.

```
x   <- ts(seq(10)^2) # Bsp. Daten
xxx <- cbind(x = x, lagx = lag(x), lag2x = lag(x,2)) # lag(...) kombiniert mit cbind(...)
xxx # Daten anschauen
```

2.3.7 Abfragen

Bereich min-max ..... `range(...)`
Bereich min-max erweitern ..... `extendrange(...)`
Buchstabenanzahl ..... `nchar(...)`
Differenz ..... `diff(...)`
Funktion zeigen (Code) ..... `showDefault(...)` `showDefault(plot.default)`
Funktion zeigen (Code) ..... `paket:::funktion.default`, `paket:::funktion`
Funktion zeigen (Code, auch unsichtbare) ..... `getAnywhere('funktion')`
Funktion Argumente zeigen ..... `args('')` `args(' plot.default')`
Funktion Argumente zeigen (auch unsichtbare) ..... `getAnywhere("funktion")`
Gemeinsamkeiten von `a, b` ..... `intersect(a, b)`
`a,b` (komplett) identisch ? ..... `setequal(a, b)`
Länge ..... `length(...)`
Namen ..... `names(...)`
Namen/Dimensionen ..... `dimnames(...)`

Listen/Objekte ..... ls(...)
NA - Werte ..... is.na(...)
Häufigkeiten ..... table(df)
Häufigkeiten ..... ftable(df)
Häufigkeiten ..... xtabs(df)
Index-Nr. einer Bedingung.....
..... which(is.na(meinObjekt))
Reihenanzahl ..... nrow(...)
Reihen + Spalten (Anzahl)..... dim(...)
Spaltenanzahl ..... ncol(...)

Spalten + Reihen (Anzahl)..... dim(...)
Spalten + Reihen (Statistiken).....
..... apply(data, SpaltenOderReihen, FUN,...)
Struktur ..... str(...)
Tabellenabfrage ..... tapply(df, grp, func)

Textbreite ..... strwidth(...)
Unterschied (was ist NUR in a!)..... setdiff(a, b)
vermischen von a, b..... union(a, b)
Zusammenfassung ..... summary(...)

Gemeinsamkeiten/Unterschiede von Werten

```
(eins <- c(sort(sample(1:20, 9)),NA))
  [1] 1 4 5 7 10 13 16 17 19 NA
(zwei <- c(sort(sample(3:23, 7)),NA))
  [1] 3 4 5 8 12 16 22 NA
union(eins, zwei)       # zusammen mischen: alles nur 1x
  [1] 1 4 5 7 10 13 16 17 19 NA 3 8 12 22
intersect(eins, zwei) # welche Werte sind gemeinsam?
  [1] 4 5 16 NA
setdiff(eins, zwei)     # welche Werte sind NUR in 'eins'
  [1] 1 7 10 13 17 19
setequal(eins, zwei)    # sind 'eins' und 'zwei' absolut identisch?
  FALSE
```

2.3.8 String „Helfer“

aufteilen ..... strsplit(x, split)
Ausgabe → String ..... toString(R-Ausgabe)
Ausgabe String „formatiert“ ..... sprintf(...)
in kleinbuchstaben ..... tolower(x)
in GROßBUCHSTABEN ..... toupper(x)

ersetzen ..... sub(pattern, replacement, x)
– " – ..... gsub(reg.pattern, replacement, x)
Teilstring . substr(was, Anfang, Länge) Teilstring in Vektor .....
... sapply(daten,substr, Argumente der Funktion ↩ substr)
zusammenfügen ..... paste('eins','zwei')

Zeichenketten formatiert ausgeben

```
sprintf("Zahl %d", 1:3) # "Zahl 1" "Zahl 2" "Zahl 3"
# f-float Zahl (double) mit 48 Stellen; normal 6 Stellen: "[-]mmm.ddd"
sprintf("%s ist gleich: %1.48f", "Pi", pi) # s string, f floating point number
# "Pi ist gleich: 3.14159265358979311599796346854418516159057617187 5"
sprintf("%s ist gleich: %e", "Pi", pi) # e+00 Version
# "Pi ist gleich: 3.141593e+00"

# mit Referenz zur Position in sprintf(...)
sprintf("zweitens:-|%2$1.0d| erstens mit Abstand:-|%1$10.5f| drittens:-|%3$1.0f|", pi, 2, 3)
# "zweitens:-|2| erstens mit Abstand:-|   3.14159| drittens:-|3|"
```

2.4 Transformieren

Mit der Funktion `scale(...)` lassen sich Daten transformieren. Siehe auch im Glossar unter Datentransformation.

scale(...)
base

```
glab <- read.table("http://folk.uio.no/ohammer/past/glaber.dat", header=TRUE, row.names=1)
# Datei mit Körpermaßen ordovizischer Trilobiten
```

...Fortsetzung umseitig

```
Zentrieren der Daten
center.glab <- scale(glab, center=TRUE, scale=FALSE)
center.glab  # Daten anschauen
Standardisierung der Daten
stand.glab <- scale(glab, center=TRUE, scale=TRUE)
stand.glab  # Daten anschauen
Normierung durch z.B. Maximum - lapply(...)
max.glab <- glab/lapply(glab, max)  # Spalten-Maxima verwenden
max.glab  # Daten anschauen
```

☞ mit `lapply(..., FUN)` lassen sich viele Funktionen auf Datensätze anwenden – `FUN` steht dabei für die entsprechende anzuwendende Funktion; siehe auch Mathematische Funktionen auf Seite 11

```
max.glab <- glab/apply(glab, 1, max)  # Zeilen-Maxima verwenden
max.glab  # Daten anschauen
```

☞ Optionen in `apply(...)`: 2. Argument: bei `apply(..., 1, FUN)` werden Reihen verwendet, bei `apply(..., 2, FUN)` werden Spalten verwendet – `FUN` steht dabei für die entsprechende anzuwendende Funktion

Daten in presence/absence Werte umwandeln geht mit `apply(data, col/row, func)`:

```
datenMatrix <- matrix(# datenMatrix speichern
  data = sample(# gib Zufallsprobe mit den Werten c(0,28,3)
    x=c(0,28,3),
    size=10,
    replace=TRUE
    ),# end sample()
    nrow=5,
    ncol=10
)# end matrix()
datenMatrix # anzeigen
#      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
# [1,]   28    3   28    3   28    3   28    3   28     3
# [2,]    3    0    3    0    3    0    3    0    3     0
# [3,]    3   28    3   28    3   28    3   28    3    28
# [4,]   28   28   28   28   28   28   28   28   28    28
# [5,]    3    0    3    0    3    0    3    0    3     0
apply(datenMatrix,
  2, # 1 = rows 2 = columns
  function(x) {ifelse(x >0, 1 , 0)} # wende diese Funktion auf (hier) Spalten an
  # wenn der Wert größer 0 gib 1 ansonsten 0 zurück
)
#      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
# [1,]    1    1    1    1    1    1    1    1    1     1
# [2,]    1    0    1    0    1    0    1    0    1     0
# [3,]    1    1    1    1    1    1    1    1    1     1
# [4,]    1    1    1    1    1    1    1    1    1     1
# [5,]    1    0    1    0    1    0    1    0    1     0
```

3 Grafik

☞ Hinweis: Es kann hilfreich sein Objekte, die berechnet oder gezeichnet werden sollen, mit dem Befehl `attach(meinedaten)` in den Suchpfad von R aufnehmen, da man sonst in vielen Funktion immer angeben muß woher die Daten kommen z.B. mit: `plot(..., data=meinedaten)`. Siehe auch Abschnitt 3.2.4 auf Seite 51. Das wichtigste (kann man kopieren):

```
test <- c(1:7)  # oder c(1, 2, 3, 4, 5, 6, 7) Beispielzahlen
zahlen <- c("eins","zwei","drei","vier","fünf","sechs","sieben")  # Beispielwörter
```

...Fortsetzung umseitig

```
plot(test,  # Zahlen 1 bis 7 plotten
  main="Beispieltitel",
  sub="Beispiel-Untertitel",
  xlim=c(0, 10),  # x-Achse 0 bis 10
  ylim=c( 0, 10),  # y-Achse 0 bis 10
  xlab="x-Achsenbeschriftung",
  ylab="y-Achsenbeschriftung",
   # log="xy" # log="x"nur x-Achse logarithmisch; analog für y
  cex=test,  # Punktgröße an Daten von 'test' binden ODER
   # cex=2,  # Punktgröße verdoppeln
  pch=16  # gefüllte Punkte zeichnen ODER
   # pch="a",  # Buchstabe 'a' zeichnen ODER
   # pch=test,  # Zahl von 'test' als Punkttypen
)
```

`type` - `"p","l","b", "c","o","h", "s","S","n"` s. auf Seite 52
`xlim=c(0, 14)` - x-Achsenskalierung
`ylim=c(0, 14)` - y-Achsenskalierung
`log` - log= "x"für x Achse (logarithmisch), "y"für y ; "xy"oder "yx" beide Achsen
`main` - Titelei \n ist gleich Zeilenumbruch
`sub` - Untertitel Grafik
`xlab` - x-Achsenbeschriftung
`ylab` - y-Achsenbeschriftung
`ann` - `TRUE, FALSE`: ob Titelei + Achsenbeschriftung sein soll
`axes` - `TRUE, FALSE`: beide Achsen ja/nein
`xaxt/yaxt` - `xaxt="n"` einzelne Achsen ausschalten
`frame.plot` - `TRUE, FALSE`: Rahmen um Grafik ja/nein
`panel.first` - Anweisung, die vor dem Zeichnen der Daten ausgeführt wird, z.B.: für Gitterlinien, Glättungskurven... (Beispiel: `grid()` auf Seite 34)
`panel.last` - dasselbe nur nach dem Zeichnen
`asp` - Höhen-Breitenverhältnis Grafik; siehe `?plot.window`

Allg. Grafikparameter: (s. auch Baum – Schema auf Seite 203f.)

`col` - Farbe s. auf Seite 45
`bg` - Hintergrundfarbe s. auf Seite 45
`pch` - Punkttyp s. auf Seite 28
`cex` - Vergrößerung/Verkleinerung Text, Symbole
`lty` - Linientyp s. auf Seite 27
`cex.main, col.lab, font.sub,...` - Titel, Labels, Untertitel ... s. auf dieser Seite und folgende

3.1 Einstellungen Zusätze

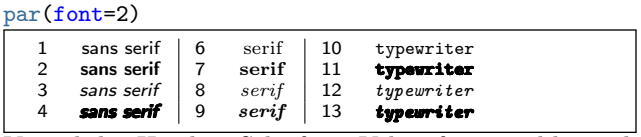
Allgemeine Einstellungen (*graphical parameters*) lassen sich für Grafiken mittels `par(...)` verändern (s. auch Baum – Schema S. 203f.). Oft empfiehlt es sich dabei mit `def.par <- par(no.readonly = TRUE)` die momentanen Einstellungen vorher zu speichern und mit `par(def.par)` entsprechend wieder zurückzusetzen.

☞ Nicht vergessen: alle Parameter, die über `par(...)` eingestellt werden können, können auch (meist) in den jeweiligen Funktionen, z.B. `plot(...)` angesprochen werden.

```
par(cex.axis=1.2)  # Achsen-Schriftskalierung geändert
```

☞ Um die Voreinstellungen leicht wieder rückgängig machen zu können, empfiehlt es sich die Einstellungen einem Objekt zuzuordnen `original <- par(cex.axis=1.2)` und später mit `par(original)` wieder zurückzusetzen.

Tabelle 1: Allgemeine Grafikeinstellungen, die mit `par(...)` vorgenommen werden können. Einzelne Werte lassen sich mit `par()$...` anzeigen. ☞ Die Einstellungen werden erst aktiv, wenn der Plot nocheinmal neu gezeichnet wird.

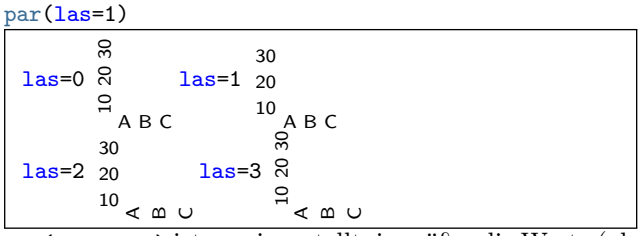
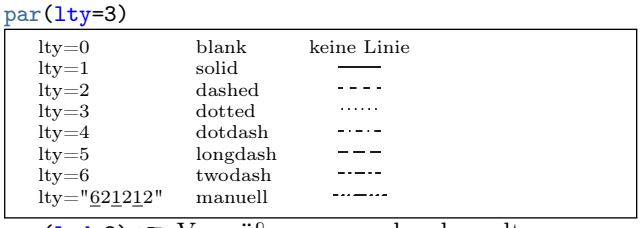
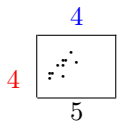
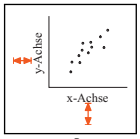
Beschreibung		Element	Erklärung/Beispiel
Hintergrundfarbe		`bg`	`par(bg="lightblue")`
Vordergrundfarbe		`fg`	`par(fg="lightblue")` ☞ ändert die Farbe für den Vordergrund - Plot
Skalierung	Schrift	`cex`	`par(cex=1.2)` ☞ bewirkt allg. Schriftvergrößerung aller Elemente
	Achse	`cex.axis`	`par(cex.axis=1.2)`
	Label	`cex.lab`	`par(cex.lab=1.2)`
	Titel	`cex.main`	`par(cex.main=1.2)`
	Untertitel	`cex.sub`	`par(cex.sub=1.2)`
Farben	allgemein	`col`	`par(col="lightblue")` [7]
	Achse	`col.axis`	`par(col.axis="lightblue")`
	Label	`col.lab`	`par(col.lab="lightblue")`
	Titel	`col.main`	`par(col.main="lightblue")`
	Untertitel	`col.sub`	`par(col.sub="lightblue")`
Schrift	allgemein (1)	`font`	`par(font=2)` 
Schrift	allgemein (2)	`family ="serif"` `family ="HersheySerif"`	Vorteil der Hershey Schriften: Vektorformat, d.h. sind beliebig skalierbar; Nachteil: z.B. pdf Datei kann sehr groß werden, dann besser in *.png umwandeln oder speichern.
Schrift	Achse	`font.axis`	`par(font.axis=5)`
Schrift	Label	`font.lab`	`par(font.lab=5)`
Schrift	Titel	`font.main`	`par(font.main=5)`
Schrift	Untertitel	`font.sub`	`par(font.sub=5)`

```
par(family="HersheySerif")
par(family="serif") # serif, sans, mono, symbol
----------------------------
cbind(## alle Hershey font Typen:
  Hershey$typeface[Hershey$allowed[,1]],
  Hershey$fontindex[Hershey$allowed[,2]]
)
"HersheySerif" plain, bold, italic, bold-italic
"HersheySans" plain, bold, italic, bold-italic
"HersheyScript" plain, bold
"HersheyGothicEnglish" plain
"HersheyGothicGerman" plain
"HersheyGothicItalian" plain
"HersheySymbol" plain, bold, italic, bold-italic
"HersheySansSymbol" plain, italic
```

... Fortsetzung umseitig

[7] Die extreme Vielfalt der Farben läßt sich mit `list(colors())` anzeigen. Dabei gibt es für jede Standardfarbe, wie `red`, `blue`, usw. jeweils von `blue1`...`blue4`, sowie eine `light`- als auch eine `dark`- Variante

☞ Die Einstellungen werden erst aktiv, wenn der Plot nocheinmal neu gezeichnet wird. Einzelne Werte lassen sich mit `par()$...` anzeigen.

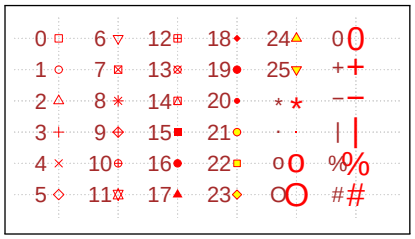
Beschreibung	Element	Erklärung/Beispiel
Teilstriche Achse	`lab`	`par(lab=c(5, 5, 7))` – ☞ ist Voreinstellung. `c(5, 5, 7)` gibt dabei die ungefähre Anzahl an Strichen für die Achsen an `c(5, 5, 7)` bedeutet `c(x, y, Labelgröße)`, siehe auch Kapitel Zusätze auf Seite 31
Teilstriche Beschriftung	`las`	`par(las=1)` 
Teilstriche Länge/Ausrichtung	`tcl`	`par(tcl=-0.5)` ist voreingestellt; je größer die Werte (also von negativ zu Null), desto kleiner werden sie; positive Werte bewirken eine Ausrichtung nach innen (!); `tcl=NA` setzt auf `tcl=-0.01` (gleich der Voreinstellung bei S-Plus)
Boxen - Typ	`bty`	`par(bty="l")` ☞ Box ähnelt dem Großbuchstaben, `bty="l"` ergibt also └ , `"7"` ┐ , `"o"` □ , `"c"` ⊂ , `"u"` ⊔ , `"]"` ⊐
Linientyp	`lty`	`par(lty=3)` 
Liniendicke	`lwd`	`par(lwd=2)` ☞ Vergrößerung um das doppelte
Linienenden	`lend`	0-"rounded" 1-"butt" 2-"square"
Linienübergang	`ljoin`	0-"round" 1-"bevel" 2-"mitre"
Rand zusätzlich	`mar`	`par(mar=c(5,4, 4,2)+0.1)` – Voreinstellung  ☞ Voreinstellung; `c(unten, links, oben, rechts)`; `mar=c(0,0,0,0)` würde nur die Plotfläche zeichnen.
Rand skalieren	`mex`	`par(mex=2)` ☞ Vergrößerung um das doppelte
Abstand äußerster Rand 	`oma, omi`	`par(oma=c(1,1,1,1))` ☞ `c(1,1,1,1)` steht für outer margin `c(unten, links, oben, rechts)` `oma` in Textzeileneinheiten, `omi` in inch
im äußersten Rand zeichnen	`xpd`	`par(xpd=TRUE)` ☞ wird z.B. `abline(...)` oder `text(...)` verwendet, kann man außerhalb des Plots zeichnen.

... Fortsetzung umseitig

☞ Die Einstellungen werden erst aktiv, wenn der Plot nocheinmal neu gezeichnet wird. Einzelne Werte lassen sich mit `par()$...` anzeigen.

Beschreibung	Element	Erklärung/Beispiel

mehrere Plots	`mfrow, mfcol`	`mfrow` steht für **m**ultiple **f**igures by **row**s, `mfcol` hingegen für **m**ultiple **f**igures by **col**umns
		`par(mfrow=c(1,2))`
		☞ ergibt 2 Reihen und 1 Spalte
		1
		2
		`par(mfrow=c(2,2))`
		☞ ergibt 2 Reihen und 2 Spalten
		1 2
		3 4
		allgemein: `par(mfrow=c(nReihen,nSpalten))`

Proportion Grafik	`fin`	`par(fin=c(5,6))` ☞ verändert die Grafikproportionen ob  oder 
Punkttypen	`pch`	`par(pch=16)` ☞ für einen vollen Punkt • – mit `pch="Text"` ließe sich auch Text statt der Punkte einzeichnen – mögliche Optionen sind: 
Achsen explizit an/aus	`xaxt, yaxt`	`xaxt="n"` keine; Standard: `"s"`
Achsentitel ausrichten ≡ ≡ ≡	`adj`	`par(adj=0.3)` ☞ alle Werte von 0 bis 1 sind erlaubt; 0 Text steht links, 1 Text steht rechts
Abstand Achsen, -beschriftung, Labelbeschriftung	`mgp`	`par(mgp=c(3,1,0))` ☞ ist Voreinstellung, wobei `mgp=c(3,1,0)` eine Verschiebung nach innen/außen (offset) sind `mgp=c( Achsenbeschriftung, Labelbeschriftung, Achsen)`
Achsenskalierung logarithmisch	`xlog, ylog`	`TRUE` oder `FALSE`
Achsenstriche: Anzahl/Beschriftung (min, max)	`xaxp, yaxp`	`xaxp=c(1, 50, 20)` bedeutet: von 1 bis 50 mit 20 Intervallen; analog `yaxp`
Zeichenregion (Ränder)	`usr`	`usr` Ausgabe durch `par()$usr` x-li, x-re, y-unten, y-oben

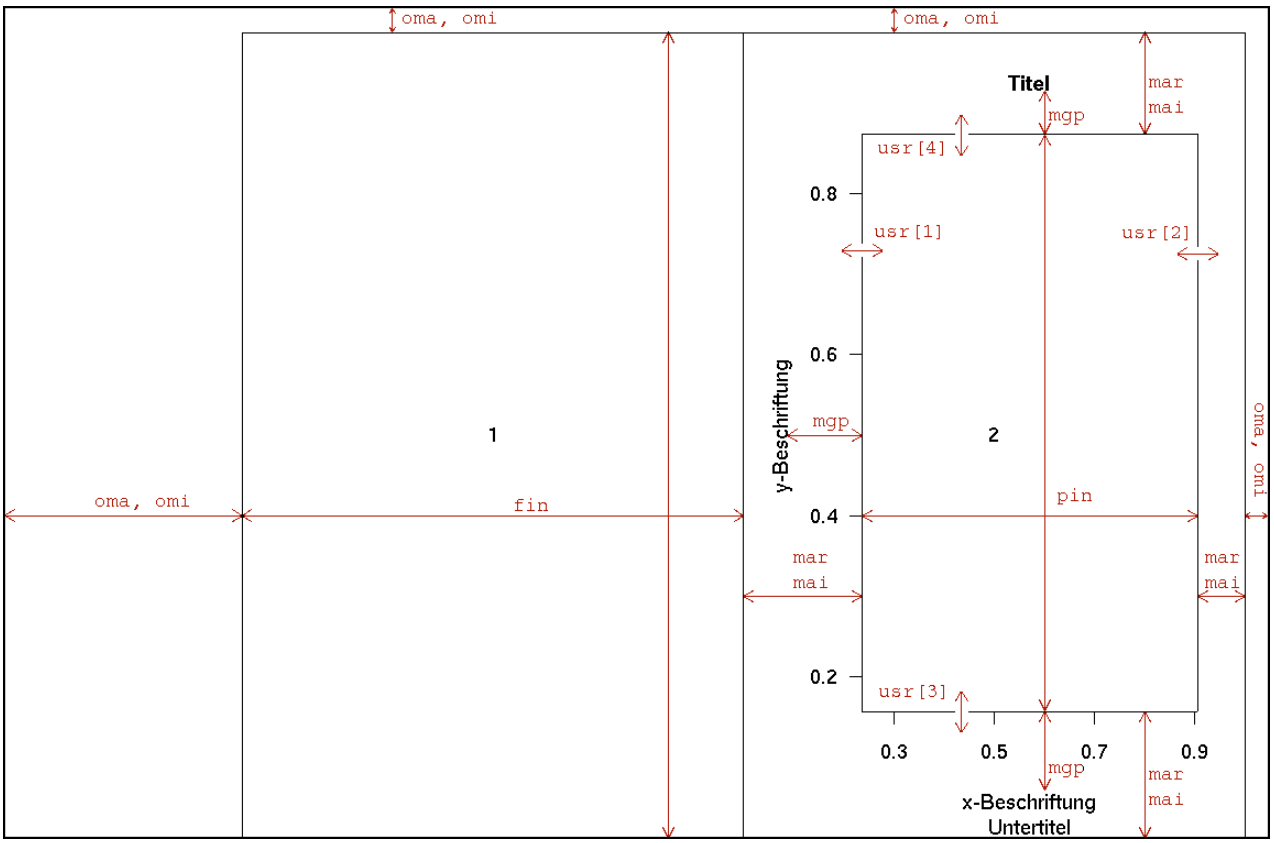


Abbildung 3: Einstellungen für Ränder in Grafiken für: `mar, mai, mgp, omi, oma`.
`mar=c(5,4,4,2)+0.1` gleich `mar=c(u,li,o,re)+0.1`, `mai=c(u,li,o,re)` Angabe in inch, `mgp=c(3,1,0)` gleich `c(Titel, Label, Achse)` in `mex`-Einheiten, `omi=c(0,0,0,0)` gleich `omi=c(u,li,o,re)`, `oma=c(0,0,0,0)` gleich `oma=c(u,li,o,re)` in Textzeileneinheiten.

3.1.1 Anordnen

Allgemein lassen sich Grafiken anordnen, indem man die *graphical parameters* `par(...)` mit dem Argument `mfrow=c(`*Reihe(n)*`,` *Spalte(n)*`)` ändert. `c(...)` ist dabei eine Funktion zum Kombinieren diverser Elemente, Objekte...

Siehe auch Beispiel, um mehrere Grafiken ineinander zu zeichnen auf Seite 32.

`par(mfrow=c(2,2))`[8] ergibt rechts stehende Beispielgrafik
Dasselbe ist durch folgende Eingabe möglich:

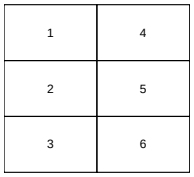
```
mat <- matrix(1:4, 2, 2) # 1:4 = 1 bis 4
layout(mat) # Layout festlegen
```

1	3
2	4

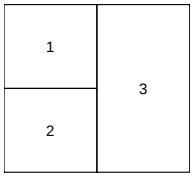
Komplizierte Anordnungen müssen mit den Befehlen `layout(...)`, `matrix(...)` und `c(...)` geändert werden. Hier einige Beispiele:

... Fortsetzung umseitig

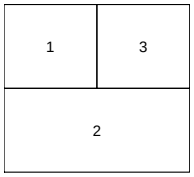
```
layout(matrix(1:6, 3, 2)) # 1:6 = 1 bis 6
```



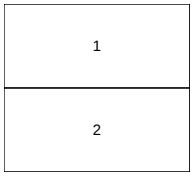

```
m <- layout(matrix(c(1, 1, 2, 3), 2, 2))
layout.show(m)
```



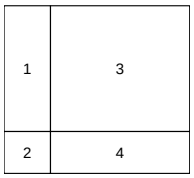

```
m <- layout(matrix(c(1, 2, 3, 2), 2, 2))
layout.show(m)
```




```
m <- layout(matrix(c(1, 2, 1, 2), 2, 2))
layout.show(m)
```

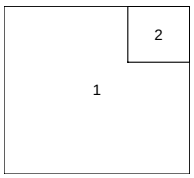



```
m <- layout(
  matrix(c(1, 2, 3, 4), 2, 2),
  widths=c(1,3),
  heights=c(3,1)
)
layout.show(m)
```

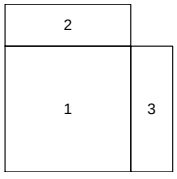


☞ `widths=c(1,3)` bezieht sich dabei auf die Größe der Spalten und `heights=c(3,1)` auf die der Reihen.

```
m <- layout(matrix(c(1,1,2,1),2,2), widths=c(2,1), heights=c(1,2))
layout.show(m)
```

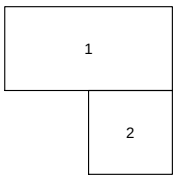



```
nf <- layout(
matrix( c(2,0,1,3), 2, 2, byrow=TRUE),
c(3,1),
c(1,3),
TRUE
)
layout.show(nf)
```

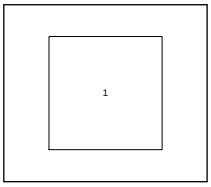


... Fortsetzung umseitig

```
nf <- layout(
matrix(c(1,1,0,2), 2, 2, byrow=TRUE),
respect=TRUE
)
layout.show(nf)
```




```
# Skalieren der Grafik
nf <- layout(matrix(1), widths=lcm(5), heights=lcm(5))
layout.show(nf) # mit 1cm×5 Länge & Breite
```



Anordnen lassen sich Grafiken auch „automatisch“ mit der Funktion `n2mfrow(...)`, so daß automatisch die richtige Anzahl an Grafiken auf ein „Blatt“ paßt. Im Beispiel kann man die Variable `wieviel` variieren:

automatisches Anordnen von Grafiken

```
wieviel <- 6 # Anzahl Grafiken
def.par <- par(no.readonly = TRUE) # Grafikeinstellungen 'speichern'
par(
  mfrow=(n2mfrow(wieviel)), # Grafikparameter
  mgp=c(3,-2,0), # -2 -> Labels nach innen
  mar=c(0,0,0,0) # kein Rand
)
for(zahl in 1:wieviel){# zahl läuft 1...wieviel durch
  plot(
    runif(zahl*10),runif(zahl*10), # Daten: Zufallszahlen mal 10
    main=paste("\n\n\n Grafik Nr.:\n",zahl), # Titelei
    tcl=0.5, # Achsenstriche nach innen
    col=rainbow(wieviel)[zahl], # Regenbogenfarben
    las=1, # Art der Achsenbeschriftung S.27
    xlab=, # x-Achsenbeschriftung
    ylab=, # y-Achsenbeschriftung
    pch=16 # Punkttyp gefüllt s. auf Seite 28
  )
}# end for
par(def.par) # Grafikeinstellung aus def.par lesen und zurücksetzen
```

3.1.2 Zusätze

Nachträglich einzeichnen bei mehreren Grafiken kann man z.B. Titel, Linien, Punkte auch in umgekehrter Reihenfolge, indem man den Grafikparameter `par(mfg)` benutzt.

```
par(no.readonly=TRUE) -> paralt # Grafikeinstellungen speichern
  par(mfrow=c(1,2)) # 1 Reihe 2 Spalten
  layout.show(n=2) # anzeigen
  x <- 1:20;y <- 1:20 # Zahlen 1...20
  plot(x,y) # Grafik 1
  plot(x,y) # Grafik 2
  par(xpd=TRUE) # außerhalb der Grafik zeichnen: an
  par(mfg=c(1,2,1,2) # Grafik 2 ansprechen
    title("Titel nachträglich\npar(mfg=pos[,1]) Grafik 2")
```

```
  par(mfg=c(1,1)) # Grafik 1 ansprechen
    title("Titel nachträglich\npar(mfg=pos[1,]) Grafik 1", col.main="red")
    lines(x,y, col="red") # Linien in Grafik 1
par(paralt) # Grafikeinstellungen wieder zurück
```

Mehrere Grafiken ineinander kann man mit Hilfe der Grafikeinstellung `par("fig"=c(`x_{li}`,` x_{re}`,` y_u`,` y_o`)` bewerkstelligen. Diese Angabe ist prozentual zu verstehen: `c(0.0, 0.5, 0.5, 1.0)` würde also links oben zeichnen.

```
n <- 1000       # Anzahl Punkte festlegen
x <- rt(n, df=10) # Student t mit 10 Freiheitsgraden
hist(x,         # Histogramm
  col = "light blue",   # Farbe
  probability = "TRUE", # als Wahrscheinlichkeiten ausgeben
  ylim = c(0, 1.2*max(density(x)$y))) # y-Wertebereich
lines(density(x), # Dichtekurve dazu
  col = "red",    # Farbe
  lwd = 3)        # Linienbreite
paralt <- par(    # Grafikeinstellungen
  fig = c(.02,.4,.5,.98),
   # fig: c(x-li, x-re, y-u, y-o) wo in der Grafik
  new = TRUE    # neue Grafik mit dazu?
)
qqnorm(x,       # Quantilplot
  xlab = "", ylab = "", main = "",
  axes = FALSE)
qqline(x, col = "red", lwd = 2) # Quantil-Linie
box(lwd=2)      # box + Liniendicke
par(paralt)     # Grafikeinstellungen wieder zurück
```

Teilstriche lassen sich mit `minor.tick(...)` aus dem package `Hmisc` einzeichnen aber auch manuell durch Benutzung der Funktion `pretty(...)`, welche „schöne“ Abstände in einem Bereich von-bis berechnet:

kleine Teilstriche mit Zusatzpaket Hmisc

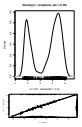
```
library(Hmisc) # Zusatzpaket laden; vorher installieren
plot(1:12) # zeichne: 1:12=1, 2, 3, 4, ....
minor.tick(ny=5, nx=0)
```

☞ zeichnet auf der y Achse 4 Teilstriche ein, d.h. 5 Teilintervalle

Teilstriche manuell mit `prettty(..)`

```
set.seed(0)   # rnorm() Anfangsbedingung: dadurch reproduzierbar
plot(rnorm(20) -> verteilung, # Zufallsdaten zeichnen
  main="Teilstriche", # Titelei
  bty ="n",   # keine Box
  axes=FALSE # keine Achsen
)
axis(1) # x-Achse
AnzInt <- 25 # 5 Intervalle
# y-Achse normal:
axis(2, at=pretty(verteilung)) # "schöne" Beschriftungs-Abstände
# y-Achse Teilstriche:
axis(2,
  labels=FALSE,    # keine Beschriftung
  tcl=-0.25,       # tick Länge verkleinern
  at=pretty(verteilung,AnzInt) # wo?
)
```

Zusätze für Marginalien wie z.B.: Anzahl der gezählten Daten als Striche lassen sich mit der Funktion `rug(...)` einzeichnen. Erweiterte Funktionen bietet das Paket `Hmisc` mit den Funktionen `scat1d(...)` und `histSpike(...)`.



```
data(faithful)
?faithful # Hilfe zu Datenset: Geysir Eruptionen, Yellowstone
  with(faithful, {
    plot(density(eruptions, bw=0.15))
    rug(eruptions)
    rug(jitter(eruptions, amount = .01), side = 3, col = "light blue")
  })
```

`with(data, {expr})` ist ein Ausdruck für R, der besagt, daß alles zusammen ausgeführt werden soll, `jitter(...)` verrauscht die Daten ein wenig.

Linien zeichnet man mit `abline(...)`[9] ein.

```
plot(1:12)
abline(3,4) # y0 =3 Ansteig=4, y = bx + a
abline(h=3)  # horiz. Linie bei y=3
abline(v=3)  # vert. Linie bei x=3
                    mit Benutzerfunktion line.labels.add(..) + Maus; s. S. 183
# Linie mit Text waagerecht-normal dazu
line.labels.add(1,text="Indikatoren", text.scale=0.7)
# Linie mit Text senkrecht
line.labels.add(1,text="Zone 1",
  text.scale=0.7,
  orientation="v",  # vertikal
  color="blue",  # Farbe Linie + Text
  color.bg="bisque",  # Farbe Hintergrund
  ypad=1  # etwas mehr Rand
)
# Nur Linien: Linientypen Farben
line.labels.add(3, # 3 Linien
  color=rainbow(3),  # 3 Regenbogenfarben
  l.width=c(1,8,1),  # Linienbreiten
  l.type=c("dashed","dotted","621212")
   # Linientypen 2x 'normal' 1x _ _ _ manuell; Typen s. auf Seite 27
)
```

☞ wird in `par(...)` (S. 26) der Schalter für `xpd=TRUE` (=„expand“) gesetzt, so lassen sich Linien über den Rand des Plots hinaus zeichnen.

Linien, Pfeile, Polygone, Rechtecke und solche Dinge lassen sich am besten mit der Maus platzieren mittels `locator(...)`:

```
                                  Beispiel Grafikelemente dazu:
plot(1:10, type="n") # "Zeichengrundlage"
  locator(2) -> wo  #  2 Positionen mit der Maus klicken
   #  Pfeile
    arrows(wo$x[1], wo$y[1], wo$x[2], wo$y[2], col="darkred")
   #  Rechteck
    locator(2) -> wo  #  2 Positionen mit der Maus klicken
    rect(wo$x[1], wo$y[1], wo$x[2], wo$y[2], border="darkred")
```

...Fortsetzung umseitig

[9] `abline(...)` ist eine Linie für die Gleichung $y = bx + a$: `abline(1,0)` zeichnet eine waagerechte Linie bei $y = 1$ und `abline(0,0.4)` zeichnet eine Linie mit dem Anstieg 0.4 ein und geht durch den Koordinatenursprung.

```
  #  kurze 2-Punkt-Linie
   locator(2) -> wo  #  2 Positionen mit der Maus klicken
   segments(wo$x[1], wo$y[1], wo$x[2], wo$y[2], col="darkred")
  #  Linien
   locator(5) -> wo  #  5 Positionen mit der Maus klicken
   lines(wo$x , wo$y, col="darkred")
  #  Polygon zeichnen
   locator(5) -> wo  #  5 Positionen mit der Maus klicken
   polygon(wo$x , wo$y, border="darkred")
  #  Kreis
   locator(1) -> wo  #  5 Positionen mit der Maus klicken
   symbols(wo$x , wo$y, circles=1, add=TRUE)  #  mit Ø 1
```

Gitternetzlinien mit der Funktion `grid()`; meist reicht nur das Aufrufen von `grid()` ohne irgendwelche Argumente:

```
plot(1:3) # Zeichengrundlage ': '=Zahlen 1 bis 3
grid(nx=NA, ny=5, lwd = 2)  # nur in y-Richtung (waagerecht)
                                    Beispiel innerhalb von plot(...)
plot(runif(1000),  # 1000 Zufallszahlen
  pch=16,  # Punkttyp gefüllt; Typen s. auf Seite 28
  cex=2,  # Punktgröße 2-fach
  panel.first= grid(NULL, NA),  # zuerst
  panel.last = grid(NA,NULL, col="red") # zuletzt
)
# Randtext dazu
mtext("zuerst gezeichnet",
  1,  # 1, 2, 3, 4 = bottom, left, top, right
  col="lightgrey",  # Farbe
  adj=0.2,  # Ausrichtung 0...1 = links...rechts
  line=2  # 2 Zeileneinheiten weg von Achse
)
mtext("zuletzt gezeichnet", 1,col="red", adj=0.2, line=3)
```

Droplines für Datengrafiken geht mit `points(..., type="h")` für x-Droplines und für y-Droplines mit Hilfe der selbstgeschriebenen Funktion `droplines.y(...)`:

```
                              selbst geschriebene Funktion dropline.y(..)
dropline.y <- function(x, y , direction="fromleft", ...){
  yaxis.left <- cbind(rep(par("usr")[1], length(x)), x)
  yaxis.right <- cbind(rep(par("usr")[2], length(x)), x)
  point.end <- cbind(y, y)
  for(i in 1:length(y)){ # für 'i'von 1 bis Länge/Anzahl von y mache...
    switch(direction,
      fromleft = lines(yaxis.left[i,],  point.end[i,],...),
       # '...'  -> Argumente zu lines(...)
      fromright = lines(yaxis.right[i,],  point.end[i,],...)
       # '...'  -> Argumente zu lines(...)
    )
  }
}
```

...Fortsetzung umseitig

```
# generelle Grafikeinstellungen (par() s. auf Seite 26) speichern
par(ask=TRUE,  # nachfragen ja
  las=1,  # (l)abel (as)signment
  mar=c(1,1,1,1)*4
) -> alteEinstellung
runif(10) -> x  # Zufallsdaten
runif(10) -> y  # Zufallsdaten
plot(x,y, type="p", main="y-droplines mit\ndropline.y(...)")  # nur Punkte
  dropline.y(x,y, lty="dashed")  # y-droplines
```

```
plot(x,y, type="p",   # nur Punkte
  main="y-droplines mit\ndropline.y(...,direction=\"fromright\")",  # Titelei
  yaxt="n"
  )
  axis(4)  # Achse rechts dazu
  dropline.y(x,y, lty="dashed",direction="fromright")  # y-droplines
```

```
plot(x,y, type="p", main="x-droplines mit\npoints(..., type=\"h\")")  # nur Punkte
  points(x,y, type="h", lty="dashed")  # x-droplines
par(alteEinstellung)  # wieder herstellen
```

Achsen/Labels zusätzlich lassen sich zum einen mit `axis(...)` einzeichnen:

```
par(ask=T)  # Nachfragen vorm Neuzeichnen
example(axis)  # Beispiel anzeigen
```

☞ `axis(1,...)` zeichnet x-Achse unten, 2 3 und 4 dann links oben rechts. Mit `axis(.., labels=c(10,15,20,40))` lassen sich z.B. Labels neu vergeben

zum anderen mit `overplot(...)` aus dem Paket `gplots`[10]. Siehe auch Bsp. um Achsenbeschriftung zu rotieren auf Seite 37.

```
library(gplots)  # Paket laden
data(rtPCR)  # Daten laden[11]
?rtPCR  # Hilfe dazu
overplot(RQ ~Conc..ug.ml. | Test.Substance,  # allg. Form: y~x|z
   data=rtPCR,  # Datenbezug
   subset=Detector=="ProbeType 7"& Conc..ug.ml. > 0,  # Unterdatenstatz: 'Detector' mit
       # entsprechenden Bedingungen
   same.scale=TRUE,  # selbe Achsenskalierung
   log="xy",  # welche Achsen logarithmisch
   f=3/4,  # Glättungsparamter von Funktion lowess(...)
   main="Detector=ProbeType 7",  # Titelei
   xlab="Concentration (ug/ml)",  # x-Achsenbeschriftung
   ylab="Relative Gene Quantification"  # y-Achsenbeschriftung
)
detach(package:gplots)  # Paket wieder loswerden
```

Achsenbrüche lassen sich mit `axis.break(...)` aus dem Paket `plotrix` zeichnen:

```
                                Achsenunterbrechung - axis.break(...)
library(plotrix)  # Paket laden
plot(3:10)  # trivialer Beispielplot 3-10
```

...Fortsetzung umseitig

[10] früher `gregmisc`

[11] Daten zu Teratogenität = chem. Potential von Substanzen, Noxen bzw. Fehlbildungen zu induzieren

```
# Voreinstellung von axis.break()
axis.break()  # Achsenumbruch
axis.break(2, 2.9,style="zigzag")
```

☞ `axis.break(1,...)` für x-Achse und `axis.break(2,...)` für y-Achse. Mögliche Stile: `"zigzag"` und `"slash"`. Noch ein Hinweis: der Umbruch wird immer nur auf die Achse gezeichnet. D.h. die Daten werden nicht neu gezeichnet. Fällt der Achsenbruch mitten in die Daten, muß man selbst für ein Neuzeichnen sorgen wie in folgendem Beispiel:

gap.plot()

```
twogrp <- c(rnorm(10)+4, rnorm(10)+20)  # 2 Gruppen Zufallsdaten
gap.plot(twogrp,
gap=c(8,15), # Lücke von .. bis
xlab="Index", ylab="Gruppen Werte",  # Achsenbeschriftung
main="2 Gruppen separiert durch ein Spalt (gap)" # Titel
)
```

Achsenunterbrechung mit Neuzeichnen der Daten

```
library(plotrix)  # Paket laden, falls noch nicht geschehen
data(bnr, package="pastecs")  # bentischen Datensatz laden
bnr.break <- bnr[,17]  # Daten kopieren
help("bnr", package="pastecs")  # Hilfe anzeigen zum Datensatz
for(i in 1:length(bnr.break))  # length() zählt die Anz. der Daten
  {if(bnr.break[i] > 200)  # wenn größer 200, dann 100 abziehen
    {bnr.break[i] <- bnr.break[i]-100}
    {bnr.break[i] <- bnr.break[i]}}
plot.ts(c(0,bnr.break,0),  # c(...) damit polygon() von y=0 zeichnet
  type="n",  # Plot ohne Inhalt aber mit Proportionen
  frame=TRUE,  # kann ein und ausgestellt werden
  ylab = "Abundanz",  # Beschriftung y-Achse
  xlab = "Stationen",  # Beschriftung x-Achse
  main = "marin-benthischer Datensatz \"bnr\"\n mit Achsenbruch",  # Titel
  axes=F,  # ohne Achsen
  ylim=c(0,300)  # y-Bereich festlegen
)
polygon(c(0,bnr.break,0), col="grey")  # Daten zeichnen
axis(2, labels=c(0,50,100,150,200,350,400))  # 2-> y-Achse mit neuen Labels
axis(1)  # 1 -> x-Achse zeichnen
axis.break(2,210)  # Achsenbruch
rect(0,208,length(bnr.break),215,col="white", border=F)  # weißes Rechteck
detach(package:plotrix)  # Paket wieder entfernen
```

Text zusätzlich

Text zusätzlich läßt sich mit `text(...)` in der Grafik platzieren.

```
text(4, 8.4, "Zusätzlicher Text", cex = .75, srt=20)
```

☞ x y Position in Grafik: (4, 8.4);`cex=.75` Schriftverkleinerung; `srt=20` Rotation um 20; wird in `par(...)` der Schalter für `xpd=T` gesetzt, so läßt sich über den Rand des Plots hinaus Text schreiben.

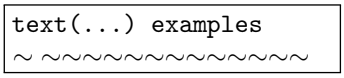
☞ mit der Maus platzieren mittels `locator()`:

```
text(locator(1)$x, locator(1)$y,"hallo")  # 1 Punkt lokalisieren
text(locator(1)$x,locator(1)$y, substitute(nicht~kursiv~italic(kursiv)))  # kursiv + normal
```

☞ ebenso gibt es `underline(...)` und `bolditalic(...)`. `plain(..)` ist normaler Text (Voreinstellung)

Ein Zeilenumbruch läßt sich durch `\n` erzeugen. So ergibt der Diagrammtitel `main` z.B.:

```
main = "text(...) examples\n~~~~~~~~~~~~~~~"
```



Text automatisch beschriften automatisch an Punkte beschriften möglichst ohne Überlappung: `example(thigmophobe.labels)`

```
x <- rnorm(20)  # x-Daten
y <- rnorm(20)  # y-Daten
names(x) <- 1:20  # Namen
plot(x=x,y=y,main="Test thigmophobe.labels")
(zeilen.namen <- paste("Index",names(x)))
thigmophobe.labels(x,y,
  zeilen.namen,  # Text
  col=c(2:6,8:12),  # Farben
  cex=0.6  # Größe
)
```

Für Text "n=34" s. Beispiel Boxplot auf Seite 50. Soll der Text abhängig von den Daten unterschiedliche Farben haben, verwende die Funktion `ifelse(Prüfung, dann, ansonsten)`:

Text Text Text
Text Text Text
Text Text Text
Text Text Text

```
text(14, n[row(as.matrix(n)),1], paste("n=", n[row(as.matrix(n)),2]), col=ifelse(
n[row(as.matrix(n)),2]>=200, "black", "gray"))
```

☞ Im Beispiel enthält die Spalte 2 von `n` die entsprechenden Daten, `n[,]` gibt an `n[Reihe , Spalte]`; `n[row(as.matrix(n)),2]` macht eine Matrix aus den Daten, um die entsprechende *richtige* Zeile anzusprechen. Im Beispiel wird der Text grau, wenn die Daten kleiner 200 sind ansonsten ist der Text schwarz. Dies Beispiel läßt sich auch für Datenpunkte o.ä. anwenden

Textplot - textplot(...) Paket gplots

```
data(eurodist)  # Datensatz Europäischer Städte
?eurodist  # Hilfe dazu anzeigen
textplot( capture.output(eurodist))  # Text als Plot
textplot(object, halign="center", valign="center", cex, ...)
```

☞ `objekt` kann z.B. `textplot(version)` sein oder die Ausgabe der Teststatistik, `cex` gibt Skalierung an

mit *kursivem* Text

```
data(iris)  # Schwertlilien-Datensatz
?iris # Hilfe dazu anzeigen lassen
reg <- lm( Sepal.Length ~Species, data=iris )  # lin. Regression
textplot( capture.output(summary(reg)), valign="top")  # Textplot
title("Regression der Blatt Laenge von Iris-Daten")  # Titel
par(xpd=TRUE) -> original # zum Zeichnen außerhalb der Diagrammfläche: xpd ="expandieren"
locator(1) -> wo  # Punkt mit Maus setzen
text(wo$x,wo$y ,
  substitute(Regression~der~Blatt~Laenge~von~italic(Iris)-Daten))
par(original)  # Zeichenfläche wieder wie Voreinstellung
?plotmath  # weitere Beispiele für Formatierungen s
```

☞ Es ist wohl etwas kompliziert kursiven Text einzeln auszugeben. Dies ist +/- eine Hilfskonstruktion, denn Zeilenumbruch mit `\n` geht nicht. Mit `par(xpd=TRUE)` kann man Text auch außerhalb zeichnen, was sonst nicht geht. Verbunden mit `locator(...)` kann man den Text mit der Maus platzieren: man muß hier aber 2x drücken: einmal für den x-Wert und den y-Wert

„Randtext“ läßt sich mit `mtext("Text" , side=3)` um das Diagramm zeichnen, dabei bedeutet die Option `side`: `1=bottom, 2=left, 3=top, 4=right`. Die Option `line` gibt die relativen Zeileneinheiten weg vom Rand an:

```
plot(1:12, xlab="", ylab="") # keine Achsenbeschriftung
mtext("2 Zeilen weg vom Rand: bottom=1",1, line=2)
mtext("-1 Zeile weg vom Rand: left=2", 2, line=-1)
mtext("0 Zeilen weg vom Rand: top=3", 3, line=0)
mtext("1 Zeile weg vom Rand: right=4", 4, line=1)
```

☞ `mtext()` läßt leider keine Drehung zu. Hierzu s. nachfolgendes Beispiel

Text/Beschriftung (Teilstriche) rotieren geht mit ein paar Tricks, daß man den Text separat und gedreht durch die Funktion `text(x, y, "text")` auf der vorherigen Seite.

```
# Rand vergrößern c( bottom, left, top, right)
par(mar = c(7, 4, 4, 2) + 0.1) -> original
# Grafik ohne x-Achsen Beschriftung
plot(1 : 8, xaxt = "n", xlab = "") # "n" - nothing
```

```
  # x - Achse zeichnen mit Hauptintervallen
  axis(1, labels = FALSE)
  # Teilstriche mit Anz. an Teilintervallen falls Paket Hmisc da
  if(require(Hmisc)) minor.tick(ny=2, nx=0)
  # Text erzeugen
  labels <- paste("Label", 1:8, sep = " ") # Label 1, Label 2, ...
  # text(x, y, .... ) an entsprechende Position schreiben
  text(x=1:8, y=par("usr")[3] - par()$cxy[2], # Koordinaten - Buchstabenhöhe
    srt = 45, adj = 1, # srt-Drehung, adj-Text rechtsbündig (0...1)
    labels = labels, xpd = TRUE, # xpd - Text darf auch außerhalb sein (=expand)
    col=rainbow(8)) # hübsche Farbe dazu ;-)
  # margintext - Randtext dazu
  mtext(side=1, text = "farbig gedrehte X Labels",
    line = 4) # +4 Zeilen vom Rand weg
par(original) # Grafikeinstellung wieder zurück
```

Punkte lassen sich mit `points(...)` dazuzeichnen. Dies kann auch datenabhängig erfolgen.

```
data(iris)  # Daten laden
?iris # Hilfe dazu anzeigen lassen
plot(iris[,1:2])  # ohne Farbe
points(iris[,1:2], pch=19, col=c("red", "blue3", "green3")[iris$Species])  # mit Farbangabe
```

Punkte als Boxplot, Thermometer, Stern an eine Beliebige Stelle der Grafik zeichnen lassen geht mit `symbols(...)` aus dem `graphics` Paket.

```
                                    Symbole als Thermometer
# s. auch example(symbols) graphics
x <- 1:(anzahl <-10) # 1...10  # Daten generieren
y <- sort(10*runif(anzahl)) # sortieren & mit Zufallszahlen vermengen
z <- runif(anzahl)
symbols(x, y,
  thermometers=cbind(0.5, 1, z),  # Breite, Höhe, Daten-%-Anteil
  inches=0.5,  # Skalierung
  fg = rainbow(anzahl)) # Farben: so viel, wie anzahl
                                  Breite und Höhe datenabhängig
z123 <- cbind(z,
  2*runif(anzahl),
  runif(anzahl)
  )
symbols(x, y,
  thermometers = z123,
  inches=FALSE) #  Skalierung nach x-Werten
text(x+1,y-1,
  apply( # apply(X, MARGIN, FUN, ...) # Text formatieren
    format(round(z123, dig=2)),
    1,
    paste, # Funktion paste(...)
    collapse = "\n"
  ),
  adj = c(-0.2,0),  # Textausrichtung
  cex = 0.75, # Textvergrößerung
  col = "purple", # Farbe
  xpd=NA)
                                    Symbole als Sterne
symbols(x, y,
```

...Fortsetzung umseitig

```
  stars=z123, # kann mehr als 3 Spalten haben
  inches=FALSE) # Skalierung nach x-Werten
  points(x,y, pch=16, cex=0.5) # Punkte hinzufügen
                                    Symbole als Boxplots
symbols(x, y,
  boxplots=cbind(
     # (1) Box:Breite (2) Box:Höhe
    0.8, runif(anzahl),
     # (3) Whiskers:oben (4) Whiskers:unten
    runif(anzahl),runif(anzahl),
     # (5) Median: [0... bis ...1]
    runif(anzahl)),
    bg="bisque", # Boxfarbe
    col="red", # Whiskers Farbe
    fg="blue", # Box Rahmenfarbe
  inches=0.5) # Box-Skalierung um 0.5
                                  Symbole als Kreis + Farben
data(trees) # Daten zu Baummessung
?trees  # Hilfe anzeigen
N <- nrow(trees) # Anzahl Zeilen
attach(trees) # in den R Suchpfad
palette(rainbow(N, end = 0.9)) # Farbpalette
symbols(Height, Volume, # x-y Werte
  circles=Girth/16, # Kreise Größenangabe
  inches=FALSE,
  bg = 1:N, # Farbe Kreise
  fg="gray30", # Farbe Kreislinie
  main="Baumhöhe:Volumen\n symbols(*, circles=Girth/16, bg = 1:N)", # Titelei
  xlab="Höhe in ft", # x-Achsenbeschriftung
  ylab="Volumen in ft^3" # y-Achsenbeschriftung
  )
palette("default") # Farbe wieder Voreinstellung
detach() # Daten wieder aus den Suchpfad entfernen
```

Punktbeschriftungen sind möglich durch benutzen der Funktion `text(...)` (auf Seite 36). Eine Bezeichnung mit zusätzlichem Anstrich ermöglicht die Funktion `spread.labels(...)` aus dem Paket `plotrix`. Sie ist aber etwas unglücklich geschrieben, da die Bezeichnungen in einem Datenvektor durcheinander kommen. Deshalb: am besten die manuell geschriebene Funktion `spread.labels2(...)` benutzen. Sie enthält zusätzlich den Parameter `wide=5`, der den Labelabstand um + 5 % weitet:

Text

```
x <- rnorm(10)/10  # Zufallszahlen erzeugen
y <- sort(rnorm(10))
plot(x,y,xlim=c(-1,1),type="p")
nums<-c("one","two","three","four","five","six","seven","eight","nine","ten")
spread.labels(x,y,nums,0.5)  # mit offset 0.5

-------------------------------------------------------------------------------
spread.labels2 <- function (x, y, labels = NULL, offset, wide=5 , col = "white", border = FALSE, adj=0,
   ...)
{
   if (missing(x))
    stop("Usage: spread.labels2(x,y,labels=NULL,offset,col=\"white\",...)")
   if (diff(range(x)) < diff(range(y))) {
    sort.index <- sort.list(y)
    x <- x[sort.index]
    y <- y[sort.index]
    ny <- length(y)
    offsets <- rep(c(offset, -offset), ny/2 + 1)[1:ny]
```

...Fortsetzung umseitig

```
    newy <- seq(y[1]*(1-wide/100), y[ny]*(1+wide/100), length = ny)
    segments(x + offsets, newy, x, y, col="grey")
    boxed.labels(x + offsets, newy , labels = labels[sort.index], col = col, border = border,
    ...)
   }
   else {
    sort.index <- sort.list(x)
    x <- x[sort.index]
    y <- y[sort.index]
    nx <- length(x)
    offsets <- rep(c(offset, -offset), nx/2 + 1)[1:nx]
    newx <- seq(x[1]*(1-wide/100), x[nx]*(1+wide/100), length = nx)
    segments(newx, y + offsets, x, y, col="grey")
    boxed.labels(newx + offsets, y , labels=labels[sort.index], col = col, border = border,
    ...)
   }
}
```

Titel

Titel lassen sich mit `title(...)` extra zeichnen. Ansonsten mit `main="Titel"` in `plot(...)`

```
x <- seq(-pi, pi, len = 65)  # Daten erzeugen
plot(x, sin(x), type = "l", ylim = c(-1.2, 1.8), col = 3, lty = 2)
points(x, cos(x), pch = 3, col = 4)  # Punkte zeichnen
lines(x, tan(x), type = "b", lty = 1, pch = 4, col = 6)  # Linien zeichnen
title("legend(..., lty = c(2, -1, 1), pch = c(-1,3,4), merge = TRUE)", cex.main = 1.1)  # Titelei
```

Den Titel z.B. mit verschiedenen Schriften, Farben formatieren, kann man mit `mtext()` gestalten:

```
plot(1:10)  # Zahlen 1 .. 10
wohin.x <- 5  # x-Position
mtext(c("title1 ", "title2"),  # Margintext
  col = c("green","red"),  # Farben; oder dasselbe mit c(2:3)
  at = c(wohin.x, wohin.x+strwidth("title2")),  # wohin
  font  = c(1,3),
  line = 1  # 1-facher Linienabstand zur Grafik
)
```

Legenden erzeugt man mit `legend(...)`. Geschickterweise kann man sie auch mit `locator(...)` benutzen, um sie gezielt mit der Maus zu platzieren. Oder man gibt statt Koordinaten einfach `legend("bottomleft", ...)` an. Es gibt natürlich alle möglichen Varianten `bottomright`, `bottom`, `bottomleft`, `left`, `topleft`, `top`, `topright`, `right` und `center`.

Auch recht einfach zu „bedienen“ ist `rlegend(...)` aus dem Paket `Hmisc`. Siehe auch Beispiel mit Scatterplot-matrizen `pairs(...)` auf Seite 53.

```
                                    Bsp. mit locator(...)
plot(runif(100), rnorm(100))  # Zufallsdaten erzeugen
legend(locator(2), "Punkte", pch=1)  # Legende Platzieren Variante 1
  locator(1) -> pos  # Variante 2
  legend(pos$x, pos$y, "Punkte", pch=1)
                                    Bsp. mit Linientypen
x <- seq(-pi, pi, len = 65)  # Daten erzeugen
plot(x, sin(x), type = "l", ylim = c(-1.2, 1.8), col = 3, lty = 2)  # Graph zeichnen
points(x, cos(x), pch = 3, col = 4)  # Punkte zeichnen
```

...Fortsetzung umseitig

```
lines(x, tan(x), type = "b", lty = 1, pch = 4, col = 6)  # Linie mit x zeichnen
legend(-1, 1.9,  # oder "bottomright", "bottom", "bottomleft", "left"usw.
  c("sin", "cos", "tan"),  # Text
  pt.bg = c(3,4,6), # Punkt Hintergrund
  lty = c(2, -1, 1),  # Linientyp auf Seite 27
  pch = c(-1, 3, 4),  # Punkttyp auf Seite 28
  merge = TRUE, # Punkte und Linien zusammenfügen
  bg='gray90' # Farbe Hintergrund
)
rm(x)  # x wieder löschen
example(legend)  # Bsp. ansehen
```

☞ Optionen `legend(...)`: `legend(-1, 1.9,...)` x, y Koordinate (oder locator(n) benutzten); `legend(x, y, legend,...)` Text od. Ausdruck `c(...)` kann benutzt werden; `fill='gray50'` kleine Farbige Box neben Legendentext; `col="gray50"` Farbe für Legenden-Punkte/Linien; `lty=2, lwd=1.5, pch=16` Linientyp, -breite, Punktform (s. `par(...)` auf Seite 26), `bty="n"` kein Rahmen zeichnen, `bg="gray50"` Legenden-Hintergrunfarbe (geht nur, wenn `bty="n"`); `pt.bg="red"` Punkthintergrundfarbe;, `cex=1.2` relative Größenangabe; `xjust=0` (Werte von 0...1) Ausrichtung zur Lokalisation (↔) von `x, y`: 0- links, 0.5- Mitte, 1- rechts; `yjust=0` dasselbe nur ↕ ; `x.intersp=1.2` relativer ↔ Abstand; `y.intersp=1.2` dasselbe nur ↕; `adj=2` Textausrichtung – je größer die Zahl, desto weiter links (nützlich für mathematische Ausdrücke s. auf dieser Seite)

Farbwerte in Legenden color.legend(..)

```
# example(color.legend) # Beispiel ansehen
library(plotrix)  # Paket laden
par(mar=c(7,4,4,6)) -> paralt  # etwas mehr Platz (s.S.26)
testcol <- color.gradient(  # Farbgradienten erzeugen
  c(0,1),  # rot
  0,  # grün
  c(1,0),  # blau
  nslices=5)  # Anzahl Farben
farb.labels <- c("kalt","warm","heiß")
(daten.matrix <- matrix(rnorm(100),nrow=10))  # Daten + ansehen
color2D.matplot(daten.matrix,  # Daten
  c(1,0),  # rot-Bereich
  0,  # grün-Bereich
  c(0,1),  # blau-Bereich
  main="Test color.legend(..)")  # Titel
color.legend(11,6,11.8,9,  # wo: xl yb xr yt
  farb.labels,  # Text
  testcol,  # Farben
  gradient="y")  # Gradient in x oder y Richtung
color.legend(10.2,2,11,5,farb.labels,testcol,
  align="rb",  # rb-right bottom?
  gradient="y")  # Gradient in x oder y Richtung
color.legend(0.5,-2,3.5,-1.2,farb.labels,testcol)
color.legend(7,-1.8,10,-1,farb.labels,testcol,align="rb",
  col=testcol[c(1,3,5)])  # Farben Text zusätzlich
par(paralt) # Grafikeinstellung wieder zurück
detach(package:plotrix)  # Paket eventuell wieder entfernen
```

Mathematischen Ausdrücke lassen sich z.B. in Plots durch Benutzen der Funktion `expression(...)` einzeichnen:

```
plot(0, main=expression( x %+-% y))
  # x ± y
### group() läßt nur kleine Klammern zu bgroup() skaliert Klammern entsprechend:
plot(0, ylab=expression('mean SV dB    ' * group("[", m^{2} * ("160 m")^{-3}, "]")))
  # mean SV dB[m^2(160m)^-3]
  # Klammern richtig mit bgroup()
  title(expression('mean~SV~dB    ' * bgroup("[", m^{2} * ("160 m")^{-3}, "]")))
  # meanSVdB [m^2(160m)^-3]
H[2] * SO[4]^paste(2," -") # H2SO4 2-
mg%.%l^-1 # mg · l^-1
```

```
Salinität ~ ~ group("(", g%.%l^-1, ")") # Salinität  g · l^-1
# '~' gibt zusätzlich Platz oder '   '
R[paste("adj")]^2 # R^2_adj
### Werte + Formatierungen (1)
  variable <- "euclidean" # Variable speichern
  plot(0,0,
    xlab = bquote(
      Distance == .(variable) * # statt == kann auch %+-% d.h. mathematische Relationen
        # .() ist in bquote() eine R-Anweisung sonst nur Zeichenkette
      " " * # " " = etwas Platz
      bgroup(
        "(", # Klammer 1 - skaliert
        wisconsin * " " *
          group(
            "(",sqrt(data),")" # Wurzel
          ),
        ")" # Klammer 1 - skaliert
      )# end bgroup
    ) # end bquote
  )
  # Distance = euclidean (wisconsin(√data))
### Werte + Formatierungen (2)
  nSpecies <- 59
  plot(0,0)
  title(sub=
    bquote(.(nSpecies) # Variable wird ausgewertet
      *''* # ersetzt == (wichtig sonst Fehler)
      ' ' * chironomid * # Text mit Leerzeichen: * verbindet
      ' ' * taxa *
      ' ' *  vs. *
      ' ' * conductivity[' '*log[10]] #
    )
  )# end title()
  # 59 chironomid taxa vs. conductivity log10
```

Formatierte Ausdrücke lassen sich auch als *Zeichenketten* auswerten, durch kombinieren der Funktionen `eval(parse(text=...))`:

```
eval(parse(text = "print(d <- 4 + 7)"))
#[1] 11
plot.new() # neue leere Grafik
Zeichenkette <- "expression('°' * C[paste(H[2]*O)])"
title( # Titel hinzu
  main = eval(
    parse(text = Zeichenkette)
  ) # eval() Ende
) # title() Ende
```

Beispiel für Vorzeichenwechsel als Erklärung in Grafik: $+\sigma_{Zufall}$ und $-\sigma_{Zufall}$:

```
nZahlen <- 2000 # Anzahl Zahlen
zufall <- rnorm(nZahlen) # Zufallszahlen erzeugen
untenOben <- c(1,-1) # für text() oben und unten
## Grafik
plot(
  zufall, # Zufallszahlen
  pch=".",# Punkttyp  auf Seite 28
  bty="L", # Boxtyp  auf Seite 27
  main="Text mit Vorzeichen"
```

```
)
## Hilfslinen
abline(h=0) # Hilfslinie x-Achse
abline(# Linien für Standardabweichung (SD)
  h=sd(zufall)*untenOben, # 2x zeichnen obere SD untere SD
  lty="dotted"
)
## Text dazu
text(# text '+SD Zufall' '-SD Zufall' dazu
  x=par()$usr[2]*0.8, # par("usr")[2] = rechter Grafikrand
  y=sd(zufall)*untenOben+0.25*untenOben, # y-Position
  labels=parse(# text als mathematischer Ausdruck expression()
    text=lapply(# apply a function over a list or vector
      ifelse(untenOben>=0,"+","-"),# wenn größer gleich 0 dann "+" sonst "-"
      paste, # Funktion paste() auf ifelse() anwenden
      "SD [' Zufallszahl']", # Argumente für paste()
      sep="" # Argumente für paste() sep="" kein Zwischenraum
    )# end lapply
  ),# end parse
  xpd=NA # außerhalb zeichnen: ja
)
```

Weitere Beispiele durch Eingabe von `demo(plotmath)`:

Arithmetic Operators	
x + y	$x + y$
x - y	$x - y$
x * y	xy
x/y	x/y
x %+-% y	$x \pm y$
x%/%y	$x \div y$
x %*% y	$x \times y$
x %.% y	$x \cdot y$
-x	$-x$
+x	$+x$
Sub/Superscripts	
x[i]	x_i
x^2	x^2
Juxtaposition	
x * y	xy
paste(x, y, z)	xyz
Lists	
list(x, y, z)	x, y, z

Radicals	
sqrt(x)	$\sqrt{x}$
sqrt(x, y)	$\sqrt[y]{x}$
Relations	
x == y	$x = y$
x != y	$x \neq y$
x < y	$x < y$
x <= y	$x \leq y$
x > y	$x > y$
x >= y	$x \geq y$
x %~~% y	$x \approx y$
x %=~% y	$x \cong y$
x %==% y	$x \equiv y$
x %prop% y	$x \propto y$
Typeface	
plain(x)	x
italic(x)	x
bold(x)	$\mathbf{x}$
bolditalic(x)	$\boldsymbol{x}$
underline(x)	$\underline{x}$

Ellipsis	
list(x[1], ..., x[n])	$x_1, \ldots, x_n$
x[1] + ... + x[n]	$x_1 + \cdots + x_n$
list(x[1], cdots, x[n])	$x_1, \cdots, x_n$
x[1] + ldots + x[n]	$x_1 + \ldots + x_n$
Set Relations	
x %subset% y	$x \subset y$
x %subseteq% y	$x \subseteq y$
x %supset% y	$x \supset y$
x %supseteq% y	$x \supseteq y$
x %notsubset% y	$x \not\subset y$
x %in% y	$x \in y$
x %notin% y	$x \notin y$
Accents	
hat(x)	$\hat{x}$
tilde(x)	$\tilde{x}$
ring(x)	$\mathring{x}$
bar(xy)	$\overline{xy}$
widehat(xy)	$\widehat{xy}$
widetilde(xy)	$\widetilde{xy}$

Arrows	
x %<->% y	$x \leftrightarrow y$
x %->% y	$x \rightarrow y$
x %<-% y	$x \leftarrow y$
x %up% y	$x \uparrow y$
x %down% y	$x \downarrow y$
x %<=>% y	$x \Leftrightarrow y$
x %=>% y	$x \Rightarrow y$
x %<=% y	$x \Leftarrow y$
x %dblup% y	$x \Uparrow y$
x %dbldown% y	$x \Downarrow y$
Symbolic Names	
Alpha - Omega	$A - \Omega$
alpha - omega	$\alpha - \omega$
phi1 + sigma1	$\varphi + \varsigma$
Upsilon1	Υ
infinity	∞
32 * degree	32°
60 * minute	$60'$
30 * second	$30''$

Style	
displaystyle(x)	x
textstyle(x)	x
scriptstyle(x)	x
scriptscriptstyle(x)	x
Spacing	
x ~ ~y	$x\ \ y$

x + phantom(0) + y	$x + \ \ + y$
x + over(1, phantom(0))	$x + \frac{1}{\ }$
Fractions	
frac(x, y)	$\frac{x}{y}$
over(x, y)	$\frac{x}{y}$
atop(x, y)	$\genfrac{}{}{0pt}{}{x}{y}$

Big Operators	
sum(x[i], i = 1, n)	$\sum_{1}^{n} x_i$
prod(plain(P)(X == x), x)	$\prod_{x} P(X = x)$
integral(f(x) * dx, a, b)	$\int_{a}^{b} f(x)dx$
union(A[i], i == 1, n)	$\bigcup_{i=1}^{n} A_i$
intersect(A[i], i == 1, n)	$\bigcap_{i=1}^{n} A_i$
lim(f(x), x %->% 0)	$\lim_{x \to 0} f(x)$
min(g(x), x >= 0)	$\min_{x \geq 0} g(x)$
inf(S)	$\inf S$
sup(S)	$\sup S$

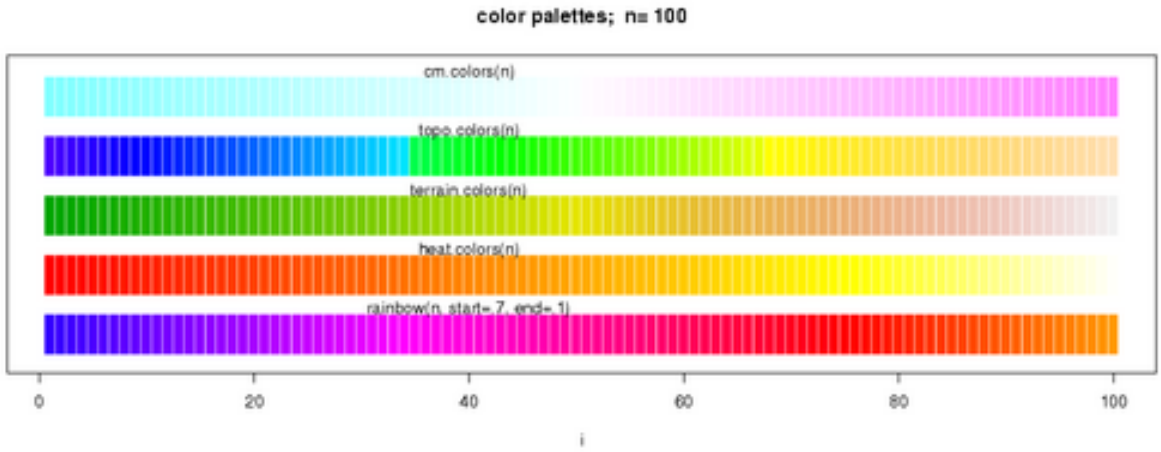
Grouping	
(x + y) * z	$(x + y)z$
x^y + z	$x^{y} + z$
x^(y + z)	$x^{(y+z)}$
x^{y + z}	x^{y+z}
group("(", list(a, b), "]")	$(a, b]$
bgroup("(", atop(x, y), ")")	$\binom{x}{y}$
group(lceil, x, rceil)	$\lceil x \rceil$
group(lfloor, x, rfloor)	$\lfloor x \rfloor$
group("|", x, "|")	$\lvert x \rvert$

Mit der Benutzerfunktion `listExpressions()` im Anhang auf Seite 184 lassen sich Mathematische Ausdrücke oder formatierte Zeichenketten als Liste ausgeben:

```
listeZeichenketten <- c(
  "expression(H[2]*O[paste('Type')])",
  "expression('°'*C[paste('July')])",
  "expression('°'*C[paste(H[2]*O)])",
  "expression(Type[paste('veget.')])",
  "expression(Human[paste('Influence')])",
  "expression(H[2]*O[paste('area')])",
  "expression(Basin[paste('closed')])",
  "expression(pH)"
)
plot(0,0) # Zeichengrundlage
listExpressions(
```

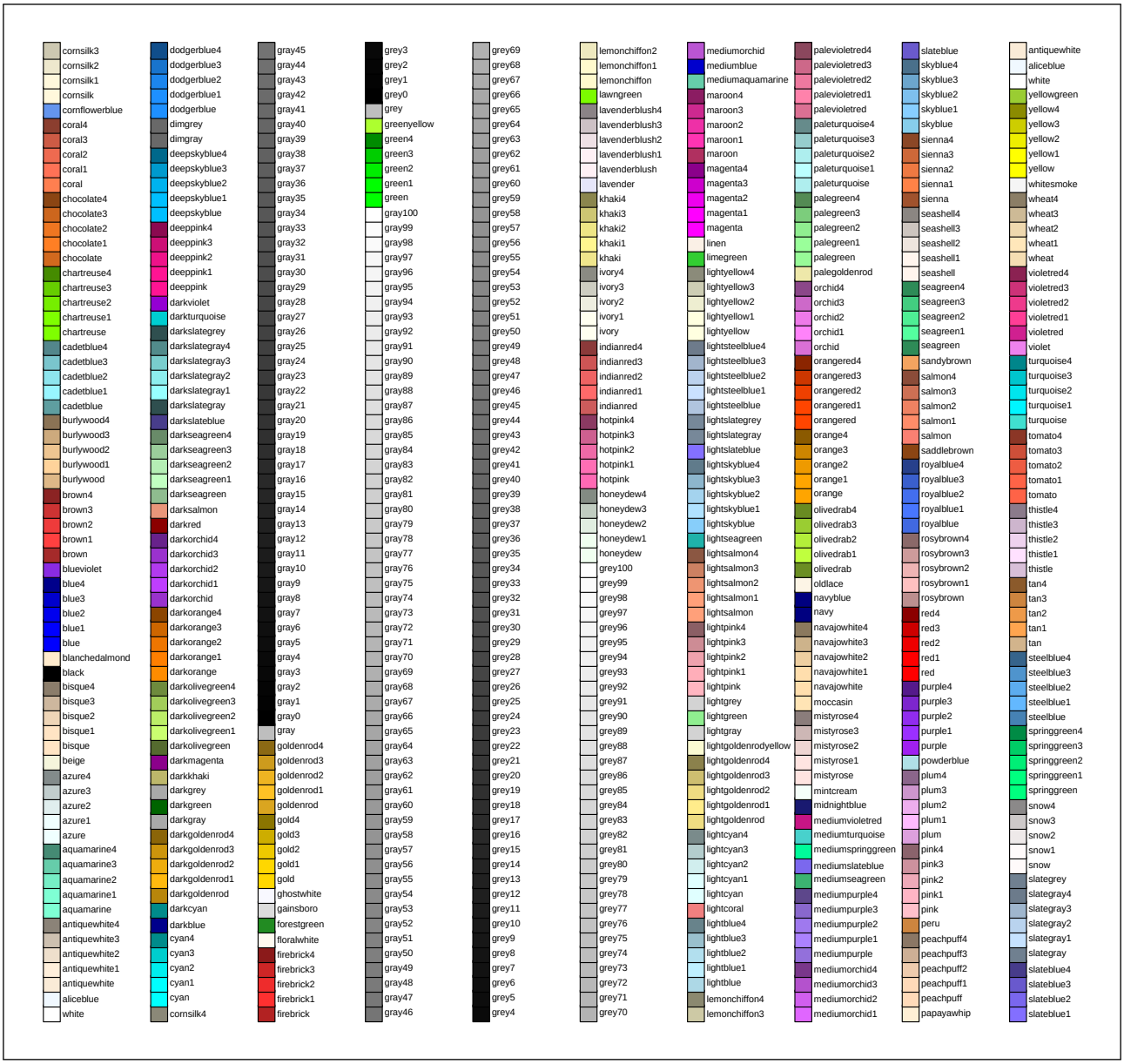
```
  expressions=listeZeichenketten,
  adj=0 # linksbündig
)
```

Farben Meist mit `col="green"`. Die extreme Vielfalt (657 Namen) der Farben läßt sich mit `list(colors(...))` anzeigen. Dabei gibt es für jede Standardfarbe, wie `red`, `blue`, usw. jeweils von `blue1`...`blue4`, sowie eine `light`- als auch eine `dark`-Variante. Fertige Farbmischungen sind: `cm.colors(..)`, `topo.colors(..)`, `terrain.colors(..)`, `heat.colors(..)`, `rainbow(..)`:



alle voreingestellten Farbe

```
(colors() -> farben)              # Farbnamen speichern + ausgeben durch Klammer außen
farben <- c(farben,rep(NULL,3)) # 657 Farben um 3 NULL-Werte erweitern
zeilen <- 66; spalten <- 10      # 660 Farben
(matrix(1:spalten,nrow=zeilen,ncol=spalten,byrow=T) -> farben.zahlen)  # Matrix für Punkte
par(mex=0.001,xaxt="n",yaxt="n",ann=F) -> paralt  # Grafikeinstellungen speichern
x_offset <- 0.5
x <- farben.zahlen[,1:spalten]  # x-Werte (Zahlen)
y <- rep(1:zeilen,spalten)      # y-Werte (Zahlen)
plot(x,y,
  pch=22,                       # Punkttyp Rechteck
  cex=2,                        # Vergrößerung Punkte
  bg=farben,                    # Hintergrundfarben
  bty="n",                 # keine Box
  xlim=c(1,spalten+x_offset)    # x-Wertebereich
)
text(x+0.1,y,farben,adj=0,cex=0.6)  # Text Farben dazu
par(paralt)                     # Grafikeinstellungen zurück
```



Palette festlegen (allg.)

```
palette(rainbow(12, s = 0.6, v = 0.75))
palette(gray(seq(0,.9,len=25)))
palette(rainbow(25, start=.7, end=.1))  # 25 Regenbogenfarben mit Start und Ende
palette(topo.colors(50)[40:10])  # Alternative: 30 Topo Farben absteigend
palette(topo.colors(50)[10:40])  # Alternative: 30 Topo Farben aufsteigend
palette("default")  # Palette zurücksetzen
```

☞ Palette `rainbow` modifizieren: 12 Farben mit `s = 0.6` Sättigung; `v = 0.75` „Value“ entspricht Grauwert: `v=0` schwarz, `v=1` „transparent“ – `gray(seq(0,.9,len=25))` 25 Grautöne von 0 bis 0.9, d.h. weiß bis fast schwarz. Beachte, daß z.B. beim plotten die Angabe `col=rainbow(13)` und `color=rainbow` verschieden ist

```
library(RColorBrewer)  # Erweiterung für Farbsequenzen
example(display.brewer.all) # Beispiel ansehen
detach(package:RColorBrewer)  # Paket eventuell entfernen
```

```
library(plotrix)  # Paket für nützliche Plotfunktion
 example(color.gradient) # Beispiel ansehen
detach(package:plotrix)  # Paket eventuell entfernen
```

...Fortsetzung umseitig

Farbgradient erzeugen color.scale(...)

```
library(plotrix)  # Paket für nützliche Plotfunktion
 x <- rnorm(20)  # 20 Zufallszahlen
y <- rnorm(20)  # 20 Zufallszahlen
# nutze 'y' für Farbskala
plot(x, y, col = color.scale(
    y,  # entlang der y-Achse
    c(0,1,1), # rot-Bereich
    c(1,1,0), # grün-Bereich
    0), # blau-Bereich
  main="Test: color.scale(..)",  # Titelei
  pch=16,  # Punkttyp s. auf Seite 28
  cex=2)  # Punktgröße
detach(package:plotrix)  # Paket eventuell entfernen
```

3.2 Diagramme

3.2.1 Datenfelder - Array

Mit `table.value(...)` aus dem Package `ade4` lassen sich Datenfelder plotten. ☞ geht nur wenn keine `NA` drin sind!

```
library(ade4) # Paket einmal laden
data(varespec, package="vegan") -> varspec # Datensatz Flechten auf Weideland
table.value(varspec) # Fehler
  Fehler in 1:ncol(df) : NA/NaN Argument  # enthält NA's; s. auf Seite 9
na.omit(varespec) -> varspec # NA entfernen
table.value(varespec, # varespec = Datenframe ("Tabelle")
  clabel.r=2,  # Zeilen um 2 vergrößert
  clabel.c=1.5,# Spalten um 1.5 vergrößert
  csize=0.5,   # verkleinert Quadrate um 0.5
  clegend=1.5, # vergrößert die Legende um 1.5
  y=nrow(varespec):-1 # in y-Richtung etwas mehr Platz
)# Ende table.value()
### Daten transponieren (Reihen und Spalten vertauschen)
table.value(t(varespec), # Daten transformieren = umdrehen
  row.labels=parse(text=paste("italic(",colnames(varespec),")")),
  # Beschriftung Reihen als expression 'italic(Vac.myr) ...'
  col.labels=rownames(varespec), # Beschriftung Spalten
  clabel.r=0.9, # Zeilen um 1.0 vergrößert
  clabel.c=1.0, # Spalten um 1.0 vergrößert
  csize=0.5,    # verkleinert Quadrate um 0.5
  clegend=1.5,  # vergrößert die Legende um 1.5
  y=ncol(varespec):-1 # in y-Richtung etwas mehr Platz
)# Ende table.value()
detach(package:ade4) # Paket eventuell wieder entfernen
help(varespec, package="vegan") # Hilfe zum Datensatz
```

Für die Legende wurden zwei zusätzliche Zeilen eingefügt: `y` soll von `ncol(...)` (number of columns) bis -1 gehen, statt normaleweise `ncol(varespec):1`

3.2.2 Blattfunktion

Die Blattfunktion ordnet Stichprobenwerte nach Größe und Menge – mit `stem()`

```
x <- c(160,165,170,177,177,177,178,178,178,179,180,180,
180,180,182,184,185,185,185,186,186,187,187,190,190,191,191,193,198)
stem(x)

The decimal point is 1 digit(s) to the right of the |
16 | 0  # 160
16 | 5  # 165
17 | 0  # 170
17 | 7778889  # 177,177,177,178...
18 | 000024
18 | 5556677
19 | 00113
19 | 8
```

3.2.3 Boxplot

Die Funktion `boxplot(...)` zeichnet einen Boxplot, wobei Minimum ($\perp$, etwa 10 % Häufigkeitsgrenze), 25 % Quantil, Median ($-$, 50 % Quantil), 75 % Quantil, Maximum ($\top$, etwa 90 % Häufigkeitsgrenze) und eventuelle Extremwert ($\circ$) eingezeichnet werden. Es lassen sich noch Einschnürungen (`notch`) einzeichnen. Sie sind eine Art Konfidenzintervall des Medians: überlappen sich zwei Intervalle gibt es keinen Unterschied auf dem 5 % Niveau, trennen sich die Intervalle, ist der Unterschied auf dem 5 % Niveau signifikant. Siehe auch auf Seite 132.

```
data(airquality)  # R-eigene Daten laden: New York Air Quality Measurements
?airquality  # Hilfe dazu anzeigen
attach(airquality)  # Datensatz in den Suchpfad aufnehmen
par(no.readonly=TRUE) -> original  # alte Grafikeinstellungen speichern
par( # Grafikparameter s. auf Seite 26
  mfrow=c(2,1), # Grafik hat jetzt 2 Reihen und 1Spalte: also übereinander
  mar=c(0,4.1,4.1,2.1) # Rand reduzieren
)
BxpSpeicherOzone <- boxplot(Ozone~Month, # Month ist hier die Variable nach der gruppiert wird
  col="gray",  # Farbe s. 3.1.2
  xlab="",  # x-Achsenbeschriftung
  ylab="Ozon ppb",  # y-Achsenbeschriftung
  main="Luftdaten New York \n 1973", # Titelei - \n ist gleich Zeilenumbruch
  notch=TRUE,  # Vertrauensintervall des Median
  xaxt = "n" # x-Achse ohne 'ticks' und 'labels'
   # names=FALSE , # alternativ zu xaxt = "n" aber 'ticks ' bleiben noch
) # Funktion boxplot() hier zu Ende
par(mar=c(5.1,4.1,0,2.1))  # Rand reduzieren
BxpSpeicherTemp <- boxplot(Temp~Month, # Month ist hier die Variable nach der gruppiert wird
  col=rainbow(5),  # 5 Regenbogenfarben zeichnen bezüglich Monat
  xlab="Monat",  # x-Achsenbeschriftung
  ylab="Temperatur (°F)",  # y-Achsenbeschriftung
   # main="Luftdaten New York \n 1973", # Titelei - \n ist gleich Zeilenumbruch
  notch=TRUE  # Vertrauensintervall des Median
) # Funktion boxplot() hier zu Ende
par(original)  # Grafikeinstellungen zurücksetzen oder Grafikfenster schließen und neuzeichnen
```

☞ Farben würde ich prinzipiell weglassen, da Gruppe ja drunter steht. Hier nur illustrativ. Ein Boxplot bei dem n=12 usw. unter den Plots steht, bietet die Funktion `boxplot.n(...)` aus dem package `gregmisc`

```
BxpSpeicherTemp  # Parameter des gespeicherten Boxplot ansehen
bxp(BxpSpeicherTemp)  # Boxplot aus Boxplot-summary zeichnen
                                    Boxplot zu Daten dazuzeichnen
plot(Ozone~Temp,  #  Daten y und x
  col=rainbow(5)[Month-4],  #  5 Regenbogenfarben zeichnen bezüglich Monat
  pch=16,  #  Punkttyp gefüllt s. auf Seite 28
  cex=0.5,  #  Punkttyp  Verkleinerung
```

...Fortsetzung umseitig

```
  main="Luftdaten New York \n 1973",  #  Titelei - \n ist gleich Zeilenumbruch
  xlab="Temperatur (°F)",  #  x-Achsenbeschriftung
  ylab="Ozon ppb"   #  y-Achsenbeschriftung
)  #  Funktion plot() hier zu Ende
rug(Ozone, side=2)  #  gibt Anzahl der Beobachtungen als Randplot aus
bxp(BxpSpeicherOzone,  #  gespeicherten Boxplot zeichnen
  add=TRUE,  #  dazu
  at=BxpSpeicherTemp$stats[3,1:5],  #  sind die Mediane
  notch=TRUE,  #  Vertrauensintervall des Median
  boxfill=rainbow(5),  #  Farben Box
  xaxt="n"   #  keine x-Achse
)  # Funktion boxplot() hier zu Ende
wo.x <- BxpSpeicherTemp$stats[3,1:5]  #  x-Koordinaten zwischenspeichern: entspricht Median
# Anmerkung Text dazuschreiben
wo.y <- rep(140,5) #  y-Koordinaten zwischenspeichern: 5 x 140
wo.xy <- cbind(wo.x, wo.y)  #  Koordinaten zusammenfügen
text(wo.xy,  # Anmerkung Text schreiben
  labels="beim Median",  #  Text
  adj=0,  #  linksbündig ; 1 wäre rechtsbündig
  srt=60,  #  Drehung
  col=rainbow(5)  #  Farben
)  #  Funktion text() hier zu Ende
# Legende dazu s. auf Seite 40
legend("bottomright",  # Position
legend=paste("Monat", 5:9),  # Text: 'Monat 1', 'Monat 2', ...
pt.bg=rainbow(5),  # Farbenhintergrund f. points
pch=22  # Viereck als Punkt
) # Funktion legend() hier zu Ende
                                    Boxplot mit Datensplitting
zufallseq <- seq(60)*runif(60)  # 1, 2,...60 mal Zufallszahl (0..1)
teilung <- round(runif(60),0)  # 1 0 1 0 0 1 0 0 1 1...
boxplot(split(zufallseq,teilung),
  main="Datensplitting mit Boxplot\n split(...)", # Titelei
  notch=TRUE,  #  Vertrauensintervall des Median
  sub="Zufallsdaten" # Untertitel
) # Ende boxplot() Funktion
                                  subset in boxplot() work - around:
# unter Linux scheint die Option 'subset' innerhalb boxplot() zu klappen, unter Windows nicht:
data <- data.frame(values=c(1:25),  # Variable'values'
  groups=rep(c("A","B","C","D","E"), each=5)  # Variable 'groups'
)
data  # Daten ansehen
data = subset(data,groups!="C")  # Teildatensatz erzeugen: hier ohne 'C'
boxplot(values~factor(groups), data=data)  # WICHTIG: factor() nicht vergessen!!!
                                Boxplotverteilung am Rand von Grafiken
library(car)  # Paket Companion to Applied Regression
scatterplot(Ozone~Temp|Month)  # Boxplots außen
# ist zwar unübersichtlich, aber zum Vergleich
scatterplot(prestige ~  income|type,  # Gruppierung nach 'type'
  data=Prestige,  # Datensatz
  span=1,  # Glättungsfaktor s. ?loess
  legend.plot=TRUE
) # Funktion scatterplot() hier zu Ende
?Prestige  # Info Hilfe zu den Daten
detach(package:car) # Paket eventuell wieder entfernen
                                Boxplot Daten nach Median sortieren
set.seed(1)  # Start Zufallsgenerator auf 1
(z <- data.frame(x = rep(LETTERS[1:3], each = 6), y = rnorm(18)))
```

...Fortsetzung umseitig

```
tapply(z$y, z$x, median)  # noch nicht sortiert
 A B C
 -0.22140524 0.53160520 -0.03056194
z$x <- with(z, ordered(x, levels(x)[order(tapply(y, x, median))]))
tapply(z$y, z$x, median)  # sortiert nach median
 A C B
 -0.22140524 -0.03056194 0.53160520
boxplot(y ~  x, data = z)
```

☞ Um `n=34` bei horizontalen Boxplots auch einzuzeichnen, folgende Beispiele:

n=... Beispiel - vertikal und horizontal

```
wieviele <- 9 # Anzahl der 'Arten': hier kann man'drehen'
(arten <- paste(rep("Art",wieviele), 1:wieviele)) # Art 1, Art 2, ...
(wieoft <- sample(wieviele, replace=TRUE)) # wieoft vorhanden
(arten <- rep(arten, wieoft)) # jede Art individuell 'gezählt'
(merkmal <- runif(length(arten))) # irgendein Merkmal
spezies <- data.frame( # Datenobjekt erzeugen
  arten=arten, # Spalte 'arten'
  merkmal=merkmal # Spalte 'merkmal'
) # Funktion data.frame() Ende
spezies # Datenobjekt anschauen
BxpSpeicher <- boxplot(merkmal~arten,
  ylim=c(-0.1, max(merkmal)), # y-Achsenskalierung
  ylab="Merkmal", # y-Achsenbeschriftung
  main="boxplot mit \"n=...\"\n Zufallsdaten"   # Titelei
) # Funktion boxplot() Ende
(table(arten) -> arten.anz ) # Anzahl zählen
(text.anz <- paste(" n=",arten.anz[1:wieviele], sep="")) # 'n=...' erzeugen
(wo.x <- 1:wieviele) # Koordinaten für 'n=...'
(wo.y <- rep(-0.05, wieviele)) # Koordinaten für 'n=...'
text(wo.x, wo.y, # Koordinaten für 'n=...'
  labels=text.anz # Text
) # Funktion text() Ende
```

```
par(las=1, xpd=TRUE) -> original
# xpd: auch außerhalb der Grafik darf gezeichnet werden
# las: labels axis - Ausrichtung
bxp(BxpSpeicher,
  horizontal=TRUE,
  xlab="Merkmal", # x-Achsenbeschriftung
  main="boxplot mit \"n=...\"\n Zufallsdaten"  # Titelei
) # Funktion bxp() Ende
par() -> grafik.param # um Grafikränder rauszubekommen
grafik.param$usr[2] -> rand.re # Zahl für rechten Rand
text(rand.re, wo.x, # Koordinaten
  labels=text.anz, # Text
  adj=0 # linksbündig
) # Funktion text() Ende
par(original) # Grafikeinstellungen wieder zurück
```

n=... mit boxplot.n() - nur vertikal!

```
library(gplots) # Paket laden
# Boxplot für 'n=...'
boxplot.n(merkmal~arten,
  shrink=0.8 # Skalierung Text
```

```
) # Funktion boxplot.n() Ende
detach(package:gplots) # Paket eventuell wieder entfernen
```

n=... mit Funktion aggregate(...)

Species	Anzahl	pH
Chironomus sp.	12	8.11
Chironomus sp.	34	8.77
Tanytarsus sp.	230	8.3
Tanytarsus sp.	84	9.9
Tanytarsus sp.	4	1.83
Micropsectra sp.	209	9.83
⋮	⋮	⋮

```
boxplot(pH~Species,  # allgemein: Wert~Gruppierung
   horizontal = T,  # Vertrauensintervall des Median
   notch=TRUE,  # ja: horizontal
   main="pH",  # Titelei
   col="gray",  # Farbe
   ylim=c(0,15)  # y-Achsenskalierung
)  # Boxplot Ende
n <- aggregate(artpH$Anzahl, list(artpH$Species), sum)
# Werte zusammenfassen, damit später dasteht: n=soundsoviel
text(14, 1:nrow(n), paste("n=",n[row(as.matrix(n)),2]), adj=0)
# Text einfügen
```

☞ die Funktion `aggregate(artpH$Anzahl, list(artpH$Species), sum)` beinhaltet `aggregate(was, Gruppierung, Berechnungsfunktion)`, das $ bewirkt dabei den Zugriff auf die entsprechende Variable; `text(...)` beinhaltet `text(x-Koord., y-Koord., "was")`. `1:nrow(n)` bedeutet 1 bis Anz. der Zeilen vom Datenobjekt `n`, `paste(...)` ermöglicht Mehreres zusammenzufügen, wie "n="und die entsprechende Zahl: `n[row(as.matrix(n)),2]` – verwendet die aktuelle Zahl aus der Spalte 2, `adj=0` linksbündiger Text – adj=1 würde rechtsbündig bewirken

Entsprechende Koeffizienten wie 25% Quantil usw. lassen sich mit `boxplot.stats(...)` anzeigen. Beispiel aus der Hilfe:

```
x <- c(1:100, 1000)  # Daten 1 bis 100 und 1000 generieren
str(boxplot.stats(x))  # Struktur anzeigen lassen
```

☞ **Ergebnis:**

```
List of 4
$ stats: num [1:5] 1 26 51 76 100 ☞ Minimum, 25%, 50% (Median), 75%, Maximum
$ n    : num 101 ☞ Anzahl von nicht-NA Werten
$ conf : num [1:2] 43.1 58.9 ☞ Grenzen der notches
$ out  : num 1000 ☞ gekennzeichnete Ausreißer
```

3.2.4 Scatterplot - Linienplot

Die Funktion `plot(...)` zeichnet eine Punktgrafik – auch mit verschiedenen Linientypen.

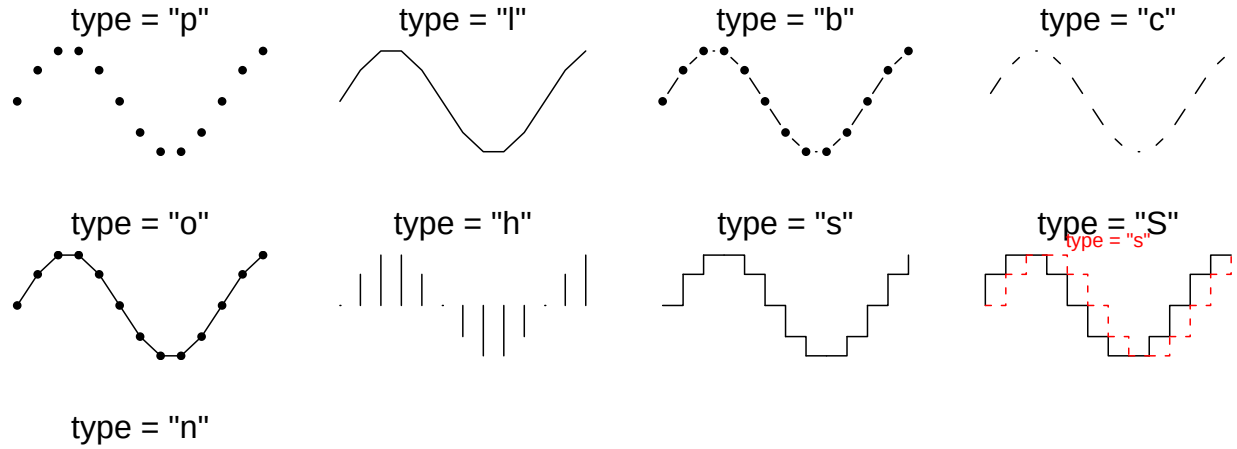
Punktgrafik

```
data(airquality)  # R-eigene Daten laden: New York Air Quality Measurements
?airquality  # Info Hilfe zu Datensatz
attach(airquality) # Daten in den Suchpfad von R
max(Ozone, na.rm=TRUE) -> maximum # Maximumwert von Ozone
plot(Wind~Temp, # airquality$Wind ist jetzt nicht mehr nötig, R weiß wo zu suchen ist
  col=rainbow(5)[Month-4],  # 5 Regenbogenfarben zeichnen bezüglich Monat
  pch=16,  # Punkttyp gefüllt s. auf Seite 28
  main="Luftdaten New York \n 1973", # Titelei - \n ist gleich Zeilenumbruch
  xlab="Temperatur (°F)",  # x-Achsenbeschriftung
  ylab="Wind (mph)",  # y-Achsenbeschriftung
  cex=Ozone/maximum*3, # Ozonwerte als Punktgröße
) # Funktion plot() hier zu Ende
# Legende dazu
sequenz <- seq(0.4,3, length.out=5)  # Sequenz für Punktgröße
sequenz.Ozone <- round(sequenz*maximum/3,0)  # Sequenz für Ozonwerte
legend("bottomleft",  # Position
  title="Ozone (ppb)",  # Titelei
  legend=paste(sequenz.Ozone, "- (Monat ", 5:9,")",sep="") , # Text: 'Monat 1', 'Monat 2', ...
  col=rainbow(5),  # Farbenhintergrund f. points
  pch=16,  # Kreis als Punkt
```

...Fortsetzung umseitig

```
  pt.cex=sequenz,  # Punktgröße
  bty = "n" # keine Box zeichnen
)  # Funktion legend() hier zu Ende
                                        Grafik mit Linien
X <- 1:30  # x-Koordinaten
Y1 <- runif(30)  # Zufallsdaten 0...1
Y2 <- rpois(30,4)  # Poisson-verteilte Daten: Mittel=4
Y3 <- 1:30  # 1, 2, 3 ... 4
plot(X, Y1,  # Daten
  type='l',  # Typ Linie s.u.
  col='red',  # Farbe s. auf Seite 45
  ylim = range(Y1, Y2, Y3)  # WICHTIG: hier wird maximaler range genommen
)
lines(X, Y2, col='green', lty=2)  # Linientyp numerisch s. auf Seite 27
lines(X, Y3, col='blue', lty=2808) # Linientyp manuell s. auf Seite 27
```

Verschiedene Linientypen sind durch Eingabe von z.B.: `type="l"`



`type="n"` zeichnet weder Punkte noch Linien:

```
x <- 0:12  # x-Werte
y <- sin(pi/5 * x)  # y-Werte
original <- par(mfrow = c(3,3), mar = 0.1+ c(2,2,3,1))
# Grafikeinstellungen speichern s. auf Seite 26
for (TYP in c("p","l","b",  "c","o","h",  "s","S","n")) {
  plot(y~x, type = TYP,
    main = paste("plot(..., type = \"",TYP,"\")",sep=""))
  if(TYP == "S") {  # nur für 'S'
    lines(x,y, type = "s", col = "red", lty = 2)
    mtext("lines(... , type = \"s\", ...)", col = "red", cex=.8)
  }  # if() Ende
}  # for() Ende
par(original)  # Grafikeinstellungen zurück
```

3.2.5 Scatterplot + Marginalhistogramm

Die Funktion `s.hist()` aus dem Paket `ade4` zeichnet Punktediagramme mit Marginalhistogrammen

```
data(airquality)  # Daten laden
?airquality # Hilfe dazu anzeigen lassen
names(airquality)  # Variablen anschauen
o3temp <- cbind(airquality$Ozone, airquality$Temp)  # Spalten zs.fassen
library(ade4)  # Paket laden
s.hist(na.omit(o3temp), clab=0, cbr=4)  # Labelgröße=0, 4 Klassen
detach(package:ade4)  # Paket wieder entfernen
```

☞ die Daten enthalten `NA` Werte, daher muß hier `na.omit(...)` verwendet werden

manuelle Variante

```
# Beispiel aus layout graphics
def.par <- par(no.readonly = TRUE)  # Grafikeinstellungen 'speichern'
  x <- pmin(3, pmax(-3, rnorm(50)))  # Maximum
  y <- pmin(3, pmax(-3, rnorm(50)))  # Minimum
  xhist <- hist(x, breaks=seq(-3,3,0.5), plot=FALSE)  # Daten für x-Seite
  yhist <- hist(y, breaks=seq(-3,3,0.5), plot=FALSE)  # Daten für y-Seite
  top <- max(c(xhist$counts, yhist$counts))  # Fixpunkt für Außenbereiche
  xrange <- c(-3,3) # x-Bereich
  yrange <- c(-3,3) # y-Bereich
  nf <- layout(matrix(c(2,0,1,3),2,2,byrow=TRUE), c(3,1), c(1,3), TRUE)
  layout.show(nf) # zeige Layout
  par(mar=c(3,3,1,1))  # Randeigenschaften: bottom, left, top, right (S. 26)
   # Punktegrafik
  plot(x, y, xlim=xrange, ylim=yrange, xlab="", ylab="")
  par(mar=c(0,3,1,1))  # Randeigenschaften: bottom, left, top, right (S. 26)
   # Balkendiagramme
  barplot(xhist$counts, axes=FALSE, ylim=c(0, top), space=0)
  par(mar=c(3,0,1,1)) # Randeigenschaften: bottom, left, top, right (S. 26)
  barplot(yhist$counts, axes=FALSE, xlim=c(0, top), space=0, horiz=TRUE)
par(def.par) # Grafikeinstellung aus def.par lesen und zurücksetzen
```

3.2.6 Scatterplotmatrix

Die Funktion `pairs(...)` ermöglicht sog. Scatterplotmatrizen abzubilden. Somit läßt sich schnell ein Überblick über die Daten gewinnen.

```
pairs(laenge~breite, pch=16, data=meinedaten)  oder
pairs(air2[,1:3], pch=16)
```

☞ `laenge~breite laenge` gegen `breite` auftragen, `pch=16` Punktsymbol: ●, `data=meinedaten` Datenherkunft s. Anmerkung Kapitel 3 auf Seite 24; `air2[,1:3]` aus dem Datenobjekt `air2` Spalten 1 bis 3 (`1:3`)

mit Glättungskurve - panel = panel.smooth

```
data(airquality) # Datensatz aus der Hilfe
?airquality # Hilfe dazu anzeigen lassen
pairs(airquality, panel = panel.smooth, main = "airquality data")  # mit Glättungskurve
```

Gruppenzugehörigkeit - factor(...)

```
zuf1 <- seq(60)*runif(60)  # 1, 2,...60 mal Zufallszahl (0..1)
zuf2 <- seq(60)*rnorm(60)  # 1, 2,...60 mal Wert (0..1) der Normalverteilung
zuf <- cbind(zuf1, zuf2)  # Daten zusammenführen
teilung <- round(runif(60),0)  # 1 0 1 0 0 1 0 0 1 1...
par(xpd=T)  # Legende außerhalb der Grafik möglich
pairs(zuf,  # zu zeichnendes Objekt
  main="Zufallszahlen",  # Titel
  bg=c("yellow","green")[factor(teilung)],  # HG-Farbe abh. von Teilung
   # col=c("yellow","green")[factor(teilung)],   # FG-Farbe abh. von Teilung
```

...Fortsetzung umseitig

```
  pch=21,  # Punkttyp s.S. 28
  gap=0,  # keine Lücke zw. Plots
  oma=c(4,4,6,10),  # Randänderung s.S.27 - geht nur so - nicht über par()
  labels=c("Zahl\n 1","Zahl\n 2"), # Beschriftung
   # row1attop = FALSE  # Diagrammitte / statt \
)
legend(0.85, 0.9,  # x, y Koordinaten
  horiz=FALSE,  # vertikal [default]
  legend=c("0","1"),  # Beschriftung
  cex=0.7,  # Verkleinerung
  pch=21,  # Punkttyp s.S. 28
  pt.bg=c("yellow","green"),  # Punktfarbe
  title="Teilung")  # Titel
par(xpd=F)  # außerhalb zeichnen zurücksetzen
```

☞ bg Hintergrundfarbe (background color)

3.2.7 Blasendiagramme – Bubbleplots

gstats

Das Paket gstats bietet die Möglichkeit Blasendiagramme darzustellen z.B. für die Visualisierung geografischer Daten. Es sei auch darauf hingewiesen, das die Größe aller möglichen Punktsymbole (S. 28) mittels der Angabe in plot(..., cex=1.2) oder cex=Datenbezug an Daten binden läßt.

```
require(gstat) # Paket gstat laden
data(meuse) # Datensatz laden
?meuse # Hilfe dazu anzeigen lassen
bubble(meuse, max = 2.5, main = "Cadmium Konzentrationen (ppm)",
  key.entries = c(.5,1,2,4,8,16))  # Legenden Eintrag
bubble(meuse, "x", "y", "zinc", main = "Zink Konzentrationen (ppm)",
  key.entries = 100 * 2^(0:4))
                                    Punktgröße datenabhängig
x <- rnorm(10)  # Zufallszahlen Normalverteilung
y <- runif(10)  # Zufallszahlen gestreut
plot(x, y, pch="z", cex=y * 5)  # Punktsymbole pch S. 1
                                Blasen einfach über Punktgröße - 'cex'
data(airquality)  # R-Datensatz laden
?airquality  # Hilfe anschauen
attach(airquality)  # Daten in den Suchpfad aufnehmen
max(Ozone, na.rm=TRUE) -> O.max  # Maximumwert von Ozone
plot(Wind~Temp,  # airquality$Wind ist jetzt nicht mehr nötig, R weiß wo zu suchen ist
  col=rainbow(5)[Month-4],  # 5 Regenbogenfarben zeichnen bezüglich Monat
  pch=16,  # Punkttyp gefüllt auf Seite 28
  main="Luftdaten New York \n 1973",  # Titelei - \n ist gleich Zeilenumbruch
  xlab="Temperatur (°F)",  # x-Achsenbeschriftung
  cex=Ozone/O.max*3,  # Ozonwerte als Punktgröße
  ylab="Wind (mph)"   # y-Achsenbeschriftung
)  # Funktion plot() hier zu Ende
                                            Legende dazu
(sequenz <- seq(0.4, 3, length.out=5))  # Sequenz für Punktgröße: von 0.4 bis 3, Länge 5
 [1] 0.4 1.1 1.7 2.4 3.0
(sequenz.Ozone <- round(sequenz*O.max/3,0))  # Sequenz für Ozonwerte
 [1]  22  59  95 132 168
```

...Fortsetzung umseitig

```
legend("bottomleft",  # Position
  title="Ozone ppb",  # Titelei
  legend=paste(sequenz.Ozone, "- (Monat ", 5:9,")",sep="") , # Text: 'Monat 1', 'Monat 2', ...
  col=rainbow(5),  # Farbenhintergrund f. points
  pch=16,  # Kreis als Punkt
  pt.cex=sequenz,  # Punktgröße
  bty = "n"   # keine Box zeichnen
)  # Funktion legend() hier zu Ende
```

Fallen mehrere Datenpunkte auf ein und dieselbe Koordinate, kann man dies mit `sizeplot(..)` aus dem Paket `plotrix` visualisieren:

```
library(plotrix)  # Paket laden
  x <- c(0.1,0.1,0.1,0.1,0.1,0.2,0.2,0.2,0.2,0.3,0.3)
  y <- c( 1,  1,  1,  1,  2,  2,  2,  3,  3,  4,  5 )
  plot(x,y)
  table(y,x) # Tabelle zusammenfassen
    x
 y   0.1 0.2 0.3
   1   4   0   0
   2   1   2   0
   3   0   2   0
   4   0   0   1
   5   0   0   1
  sizeplot(x,y)
  sizeplot(x,y,pch=2)
detach(package:plotrix)  # Paket wieder entfernen
```

3.2.8 3D Scatterplots

Das Paket `scatterplot3d()` zeichnet 3D-Punktediagramme. Siehe auch Paket `rgl` für echte 3D Grafiken.

```
require(scatterplot3d)    # Paket laden oder library(scatterplot3d)
  z <- seq(-10, 10, 0.01) # Daten erzeugen: -10.00, -9.99, -9.98, -9.97,...
  x <- cos(z) # neue Zuweisungen mit cos(...) u. sin(...)
  y <- sin(z) # und sin(...)
  grafik3D <- scatterplot3d(x, y, z,
    highlight.3d=TRUE,
    col.axis=TRUE,
    col.grid="lightblue",
    main="Titel scatterplot3d - 1",
    pch=20, # Punkttyp s. auf Seite 28
    scale.y = 0.6
  )
names(grafik3D) # Funktionen bezüglich gezeichneter Grafik
# [1] "xyz.convert" "points3d"    "plane3d"     "box3d"
## Y Achse ändern
Yachse <- pretty(y, n=25) # gibt guten Zahlenabstände min mglst. n=25
  y.xy <- grafik3D$xyz.convert(# konvertiert xyz Punkte zu x-y Punkten
    x=rep(max(x), length(Yachse)), # max(x) n-Mal Länge Yachse
    y=Yachse,
    z=rep(min(z),length(Yachse))   # min(z) n-Mal Länge Yachse
  )
  segments(# zeichnet Linien
    x0=y.xy$x,
    y0=y.xy$y,
    x1=y.xy$x+0.1,
```

```
    y1=y.xy$y
  )
```

3.2.9 Punktreihen – Dotchart

Eine Abwandlung des Scatterplots ist die Funktion `dotchart(...)`. Ein Beispiel aus der Hilfe:

```
data(islands)  # Daten einlesen
?islands  # Hilfe dazu anzeigen lassen
dotchart(log(islands[order(islands)], 10), pch=16,
main = "Areas of the World's Major Landmasses \n log10(area) (log10(sq. miles))")
```

☞ `log(..., 10)` Darstellung lg_{10}, `islands[order(islands)]` Daten sortieren, `pch=16` Punkt: •, `main=""` Titelei `\n` macht Zeilenumbruch; die Funktion `dotchart2(...)` aus dem Package `Hmisc` bietet noch weitere Optionen

3.2.10 Werteplot + Fehlerbalken – `centipede.plot(..)`

Eine ähnliche Grafik, die Mittelwerte + Fehlerbalken zeigt ist `centipede.plot(..)` aus dem Paket `plotrix`. Es müssen hierbei aber nicht Mittelwerte (Funktion `mean`) sein, sondern es kann jede andere R-Funktion angegeben werden.

```
library(plotrix)  # Paket laden
   # example(centipede.plot) # Beispiel für Mittelwertgrafik
  (sample <- list("",40)) # Beispieldaten als Liste erstellen
   # für i gleich 1 bis 40 ...
  (for(i in 1:40) sample[[i]] <- rnorm(sample(1:8,1)*50))
  str(sample)  # Struktur: Listen mit 40 Teillisten
  (sample.segs <- get.segs(sample))  # für jede Teilliste: mean & +/- std.err & n
  centipede.plot(sample.segs,
    main="Test centipede plot\nhier Mittelwerte + Fehlerbalken", # Titel
    vgrid=0) # vertikale Linie bei 0
detach(package:plotrix)  # Paket eventuell wieder entfernen
```

Weitere Möglichkeiten Fehlerbalken darzustellen:

example(plotCI) Paket `plotrix` – Konfidenzgrenzen als Fehlerbalken in x oder y-Richtung

3.2.11 Polygonplot

Verbindet man die Funktion `plot(...)` und `polygon(...)`, lassen sich Polygonplots zeichnen.

```
y <- rnorm(10)  # 10 Zufallszahlen
x <- seq(1,5, length=10)  # 10 Zahlen von 1 "bis" 5
plot(y~x, type="n")  # Plot zeichnen, aber keine Punkte
polygon(x, y, col="grey")  # Polygon einzeichnen
```

☞ das Beispiel läßt sich auch kopieren...

3.2.12 Tiefendiagramme

Nach Kopieren der Funktionen `plot.depth(...)` auf Seite 175 und `line.labels.add(...)` (S. 183) aus dem Anhang in die R-Konsole, kann man die folgende Tiefen-Diagramme darstellen. Es empfiehlt sich das Paket `Hmisc` zu installieren, damit Teilintervalle als Striche verfügbar sind.

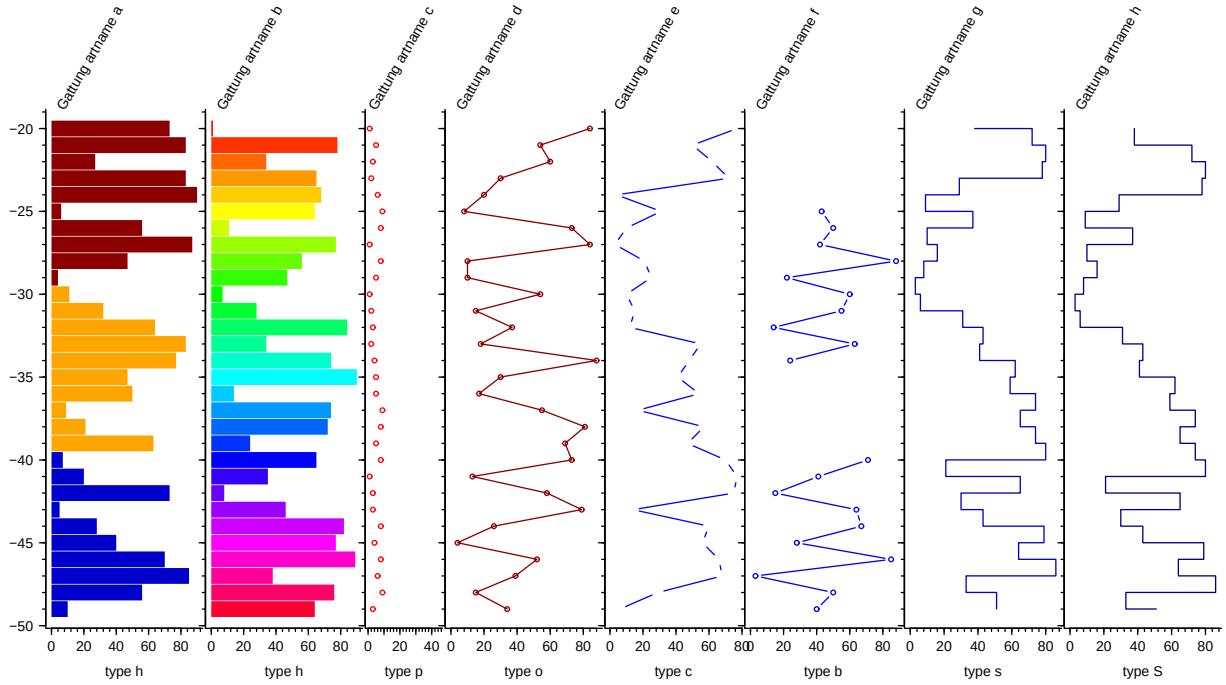
Die erste Spalte im Datensatz ist explizit für die Tiefe vorgesehen, d.h. zum Zeichnen der y-Achsen. Daher sollte sie auch an erster Stelle sein. Man kann dieses Verhalten auch abschalten mit der Option `yaxis.first=FALSE`, aber das macht meist kein Sinn, denn die Daten werden dann von 1 bis Anzahl-der-Zeilen gezeichnet mit Zeilennamen, wenn vorhanden – also quasi als Kategorien behandelt. Um Daten von Excel oder über die Zwischenablage einzulesen s. auf Seite 13.

Zufallsdatensatz generieren

```
test <- data.frame( # 30 Tiefendaten + 8 Spalten mit Zufallszahlen
  "tiefe"= tiefe <- 0:(-29)-20, # (0, 1, 2, ...) − 20
  a <- sample(90, 30, replace = TRUE), # 30 Zufallszahlen 0...90, Wdh. möglich
  b <- sample(90, 30, replace = TRUE), # 30 Zufallszahlen 0...90, Wdh. möglich
  c <- sample(9, 30, replace = TRUE), # 30 Zufallszahlen 0...90, Wdh. möglich
  d <- sample(90, 30, replace = TRUE), # 30 Zufallszahlen 0...90, Wdh. möglich
  e <- sample(90, 30, replace = TRUE), # 30 Zufallszahlen 0...90, Wdh. möglich
     # Daten mit fehlenden Werten: NA
  f <- c( # c() = combine - kombiniere
    rep(NA, 5),  # 5 x NA
    sample(90, 10, replace = TRUE),  # 10 Zufallszahlen von 0-90; Zahlendopplung TRUE
    rep(NA, 5),  # 5 x NA
    sample(90, 10, replace = TRUE)), # 30 Zufallszahlen 0...90, Wdh. möglich
  g <- sample(90, 30, replace = TRUE), # 30 Zufallszahlen 0...90, Wdh. möglich
  h <- g # Daten von g nochmal kopieren
)
# Spaltennamen erzeugen aus "Gattung artname"+ Buchstabe
colnames(test)[2:ncol(test) ] <- paste("Gattung artname",letters[1:(ncol(test)-1)])
colnames(test) # Spaltennamen anzeigen
```

```
# Voreinstellung von plot.depth(...) ansehen
par(las = 1)  # Ausrichtung Achsenbeschriftung s.  auf Seite 27
plot.depth(test)  # nach Voreinstellung
plot.depth(test, polygon = TRUE)  # mit Polygon + "Fehler"- Meldung wegen fehlender Werte
```

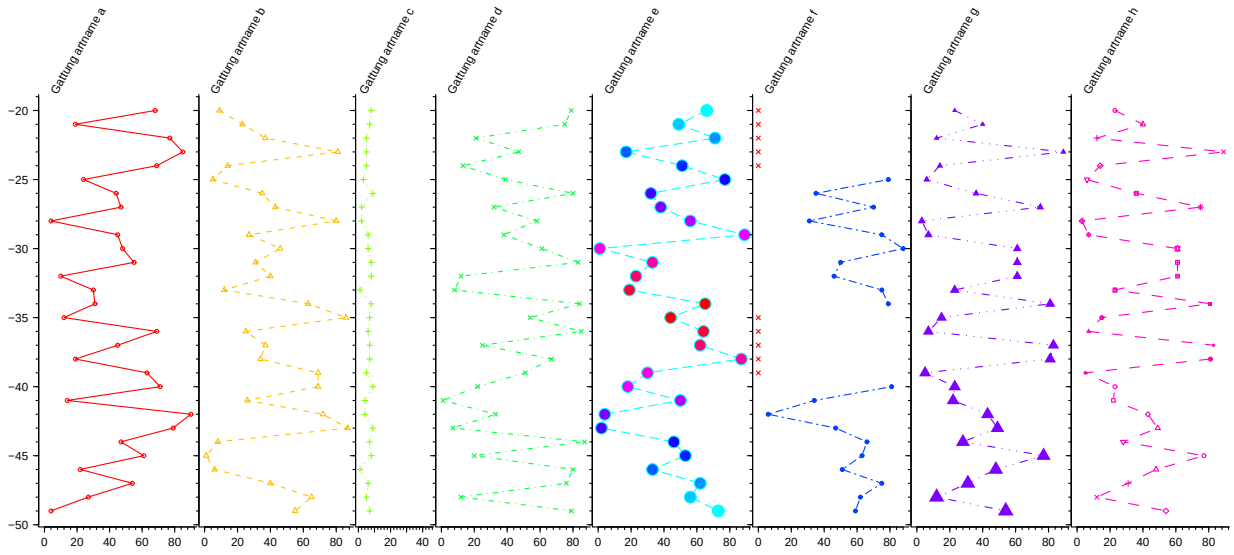
```
                Bsp. für Option plot.type + Farbe - plot.depth(...) (Anhang S. 175)
farbe <-  rep(c("darkred","orange","blue3"), each = 10) # Wörter für Farben generieren
farbe  # Farben ansehen
plot.depth(test,
  plot.type = c("h","h","p","o","c","b", "s", "S") -> type, # Typen allgemein auf Seite 52
  xlabel = paste("type ",type, sep = ""),  # Info für Typen einfügen
  l.width = c(12,12,1,1,1,1,1,1),  # Linienbreiten
  lp.color = list(farbe,  # Grafik 1
    rainbow(30),  # Grafik 2; 30 Regenbogenfarben
    "red", # Grafik 3
    "darkred", # Grafik 4
    "blue1", # Grafik 5
    "blue2", # Grafik 6
    "blue3", # Grafik 7
    "blue4" # Grafik 8
  ) # Ende Farbliste
) # Ende plot.depth()
```



Bsp.: Linientypen /Punkte - plot.depth(...) (Anhang S. 175)

```
# Bsp. Punktgröße /- Farben an Daten anbinden
blaubisrot <- rainbow(15, s=0.5, start=0.6, end=1)  # 15 Regenbogenfarben
p.farbe <- c(blaubisrot, blaubisrot[15:1])  # ergibt blau nach rot nach blau
```

```
plot.depth(test,
  plot.type = "o",  # alles Punkt-Linie; Typen allgemein auf Seite 52
  p.type = list(1,2,3,4,21,16,17,c(1:25,1:5)), # Punkttypen allgemein auf Seite 28
     # fuer letzten plot "c(1:25,1:5)": alle Punkttypen
  p.bgcolor = list( # Punkt Hintergrund
    "white","white","white","white",  # Grafiken 1, 2, 3, 4
    p.farbe,  # Grafik 5
    "white","white","white"   # Grafiken 6, 7, 8
  ),
   # Punktgröße an Daten anbinden
  p.size = list(1, 1, 1, 1, 3, 1, seq(1,3, length.out=30) , 1),
     # seq(1,3, length.out=30) von 1 bis 3 aber 30 Zahlen; hier natürlich auch Daten mgl.
  l.type = list("solid",2,3,4,5,6,"64141414","88"),
     # numerisch, manuell: "6/4/1/4/1/4/1/4"   farbig/weiß/farbig/w/f/w ... allg. S. 27
  lp.color = rainbow(8)  # 8 Regenbogenfarben
)
```



Bsp.: Beschriftung - plot.depth(...) (Anhang S. 175)

```
plot.new()  # alten Plot löschen
par(las = 1)  # Ausrichtung Achsenbeschriftung
plot.depth(test,
  colnames = c(T,T,F,T,F,F,T,T), # T - TRUE, F - FALSE
  xlabel = list("Anzahl", "%",
    expression(mg%.%l^-1), # mg · l^-1; Ausdrücke s. auf Seite 41
    expression(sum(Arten)),  # Summenzeichen
    expression(paste("%-",O[2])), # % − O2
    expression(CO[3]^paste(2,"  -")), # CO3 2−
    "Anzahl",""
  ),
  subtitle = list("","","","","","","","",
    expression(over(Anzahl[x],Anzahl[y]) ) # Bruch: Anzahl_x / Anzahl_y
  )
)
# falls Titel falsch gezeichnet: Grafikfenster schließen und
# nochmal alles neu zeichnen/bzw kopieren
title("Zufallsdaten eines\nSedimentkernes", ylab = "Tiefe m")
```

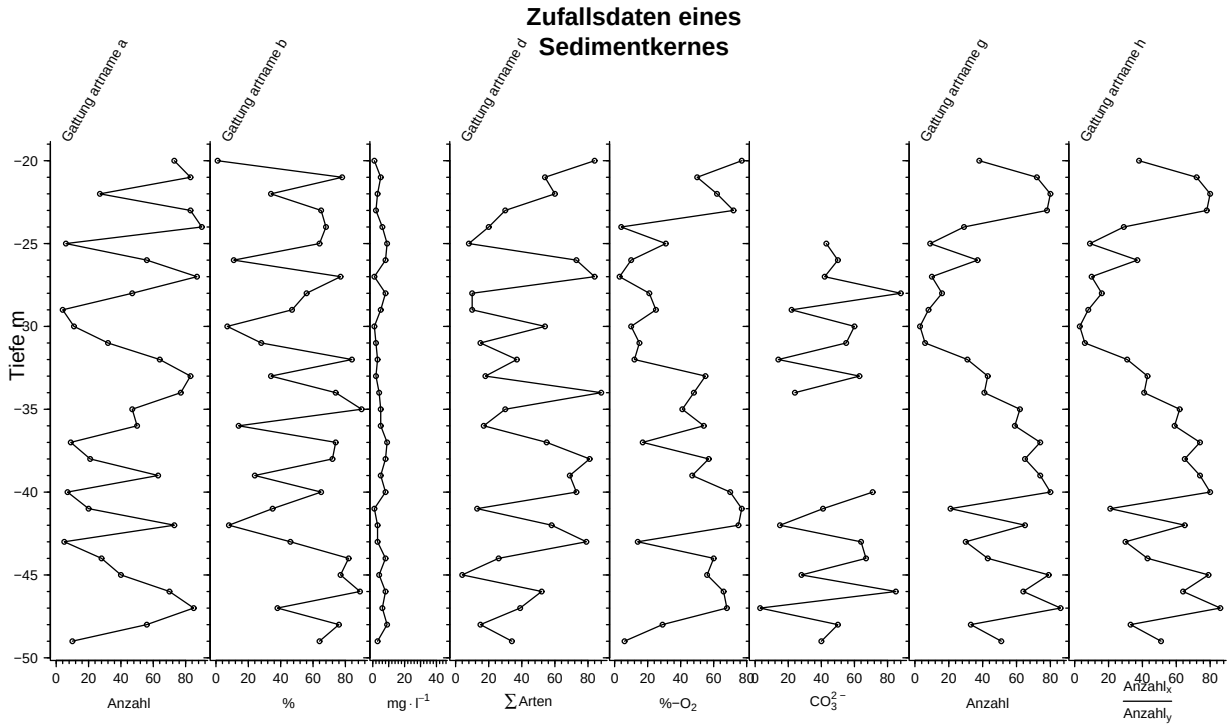
alternativ Titel y-Achsenbeschriftung mit Maus platzieren

```
par(xpd=TRUE) -> original  # auch außerhalb der Grafik zeichnen
   # Text y-Achse mit Maus platzieren mit locator()
  locator(1) -> wo  # mit Maus setzen
  text(wo$x, wo$y, "Tiefe m", srt=90)  # srt=90 Grad drehen
   # mtext("Tiefe m", side=2, line=2)  # alternativ eventuell line=2 2-Zeilen weg vom Rand
   # Text Titel mit Maus platzieren
  locator(1) -> wo  # mit Maus setzen
  text(wo$x, wo$y, "Zufallsdaten eines \nSedimentkernes", font=2 )  # font=2 fett
```

Text kursiv, unterstrichen, ...

```
  locator(1) -> wo  # Punkt mit Maus wählen
  text(wo$x, wo$y,  # Koordinaten
    labels= # hier der eigentliche Text
      expression( # Schriftvarianten
        Überschrift ~ bold(fett) ~
        italic(kursiv) ~
        underline(unterstrichen) ~
        bolditalic(fettkursiv)
      ),
    cex=1.5 # Schriftvergrößerung
  )
  text(wo$x,wo$y-diff(par()$usr[3:4])/10 , # Koordinaten y minus 10-tel Grafikregion; Info:
    "Überschrift ~ bold(fett) ~ italic(kursiv) ~ underline(unterstrichen) ~  bolditalic(fettkursiv)",
    cex=0.8  # Verkleinerung 80%
  )
par(original)  # Grafikeinstellungen für xpd wieder zurück
```

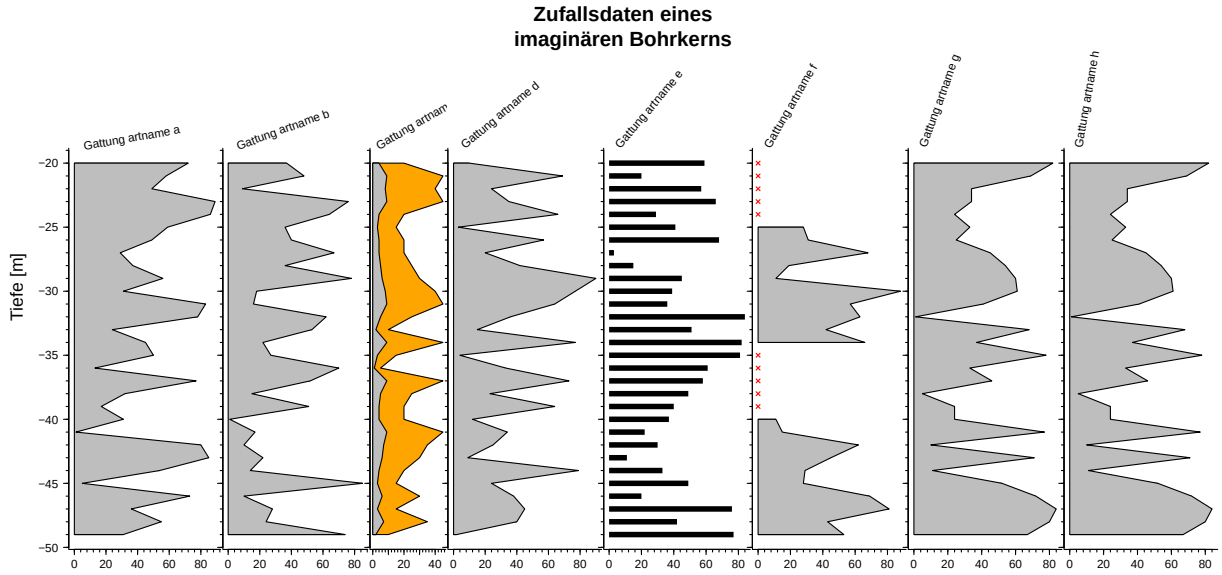
☞ Die Spaltenbeschriftung kann auch „manuell“ mit der Maus vorgenommen werden: mit Option `locator=TRUE` und dann linke untere Ecke = Mauspunkt


Zufallsdaten eines Sedimentkernes
Tiefe m
−20
−25
−30
−35
−40
−45
−50
Gattung artname a
Gattung artname b
Gattung artname d
Gattung artname g
Gattung artname h
0 20 40 60 80
Anzahl
%
mg · l−1
∑ Arten
%−O2
CO3 2−
Anzahl
Anzahlx / Anzahly

Skalieren von wenig Daten - plot.depth(...) (Anhang S. 175)

```
par(las = 1)
plot.depth(test,
  plot.type = "h", # Option polygon zeichnet hier drüber; Typen allgemein auf Seite 52
  mar.outer = c(1,10,4,1), # mehr Rand rechts
  mar.top = 12, # mehr Rand oben
  polygon = c(T,T,T,T,F,T,T,T), # T - TRUE, F - FALSE
  rotation = c(1:8)*10,  # 10 20 30 ... 80
  l.width = c(1,1,1,1,5,1,1,1),  # Histogramme imitieren
  min.scale.level = 0.12,  # 12%-Schranke vom maximalsten Wert in den Daten
  min.scaling = c(F,F,5,F,F,F,F,F), # T - TRUE, F - FALSE
  color.minscale = "orange"   # Farbe
)
title("Zufallsdaten eines\nimaginären Bohrkerns", ylab = "Tiefe [m]")
```

☞ fehlende Werte (NA) werden als rote x dargestellt. Abschalten über Option show.na=FALSE



Um Markierungslinien mit/ohne Beschriftung zu Grafiken hinzufügen am besten die Funktion `line.labels.add(...)` aus dem Anhang S. 183 benutzen:

Markierungslinien + zusätzliche Grafiken - line.labels.add(...) (Anhang S. 183)

```
plot.new()  # alten Plot löschen
par(las=1)  # Ausrichtung Achsenbeschriftung
```

...Fortsetzung umseitig

Gitternetz + zusätzliche Grafiken

```
plot.depth(test,
    # bevor die eigentliche Grafik gezeichnet wurde:
  plot.before=expression(c(par(xpd=F),grid())),
     # Gitternetz für alle dazu, aber nicht außerhalb zeichnen
    # nachdem die eigentliche Grafik gezeichnet wurde:
  plot.after=list(
    NULL, # 1. Grafik
    expression( # an 2. Grafik folgendes ausführen
      # was an erster Stelle ist, wird auch zuerst ausgeführt
      # Daten des 4. Graphen als imitiertes Balkendiagramm
      segments(0, test[,1], test[,4], test[,1], lend=2, lwd=10, col="darkred"),
      # Daten des 4. Graphen als Linie
      # lines(x=test[,4],y=test[,1], col="darkred", lty="solid", pch=16, type="o")
      # Daten des 2. Graphen als Linie
      lines(x=test[,2],y=test[,1], col="red", pch=21 , lty="dotted", type="o", bg="white")
    ),  # end expression(..)
    NULL, NULL, NULL, NULL, NULL, NULL
     # 3. 4. 5. 6. 7. 8. Grafik
  ),  # end list(..) in Option 'plot.after'
  axis.top=list( # top lables
    c(TRUE, FALSE),  # 1. Striche-ja, Zahlen-nein
    c(T, T), # 2.
    c(F, T), # 3.
    c(F, F), # 4.
    c(T, F), # 5.
    c(T, F), # 6.
    c(T, F), # 7.
    c(T, F) # 8.
  ),
  colnames=c( # Spaltennamen
    T, F, T, T, T, T, T, T
  )
) # Ende plot.depth(..)
```

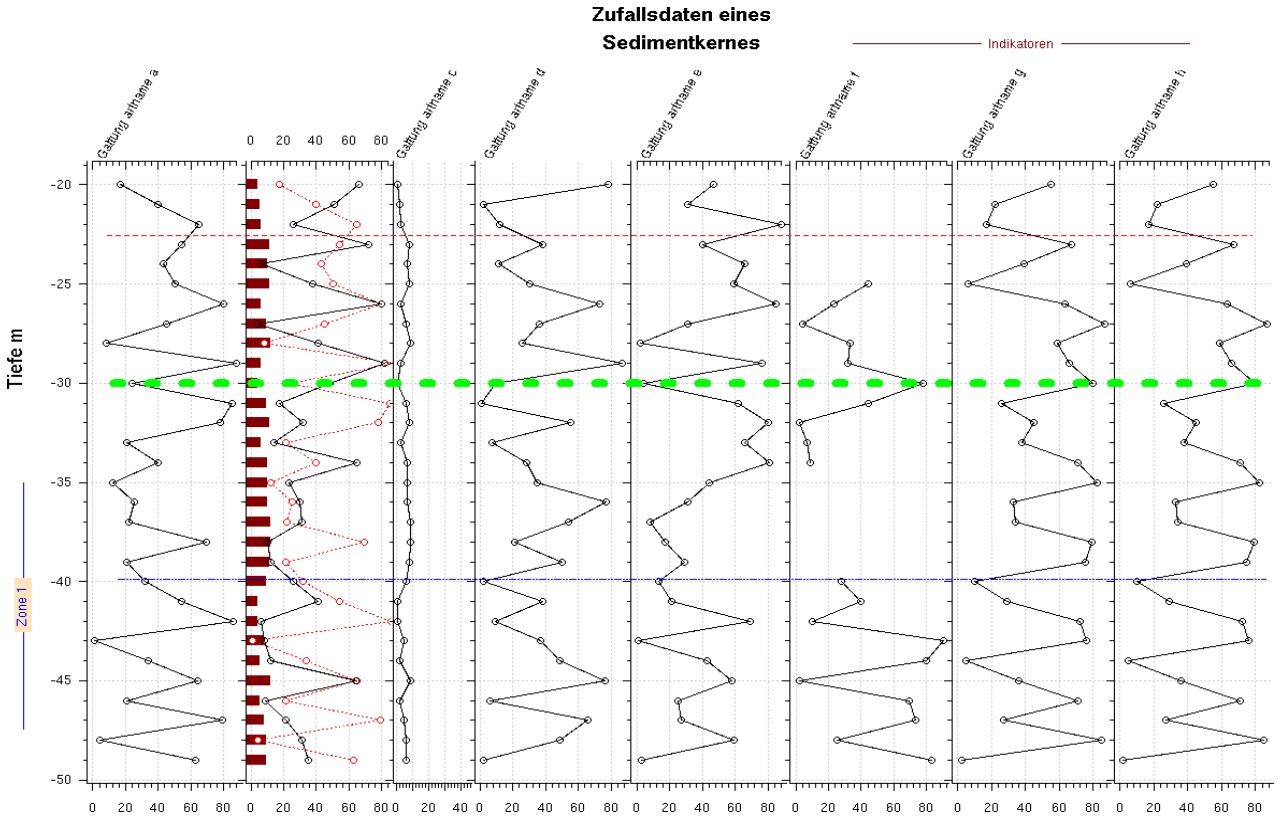
Markierungslinie mit Text waagerecht-normal dazu

```
# line.labels.add(1)  # Voreinstellung
line.labels.add(1,text="Indikatoren", text.scale=0.7)
# Linie mit Text senkrecht
line.labels.add(1,text="Zone 1",
  text.scale=0.7,
  orientation="v",  # vertikal
  color="blue",  # Farbe Linie + Text
  color.bg="bisque",  # Farbe Hintergrund
  ypad=1  # etwas mehr Rand für y-Richtung
)
# Nur Linien: Linientypen Farben
line.labels.add(3,  # 3 Linien
  color=rainbow(3),  # 3 Regenbogenfarben
  l.width=c(1,8,1),  # Linienbreiten
  l.type=c("dashed","dotted","621212")
    # Linientypen 2x 'normal'  1x _ _ _ manuell; Typen s. auf Seite 28
)
title("Zufallsdaten eines\n Sedimentkernes")
```

...Fortsetzung umseitig

```
# y-Achsenbeschriftung manuell setzen
par(xpd=TRUE) -> original  # auch außerhalb der Grafik zeichnen
  locator(1) -> wo  # mit Maus setzen
   # Text y-Achse mit Maus platzieren
  text(wo$x, wo$y, "Tiefe m", srt=90)  # srt=90 Grad drehen
   # mtext("Tiefe m", side=2, line=2)  # alternativ eventuell line=2 2-Zeilen weg vom Rand
par(original)  # Grafikeinstellungen für xpd wieder zurück
```

☞ ein einfaches Benutzen der Linienfunktion, wie `line.labels.add(1)` reicht aus, um dunkelrote, durchgezogene Linien zu zeichnen



Gleichskalierung Achsen FALSE; zusätzliche chemische Parameter + Achsen

```
# Daten erweitern um 2 Spalten nach der ersten mit cbind(..)
testabiotic <- cbind( # Spalten zusammenfügen
  test[,1],  # Tiefen Daten aus 1.Spalte von 'test'
  "TOC"= toc <- sample(30, replace=TRUE), # 30 Zufallszahlen, Wdh. möglich
  "deltaN"=deltaN <- rnorm(30),  # 30 Zufallszahlen um 0 herum
  test[,-1]  # der Rest Daten ohne 1.Spalte
)
# Text für Achse
text <-  letters[1:12]  # Bsp.Text für Zonen
# Achsenpositionen an besonderen Stellen
wo.y <- c(-20, -26, -27, -28, -33, -35, -38, -40, -42,-46, -48, -49)
```

...Fortsetzung umseitig

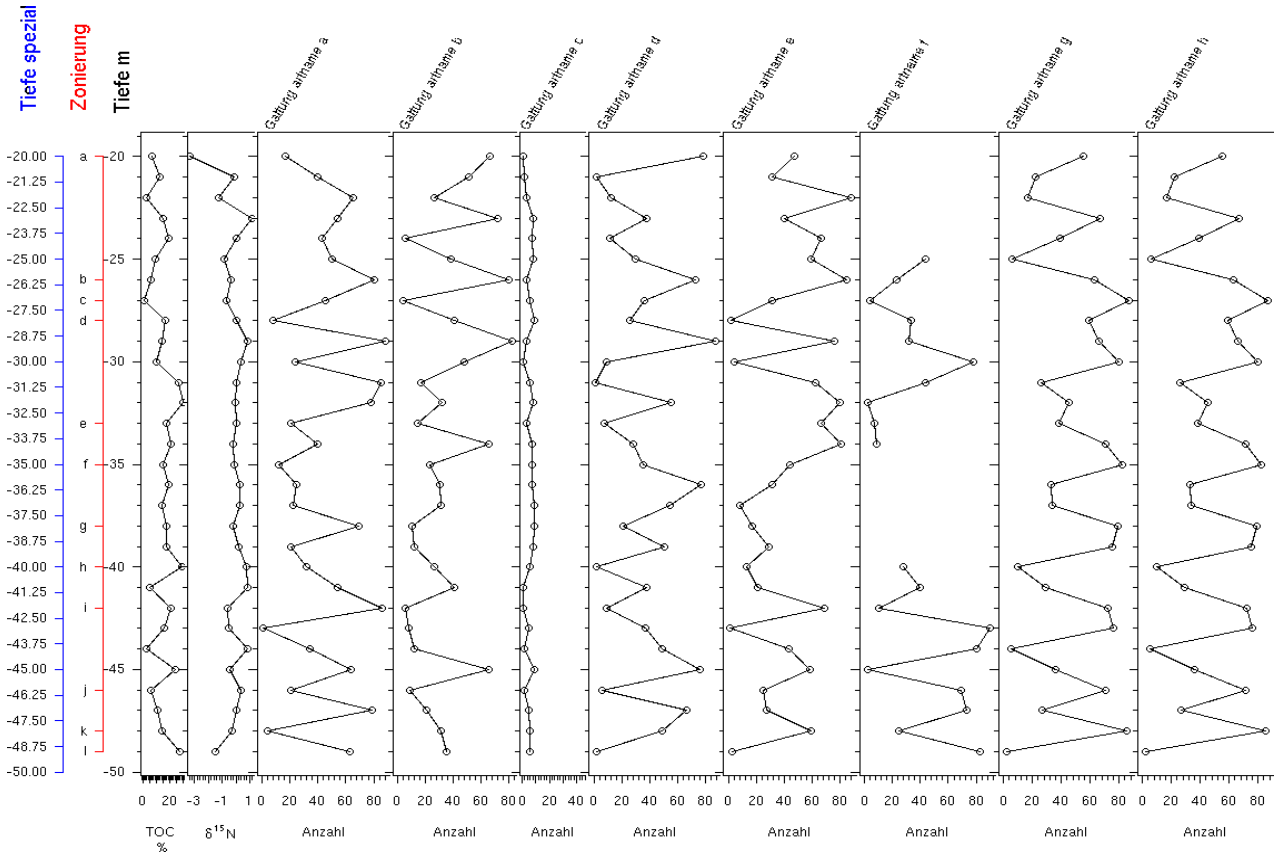
```
par(las=1)  # Achsenausrichtung, falls noch nicht gesetzt
# zusätzliche Achse als plot.before
plot.depth(testabiotic,
  mar.outer = c(1, 12, 4, 1),  # c(unten , li , oben, re) links etwas mehr Platz
  bty ="c", # Box ähnelt dem Großbuchstaben
  plot.before =list(
    expression(  # 1. Grafik
      axis(side=2, labels=text, at=wo.y, pos=-30, col="red"),
       # y-Achse mit text + at: wo die ticks sind, pos: x-Position
      axis(side=2, pos=-60, col="blue", yaxp=c(-20, -50, 24))
       # yaxp=c(-20, -50, 24) von -20 bis -50 sollen 24 Intervalle sein
    ),
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
     # 2. 3. 4. 5. 6. 7. 8. 9. 10. Grafik
  ),
  xaxes.equal = c(F,F,rep(T, 8)),  # FALSE, FALSE, + 8x TRUE
  colnames = c(F,F,rep(T, 8)),  # FALSE, FALSE, + 8x TRUE
  xlabel = list(
    "", # 1. Grafik
    expression(delta^15 ~ N),  # 2. Grafik
    "Anzahl", # 3. Grafik
    "Anzahl", # 4. Grafik
    "Anzahl", # 5. Grafik
    "Anzahl", # 6. Grafik
    "Anzahl", # 7. Grafik
    "Anzahl", # 8. Grafik
    "Anzahl", # 9. Grafik
    "Anzahl" # 10. Grafik
  ),
  subtitle = c(
    "TOC \n%",  # 1. Grafik
    rep("", 9)  # 2. ... 10. Grafik
  )
) # Ende plot.depth(..)
```

Achse beschriften

```
par(xpd=TRUE) -> original  # auch außerhalb der Grafik zeichnen
   # Text y-Achse mit Maus platzieren v. außen n. innen
  locator(1) -> wo  # mit Maus setzen
  text(wo$x, wo$y, "Tiefe spezial (m)", adj=0, srt=90, col= "blue")
   # adj=0 linksbündig; srt=90 Grad drehen
  locator(1) -> wo  # mit Maus setzen
  text(wo$x, wo$y, "Zonierung", adj=0, srt=90, col= "red")
  locator(1) -> wo  # mit Maus setzen
  text(wo$x, wo$y, "Tiefe (m)", adj=0, srt=90)
par(original)  # Grafikeinstellungen für xpd wieder zurück
```

Gleichskalierung für alle Achsen

```
plot.depth(testabiotic,
  min.scale.level=1,
  min.scale.rel=1,
)
```

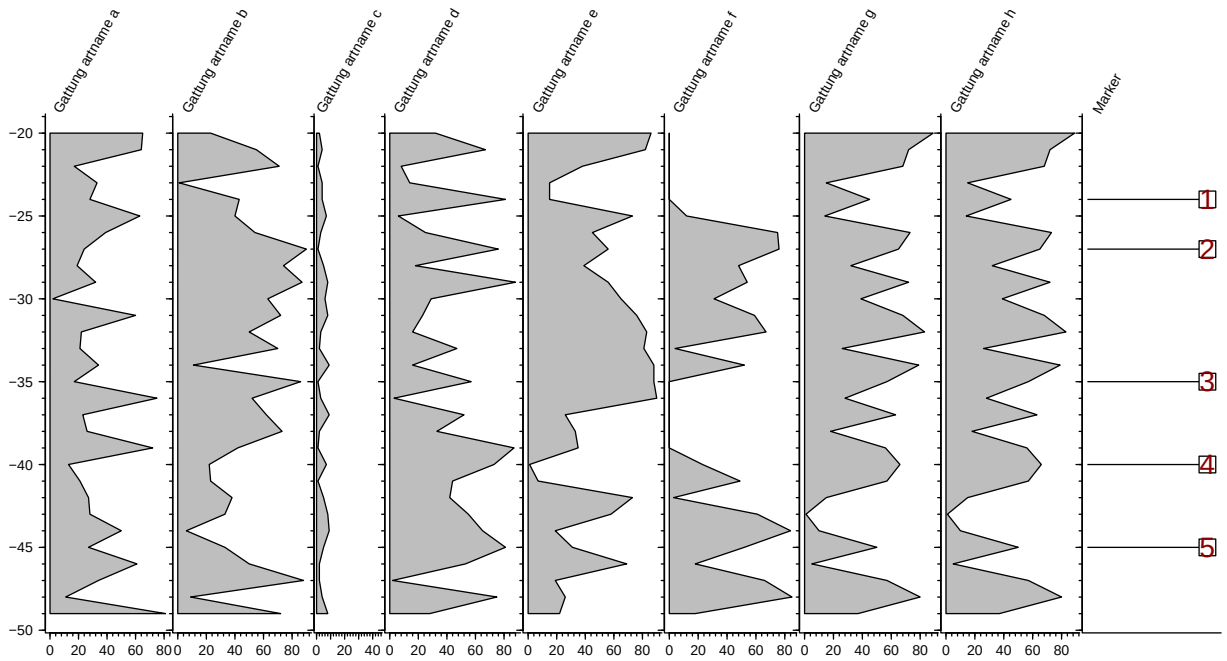


Beispiel mit Markern über Option 'plot.after'

```
# Datensatz um Marker erweitern mit cbind() column-bind;
# WICHTIG wo keine Marker sind, sind fehlende Werte, also NA
testmarker <- cbind(
  test,  # alle Daten von 'test'
  "Marker"  = marker <-  c(
    rep(NA,4),  # 4x NA
    30,  # 1x 30
    rep(NA,2),  # 2x NA
    30,  # 1x 30
    rep(NA,7),  # 7x NA
    30, # 1x 30
    rep(NA,4), # 4x NA
    30,
    rep(NA,4), # 4x NA
    30,
    rep(NA,4) # 4x NA
  )
)
# Spaltenindx wo keine NA-Werte enthalten
marker.wo.index  <- which(!is.na(testmarker[,10]))
# marker.text <- LETTERS[1:5] # BUCHSTABEN Beispiel
# marker.text <- letters[1:5] # Buchstaben Beispiel
marker.text <- 1:5  # 1, 2, 3, 4, 5
```

...Fortsetzung umseitig

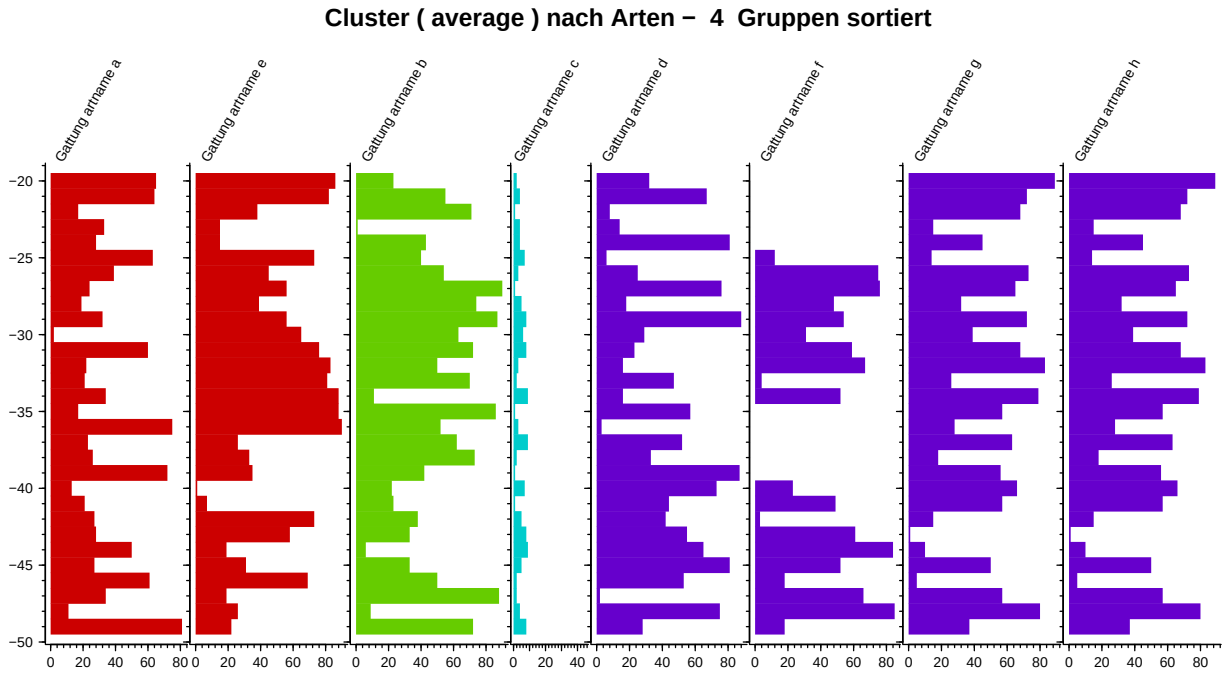
```
par(las=1)  # Ausrichtung Achsenbeschriftung
plot.depth(testmarker,
  xaxis.num=c(rep("s", 8), "n"),  # 8x x-Achse ja, 9. keine
  plot.type=c(rep("n", 8), "h"),  # 8x Grafiktyp "n",  9. "h"-histogrammartig
  polygon=c(rep(TRUE, 8), FALSE ),  # 8x Polygon 9. kein
  plot.after=list( # was NACH dem eigentlichen Zeichnen passieren soll
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
     # 1. 2. 3. 4. 5. 6. 7. 8. Grafik
    expression( # 9. Grafik; WICHTIG die Reihenfolge: Punkte zuerst, dann Text
      points( # Punkte
        y=testmarker[,1], # Spalte 1
        x=testmarker[,10]-2 ,  # Werte Spalte 10 subtrahiert um 2
        pch=22,  # gefülltes Rechteck; Typen allgemein auf Seite 28
        bg="white",  # Punkt-Hintergrund weiß
        cex=4  # 4-fache Vergrößerung
      ),
      text( # Text die eigentlichen Marker
        y=testmarker[marker.wo.index, 1],  # nur wo keine NA enthalten
        x=testmarker[marker.wo.index, 10]-2 ,
        labels=marker.text,  # entsprechender Text
        col="darkred",  # Farbe
        cex=2  # 2-fache Vergrößerung
      )
    ) # Ende 9. Grafik
  ) # Ende plot.after list()
) # Ende plot.depth()
```



Clusteranalyse farbig darstellen

```
# Cluster nach Proben
par(las=1)  # Achsenausrichtung, falls noch  nicht gesetzt
n.gruppen <- 4  # Gruppenanzahl als Variable: praktisch braucht man bloß hier drehen
# Ward Clusteranalyse mit euklidischer Distanz also: dist(...)^2
hclust(dist(test[,-1])^2, method="ward") -> cluster
# Cluster 'kappen'  bei soundsoviel Gruppen (..) bewirkt gleichzeitig Ausgabe
cutree(cluster, k=n.gruppen) -> grp.index
plot.depth(test,
  plot.type="h", # h-histogrammartig
  l.width=12,  # Linienbreite
  lp.color=list(rainbow(n.gruppen)[grp.index])
   # Farben automatisch nach 'n.gruppen'
)
# Titel automatisch nach 'n.gruppen'
title(paste("Cluster (",cluster$method,") nach Proben - ",n.gruppen,"  Gruppen"))
```

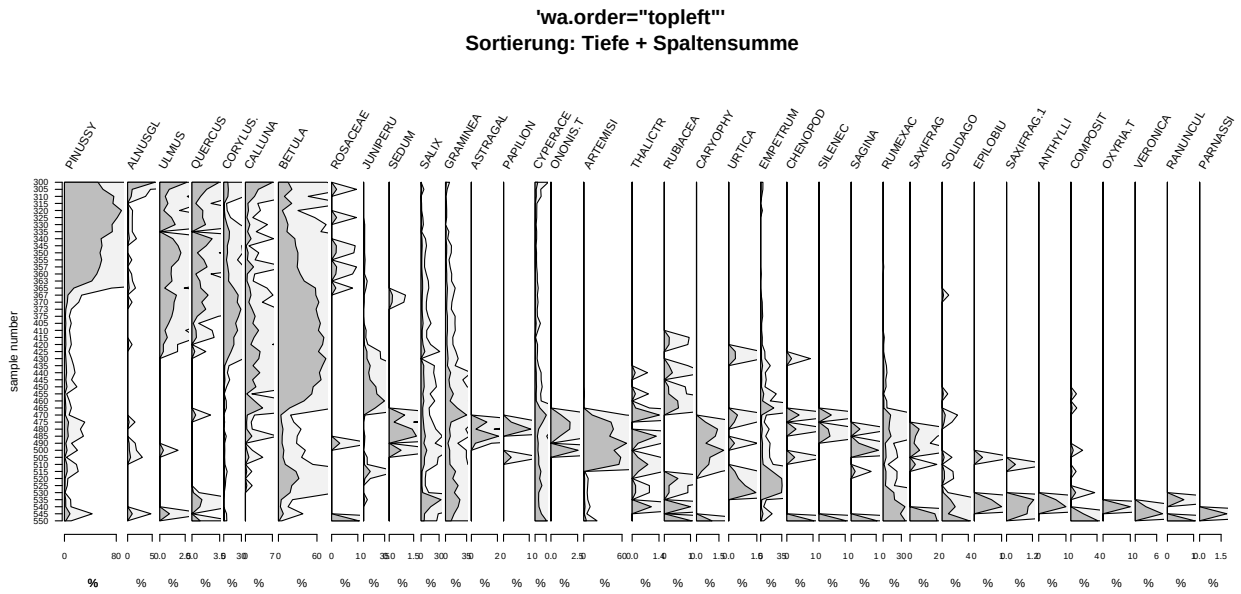
```
# Cluster nach Arten: Daten einfach transponieren mit t()
# Clusteranalyse average mit sinnvollerem Bray-Curtis Distanzmaß s.Distanzmaße
library(vegan) # Zusatzpaket laden; Installation auf Seite 10
hclust(vegdist(t(test[,-1]), na.rm=TRUE), method="average") -> cluster
# Cluster 'kappen'  bei soundsoviel Gruppen (..) bewirkt gleichzeitig Ausgabe
(cutree(cluster, k=n.gruppen) -> grp.index)
(sort(grp.index, index.return=T) -> sort) # Indizes zusätzlich ausgeben
# sort$ix enthält Spaltenindx
# sort$x die gewünschte Gruppierung
plot.depth(test[,c(1,sort$ix+1)],  # Spalte 1 + Spalte nach sortiertem Spaltenindx
   # sort$ix+1 damit die Original-Spalten stimmen, denn 1.Spalte enthält ja Tiefe
  mar.top=12, # mehr Rand oben
  plot.type="h", # h-histogrammartig
  l.width=12, # Linienbreite
  lp.color=rainbow(n.gruppen, v=0.8)[sort$x]
   # Farben automatisch nach 'n.gruppen'+ Sortierung; v=0.8 etwas weniger farbintensiv
)
# Titel automatisch nach 'n.gruppen'
title(paste("Cluster (",cluster$method,") nach Arten - ",n.gruppen , "  Gruppen sortiert"))
```



Um ein Tiefendiagramm nach Tiefe gewichtet zu sortieren kann man die Option `wa.order = "topleft"` verwenden:

```
weighted average: Daten sortieren Tiefe + Spaltensumme
library(palaeo) # Paket laden[12]
par(las=1) # Labelassignment, s. auf Seite 27
plot.depth(aber,
  yaxis.first=FALSE,
  polygon=TRUE, # Polygon
  min.scaling=TRUE, # minimal-Skalierung 5 fach
  plot.type="n", # keine Punkte
   # xaxes.equal=FALSE,  # Gleichskalierung aus
  xlabel="%",
  ylabel="sample number",
  xaxis.ticks.minmax=TRUE, # nur min-max
  nx.minor.ticks=0, # keine Teilstriche
  ny.minor.ticks=0, # keine Teilstriche
  bty="n", # keine Box
  yaxis.ticks=FALSE,
  wa.order = "topleft" # Sortierung: Tiefe + Spaltensumme
)
title("'wa.order=\"topleft\"'\nSortierung: Tiefe + Spaltensumme")
```

[12] http://www.campus.ncl.ac.uk/staff/Stephen.Juggins/data/palaeo_1.0.zip



Optionen für plot.depth(...) (Anhang S. 175)

```
data,  # Daten als data.frame() oder matrix() vorzugsweise 1.Spalte mit Tiefe
yaxis.first = TRUE,  # enthält erste Spalte Tiefendaten?
yaxis.num = "n",  # Zahlen an/aus an = "s" aus = "n"
xaxis.num = "s",  # Zahlen an/aus an="s"aus="n"; für alle gültig oder separat durch c(...)
xaxes.equal=TRUE,  # Gleichskalierung TRUE/FALSE; für alle gültig oder separat durch c(...)
xaxis.ticks.minmax=FALSE,  # nur min-max an x-Achse?
cex.x.axis=par("cex.axis")*0.8, # Größe x-Achsenbeschriftung
cex.y.axis=par("cex.axis")*0.8, # Größe y-Achsenbeschriftung
yaxis.lab = FALSE,  # keine zusätzlichen labels an restliche y-Achsen
yaxis.ticks = TRUE,  # y-ticks zusätzlich ja
axis.top = c(FALSE, FALSE),  # x-Achse auch top? für c(axis = TRUE, labels = TRUE)
nx.minor.ticks = 5, # Anz. Intervalle f. x-Teilstriche wenn Paket Hmisc installiert
ny.minor.ticks = 5,  # Anz. Intervalle f. y-Teilstriche wenn Paket Hmisc installiert
bty = "L", # L, c, o ... Boxtyp: für alle gültig oder separat durch c(...)
   # Box ähnelt dem Großbuchstaben; allgemein 27
plot.type = "o", # Linien/Punkttyp: für alle gültig oder separat mit c(...),
   # möglich: o, b, c, n, h, p, l, s, S; allgemein auf Seite 52
   # o = Punkt + Linie durchgezogen
   # b = Punkt + Linie unterbrochen
   # c = Linie unterbrochen + ohne Punkte
   # h = Histogramm-artige Linien
   # p = nur Punkte
   # l = Linie durchgezogen
   # s oder S = Stufen
plot.before = NULL,  # was VOR dem eigentlichen Zeichnen ausgeführt werden soll
   # z.B.: grid() als expression() angeben, also 'expression(c(par(xpd=F),grid()))';
   # kann auch mit list(...) verschachtelt werden;
   # für alle gültig oder separat/verschachtelt durch list(...)
plot.after = NULL,  # was NACH dem eigentlichen Zeichnen ausgeführt werden soll
   # z.B.: zusätzliche Grafiken z.B.: points(), lines() als expression() angeben:
   # expression(lines(...)) - kann auch mit list(...) verschachtelt werden;
   # für alle gültig oder separat/verschachtelt durch list(...)
l.type = "solid",
   # Linientyp: für alle gültig oder separat/verschachtelt durch list(...); s.S.27
l.width = 1,  # Linienbreite: für alle gültig oder separat/verschachtelt durch list(...)
```

...Fortsetzung umseitig

```
lp.color = "black",  # Linien-/ Punktaußenfarbe: für alle gültig oder separat/verschachtelt durch
list(...)
# Punkte; Typen allgemein auf Seite 28
p.type = 21,  # Punkttyp Kreis weiß gefüllt: für alle gültig oder verschachtelt durch list(...)
p.bgcolor = "white",  # HGrund Punkte: für alle gültig oder separat durch c(...)
p.size = 1 ,  # Punktgröße: für alle gültig oder separat/verschachtelt durch list(...)
mar.outer = c(1,6,4,1),  # Rand außen c(unten , li , oben, re)
mar.top = 9,  # Rand oben
mar.bottom = 5,  # Rand unten
mar.right = 0,  # Rand rechts
txt.xadj=0.1,  # align Text x-Richtung 0...1 links ... rechts vom plot
txt.yadj=0.1,  # align Text y-Richtung in Skaleneinheiten + -> nach oben; - -> nach unten
colnames = TRUE, # Spaltenbeschriftung an/aus für alle gültig oder separat durch c(...)
rotation = 60, # Text Rotation: für alle gültig oder separat durch c(...)
subtitle = "",  # Untertitel: für alle gültig oder separat durch c(...)
xlabel = "",  # x-Achsenbeschriftung: für alle gültig oder separat durch c(...)
ylabel = "",  # y-Achsenbeschriftung: für 1. Achse
main = "", # Titel der einzelnen Plots: für alle gültig oder separat durch c(...)
polygon = FALSE,  # Polygonplot an/aus: für alle gültig oder separat durch c(...)
polygon.color = "gray", # Farbe Polygonplot: für alle gültig oder separat durch c(...)
show.na=TRUE,  # Zeige fehlende Werte als rote 'x' an
min.scale.level = 0.2,
   # 0...1 wenn Daten kleiner als 0.2( = 20%) vom maximalsten Wert,
   # dann wird mehr Platz für den Plot gemacht
min.scale.rel = 0.5,
   # 0...1 relativer Platz/Breite der Teilgrafik zur maximal möglichen Breite
   # 1 = maximale mögliche Breite
min.scaling = FALSE,  # bei TRUE nur separate Angabe mit c(FALSE, FALSE, TRUE, .. ) sinnvoll
color.minscale = "gray95",
   # Farbe minimal-skalierter Daten:
   # für alle gültig oder separat/verschachtelt durch list(...)
wa.order ="none" # Sortierung der Spalten nach weighted average
   # "bottomleft"oder "topright"
```

Funktion `line.labels.add(...)` um waagerechte/senkrechte Linien + Text mit der Maus in Grafiken einzuzeichnen:

Optionen für `line.labels.add(...)` (Anhang S. 183)

```
  wieoft=1,  # wieviele Linien
  color="darkred",  # Farbe Linie + Text; Liste mit c(...) möglich
  color.bg="white",  # Box-Hintergrund; Liste mit c(...) möglich
  l.type="solid",  # Linientyp; Liste mit c(...) möglich
  l.width=1,  # Linienbreite; Liste mit c(...) möglich
  orientation="h",  # h-horizontal, v-vertikal n-keine
  text=FALSE,  # zusätzlicher Text; Liste mit c(...) möglich
  border=FALSE, # Rahmen um Text; Liste mit c(...) möglich
  xpad=0.6,  # padding Abstand Boxrand-Text; Liste mit c(...) möglich
  ypad=0.6,  # padding Abstand Boxrand-Text; Liste mit c(...) möglich
  text.scale=1,  # Skalierung von Text; Liste mit c(...) möglich
```

☞ `line.labels.add(1, orientation="n")` zeichnet die Linie so, wie sie mit der Maus plaziert wird, also weder waagerecht noch senkrecht

3.2.13 Violinplot

Violinplots lassen sich mit `simple.violinplot(...)` aus dem Paket `Simple` zeichnen oder bei Version 2.0 auch mit dem Paket `vioplot` zeichnen.

```
library(Simple)  # Paket laden[13]
par(ask=T)  # vor Neuzeichnen nachfragen
examlpe(simple.violinplot)  # Beispiele anzeigen
detach(package:Simple)  # Paket wieder loswerden
```

Paket vioplot v2.0

```
library(vioplot)  # Paket laden
par(ask=T)  # vor Neuzeichnen nachfragen
examlpe(violplot)  # Beispiele anzeigen
detach(package:vioplot)  # Paket wieder loswerden
```

3.2.14 Funktionen plotten

Es lassen sich mit `curve(...)` beliebige Funktionen zeichnen.

```
curve(x^3-3*x, -2, 2)
```
☞ zeichnet $x^3 - 3 \cdot x$ von -2 bis 2

3.2.15 Artenarealkurven

Mit dem package `vegan` und der Funktion `specaccum(...)` lassen sich Arten - Areal Kurven darstellen. Eine Möglichkeit, um abzuschätzen, wieviel Proben man für eine bestimmte Anzahl von gesammelten Arten braucht. Ein Beispiel aus der Hilfe gekoppelt mit Boxplots:

Beispiel von example(specaccum)

```
library(vegan)  # package vegan einlesen
data(BCI)  # Datensatz einlesen - Baumzählungen
?BIC # Hilfe anzeigen
sp1 <- specaccum(BCI)  # Kurve berechnen
sp2 <- specaccum(BCI, "random")  # Zufallskurve aus Daten berechnen
sp2  # Berechnung anzeigen
summary(sp2)  # Zusammenfassung
plot(sp1, ci.type="poly", col="blue", lwd=2, ci.lty=0, ci.col="lightblue")
boxplot(sp2, col="yellow", add=TRUE, pch="+")
detach(package:vegan)  # Paket wieder entfernen
```
☞ `ci.type="poly"` Art des Konfidenzintervalls, `col="blue"` Farbe blau, `lwd=2` Liniendicke, `ci.lty=0` Linientyp Konfidenzintervall, `ci.col="lightblue"` Farbe Konfidenzintervall; `boxplot(...)`: `add=TRUE` Boxplot dazuzeichnen, `pch="+"` Punkttyp (S.28) auf + stellen

```
detach(package:vegan)  # Paket eventuell wieder entfernen
```

3.2.16 Balkendiagramme/Histogramme

`barplot(...)` – Allgemeine Funktion für Balkendiagramme ist `barplot(...)`. Histogramme (von z.B. Verteilungen) werden mit `hist(...)` dargestellt. In Paket `gplots` gibt es noch eine Funktion `barplot2(...)` mit mehr Möglichkeiten, z.B.: Konfidenzintervalle einzeichnen.

☞ `barplot(...)` zeichnet an sich Rechtecke in vielen Varianten. Muß mal ein `barplot(..., add=TRUE)` in negativer x- oder y-Skala hinzugefügt werden, kann man der Option `width` in `barplot(..., width=-1, add=TRUE)` auch negative Werte zuweisen. An sich beginnt `barblot(...)` intern immer am 0 - Punkt der entsprechenden Achse die Balken zu zeichnen.

[13] Installation: `install.packages("Simple",contriburl="http://www.math.csi.cuny.edu/Statistics/R/simpleR/")`

Balkendiagramme - barplot(...)

```
# Beispiel Farbe
(tab.zahlen <- table(zahlen <- rpois(100, lambda=5)))
farbe <- rainbow(20) # 20 Regenbogenfarben
(bplot <- barplot(tab.zahlen, col=farbe)) # bplot enthält Mittelpunkte der Balken
# Linien einzeichnen
lines(bplot, tab.zahlen,  # x-y Koordinaten
  type='h',  # 'h'  histogramme-artig
  col=farbe[20:1],
  lwd=2  # Linienbreite
)
# Anzahlen als Text dazu
text(bplot, tab.zahlen+1, # x-y Koordinaten
  labels=tab.zahlen,  # Text
  xpd=TRUE  # darf auch außerhalb des Grafikbereichs zeichnen
)
```

```
# Keine Achsen + mehr Zwischenraum
barplot(tab.zahlen, space = 1.5, axisnames=FALSE,
  sub = "space = 1.5, \naxisnames = FALSE"
)
```

gruppierten Daten zeigen: übereinander

```
data(VADeaths)  # R-eigene Daten laden
?VADeaths  # direkte Hilfe ansehen
VADeaths  # Daten ansehen
barplot(VADeaths,
  col.sub="orange",
  main="Todesraten in Virginia\n(1940)",
  sub="Daten liegen als Matrix vor:\n 5 Spalten 5 Zeilen"
)
# Mittelwerte einzeichnen
bplot <- barplot(VADeaths,
  col.sub="blue",  # Farbe Untertitel
  sub="Mittelwerte gesamt" # Untertitel
)
mittel.ges <- colMeans(VADeaths)
text(bplot, mittel.ges + 3, format(tot),
  xpd = TRUE, # darf auch außerhalb der Zeichenfläche zeichnen
  col = "blue"
)
```

```
# Gruppierung nebeneinander
barplot(VADeaths,
  col.sub="red",
  beside=TRUE, # Gruppierung umschalten
  main="Todesraten in Virginia\n(1940)", # Titelei
  sub="dasselbe, nur gruppiert:\nbeside=TRUE" # Untertitel
)
```

...Fortsetzung umseitig

```
# mit Legende
barplot(VADeaths,
  beside = TRUE, # Gruppierung umschalten
  col = c(
    "lightblue", # Farbnamen
    "mistyrose",
    "lightcyan",
    "lavender",
    "cornsilk"
  ),
  legend = rownames(VADeaths),  # Legende
  ylim = c(0, 100)  # Wertebereich
)
title(main = "Todesraten in Virginia\n(1940)",
  font.main = 4,  # kursive Überschrift
  col.sub="red",  # Farbe Untertitel
  sub="mit Legende"   # Untertitel
)
```

Fehlerbalken

```
(alter.grp <- t(VADeaths)[, 5:1])  # Daten umgruppieren t(...) = transponieren
col.fehler <- "orange"   # Fehlerfarbe
bplot <- barplot(alter.grp,
  beside = TRUE, # Gruppierung umschalten
     # Farben:
  col = c("lightblue", "mistyrose", "lightcyan", "lavender"),
  legend = colnames(VADeaths),  # Legende
  ylim= c(0,100),  # Achsenskalierung
  main = "Todesraten in Virginia\n(1940)",  # Titel
  font.main = 4,  # Schrift Titel kursiv
  sub = expression(paste("Fehlerbalken  ",2%.%sigma) ) ,
  col.sub = col.fehler,  # Farbe
  cex.names = 1.5  # Skalierung Text
)
# 'Fehlerbalken'  als Linie dazu
segments( # Koordinaten x0, y0, x1, y1
  bplot, alter.grp, bplot, alter.grp + 2*sqrt(1000*alter.grp/100),
  col = col.fehler,  # Farbe
  lwd = 1.5  # Linienbreite
)
# mtext = margin-text also Text am Rand von Grafiken
mtext(side = 1, # 1=bottom, 2=left, 3=top, 4=right
  at = colMeans(bplot),  # wo - Position
  line = -1,  # Textzeilen-Einheiten von x-Achse weg
     # Text zusammenbasteln
  text = paste("Mittel:\n", formatC(colMeans(alter.grp))),
  col = "red"   # Farbe
)
```

Histogramme - hist(...)

```
hist(rnorm(1000, mean=103, sd=2), density=30, freq=F)  # zeichnet Densityfunktion
```

☞ `hist(...)` Optionen: `freq=F` zeichnet Densityfunktion bei `TRUE` die Häufigkeit, `density=30` gibt Schattierung an sowie `angle=30` den dazugehörigen Winkel, `nclass=10` 10 Klassen werden eingeteilt

Histogramm + Boxplot

```
 library(Simple)  # [14]
  x <- rnorm(100)             # normalverteilte Zufallszahlen
  simple.hist.and.boxplot(x)  # Histogramme + Boxplot
detach(package:Simple)        # Paket eventuell entfernen
```

[14] install.packages("Simple",contriburl="http://www.math.csi.cuny.edu/Statistics/R/simpleR/")

`histbackback(...)` – Ein Beispiel für Populationskurven Romain (2006)

```
library(Hmisc)  # nötiges Paket laden
age <- rnorm(1000,50,10)  # Zufallszahlen erzeugen
sex <- sample(c("männlich","weiblich"),
  1000,  # Größe 1000
  TRUE)  # Wiederholung: TRUE
out <- histbackback(split(age, sex), probability=TRUE,
  xlim=c(-.06,.06),  # x Wertebereich
  main = "Histogramm einer\n generierten Population")  # Titel
# Farbe dazu
barplot(-out$left, col="blue", horiz=TRUE, space=0, add=TRUE, axes=FALSE)
barplot(out$right, col="red", horiz=TRUE, space=0, add=TRUE, axes=FALSE)
detach(package:Hmisc)  # Paket wieder entfernen
```

3.2.17 Kreisdiagramme

`pie()` – Kreisdiagramme werden mit `pie()` erzeugt.

```
pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12) # "Dateneingabe"
  names(pie.sales) <- c("Blaubeere", "Kirsche",
   "Apfel", "Boston Eiskrem", "andere", "Vanille") # Namen festlegen
  pie(pie.sales) # Voreinstellung
  # % breechnen: Funktion auf Vektor anwenden
  perc <- sapply(X=pie.sales,FUN=function(x) x/sum(pie.sales)*100)
  names(pie.sales) <- paste(# als Namen zusammenfügen
    perc,"%: ",names(pie.sales),
    sep="" # ohne Leerzeichen
  )# end paste()
  pie(pie.sales) # geänderte Namen
  pie(pie.sales, # Farben explizit
    col = c("purple", "violetred1", "green3", "cornsilk", "cyan", "white")
  )
  pie(pie.sales, col = gray(seq(0.4,1.0,length=6))) # in grau
  pie(pie.sales, density = 5*1:6, angle = 15 + 20 * 1:6) # Schraffur
```

Eine interessante und informative Umfrage Grafik aus der LaTeX–Doku:

```
umfrage <- c(
  "keine"=20,
  "sehr gut"=3,
  "gut"=9,
  "OK"=10,
  "schlecht"=8,
  "sehr schlecht"=0
  )
  sum(umfrage) -> umfrage.sum # Summe
pie(rep(1,50),
  col= rep( # Farben nach Umfragedaten wiederholen
      c("white","green4","green","gray","red","darkred"),umfrage
  ), labels=NA
)
# riesen Punkt
  points(0,0, cex=35,col="black",pch=21,bg="white")
  text(0,0,"50 Teilnehmer mit\n abgegebenen Stimmen", cex=1.2)
# (locator(6) -> xy) # mit Maus 6 Punkte wählen oder Koordinaten festlegen:
  xy <- list(
    x= x <- c(0.08, -1.09, -1.15, -0.05, 1.04, 1.06),
```

```
    y= y <- c(1.08, 0.43, -0.41, -0.95, -0.46, 0.00)
  )
par(xpd=TRUE) -> paralt # darf auch außerhalb zeichnen
text(xy, paste(        # Text zusammenfügen
  names(umfrage),"\n",
  umfrage, " (",umfrage/umfrage.sum*100,"%)", sep="")
)
par(paralt)          # Grafikeinstellung wieder zurück
```

`floting.pie()` – Will man Kreisdiagramme in Grafiken zusätzlich an bestimmten x-y Positionen einzeichnen, dann `floting.pie()` aus Paket `plotrix` benutzen.

```
library(plotrix)  # Paket für div. Grafik-Tools laden
par(las=1,xpd=TRUE)-> paralt  # Grafikeinstellung zwischenspeichern
# las=Ausrichtung Achsenbeschriftung, xpd=außerhalb zeichnen
plot(1:5, 1:(-3),type="n",  # type=Grafiktyp
  main="floating.pie() - Test", # Titel
  xlab="", # x-Achsenbeschriftung
  ylab="Tiefe [m]",axes=FALSE)  # y-Achsenbeschriftung
grafikrand <- par("usr")
# par("usr") -> Grafikgrenzen: x1 x2 y1 y2
```

```
# 'Seeboden'  mit polygon() einzeichnen
x.boden <- c(grafikrand[1], grafikrand[2], grafikrand[2], grafikrand[1])
y.boden <- c(0,0,grafikrand[3],grafikrand[3])
polygon(x.boden, y.boden, border=NA, col="gray90" # Farbe
   # für Muster:
   # density=10, angle=-45, lwd=1
)
# 'See'  mit polygon() einzeichnen
# Tip: mit locator(8) -> xy kann man 8 Mauspositionen in 'xy'  speichern
x.see <- c(grafikrand[1], grafikrand[2], 4.6, 3.9, 2.7, 2.2, 1.6, 1.1)
y.see <-c(0.0, 0.0, -0.9, -1.7, -1.9, -1.3, -2.1, -1.1)
polygon(x.see, y.see, border="black", col="white")
axis(2)  # 2=y-Achse einzeichnen
                                    Kreisdiagramme floating.pie()
# farbe <- rainbow(5)  # Farben s.S.45
farbe <- gray.colors(5)  # Farben s.S.45
floating.pie(1.7,-0.3,  # x-y Position
  c(2,2,8,2,8), # Daten
  radius=0.3,
  col=farbe)  # 5 Ragenbogenfarben
floating.pie(3.1,-0.3,c(2,4,4,2,8),radius=0.3,  col=farbe)
floating.pie(1.7,-1.5,c(2,4,4,2,8),radius=0.3,  col=farbe)
floating.pie(3.1,-1.4,c(1,4,5,2,8),radius=0.3,  col=farbe)
floating.pie(4,-1, c(3,4,6,7,1),radius=0.3,  col=farbe)
                                                Legende
# par(xpd=TRUE) muß TRUE sein, sonst kein Zeichnen außerhalb der Grafik möglich
y.legende <- grafikrand[3]+0.5
x.legende <- grafikrand[2]-0.5
rect( # Viereck
  grafikrand[2]-1.2, grafikrand[3], # xy-links unten
  grafikrand[2], grafikrand[3]+1.4,  # xy-rechts oben
  col="white"   # Hintergrund weiß
)
# automatischer Maximalwert für Legende
(ind.max <- max(daten))
```

...Fortsetzung umseitig

```
# Text zusammenfügen mit paste()
text(
  x=grafikrand[2]-0.6,
  y=grafikrand[3]+1.2,
  paste("Anzahl Ind.\nMaximum:",ind.max))
floating.pie(x.legende, y.legende, c(3,4,6,7,1), radius=0.3, col=farbe) -> labelposition
pie.labels(x.legende, y.legende, labelposition,
  labels =paste("Art",1:5),  # 1:5= 1, 2, ... 5
  radius=0.35,  # Beschriftung
  bg=NA,  # Hintergrundfarbe
  border=FALSE)
par(paralt) # Grafikeinstellungen wieder zurück
detach(package:plotrix) # Paket eventuell wieder entfernen
```

Um die Werte einer Datenmatrix (Tabelle) als halb gefüllte Kreisdiagramme darzustellen, bietet sich `floating.pie()` als Funktion an:

```
# Test Matrix-Tabelle Export
library(plotrix) # Paket laden
daten <- matrix(0:24,5,5)
  (colnames(daten) <- paste("sample",sep="_",1:5))
  (rownames(daten) <- paste("Art",sep="_",1:5))
   # number of rows/columns speichern
  (nr <- nrow(daten))
  (nc <- ncol(daten))
plot(
  x=0:(nc+1), # x-Werte
  y=0:(nr+1), # y-Werte
  type="n",   # Plot-Typ
  xlab="",ylab="",  # keine Beschriftung
  axes=F)   # keine Achsen
par(xpd=T)  # außerhalb zeichnen TRUE
# Text
  ypos <- par()$usr[4]  # plot-Rand-Koordinaten
  xpos <- par()$usr[1]  # plot-Rand-Koordinaten
text( # Spalten
  1:nc,   # x-Werte
  rep(ypos,nc), # y-Werte
  colnames(daten),  # Text
  srt=90, # Drehung
  adj=0)  # Ausrichtung 0..1 links...rechts
text(
  rep(xpos,nr), # x-Werte
  1:nr, # y-Werte
  rownames(daten),  # Text
  srt=0, # Drehung
  adj=1)  # Ausrichtung 0..1 links...rechts
# Kreisdiagramm floating.pie(plotrix-pkg)
(dmax <- max(daten))  # Maximum ermitteln
# Spalten und Reihen durchlaufen
for(ri in 1:nr){
  for(ci in 1:nc){
```

...Fortsetzung umseitig

```
    floating.pie(ci,ri,
      c(
        ifelse(dmax-daten[ri,ci]==0,0.00001,dmax-daten[ri,ci]),
        ifelse(daten[ri,ci]==0,0.00001,daten[ri,ci])
      ),
      startpos=pi/2,
      col=c("white","black"),
      radius=0.3
    )
     # optionale Textausgabe
     # text(ci,ri-0.5,daten[ri,ci])
  }
}
detach(package:plotrix) # Paket wieder entfernen
```

`stars(...)` – Eine Alternative zu den Standard-Kreisdiagrammen stellt die Funktion `stars(...)` dar.

```
                                        Sterndiagramme - stars(...)
par(las=1,xpd=TRUE)-> paralt  # Grafikeinstellung zwischenspeichern
# las=Ausrichtung Achsenbeschriftung, xpd=außerhalb zeichnen
plot(1:5, 1:(-3),type="n",  # type=Grafiktyp n-nothing
  main="stars() - Test", # Titel
  xlab="", # x-Achsenbeschriftung
  ylab="Tiefe [m]",axes=FALSE)  # y-Achsenbeschriftung
grafikrand <- par("usr")
# par("usr") -> Grafikgrenzen: x1 x2 y1 y2
```

```
# 'Seeboden'  mit polygon() einzeichnen
x.boden <- c(grafikrand[1], grafikrand[2], grafikrand[2], grafikrand[1])
y.boden <- c(0,0,grafikrand[3],grafikrand[3])
polygon(x.boden, y.boden, border=NA, col="gray90" # Farbe
   # für Muster:
   # density=10, angle=-45, lwd=1
)
# 'See'  mit polygon() einzeichnen
# Tip: mit locator(8) -> xy kann man 8 Mauspositionen in 'xy'  speichern
x.see <- c(grafikrand[1], grafikrand[2], 4.6, 3.9, 2.7, 2.2, 1.6, 1.1)
y.see <-c(0.0, 0.0, -0.9, -1.7, -1.9, -1.3, -2.1, -1.1)
polygon(x.see, y.see, border="black", col="white")
# stars verlangt Daten-Matrix z.B.:
#          Art 1 Art 2 Art 3 Art 4 Art 5
# Probe 1     2     2     8     2     8
# Probe 2     2     4     4     2     8
# Probe 3     2     4     4     2     8
# Probe 4     1     4     5     2     8
# Probe 5     3     4     6     7     1
axis(2)  # 2=y-Achse einzeichnen
```

```
# Daten erstellen (einlesen s.S.13)
(daten <- data.frame(
  rbind( # Reihen zusammenfügen
   "Probe 1"= c(2,2,8,2,8), # Zeile 1
   "Probe 2"= c(2,4,4,2,8), # Zeile 2 u.s.w.
   "Probe 3"= c(2,4,4,2,8),
   "Probe 4"= c(1,4,5,2,8),
   "Probe 5"= c(3,4,6,7,1)
  )
))  # zusätzliche Klammer bewirkt Ausgabe
```

...Fortsetzung umseitig

```
# Spaltennamen erzeugen
(colnames(daten) <- paste("Art",1:5))
# Positionen der stars-Grafiken
wo <- rbind(
    c(1.7,-0.3), # Grafik 1
    c(3.1,-0.3), # Grafik 2 u.s.w.
    c(1.7,-1.5),
    c(3.1,-1.4),
    c(4,-1)
)
```

```
# Position Legende
y.legende <- grafikrand[3]+0.4
x.legende <- grafikrand[2]-0.6
rect(grafikrand[2]-1.2, grafikrand[3], grafikrand[2], grafikrand[3]+1.2,
  col="white"   # Hintergrund weiß, Farben s.S.45
)
(ind.max <- max(daten))  # automatischer Maximalwert für Legende
# Text zusammenfügen mit paste()
text(x=grafikrand[2]-0.6, y=grafikrand[3]+1, paste("Anzahl Ind.\nmax.",ind.max))
```

```
stars(daten, # Daten
  location=wo,  # x-y Position
  len=0.3,  # Größe Radius
  draw.segments = TRUE, # mit Segmenten
  col.segments=gray.colors(5),  # Segmente Farben, Farben s.S.45
  add=TRUE,  # dazu: ja
  labels=NULL, # keine labels à la 1, 2, 3, 4, ...
  key.loc = c(x.legende, y.legende)  # wo Legende?
)
par(paralt)  # Grafikeinstellung wieder zurück
```

`fan.plot(..)` – Eine weitere Möglichkeit Kreisdiagramme zu zeichnen, bietet `fan.plot(..)` aus dem Paket `plotrix`. Hier werden die Sektoren überlappend dargestellt.

```
# IUCN counts of threatened species by geographical area
library(plotrix)  # Paket laden
iucn.df <- data.frame(  # Daten erstellen
  area=c("Africa","Asia","Europe","N&C America",  "S America", "Oceania"),
  threatened = c(5994,7737,1987,4716,5097,2093)
)
fan.plot(iucn.df$threatened,
  labels = paste(iucn.df$area, iucn.df$threatened, sep = "-"),
  label.radius = c(1.2, 1.15, 1.2, 1.2, 1.2, 1.2),  # Abstand Labels
  main = "Bedrohte Arten nach geografischer Region",
  ticks = 276  # Anzahl Teilstriche
)
detach(package:plotrix)  # Paket wieder entfernen
```

3.2.18 Radial-/Uhrendiagramme

Solche Art von Diagrammen, bei dem die Zeiger- oder Strichlängen die Daten zeigen, lassen sich mit 3 verschiedenen Plots zeichnen aus dem Paket `plotrix` zeichnen:
`clock24.plot(...)` – Uhrendiagramm, `radial.plot(...)` – Radialdiagramm, `polar.plot(...)` – Polarplot.
Gib ein `library(plotrix)` und `example(clock24.plot)`, `example(radial.plot)`, `example(polar.plot)`.

3.2.19 3D - Diagramme

3D - Diagramme werden mit `persp()` erzeugt. Tippe auch `demo(persp)` ein.

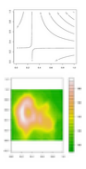
```
data(volcano)  # Beispieldatensatz laden
z <- 2 * volcano  # Relief vergrößern
x <- 10 * (1:nrow(z))  # 10 Meter Abstand (S nach N)
y <- 10 * (1:ncol(z))  # 10 Meter Abstand (O nach W)
par(bg = "slategray")  # Hintergrund einstellen
persp(x, y, z, theta = 135, phi = 30, col = "green3", scale = FALSE, ltheta = -120, shade = 0.75,
border = NA, box = FALSE)
?volcano # Hilfe anzeigen lassen
```

☞ `theta=135` Drehung des Objektes: ⚓, `phi =30` Drehung des Objektes: ☽; analog die `ltheta` und `lphi` Varianten: `ltheta = -120` steuert Lichteinfall: ⚓; `shade = 0.75` Schattendefinition; `border = NA` kein Gitter zeichnen; `box = FALSE` verhindert, daß der Graph umrahmt wird

Das Paket `rgl` bietet ebenfalls die Möglichkeiten 3D Grafiken zu zeichnen.

```
data(volcano)            # Topographie Vulkandaten
z <- 2 * volcano         # Relief aufbauschen
x <- 10 * (1:nrow(z))    # 10 meter spacing (S to N)
y <- 10 * (1:ncol(z))    # 10 meter spacing (E to W)
# Daten vorbereiten
z.lim <- range(y)        # min max
z.len <- z.lim[2] - z.lim[1] + 1
farben <- terrain.colors(z.len)  # height color lookup table
col <- farben[z - z.lim[1] + 1]  # assign colors to heights for each point
library(rgl)  # 3D-Grafik Paket laden
                                        Oberflächengrafik
open3d()  # 3D Grafik öffnen
  bg3d("slategray")  # Hintergrund
  material3d(col="black")  # Farbe
  surface3d(x, y, z, color=col, back="lines")  # Oberflächengrafik
  title3d('Titel: Vulkandaten','Untertitel','x-Achse','y-Achse','z-Achse')  # Titel
# rgl.snapshot("Vulkan.png",fmt="png") # abspeichern (muß im Vordergrund sein)
# rgl.postscript("Vulkan.pdf",fmt="pdf") # abspeichern (muß im Vordergrund sein)
                                          Kugeln
open3d()  # 3D Grafik öffnen
  spheres3d(rnorm(10), rnorm(10), rnorm(10),  # x y z
    radius=runif(10), color=rainbow(10))      # Radius + Farben
  title3d('spheres3d()','Untertitel','x-Achse','y-Achse','z-Achse')  # Titel
  axes3d(c('x','y','z')) # Achsen dazu
                                      Punkte 3D Plot
open3d()                 # 3D Grafik öffnen
  x <- sort(rnorm(1000)) # normalverteilte Zufallsdaten sortieren
  y <- rnorm(1000)       # normalverteilte Zufallsdaten
  z <- rnorm(1000) + atan2(x,y)  # arctang (?)
  plot3d(x, y, z,        # Daten xyz
   col=rainbow(1000),    # 1000 Regenbogenfarben
  size=3)                # Punktgröße
detach(package:rgl)      # Paket wieder entfernen
```

3.2.20 Konturenplot



Mit dem Paket `graphics` lassen sich Konturengrafiken erstellen

```
library(graphics)  # Paket laden
```

...Fortsetzung umseitig

```
x <- -6:16  # Zahlen von -6 bis 16
contour(outer(x, x), method = "edge", vfont = c("sans serif", "plain"))
```

☞ `method` bewirkt die Zahlenausrichtung, mögliche Werte: `"simple"`, `"edge"`, `"flattest"`, `vfont` Schriftparameter. Existieren extrem viele Punkte für die 3. Dimension z, dann macht der Linientyp `lty` sich scheinbar nicht bemerkbar, da so viele Punkte gezeichnet weden müssen.

```
filled.contour(...)
data(volcano)  # Daten laden
?volcano # Hilfe anzeigen
filled.contour(volcano, color = terrain.colors, asp = 1)
filled.contour(volcano, col = terrain.colors(10), asp = 1)  # reduzierter Farbumfang
detach(package:graphics)  # Paket wieder entfernen
```

☞ `color` Farbauswahl – möglich: `terrain.colors, rainbow, heat.colors, topo.colors, cm.colors` – beachte `col` und `color` sind verschieden , `asp = 1` x zu y zu Verhältnis

3.2.21 Karten zeichnen

Die Pakete `clim.pact` und `maps, mapproj` bieten die Möglichkeit Umrißkarten und Karten zu zeichnen. Mit der Funktion `map.grids(...)` lassen sich auch Gitternetzlinien einzeichnen. Siehe auch http://www.stat.cmu.edu/ minka/courses/36-315/handout/handout17.pdf

```
library(clim.pact)  # Paket laden
data(addland)  # Koordinatenfile laden
?addland # Hilfe zum Datensatz
plot(c(-90,90),c(0,80),type="n")  # leeren Rahmen zeichnen
addland()  # Konturen eintragen
grid()  # Raster darüber legen
detach(package:maps)  # Paket wieder entfernen
library(maps)  # Paket laden
example(map)  # Beispiele Karten
example(map.grids)  # Gitternetzlinien Beispiele
```

```
Hauptstädte einzeichnen
map("world", "China")
map.cities(country = "China", capitals = 2)
detach(package:maps)  # Paket wieder entfernen
```

3.2.22 Windrosen zeichnen

Das Paket `climatol` bietet die Möglichkeit Windrosendiagramme zu zeichnen.

```
library(climatol)  # Paket laden
data(windfreq.dat)
?windfreq.dat # Hilfe zum Datensatz
rosavent(windfreq.dat,4,4, ang=-3*pi/16, main="Annual windrose")
detach(package:climatol)  # Paket wieder entfernen
```

Eine weitere Möglichkeit ist `oz.windrose(..)` aus dem Paket `plotrix`.

```
library(plotrix) # Paket laden
   (windagg <- matrix(c(8,0,0,0,0,0,0,0,4,6,2,1,6,3,0,4,2,8,5,3,5,2,1,1,
   5,5,2,4,1,4,1,2,1,2,4,0,3,1,3,1), nrow=5, byrow=TRUE))
oz.windrose(windagg)
detach(package:plotrix)  # Paket wieder entfernen
```

3.2.23 Klimadiagramme zeichnen

Das Paket `climatol` bietet die Möglichkeit Klimadiagramme zu zeichnen.

```
library(climatol)  # Paket laden
data(cli.dat)
diagwl(cli.dat,est="Example station",alt=100,per="1961-90",mlab="en")
title("Klimadiagramm\nPaket 'climatol'") # Titel
?cli.dat # Hilfe zum Datensatz
detach(package:climatol)  # Paket wieder entfernen
```

3.2.24 Dreiecks-, Ternäre-, Triangeldiagramme

Dreiecksdiagramme (auch ternäre ∼) kann man mit `triangle.plot(...)` aus dem Paket `ade4` erstellen, mit `ternaryplot(...)` aus dem Paket `Zelig` oder `plot.acomp()`/`plot.rcomp()` aus dem Paket `compositions`.

triangle.plot(...)
ade4

`triangle.plot(...)`

```
library(ade4) # Paket laden
  data (euro123)  # Daten laden
  triangle.plot(euro123$in78,
  clab = 0,  # Labelgröße auf Null
  cpoi = 2,  # Punktgröße
  addmean = TRUE,  # Mittelpunkt einzeichnen
  show = FALSE)  # zeige verwendeten Bereich, da Skalierungen ja von 0 bis 1
detach(package:ade4) # Paket eventuell wieder entfernen
```

☞ Optionen `triangle.plot(...)`: zuerst muß ein dreispaltige Matrix angegeben werden (wird in Spaltenprozent transformiert)

`ternaryplot(...)`

```
library(Zelig) # Paket laden
a <- rnorm(12, mean=12)  # 12 Normalverteilungs-Daten
b <- rnorm(12, mean=20)
c <- 1:12  # 1, 2, ... 12
ternaryplot(cbind(a,b,c))  # cbind(...) column bind
detach(package:Zelig) # Paket eventuell wieder entfernen
```

☞ Optionen: 1. Argument 3-spaltige Matrix; `scale=1` Achsen Saklierung auf 1 (hat nur optischen Einfluß), `id='p'` kleines `p` wird unter jeden Punkt gezeichnet; `id.color` entsprechende Farbe; `coordinates=FALSE` falls `TRUE` Koordinaten unter jedem Punkt gezeigt; `grid=TRUE` ist Voreinstellung, auch möglich: Argumente von Linientyp `lty` (s.S. 27);`grid.color` entsprechende Farbe; `labels='inside'` Voreinstellung, auch `outside`, `none`, `labels.color` Farbe; `border=TRUE` Voreinstellung, auch `FALSE` möglich – dann kein △ oder Farbangabe (s.S. 45); `bg` Farbhintergrund; `pch` Punktypen (S. 28), `cex=1` numerische Vergrößerung/Verkleinerung; `prop.size` falls `TRUE` dann entspricht Symbolgröße der Reihensumme, d.h. repräsentiert die Wichtung der Beobachtungen, `col` Punkfarbe; `main='Titelei'`; zusätzlich noch Argumente zu `par(...)` s.S. 25

`plot() aus Paket 'compositions'`

```
library(compositions)  # Paket laden
example(plot.acomp) # Beispiele ansehen
detach(package:Zelig) # Paket eventuell wieder entfernen
```

Speziell für die Bodenanalyse gibt es `soil.texture(..)` aus dem Paket `plotrix`

```
library(plotrix)  # Paket laden
data(soils)  # internen Datensatz laden
soil.texture(soils[1:10,],main="Voreinstellung", label.points=T)
detach(package:plotrix) # Paket eventuell wieder entfernen
rm(soils)  # Datensatz aus dem Speicher entfernen
```

3.2.25 Interaktive Plots

identify(...)

Mit `identify(...)` läßt sich direkt in die Grafik Plotten und dabei lassen sich auch die Koordinaten von

Punkten bei sehr geringem Abstand der Punkte zuordnen.

locator(...)

`locator(...)` zeichnet an jede beliebige Stelle in der Grafik gewünschte Objekte.

```
                    itentify(...) - einfach mal kopieren und ausprobieren.
data(eurodist)  # Distanzmatrix laden
?eurodist # Hilfe zum Datensatz
plot(cmdscale(eurodist))  # MMDS darstellen
identify(cmdscale(eurodist)[,1],cmdscale(eurodist)[,2], labels=rownames(cmdscale(eurodist)))  #
Punkte benennen
                              locator(...) - Objekte platzieren
plot(runif(100), rnorm(100))  # Zufallsdaten erzeugen
legend(locator(1), "Punkte", pch=1)  # Legende mit locator(...)
```

☞ Optionen `locator(...)`: `locator(n=5, ...)` Anzahl zu setzender Punkte/Objekte (max=512), `locator(..., type="p")` p-Punkte werden gezeichnet, `l`-Punkte werden zu Linien verbunden; es können auch grafische Parameter als letztes Argument übergeben werden. (s. `par(...)` auf Seite 26)

3.2.26 Plots für GRASS 5.0 - geographical information system

Mit `plot.grassmeta(...)` aus dem Paket `GRASS` lassen sich geografische Informationen plotten, wie z.B. Dichten, Trends,...

plot.grassmeta()
GRASS

```
example(plot.grassmeta)  # Interpolierte Rasterdaten
example(trmat.G)  # Trend-Plots
```

3.3 Korrelationen visualisieren

Mit `plotcorr(...)` aus dem Paket `ellipse` lassen sich ganze Korrelations-Matrizen plotten.

plotcorr()
ellipse

```
library(ellipse) # Paket laden
example(plotcorr)# Beispiel aufrufen
```

Korrelationen als übersichtliche Textausgabe lassen sich mit der Funktion `symnum()` berechnen.

```
set.seed(25)    # Zufallsgenerator fixieren
(corrmatrix <- cor(matrix(rnorm(100),  10, 10)))
corrmatrix[c(3,6), 2] <- NA  # Na Werte erzeugen
(rownames(corrmatrix)  <- letters[1:10])  # Zeilennamen zuweisen
symnum(corrmatrix)   # Korrelationsmatrix
 a 1
 b   1
 c . ? 1
 d ,     1
 e .       1
 f . ?     . 1
 g . . .     . 1
 h         , .   1
 i             . , 1
 j             .   . 1
 attr(," legend" )
 [1] 0 ‘ ’ 0.3 ‘.’ 0.6 ‘,’ 0.8 ‘+’ 0.9 ‘*’ 0.95 ‘B’ 1 \t    ## NA: ‘?’
```

4 Statistik

4.1 Prinzipien des Testens

Allgemeines Vorgehen bei Tests
(Quelle www.uni-konstanz.de/FuF/Verwiss/ Schnell/DAnalyse/14Signifikanztests.pdf)

1. Inhaltliches Problem quantifizieren
2. Modellannahmen formulieren
3. Darstellung des inhaltlichen Problems als Test des Modellparameters
4. Festlegung des Signifikanzniveaus
5. Konstruktion des Ablehnungsbereichs (s.Fehler 1. und 2. Art) der Nullhypothese H_0
   a) Prüfgröße festlegen
   b) Verteilung der Prüfgröße unter Annahme der Nullhypothese H_0
   c) Bestimmung des Ablehnungsbereichs
6. Berechnung der Prüfgröße
7. Vergleich der Prüfgröße mit dem Ablehnungsbereich

4.1.1 Modellbeschreibungen

Tabelle 2: Modellformeln wie sie in R immer wieder an verschiedener Stelle gebraucht werden. Quelle: http://www-m1.ma.tum.de/nbu/statprakt/Kap4/Statistik_in_R.shtml

`y~x, y~1+x`	Beide Formeln beschreiben das gleiche Modell der einfachen linearen Regression von y auf x, wobei die erste Formel einen impliziten Intercept enthält und die zweite einen expliziten.
`y~0+x, y~-1+x, y~x-1`	Einfache lineare Regression von y auf x durch den Ursprung, also ohne Intercept
`y~A/(1+x)-1, y~A/x-1`	getrennte einfache lineare Regressionsmodelle innerhalb der Klassifikationen von A.
`log(y)~x1+x2`	Multiple Regression der transformierten Variable $log(y)$ auf $x1$ und $x2$ mit einem impliziten Intercept
`y~poly(x,2),` `y~1+x+I(x^2)`	Polynomiale Regression vom Grade 2. Die erste Formel benutzt orthogonale Polynome und die zweite benutzt explizite Potenzen (Modell mit explizit quadratischen Potenzen) als Basis-Grundlage.
`y~poly(x,3)`	Polynomiale Regression vom Grade 3, also kubische Regression mit orthogonalen Polynomen
`y~X+poly(x,2)`	Multiple Regression mit einer Designmatrix die aus der Matrix X, sowie aus orthogonalen Termen in x vom Grade 2
`y~A`	ANOVA-Modell mit Klassen, die durch den Faktor A bestimmt sind
`y~A+x`	ANOVA-Modell mit Kovariable x
`y~A*B, y~A+B+A:B,` `y~B%in%A, y~A/B`	2-Faktor nichtadditives Modell von y auf A und B. Die ersten beiden Formeln bezeichnen die gleiche „gekreuzte“ Klassifikation und die anderen beiden die gleiche „nested“ Klassifikation. Abstrakt gesehen bezeichnen alle 4 Formeln das gleiche zugrundeliegenden Modell.
`y~(A+B+C)^2,` `y~A*B*C-A:B:C`	Beide Formeln bezeichnen das selbe Modell, ein Experiment mit 3 Faktoren und einem Modell mit Main Effects aber lediglich zwei Interaktionen
`y~.`	ein Modell mit allen Parametern, die es gibt wird berechnet
`y~. + Condition(A+B)`	bei CCA mit Paket `vegan`: Modell mit allen Variablen und Covariablen A und B herausgerechnet

4.2 Zahlen generieren - Zufallszahlen, Verteilungen

Um Zufallszahlen für jeden reproduzierbar zu machen, kann man mit `set.seeds(28)` z.B. den Zufallsgenerator zwingen bei 28 anzufangen.

Normalverteilung - rnorm(...)

```
plot(rnorm(2000),rnorm(2000, mean=3 , sd=0.5))  # 20000 Zufallszahlen   der Normalverteilung
```

☞ `mean` Mittelwertangabe, `sd` Standardabweichung

```
plot(runif(2000, min=0.2, max=0.8))  # 2000 Zufallszahlen gleichmäßig gestreut
```

☞ `min` und `max` müssen zwischen `0, 1` liegen.

```
hist(rnorm(1000, mean=103, sd=2), density=30, freq=F)  # Mittel=103 & Standardabweichung 2
lines(density(103,2), col="red") # Linie dazuzeichnen
```

Poissonverteilung - rpois(...)

```
hist(rpois(1000,103), density=30, freq=F, nclass=10)
```

☞ `rpois(1000,103)` 1000 Wiederholungen mit $\lambda = 103$ (Anm.: λ ist der Lageparameter der Poissonverteilung, wie das arithmetisches Mittel für die Normalverteilung), theoretisch kann man auch sagen dies ist die Wahrscheinlichkeitsverteilung, um bei 1000 Versuchen aus einer Population genau 103 Tiere zu zählen

Binomialverteilung - rbinom(...)

```
hist(rbinom(1000,103,0.99), density=30, freq=F)
```

☞ die Wahrscheinlichkeitsverteilung den Wert 103 mit einer Wahrscheinlichkeit von $p = 0.99$ zu ziehen oder nicht zu ziehen, wird mit 1000 Zahlen simuliert.

4.3 Regressionsanalyse

Sehr ausführlich unter http://www-m1.ma.tum.de/nbu/modreg/. Ein Hinweis: es gibt im Anhang auf Seite 194 eine Benutzerfunktion `modelEquation(lm-modell, ndigits, format)`, um lineare Modellgleichungen in Grafiken darzustellen. Für die generelle grafische Darstellung von Regressionen gibt es die Funktionen `termplot`.

allgemeine Modell Ausgabe

```
 summary(model)
all:
m(formula = Volume ~ Height * Girth, data = trees)

Residuals:
   Min     1Q Median     3Q    Max
6.582 -1.067 0.303   1.564  4.665

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
Intercept)   69.3963    23.8358    2.91 0.00713
Height        -1.2971     0.3098   -4.19 0.00027
Girth         -5.8558     1.9213   -3.05 0.00511
Height:Girth   0.1347     0.0244    5.52 7.5e-06

Residual standard error: 2.71 on 27 degrees of freedom
Multiple R-Squared: 0.976,Adjusted R-squared: 0.973
R-statistic: 359 on 3 and 27 DF, p-value: <2e-16
```

☞ Modellformel;

☞ Ergebnis Residuen; per Definition: das arithmetische Mittel ist 0;

☞ Modell-Koeffizienten (1.Spalte), Standard Fehler der Koeffizienten (2.Spalte), Signifikanzniveaus (3.Spalte): dies ist die Wahrscheinlichkeit, daß die Nullhypothese H_0 stimmt;

☞ Residuen Standard Fehler, der definiert ist als die Quadratwurzel der geschätzten Varianz des Zufallfehlers: $\sigma^2 = \frac{1}{n-p} \cdot \sum(r_i^2)$, wobei r_i eine der n Residuen ist und p die Anzahl der Koeffizienten;

...Fortsetzung umseitig

☞ Bestimmtheitsmaß[15] R^2: zum einen der unangepaßte Anteil der erklärenden Varianz durch das Modell: $R^2 = 1 - \frac{ResiduenQuadratsumme}{Total-Quadratsumme}$ zum anderen der für die Anzahl der Parameter des Modells angepaßte Koeffizient $R^2 = 1 - \left[\frac{n-1}{n-p} \cdot (1 - R^2)\right]$

4.3.1 Liniear, einfach

$y = \beta_0 + \beta_1 \cdot x + \epsilon$

Ein Modell der Form $y = \beta_0 + \beta_1 \cdot x\epsilon$

```
# Beispieldatensatz
x = c(18,23,25,35,65,54,34,56,72,19,23,42,18,39,37)
y = c(202,186,187,180,156,169,174,172,153,199,193,174,198,183,178)
plot(x,y)  # Plot zeichnen
abline(lm(y ~x))  # Regressionsgerade
yx <- lm(y ~x)  # das Modell
summary(yx)  # Signifikanztest, Summary
```

☞ das Modell: $y = 210.04846(\pm 2.8669) - 0.7977(\pm 0.0699)x$ mit $R^2 = 0.9091$ (Bestimmtheitsmaß) , das Modell erklärt also 90,91% der Varianz, F-statistic s.F - Verteilung ; ohne Intercept wäre in diesem Fall nicht sinnvoll, geht aber generell mit der Angabe `lm(y~x-1)`

```
                                    Residuenanalyse
par(mfrow=c(2,2))  # Grafik auf 2x2 stellen
plot(yx)  # diagnostische Plots, s. Regressionsanalyse
                                Vorhersagen - Konfidenzintervalle
library(Simple)  # Paket Simple[16]
x <- 1:10  # Daten generieren
y <- 5*x + rnorm(10,0,1)  # Daten generieren
tmp <- simple.lm(x,y)
summary(tmp)  # anzeigen
simple.lm(x,y, show.ci=T)  # Konfidenzintervalle bei α = 0.95
simple.lm(x,y,pred=c(5,6,7)) # Vorhersagen für 5, 6, 7
detach(package:Simple)  # Paket wieder entfernen
```

...Fortsetzung umseitig

[15] aufpassen bei Vergleich mit und ohne Achsabschnitt (Intercept) – aus der R-help Diskussionsliste: Betreff „R vs. Excel (R-squared)“ http://tolstoy.newcastle.edu.au/R/help/06/01/19742.html
> Hello All-
>
> I found an inconsistency between the R-squared reported in Excel vs.
> that in R, and I am wondering which (if any) may be correct and if
> this is a known issue. While it certainly wouldn't surprise me if
> Excel is just flat out wrong, I just want to make sure since the R-
> squared reported in R seems surprisingly high. Please let me know if
> this is the wrong list. Thanks!
Excel is flat out wrong. As the name implies, R-squared values cannot be less than zero (adjusted R-squared can, but I wouldn't think that is what Excel does).
R-squared is a bit odd in the zero intercept case because it describes how much better the line describes data compared to a horizontal line *at zero*. However, it doesn't really makes sense to compare with a non-zero constant, because the models are not nested.
R-help Diskussionsliste: Betreff „R vs. Excel (R-squared)“ http://tolstoy.newcastle.edu.au/R/help/06/01/19793.html
Hi

In model without intercept Rsqared is high.

See e.g. Julian J. Faraway - Practical regression ....

Warning: R^2 as defined here doesn't make any sense if you do not have an intercept in your model. This is because the denominator in the definition of R2 has a null model with an intercept in mind when the sum of squares is calculated. Alternative definitions of R^2 are possible when there is no intercept but the same graphical intuition is not available and the R^2's obtained should not be compared to those for models with an intercept. ***Beware of high R^2's reported from models without an intercept***.

[16] Installation: `install.packages("Simple",contriburl="http://www.math.csi.cuny.edu/Statistics/R/simpleR/")`

eigene Funktion f. Teststatistik und Konfidenzintervalle

```
lm.reg.plot <- function(y,x, alpha=0.05)
{
   modell <- lm(y~x)
   summary.mod <- summary(modell)
   layout(matrix(c(1, 2, 3, 4, 5, 5), 2, 3), widths=c(1,1,2))
   plot(modell)
   newData <- data.frame(x = seq(min(x), max(x),
   by = (max(x) - min(x)) / 49))
   pred.lim <- predict(modell, newdata = newData,
   interval = "prediction", level = 1-alpha)
   conf.lim <- predict(modell, newdata = newData,
   interval = "confidence", level = 1-alpha)
   plot(x,y, ylab="y", xlab="x")
   matplot(newData$x,cbind(conf.lim, pred.lim[,-1]),
   lty=c("10","22","22","13","13"),
   col=c("black","black","black","black","black"),
   type="l", add=T)
   print(summary.mod)
   par(mfrow=c(1,1))
}
```

☞ Die Vorhersage - Bänder sind weiter von der Gerade entfernt, als die Konfidenz - Bänder. Das 95% Vorhersageintervall ist die Fläche, in die 95% aller Datenpunkte fallen. Das Konfidenzintervall ist die Fläche in der die Regressionsgerade mit 95% Wahrscheinlichkeit liegt. (Die Voreinstellung dieser Funktion ist $\alpha = 0.05$.)

Konfidenz/Vorhersageintervalle (1) mit predict() + matplot() (linear/nichtlinear)

```
# Daten
x <- rnorm(30)  # normalverteilte Zufallsdaten
y <- x + rnorm(30)  # normalverteilte Zufallsdaten
modell <- lm(y ~  x) # Modell y= a*x + b
predict(modell)  # Ausgabe der berechneten Zahlen des Modells
# neue Daten
x.neu <- data.frame(x = seq(-3, 3, 0.5))
predict(modell, x.neu, se.fit = TRUE)  # plus se.fit = standard errors
# Vertrauensbereiche
modell.pred <- predict(modell, x.neu, interval="prediction")
modell.conf <- predict(modell, x.neu, interval="confidence")
# matplot() zeichnet eine Matrix gegen Daten einer anderen
matplot(x.neu$x, cbind(modell.conf, modell.pred[,-1]),
  lty=c(1,2,2,3,3),
  col=c("black", "red", "red",  "green", "green"),
  type="l",
  ylab="y-Werte",
  xlab="x-Werte",
  main="linare Regression + rug()"
)
points(x,y) # Original Punkte
legend("bottomright",
  legend=c("Modell Gerade", "Vertrauen (Gerade mit 95% hier)", "Vorhersage (95% aller Werte)") ,
  lty=c("solid","dashed", "dotted"),
  col=c("black", "red", "green"),
)
# eventuell "Striche-Teppich"  an den Rand rug()
rug(y,side=2) # ; rug(x)
```

...Fortsetzung umseitig

```
# Zusammenfassung/Modellparameter ausgeben
(summary(modell) -> modell.sum)
# modell.sum$r.squared R²
# modell.sum$adj.r.squared R²adj
(round(coefficients(modell)[1],2) -> b)  # Intercept
(round(coefficients(modell)[2],2) -> a)  # Anstieg
(locator(1)  -> wo.xy)  # Position mit der Maus
# Gleichung y = ax +/- b
text(wo.xy, paste("y = ", a, "x", if(b<0) "+"  else "-", b))
# R²adj
text(wo.xy, expression(R[paste("adj")]^2), adj=c(1,1.5))
# R²adj
text(wo.xy, label=paste("=",round(modell.sum$adj.r.squared,2)), adj=c(-0.3,2.5))
          Konfidenz/Vorhersageintervalle (2) mit predict() + curve() (linear/nichtlinear)
daten <- data.frame(  #  irgendwelche Daten
  x = x <- c(1.9, 0.8, 1.1, 0.1, -0.1, 4.4, 4.6, 1.6, 5.5, 3.4),
  y = y <- c(0.7, -1, -0.2, -1.2, -0.1, 3.4, 0, 0.8, 3.7, 2)
)
#  alternativ aus Zwischenablage mit Tabstop:
#  daten <- read.csv2("clipboard",header=TRUE,sep="")
attach(daten)  #  in den Suchpfad
modell <- lm(y ~x )  #  lin. Modell
plot(x, y, bty="l")  #  Grafik
curve( #  normale Datenlinie
  predict(modell,
    newdata = data.frame(x = x)),
  add = TRUE)
curve( #  Linie Konfidenzintervall
  predict(modell,
    newdata = data.frame(x = x),
    interval = "confidence")[ , "lwr"],  #  lower Intervall
  add = TRUE,
  lty="dashed") # Linientyp s. 27
curve( #  Linie Konfidenzintervall
  predict(modell,
    newdata = data.frame(x = x),
    interval = "confidence")[ , "upr"],
  add = TRUE,
  lty="dashed") # Linientyp s. 27
```

☞ Vorhersageintervalle analog mit `interval = "prediction"`; `predict()` und `curve()` funktionieren auch mit anderen (nichtlinearen) Modellgleichungen

4.3.2 Linear, multipel

$y=\beta_0+\beta_1\cdot x_1+\beta_2\cdot x_2+...+\epsilon$

Ein Modell der Form $y = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + ... + \epsilon$

```
data(airquality)  # Daten einlesen
?airquality # Hilfe zum Datensatz
names(airquality)  # welche Variablen?
attach(airquality)  # wenn attach(...) dann entfällt 'data=airquality'i.d.R.
ozone.lm <- lm(Ozone~Solar.R + Wind + Temp + Month + Day )
ozone.lm; summary(ozone.lm)  # Formel + Statistik
ozone.lm <- lm(Ozone~., data=airquality)  # oder einfacher: 'Ozone~.'
```

☞ sobald man die verkürzte Schreibweise `Ozone~.` benutzt, muß man jedoch die Datenherkunft hinzufügen; hier durch `data=airquality`

```
ozone.lm; summary(ozone.lm)  # Formel + Statistik
plot(ozone.lm)  # diagnostische Plots, s. Regressionsanalyse
```

...Fortsetzung umseitig

```
bestes Modell automatisch - step(...)
```
☞ es ist nicht trivial, wie die Gleichungen angegeben werden!
```
airquality.noNA <- na.omit(airquality)  # Daten enthalten NA's
ozone.lm <- lm(Ozone~Temp, data=airquality.noNA)
step(ozone.lm,~Solar.R + Wind + Month + Day + Temp, data=airquality.noNA)
```
☞ nur 1 Variable wird ausgewählt
```
step(lm(Ozone~., data=airquality.noNA))  # bewirkt dasselbe
```
☞ mehrere Variablen werden ausgewählt
```
Variablen hinzufügen/entfernen - add1(...)/drop1(...)
ozone.lm <- lm(Ozone~Wind)  # lm nur mit Wind & Ozon
add1(ozone.lm, ~Temp + . ,data=airquality)  # Temp wird hinzugefügt, AIC verbessert sich
drop1(lm(Ozone~. ,data=airquality))  # Modell mit allen Variablen: je eine Variable wird entfernt
```
☞ <none> -> zeigt volles Modell, die restlichen Zeilen zeigen die AIC Werte für das Modell ohne den entsprechenden Faktor

4.3.3 Polynomial, multipel

Ein Modell der Form $y = \beta_0 + \beta_1 \cdot x_1^n + \beta_2 \cdot x_2 + ... + \epsilon$

$y=$
$\beta_0+\beta_1\cdot x_1^n+$
$\beta_2\cdot x_2+...+\epsilon$

```
data(airquality)  # Daten einlesen
?airquality # Hilfe zum Datensatz
names(airquality)  # welche Variablen?
attach(airquality)  # wenn attach(...) dann entfällt 'data=airquality'i.d.R.
```
☞ sobald man die verkürzte Schreibweise `Ozone~.` benutzt, muß man jedoch die Datenherkunft hinzufügen; hier durch `data=airquality`
```
pairs(airquality, panel = panel.smooth, main = "airquality data") # Daten anschauen
ozone.lm.wind <- lm(Ozone~., data=airquality)  # normales Modell
ozone.lm.wind2 <- lm(Ozone~+ poly(Wind,2) + ., data=airquality) # Wind -> polynom 2.Grades
step(lm(Ozone~. , data=airquality)) # AIC Auswahl
step(lm(Ozone~+ poly(Wind,2) + . , data=airquality)) # AIC Auswahl
summary(ozone.lm.wind) # R^2 = 0.6249
model.matrix(ozone.lm.wind) # Modellmatrix ansehen
summary(ozone.lm.wind2, cor=T) # R^2 = 0.7073 verbessert sich + Korrelationsmatrix
model.matrix(ozone.lm.wind2)  # Modellmatrix ansehen
op <- par(mfrow=c(2,2))  # Grafik 2x2
plot(ozone.lm.wind2)  # diagnostische Plots, s. Regressionsanalyse
par(op) # 2x2 zurücksetzten
```

4.3.4 one-way-ANOVA

Nur ein Faktor hat Einfluß auf das Modell. (Siehe auch „ Practical Regression and Anova using R“ http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf und allgemein http://www.zoologie.sbg.ac.at/LVAMinnich/varistat.htm)

```
data(InsectSprays)  # Daten laden
?InsectSprays # Hilfe zum Datensatz
dimnames(InsectSprays)  # Namen anschauen
attach(InsectSprays)  # Suchpfadaufnahme
plot(InsectSprays)  # einfacher plot
plot(count~spray)  # Boxplot
plot.design(InsectSprays)  # Design der Daten
```
mit Intercept
```
ins.lm <- lm(count~spray)
par(mfrow=c(2,2));plot(ins.lm);par(mfrow=c(1,1))  # diagnostische Plots, s. Regressionsanalyse
model.matrix(ins.lm)  # Modell als Matrix anschauen
summary(ins.lm)  # Intercept = Gruppe A
```
...Fortsetzung umseitig

```
                                        ohne Intercept '-1'
ins.lmi <- lm(count~spray -1)
par(mfrow=c(2,2));plot(ins.lmi);par(mfrow=c(1,1))  # diagnostische Plots, s. Regressionsanalyse
model.matrix(ins.lmi)  # Modell als Matrix anschauen
summary(ins.lmi)  # Modelle für jede Gruppe
                          übereinandergezeichnete Punkte verrauschen jitter(...)
par(mfrow=c(1,2))  # Grafik 1x2
plot(jitter(ins.lm$fit),ins.lm$res,xlab="Fitted",ylab="Residuals",main="Jittered plot")
plot(ins.lm$fit,ins.lm$res,xlab="Fitted",ylab="Residuals",main="Not Jittered plot")
par(mfrow=c(1,1))  # Grafik 1x1
```

☞ Antwort die Gruppen sind bis auf Gruppe C signifikant, aber wer von wem ist nicht geklärt

```
                          Tukey's (HSD) - paarweise Vergleiche, s. post-hoc Tests
lm.turkey <- TukeyHSD(aov(ins.lm))
op <- par(mfrow=c(1,2))  # Grafik 1x2
plot(lm.turkey, las=1)  # Mittelwertvergleiche
boxplot(count~spray, notch=T)  # Boxplot mit notch
par(op)  # Grafik wieder 1x1
```

☞ A und B sind nicht voneinander verschieden, aber A und C

```
                                        Varianzhomogenität
summary(lm( abs(ins.lm$res)  InsectSprays$spray))
```

☞ Aufpassen es besteht keine Varianzhomogenität! Da der Test ja immer von der Nullhypothese H_0 her testet, die ja behauptet es gibt keine Unterschiede zwischen den zu testenden Variablen. Da der Test aber einen signifikanten Unterschied zeigte müssen wir schlußfolgern: keine Varianzhomogenität. Es müßte also ein nichtparametrischer Test durchgeführt werden. s. Verteilungsanpassungstest und post-hoc Tests.

4.3.5 Post Hoc Tests

Insbesondere für die Post Hoc Tests nach Scheffé und Bonferroni, stehen nur die selbstgeschriebenen Funktionen zur Verfügung (s. Beispiele am Ende der Tabelle)
http://www.agr.kuleuven.ac.be/vakken/statisticsbyR/ANOVAbyRr/multiplecompJIMRC.htm

```
all.pairs <- function(r)  # von nachfolgenden Funktionen benötigt
list(first = rep(1:r,rep(r,r))[lower.tri(diag(r))],
second = rep(1:r, r)[lower.tri(diag(r))])
                                        Test nach Tukey - tukeyCI(...)
tukeyCI <- function(fitted, nis, df, MSE, conf.level=.95)
{
   # fitted is a sequence of means
   # nis is a corresponding sequence of sample sizes for each mean
   # df is the residual df from the ANOVA table
   # MSE = mean squared error from the ANOVA table
   # conf.level is the family-wise confidence level, defaults to .95
  r <- length(fitted)
  pairs <- all.pairs(r)
  diffs <- fitted[pairs$first] - fitted[pairs$second]
  df <- sum(nis) - r
  T <- qtukey(conf.level, r, df)/sqrt(2)
  hwidths <- T*sqrt(MSE*(1/nis[pairs$first] + 1/nis[pairs$second]))
  val <- cbind(diffs - hwidths, diffs, diffs + hwidths)
  dimnames(val) <- list(paste("mu",pairs$first,"- mu", pairs$second,
  sep=""), c("Lower", "Diff","Upper"))
  val
}
```

☞ s. auch `TukeyHSD(...)` aus `library(stats)`

...Fortsetzung umseitig

Test nach Scheffé - scheffeCI(...)

```
scheffeCI <- function(fitted, nis, df, MSE, conf.level=.95)
{
   # fitted is a sequence of means
   # nis is a corresponding sequence of sample sizes for each mean
   # df is the residual df from the ANOVA table
   # MSE = mean squared error from the ANOVA table
   # conf.level is the family-wise confidence level, defaults to .95
  r <- length(fitted)
  pairs <- all.pairs(r)
  diffs <- fitted[pairs$first] - fitted[pairs$second]
  T <- sqrt((r-1)*qf(conf.level,r-1,df))
  hwidths <- T*sqrt(MSE*(1/nis[pairs$first] + 1/nis[pairs$second]))
  val <- cbind(diffs - hwidths, diffs, diffs + hwidths)
  dimnames(val) <- list(paste("mu",pairs$first,"- mu", pairs$second,
  sep=""), c("Lower", "Diff","Upper"))
  val
}
```

Test nach Bonferroni - bonferroniCI(...)

```
bonferroniCI <- function(fitted, nis, df, MSE, conf.level=.95)
{
   # fitted is a sequence of means
   # nis is a corresponding sequence of sample sizes for each mean
   # df is the residual df from the ANOVA table
   # MSE = mean squared error from the ANOVA table
   # conf.level is the family-wise confidence level, defaults to .95
  r <- length(fitted)
  pairs <- all.pairs(r)
  diffs <- fitted[pairs$first] - fitted[pairs$second]
  T <- qt(1-(1-conf.level)/(2*r*(r-1)),df)
  hwidths <- T*sqrt(MSE*(1/nis[pairs$first] + 1/nis[pairs$second]))
  val <- cbind(diffs - hwidths, diffs, diffs + hwidths)
  dimnames(val) <- list(paste("mu",pairs$first,"- mu", pairs$second,
  sep=""), c("Lower", "Diff","Upper"))
  val
}
```

Beispiel - Insektenspray

☞ zuerst Argumente berechnen für `fitted`-(Mittelwert), `nis`-number of inner samplesize (Anz. Proben für jeden MW), `df` Residuen von ANOVA, `MSE` Fehlerquadratsumme der MW:

```
data(InsectSprays)  # Daten laden
?InsectSprays # Hilfe zum Datensatz
Ins.means <- tapply(InsectSprays$count, InsectSprays$spray, mean)
  Ins.means  # MW anschauen
Ins.len <- tapply(InsectSprays$count, InsectSprays$spray, length)
  Ins.len  # Anz. anschauen
Ins.aov <- aov(InsectSprays$count ~InsectSprays$spray)
dfMSE=Ins.aov$df.residual
  dfMSE  # Residuen von ANOVA
MSE=sum(Ins.aov$residuals^2)/dfMSE
  MSE  # Fehlerquadratsumme der MW
anova(Ins.aov)  # Überprüfung der letzten beiden Werte
tukeyCI(Ins.means, Ins.len, dfMSE, MSE, conf=.95)
# Vergleiche mit TukeyHSD(...) aus library(stats)
Ins.Tukey <- TukeyHSD(Ins.aov,"InsectSprays$spray")
  Ins.Tukey  # TukeyHSD anschauen
bonferroniCI(Ins.means, Ins.len, dfMSE, MSE, conf=.95)
scheffeCI(Ins.means, Ins.len, dfMSE, MSE, conf=.95)
```

4.3.6 GLM

Allgemeine Linaere Modelle (GLM) lassen sich mit `glm(...)` berechnen.

```
data(InsectSprays)  # Daten laden
?InsectSprays # Hilfe zum Datensatz
dimnames(InsectSprays)  # Namen anschauen
attach(InsectSprays)  # Suchpfadaufnahme
plot(InsectSprays)  # einfacher plot
plot(count~spray)  # Boxplot
plot.design(InsectSprays)  # Design der Daten
```

mit Intercept

```
ins.glm <- glm(count~spray, family=poisson)
par(mfrow=c(2,2));plot(ins.glm);par(mfrow=c(1,1))  # diagnostische Plots, s. Regressionsanalyse
model.matrix(ins.glm)  # Modell als Matrix anschauen
summary(ins.glm)  # Intercept = Gruppe A
anova(ins.glm, test="Chi")  # mit welcher Ws.keit sind die Verteilungen gleich?
```

ohne Intercept: '-1'

```
ins.glmi <- glm(count~spray -1, family=poisson)
par(mfrow=c(2,2));plot(ins.glmi);par(mfrow=c(1,1))
model.matrix(ins.glmi)  # Modell als Matrix anschauen
summary(ins.glmi)  # Modelle für jede Gruppe
```

☞ Die Linkfunktion der Poissonverteilung ist allgemein wie folgt definiert: der natürliche Logarithmus des Erwartungswertes $\ln(\mu)$ ist gleich dem linearen Prädiktor $X\beta$ (der wiederum mit η-eta bezeichnet wird). Also mathematisch $\ln(\mu) = X\beta$ oder kurz $\ln(\mu) = \eta$. Daher ist das Modell für die erste Gruppe `"A"`: $\texttt{count}_A = e^{2.67415}$

```
anova(ins.glmi, test="Chi")  # mit welcher Ws.keit sind die Verteilungen gleich?
library(exactLoglinTest)  # Paket für z.B. Monte Carlo Test
ins.mc <- mcexact(count~spray -1, data=InsectSprays)
summary(ins.mc)
```

☞ `mcexact(...)` berechnet einen Monte Carlo Iterationstest für die Gütekoeffizienten (goodness of fit – deviance und pearson) des GLM Modells; ausgegeben werden die zwei deviance und `pearson` sowie die P Werte (?Ws.keit, daß die Fehlerabweichungen durch das Modell erklärt werden) und die Standardfehler `mcse`

Konfidenzintervalle

```
exp(cbind(coef(ins.glmi), confint(ins.glmi)))
```

☞ `confint(...)` berechnet die Konfidenzintervalle; `exp(...)` wird verwendet, wegen der Linkfunktion bei Daten mit Poissonverteilung

☞ Für den Fall, daß es mehrere Möglichkeiten für die Linkfunktion gibt, schreibt man: `family=Familienname(link=Linkfunktion)` Beispiel: `mod <- glm(y~x1+x2, family=binomial(link=probit), data=sales)`

4.4 Clusteranalyse

Siehe Glossar Cluster Analyse Verfahren, k-means, Modell basiertes Clustering, Fixed Point Cluster Analyse und auch Oksanen (2008). In jedem Fall kann man Cluster mit `plot(...)` darstellen. Mehr Möglichkeiten zeigt das Beispiel `example(dendrogram)` aus dem `stats` Paket. Ebenso kann man zu jedem Punkt im Diagramm eine Funktion was machen lassen. Siehe `example(dendrapply)`.

4.4.1 Hierarchische Clusteranalyse

Für hierarchische Methoden wählt man am besten das Package `amap` aus. (Installation/Benutzung s. Kapitel 1.4 auf Seite 10)

hcluster(...)
amap

Clusteranalyse - amap Paket

```
data("USArrests")  # Daten Verhaftungen in USA
?USArrests # Hilfe zum Datensatz
attach(USArrests) # in den Suchpfad
library(amap)  # Paket laden
hc <- hcluster(USArrests, method = "euclidean",link = "ward")
```

...Fortsetzung umseitig

```
plot(hc, hang=-1 , main="Ward - Methode", xlab="Vergleich Kriminalitaetsraten\n in den USA",
sub="")
rect.hclust(hc, k=3, border="red")  # stats-package, Box bei 3 Gruppen zeichnen
rect.hclust(hc, h=50, which=c(2,7))  # Box zeichnen mit Clusterangabe 2 und 7
```

☞ mit `t(USArrests)` kann die Datenmatrix `USArrests` auch transponiert werden, `method = "euclidean"` Euklid-Distanz, `link = "ward"` Clustermethode nach Ward; `plot(hc, ...)` zeichnen des berechneten Clusterobjektes, `hang=-1` zieht das Cluster bis ganz nach unten, `main="Ward - Methode"` Titel, `xlab="..."` x-Labelbeschriftung, `sub=""` Untertitel löschen; `rect.hclust(...)` (aus stats-package) zeichnet bei `k=3` 3 Clustern rote Box (border="red"); 2. Beispiel `rect.hclust(...) h=50` (Distanz - „height“) gleich 50 (entspricht der Distanz der Gruppen untereinander – abhängig von Distanzmethode); `which=c(2,7)` zeichnet um Cluster 2 und 7

farbabhängige Gruppierung

```
cutree(hc, k=3) -> gruppierung  # Gruppenzugehörigkeit bei k=3 Gruppen
wievielgruppen <- length(table(gruppierung))  # Anzahl der Gruppen
plot(Assault, UrbanPop,
  col=rainbow(wievielgruppen)[gruppierung],  # 3 Regenbogenfarben
     # gruppierung gibt:  1 1 1 2 1 2 3  für Farbe
  pch=16,  # Punkttyp (S.28)
  xlab="Angriffe pro 100 000",  # x-Achsenbeschriftung
  ylab="Urbane Bevölkerung in %",  # y-Achsenbeschriftung
  main= paste("data(USArrests) - Verhaftungen
    Farben nach Gruppierung (Ward", wievielgruppen ,"Gruppen)")
     # Titelei \n = Zeilenumbruch
)
detach(package:amap)  # Paket wieder entfernen
```

Das Package `stats` (ab Version 2.0) bietet auch Möglichkeiten hierarchischer Clusteralgorithmen.

hclust(...)
stats

```
library(stats)  # Paket laden
ostr <- read.table("ostr.csv", sep=";",header=TRUE, dec=",",row.names=1)  # Daten einlesen
hc.ostr <- hclust(dist(ostr)^2, method="ward")  # Ward mit quadr. Euklid-Distanz[17]
```

☞ transponieren geht ganz einfach mit `t(...) hclust(dist(t(ostr))^2, method="ward")`

```
plot(hc.ostr, hang =-1)  # Cluster anzeigen
detach(package:stats)  # Paket wieder entfernen
```

☞ in `hclust(...)` stehen zur Verfügung: Single Linkage Minimaler Abstand zweier Punkte Complete Linkage Maximaler Abstand zweier Punkte Average Linkage Durchschnittlicher Abstand der Punkte zweier Cluster Centroid Abstand der Cluster Zentren Ward s Methode (betrachtet die sum of squares bzgl. der Cluster Zentren) `"ward"` (betrachtet die sum of squares bzgl. der Cluster Zentren), `"single"` (minimaler Abstand zweier Punkte), `"complete"` (maximaler Abstand zweier Punkte), `"average"` (durchschnittlicher Abstand der Punkte zweier Cluster), "mcquitty", "median" (Abstand der Cluster Mediane) oder "centroid" (Abstand der Cluster Zentren) ; Distanzmaße in `dist(...)` können sein: "euclidean", "maximum", "manhattan", "canberra", "binary"oder "minkowski".

Tool - rect.hclust(...)

```
rect.hclust(hc.ostr, k=3, border="red")  # Box um k=3 Gruppen zeichnen
rect.hclust(hc.ostr, h=500000, which=c(2,7))  # Box zeichnen mit Clusterangabe
```

☞ `rect.hclust(...)` umrahmt zuerst 3 Gruppen, dann durch die Angabe `h=500000, which=c(2,7)` bei der Distanz („height“) `h=500000` in den Clustern 2 und 7 entsprechende Rahmen

Tool - identify(...)

```
(ausgeben <- identify(hc.ostr))  # Cluster identifizieren
```

☞ identify(...) identifiziert Gruppen und druckt die entsprechenden Zeilen- oder Spaltennumern aus

4.4.2 k-means Algorithmus

Eine k-means Clusteranalyse läßt sich mit der Funktion `kmeans(...)` aus dem package `stats` durchführen .

kmeans(...)
stats

```
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))  # Zufallszahlen erzeugen
cl <- kmeans(x, 2, 20)  # 2 Zentren, 20 Iterationen
plot(x, col = cl$cluster)  # Punkte zeichnen
```

...Fortsetzung umseitig

[17] In Pruscha (2006) steht Ward wird mit quadr. Euklid-Distanz gerechnet. Jedoch findet man hin und wieder auch von R-Statistikern die einfache Euklid-Distanz angewendet: ???

```
points(cl$centers, col = 1:2, pch = 8)  # Farbe nach den Clustern zeichnen
```

☞ Ausgabe ...$cluster Clusterzugehörigkeit, ...$centers Clusterzentren, ...$withinss Fehlerquadratsumme der Cluster, ...$size Verteilung der Anzahlen in die Cluster

4.4.3 k - medoid Algorithmus

pma(...)
cluster

Eine k - medoid Clusteranalyse läßt sich mit der Funktion `pma(...)` aus dem package `cluster` durchführen.

```
x <- rbind(cbind(rnorm(10,0,0.5), rnorm(10,0,0.5)),
           cbind(rnorm(15,5,0.5), rnorm(15,5,0.5)))  # Zufallsdaten erzeugen
plot(x)  # Punkte anschauen
pamx <- pam(x, 2)  # k - medoid mit 2 Clustern durchführen
pamx  # Infos anzeigen
summary(pamx)  # Zs.fassung
plot(pamx)  # geclusterte PCA + Silhouette Grafik
```

☞ Die Silhouette Grafik stellt für die partitionierenden Verfahren dar, wie gut die einzelnen Objekte in ihrem Cluster liegen: $s_i = 1$ Objekt liegt gut in seinem Cluster; $s_i \approx 0$ ... liegt zwischen 2 Clustern; $s_i < 0$ Objekt wahrscheinlich im falschen Cluster. (Silhouette in R mit `silhouette(...)`)

4.4.4 Modellbasierte Cluster

Ein sogenanntes Modell basiertes Clustering läßt sich mit der Funktion `Mclust(...)` aus dem Paket `mclust` berechnen.

```
library(mclust)  # Paket laden
data(iris)  # Daten laden
?iris # Hilfe zum Datensatz
irisMatrix <- as.matrix(iris[,1:4])  # Matrize erzeugen
irisMclust <- Mclust(irisMatrix)
plot(irisMclust)  # BIC Werte für Anzahl der Gruppen
plot(irisMclust,irisMatrix)  # interaktiver Plot
...
0:exit
1:plot: BIC - für jede Gruppe Anz. Dimensionen
2:plot: Pairs - wie der Plot pairs(...)
3:plot: Classification (2-D projection) - zeichnet berechnetes Cluster
4:plot: Uncertainty (2-D projection) - zeigt ?ungewisses Cluster
5:plot: All  # alle Graphen nacheinander
summary(irisMclust)  # bestes Modell ausgeben
detach(package:mclust)
```

4.4.5 Fixed Point Cluster

Fixed Point Cluster Analysen lassen sich mit dem Paket `fpc` durchführen.

```
library(fpc)  # Paket laden
set.seed(190000)  # Beginn Zufallsgenerator: jetzt reproduzierbar!
data(tonedata)  # Daten zu Tonexperiment
?tonedata # Hilfe zum Datensatz
attach(tonedata)  # Suchpfadaufnahme
tonefix <- fixreg(stretchratio,tuned,mtf=1,ir=20,plot=T)
```

☞ `fixreg(...)` Optionen: allgemein: `fixreg(unabh-Var, abh-Var)`; `ca=3` 3 Cluster vorgeben; `mtf=4` es müssen 4 mal dieselben Cluster gefunden werden, bevor aufgehört werden darf; `ir=20` Iterationsanzahl; `plot=T` simultan einen Plot zeichnen

```
summary(tonefix)  # Ausgabe anzeigen
detach(package:fpc)  # Paket wieder entfernen
```

...Fortsetzung umseitig

4.4.6 Clustern von Artenabständen bei Speziesdaten aus Binärmatrizen

Hat man Artdaten in Form von Binärdaten, z.B.: Fundorte, so kann man mit der Funktion `prabclust(...)` aus dem Paket `prabclus` herausfinden, ob die Arten zufällig streuen, oder ob sich Gruppen finden lassen.

```
library(prabclus)   # Paket laden
example(prabclust)   # Beispiel anzeigen
```
☞ Beispiel zeigt Daten von Meeresschnecken aus dem Ägäischen Meer; als Summary wird eine MDS dargestellt, „N“ bezeichnet Rauschen
```
detach(package:prabclus)   # Paket wieder entfernen
```

4.4.7 Tests - Cluster

Für die grafische Clustergüte (s. Anm. auf der vorherigen Seite) kann die Funktion `silhouette(..)` im Paket `cluster` in Zusammenhang mit partitionierenden Verfahren wie k-means oder k-medoid verwendet werden, denn die Gruppengröße MUß vorgegeben sein. Eine andere Möglichkeit ist die Gruppengröße mit `cutree()` festzulegen. Hier ein Beispiel mit Grafik der optimalen Gruppengröße:

```
library(vegan)  # Paket laden
library(cluster)# Paket laden
data(dune) #Datensatz laden
dune.dist <- vegdist( # Bray Curtis Distanz vegan Paket
  t(dune) # transponieren für Arten
)
cluster <- hclust(
  dune.dist,
  method="complete"
)
plot(cluster, # Denrogramm anschauen
  hang=-1, # Denrogramm bis runter ziehen
  cex=1.3  # Schrift größer, kleiner
)
nCluster <- 20 # Anzahl Cluster speichern
# "average s-i width"  als asw mal abgekürzt
# ist gleich Maß für die Clustergüte:
# -1 .. 0 .. 1 ist gleich: schlecht .. zwischen Clustern .. gut im Cluster
asw <- numeric(nCluster) # erzeugt: 0,0,0,0,0,0,...
#### durchlaufe mehrere Clustermöglichkeiten
for(k in 2:nCluster){# für 'k' von 2 bis nCluster mache:
  clusterSilhouette <- silhouette(#silhouette speichern
    x=cutree(cluster, k=k), # k-Gruppen berechnen
    dist=dune.dist # Distanz
  )
  #
  asw[k] <- summary(clusterSilhouette)$avg.width
}
k.best <- which.max(asw) # was ist der maximale Wert?
# Ausgabe als Info in Konsole
cat("silhouette-optimal number of clusters:", k.best, "\n")
#### Grafik
plot(1:nCluster, asw,
  type= "h", # "h"-plottyp histogrammartig
  main = paste("Cluster:",cluster$method,"Distanz:",cluster$dist.method),
  xlab= "k  (# cluster-Gruppen)",
  ylab = "durchschnittliche silhouette-Breite"
```

```
)
#### Achse mit Info
axis(1, k.best, paste("bestes k",k.best,sep="\n"), col = "red", col.axis = "red")
detach(package:vegan)# Paket entfernen
detach(package:cluster)# Paket entfernen
```

Gruppenvergleich analog einem Chi^2 - Test zweier Arten ist mit `comp.test(...)` aus dem Paket `prabclus` möglich.

```
set.seed(1234) # Beginn Zufallsgenerator: jetzt reproduzierbar!
g1 <- c(rep(1,34), rep(2,12), rep(3,15))  # Probe 111...222...3333
g2 <- sample(3,61,replace=TRUE)  # Zufallsample 122232112...
comp.test(g1,g2)  # Vergleich
 ----8<-- output -->8-------
        Pearson's Chi-squared test with simulated p-value
        (based on 10000 replicates)

data: cl and spg
X-squared = 2.2804, df = NA, p-value = 0.6889
```

☞ $p = 0.6889$ die Zähldaten der Proben sind homogen

bootstrap für hierarchische Cluster Analyse Verfahren ist möglich mit `pvclust(...)` aus dem Paket `pvclust` sowie für presence-absence Cluster Analyse Verfahren mit `prabtest(...)` aus dem Paket `prabclus`.

```
                                    bootstrap - hierarchische Cluster
library(MASS); library(pvclust)  # Pakete laden
data(Boston)  # Häuserdaten von Bostons Vororten
?Boston # Hilfe zum Datensatz
boston.pv <- pvclust(Boston, nboot=100)  # multiscale bootstrap resampling
# ACHTUNG: nboot=100 könnte etwas zu wenig sein.
#  nboot=1000 oder größer sind besser,
#  dauern aber länger zur Berechnung
plot(boston.pv, hang=-1)  # Dendrogram mit p-Werten in %
ask.bak <- par()$ask  # par(ask)-Nachfragen zwischenspeichern
par(ask=TRUE)
pvrect(boston.pv)  # Clusters mit hohem au p-Value zeichnen
print(boston.pv, digits=3)  # Ergebnis des multiscale bootstrap resampling
msplot(boston.pv, edges=c(2,4,6,7))  # plot diagnostic for curve fitting
par(ask=ask.bak)  # par(ask) Nachfragen zurücksetzten
boston.pp <- pvpick(boston.pv)  # Cluster mit hohem p-Wert
boston.pp  # Clusterzugehörigkeit
detach(package:MASS); detach(package:pvclust)  # Pakete entfernen
```

☞ aus der Hilfe (`?msfit`) von R: `AU` ist der „approximately unbiased“ p-Wert, der akkurater ist als der `BP` p-Wert (bootstrap probability). Mit `edge` werden die einzelnen Cluster bezeichnet. `pchi` ist der p-Wert des Chi^2 - Test basierend auf „asymptotic theory“[18]

```
                                    bootstrap - presence-absence Cluster
... kommt noch s.: help("prabtest", package="prabclus")
```

4.4.8 Ähnlichkeitsvergleich – Matrizen

Mit dem Mantel - Test aus dem package `vegan` lassen sich zwei Matrizen auf ihre Gleichheit prüfen:

[18] ?

```
Manteltest - mantel(...)
test <- cbind (  # Testdaten erzeugen
  "sample1"= "sample"<- sample(30),  # 30 Zufallsdaten
  "sample2"= "sample"<- sample(30),  # 30 Zufallsdaten
  "sample3"= "sample"<- sample(c(5,6), 30, replace = TRUE),  # 30x zw. 5 oder 6 + Variation
  "sample4"= "sample"<- sample3*4  # "sample3" mal 4
)
rownames(test) <- ("Arten"<- paste("Art_",1:30, sep="")) # Artnamen zs.fügen
test  # Tabelle anschauen
library(vegan)  # paket laden
  mantel(dist(test[,1]), dist(test[,2]))  # Mantel statistic r: -0.0546 Significance: 0.841
  mantel(dist(test[,1]), dist(test[,3]))  # Mantel statistic r: -0.0004416 Significance: 0.372
  mantel(dist(test[,3]), dist(test[,4]))  # Mantel statistic r: 1 Significance: <0.001
detach(package:vegan)  # Paket wieder entfernen
```

☞ `method="collector"` nimmt Proben hinzu, wie sie gefunden wurden; `method="random"` nimmt Proben in zufälliger Reihenfolge hinzu; `method="exact"` findet die zu erwartende (mittlere) Richness; `method="coleman"` findet die zu erwartende Richness nach Coleman u. a. (1982); `method="rarefaction"` (=„Verdünnung“) findet mittlere Artenzahl wenn die Arten statt der Proben aufsummiert werden.
☞ Da die Nullhypothese H_0 lautet: die Matrizen sind verschieden, lautet das Testergebnis also: die Gruppen 1-2, 1-3 sind nicht gleich und korrelieren auch nicht miteinander, da r praktisch 0 ist.; 3-4 korrelieren absolut identisch, da $r = 1$ – ist ja auch nicht anders zu erwarten. Die Signifikanz bei Vergleich 3-4 zeig an, daß die Proben als gleich angesehen werden dürfen.
☞ Bestehen die Daten aus 2 Spalten mit einer für Arten und einer für Proben, wie rechts zu sehen, dann kann man, wenn die Daten im Datenframe `test` enthalten sind, mit `table(test)` eine Häufigkeitstabelle generieren.

Arten	Probe
Chironomus sp.	1
Chironomus sp.	1
Tanytarsus sp.	1
Tanytarsus sp.	2
Tanytarsus sp.	2
Micropsectra sp.	2
⋮	⋮

4.4.9 Visualisierung von Clustern

Diverse Spezifikationen bei Clustern können mit `dendrogram(...)` gemacht werden. Ob nun Text an die Knoten soll oder Linien blau gestrichelt werden sollen oder die Cluster sich dichotom, statt rechteckig... alles möglich.

```
# s. auch:   # par(ask=TRUE) Grafik nur wenn Benutzer will
# example(dendrogram)
# example(dendrapply)
Visualisierung von Clustern
hc <- hclust(dist(USArrests), "ave") # Verhaftungen in USA
(dend1 <- as.dendrogram(hc))
str(dend1) # Struktur ausgeben
str(dend1, max = 2)  # nur erste zwei Untergruppen
op <- par(mfrow= c(2,2), # 2x2 Grafiken
  mar = c(5,2,1,4))  # Grafikeinstellungen u, li, o, re
plot(dend1) # einfaches Diagramm
Triangel Typ + Knoten
plot(dend1,
  nodePar=list( # nodePar für node Parameter
    pch = c(1,1),  # Punkttypen an Knoten (S.28)
    cex=0.8, # Skalierung
    lab.cex = 0.8), # Labelgröße
  type = "t", # Typ: "rectangle"oder "triangle"
  center=TRUE) # Knoten zentrieren
Linientypen & Farben
plot(dend1,
  edgePar=list( # Verzweigungsprameter (edge)
    col = c("blue", "red"),  # unterschiedliche Farben
    lty = 2:3  # Linientypen
    ),
  dLeaf=1, # Beschriftungsabstand
  edge.root = TRUE)  # Clusteranfang TRUE/FALSE
```

...Fortsetzung umseitig

horizontal

```
plot(dend1,
  nodePar=list(
    pch = 2:1, # Punkttypen
    cex= 0.4*2:1, # jeweilige Skalierung
    col = 2:3), # Farben im Nummerncode
    horiz=TRUE) # Grafik horizontal
```

Dendrogramm verkürzen

```
dend2 <- cut(dend1, h=70) # bei einer „Höhe"(Distanz) von 70
dend2
# --8<-- Ausgabe der Unterteilung
 $upper
 'dendrogram' with 2 branches and 4 members total, at height 152.314
 $lower
 $lower[[1]]
 'dendrogram' with 2 branches and 2 members total, at height 38.52791
 $lower[[2]]
 'dendrogram' with 2 branches and 14 members total, at height 44.28392
 $lower[[3]]
 'dendrogram' with 2 branches and 14 members total, at height 44.83793
 $lower[[4]]
 'dendrogram' with 2 branches and 20 members total, at height 54.74683
```

Dendrogramm verkürzen: oberes Cluster zeichnen

```
plot(dend2$upper,
  nodePar=list(
    pch = c(1,7), # Punkttypen S.28
    col = 2:1)
  )
```

Dendrogramm verkürzen: unteres Cluster zeichnen

```
# dend2$lower ist !!KEIN!! Dendrogramm, aber ne Liste von... :
plot(dend2$lower[[3]],
  nodePar=list(col=4),
  horiz = TRUE,
  type = "tr")
```

verschiedene Farben/Punkttypen

```
plot(dend2$lower[[2]],
  nodePar=list(col=1), # darf keine leere Liste sein
  edgePar = list(lty=1:2, col=2:1),
  edge.root=TRUE)
par(op) # Grafikeinstellungen zurück
```

Bsp. mit Funktionsanwendung

```
str(dend3 <- dend2$lower[[2]][[2]][[1]])

knotenParameter <- list(col=3:2,
  cex=c(2.0, 0.75),
  pch= 21:22,
  bg= c("light blue", "pink"),
  lab.cex = 0.75,
  lab.col = "tomato")
plot(dend3,
  nodePar= knotenParameter,
  edgePar = list(col="gray", lwd=2),
  horiz = TRUE)
```

...Fortsetzung umseitig

```
randFkt <- function(n) {
  if(!is.leaf(n)) {
    attr(n, "edgePar") <- list(p.col="lightblue") # Farbe
    attr(n, "edgetext") <- paste(attr(n,"members"),"\n Gruppen") # Text: 2 Gruppen
  }
  n  # n intern ausgeben
}
```

dendrapply(...)

```
dendRand3 <- dendrapply(dend3, randFkt)
plot(dendRand3, nodePar= knotenParameter)
plot(dendRand3,
  nodePar= knotenParameter,
  leaflab = "textlike")  # Endpunkte als Text
```

Umrandungspolygone um die jeweiligen Cluster läßt sich mit der Funktion `chull()` zeichnen. Im folgenden Beispiel wird `chull()` mit `lapply()` verbunden, um `chull()` auf die Daten anzuwenden.

```
par(no.readonly=TRUE) -> paralt   # alte Grafikeinstellungen speichern
set.seed(1)   # Zufallsgenerator auf Start=1 (damit wiederholbar!)
(xy <- matrix(runif(500, 0, 10), ncol=2))  # Matrix 2 x 250
# complete linkage Hierarchisches Gruppieren
xy_cluster <- hclust(dist(xy), method="complete")
# Grafik ansehen
par(las=1)  # S.27, Label assignment
plot(xy_cluster, hang=-1,  # bis y-Achse herunterziehen
  cex=0.4,                 # Beschriftung kleiner
  ylab="Distanz"            # y-Label
)
```

Cluster unterteilen

```
gruppen <- 10;cat(" # > v v Indizes für",gruppen,"Gruppen\n")
(cl_10 <- cutree(xy_cluster, gruppen))  # Indizes ausgeben
rect.hclust(xy_cluster, k=gruppen, border="red")
gruppen <- 20; cat(" # > v v Indizes für",gruppen,"Gruppen\n")
(cl_20 <- cutree(xy_cluster, gruppen))  # Indizes ausgeben
rect.hclust(xy_cluster, k=gruppen, border="blue")
gruppen <- 30; cat(" # > v v Indizes für",gruppen,"Gruppen\n")
(cl_30 <- cutree(xy_cluster, gruppen))  # Indizes ausgeben
rect.hclust(xy_cluster, k=gruppen, border="darkgreen")
```

Legende

```
legend("topleft", # Position
  legend=c(10,20,30),
  title="Gruppen",
  text.col=c("red","blue","darkgreen"),
  bty="n"    # Boxtyp
)
legend("topleft",  # schwarzen Titel nochmal drüber
  bty="n",  # Boxtyp
  legend="",
  title="Gruppen"
)
```

Dendrogramm abschneiden - enthält Indizes der Gruppierung

```
(welche_cl_10 <- tapply(1:nrow(xy), cl_10, function(i) xy[i,]))
(umrand_cl_10 <- lapply(welche_cl_10, function(x) x[chull(x),]))
```

...Fortsetzung umseitig

```
# convex hull polygons for each cluster
plot(xy,
  main=paste(max(cl_10),"Gruppen"),
  col=rainbow(max(cl_10))[cl_10], # Farbe datenabhängig
  pch=16    # Punkttyp S. 28
)
res <- lapply(umrand_cl_10, polygon)  # Funktion polygon anwenden
```

```
# Grafiken für 20, 30 Gruppen
welche_cl_20 <- tapply(1:nrow(xy), cl_20, function(i) xy[i,])
umrand_cl_20 <- lapply(welche_cl_20, function(x) x[chull(x),])
plot(xy,
  main=paste(max(cl_20),"Gruppen"),
  col=rainbow(max(cl_20))[cl_20], # Farbe datenabhängig
  pch=16    # Punkttyp S. 28
)
```

```
res <- lapply(umrand_cl_20, polygon)
welche_cl_30 <- tapply(1:nrow(xy), cl_30, function(i) xy[i,])
umrand_cl_30 <- lapply(welche_cl_30, function(x) x[chull(x),])
plot(xy,
  main=paste(max(cl_30),"Gruppen"),
  col=rainbow(max(cl_30))[cl_30], # Farbe datenabhängig
  pch=16    # Punkttyp, S. 28
)
res <- lapply(umrand_cl_30, polygon)
par(paralt)        # alte Grafikeinstellungen zurücksetzten
```

4.4.10 Heatmaps

Eine heatmap ist eine Falschfarbendarstellung üblicherweise großer Datenmengen, bei der Cluster-Dendrogramme zusätzlich für Spalten- und Reihendaten dazugezeichnet werden. Es findet auch eine Neuordnung der Daten nach Spalten- bzw. Reihen-Mittelwert statt.

```
library(palaeo) # Steve Juggins Paket
  # install.packages("palaeo",contriburl="http://www.campus.ncl.ac.uk/staff/Stephen.Juggins/data/"
  # oder manuell: http://www.campus.ncl.ac.uk/staff/Stephen.Juggins/data/palaeo_1.0.zip
data(aber) # Daten laden library(palaeo)
library(gplots) # für heatmap.2()
library(vegan) # für besseres Distanzmaß vegdist()
library(plotrix) # für gradient.rect() - Legende
######## heatmap.2 aus gplots - Paket
# Farben für Ränder
# aber.m <- as.matrix(aber) # Daten in matrix umwandeln, falls kein dataframe
  r.col <- topo.colors(nrow(aber)) # "row color" topografische Farben #, start=0, end=.9)
  c.col <- topo.colors(ncol(aber)) # "column color" start=0, end=.9)
  heatmap.farbe <- heat.colors(64)[64:1] # 64 heat.colors umkehren mit [64:1]
heatmap.2(
  aber,
  distfun = vegdist,
  hclustfun = function(d) hclust(d, method = "single"), # hier Clustermethode
  col=heatmap.farbe,
  tracecol="gray50", # "Spur" Farbe
  # margins = c(5, 15), # Platz Spalten (unten) / Reihen (rechts)
  RowSideColors=r.col, # Indexfarbe für Daten (Reihen)
  ColSideColors=c.col # Indexfarbe für Daten (Spalten)
  # scale="column" # standardisieren
  # weitere Argumente s. ?heatmap.2
)
```

```
## Legende für Indexfarbe (Daten) mit Mausklick dazu
# ohne xpd=TRUE läßt sich nicht außerhalb der Grafikfläche zeichnen
par(xpd=TRUE) -> grafikeinstellung.alt
  locator(1) -> liunten # Mausposition
  locator(1) -> reoben   # Mausposition
  gradient.rect(liunten$x, liunten$y, reoben$x, reoben$y,
    col=topo.colors(ncol(aber)) # Farbe Legende
  )
  legende.y <- mean(liunten$y, reoben$y)
   # Text schreiben: Koordinaten x, y, "Text", adj-Ausrichtung (x,y), cex-Verkleinerung
  text(liunten$x, legende.y,"data\nIndex\nlow" , adj=c(1.1,0.2), cex=0.6 )
  text(reoben$x, legende.y,"data\nIndex\nhight", adj=c(-0.1,0.2), cex=0.6)
par(grafikeinstellung.alt) # Grafikeinstellung wieder zurücksetzen
######## ohne Sortierung
sortierung <- heatmap.2(
  aber, # daten muß Matrize sein
  Rowv=FALSE, # Sortierung aus/an
  Colv=TRUE,   # Sortierung aus/an
  distfun = function(c) dist(c), # Distanzmaß
  hclustfun = function(d) hclust(d, method = "ward"), # Clustermethode
  tracecol="gray50",     # "Spur" Farbe
  RowSideColors=r.col,   # Indexfarbe für Daten (Reihen)
   # ColSideColors=c.col, # Indexfarbe für Daten (Spalten)
  col=heatmap.farbe,
   # scale="column",     # standardisieren
  dendrogram = "column"   # welches dendrogramm "both","row","column","none"
   # weitere Argumente s. '?heatmap.2'
)
sortierung   # Sortierung ausgeben
title("Rowv=FALSE\nSortierung aus/an")      # Titelei
# rm(list=ls()) # alles löschen
detache(package:palaeo) # Paket entfernen: Steve Juggins Paket
detache(package:gplots) # Paket entfernen: für heatmap.2()
detache(package:vegan) # Paket entfernen: für besseres Distanzmaß vegdist()
detache(package:plotrix) # Paket entfernen: für gradient.rect() - Legende
```

4.4.11 Entscheidungs„hilfe“ Distanzmaß

Die folgenden „Bestimmungswege“ sind aus Legendre und Legendre (1998) S.299 [19]. Siehe auch am Ende der „Bestimmungswege“: Tabelle 3 auf Seite 104.
Zur Berechnung von Distanzmaßen stehen folgende Funktionen zur Verfügung:

`dist(...) - stats` ..... allg./gängige Distanzmaße
`Dist(...) - amap` ..... allg./gängige Distanzmaße
`distance(...) - (ecodist)` ..... allg. + auch Mahalanobis Distanz
`vegdist(...) - vegan` ..... auch spezielle Distanzen für Ökologie (z.B.: Bray-Curtis)
`jaccard(...) - (prabclus)` ..... Jaccard Distanz
`kulczynski(...) - (prabclus)` ..... Kulczynski Distanz

☞ Im folgenden für Objekte Q-Modus, die mit Arten als Deskriptoren (asymmetrische Koeffizienten) assoziiert werden sollen

1. Descriptors: presence-absence or ordered classes on a scale of relative abundances (no partial similarities computed between classes)

[19] dabei entsprechen die Similarity-Indizes (S_{index}) oder Distanz-Indizes (D_{index}) denen aus dem Programm „R Package“ (http://www.bio.umontreal.ca/Casgrain/R/index.html), das es eigentlich nur für Mac gibt, aber wohl auch mit Windowsemulatoren laufen soll

2. Metric coefficients: . . . . . **coefficient of community (S_7) and variants (S_{10}, S_{11})**
   Semi metric coefficients: . . . . . **variants of the coef. community (S_8, S_9, S_{13}, S_{14})**
   Nonmetric coefficient: . . . . . **Kulczynski (S_{12}) (non-linear: not recommended)**
2. Probabilistic coefficient: . . . . . **S_{27}**

1. Descriptors: quantitative or semiquantitative (states defined in such a way that partial similarities can be computed between them)
   3. Data: raw abundances
      4. Coefficients without associated probability levels
         5. No standardization by object; the same difference for either abundant or rare species, contributes equally to the similarity between sites: . **coefficients of Steinhaus (S_{17}) and Kulczynski (S_{18})**
         5. Standardization by object-vector; differences for abundant species (in the whole data set) contribute more than differences between rare species to the similarity (less to the distance) between sites: **χ^2 similarity (S_{21}), χ^2 metric (D_{15}), χ^2 dist. (D_{16}), Hellinger dist. (D_{17})**
      4. Probabilistic coefficient: . . . . . **probabilistic χ^2 similarity (S_{22})**
   3. Data: normalized abundances (or, at least, distributions not skewed) or classes on a scale of relative abundances (e.g. 0 to 5, 0 to 7). [Normalization is useful when abundances cover several orders of magnitude]
      6. Coefficients without associated probability levels
         7. No standardization by object
            8. The same difference for either abundant or rare species, contributes equally to the similarity between sites: . . . . . **coefficients of Steinhaus (S_{17}) and Kulczynski (S_{18})**
               . . . . . **mean character difference (D_8), percentage difference (D_{14})**
               Differences for abundant species (for the two sites under consideration) contribute more than differences between rare species to the similarity (less to the distance) between sites: **Canberra metric (D_{10}), coefficient of divergence (D_{11})**[20]
            8. Differences for abundant species (in the whole data set) contribute more than differences between rare species to the similarity (less to the distance) between sites: **asymmetrical Gower coefficient (S_{19}),**
               . . . . . **coefficient of Legendre & Chodorowski (S_{20})**
         7. Standardization by object-vector; if objects are of equal importance, same contributions for abundant or rare species to the similarity between sites: . . **chord distance (D_3), geodesic metric (D_4),**
            . . . . . **complement of index of association (D_9)**
      6. Probabilistic coefficient: . . . . . **Goodall coefficient (S_{23})**

☞ Im folgenden für Assoziationen von Objekten (Q - Modus), bei denen chemische, geologische, physikalische ... Deskriptoren benutzt werden (symmetrische Koeffizienten, die Doppel Nullen[21] benutzen)

1. Association measured between individual objects
   2. Descriptors: presence-absence or multistate (no partial similarities computed between states)
      3. Metric coefficients: . . . . . **simple matching (S_1) and derived coefficients (S_2, S_6)**
         Semimetric coefficients: . . . . . **S_3, S_5**
      3. Nonmetric coefficient: . . . . . **S_4**
   2. Descriptors: multistate (states defined in such a way that partial similarities can be computed between them)
      4. Descriptors: quantitative and dimensionally homogeneous
         5. Differences enhanced by squaring: . . . . **Euclidean distance (D_1) and average distance (D_2)**

[21] ?

5. Differences mitigated: . . . . . **Manhattan metric (D_7), mean character difference (D_8)**

4. Descriptors: not dimensionally homogeneous; weights (equal or not, according to values w_j used) given to each descriptor in the computation of association measures

6. Descriptors are qualitative (no partial similarities computed between states) and quantitative (partial similarities based on the range of variation of each descriptor):
symmetrical Gower coefficient (S_{15})

6. Descriptors are qualitative (possibility of using matrices of partial similarities between states) and semiquantitative or quantitative (partial similarity function for each descriptor):
coefficient of Estabrook & Rogers (S_{16})

1. Association measured between groups of objects

7. Removing the effect of correlations among descriptors: . . . **Mahalanobis generalized distance (D_5)**

7. Not removing the effect of correlations among descriptors: . . . . . **coefficient of racial likeness (D_{12})**

☞ Im folgenden beim Bestimmen von Abhängigkeiten von Deskriptoren (R - Modus)

1. Descriptors: species abundances

2. Descriptors: presence-absence

3. Coefficients without associated probability levels:[22] . . . . . .

3. Probabilistic coefficient:[22] . . . . . .

2. Descriptors: multistate

4. Data are raw abundances: . . . . . χ^2 **similarity (S_{21}), χ^2 metric (D_{15}), χ^2 distance (D_{16}),** . . . . . **Hellinger distance (D_{17}), Spearman r, Kendall τ**

4. Data are normalized abundances

5. Coefficients without associated probability levels:
covariance or Pearson r, after elimination of as much double-zeros as possible, ... **Spearrnan r, Kendall τ**

5. Probabilistic coefficients:
probabilities associated to Pearson r, Spearman r or Kendall τ, ... **Goodall coefficient (S_{23})**

1. Descriptors: chemical, geological, physical, etc.

6. Coefficients without associated probability levels

7. Descriptors are quantitative and linearly related: . . . . . **covariance, Pearson r**

Descriptors are ordered and monotonically related: . . . . . **Spearman r, Kendall τ**

7. Descriptors are qualitative or ordered but not monotonically related:
χ^2, reciprocal information coefficient, symmetric uncertainty coefficient

6. Probabilistic coefficients

8. Descriptors are quantitative and linearly related: . . . . . **probabilities associated to Pearson r**

Descriptors are ordered and monotonically related:
probabilities associated to Spearman r and Kendall τ

8. Descriptors are ordered and monotonically related: . . . . . **probabilities associated to χ^2**

Für die in Tabelle 3 auf der nächsten Seite stehenden Gleichungen gilt für presence-absence Daten:

[22] in Legendre und Legendre (1998) steht an dieser Stelle nichts: ?Duckfehler oder gemeint siehe weiter unten bei „Coefficients without associated...“

Art $_2$ \ Art $_1$	1	0
1	a	b
0	c	d

, wobei mit p gemeint ist: $p = a + b + c + d$

Für quantitative Distanzmaße soll das folgende Beispiel die Bedeutung der Buchstaben A, B und W verdeutlichen:

	Artabundanzen						A	B	W
Probe x_1	7	3	0	5	0	1	$\sum 16$		
Probe x_2	2	4	7	6	0	3		$\sum 22$	
Minimum	2	3	0	5	0	1			$\sum 11$

Tabelle 3: Ähnlichkeiten (similarities)/ Distanzen aus Legendre und Legendre (1998)

similarities distances	Referenz/Name	Gleichung	R-Funktion
S_1	simple matching coefficient Sokal und Michener (1958)	$\frac{a+d}{p}$	`dist.binary(..., method=2) ade4`
S_2	coefficient of Rogers und Tanimoto (1960)	$\frac{a+d}{a+2b+2c+d}$	`dist.binary(..., method=4) ade4`
$S_{3...6}$	Sokal und Sneath (1963)	$S_3 = \frac{2a+2d}{2a+b+c+2d}$, $S_4 = \frac{a+d}{b+c}$, $S_5 = \frac{1}{4}\left[\frac{a}{a+b} + \frac{a}{a+c} + \frac{d}{b+d} + \frac{d}{c+d}\right]$	
S_6	Sokal und Sneath (1963)	$\frac{a}{\sqrt{(a+b)(a+c)}} \frac{d}{\sqrt{(b+d)(c+d)}}$	`dist.binary(..., method=8) ade4`
S_7	Jaccard's coefficient	$\frac{a}{a+b+c}$	`dist.binary(..., method=1) ade4`
S_8	Sørensen's coefficient	$\frac{2a}{2a+b+c}$	`dist.binary(..., method=5) ade4`
S_9		$\frac{3a}{3a+b+c}$	
S_{10}	Sokal und Sneath (1963)	$\frac{a}{a+2a+2c}$	`dist.binary(..., method=3) ade4`
S_{11}	Russel und Rao (1940)	$\frac{a}{p}$	`dist.binary(..., method=10) ade4`
S_{12}	Kulczynski (1928)	$\frac{a}{b+c}$	`kulczynski()` `prabclus`
S_{13}	Sokal und Sneath (1963) von Kulczynski (1928) abgeleitet	$\frac{1}{2}\left[\frac{a}{a+b} + \frac{a}{a+c}\right]$	
S_{14}	Ochiai (1957)	$\frac{a}{\sqrt{(a+b)(a+c)}}$	`dist.binary(..., method=7) ade4`
S_{15}	Gower (1971a)	$\frac{1}{p}\sum_{j=1}^{p} s_{12_j}$	
S_{16}	Estabrook und Roger (1966)		
S_{17}	Steinhaus coefficient by Motyka (1947)	$\frac{2W}{A+B}$	
S_{18}	Kulczynski (1928)	$\frac{1}{2}\left[\frac{W}{A} + \frac{W}{B}\right]$	
S_{19}	Gower (1971a)		
S_{20}	Legendre und Chodorowski (1977)		
S_{21}	χ^2-similarity, komplementär zur χ^2-Distanzmatrix D_{15}	$1 - \chi^2$	

...Fortsetzung umseitig

similarities distances	Referenz/Name	Gleichung	R-Funktion
	Fortsetzung Indizes S und D ...		
S_{22}	probabilistic χ^2_p-similarity		
S_{23}	Goodall's similarity Goodall (1964)	$\frac{2(\sum d)}{n(n-1)}$	
S_{26}	Faith (1983)	$\frac{(\frac{a+d}{2})}{p}$	
D_1	Euklid - Distanz Distanz		`dist(x, method = "euclidean") stats`
D_2	Average Distanz		
D_3	Chord Distanz		`vegdist(decostand(x, "norm"), "euclidean") vegan`
D_4	geodesic Metrik		
D_5	Mahalanobis generalized Distanz		`mahalanobis('stats'), distance('ecodist')`
D_6	Minkowski Distanz		`dist(x, method = "minkowski") stats`
D_7	Manhattan-Metrik Metrik		`dist(x, method = "manhattan") stats`
D_8	Durchschnittliche Differenz Czekanowski (1909)		
D_9	index of association Whittaker (1952)		
D_{10}	Canberra Metrik		`dist(x, method = "canberra") stats`
D_{11}	coefficient of divergence Clark (1952)		
D_{12}	coefficient of racial likeness Pearson (1926)		
D_{13}	nonmetric coefficient Watson, Williams und N. (1966)		
D_{14}	nonmetric coefficient Watson, Williams und N. (1966)		
D_{15}	χ^2-Metrik		
D_{16}	Chi Quadrat (X^2) Distanz		`decostand(x, method="chi.square")`[23] `vegan`
D_{17}	Hellinger Distanz		

Tabelle 4: Eine Übersicht über Clustermethoden aus Legendre und Legendre (1998) soll helfen die richtige Entscheidung zu treffen

Methode	pro & contra	Beispiele in der Ökologie
Hierarchical agglomeration: linkage clustering	Pairwise relationships among the objects are known.	
Single linkage `amap`-Paket: `hcluster(..., link = "single")`	Computation simple; contraction of space (chaining); combinatorial method.	Good complement to ordination.

...Fortsetzung umseitig

[23] ? weiß nicht, ob das stimmt

Methode	pro & contra	Beispiele in der Ökologie
	Fortsetzung Übersicht Clustermethoden	
Complete Linkage `amap`-Paket: `hcluster(..., link = "complete")` (see also species association on the facing page)	Dense nuclei of objects; space expansion; many objects cluster at low similarity; arbitrary Eules to resolve confficts; combinaturial method.	To increase the contrast among clusters.
Intermediate linkage	Preservation of reference space A; non-combinatorial: not included in Lance & Williams' general model.	Preferable to the above in most cases where only one clustering method is to be used.
Hierarchical agglomeration: average clustering	Preservation of reference space A; pairwise relationships between objects are lost; combinatorial method.	
Unweighted arithmetic average (UPGMA)	Fusion uf clusters when the similarity reaches the mean intercluster similarìty value.	For a collection of objects obtained by simple random or systematic sampling.
Weighted arithmetic average (WPGMA)	Same, with adjustment for group sizes.	Preferable to the previous method in all other sampling situations.
Unweighted centroid (UPGMC)	Fusion of clusters with closest centroids; may produce reversals.	For simple random or systematic samples of objects.
Weighted centroid (WPGMC)	Same, with adjustment for group sizes; may produce reversals.	Preferable to the previous method in all other sampling situations.
Ward's method `amap`-Paket: `hcluster(..., link = "ward")`	Minimizes the within-group sum of squares.	When looking for hyperspherical clusters in space A.
Hierarchical agglomeration: flexible clustering	The algorithm allows contraction, conservation, or dilation of space A; pairwise relationships between objects are lost; combinatorial method.	This method, as well as all the other combinatorial methods, are implemented using a simple algorithm.
Hierarchical agglomeration: information analysis	Minimal chaining; only for Q-mode clustering based upon presence-absence of species.	Use is unclear: similarities reflect double absences as well as double presences.
Hierarchical division	Danger of incorrect separation of members of minor clusters near the beginning of clustering.	
Monothetic	Division of the objects following the states of the ''best" descriptor. clustering may depend on different phenomena.	Useful only to split objects into large clusters, inside which
Polythetic	For small number of objects only.	Impossible to compute for sizable data sets.
Division in ordination space	Binary division along each axis of ordination space; no search is done for high concentrations of objects in space A.	Efficient algorithms for large data sets, when a coarse division of the objects is sought.
TWINSPAN	Dichotomized ordination analysis; ecological justincation of several steps unclear.	Produces an ordered two-way table classifying sites and species.

...Fortsetzung umseitig

Methode	pro & contra	Beispiele in der Ökologie
	Fortsetzung Übersicht Clustermethoden	
K-means partitioning `amap`-Paket: `Kmeans(...)`	Minimizes within-group sum of squares; different rules may suggest different optimal numbers of clusters.	Produces a partition of the objects into K groups, K being determined by the user.
Species associations	Non-hierarchical methods; clustering at a pre-selected level of similarity or probability.	Concept of association based on cooccurence of species (for other concepts, use the hierarchical methods).
Non-hierarchical complete linkage	For all measures of dependence among species; species associated by complete linkage (no overlap); sate11ite species joined by sing1e linkage (possible overlap).	Straightforward concept; easy application to any problem of species associaton.
Probabilistic clustering	Theoretically very consistent algorithm ; test of signincance on species, associations; limited to similarities computed using Goodall's probabilistic coefficient.	Not often used because of heavy computation.
Seriation	One-dimensional ordination along the main diagonal of the similarity matrix.	Especially useful for non-symmetric association matrices.
Indicator species		
TWINSPAN	Only for classincations of sites obtained by splitting CA axes; eco1ogical justification of several steps unclear.	Gives indicator values for the pseudospecies.
Indicator value index	For any hierarchical or non-hierarchical classincation of sites; *IndVal* for a species is not affected by the other species in the study.	Gives indicator values for the species under study; the *IndVal* index is tested by permutation.

4.5 Ordinationsmethoden

Ganz gute Tutorien für das Paket `vegan` und auch Ordinationsmethoden im Allgemeinen sind in Oksanen (2008, 2004) enthalten.

Ein allgemeiner Hinweis: wenn mit einer Korrelationsmatrix oder mit standardisierten Speziesdaten gerechnet wird, so bekommen alle Arten die gleich Wichtung. Beim Rechnen mit einer Kovarianzmatrix erhalten abundante Arten mehr Gewicht. Die Vektorpfeile zeigen eine Zunahme der Arten oder Umweltvariablen; in entgegengesetzter Richtung ist eine entsprechende Abnahme zu verzeichnen, s.a.Ordinationstechniken.

4.5.1 PCA - linear, indirekt

In R gibt es viele Möglichkeiten eine PCA zu rechnen z.B.: `dudi.pca(...)` `ade4`-Paket, `princomp(...)` `stats`-Paket, früher `mva` oder mit `rda(...)` `vegan`-Paket. Um Eigenwerte darzustellen bietet Paket `stats` den `screeplot(..)` an.

dudi.pca(...)
ade4

```
library(ade4)  # Paket laden
ostr <- read.table("ostr.csv", sep=";", header=TRUE, row.names=1)  # Daten einlesen
```

☞ darauf achten, daß Reihen und Spalten auch als Variablen eingelesen werden – hier durch `header=TRUE, row.names=1`.

```
dudi.pca(ostr)  # zeichnet Eigenwerte + Auswahl berechneter Achsen
```

...Fortsetzung umseitig

```
ostr.pca <- dudi.pca(ostr)
```

☞ bei den Daten sollten Reihennamen vorliegen: geschieht hier im Beispiel beim Einlesen mit `row.names=1`; in `dudi.pca(...)` sind Zentrierung durch den Mittelwert – `center=T` – sowie Normalisierung der Spalten – `scale=T`, d.h. die Arten bekommen gleiches Gewicht, da sie standardisiert werden; bei nichtstandardisierten Arten fallen die Abundanzen deutlicher ins Gewicht

```
                                    s.arrow - Vektoren
s.arrow(ostr.pca$ci)  # Spalten; nicht normiert -> $ci
s.arrow(ostr.pca$c1)  # Spalten; normiert -> $c1
s.arrow(ostr.pca$li)  # Zeilen; nicht normiert -> $li
s.arrow(ostr.pca$l1)  # Zeilen; normiert -> $l1
```

☞ `s.arrow` - Optionen: `xax=2` Achsenangabe, `clabel` Labelgröße, `pch=16` Punkttyp, `edge=FALSE` Pfeilspitze ausstellen, `xlim=c(-2, 2)` Achsenbereich, `grid=F` Gitterlinien ausschalten, `cgrid=2` Vergrößerung der Angabe für Gitter, `sub="..."` extra Titel, `csub=2` entsprechende Vergrößerung, `pixmap, contour, area` Datenzusätze, `add.plot=T` Plot nur ergänzen, nicht neu zeichnen

```
                                    s.label - Punkte
s.label(ostr.pca$ci)  # Spalten; nicht normiert -> $ci
s.label(ostr.pca$c1)  # Spalten; normiert -> $c1
s.label(ostr.pca$li)  # Zeilen; nicht normiert -> $li
s.label(ostr.pca$l1)  # Zeilen; normiert -> $l1
                              Farbangabe - durch überzeichnen mit points(...)
points(ostr.pca$l1, pch=19, col=c("red", "blue3", "green3")[ostr$ort])  # mit Farbangabe für die
Var. ort
```

☞ `s.label` - Optionen: `xax=2` Achsenangabe, `clabel` Labelgröße, `pch=16` Punkttyp, `cpoint=2` Punktvergrößerung, `neig` neighbouring Graf mit einzeichnen, `cneig=1` Liniendicke, `xlim=c(-2, 2)` Achsenbereich, `grid=F` Gitterlinien ausschalten, `cgrid=2` Vergrößerung der Angabe für Gitter, `sub="..."` extra Titel, `csub=2` entsprechende Vergrößerung, `pixmap, contour, area` Datenzusätze, `add.plot=T` Plot nur ergänzen, nicht neu zeichnen

```
                                    Loadings
barplot(ostr.pca$li[,1])  # Loadings (nicht normiert) der Achse 1 -> [,1] )
barplot(ostr.pca$l1[,1])  # Loadings (normiert) der Achse 1 -> [,1] )
                                Eigenwerte - kumulativer Anteil
cumsum(ostr.pca$eig * 100.0/sum(ostr.pca$eig))
detach(package:ade4)  # package wieder entfernen
```

☞ Farbe läßt sich z.B. durch globales Einschalten der Farben über `par(fg="blue4")` z.B. ändern oder durch Neuzeichnen der Punkte mit `points(...)` s.Bsp. auf Seite 38. Anders geht dies auch, wenn man das Paket `vegan` dazunimmt und mit `biplot` arbeitet:

```
                                ☞ biplot(...) - vegan package
biplot(ostr.pca$li, fora.ca$co)  # direkte grafische Gestaltungsmöglichkeiten
                                PCA mit rda(...) - vegan Paket
data(dune)  # Daten laden
?dune # Hilfe zum Datensatz
dune.pca <- rda(dune)
dune.pca  # Statistik Ausgabe
plot(dune.pca)  # einfacher Plot
plot(dune.pca, type="n", choices=c(1,2), main="PCA \"dune\"Daten")
# kein Plot: "n"; Achsen c(1,2); Titel: main
points(dune.pca, "sites", pch=16, cex=1.5, col="blue")
# Punkte: sites; P-Typ: 16; Vergrößerung: 1.5x, Farbe: blau
text(dune.pca, "sites", cex=0.7, col="white")
# Text: sites weiß hinein
text(dune.pca, "species", col="darkgreen", cex=0.7)
# Text: species; Farbe darkgreen; Verkleinerung 0.7x
legend("bottomright",  # Legende
  legend=c("Pflanzen", "Probestellen"), # Text
  text.col=c("darkgreen", "blue"), # Farben f. Text
  col=c("darkgreen", "blue"), # Farben f. Punkte
  pch=16, # Punkt-Typ
  pt.cex=c(0,1.5) # Punkt-Größe
  )
vegandocs("vegan-FAQ")  # kleine FAQ Doku, ganz hilfreich
```

score(...)
ade4

Grafische Faktorenanalyse Mit Hilfe der Funktion `score(...)` aus dem package `ade4` hat man einen besseren Überblick, wie sich die entsprechenden Proben zu den Arten verhalten.

Bsp. von Objekt 'ostr.pca' auf Seite 107

```
score(ostr.pca)  # grafische Faktorenanalyse
```

☞ es werden für die ersten beiden Achsen alle 'sites'und 'scores'grafisch aufgelistet

`princomp(...)` bietet mehr grafische Eingriffsmöglichkeiten über `biplot(...)` als die Funktion `dudi.pca(...)` aus dem Paket `ade4` auf Seite 107.

princomp(...)
stats

```
library(stats)  # Paket laden
fora <- read.table("foramenifera.csv", sep=";", header=TRUE, row.names=1)  # externe Daten einlesen
names(fora.pca)  # durch fora.pca$... anwählbare Objekte zeigen
```

☞ bei den Daten sollten Reihennamen vorliegen: geschieht hier im Beispiel beim Einlesen mit `row.names=1`; in `princomp(...)` ist die Voreinstellung `cor=FALSE` (keine Kovarianzmatrix) besser ist jedoch auf `cor=T` zu schalten (dies ist das gleiche Ergebnis, wie Voreinstellung in `dudi.pca`

Plotten - Vektoren, Punkte

```
biplot(fora.pca)
```

☞ `biplot` Optionen: `var.axes=F` Vektordarstellung der Spaltendaten ausstellen, `col=c("blue ", "green3")` Farbe einstellen, `cex=c(2,1)` Größe der Labels, `xlabs= c("Art1", "Art2",...)` optionale Labelbezeichnung, `expand=0.7` Vektorenlänge änderbar, `arrow.len=.3` Pfeilspitzen änderbar, `xlim=c(-2, 4)` Achsenskalierung, `main, sub, xlab, ylab` Grafikbeschriftungen

Loadings

```
barplot(fora.pca$loadings[,1])  # für Achse 1
```

Eigenwerte

```
cumsum(fora.pca$sdev^2 * 100.0/sum(fora.pca$sdev^2))
detach(package:stats)  # Paket wieder entfernen
```

☞ ausgegeben wird die Standardabweichung der Komponenten mit `...$sdev`. Um Eigenwerte zu erhalten, müssen diese quadriert werden: `^2`

Anmerkung: `pca` scheint nur in älteren Versionen zu existieren s. `?princomp` und `?prcomp`. Es gibt auch einen `screeplot(..)`, der die Eigenwerte der PCA darstellt.

pca(...)
multiv
bis V. 1.9

```
library(multiv)  # Paket laden für pca(...)
artdaten.pca <- pca(as.matrix(artdaten), method=3)  # pca berechnen
str(artdaten.pca)  # Struktur anzeigen lassen
plot(artdaten.pca$rproj[,1], artdaten.pca$rproj[,2])  # Achse 1 gegen 2 auftragen
cumsum(artdaten.pca$evals*100.0/sum(artdaten.pca$evals))  # % Anteil der Eigenwerte
biplot(artdaten.pca$cproj, artdaten.pca$rproj, cex=c(0.5,0.5), col=c("brown", "darkblue"))
detach(package:multiv)  # Paket wieder entfernen
```

☞ `pca(..., method=3)` muß eigentlich nicht angegeben werden, da dies die Voreinstellung ist; in der Funktion `bipot(...)` werden die Labels verkleinert `cex=c(0.5,0.5)`, sowie braun und dunkelblau gezeichnet.

Tabelle 5: Verschiedene Methoden bei der PCA mit der Funktion `pca(a, method=3)`, aus der R Hilfe

`pca(a, method=3)`	
1	keine Transformation
2	Zentrierung auf Null durch den Mittelwert
3	Zentrierung auf Null durch den Mittelwert sowie Standardisierung
4	Beobachtungen normalisiert durch Intervallgrenzen der Variablen und Varianz/Kovarianzmatrix wird verwendet
5	rechnen mit einer Kendall (rank-order) Korrelationsmatrix
6	rechnen mit einer Spearman (rank-order) Korrelationsmatrix
7	rechnen mit einer Kovarianzmatrix
8	rechnen mit einer Korrelationsmatrix
Hinweis: `method=3` und `method=8` liefern identische Ergebnisse	

4.5.2 RDA - linear, direkt & partiell

rda(...)
vegan

```
library(vegan)  # package laden
data(dune)  # Species Daten einlesen
?dune # Hilfe zum Datensatz
data(dune.env)  # Umweltdaten einlesen
?dune.env # Hilfe zum Datensatz
```

☞ Wenn man selbst Daten einliest: darauf achten, daß Reihen und Spalten auch als Variablen eingelesen werden durch `header=TRUE, row.names=1` in `read.table()` `fora <-read.table("fora.csv",header=TRUE, row.names=1)`.

```
dune.Manure <- rda(dune ~Manure, dune.env)
plot(dune.Manure)  # ansehen
detach(package:vegan)  # Paket wieder entfernen
```

☞ Eine partielle RDA ist auch einfach möglich, da die Funktion `rda(...)` 3 Argumente enthalten kann: `rda(artdaten, umwdaten, umwdaten[,3:5])`. `umwdaten[,3:5]` bedeutet hier, daß die Spalten 3-5 (`'[,3:5]'`) herausgerechnet werden.

☞ Will man mit anderen Distanzmatrizen als mit der Euklid-Distanz rechnen, so kann man die Funktion `capscale(...)` benutzen.

4.5.3 CA - unimodal, indirekt

Eine Korrespondenzanalyse kann man mit `dudi.coa(...)` aus dem Paket `ade4` oder mit `cca(...)` aus dem Paket `vegan` durchführen.

dudi.coa(...)
ade4
cca(...)
vegan

```
CA - ade4
fora <-read.table("fora.csv",header=TRUE, row.names=1)  # Daten laden
```

☞ darauf achten, daß Reihen und Spalten auch als Variablen eingelesen werden – hier durch `header=TRUE, row.names=1`.

```
library(ade4)  # Paket laden
fora.ca <- dudi.coa(fora)  # CA berechnen und zuweisen
Eigenwerte - kumulative Varianz
cumsum(fora.ca$eig*100.0/sum(fora.ca$eig))  # cumulative Varianz
Vektoren und Scores einzeichnen - a.arrow(...), s.label(...)
s.arrow(fora.ca$c1, xlim=c(-5,8), csub=1.5, sub="CA",clabel=1)
s.label(fora.ca$l1, add.plot=TRUE, clabel=0.5)  # Plot dazuzeichnen, Labels kleiner
detach(package:ade4)  # Package wieder entfernen
CA - vegan
```

☞ analog dem Beispiel CCA in Abschnitt 4.5.4 nur ohne Umweltdaten

```
CA - vegan (Analysestatistik)
```

4.5.4 CCA - unimodal, direkt

Eine Kanonische Korrespondenzanalyse (CCA) kann man mit der Funktion `cca` im package `vegan` berechnen lassen. ☞ eine gleiche Funktion gibt es im package `ade4`.

cca(...)
vegan

```
CCA - vegan (Hilfebeispiel)
library(vegan)  # Paket laden
data(dune)  # Beispieldatensatz (Arten - Gräser)
data(dune.env)  # Beispieldatensatz (Umweltdaten)
modell <- cca(dune ~A1 + Moisture + Management, dune.env)  # CCA-Modell
```

☞ falls eine Warnmeldung auftaucht: `Warning message: Some species were removed because they were missing in the data in:...` dann wurden die Spalten entfernt, deren Spaltensumme 0 ist.

```
plot(modell, type="n")  # nichts, nur Plotrahmen zeichnen
text(modell, dis="cn", arrow = 2)  # zentrierte Faktoren aus Umweltdaten
```

☞ dis steht für display: `'sp'` für species scores, `'wa'` für site scores, `'lc'` für linear constraints oder 'LC scores', `'bp'` für biplot Vektoren (der Umw.var.) oder `'cn'` für 'centroids of factor constraints'.

...Fortsetzung umseitig

```
# points(modell, pch=21, col="red", bg="yellow", cex=1.2)  # Artenscores normal
# text(modell, "species", col="blue", cex=0.8)  # Artnamen einzeichnen
```

goodness als Zeichenparameter + Skalierung f. Punktgröße

☞ die Funktion `goodness(...)` summiert die Eigenwerte jeder Art über jeden einzelnen Faktor auf. Siehe auch in der Hilfe: `?goodness.cca`

```
goodness(modell)[,2] -> good ; good.scale <- 4
# Artenscores kombiniert mit goodness(...) mal 4
# goodness über Punktgröße
points(modell, "species", col="blue4", cex=good*good.scale, pch=16)
# goodness über Schriftgröße
   # text(modell, "species", col="blue4", cex=good*good.scale, pch=16)
# goodness-Beschriftungs-Filter:
scores <- scores(modell, choices=c(1,2))  # Achsen 1+2
grenze <- 0.1  # goodness geht von 0...1 also 0%...100%
(which(goodness(modell)[,2] > 0.1) -> filter)  # Spaltenindex für alles was größer 10%
# scores + filter + "offset"
  scores$species[filter,1] + good[filter]*0.2 -> scores.x
  scores$species[filter,2] + good[filter]*0.2 -> scores.y
  rownames(scores$species)[filter] -> namen # Beschriftung
text(scores.x, scores.y, labels=namen, adj=0, col="red")
# automatische Angabe der Grenze für goodness
title(paste("'dune'  - Daten nach 'goodness'  > ", grenze,"  beschriftet"))
```

Legende

```
# Sequenz Länge 6 von min nach max
good.summary <- seq(min(good), max(good), length.out=6)
# Text aus summary ohne 1.Spalte + Runden 1 Stelle
good.legend <- paste(round(good.summary[-1],2)*100,"%")
legend("bottomleft",  # Legende
  title="goodness",
  legend=good.legend,  # Text aus summary ohne 1.Spalte + Runden 1 Stelle + %-Zeichen
  text.col="blue4",  # Farben f. Text
  col="blue4",  # Farben f. Punkte
  pch=16,  # Punkttyp gefüllt s. auf Seite 28
  pt.cex=good.summary[-1]*good.scale,  # Punkt-Größe aus summary ohne 1.Spalte
  bty="n"   # keine Box
)
```

```
# mit Maus platzieren
locator(1) -> wo
legend(wo$x, wo$y,  # Legende
  title="goodness",
  legend=good.legend,  # Text aus summary ohne 1.Spalte + Runden 1 Stelle + %-Zeichen
  text.col="blue4",  # Farben f. Text
  col="blue4",  # Farben f. Punkte
  pch=16,  # Punkt-Typ
  pt.cex=good.summary[-1]*good.scale,  # Punkt-Größe aus summary ohne 1.Spalte
  bty="n"   # keine Box
)
# detach(package:vegan)  # Paket eventuell wieder entfernen
```

CCA - vegan

```
ostr.dca <- read.table("dca-spe.csv", header=TRUE, row.names=1)  # Daten einlesen
ostr.env <- read.table("dca-env.csv", header=TRUE, row.names=1)
```

☞ darauf achten, daß Reihen und Spalten auch als Variablen eingelesen werden – hier durch `header=TRUE, row.names=1`.

```
library(vegan)  # Paket laden
cca.all <- cca(ostr.dca, ostr.env)  # cca berechnen
```

☞ Optionen in `cca`: es können auch direkt Formeln angegeben werden, wie im Beispiel: `cca(varespec ~Al + P*(K + Baresoil), data=varechem)`; man kann natürlich auch mit `cca(ostr.dca, ostr.env[,c(1:2,4)])` gezielte Spalten einbinden, falls die Datenmatrix noch andere Daten enthält, die nicht zur CCA gehören – aber VORSICHT sollen Variablenherausgenommen werde, so ist eine pCCA durchzuführen!

...Fortsetzung umseitig

```
plot(cca.all, main="CCA")  # darstellen oder mit biplot(...)
str(cca.all)  # Struktur des Datenobjektes
biplot(cca.all$CCA$u.eig, cca.all$CCA$v.eig, cex=c(0.7,0.7), col=c("brown", "green4"))
```
☞ die Funktionen `plot.cca, text.cca, points.cca, scores.cca` bieten grafische Gestaltungsmöglichkeiten

Eigenwerte

```
names(cca.all$CCA)  # durch "...$" abfragbare Objekte anzeigen
summary(cca.all)  # Summary mit total, un- und constrained eigenvalues
cca.all$CCA$eig  # zeigt die constrained Eigenwerte an
summary(cca.all)$ev.con  # ergibt dasselbe
summary(cca.all)$ev.uncon  # zeigt die unconstrained Eigenwerte an
detach(package:vegan)  # Paket wieder entfernen
```

4.5.5 pCCA - unimodal, direkt, partiell

cca(...)
vegan

Eine partielle Korrespondenzanalyse, bei der gezielt Variablen herausgerechnet werden läßt sich ebenfalls mit der Funktion `cca(...)` aus dem package `vegan` berechnen

pCCA - vegan

```
ostr.dca <- read.table("dca-spe.csv", header=TRUE, row.names=1)  # Daten einlesen
ostr.env <- read.table("dca-env.csv", header=TRUE, row.names=1)
```
☞ darauf achten, daß Reihen und Spalten auch als Variablen eingelesen werden – hier durch `header=TRUE, row.names=1`.

```
library(vegan)  # Paket laden
cca.all.ohne.pH <- cca(ostr.dca, ostr.env, ostr.env[,2])  # cca berechnen
```
☞ als 3. Argument gibt man die Daten an, die herausgerechnet werden; Optionen in `cca`: es können auch direkt Formeln angegeben werden, wie im Beispiel: `cca(varespec ~Al + P*(K + Baresoil), data=varechem)`

```
plot(cca.all.ohne.pH, main="pCCA")  # darstellen
```
☞ die Funktionen `plot.cca, text.cca, points.cca, scores.cca` bieten grafische Gestaltungsmöglichkeiten

Eigenwerte

```
names(summary(cca.all.ohne.pH))  # durch "...$"abfragbare Objekte anzeigen
summary(cca.all.ohne.pH)  # Summary mit total, un- und constrained un constrained out Eigenvalues
summary(cca.all.ohne.pH)$ev.con  # zeigt die constrained Eigenwerte an
summary(cca.all.ohne.pH)$ev.uncon  # zeigt die unconstrained Eigenwerte an
detach(package:vegan)  # Paket wieder entfernen
```

4.5.6 DCA - detrended, unimodal, indirekt (auch Decorana)

Eine detrended CA (s. arch effect) läßt sich mit der Funktion `decorana(...)` aus dem package `vegan` durchführen.

```
ostr <- read.table("ostr.csv",header=TRUE, row.names=1  # Daten einlesen
```
☞ darauf achten, daß Reihen und Spalten auch als Variablen eingelesen werden – hier durch `header=TRUE, row.names=1`.

```
library(vegan)  # Paket laden
ostr.dca <- decorana(ostr)  # DCA berechnen
ostr.dca  # summary anschauen
plot(ostr.dca, cols=c("brown","blue4"))
```

Loadings

```
scores(ostr.dca, display=c("species"))  # Loadings der Arten
scores(ostr.dca, display=c("sites"))  # Loadings der Probestellen
detach(package:vegan)  # package wieder entfernen
```

4.5.7 „d“CCA - „detrended“, unimodal, direkt

Um eine „detrended“ CCA durchzuführen kann man die Funktion `downweight(...)` (`vegan` - package) verwenden. Die CCA selbst wird wie üblich durchgeführt (s. auf Seite 110)

```
ostr <- read.table("ostr.csv",header=TRUE, row.names=1  # Daten einlesen
```
☞ darauf achten, daß Reihen und Spalten auch als Variablen eingelesen werden – hier durch `header=TRUE, row.names=1`.
```
library(vegan)  # Paket laden
dn.ostr <- downweight(ostr, fraction=5)  # seltene Arten abwichten
detach(package:vegan)  # Paket wieder entfernen
```
☞ `fraction=5` bedeutet, daß alle Arten, die in geringerer Zahl als 5 vorkommen, abgewichtet werden.

4.5.8 3D - Ordinationsgrafiken

Im Paket `vegan` gibt es die Funktion `ordiplot3d(...)`, mit der man 3D-Diagramme erzeugen kann. Sie braucht aber die Pakete `scatterplot3d` und `rlg`.

ordiplot3d - vegan
```
library(rgl,scatterplot3d,vegan)  # Pakete Laden
data(dune);data(dune.env)  # Beispieldatensätze
par(mfrow=c(1,2))  # Grafik: |1|2|
  ord <- cca(dune ~  A1 + Moisture, dune.env)  # CCA-Modell
  ordiplot3d(ord)  # normaler Plot
  pl <- ordiplot3d(ord, angle=15, type="n")
  points(pl, "points", pch=16, col="red", cex = 0.7)  # Punkte zeichnen
  identify(pl, "arrows", col="blue")  # bessere Positionierung
  text(pl, "arrows", col="blue", pos=3)
  text(pl, "centroids", col="blue", pos=1, cex = 1.2)
par(mfrow=c(1,1))  # Grafik wieder |1|
?dune # Hilfe zum Datensatz
detach(package:vegan)  # Paket eventuell entfernen
detach(package:rgl)  # Paket eventuell entfernen
detach(package:scatterplot3d) # Paket eventuell entfernen
```

4.5.9 Teststatistiken Ordination

Damit man weiß wie gut eine Ordination ist, dafür werden im Paket `vegan` verschiedene diagnostische Funktionen bereitgestellt:

`bioenv(...)` ..... maximale Umwelt-Arten-Korrelation
`goodness.cca(...)` ..... Eigenwert Statistik für Arten oder Proben
`inertcomp(...)` ..... Inertia composition
`vif.cca` ..... variance inflation[24]faktor

goodness.cca(...) - vegan (Analysestatistik)
```
library(vegan)  # Paket laden
data(dune)  # Artdaten
data(dune.env)  # Umweltdaten
mod <- cca(dune ~A1 + Management + Condition(Moisture), data=dune.env)  # CCA-Modell
goodness(mod)  # Tabelle aller Eigenwerte f. jeden Faktoren; aufsummiert
goodness(mod, summ = TRUE)  # Gesamtsumme der Eigenwerte (hier: der Arten)
```
inertcomp(...) - vegan (Analysestatistik)
```
 ⋮
inertcomp(mod, prop = TRUE)  # Eigenwertanteil (hier: der Arten) an pCCA, CCA, CA
```
☞ für pCCA: die Umweltfaktoren wurden herausgerechnet
```
inertcomp(mod, stat="d")  # Distanzstatistik
```
☞ die Distanzstatistik gibt die Distanz der Art/Probe zum Zentrum an, aber für alle Faktoren, und damit auch Dimensionen/Achsen aufsummiert

...Fortsetzung umseitig

[24] von engl.: aufgeblasen

vif.cca(...) - vegan (Analysestatistik)

```
 ⋮
vif.cca(mod)  # variance inflation[25] factor
```

☞ dieser „variance inflation factor“ ist ein diagnostisches Mittel, um „unnötige“ Umweltfaktoren zu entdecken. Faustregel: Werte größer 10 zeigen Redundanz des (Umwelt)Faktors an.

```
mod <- cca(dune ~., dune.env)  # Modell mit allen Umweltvariablen
mod
vif.cca(mod)
```

alias(...) - vegan (Analysestatistik)

```
# Aliased[26] constraints - Abhängigkeiten
 ⋮
alias(mod)  # findet Abhängigkeiten der Faktoren untereinander
with(dune.env, table(Management, Manure))  # gibt Tabelle von 'Management' und 'Manure' aus
```

☞ `with(object, function)` läßt sich dazu verwenden, das man z.B. einen Datensatz mit einer oder mehreren Funktionen durchlaufen läßt.

bioenv(...) - vegan

```
# nur um zu zeigen wie es geht:
chiro <- read.csv2("gen2sam.csv", header=TRUE, row.names=1)  # Daten einlesen
chiro.env <- read.csv2("gen2sam.env.csv", header=TRUE, row.names=1, dec=".")  # Daten einlesen
bioenv(chiro, chiro.env)  # bestes Modell: 3 Variablen Korrelation: 0.3996798
summary(bioenv(chiro, chiro.env))
                                        size correlation
    d_H2O_max                              1      0.2585
    d_secci DO_.                           2      0.3913
    d_secci cond DO_.                      3      0.3997 <- max Korrelation
    d_H2O_max d_secci cond DO_.            4      0.3693
    elevation d_secci cond pH DO_.         5      0.3384
    elevation d_H2O_max d_secci cond pH DO_.    6      0.3124
```

☞ `bioenv()` erlaubt maximal 6 Variablen zu vergleichen. Dabei werden immer diejenigen ausgewählt, die zusammen die größte Korrelation aufweisen.

Permutaionstests für constrained Ordinationen [27] lassen sich mit `anova.cca(...)` aus dem Paket `vegan` durchführen.

anova.cca(...) - vegan

```
data(varespec)  # Daten laden
?varespec # Hilfe zum Datensatz
data(varechem)  # Daten laden
?varechem # Hilfe zum Datensatz
vare.cca <- cca(varespec ~Al + P + K, varechem)  # CCA-Modell
anova(vare.cca) permutest.cca(vare.cca)  # Test for adding variable N to the previous model:
anova(cca(varespec ~N + Condition(Al + P + K), varechem), step=40)
```

☞ mit `Condition(...)` wird eine Untergruppe erzeugt, mit `step` die Anzahl der Permutationen. Ob man `anova.cca(...)` oder `permutest.cca(...)` verwendet, macht nur den Unterschied, daß die Ausgabe anders aussieht und das Interface anders ist. `anova.cca(...)` gibt die Argumente an `permutest.cca(...)` weiter.

4.5.10 Vorhersagen für unimodale Ordination

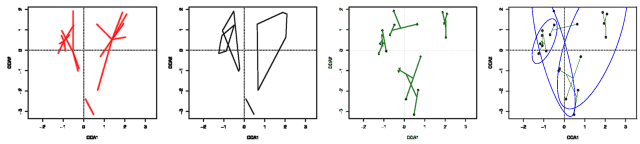
Derzeit nur kurz hingewiesen sei auf die Funktionen in `predict.cca(...)` aus dem package `vegan`.

[25] von engl.: aufgeblasen
[26] engl.: alias Deckname auch Falschname (jur.)
[27] CCA, CA, RDA, PCA

4.5.11 Grafische Zusätze in Ordinationsdiagrammen – vegan

Das Paket **vegan**bietet mit den Funktionen `ordihull(...)`, `ordispider(...)` und diversen anderen eine Auswahl zusätzliche Informationen, wie Cluster, Gruppenzentriod... mit einzuzeichnen.


ordihull(...)
vegan



```
library(vegan)  # Paket laden
data(dune); data(dune.env)  # Daten aus R laden
mod <- cca(dune ~Moisture, dune.env)  # CCA rechnen
attach(dune.env)  # 'dune.env'in den R-Suchpfad aufnehmen
plot(mod, type="n")  # nur passenden Diagrammrahmen zeichnen
```

Umrandung der Untergruppen - ordihull(...)

```
ordihull(mod, Moisture)
```

☞ `Moisture` gibt den zu zeichnenden Faktor an; Optionen: `display = "sites"` oder `display = "species"` möglich.

Schwerpunkt jeder Gruppe (Zentroid) - ordispider(...)

```
ordispider(mod, Moisture ,col="red")
```

☞ `Moisture` gibt den zu zeichnenden Faktor an; Optionen: `display = "sites"` oder `display = "species"` möglich.

Clustergruppen - ordicluster(...)

```
plot(mod, type = "p", display="sites")  # Punkte einzeichnen
ordicluster(mod, hclust(vegdist(dune)), prune=3, col = "blue")
```

☞ Optionen: `prune=3` – `prune=0` alle Gruppen verbinden, `prune=1` Trennung in der ersten Hierarchie (also 2 Gruppen), `prune=3` Trennung in der 3. Hierarchie (also mindestens 3 Gruppen); hierarchische Clusteranalysen mit package `hclust` und `agnes` möglich (s.S.)

Verteilung als Ellipse zeichnen - ordiellipse(...)

```
ordiellipse(mod, Moisture, kind="sd", conf=0.95, lwd=2, col="blue")
```

☞ `Moisture` gibt den zu zeichnenden Faktor an; Optionen: `kind="sd"` – `"sd"` Standardabweichung der Punkte oder Standardfehler wird verwendet; `conf=0.95` Vertrauensintervall manuell festlegen.

```
detach(package:vegan)  # Paket wieder entfernen
```

Randbeschriftungen von Arten/Proben in einem Ordinationsdiagramm, die recht übersichtlich aussieht geht mit `linestack(...)`:

```
data(dune)
?dune # Hilfe zum Datensatz
ord <- decorana(dune)  # Daten laden
linestack(scores(ord, choices=1, display="sp"))  # Arten anzeigen
linestack(scores(ord, choices=1, display="si"), label="left", add=TRUE)  # sites anzeigen, aber
links
title(main="DCA axis 1")  # Titelei
```

oder neben Diagramm:

```
par(mar=c(6,4, 4,6)+0.1)  # Rand rechts erweitern
plot(ord)
linestack(scores(ord, choices=2, display="sp"), at=3.5, add=T)  # "Beschriftung"
title(main="DCA axis 1")  # Titelei
par(mar=c(5,4, 4,2)+0.1)  # Rand zurücksetzten
```

4.5.12 MMDS - Multidimensionale Metrische Skalierung

Die Multidimensionale Metrische Skalierung (MMDS) kann man im Paket `mva` oder `stats` - v2.0 rechnen .

```
library(stats)
data(eurodist)  # Daten laden
?eurodist # Hilfe zum Datensatz
cmdscale( eurodist, k=10 , eig =T)$GOF
☞ GOF= 0.9167991 0.9985792  # goodness of fit
cmdscale(eurodist, k=2 , eig =T)$GOF
☞ GOF= 0.7968344 0.8679134  # goodness of fit
detach(package:stats)
```

4.5.13 NMDS - Nichtmetrische Multidimensionale Metrische Skalierung

Für nicht-metrische Daten: die klassische nach Kruskall `isoMDS(..)` aus dem MASS.

```
#######  klassisch nach Kruskall
library(MASS) # Paket laden
  data(swiss) # Daten zur Fertilität[28]
  ?swiss # Hilfe zum Datensatz
  swiss.x <- as.matrix(swiss[, -1]) # Fertilität als Matrix
  swiss.dist <- dist(swiss.x) # Euklid-Distanz berechnen
  swiss.mds <- isoMDS(swiss.dist)
  plot(swiss.mds$points, type = "n") # type = "n" zeichnet ohne Werte
    text(swiss.mds$points, labels = as.character(1:nrow(swiss.x))) # text dazu
  swiss.sh <- Shepard(swiss.dist, swiss.mds$points) # Shepard Diagramm
  plot(swiss.sh, pch = ".") # Punkttypen '.'  auf Seite 28
  lines(swiss.sh$x, swiss.sh$yf, type = "S") # als Linie verbinden
########  optimale NMDS
data(dune) # Daten laden
library(MASS) # isoMDS
sol <- metaMDS(dune) # versucht Optimum: Datentransformation + stabile Lösung
sol # Ergebnis
plot(sol, type="t") # voreingestellte Grafik mit Text
########  NMDS ↔ Vgl. abiotischer Variablen
data(varespec) # Flechtendatensatz
data(varechem) # dazu gehörige abiotische Daten
library(vegan) # Paket laden
library(MASS)
vare.dist <- vegdist(varespec) # Bray-Curtis Distanzmaß
vare.mds <- isoMDS(vare.dist)  # NMDS
attach(varespec) # in den Suchpfad aufnehmen
attach(varechem)
# Bodenbedeckung vs. MMDS
ordisurf(vare.mds, Baresoil, xlab="Dim1", ylab="Dim2")
# Totale Bedeckung der Reentierflechten
ordisurf(vare.mds, Cla.ste+Cla.arb+Cla.ran, xlab="Dim1", ylab="Dim2")
```

4.6 Zeitreihen - time series

plot.ts(...)
pastecs

Das Paket `pastecs` bietet eine Vielzahl nützlicher Funktionen. Einfache Funktionen sind auch im Standard-Paket `stats` enthalten.

```
                                  Zeitreihen plotten - plot.ts(...)
library("stats")  # package laden
data("LakeHuron")  # Beispiel-Datensatz
```

...Fortsetzung umseitig

[28] standardisierter Datensatz zur Fertilität von je 47 französischsprachigen Schweizer Provinzen (1888), aufgenommene Parameter: `Fertility, Agriculture, Examination, Education, Catholic, Infant Mortality`.

```
?LakeHuron # Hilfe zum Datensatz
plot.ts(LakeHuron)  # plotten
detach(package:stats)  # package entfernen
# auch mal probieren:
par(ask=TRUE)  # vorm Neuzeichnen nachfragen
example(plot.ts)  # Beispiele
library(pastecs)  # package laden
data(marbio)  # Zooplanktondaten
?marbio # Hilfe zum Datensatz
plot.ts(marbio[,1:10])  # mehr als 10 untereinander geht nicht
detach(package:pastecs)  # package entfernen
rm(marbio)  # Datensatz 'marbio' entfernen
                                    gehören die Daten zu einer Zeitreihe?
library(pastecs)
tser <- ts(sin((1:100)/6*pi)+rnorm(100, sd=0.5), start=c(1998, 4), frequency=12)
is.tseries(tser)  # TRUE
not.a.ts <- c(1,2,3)
is.tseries(not.a.ts)  # FALSE
detach(package:pastecs)  # package entfernen
```

4.6.1 Umkehrpunkte

Mit dem Paket `pastecs` und der Funktion `turnpoints(...)` lassen sich Berechnungen zu den Umkehrpunkten[29] durchführen.

```
                                    turnpoints(...) - pastecs
library(pastecs)  # Paket laden
data(marbio)  # Zooplanktondaten
?marbio # Hilfe zum Datensatz
plot(marbio[, "Nauplii"], type="l")  # als Linie darstellen ("l")
# turning points von "Nauplii"berechnen
Nauplii.tp <- turnpoints(marbio[, "Nauplii"])
summary(Nauplii.tp)  # peak=max, pit=min
```

☞ es wird auch Kendalls Informations-Kriterium berechnet: je höher der Wert, desto eher ist es ein Umkehrpunkt

```
plot(Nauplii.tp)  # Informations-Kriterium anzeigen
```

☞ es wird ein Plot für das Informations-Kriterium ausgegeben und eine Linie eingezeichnet über welcher alle Umkehrpunkte liegen, die mit einer Irrtumswahrscheinlichkeit von $\alpha = 0.05$ wirkliche Umkehrpunkte darstellen.

```
# Orginaldaten + Median von Min + Max einschließen (envelope)
plot(marbio[, "Nauplii"], type="l")  # Orginaldaten
lines(Nauplii.tp) # Median + Min + Max dazuzeichnen
title("Orginaldaten, einschließlich Maxi., Mini. und Median Linie")  # extra Titel
detach(package:pastecs)  # Paket wieder loswerden
```

Die Funktion `turnogram(...)` (`pastecs`) zeichnet eine Linie der Umkehrpunkte für ± regelmäßige Zeitreihendaten so, daß die Beziehung 'wenig Proben'↔ 'viel Information'statistisch optimiert wird. Dabei wird mit einer Zufallsverteilung verglichen (näheres s. `?turnogram`)

```
                                    turnogram(...) - pastecs
library(pastecs)
data(bnr)  # Daten marines Zoobenthos
?bnr # Hilfe zum Datensatz
# Serie 4 als Zeitreihe (angenommen sie sei regelmäßig)
bnr4 <- as.ts(bnr[, 4])  # neues time series Objekt
plot(bnr4, type="l", main="bnr4: Org. Daten", xlab="Time")
```

...Fortsetzung umseitig

[29] engl.: peak, pits – Maximum (Peak), Minimum

```
bnr4.turno <- turnogram(bnr4)  # einfaches turnogram berechnen
summary(bnr4.turno)  # Optimum bei Intervall 3
turnogram(bnr4, complete=TRUE)  # komplettes turnogram: Interval 3 -> guter Wert
bnr4.interv3 <- extract(bnr4.turno)  # Daten mit max. Info (Intervall=3) extrahieren
plot(bnr4, type="l", lty=2, xlab="Time")
lines(bnr4.interv3, col=2)  # Linie dazu
title("Original bnr4 (...) vs. max. Infokurve (___)")
bnr4.turno$level <- 6  # anderes Bsp., Intervall=6
bnr4.interv6 <- extract(bnr4.turno)
# beides zusammen plotten
plot(bnr4, type="l", lty=2, xlab="Time")
lines(bnr4.interv3, col=2)  # Linie rot
lines(bnr4.interv6, col=3)  # Linie grün
legend(70, 580, c("original", "interval=3", "interval=6"), col=1:3, lty=c(2, 1, 1))  # Legende
```

4.6.2 Epochen bestimmen

Das Paket `strucchange` bietet die Möglichkeit eine Zeitreihe in Epochen einzuteilen anhand des BIC Kriteriums. Ein Tip noch: man kann auch eine Clusteranalyse über die Daten laufen lassen und die Punkte dann farblich oder punkttypmäßig unterschiedlich darstellen mit dem Beispiel einer Hierarchischen Clusteranalyse auf Seite 93

```
                    Anzahl der Epochen bestimmen breakdates(...)
library(strucchange)  # Paket laden
data(Nile)  # Datensatz laden
?Nile # Hilfe zum Datensatz
plot(Nile)  # ansehen
bp.nile <- breakpoints(Nile ~1)  # berechnen wieviele breakpoints
summary(bp.nile)
plot(bp.nile)  # anzeigen
breakdates(bp.nile)  # Datum anzeigen
ci.nile <- confint(bp.nile)  # Konfidenzintervalle
breakdates(ci.nile)
ci.nile
plot(Nile)
lines(ci.nile)  # mit Konfidenzintervall zeichnen
```

4.6.3 Daten sortieren

Daten von Zeitreihen lassen sich mit der Funktion `abund(...)` aus dem Paket `pastecs` sortieren. Dabei wird ein Wert `f` verwendet, um beim Sortieren eine unterschiedliche Wichtung zu geben:

	`0...`	`...f...`	`...1`
Wichtung	keine Nullwerte	...	hohe Abundanz

Am Ende des folgenden Beispiels von einem marin-benthischen Datensatz `bnr` ist eine Funktion `bnr.f.value(...)` gegeben, mit der man sehen kann wie sortiert wird (zum Kopieren):

```
                              abund(..., f=0.2) - pastecs
library(pastecs)  # Paket laden
data(bnr)  # Daten laden
?bnr # Hilfe zum Datensatz
bnr.abd <- abund(bnr)  # Abundanzen sortieren mit f=0.2
summary(bnr.abd)  # Zusammenfassung
plot(bnr.abd, dpos=c(105, 100))  # sortierte Daten plotten
```

...Fortsetzung umseitig

```
bnr.abd$n <- 26  # li von der 26.Variable sind Nichtnull-Daten
# zum Punkte identifizieren: bnr.abd$n <- identify(bnr.abd)
lines(bnr.abd)  # Trennlinie per 'Hand' einzeichnen
bnr2 <- extract(bnr.abd)  # entsprechende Variablen extrahieren
names(bnr2)  # ... und anzeigen
detach(package:pastecs)  # Paket wieder loswerden
                          Funktion 'bnr.f.value'- Wichtung der Sortierung
library(pastecs)  # Paket laden
data(bnr); attach(bnr)  # Daten laden und in Suchpfad aufnehmen
?bnr # Hilfe zum Datensatz
bnr.f.value <- function(f=0.2)  # Funktion mit Voreinstellung f=0.2
{
bnr.abd <- abund(bnr, f=f)  # Abundanz sortieren je nach f-value
par(mfrow=c(3,3))  # Grafik 3x3
for(i in 1:9)  # 1, 2, ...9 Schritte definieren
    {
       if(i>1) # falls i>1, dann Sortierspektrum durch 9 dazuaddieren
       i=i * round(length(summary(bnr.abd)$sort)/9, 0)
       bnr.name <- names(summary(bnr.abd)$sort[i])  # Name in eine Variable schreiben
       plot.ts(rbind(0,bnr[bnr.name],0), ylim=c(0,500), type="n")  # Objekt ohne Linie
       polygon(rbind(0,bnr[bnr.name],0), col="grey")  # Hervorhebung 'polygon(...)'
    }
par(mfrow=c(1,1))  # Grafik wieder 1x1 setzen
title(paste("f-value:",f,"(Sortierung von lo nach ru)")) # Titelei
}
bnr.f.value(0.9)  # Bsp. mit f=0.9
```

4.6.4 Daten zeitlich versetzten

Siehe auf Seite 22.

4.7 Morphometrie/Landmarks

shapes

Das Paket `shapes` bietet Möglichkeiten der Formanalyse von Landmarks. Siehe auch unter der Webseite von Ian L. Dryden:
http://www.maths.nott.ac.uk/personal/ild/shapes/quick-tutorial.txt Siehe auch Benutzerfunktionen auf Seite 185 für Umrißanalyse/Outlines mittels (normalisierter) elliptischer Fourier Analyse nach Kuhl und Giardina (1982).

Eine gutes und auch umfangreiches Buch zu Morphometrie und R ist in *Morphometrics with R* von Claude (2008) finden.

```
                                        Landmarks darstellen
require(shapes) # Paket laden, falls da
k <- 8  # Anzahl Punkte
m <- 2  # Anzahl Dimensionen
# Datei ist x1 y1 z1   x2 y2 z2   x3 y3 z3 ... xk yk zk
gorf <- read.in("http://www.maths.nott.ac.uk/personal/ild/bookdata/gorf.dat",k,m)
gorm <- read.in("http://www.maths.nott.ac.uk/personal/ild/bookdata/gorm.dat",k,m)
plotshapes(gorf,gorm,joinline=c(1,6,7,8,2,3,4,5,1))  # Punkte verbinden
```

☞ Die Daten lassen sich auch mit `data(gorf.dat)` einlesen; hier nur um zu verdeutlichen wie was vorsichgeht

...Fortsetzung umseitig

Formenvergleich - testmeanshapes(...)

```
# 2D Bsp: weib. und männ. Gorillas
# Hotelling's T^2 Test
test1 <- testmeanshapes(gorf, gorm)
# Hotelling's T^2 test: Test statistic = 26.47
# p-value = 0 Degrees of freedom = 12 46
```

☞ Nullhypothese H_0 ist Verteilungen sind gleich, also es gilt die Alternativhypothese: die Formen sind verschieden

```
# Goodall's isotropic test
test2 <- testmeanshapes(gorf, gorm, Hotelling=FALSE)
# Goodall's F test: Test statistic = 22.29
# p-value = 0 Degrees of freedom = 12 684
```

Bootstrap Test

```
# 2D example : female and male Gorillas (cf. Dryden und Mardia 1998)
data(gorf.dat)  # female
data(gorm.dat)  # male
# just select 3 landmarks and the first 10 observations in each group
select <- c(1,2,3)
A <- gorf.dat[select,,1:10]
B <- gorm.dat[select,,1:10]
resampletest(A,B,resamples=100)
 $lambda
 [1] 33.29139
 $lambda.pvalue
 [1] 0.00990099
 $lambda.table.pvalue
 [1] 5.900204e-08
 $H
 [1] 12.50680
 $H.pvalue
 [1] 0.00990099
 $H.table.pvalue
 [1] 0.0004570905
 $J
 [1] 26.48498
 $J.pvalue
 [1] 0.00990099
 $J.table.pvalue
 [1] 0
 $G
 [1] 14.99155
 $G.pvalue
 [1] 0.00990099
 $G.table.pvalue
 [1] 1.83508e-05
```

☞ Amaral, Dryden und Wood 2007, Discussion and Conclusions: The pivotal bootstrap test based on λ_{min} generally outperforms the other bootstrap tests in the full range of examples considered and is the recommended test.

Thin-plate spline transformation grids

```
# TPS Gitter mit 2facher Überhöhung (2x)
gorf.proc <- procGPA(gorf) #  Erstellung s. Prokrustes-Test
gorm.proc <- procGPA(gorm)
mag <- 2  # Vergrößerung
TT <- gorf.proc$mshape  # mittlere Form
YY <- gorm.proc$mshape
par(mfrow=c(1,2))  # Grafik 2 Spalten
YY <- TT + (YY - TT) * mag
tpsgrid(TT, YY, -0.6, -0.6, 1.2, 2, 0.1, 22)
title("TPS Gitter - Mittelvergleich\n weiblich (links) zu Männchen (rechts)")
```

...Fortsetzung umseitig

☞ in `tpsgrid(TT, YY, xbegin, ybegin, xwidth, opt, ext, ngrid)` bedeuten `TT` erstes Objekt (Daten-Quelle): (k x 2 Matrix), `YY` zweites Objekt (Daten-Ziel): (k x 2 Matrix), `xbegin` kleinster x-Wert für Grafik, `ybegin` kleinster y-Wert, `xwidth` Grafikbreite, `opt` Option 1: (nur das deformierte Gitter von `YY` wird angezeigt), Option 2: beide Gitter, `ext`[30] Wert um wieviel das Gitter über das Objekt hinausragt, `ngrid` Anzahl der Gitterpunkte: Größe ist `ngrid * (ngrid -1)`

PCA

```
# 2D example : female and male Gorillas (cf. Dryden und Mardia 1998)
data(gorf.dat)  # Daten female
data(gorm.dat)  # Daten male
gorf <- procGPA(gorf.dat)  # Prokrustanalyse
gorm <- procGPA(gorm.dat)  # Prokrustanalyse
shapepca(gorf,type="r",mag=3)  # Rohdaten
mtext("mittlere Form + mag=3 Standardabweichungen",
  side=3,   # Position u, li, o, re = 1, 2, 3, 4
  cex=0.8,  # character expansion
  line=2,   # 2 Zeilen außerhalb
  col="darkred",
  xpd=NA    # darf auch außerhalb zeichnen
)
shapepca(gorf,type="v",mag=3)  # mit Änderungsvektoren
mtext("mittlere Form + mag=3 Standardabweichungen\nals Vektoren",
  side=3,   # Position u, li, o, re = 1, 2, 3, 4
  cex=0.8,  # character expansion
  line=2,   # 2 Zeilen außerhalb
  col="darkred",
  xpd=NA    # darf auch außerhalb zeichnen
)
```

Durchschnittsform nach Bookstein (1986) - bookstein2d(..)

```
  # 2D example : female and male Gorillas (cf. Dryden und Mardia 1998)
data(gorf.dat)  # Daten female
data(gorm.dat)  # Daten male
  bookf <- bookstein2d(gorf.dat)
  bookm <- bookstein2d(gorm.dat)
plotshapes(bookf$mshape, bookm$mshape, joinline=c(1,6,7,8,2,3,4,5,1))
```

Clusteranalyse - Daten farbabhängig

```
# 2D example : female and male Gorillas (cf. Dryden und Mardia 1998)
data(gorf.dat)  # Datensatz laden
str(gorf.dat)   # Datenstruktur
 num [1:8, 1:2, 1:30] 5 53 0 0 -2 18 72 92 193 -27 ...
# 30 Individuen
anzDaten <- 30  # Anzahl der Individuen (Stellschraube)
(rdist <- matrix(1,anzDaten,anzDaten))  # 30x30 Matrix als Grundlage
for(zeilen in 1:anzDaten) {
  for(spalten in 1:anzDaten)
    rdist[zeilen,spalten] <- riemdist(gorf.dat[,spalten],gorf.dat[,zeilen])
}
rdist   # Matrix ausgeben
(rdist <- as.dist(rdist))        # Distanzmatrix daraus machen
cluster <- hclust(rdist,"complete")
par(no.readonly=TRUE) -> paralt  # Grafikeinstellungen speichern
par(las=1)      # Ausrichtung Achsenlabels
plot(cluster,hang=-1,
  sub=paste("Methode:",cluster$method),
  xlab="30 Individuen",
  ylab="Riemann Gruppen-Distanz (rho)",
  main="Gorilla Schädeldaten")
axis(2)         # y-Achse noch dazu
```

...Fortsetzung umseitig

[30] von engl.: extend

```
gruppen <- 2   # Anzahl der Gruppen (Stellschraube)
gruppenzugehoerigkeit <- cutree(cluster, k=gruppen)
(farbe <- rainbow(gruppen,s=0.5,v=0.9)[gruppenzugehoerigkeit])
gorf.dat.proc <- procGPA(gorf.dat)  # Prokrustanalyse s. Prokrustes-Test
# etwas unübersichtlich aber zeigt Prinzip
plot(
  x=gorf.dat.proc$rotated[,1,1:30],
  y=gorf.dat.proc$rotated[,2,1:30],
  type="n",    # keine Grafik, nur Proportionen
  main=paste("Gorilla Schädeldaten nach Cluster: ",
    cluster$method,"\n",
    gruppen,"-Gruppen",sep=""   # Titel zs.fügen, kein Leerzeichen
  )
)
matpoints(
  x=gorf.dat.proc$rotated[,1,1:30],
  y=gorf.dat.proc$rotated[,2,1:30],
  pch=(1:anzDaten)[gruppenzugehoerigkeit],
  col=farbe
)
PktVerbindung <- c(1,5,4,3,2,8,7,6,1)
matlines(
  x=gorf.dat.proc$rotated[PktVerbindung,1,1:30],
  y=gorf.dat.proc$rotated[PktVerbindung,2,1:30],
  col=farbe,
  type="c",    # 'c'  = für Punkte Platz lassen
  lty="dashed"
)
par(paralt)    # Grafikeinstellungen wieder zurück
```

4.8 Paläo – Rekonstruktionen

Hierzu gibt es die Pakete `analogue`, `palaeo`, `rioja` (Analysis of Quaternary Science Data). Ein ebenfalls interessanter Ansatz mit neuronalen Netzwerken findet sich bei Racca et al. (2007).

⚠ Im `analogue`-📦 (v0.6-22) scheinen die Residuengrafiken − und + zu vertauschen, wenn man die Ergebnisse mit dem `rioja`-📦 (v0.5.6) vergleicht. Das Windows-Programm C^2 (1.6.5 Build 1, 22/03/2010)[31] berechnet ebenfalls verdrehte Residuen.

4.8.1 Modern analogue technique – MAT

Hierzu eignet sich das `analogue`-📦. Eine alternative Funktion bietet das Paket `rioja` von Steve Juggins Funktion `?MAT`.

```
require(analogue) # Zusatz Paket laden
data(swapdiat) # Diatomeen Surface Waters Acidifcation Project (SWAP)
data(swappH) # pH Daten
swap.mat <- mat(swapdiat, swappH, method = "bray", weighted=TRUE)
summary(swap.mat)
# ... 10 erste 'analogues' ausgegeben
fitted(swap.mat) # ausgerechnete Modellwerte
resid(swap.mat) # Residuen
par(mfrow = c(2,2)) # Grafik 2x2:
  # | 1 | 2 |
  # | 3 | 4 |
```

[31] Siehe http://www.staff.ncl.ac.uk/staff/stephen.juggins/software/C2Home.htm

```
  # plot(swap.mat) # siehe zeichnet alle 4 auf einmal; siehe ?plot.mat
  plot(swap.mat, which = 1) # gemessen vs Modellwerte
  plot(swap.mat, which = 2) # gemessen vs Residuen s. auch ?panel.smooth
  # LLSESP linear least squares error slope parameter siehe Vasko et al. (2000)
    abline(
      lm(resid(swap.mat)$residuals ~ swappH) -> swap.residuals.lm,
      lty="longdash", col="green"
    )
    legend("topright",
      legend=c("Avg. bias","Max. bias", "LLSESP"),
      lty= c("dashed", "solid", "longdash"),
      col= c("blue", "blue", "green"),
      cex=0.8
    )
    legend("bottomleft",
      legend=sprintf("LLSESP: %1$+.3f",coef(swap.residuals.lm)[2]), # Anstieg
      bty="n", # ohne Box
      cex=0.8
    )
  plot(swap.mat, which = 3) # k-Analogues vs. RMSEP
    # optimales k
    abline(v=getK(swap.mat)[1], col="gray", lty="dotted")
  # plot(swap.mat, which = 4) # average bias
  plot(swap.mat, which = 5) # maximum bias
    abline(v=getK(swap.mat)[1], col="gray", lty="dotted")
par(mfrow = c(1,1)) # Grafik wieder 1x1
```

4.8.2 Weighted averaging – WA

```
require(rioja) # Paket laden
# pH Rekonstruktion eines K05-Kerns aus Round Loch of Glenhead,
# Galloway, SW Scotland. Dieser See versauerte im Laufe der letzten
# ca. 150 Jahre
data(SWAP) # surface sediment diatom data and lake-water pH
  spec <- SWAP$spec
  obs  <- SWAP$pH
data(RLGH) #stratigraphische Diatomeendaten von Round Loch of Glenhead, Galloway, Southwest Scotland
  core <- RLGH$spec
  age  <- RLGH$depth$Age
# Weighted averaging
fit <- WA(spec, obs, tolDW=TRUE)
dimnames(residuals(fit))[[2]]
#"WA.inv"     "WA.cla"     "WA.inv.tol" "WA.cla.tol"
# zentral steuern, welche Methode angezeigt
WA.methode <- "WA.inv" #"WA.inv" "WA.cla" "WA.inv.tol" "WA.cla.tol"
{# Start Grafik 1x2:
  par(mfrow=c(1,2))
  plot(fit,
    xlab="pH gemessen", ylab="pH geschätzt",
    # wenn "inv" gefunden, dann ...
    deshrink = if(grepl("inv",WA.methode)) "inverse" else "classical",
    # wenn "tol" gefunden, dann
    tolDW    = if(grepl("tol",WA.methode)) TRUE else FALSE
  )# Ende Grafik gemessen vs geschätzt
    abline(# lineare Modellinie
      lm(fitted(fit)[,WA.methode] ~ obs) -> swap.lm,
      lty="longdash"
    )
```

```
  plot(fit, resid=TRUE,
    xlab="pH gemessen", ylab="Residuen: pH geschätzt",
    # wenn "inv" gefunden, dann ...
    deshrink=if(grepl("inv",WA.methode)) "inverse" else "classical",
    # wenn "tol" gefunden, dann
    tolDW    = if(grepl("tol",WA.methode)) TRUE else FALSE,
    #add.smooth=TRUE # separat s.unten
  )# Ende Grafik Residuen
  # richtige Residuen!
  # plot(x=obs, y=obs - fitted(fit)[,WA.methode],
  #   xlab="pH gemessen", ylab="Residuen: pH geschätzt",
  #   las=1
  # )
  # abline(h=0, col='gray')
  # LLSESP linear least squares error slope parameter siehe Vasko et al. (2000)
    abline(
      lm(residuals(fit)[,WA.methode] ~ obs) -> swap.residuals.lm,
      lty="longdash"
    )
    panel.smooth(x=obs,y=residuals(fit)[,WA.methode] , pch=NA, col="red")
    legend("bottomright",
      # Glättungskurve + LLSESP
      legend=c("smooth (span = 2/3)", sprintf("LLSESP: %1$+.3f",coef(swap.residuals.lm)[2])),
      lty= c("solid", "longdash"),
      col= c( "red", "black"),
      cex=0.8
    )
  par(mfrow=c(1,1))
  title(# Titel zusammenbasteln
    paste("SWAP Datensatz (Diatomeendaten): WA",
      switch(WA.methode,# je nach Methode
        WA.inv = "inverse", WA.inv.tol = "inverse, tolerance DW",
        WA.cla = "classical", WA.cla.tol = "classical, tolerance DW",
        NULL # default
      )# Ende switch(WA.methode)
    )# Ende paste-f(x)
  )# Ende title-f(x)
}# Ende Grafik 1x2

# RLGH Rekonstruktion
  (pred <- predict(fit, core)) # (..) zusätzliche Ausgabe
#zeichne Rekonstruktion
  plot(age, pred$fit[, 1], type="b", las=1)
# Kreuzvalidierung mit → bootstrap
  fit.xv <- crossval(fit, cv.method="boot", nboot=1000)
{# Start Grafik 2x2:
  par(mfrow=c(2,2))
  plot(fit)
    title("Normales Modell...")
  plot(fit, resid=TRUE)
  plot(fit.xv, xval=TRUE)
    title("Bootstrap Modell...")
  plot(fit.xv, xval=TRUE, resid=TRUE)
  par(mfrow=c(1,1))
}# Ende Grafik 2x2
# RLGH Rekonstruktion mit Proben-speziefischen Fehlern
  (pred <- predict(fit, core, sse=TRUE, nboot=1000)) # (..) zusätzliche Ausgabe
```

4.8.3 Partial least squares – PLS

```
require(rioja) # Paket laden
data(IK) # Imbrie und Kipp Formaniferen Datensatz (Imbrie und Kipp 1971)
spec   <- IK$spec # 22 Formaniferen-Arten separat speichern (%-Abundanzen)
SumSST <- IK$env$SumSST # Sum Sea Surface Temperature
  core <- IK$core # Bohrkern 28 Formaniferen Taxa in 110 Proben
# WAPLS (Voreinstellung) oder PLS explizit
  fit <- WAPLS(spec, SumSST, iswapls=TRUE)
  fit # Modelparameter der 5 ersten Komponenten
# Kreuzvalidierung LOOCV
  fit.cv <- crossval(fit, cv.method="loo")
# Wieviele Komponenten?
  rand.t.test(fit.cv)# Zusammenfassung Test
  screeplot(fit.cv)  # RMSE vs. Anzahl der Komponenten
# Transfermodell schätzen
  pred <- predict(fit, core, npls=2) # mit 2 Komponenten
# Schätzung darstellen - depths are in rownames
depth <- as.numeric(rownames(core))
plot(y=depth*(-1), x=pred$fit[, 2], type="b",
  xlab="", ylab="Depth (m?)", # Achsenbeschriftung
  bty="n", # Box unterdrücken
  xaxt="n", yaxt="n", # y/x-Achsen unterdrücken
  las=1 # label assingnment: Teilstrichbeschriftung
)
# Achsen
  axis(2,at=axTicks(2), labels=axTicks(2), las=1)
  mtext(side=3,
    text=expression('Predicted '  * sum(Sea -  Surface - Temperature) * ' (°C)'),
    line=2
  )
  axis(3) # Achse oben
# Modell mit  evaluieren
pred <- predict(fit, core,
  npls=2,    # Anzahl Komponenten
  sse=TRUE,  # sample specific errors dazu?
  nboot=1000 # Anzahl bootstrap Schritte
)
pred # Ergebnis Schätzungen
```

4.8.4 Maximum Likelihood Response Surfaces – MLRC

```
require(rioja) # Paket laden
data(IK) # Imbrie und Kipp Formaniferen Datensatz
spec   <- IK$spec / 100 # in % umwandeln
SumSST <- IK$env$SumSST # Sum Sea Surface Temperature
core <-   IK$core / 100 # in % umwandeln

(fit <- MLRC(y=spec, x=SumSST)) # (..) + Ausgabe
# Method : Maximum Likelihood using Response Curves
# Call   : MLRC(y = spec, x = SumSST)
#
# No. samples  : 61
# No. species  : 22
# Cross val.   : none
#
# Performance:
#         RMSE      R2  Avg.Bias  Max.Bias
```

```
# MLRC  1.7313  0.9468   -0.0415   -2.1113

{# Start Grafik 1x2:
  par(mfrow=c(1,2))
  # Grafik gemessen vs geschätzt
  plot(fit, xlab="pH gemessen", ylab="pH geschätzt")
    abline(# lineare Modellinie
      lm(fitted(fit) ~ SumSST) -> IK.lm,
      lty="longdash"
    )
  plot(fit, resid=TRUE,
    xlab="pH gemessen", ylab="Residuen: pH geschätzt",
    #add.smooth=TRUE # separat s.unten
  )# Ende Grafik Residuen
  # richtige Residuen!:
  # plot(x=obs, y=obs - fitted(fit)[,"WA.inv"],
  #   xlab="pH gemessen", ylab="Residuen: pH geschätzt",
  #   las=1
  # )
  # abline(h=0, col='gray')
  # LLSESP linear least squares error slope parameter sieheVasko et al. (2000)
    abline(
      lm(residuals(fit) ~ SumSST) -> IK.lm,
      lty="longdash"
    )
    panel.smooth(x=SumSST,y=residuals(fit), pch=NA, col="red")
    legend("topright",
      # Glättungskurve + LLSESP
      legend=c("smooth (span = 2/3)", sprintf("LLSESP: %1$+.3f",coef(IK.lm)[2])),
      lty= c("solid", "longdash"),
      col= c( "red", "black"),
      cex=0.8
    )
# Grafik wieder 1x1
  par(mfrow=c(1,1))
  title(# Titel zusammenbasteln
    paste(" Imbrie & Kipp Datensatz (Forameniferen): MLRC")
  )# Ende title-f(x)
}# Ende Grafik 1x2
```

Schätzung anhand des Bohrkerns

```
  (pred <- predict(fit, core))# (..) + Ausgabe
#zeichne Rekonstruktion - Tiefen sind in Zeilennamen enthalten!
  depth <- as.numeric(rownames(core))
# plot(depth, pred$fit[, 1], type="b") # aus ?plot.MLRC
{# Start gedrehte Grafik
  plot(y=depth*(-1), x=pred$fit[, 1], type="b",
    xlab="", ylab="Depth (m?)", # Achsenbeschriftung
    bty="n", # Box unterdrücken
    xaxt="n", yaxt="n", # y/x-Achsen unterdrücken
    las=1 # label assingnment: Teilstrichbeschriftung
  )
  axis(2,at=axTicks(2), labels=axTicks(2), las=1)
  mtext(side=3,
    text=expression('Predicted ' * sum(Sea - Surface - Temperature) * ' (°C)'),
    line=2
  )
  axis(3) # Achse oben
} # Ende gedrehte Grafik
```

```
# Braucht Zeit!
# Kreuzvalidierung LOOCV
  fit.cv <- crossval(fit, cv.method="loo", verbose=5)
# Schätzung mit proben-speziefischem Fehler; → bootstrap
  pred <- predict(fit, core, sse=TRUE, nboot=1000, verbose=5)
```

5 Programmierung

Wer sich mit Programmierung beschäftigt, sollte sich als erstes einen *guten* Text-Editor, wie auf Seite 135f genannt, besorgen.

Will man Dinge programmieren, packt man dies am besten in eine separate Datei, die man dann mit `source('pfad/zur/Datei.R')` ausführen lassen kann. Da Ergebnisse von Funktionen *nicht* zur Konsole ausgegeben werden wenn sie in einer separaten Datei ausgeführt werden, muß man dies explizit mit `print(...)` oder `cat(...)` anfordern z.B.:

```
require(exactRankTests)
cat("Wilcoxon Rang Summen Test...\n")# Meldung zur Konsole
  a <- c(1,1,1,1,1,1,1,1,1,1,1)
  b <- c(4,4,4,4,4,4,4,4,4,4,4)
  wilcox.result <- wilcox.exact (a,b)
  print(wilcox.result) # "Ausdrucken" muß innerhalb von source() explizit angegeben werden
```

5.1 Benutzerfunktionen

Eine Benutzer-Funktion zu programmieren ist nicht schwer. Um z.B. die Quadratzahl oder andere polynome Zahlen durch eine Funktion berechnen zu lassen kann man mit folgender Vorlage beginnen:

```
polyzahl <- function(data, exponent=2){
  if(missing(data))
    stop("Stop hier 'data' fehlt. Benutze: polyzahl(5, exponent=4)")

} # end polyzahl()
```

In die Leerzeile schreibt man dann einfach . . .

```
    data^exponent
```

. . . und nun kann man die Funktion schon benutzten. `polyzahl()` benutzt also 2 Argumente, wobei das 2. schon vordefiniert ist. Alle Arten von Definition sind natürlich möglich, wie auch die häufig benutzten Bedingungen `NULL`, `TRUE` oder `FALSE`. Wenn man nun den ganzen Funktions-Code einmal von R hat durchlesen lassen, indem man die ganze Funktion entweder in die Konsole kopiert oder über eine separate Datei via `source('Benutzerfunktionen.R')` durchlesen läßt, erhält man beispielsweise:

```
polyzahl()
  Fehler in polyzahl() : Stop hier 'data' fehlt. Benutze: polyzahl(5, exponent=4)
polyzahl(7)
  49
polyzahl(7, 3)
  343
```

Wie man sieht, wird das 2. Argument automatisch „`exponent`“ zugeordnet. Die Reihenfolge ist also WICHTIG. Weiß man nicht an welcher Stelle sich welche Option befindet, gibt man das entsprechende Argument explizit an, z.B.:

```
polyzahl(exponent=3, data = 7)
  343
```

Häufig sind die Benutzerfunktionen nicht so simpel, sondern verschachtelt auch mit schon vorhanden Funktionen, die ja wiederum ihre eigenen Argumente haben. Will man diese ebenfalls benuzten, aber nicht explizit ausformulieren, gibt man bei den Funktions-Argumenten zusätzliche noch drei Punkten an „...“, und somit wird alles Übrige zu den entsprechenden verschachtelten Funktionen weitergeleitet, wo eben auch `...` steht:

```
meinefunktion <- function(data,...){
  if(missing(data))
    stop("Stop hier 'data' fehlt. Benutze: meinefunktion(data)")
    # anderer
    # code
    # hier
    text(data, ...) # alle anderen Argumente via '...' hierher geleitet
} # end meinefunktion()
```

Wird diese Art von Weiterleitung zu mehreren Funktionen benutzt, kann es unter Umständen dazu führen, daß R nicht weiß, was nun zu wem weitergeleitet werden soll. R gibt in solchen Fällen dann eine entsprechende Fehlermeldung aus.

5.1.1 Eine Legende platzieren: `click4legend(...)`

Ein praktisches Beispiel zur Weiterleitung wäre eine Funktion, die mit der Maus eine Legende auf der Grafikfläche platziert. Wir nennen sie mal `click4legend`. Die Funktion `legend(...)` existiert schon und wir kombinieren sie mit `locator()`, welche die Mausposition erfaßt:

```
click4legend <- function(
  text, # Legendentext
  ...   # weiterleiten der Argumente an Funktion legend()
  ){
  if(missing(text))
    stop("Stop hier 'text' fehlt. click4legend('Legendentext'). Siehe auch ?legend")
    xypos <- locator(1) # 1x Mausposition speichern
    legend(
      x=xypos$x,
      y=xypos$y,
      legend=text, # text übernehmen
      ...
    )
} # end click4legend()
```

Nun können wir sie benutzen:

```
plot(1)
click4Legend('text')
```

Für den uneingeweihten Benutzer macht die Funktion vielleicht den Eindruck, daß da was hängt. Daher werden wir mit `cat("...\n")` noch eine Info einfügen, sowie mit `par(xpd=NA)` die Möglichkeit einbauen, die Legende auch über den Grafikrand hinaus zu platzieren:

Platzieren einer Legende auf der Grafikfläche mit Maus

```
click4legend <- function(
  text, # Legendentext
  randZeichnen = TRUE, # auch außerhalb der Grafikfläche?
  ...   # weiterleiten anderer Argumente an Funktion legend()
  ){
```

```
  if(missing(text))
    stop("Stop hier 'text' fehlt. click4legend('Legendentext'). Siehe auch ?legend")
  # Info
    cat("Klicke mit der Maus eine Position für die Legende ...\n")
  # Mausposition speichern
    xypos <- locator(1)
  # auch außerhalb der Grafikfläche
    if(randZeichnen)
      par(xpd=NA) -> parxpd # vorherige Einstellung zwischenspeichern
  # Legende einfügen
  legend(
    x=xypos$x,
    y=xypos$y,
    legend=text, # text übernehmen
    ...
  )
  if(randZeichnen)
    par(parxpd) # wieder ursprüngliche Einstellung
} # end click4legend()
```

Die Benutzerfunktion `arrowLegend()` auf Seite 196 ist eine erweiterte Variante der hier besprochenen Funktion.

Das Kommentieren von Programmcode sollte man generell nicht vergessen, denn nach 3 Jahren weiß man nämlich sehr wahrscheinlich nicht mehr, wofür diese oder jene Funktion eigentlich mal gedacht war . . . ;-).

5.2 Funktionsbausteine

Mit Hilfe einfacher Kontrollstrukturen, wie `if`, `switch` oder `for` läßt sich so manches leicht zusammenbasteln. Man findet sie nicht über `?if`, sondern in der Hilfe unter `?Control`. Als gute Praxis hat es sich bewährt, alle Klammern *immer* zu kommentieren: {}

```
{# start ...
  # ... Code hier
}# end ...
```

Die Funktion `if` ist für einfache Abfragen: if()

```
zahl <- 5
if(zahl > 3){
  # größer 3
  # Code hier...
} else {
  # kleiner gleich 3
  # Code hier ...
}# end if(zahl)
if(zahl > 3) 'größer' else 'kleiner' # Kurzvariante ohne {}
```

Die Kurzvariante ohne `{...}` funktioniert im Allgemeinen nur bis einschließlich der nächsten Zeile. ifelse()

```
# Kurzvariante von if ... else ...:
  ifelse(daten > 3, "black", "gray")
# funktioniert z.B. innerhalb von text(...)
```

Innerhalb von Funktionen ist ein `if()` kombiniert mit `stop()` recht nützlich, um dem Benutzer weitere Hinweise zu geben, wie die Funktion zu gebrauchen ist oder welche Argumente noch fehlen: stop()

```
if(missing(text)) # stop hier mit Erklärung
  stop("Stop hier 'text = \"some text\"' fehlt. Benutze:\nfunktion(text='Mein Text.')")
```

Die Funktion `switch()` erledigt Abfragen zu mehreren Werten. Sie ist quasi eine mehrfach `if()`-Funktion: switch()

```
text <- "mean"
switch(text,
  mean   = mean(x),
  median = median(x),
  NULL # Wenn kein Wert zutrifft (default)
)# end switch()
# da text "mean" enthält wird die Zeile bei mean = ... ausgeführt
```

```
text <- "meaN"
switch(text,
  Mean   =,
  MEAN   =,
  mean   = mean(x),
  median = median(x),
  NULL # default
)# end switch()
# da text "meaN" enthält wird NULL ausgegeben, weil meaN zu nichts paßt. Falls GROß/klein egal, ↩
    dann effizienterer Test mit tolower(text), d.h. ohne alle GrOß/kLeiN Varianten schreiben zu ↩
    müssen.
```

Beachte, daß wenn man „`Mean`“ als Text übergeben hätte, wäre der Mittelwert berechnet worden, denn „`Mean`“ ist zwar leer, aber das nächste (wieder) definierte nachfolgende Element ist `mean   = mean(x)`. Fällt der Code der einzelnen Abfragen länger aus, klammert man in besser à la:

```
  # -----8< --- Teil innerhalb switch:
  mean   = { # langer Code hier, der auch Zeilen umbrechen kann
  }, # end langer Code
  # -----8< --- ... weiter mit switch
```

for() Die `for`-Schleife führt etwas mit einer Reihe von Werten aus:

```
for(index in 1:4){
  # tue was mit 1, 2, 3 und 4
  # gespeichert in index
}# end for index
for(index in c("eins", "zwei", "drei")){
  # tue was mit "eins", "zwei", "drei"
  # gespeichert in index
}# end for index
```

while() Die `while`-Schleife führt etwas solange aus, bis die definierte Bedingung nicht mehr stimmt:

```
# while(Bedingung) Ausdruck oder while(Bedingung) {langer zeilenumbrechender Ausdruck}
z <- 0
# so lange bis z < 5 nicht mehr TRUE
while(z < 5) {
  z <- z+2
  print(z)
}# Ende while(z < 5)
```

Da `z` innerhalb der `{...}`-Klammern gesetzt wird, gibt die Bedingung `z < 5` erst *dann* `FALSE` zurück, wenn `6` erreicht ist und bricht dann erst ab. Die Funktion ist generell etwas mit Vorsicht zu gebrauchen, falls man sich verprogrammiert und die Schleife immer `TRUE` ist. Dann hilft nur ein kill des R-Programms, wenn es Esc nicht tut. Mit `break` kann auch zwischenzeitlich abgebrochen werden.

6 Diverses

6.1 Diversitätsindizes

Verschiedene Diversitätsindizes lassen sich mit `diversity(...)` aus dem Paket `vegan` berechnen.

6.2 Interpolation räumlich irregulärer Daten

Die Funktion `interp(...)` aus dem Paket `akima` bietet die Möglichkeit irregulär verteilte Daten linear und nicht-linear zu interpolieren. Dabei sind keine NA's erlaubt. das folgende Beispiel zeichnet eine Karte von China mit Falschfarbendarstellung (mit `image(...)`) sowie Isolinien mit `contour(...)`.

```
### Bsp.: mit Datensatz http://www.webgis-china.de/
library(akima) # Paket laden
webgis <- read.csv("./webgis.csv"); # Daten einlesenaus aktuellem Verzeichnis
attach(webgis) # in den Suchpfad aufnehmen
### Interpolation
# nichtlinear
webgis.interp.nlin <- interp.new(lon,lat,T_Jul,
    xo=seq(# sequenz ...
      range(lon)[1], # von Minimum 'longitude'...
      range(lon)[2], # ...bis Maximum 'longitude'
      length=round(range(lon)[1],0)*10 # wieviel Werte?
    ), # xo Ende
    yo=seq(# sequenz ...
      range(lat)[1], # von Minimum 'latitude'...
      range(lat)[2], # ...bis Maximum 'latitude'
      length=round(range(lat)[1],0)*10), # wieviel Werte?
      linear=FALSE # nichtlinear
  )# interp.new Ende
# linear
webgis.interp.lin <- interp.old(lon,lat,T_Jul,
  xo=seq(range(lon)[1],range(lon)[2], length=round(range(lon)[1],0)*10),
  yo=seq(range(lat)[1],range(lat)[2], length=round(range(lat)[1],0)*10)
  ) # linear
```

Bei `xo=` sowie `yo=` wird lediglich eine höhere Interpolation vorgenommen aber abhängig von Längen- (`lon`) und Breitengrad `lat`: `range(...)` gibt min. & max. zurück, `seq(...)` erzeugt eine Sequenz mit jeweils Minimum als Anfang und Maximum als Ende. `[1]` oder `[2]` greift jeweils auf min. oder max. zu.

```
### Daten in Karte einzeichnen
library(maps) # Paket für Kartendaten
  map("world","China", type="n") # Grafikproportionen vorgeben, aber nicht zeichnen
  map("world",add=TRUE, fill=TRUE, col="lightgrey") # restliche Länder mit grau
  title("mittl. Juli Luft Temp. [C] nicht-linear", # Titelei
    xlab="Länge (°)", # x-Achse
    ylab="Breite (°)", # y-Achse
    sub="Daten webgis China " # Untertitel
  )# Ende title()
map("world","China", add=TRUE, fill=TRUE, col="white") # weiße China-Karte
### Falschfarbenbild
# nlinear:
image(webgis.interp.nlin,col=rainbow(20, start=0.51, end=0.22),add=T) # nlinear
# linear
# image(webgis.interp.lin,col=rainbow(20, start=.75, end=.1),add=T) # linear
map("world","China", add=T ) # nochmal Grenzen zeichnen
### Isolinien
```

```
contour(webgis.interp.nlin,
  add=TRUE,
  col="darkred",
  lwd=0.5, # line width
  nlevels=28) # Anzahl der Stufen
### Kartenzusätze dazu
map.axes() # Achsen dazu
points(lon,lat, pch=16, cex=0.5) # Stationen dazu
# Skalierbalken mit Maus setzen
map.scale(locator(1)$x,locator(1)$y, cex=0.7)
grid(col="grey") # Gitternetz in grau
```

☞ Mal eine Frage: ist es sinnvoll die Daten nicht-linear zu interpolieren?

6.3 Minimalbaum

Einen sogenannten Minimum Spanning Tree kann man mit dem Paket `ade4` berechnen.

```
x <- runif(10) # Zufallszahlen generieren
y <- runif(10)
xy <- cbind(x=x,y=y) # Zahlen zusammenfügen
mst.xy <- mstree(dist(xy), ngmax = 1)  # Minimum Spanning Tree berechnen
s.label(xy,
  clab = 1, # Größe der Beschreibung hier Index
  cpoi = 0, # Punktgröße
  neig = mst.xy, # von neighbour
  cnei = 1) # Liniendicke
```

6.4 Ausreißer Test

Das Paket `car` bietet einen Ausreißer Test an. Man kann natürlich auch einen Boxplot darstellen siehe hierzu auf Seite 48. Alternativ gibt es auch ein ganzes Paket `outliers` dafür.

```
library(car) # Paket laden
data(Duncan) # Daten laden
?Duncan # Hilfe zum Datensatz
  outlier.test(lm(prestige income+education, data=Duncan))
# max|rstudent| = 3.134519, degrees of freedom = 41,
# unadjusted p = 0.003177202, Bonferroni p = 0.1429741
#
# Observation: minister
# bei Observation könnte auch der Zeilenindex stehen
```

6.5 LaTeX/HTML Ausgaben

Das Paket `Hmisc` bietet viele kleine Helfer. So lassen sich auch Ausgaben für LaTeX generieren oder HTML ebenfalls.

```
require(Hmisc)  # Paket laden
x <- matrix(1:6, nrow=2, dimnames=list(c('a','b'),c('c','d','enLinie 2')))
x
    c d enLinie 2
  a 1 3        5
  b 2 4        6
```

...Fortsetzung umseitig

```
latex(x)  #  erstellt temporäre *.tex Datei + dvi-Ausgabe
html(x)  #  erstellt x.html Datei im Arbeitsverzeichnis
```

☞ siehe hierzu auch „Some examples of conditional typesetting using the latex() function.“ (David Whiting, 2005) http://biostat.mc.vanderbilt.edu/twiki/pub/Main/StatReport/latexFineControl.pdf

Das Paket `xtable` bietet ebenfalls die Möglichkeiten LaTeX oder HTML auszugeben. Bei LaTeXsowohl `longtable` als auch die normale Tabellenumgebung `tabular` (Voreinstellung):

```
(x <- matrix(rnorm(1000), ncol = 10))  # 100 x 10 Tabellen Matrize + Ausgabe durch Klammer (...)
außen
x.big <- xtable(x,
  label='tab:gross',  # LaTeX Label
  caption='Example of longtable spanning several pages',  # LaTeX caption
  caption.placement = "top",  # platzieren Tabellen-Überschrift
  digits = 2,  # nur 2 Kommastellen
    display = c(  # explizites angeben, welches Format
      "d",  # Ganzzahl - betrifft fortlaufende Zeilennummer (automatisch ausgegeben)
      "d",  # Ganzzahl
      "g",  # "g"  and "G"  put x[i] into scientific format only if it saves space to do so
      "g",  # "g"  and "G"  put x[i] into scientific format only if it saves space to do so
      "G",  # "g"  and "G"  put x[i] into scientific format only if it saves space to do so
      "G",  # "g"  and "G"  put x[i] into scientific format only if it saves space to do so
      "e",  # Format 1.010e+01
      "E",  # Format 1.010E+01
      "s",  # String
      "fg", # 1. + Anzahl echter Kommastellen (digits)
      "f"    # nur Anzahl digits incl. erste Zahl vor dem Komme
    ),
  align = "llllllccrrrr"   # Ausrichtung in LaTeX l-links c-center r-rechts
)
print(x.big,tabular.environment='longtable',floating=FALSE)
# gibt eigentliche Tabelle aus
 % latex table generated in R 2.4.1 by xtable 1.4-6 package
 % Sat Feb 23 22:01:48 2008
 \begin{longtable}{llllllccrrrr}
   \hline
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
   \hline
   1 &   0 & 0.55 & $-$0.55 & $-$1.9 & 0.77 & $-$1.32e$-$01 & 6.62E$-$01 & $-$0.444459229310107 &
$-$0.88 & $-$0.28 \\
   2 &   0 & $-$1.4 & 0.93 & $-$0.91 & 1.1 & 1.06e+00 & $-$6.29E$-$01 & 0.394973846147042 & $-$0.83
& 0.57 \\
   : & : &   : & : & : & : & : & : & : & : & :  \\
   100 &  $-$2 & 0.63 & $-$1.1 & 0.31 & $-$0.97 & $-$6.70e$-$01 & $-$8.91E$-$01 &
0.0149816467544707 & 0.37 & 1.89 \\
    \hline
 \hline
 \caption{Example of longtable spanning several pages}
 \label{tab:gross}
 \end{longtable}
```

☞ es wird immer die laufende Zeilennummer mit ausgegeben. Es sind also immer $n + 1$ Spalten.

Mit der Funktion `Sweave(..)` aus dem Paket `utils` kann man ganze Dokumente incl. Grafiken und R-Anweisungen generieren. Dazu müssen im Vorlagendokument (LaTeX oder Open Document Format [Open Office]) gewisse Anweisungen eingefügt werden an deren Stelle dann die Ersetzung erfolgt. So ergibt

```
We can also emulate a simple calculator:
<< echo=TRUE,print=TRUE >>=     # 1. Startteil
1 + 1                           # 2. Anweisungen
1 + pi
sin(pi/2)
@                               # 3. Schlussteil
```

```
We can also emulate a simple calculator:
  1+1
 [1] 2
  1 + pi
 [1] 4.141593
  sin(pi/2)
 [1] 1
```

```
                                    Vorlagen aufrufen + generieren
install.packages("odfWeave", dependencies=TRUE) # eventuell installieren
library(odfWeave)                # Laden der neuinstallierten Pakete
library(utils)                   # für Sweave()
# roffice.odt Datei mit obigen Anweisungen erstellt
odfWeave("/Pfad/zur/Datei/roffice.odt","/Pfad/zur/Datei/rofficeout.odt")
# Datei Sweave-test-1.Rnw mit obigen Anweisungen erstellt
testfile <- system.file("Sweave", "Sweave-test-1.Rnw", package = "utils")
# par(ask=FALSE) Grafik neuzeichnen nicht nachfragen
options(par.ask.default=FALSE)
# LaTeX- Datei erstellen
Sweave(testfile) # es reicht auch einfach eine Vorlagendokument *.Rnw zu haben
                                    Automatisieren unter Linux
bash> echo "library(\"utils\"); Sweave(\"/Pfad/Dateiname.Rnw\")"  | R -no-save -no-restore  ;
bash> latex /Pfad/Dateiname.tex
```

7 Linkliste - Tutorien - Pakete

Programme

Das Programm R findet sich unter:
.....http://www.r-project.org/
Ich empfehle unbedingt einen Editor mit Syntaxhervorhebung zu benutzen. Man behält dadurch die Übersicht, wenn man mit R arbeitet. So braucht man sich auch keine großen Excel Dokumente aufzuheben, sondern nur die Ursprungsdaten und das R-Skript.
Tinn-R Editor mit Syntaxhervorhebung, Referenzkartei(!!!) für Funktionen, R-Suchfunktion sowie Übermittlung der Anweisungen an R
.....http://www.sciviews.org/Tinn-R/

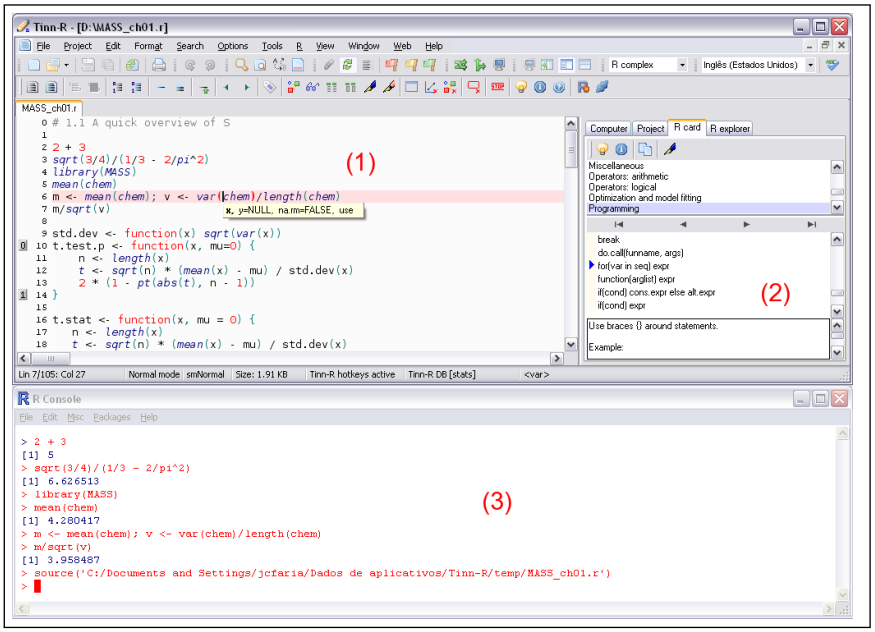


Abbildung 4: Tinn-R: Editor mit Syntaxhervorhebung (1), Referenzkartei(!!!) (2) sowie Übermittlung der Anweisungen (3) an die R Konsole

Notepad++ (Freeware): sehr guter unicode-fähiger Editor mit Syntaxhervorhebung, Makroaufzeichnung und nützlichen Helferleins wie: Autovervollständigung, Tipp-Hilfe, Projektsitzungen, Textschnippsel ...
http://notepad-plus.sourceforge.net/de/site.htm

Crimson Editor (Freeware): mit Syntaxhervorhebung (Syntaxtyp unter `Document -> Syntaxtype` neuen Syntaxtyp für R einstellen) und Makrofunktion
http://www.crimsoneditor.com

R-WinEdit (Shareware): Editor mit Syntaxhervorhebung und vielen, vielen Makrofunktionen
http://cran.r-project.org/contrib/extra/winedt/ReadMe.txt

Kate (Linux): Editor mit Syntaxhervorhebung
http://kate.kde.org/

John Fox' R Commander – grafische Oberfläche mit Menüführung als „Rcmdr" Paket in R
http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/

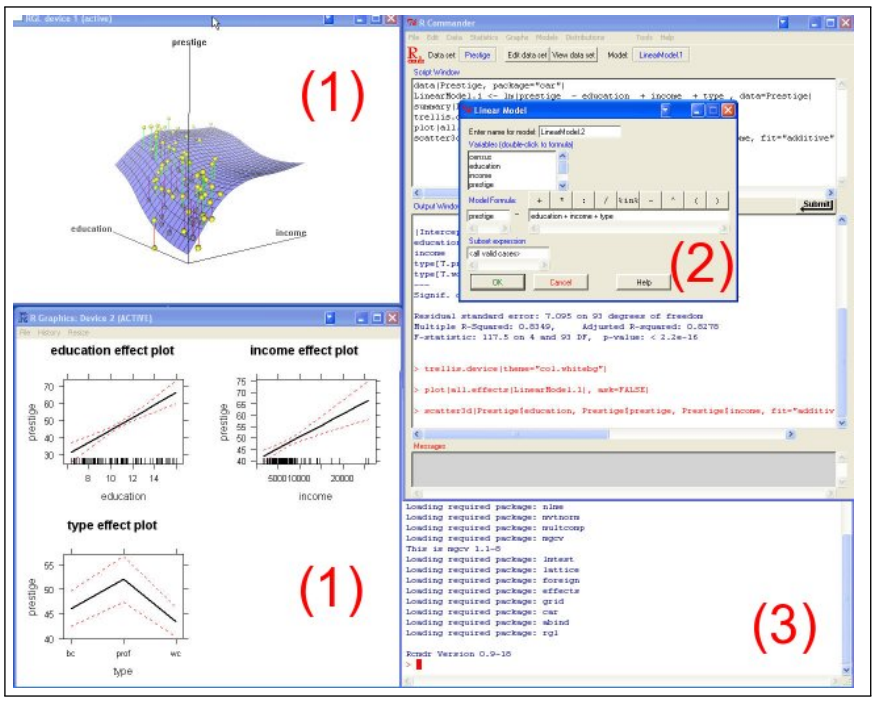


Abbildung 5: John Fox' R Commander mit 2 R-Grafikfenstern (1), menügeführter Eingabemaske (2) sowie mit R Konsole (3)

Wer sich mit einer grafischen Oberfläche eher anfreunden kann und Freeware sucht, findet durch das R-ähnliche Programm **Statistiklabor** der FU-Berlin.
..... http://www.statistiklabor.de/

Ein ganz gutes, speziell für Paläontologen entwickeltes Programm ist PAST
..... http://folk.uio.no/ohammer/past
sowie für Stratigraphie (transfer functions): **C2** – bis 75 Proben „Freeware“
..... http://www.campus.ncl.ac.uk/staff/Stephen.Juggins/index.htm
Ebenfalls ein grafisches Freewareprogramm ist **Vista** mit einigen Plugins zur Erweiterung in der Multivariaten Datenanalyse. Jedoch kann man z.B. keine Clusteranalysen durchführen.
..... http://www.visualstats.org/vista-frames/online/index.html
Unter Linux gibt es zur Visualisierung auch für KDE das Programm **LabPlot**
..... http://labplot.sourceforge.net/
Wer kein Grafikprogramm besitzt, um vielleicht doch die eine oder andere Grafik zu bearbeiten, kann das Photoshop-Äquivalent GIMP für Windows benutzten.
..... http://www.gimp.org
Vektor-Grafikprogramm
..... http://www.inkscape.org/
Datenbanken auf Internet-/HTML-Basis gehen sehr gut mit `MySQL`. Das Werkzeug **phpMyAdmin** (http://www.phpmyadmin.net/) ist dabei eine sehr nützliche HTML-Datenbank-Oberfläche. Einfache Installation über:
..... http://www.apachefriends.org/de/

Tutorien, Glossars, Übungen, Anschauliches

Eine ganz gute **Einführung** in R (auch mit Übungsaufgaben) ist auf Seite:
..... http://www-m1.ma.tum.de/nbu/statprakt/
Ein **Wiki** für R
..... http://wiki.r-project.org/rwiki/
Eine **Grafiksammlung** zu R ist zu finden unter:
..... http://addictedtor.free.fr/graphiques/

Eine weitere `Grafiksammlung` von Paul Murrell
..... http://www.stat.auckland.ac.nz/~paul/RGrafics/rgraphics.html
Änderungen in Grafiken + Statistik (sehr ausführlich) „From Data to Grafics“ Dokumente gehen von `01.html` bis `18.html`
..... http://zoonek2.free.fr/UNIX/48_R/03.html
Eine **Einleitung** für R auch mit Anwendungsbeispielen findet sich unter:
..... http://www-m1.ma.tum.de/nbu/statprakt/
Ein sehr vielseitiges `Tutorium` das auch das Paket `Simple`[32] für R behanselt:
..... http://www.math.csi.cuny.edu/Statistics/R/simpleR/index.html
Ein umfangreiches Skript, was viele **Probleme in der Statistik** erklärt und sehr weit geht – in englisch:
..... http://home.ubalt.edu/ntsbarsh/Business-stat/opre504.htm
ANOVA für R wird eingehend behandelt in:
..... http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf
Ordinationsmethoden – englisch – mit gutem **Glossar** unter:
..... http://www.okstate.edu/artsci/botany/ordinate/
Glossar für morphometrische Begriffe
..... http://life.bio.sunysb.edu/morph/
multilinguales **Statistik Glossar**
..... http://isi.cbs.nl/glossary.htm
Interaktive Statistik (Java unterstützte Münsteraner Biometrie-Oberfläche) kann man ausprobieren unter:
..... http://medweb.uni-muenster.de/institute/imib/lehre/skripte/biomathe/bio/index.html
Um sich ein Bild über verschiedene **Verteilungstypen** zu machen, mal auf der folgenden Seite schauen
..... http://www.uni-konstanz.de/FuF/wiwi/heiler/os/index.html
Ein Skript zu **Verteilungsanpassungen** findet sich auf dem R-Server
..... http://cran.r-project.org/doc/contrib/Ricci-distributions-en.pdf
Eine gutes Skript zu R und Multivariate Statistik ist Oksanen (2004)
..... http://cc.oulu.fi/~jarioksa/opetus/metodi/notes.pdf

Pakete

Eine Paketzusammenstellung speziell für Umweltdaten findet sich unter:
..... http://cran.r-project.org/contrib/main/Views/Environmetrics.html
Zusammenstellung für Raummusteranalysen:
..... http://cran.r-project.org/src/contrib/Views/Spatial.html
Pakete thematisch
..... http://cran.r-project.org/web/views/
Alle aktuellen Pakete mit Kurzerklärung unter „Packages“
..... http://stat.ethz.ch/CRAN/

Tabelle 6: Pakete in R: hier eine Liste für Version v2.0

`ade4`	Analysis fof Environmental Data : Exploratory and Euclidean methods in Environmental sciences
`adehabitat`	Analysis of habitat selection by animals
`amap`	Another Multidimensional Analysis Package
`analogue`	Modern Analogue Technique transfer function models for prediction of environmental data from species data
`akima`	Interpolation of irregularly spaced data
`base`	The R Base Package
`boot`	Bootstrap R (S-Plus) Functions (Canty)
`class`	Functions for Classification

...Fortsetzung umseitig

[32] Installation: `install.packages("Simple",contriburl="http://www.math.csi.cuny.edu/Statistics/R/simpleR/")`

Fortsetzung R - Pakete...

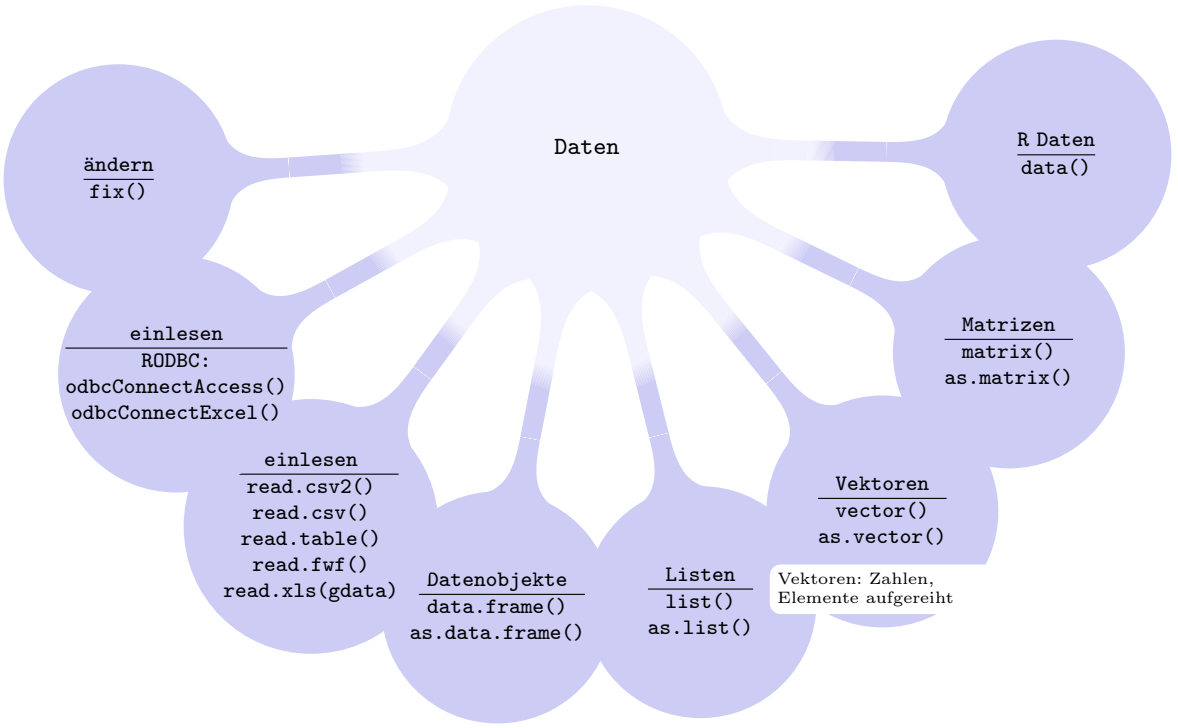
clim.pact	Climate analysis and downscaling package for monthly and daily data.
clines	Calculates Contour Linies
clue	Cluster ensembles cluster Functions for clustering (by Rousseeuw et al.)
CoCoAn	Constrained Correspondence Analysis
datasets	The R Datasets Package
Design	Regression modeling, testing, estimation, validation, graphics, prediction (package Hmisc is required)
dse	Dynamic System Estimation, a multivariate time series package. Contains dse1 (the base system, including multivariate ARMA and state space models), dse2 (extensions for evaluating estimation techniques, forecasting, and for evaluating forecasting model), tframe (functions for writing code that is independent of the representation of time). and setRNG (a mechanism for generating the same random numbers in S and R).
dynamicGraph	dynamicGraph
dyn	Time Series Regression
eda	Exploratory Data Analysis
exactLoglinTest	Monte Carlo Exact Tests for Log-linear models
exactRankTests	Exact Distributions for Rank and Permutation Tests
fda	Functional Data Analysis
foreign	Read data stored by Minitab, S, SAS, SPSS, Stata, ...
fpc	Fixed point clusters, clusterwise regression and discriminant plots
gclus	Clustering Grafics
graphics	The R Grafics Package
GRASS	Interface between GRASS 5.0 geographical information system and R
grDevices	The R Grafics Devices and Support for Colours and Fonts
grid	The Grid Grafics Package
gstat	uni- and multivariable geostatistical modelling, prediction and simulation
Hmisc	Harrell Miscellaneous, LaTeXconverting, many functions imputing missing values, advanced table making, variable
its	Irregular Time Series
KernSmooth	Functions for kernel smoothing for Wand & Jones (1995)
klaR	Classification and visualization lattice
Lattice	Grafics methods Formal Methods and Classes
MASS	Main Package of Venables and Ripley's MASS
mgcv	GAMs with GCV smoothness estimation and GAMMs by REML/PQL
nlme	Liniear and nonlinear mixed effects models
nnet	Feed-forward Neural Networks and Multinomial Log-Liniear Models
PASTECS	Package for Analysis of Space-Time Ecological Series
pheno	Some easy-to-use functions for time series analyses of (plant-) phenological data sets.
pls.pcr	PLS and PCR functions
prabclus	Test for clustering of presence-absence data
rpart	Recursive Partitioning
scatterplot3d	3D Scatter Plot
seas	Seasonal analysis and graphics, especially for climatology
shapes	Statistical shape analysis spatial Functions for Kriging and Point Pattern Analysis
Simple[33]	functions and data to accompany simpleR
splines	Regression Spline Functions and Classes
stats	The R Stats Package
stats4	Statistical functions using S4 classes

...Fortsetzung umseitig

[33] Installation: `install.packages("Simple",contriburl="http://www.math.csi.cuny.edu/Statistics/R/simpleR/")`

Fortsetzung - Pakete...

`survival`	Survival analysis, including penalised likelihood.
`tcltk`	Tcl/Tk Interface
`tools`	Tools for Package Development
`utils`	The R Utils Package
`vegan`	Community Ecology Package
`vioplot`	Violin plot



Vektoren abfragen

`x[n]` das `n`-te Element
`x[-n]` ohne das `n`-te Element
`x[1:n]` Elemente `1` bis `n`
`x[-(1:n)]` ohne Elemente `1` bis `n`
`x[c(1,4,2)]` kombiniere Elemente `1`, `4` und `2`
`x["name"]` Element mit Namen `"name"`
`x[x > 3]` alle Elemente größer `3`
`x[x > 3 & x < 5]` alle Elemente zwischen `3` und `5`
`x[x %in% c("Faktor1","Faktor2")]` Elemente mit den entsprechenden sog. `factors` (Einflußgroßen-Faktoren)

Matrizen/Tabellen abfragen

`x[r,z]` Element Zeile `r`, Spalte `z`
`x[r,]` Zeile `r`
`x[,z]` Spalte `z`
`x[,c(1,3)]` Spalten `1` und `3` kombinieren
`x["name",]` Zeile mit Name `"name"`
`x[["name"]]` Spalte mit Name `"name"`
`x$name` Spalte mit Namen `"name"`

Listen abfragen

`x[n]` oder `x[[n]]` Liste `x` mit `n` Elementen: das `n`-te Element der Liste
`x[["name"]]` oder `x$name` Liste `x` mit Elementen `"name"`

Abbildung 2: Verschiedene Möglichkeiten, um mit Daten umzugehen.

Glossar

A

abhängig $X \to Y$ – Hier ist das eine Merkmal Y in seiner Ausprägung vom anderen Merkmal X abhängig, während umgekehrt X von Y unbeeinflußt ist. Es liegt also ein unabhängiges und ein abhängiges Merkmal vor. In Schaubildern trägt man das unabhängige Merkmal auf der Abszisse (x-Achse) auf und das unabhängige Merkmal auf der Ordinate (y-Achse).

AIC Das AIC (Akaikes Informationskriterium nach engl.: Akaike's Information Criterion) ist ein Maß zur Beurteilung der „Güte" von multivariaten Modellen, die auf Maximum Likelihood-Schätzungen basieren (beispielsweise die logistische Regression und verwandte Verfahren). Es soll vor allem helfen, unterschiedliche nicht-geschachtelte Modelle (über den gleichen Datensatz!) zu vergleichen. (Nicht-geschachtelte Modelle sind solche, von denen sich nicht eines als „Unterfall" des anderen verstehen läßt, anders formuliert, von denen jedes mindestens eine Variable enthält, die in dem jeweils anderen Modell nicht enthalten ist.)
Ein Modell ist umso besser zu den Daten (oder umgekehrt), je kleiner der Wert des AIC ist.
Eine Alternative zum AIC, die in jüngster Zeit mehr favorisiert wird, ist das BIC.
(Quelle: http://www.lrz-muenchen.de/~wlm/ilm_a27.htm)

Alpha-Fehler Ein Signifikanztest befindet den Unterschied zwischen zwei Meßwerten dann als signifikant, wenn der Unterschied so groß ist, daß es nach den Gesetzen der Wahrscheinlichkeitsrechnung als extrem unwahrscheinlich angesehen werden kann, daß kein Unterschied besteht. Nun ist es jedoch relativ offen, welche Wahrscheinlichkeit denn als klein genug gelten kann. Es handelt sich daher um eine Übereinkunft, daß gemeinhin bei einer Wahrscheinlichkeit von 5 % (und darunter) von Signifikanz gesprochen wird. Nun heißt dies jedoch, daß ein Signifikanztest, der zwei Meßwerte nur noch mit 5 % Wahrscheinlichkeit für ähnlich hält, dazu verleitet, die beiden Meßwerte eben für unterschiedlich zu halten. Dennoch besteht laut Test aber eine Wahrscheinlichkeit von 5 %, daß sie doch ähnlich sind und sich nicht unterscheiden. Wenn man aufgrund des Tests also davon ausgeht, daß sie sich unterscheiden, macht man mit eben jener 5 %-tigen Wahrscheinlichkeit einen Fehler. Dieser Fehler wird Alpha-Fehler genannt. s.auch Alpha-Fehler-Angepaßt
(Quelle: http://www.wu-wien.ac.at/inst/ivm/strunk/pdf/StatistikGlossar.pdf)

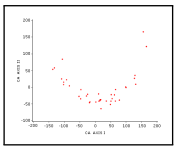
Alpha-Fehler-Angepaßt In der Regel sind Signifikanztests in der Lage, nur zwei Meßwerte miteinander zu vergleichen. Einige Fragestellungen machen daher mehrere Vergleiche zwischen jeweils zwei Meßwerten nötig, um die Frage insgesamt beantworten zu können. Beantworten drei Personengruppen einen Fragebogen (Gruppe A, B, C), so kommt man auf insgesamt drei paarweise Vergleiche (A mit B; A mit C und B mit C). Allgemein gilt: Anzahl der Vergleiche = [Anzahl der Gruppen mal [Anzahl der Gruppen minus eins]] geteilt durch 2. So ergeben sich für vier Gruppen bereits: (4 x 3)/2 = 6 Vergleiche. Wenn die Fragestellung relativ offen formuliert ist und generell nach Unterschieden zwischen den Gruppen gefragt wird, so wächst die Wahrscheinlichkeit, einen Unterschied zu finden, je mehr Vergleiche möglich werden. Da man ja bei jedem Paarvergleich einen Alpha-Fehler von 5% begeht, summieren sich die Fehler von Paarvergleich zu Paarvergleich. Bei drei Vergleichen macht man also einen viel höheren Fehler, als bei nur einem. Höhere Fehler als 5% sind jedoch nach der oben angesprochenen Vereinbarung nicht signifikant. Um insgesamt nur auf einen Fehler von 5% zu kommen, müssen für jeden Einzelvergleich strengere Alpha-Fehler-Grenzwerte festgelegt werden. Für 3 Vergleiche ergibt sich z.B. ein Wert von 1,7%, bei vier Vergleichen sind es 1,3%, bei 10 Vergleichen 0,5%, usw. Eine Alternative für die Berechnung vieler Signifikanztests, die nur jeweils zwei Meßwerte vergleichen können ist die sogenannte Varianzanalyse ANOVA.
(Quelle: http://www.wu-wien.ac.at/inst/ivm/strunk/pdf/StatistikGlossar.pdf)

Alternativhypthese siehe Nullhypothese H_0

ANOVA auch Varianzanalyse (engl.: analysis of variance). In der Regel sind Signifikanztests in der Lage, nur zwei Meßwerte miteinander zu vergleichen. Einige Fragestellungen machen daher mehrere Vergleiche zwischen jeweils zwei Meßwerten nötig, um die Frage insgesamt beantworten zu können. Beantworten drei Personengruppen einen Fragebogen (Gruppe A, B, C), so kommt man auf insgesamt drei paarweise Vergleiche (A mit B; A mit C und B mit C). Obwohl es hier möglich ist jede Kombination der Gruppen einzeln zu vergleichen und eine Alpha-Fehler-Angepaßt vorzunehmen, ist eine Varianzanalyse eleganter und weniger aufwendig zu rechnen. Die Varianzanalyse löst das Problem durch einen Trick: Es werden im wesentlichen zwei Varianzen ermittelt und diese mit einem F-Test verglichen (s.auch F - Verteilung). Es werden also auch hier nur zwei Werte durch den Test verglichen. Die eine Varianz ist die innerhalb der Gruppen, die andere ist die zwischen den Gruppen. Sind die Unterschiede (also die Varianz) zwischen den Gruppen größer als die Unterschiede innerhalb

der Gruppen, so unterscheiden sich die Gruppen. Allerdings ist dann noch nicht bekannt, welche Gruppen sich voneinander unterscheiden. Um dies heraus zu finden werden anschließend doch wieder paarweise Vergleiche durchgeführt.
(Quelle: http://www.wu-wien.ac.at/inst/ivm/strunk/pdf/StatistikGlossar.pdf)

Anteilswert Eine relative Häufigkeit betrachtet die absolute Häufigkeit einer Ausprägung einer Variablen in Relation zur Gesamtzahl aller Untersuchungseinheiten. Diesen Quotienten bezeichnet man auch als Anteilswert (engl.: proportion). Unter methodischen Gesichtspunkten ist ein Anteilswert eine Gliederungszahl. Anteilswerte sind immer positiv. Ihr Wertebereich reicht von 0 bis 1. Der prozentuale Anteil entspricht dem Anteilswert multipliziert mit 100 und hat daher einen Wertebereich von 0 bis 100. Hat eine Variable insgesamt c verschiedene Ausprägungen (engl.: categories), dann gibt es $(c-1)$ voneinander unabhängige Anteilswerte, weil die Summe aller Anteilswerte notwendigerweise 1 ergibt und man daher immer einen Anteilswert durch Subtraktion der Summe der $(c-1)$ anderen Anteilswerte von 1 errechnen kann. Notation: Der Anteil einzelner Ausprägungen in der Stichprobe wird mit p abgekürzt, wobei die Variable und die jeweilige Ausprägung, deren Anteil gemessen wird, als Superskript bzw. als Index angegeben werden: z.B. p_k^X für den Anteil der Ausprägung $X = k$. Ist aus dem Kontext erkennbar, welche Variable betrachtet wird, verzichtet man in der Regel auf die Nennung der Variablen im Superskript und bezeichnet den Anteil mit p_k. Anteilswerte in der Grundgesamtheit werden mit θ (griech.: theta) bezeichnet. (Quelle: http://www.homes.uni-bielefeld.de/hjawww/glossar/)



arch effect Der „arch effect“ tritt als ein Artefakt bei Ordinationstechniken auf, bei der die zweite Achse eine Bogenfunktion (arch) der ersten darstellt. Dies ist auf die unimodale ( ) Verteilung von Arten entlang von Gradienten zurückzuführen. Der Bogen tritt dabei bei der Korrespondenzanalyse und anderen Ordinationstechniken auf. Die Detrended Correspondence Analysis (DCA) (auch Decoarana) beseitigt diesen Effekt. Bei der Faktorenanalyse PCA nennt man diesen Effekt auch den „horseshoe effect“. Der Algorithmus geht dabei so vor, daß der erste Gradient senkrecht in Segmente geteilt werde, die anschließend unterschiedlich gewichtet werden. In R ist die Voreinstellung bei der Funktion `decorana(...)` (package `vegan`): 1, 2, 3, 2, 1. Dieses unterschiedliche Wichten behebt den „arch effect“. Die andere Möglichkeit der „Enttrendung“ besteht darin Abschnittsweise eine Lineartransformation vorzunehmen und seltene Arten abzuwichten, da sie einen zu großen Einfluß auf die Ordination haben. Diese detrended - Methode sollte man jedoch mit Bedacht verwenden, da einige Autoren kritisieren, daß zu wenig Informationen über die Stärke der Glättung existieren. Aus der Hilfe von R (Funktion `decorana - vegan - package`): Nachdem die Achsen neu gewichtet werden, wird die Achse auf Einheiten der Standardabweichung skaliert, so daß bei der Betrachtung des Ordiantionsdiagrammes die Arten bezüglich des gesamten Gradienten zu beurteilen sind

arithmetisches Mittel Das arithmetische Mittel $\bar{x}$ einer Stichprobe wird berechnet, indem die Summe aller Werte durch die Anzahl aller Werte dividiert wird. Eine wichtige Eigenschaft von $\bar{x}$ ist, daß die Summe der quadrierten Abweichungen all der Werte aus denen das arithmetischen Mittel berechnet wurde, minimal ist. . Wird das arithmetische Mittel aus Sicht der Stochastik gesehen, so ändert sich seine Bezeichnung, es wird dann mit μ bezeichnet.

Autokorrelation Die Autokorrelation ist ein Begriff aus der Statistik und der Signalverarbeitung.
Im statistischen Modell geht man von einer geordneten Folge von Zufallsvariablen aus. Vergleicht man die Folge mit sich selbst, so spricht man von Autokorrelation. Da jede unverschobene Folge mit sich selbst am Ähnlichsten ist, hat die Autokorrelation für die unverschobenen Folgen den Wert 1. Wenn zwischen den Gliedern der Folge eine Beziehung besteht, hat auch die Korrelation der ursprünglichen Folge mit der verschobenen Folge einen Wert wesentlich größer als 0. Man sagt dann die Glieder der Folge sind autokorreliert. Quelle: http://de.wikipedia.org/wiki/Autokorrelation

B

Bayes Ansatz Bayes Statistik (Original siehe Bayes 1763) ist eine recht vielversprechende Richtung der Statistik, bei der es keine „schwarz/weiß Aussagen“ gibt, sondern „Graustufen“ und die Information der Unsicherheit *mit* einfließt. Dies wird dadurch erreicht, daß (allen) bekannten Ereignissen Wahrscheinlichkeiten zugerechnet werden. Es spielt sich also alles innerhalb von 0 bis 1 ab (beliebig komplex). Genereller Ablauf: Daten + Prior (=Annahmen) → Bayes Theorem → Posterior, wobei dann mit der Posterior-Verteilung modelliert wird. Klassische Statistik beschäftigt sich vielmehr (nur) mit dem linken Teil.

Bestimmtheitsmaß Das Bestimmtheitsmaß oder der Determinationskoeffizient wird in der Statistik dazu verwendet, den Zusammenhang von bei der Varianz-, Korrelations- und Regressionsanalyse untersuchten Daten-

reihen (Variablen) anzugeben. Es wird oft mit R^2 abgekürzt und liegt zwischen 0 (kein Zusammenhang) und 1 (starker Zusammenhang). Das Bestimmtheitsmaß ist das Quadrat des Pearson'schen Korrelationskoeffizienten. (s.Rangkorrelationskoeffizienten) Es gibt an, in welchem Maße die Varianz einer Variablen durch die Varianz einer anderen Variablen bestimmt wird. Das Bestimmtheitsmaß ist ein Maß für den linearen Zusammenhangs zweier Meßreihen bzw. Variablen. Ist $R^2 = 0.6$, so werden 60% der Varianz durch das Modell erklärt. Das Bestimmtheitsmaß sagt allerdings nichts über die Signifikanz des ermittelten Zusammenhangs aus. Dazu muß zusätzlich ein Signifikanztest durchgeführt werden.
Angepaßtes Bestimmtheitsmaß: $\overline{R^2} = \frac{n-1}{n-k}(1 - R^2)$ (http://de.wikipedia.org)
Ein Problem des Bestimmtheitsmaßes R^2 ist, daß dieses bei Hinzufügen eines weiteren, aber evtl. ungeeigneten Regressors, nicht kleiner werden kann. Das Angepaßte Bestimmtheitsmaß ($\overline{R^2}$) steigt dagegen nur, falls R^2 ausreichend steigt, um den gegenläufigen Effekt des Quotienten $\frac{n-1}{n-k}$ auszugleichen und kann auch sinken.
Auf diese Weise läßt sich $\overline{R^2}$ als Entscheidungskriterium bei der Auswahl zwischen zwei alternativen Modellspezifikationen (etwa einem restringierten und einem unrestringierten Modell) verwenden.
bias Beispiel aus (Köhler et. al 1996): das Körpergewicht einer Person wird mit einer Badezimmerwaage bestimmt. Ist die verwendete Waage alt und die Feder ausgeleiert, so wird wegen dieses systematischen Fehlers im Mittel ein zu hohes Körpergewicht angezeigt werden. Diese *systematische* Abweichung (Bias[34]) des gemessenen Mittelwertes vom wahren Körpergewicht wäre auf mangelnde Treffgenauigkeit zurückzuführen. Präzision hingegen ist die Streuung um den experimentellen Mittelwert.
BIC Das BIC (Bayesianisches Informationskritierium; nach engl.: Bayesian Information Criterion) ist ein Maß zur Beurteilung der „Güte" von multivariaten Modellen, die auf Maximum Likelihood-Schätzungen basieren (beispielsweise die logistische Regression und verwandte Verfahren). Es soll vor allem helfen, unterschiedliche nicht-geschachtelte Modelle (über den gleichen Datensatz!) untereinander zu vergleichen. (Nicht-geschachtelte Modelle sind solche, von denen sich nicht eines als „Unterfall" des anderen verstehen läßt, anders formuliert, von denen jedes mindestens eine Variable enthält, die in dem jeweils anderen Modell nicht enthalten ist.) Ein Modell paßt umso besser zu den Daten (oder umgekehrt), je kleiner der Wert des BIC ist. Eine Alternative zum BIC, die in jüngster Zeit jedoch manchmal als weniger brauchbar beurteilt wird, ist das AIC. (Quelle: http://www.lrz-muenchen.de/~wlm/ilm_b7.htm)
Binomialverteilung Die Binomialverteilung[35] beschreibt die Verteilung die entsteht, wenn das Ereignis, welches eintritt die Werte 0 und 1 annimmt. Ein Beispiel ist das Ziehen von roten und weißen Kugeln aus einer Urne, oder das werfen einer Münze. Da die Binomialverteilung nur diskrete Werte annehmen kann heißt sie auch *diskret*. Das Gegenteil wäre stetig, d.h. es können ∞ Werte angenommen werden. Siehe auch Verteilungen
Biplot Ein Biplot ist ein Diagramm, das bei Ordinationstechniken eingesetzt wird, um die Verteilung der Arten und Proben gleichzeitig im Graph zu plotten.
bootstrap Die Idee des bootstrap ist, daß man die n gemessenen Werte x zufällig unter Zurücklegen beprobt. Somit liegt dem neu entstehenden Datensatz (üblicherweise auch der Größe n) genau die gleiche Verteilung zugrunde, wie den Originaldaten. Von diesem neuen Datensatz x' können wir ebenfalls den Median berechnen. Und wenn wir diese Prozedur sagen wir 1000 mal wiederholen, können wir arithmetisches Mittel und die Varianz der Mediane berechnen. Aus den nach Wert geordneten bootstrap-samples können wir dann auch 95% Konfidenzintervalle ableiten (nämlich bei dem 25 und 975 Wert). In R gibt es für das bootstrap eine eigene Funktionen `bootstrap` im package `bootstrap`. Dies ist eine Implementierung der Ideen des Standardwerkes zum bootstrap (Quellen: Efron und Tibshirani 1993; Dormann und Kühn 2004). Der[36] bootstrap ist ein nichtparametrisches Verfahren.

C

CA Die Korrespondenzanalyse (CA) galt lange Zeit als Analyseverfahren für Zwei-Wege-Tafeln. Die CA dient vor allem der Dimensionsreduzierung des Datensatzes. In der ökologie wird die CA hauptsächlich für Speziesdaten, d.h. für Vorkommen oder Proportionen, verwendet. Die Deskriptoren müssen alle dieselben physikalischen Dimensionen aufweisen und die Zahlenwerte müssen ≥ 0 sein. Der Chi-Quadratabstand (s.auch Chi Quadrat (X^2) Distanz) wird zur Quantifizierung der Beziehungen zwischen den Objekten benutzt. Er wird im Ordinationsraum, ähnlich wie bei der PCA, beibehalten.

[34] auch „statistische Verzerrung"
[35] binomisch = zweigliedrig
[36] oder das?

Canberra Metrik Dieses Distanzmaß wird ebenso berechnet, wie die Manhattan-Metrik, jedoch werden die Punktekoordinaten durch die Anzahl der Objekte und Variablen gewichtet. s. Distanzmaße

CAP Constrained Analysis of Principal Coordinates ist eine Ordinationstechnik, die fast ähnlich mit der der linearen RDA ist. Der Unterschied besteht nur darin, daß andere Distanzmaße verwendet werden können statt der für Artendaten ungünstigen Euklid - Distanz. So z.B.: Sørensen, Jaccard u.a. Ist als Distanz `"euclidean"`, so ist das Ergebnis mit der RDA identisch.

CCA Die CCA (Kanonische Korrespondenzanalyse[37]) ist die Erweiterung der Korrespondenzanalyse (CA). Sie wurde vor allem für die Analyse von Speziesdaten entwickelt. Das Verfahren ist dem der RDA gleichzusetzen. Die CCA hat das Ziel, die Auswirkungen der Variablen der Spezies-Matrix Y durch sie beeinflussende Variablen (Matrix X) zu modellieren. Die Spezies in Y reagieren dabei entlang eines Gradienten *nichtlinear*.

CCorA Die Kanonische Korrelationsanalyse (engl.: canonical correlation analysis) untersucht die Korrelation zwischen zwei Matrizen. Wobei ein linearer Zusammenhang angenommen wird. Sie ist ähnlich der RDA, PCA und CCA, welche aber besser zu den Daten korrespondierenden als die CCorA (Legendre und Legendre 1998). In R mit `cancor(...)` aus dem Paket `stats`.

Chi Quadrat (X^2) Distanz Dieses Distanzmaß wird ebenso berechnet, wie die Euklid - Distanz, jedoch werden die Punktekoordinaten durch die Anzahl der Objekte und Variablen gewichtet. [38]
Bei Ordninationstechniken, wie CA und CCA wird dieses Distanzmaß ebenso verwendet. Führt aber dazu, daß seltene Arten zu stark gewichtet werden. Vermeiden läßt sich dies nur durch abwichten[39] seltener Arten oder durch Datentransformation. s.a. Distanzmaße

Chi^2 - Test Mit dem Chi-Quadrat-Test kann man einerseits eine beobachtete Häufigkeit gegen eine erwartete testen oder eine Häufigkeitsverteilung auf Homogenität prüfen. Es genügen Daten der Nominalskala. (Köhler et. al 1996)

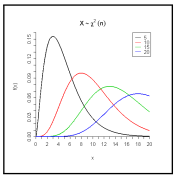
- *Fragestellung:* Weichen die beobachteten Häufigkeiten einer Stichprobe signifikant von erwarteten Häufigkeiten einer vermuteten Verteilung ab? Grober Ablauf: berechne aus den Daten den Wert $\chi^2_{Versuch}$ und vergleiche ihn mit dem Wert $\chi^2_{erwartet}$ bei den Parametern Freiheitsgrade FG[40]. Wenn $\chi^2_{Ver} \leqq \chi^2_{erw}$, dann gilt die Nullhypothese H_0 keine Abweichungen zwischen den Beobachtung und Erwartung. Falls $\chi^2_{Ver} > \chi^2_{erw}$, dann gilt H_A : beobachtete Häufigkeiten können nicht an die erwarteten angepaßt werden
- *Fragestellung:* ist das Stichprobenmaterial inhomogen, d.h. gibt es signifikante Unterschiede zwischen den Verteilungen in den Stichproben? Ablauf: s.o.; Nullhypothese H_0: das Material ist homogen, H_A : mindestens eine Stichprobe weicht ab

In R mit `chisq.test(x, y) - ctest/stats`

```
chisq.test(c(12, 20)) # Vgl. beobachtet <-> erwartet
                                    Hilfebeispiel
data(InsectSprays) # Not really a good example
chisq.test(InsectSprays$count > 7, InsectSprays$spray) # Ergebnis -> inhomogen
chisq.test(InsectSprays$count > 7, InsectSprays$spray)$obs # Ergebnis unter H0
chisq.test(InsectSprays$count > 7, InsectSprays$spray)$exp # Ergebnis für erwartete Hfg. unter
H0
```

Der Chi^2 - Test darf nur benutzt werden, wenn die Anzahl erwarteter Elemente einer Kategorie > 5. Sonst müssen man auf Fischers exakter Test zurückgreifen (`fisher.test(x,...)` Paket `ctest/stats`)

Chi^2 - Verteilung Die Chi-Quadrat-Verteilung ist eine stetige Wahrscheinlichkeitsverteilung. Sie hat einen einzigen Parameter, n, der eine natürliche Zahl sein muß. Man sagt auch n ist der



[37] lat. canonicus = regelmäßig; mlat. correspondere = übereinstimmen

[38] In einer Tabelle, wie:

	Art_1	Art_2	Art_3
$Fläche_1$	1	0	5
$Fläche_2$	23	2	4
$Fläche_3$	33	1	7

werden die Distanzen der Punkte ($Fläche_1$, Art_1); (..., ...) durch die Anzahl der Flächen und die Anzahl der Arten gewichtet. Verzerrungseffekte, die durch unterschiedliche Spaltenhäufigkeiten hervorgerufen werden, werden so eliminiert. Dennoch neigt dieses Maß dazu zahlenmäßig gering vorkommende Arten in den Daten überzubewerten. (Ist ein häufiges Distanzmaß bei vielen klassischen Ordinationstechniken.)

[39] z.B. durch das Argument `iweight` bei der Funktion `decorana(...)` im Paket `vegan`

[40] genau: FG gleich Anz. der Merkmalsklassen minus eins minus Anz. der aus den Daten geschätzten Parameter

Freiheitsgrad der Chi-Quadrat-Verteilung. In Symbolen:

$$X \sim \chi^2(n) \text{mit } n \in \mathbb{N}$$

(Die Zufallsgröße X ist ähnlich Chi^2 von n mit n Element der Natürlichen Zahlen). Die Formel zur Chi-Quadrat-Verteilung wird hier weggelassen.
Chord Distanz ist die Euklid - Distanz nachdem die Vektoren der Probestellen auf eins skaliert wurden. Formel

$$D_{Chord}(site_1, site_2) = \sqrt{2\left(1 - \frac{\sum_{spec_j=1}^{spec_p} a_{site_1} \cdot a_{site_2}}{\sum_{spec_j=1}^{spec_p} a_{site_1}^2 \cdot \sum_{spec_j=1}^{spec_p} a_{site_2}^2}\right)}$$

wobei a die Abundaz ist und j die Spalte ist. Diese Distanz hat ihr Maximum, wenn 2 Proben keine gemeinsamen Arten haben.
Cluster Analyse Verfahren

- Partitionierungsverfahren: ausgehend von k a-priori gegebenen Clustern werden diese so lange optimiert, bis sie untereinander so heterogen und innerhalb so homogen wie möglich sind.
  - k - means arbeitet iterativ bei a-priori bekannter Anzahl der Cluster k.
  - k - medoid funktioniert ähnlich wie k - means, nur werden nicht Mittelwerte benutzt, sondern repräsentative Werte der Stichprobe als Repräsentaten der Gruppen, die sog. „Medoide“Kaufman und Rousseeuw (1990); ist wohl auch robuster als k - means und nicht nur auf quantitative Daten anwendbar[41] . (in R s. `example(pam)`, Paket `cluster`).
- Hierarchische Cluster Verfahren:
  - agglomerative Beginnend mit n Clustern werden die ähnlichsten Cluster zusammengefaßt: Ward Clusteranalyse, Zentroid Clusteranalyse, Median - Clustering.
  - divisive Beginnend mit einem Cluster werden die Daten weiter aufgeteilt in immer heterogenere Cluster: nearest neighbor, complete linkage.
- Modell basierte Methoden: unter Annahme einer bestimmten Verteilung (und Anzahl von Clustern) wird die Zugehörigkeit zu einem Cluster mittels einer Wahrscheinlichkeit bestimmt. (s. Modell basiertes Clustering)
- Fuzzy Clustering: fordert keine 100% Zugehörigkeit zu einem bestimmten Cluster, sondern erlaubt die Zugehörigkeit zu α_i - %. (in R im Paket `cluster` die Funktion `fanny(...)`)

(Quelle http://stats.math.uni-augsburg.de/lehre/SS04/stat3.shtml), s.a. Distanzmaße

Vergleich der verschiedenen Verfahren

- Nachteil des Single-Linkage Verfahrens: Verkettungseigenschaft, sensitiv gegenüber Ausreißern, Vorteil: auch Cluster mit beliebiger Form (anstatt von Kreisen oder Ellipsen) können entdeckt werden
- Nachteil des Complete-Linkage-Verfahrens: kleine, kompakte Gruppen; Fusion zweier Gruppen unterbleibt
- Vorteile Zentroid und Average Linkage: space conserving
- Vorteile Ward, Average Linkage und Zentroid: Ausreißerrobust
- Ward Verfahren: findet tendenziell kreisförmige Cluster ähnlicher Größe
- Complete Linkage: tendenziell Cluster ähnlicher Größe und Gestalt, Nachteil: ausreißerempfindlich

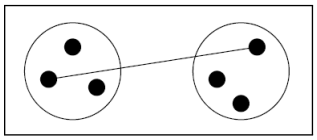
[41] http://www.quantlet.com/mdstat/scripts/mst/html/msthtmlnode85.html#36108

aus R Hilfe: A number of different clustering methods are provided. Ward's minimum variance method aims at finding compact, spherical clusters. The complete linkage method finds similar clusters. The single linkage method (which is closely related to the minimal spanning tree) adopts a 'friends of friends' clustering strategy. The other methods can be regarded as aiming for clusters with characteristics somewhere between the single and complete link methods.

Viele Studien empfehlen Ward und Average Linkage, dennoch können die Ergebnisse bezüglich der Performance der Verfahren von den Daten abhängen. Empfohlene Strategie: mehrere Alternativen testen, so zeigt sich auch, ob die Gruppen quasi „robuste" Gruppen sind.

Co - Inertia Die Co - Inertia Analyse untersucht die Beziehungen zweier Tabellen oder Matrizen miteinander. Man kann sie z.B. benutzen wenn man die Beziehungen zwischen gezählten Arten einerseits und gemessenen Umweltvariablen andererseits untersuchen will. s.a. Inertia

complete linkage – Farthest neighbor Auch hier wird aus jedem Cluster nur ein Objekt betrachtet. Dabei wird jedoch das Objektpaar ausgewählt, das die größte Distanz aufweist. Diese Distanz bildet dann den Abstand zwischen den beiden Clustern. (s.a.Cluster Analyse Verfahren)



conditioning variables legen Untergruppen des Datensatzes fest http://stats.math.uni-augsburg.de/lehre/WS04/SeminarPFDs/Trellis.pdf

constrained Ordination Unter Ordinationsmethoden, die als *constrained* [42] bezeichnet werden, versteht man jeweils die direkten (=kanonischen[37]) Ordinationsmethoden, wie RDA und CCA. Unconstrained bezeichnet die indirekten Ordinationsmethoden, wie PCA und CA. Man kann sich sozusagen vorstellen, daß die consrtained - Methoden zur Erklärung auf die Umweltfaktoren beschränkt sind, während die indirekten Methoden hypothetische Faktoren errechnen, die dann interpretiert werden müssen.

D

Datentransformation Die Transformation von Daten hat u.a. das Ziel, verschiedene Datenreihen vergleichbar zu machen oder den Daten Eigenschaften zu geben, mit denen sie besser analysiert werden können. Zu den gebräuchlichsten Transformationen gehören das Zentrieren, die Standardisierung, die Normierung und das Symmetrisierung. (Quelle: www.bio.uni-potsdam.de/oeksys/vstatoek.pdf)
Zu Datentransformation bei Ordinationstechniken s. Legendre und Gallagher (2001).

DCA Detrended Korrespondenz Analyse s. arch effect und CA

Decorana siehe arch effect

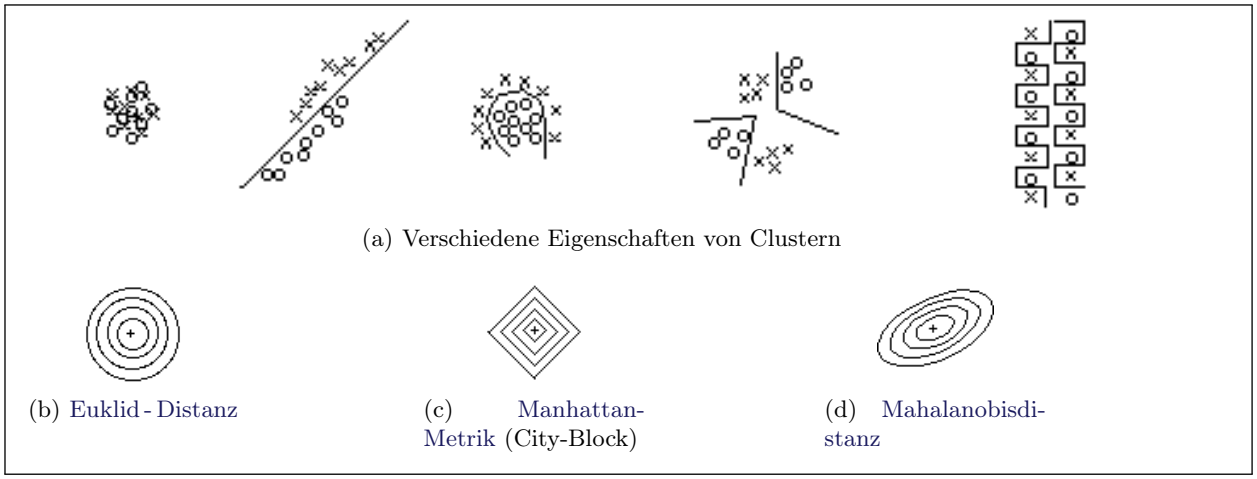
detrended siehe arch effect

deviance Summe der Abweichungsquadrate

Diskriminanzanalyse Wir betrachten ein Objekt und mehrere gleichartige Klassen. Das Objekt gehört einer dieser Klassen an, aber welcher, ist unbekannt. Mit Hilfe der Diskriminanzanalyse [43] ordnet man das Objekt einer der Klassen zu. Die Diskriminanzanalyse ist also ein Klassifikationsverfahren. An diesem Objekt kann mindestens ein statistisches metrisch skaliertes Merkmal x beobachtet werden. Dieses Merkmal wird im Modell der Diskriminanzanalyse als eine Zufallsvariable X interpretiert. Die Annahme also: es gibt mindestens zwei verschiedene Gruppen (Populationen, Grundgesamtheiten). Aus einer dieser Grundgesamtheiten stammt X. Mittels einer Zuordnungsregel, der Klassifikationsregel wird das Objekt einer dieser Grundgesamtheiten zugeordnet. Die Klassifikationsregel kann oft durch eine Diskriminanzfunktion angegeben werden.

Distanzmaße Gäbe es EIN angemessenes Proximitätsmaß, das alle Distanzen gut beschreibt, so gäbe es keinen Grund dieses nicht zu verwenden. Meistens jedoch unterliegen diese Distanzmaße zu vielen Fehlern:
die Eigenschaften können unzulänglich sein, um unterschieden zu werden, sie können hoch korreliert sein, die entscheidende Grenze könnte gekrümmt sein, es kann eindeutige Unterklassen in den Daten geben, die räumlichen Eigenschaften können einfach zu komplex sein.

[42] engl.: gezwungen, genötigt
[43] lat. discriminare = trennen, absondern



(a) Verschiedene Eigenschaften von Clustern

(b) Euklid - Distanz

(c) Manhattan-Metrik (City-Block)

(d) Mahalanobisdistanz

Abbildung 6: Verschiedene Eigenschaften von Clustern sowie Visualisierung der Distanzmessung unterschiedlicher Distanzmaße

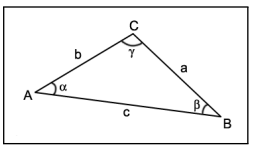
Tabelle 7: Distanzmaße und ihre Abhängigkeit zu verschiedenen Skalenniveaus.

	Metrisch	Intervall	Binär	`Berechnung`
Mahalanobisdistanz	✓	✓	✓	`dist.quant(df, method=3) - ade4`
Euklid - Distanz	✓	✓	✓	`dist(x, method = "euclidean") - mva/stats`
Quadrierte Euklid - Distanz	✓	✓	✓	`dist(x, method = "euclidean")^2 - mva/stats`
Manhattan-Metrik		✓	✓	`dist(x, method = "manhattan") - mva/stats`
Sørensen		(✓)	✓	`dist.binary(df, method=5) - ade4`
Jaccard			✓	`dist.binary(df, method=1) - ade4`
Ochiai/Kosinus		✓	✓	`dist.binary(df, method=7) - ade4`
Canberra Metrik		✓	✓	`dist(x, method = "canberra") - mva/stats`
Chi Quadrat (X^2) Distanz		✓	✓	

Tabelle 8: Eigenschaften verschiedener Distanzmaße und ihre Verwendung. (aus ter Braak 1995) ($*$ → qualitative Eigenschaft, $+/-$ → Sensitivität gegenüber bestimmten Eigenschaften)

	Sensitivität Gesamtproben	Sensitivität dominate Arten	Sensitivität Artendiversität	Ähnlichkeit	Unähnlichkeit	quantitativ	qualitativ
Kosinus (Ochiai)	*	*		*	+	+	−
Jaccard	*			*	++	−	−
Sørensen	*	*		*	+	−	−
Euklid - Distanz	*	*	*		++	++	++
Quadrierte Euklid - Distanz	*	*	*		+++	+++	+++

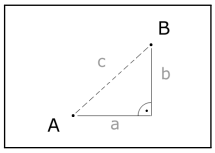
Dreiecksungleichung Nach der Dreiecksungleichung (engl.: triangle's inequality) ist im Dreieck die Summe der Längen zweier Seiten a und b stets größer oder gleich der Länge der dritten Seite c. Das heißt formal: $c \leqslant a + b$. Man kann auch sagen, der Abstand von `A` nach `B` ist stets kleiner oder gleich dem Abstand von `A` nach `C` und von `C` nach `B` zusammen, oder um es populär auszudrücken: „Der direkte Weg ist immer der Kürzeste." (http://de.wikipedia.org/wiki/Dreiecksungleichung)



dummy - Variable Die dummy - Variable ist eine Variable, die die die Werte 0 und 1 annimmt. 1 für z.B. „vorhanden", 0 „nicht vorhanden" oder dgl.

E

Eigenvektor Eigenvektoren eines linearen Operators (etwa durch eine Matrix dargestellt) sind Vektoren, auf welche die Anwendung des Operators (etwa die Multiplikation mit der Matrix) ein skalares Vielfaches ihrer selbst ergeben. Der Nullvektor kann definitionsgemäß nicht ein Eigenvektor sein. Den entsprechenden Skalar nennt man Eigenwert. Ist A eine (n, n) - Matrix, so heißt $\vec{x}$ ein Eigenvektor zum Eigenwert λ, wenn gilt: $A \cdot \vec{x} = \lambda \cdot \vec{x}$
Eigenwert Der Eigenwert λ ist der Varianzanteil, der durch einen (hypothetischen) Faktor j erfaßt wird. Der Eigenwert eines Faktors j berechnet sich als Summe der quadrierten Ladungen eines Faktors. Siehe auch PCA
Erwartungswert Der Mittelwert einer Zufallsvariablen oder einer Verteilung wird Erwartungswert μ genannt. Mit der Varianz σ^2 gehört der Erwartungswert zu den Parametern, die eine Zufallsvariable oder eine Verteilung charakterisieren.
Euklid - Distanz In einem zweidimensionalen Zahlenraum läßt sich die direkte Distanz zwischen zwei Punkten nach dem Satz von Pythagoras[44] als Hypotenuse eines „gedachten" rechtwinkligen Dreiecks berechnen. Dieses Distanzmaß ist verglichen mit anderen Distanzmaßen mit einigen Schwächen behaftet: es tendiert dazu Ausreißern mehr Wichtung zu verleihen, als bei Sørensen und verliert an Sensitivität, wenn die Heterogenität des Datensatzes zunimmt. Siehe Distanzmaße



Anm.: die bei den Ordinationstechniken PCA und RDA ebenso verwendete Euklidische Distanz ist ungeeignet sobald viele Arten in der Abundanztabelle mit Nullwerten vorkommen. Hilfreich kann dann eine Datentransformation sein (Legendre und Gallagher 2001).

F

Faktor Einflußgrößen, die im Versuchsplan berücksichtigt und erfaßt werden, heißen Faktoren.
Fehler 1. und 2. Art Bei unserer Test-Entscheidung zwischen Nullhypothese H_0 und Alternativhypthese kann es durchaus passieren, daß eine „unglückliche" Stichprobenzusammenstellung uns veranlaßt die Nullhypothese H_0 zu verwerfen, obwohl sie in Wirklichkeit richtig ist. Einen solchen Fehler bezeichnet man als Fehler 1.Art oder Alpha-Fehler. Neben einer unberechtigten Ablehnung der Nullhypothese H_0 (Fehler 1.Art) ist es ebenso

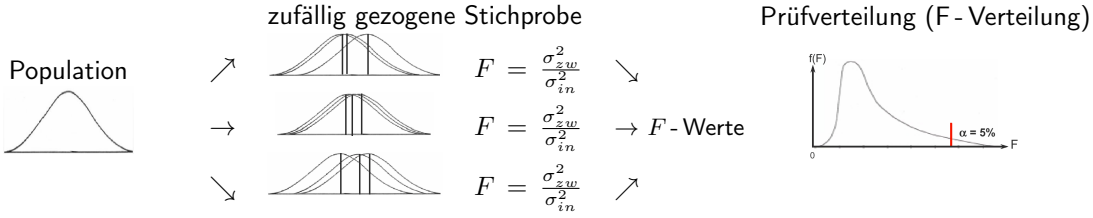
[44] Gleichung: $c = \sqrt{a^2 + b^2}$

möglich, daß man die Nullhypothese H_0 beibehlt, obwohl sie in Wirklichkeit falsch ist, dies nennt man einen Fehler 2.Art oder β Fehler. (Köhler et. al 1996)
Fischers exakter Test Ein besonders sicherer Test ist Fischers exakter Test, da er kaum an Voraussetzungen gebunden ist und immer berechnet werden kann, wenn es um den Vergleich zweier Prozentzahlen/Häufigkeiten geht. Eine Berechnung durch einen Computer setzt jedoch meist voraus, daß insgesamt nicht mehr als 1000 Personen befragt wurden, da bei der Berechnung extrem hohe Zahlen als Zwischenergebnisse auftreten. Neben der exakten Variante dieses Tests gibt es für große Stichproben daher auch Näherungsformeln über den t - Test, die jedoch mit Vorsicht zu genießen sind. Fischers exakter Test liefert ohne weitere Kennwerte die Wahrscheinlichkeit für die Übereinstimmung der beiden Prozentzahlen. Die Wahrscheinlichkeit ist das Ergebnis des Tests. Man spricht von einer statistischen Signifikanz, wenn diese Wahrscheinlichkeit kleiner als der vorher festgelegte Alpha-Fehler ist. Nullhypothese H_0 beide Gruppen sind gelich (odds - ratio = 1) – in R `fisher.test(...), ctest` oder `stats`
(Quelle: http://www.wu-wien.ac.at/inst/ivm/strunk/pdf/StatistikGlossar.pdf)

```
Zähldaten. 12 gegen 4 testen
m <- matrix(c(12, 4, 4, 12), nr=2)
fisher.test(m)
```

Fixed Point Cluster Analyse Diese Clusteranalyse ist eine neue Methode für nicht-hierarchische Clusteranalysen sie arbeitet ähnlich einer Ausreißeranalyse. Das Ziel dabei ist, Gruppen von Punkten iterativ über ein stochastisches Modell zu finden. Dabei wird kein globales Modell für den gesamten Datensatz angenommen. Vielmehr versucht diese Methode Untergruppen so zusammenzufügen, daß sie keine Ausreißer enthalten. Cluster unterschiedlicher Form oder überlappende Cluster können gefunden werden. Dabei wird pro Cluster die lineare Regression verwendet. Diese Methode kann für p–dimensional metrische Daten, 0-1-Vektoren und Daten zur linearen Regression angewendet werden. (Christian 2002)
Freiheitsgrad In der statistischen Prüftheorie die Höchstzahl der in einem mathematischen System, z.B. in einer Prüfverteilung, frei bestimmbaren Variablen, die variiert werden können, ohne daß die Bedingungen des Systems gestört sind. Es wird dadurch die Gesamtzahl der insgesamt möglichen und daher nicht mit Sicherheit voraussehbaren Ausprägungen der Daten einer Prüfverteilung angegeben. Handelt es sich also um eine Verteilung von insgesamt k Beobachtungen, so beträgt die Höchstzahl der frei variierbaren Werte k - 1, weil der letzte Wert durch alle vorangegangenen Werte festgelegt ist. Bei Stichproben hängt die Zahl der Freiheitsgrade von der Zahl der Operationen ab, die zur Schätzung des entsprechenden Werts erforderlich waren. Die Berechnung der Freiheitsgrade ist erforderlich, damit der kritische Wert berechnet werden kann und so durch Vergleich des empirischen Prüfwerts mit dem kritischen Wert eine Aussage über die Signifikanz des empirischen Werts möglich ist. (auch *degree of freedom*)
Friedman - Test Der Friedman - Test führt eine einfaktorielle Varianzanalyse durch, um zu prüfen, ob die k Faktorstufen systematische Unterschiede aufweisen. Im Gegensatz zur Varianzanalyse setzt man keine Normalverteilung voraus und man kann den Test auch bei ordinalskalierten Daten anwenden (Köhler et. al 1996). H_0 : sind die Mediane gleich. Der Friedman - Test benutzt die Chi ^{2} - Verteilung, indem er den
F - Test Der F - Test verwendet zum Testen nicht den Ansatz, daß die Lageparameter (arithmetisches Mittel) von Variablen mit Normalverteilungen verglichen werden, sondern er schaut sich die Unterschiede in den Streuungen an. Mit anderen Worten in den Varianzen. Daher kann man mit dieser Art von Test prüfen ob sich aus statistischer Sicht Wechselwirkungen zwischen zwei oder mehr Variablen aufdecken lassen. Voraussetzungen, um diesen Test anzuwenden, sind: die Stichproben seien aus Grundgesamtheiten, die der Normalverteilung gleichen; die Varianzen σ^2 seien für alle Stichproben gleich (also $\sigma_1^2 = \sigma_2^2 = ...$), die Stichproben seinen unabhängig mit gleichem Stichprobenumfang $n > 1$. Die Nullhypothese H_0 ist: die Effekte (i.w.S. Unterschiede) zwischen zwei oder mehr Faktoren sind gleich null, so daß die Mittelwerte μ_i alle gleich sind. Mathematisch: $\mu_1 = \mu_2 = ... = \mu_k$; die Wechselwirkungen zwischen den Faktoren sind null. Anm.: der F - Test wird auch verwendet beim Modelltest der Regressionsanalyse. Test auf Normalverteilung: Shapiro-Wilk Test (`shapiro.test(x)` Paket `ctest/stats` für $n = 3...5000$, H_0 : die Streuung von x gleicht der, der Normalverteilung – Bsp.: $P = 0.004$, dann ist x NICHT normalverteilt) oder Kolmogorov-Smirnov-Test (testen, wenn 2 Variablen unabhängig sind, `ks.test(x, y)` Paket `ctest/stats`; H_0 : x und y sind aus der selben Verteilung).
F - Verteilung Die F Verteilung ergibt sich aus der Verteilung des Verhältnisses zweier Varianzschätzungen zueinander. Mit ihrer Hilfe werden die Wahrscheinlichkeiten bei der Varianzanalyse berechnet. Vergleicht man nun verschiedene Stichproben miteinander, so lassen sich Unterschiede aufzeigen, indem man sich die Varianz

(also die Streuung) zwischen den Stichproben (Sigma σ^2_{zw}) und die Varianz innerhalb der gesamten Stichprobe σ^2_{in} vergleicht. Dabei werden sogenannte F - Werte berechnet, die dann mit einer Prüf - F - Verteilung verglichen werden. (s. Schema; siehe auch Verteilungsanpassungstest)



G

Gammaverteilung Die Gammaverteilung ist wie die Normalverteilung eine stetige Verteilung. D.h. sie kann Werte von 0,..., ∞ annehmen. Das Gegenteil wäre eine diskrete Verteilung, die nur zählbare Werte annehmen kann, wie Binomialverteilung und Poissonverteilung. Ein Beispiel für die Gammaverteilung ist die Lebensdauer von Systemen. Siehe auch Verteilungen

GLM – (von engl.: general linear models oder Allgemeines lineares Modell)Das Ziel von statistischen Methoden ist es eine *Zielvariable*, auch *Response* genannt (hier mit Y bezeichnet), in Abhängigkeit von unabhängigen *Kovariablen* ($x_1 ... x_p$) darzustellen. Ein *lineares Modell* besitzt die allgemeine Form:

$$Y_i = \beta_0 + \sum_{i=1}^{p} x_{ik}\beta_k + \varepsilon_i \quad \text{mit} \quad i = 1, ..., n.$$

Oder in Matrixschreibweise: $Y = X\beta + \varepsilon_i$ mit der *Response* Y, dem unbekannten *Regressionsparameter* β, dem *Kovariablenvektoren* X (s.Kovariable) und einem *Vektor der Fehlerterme* ε. Es gilt: der Erwartungswert für den Vektor der Fehlerterme ε ist null, man schreibt $\mathrm{E}[\varepsilon] = 0$ und die Varianz des Vektorfehlerterms ε ist für die Beobachtungen $\mathbb{I}_n$ *homogen*, man schreibt $\mathrm{Var}[\varepsilon] = \sigma^2\mathbb{I}_n$. Außerdem nimmt man an, daß die Fehlerterme ε_i $(i = 1, \ldots, n)$ *normalverteilt* sind. Kurz zusammen gefaßt kann man auch schreiben $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ für $i = 1, \ldots$. Unter diesen Annahmen ist auch Y normalverteilt! Es gilt für den Erwartungswert E von Y und die Varianz Var von Y :

$$\mu \overset{\text{def}}{=} \mathrm{E}[Y] = \beta_0 + \sum_{j=1}^{p} \beta_j x_j = X\beta \quad \text{mit} \quad \mathrm{Var}[Y] = \sigma^2\mathbb{I}_n$$

Die klassischen linearen Modelle können verallgemeinert werden, indem man die (häufig unbefriedigende und in der Praxis unrealistische!) Normalverteilungsannahme der Y_i $(i = 1, \ldots, n)$ aufhebt und stattdessen auch Verteilungen von (allgemeineren) *Exponentialverteilungen* zuläßt. Zu den Exponentialverteilungen zählen alle Verteilungen, deren Dichteverteilung auf die Form

$$f_{\theta,\Phi}(y) = \exp\left[\frac{\langle y, \theta\rangle - b(\theta)}{a(\Phi)} + c(y, \Phi)\right]$$

gebracht werden kann, wobei ϕ der natürliche Parameter und Φ der Dispersionsparameter ist. (Spezialfälle hiervon sind die Normalverteilung und die Poissonverteilung.)

Darüber hinaus wird bei den verallgemeinerten linearen Modellen der *lineare Prädiktor* $\eta \overset{\text{def}}{=} X\beta$ über die sogenannte *Linkfunktion* g mit dem Erwartungswert μ verbunden, man schreibt das: $\eta = g(\mu)$. Die Linkfunktion wird zumeist als umkehrbar und differenzierbar vorausgesetzt. Gilt $\eta = \theta$, wobei θ (wie gesagt) der natürliche Parameter der Exponentialverteilung ist, dann spricht man von einem *kanonischen Link*. Im Gegensatz zu den klassischen linearen Modellen wird zudem die Varianz nicht mehr als konstant angenommen. Wenn man zum Beispiel annimmt, daß Y poissonverteilt (s.Poissonverteilung) ist (wie ja häufig im (Sach-)Versicherungsbereich oder bei Zähldaten), dann gilt: $\eta = \theta = \log(\mu)$ denn: $\mu = e^\theta$, das heißt, man hat die Linkfunktion $g(\mu) = \log(\mu)$. (Quelle: http://www.matheraum.de/read?t=10829&v=t)

Tabelle 9: Varianzfunktion und kanonische Linkfunktion wichtiger GLM mit dem Erwartungswert μ und dem linearen Prädiktor eta $X\beta \stackrel{\text{def}}{=} \eta$

Verteilung d. Fehler	Varianzfkt. $V(\mu)$[45]	Kanonische Linkfunktion	Modellgleichung	in R
Binomialverteilung	$\mu\,(1\,-\,\mu)$	$\eta\ =\ ln\,(\mu\,/(1\,-\,\mu))$	$y = \frac{e^{\beta x+a}}{1+e^{\beta x+a}}$	`binomial()`
Poissonverteilung	μ	$\eta\ =\ ln(\mu)$	$y = e^{\beta x+a}$	`poisson()`
Normalverteilung	1	$\eta\ =\ \mu$	$y = \beta x + a$	`gaussian()`
Gammaverteilung	μ^2	$\eta\ =\ 1/\mu$	$y = \frac{1}{\beta x+a}$	`Gamma()`
Invers - Normal	μ^3	$\eta\ =\ 1/\mu^2$	$y = \frac{1}{\sqrt[2]{\beta x+a}}$	`inverse.gaussian()`

Grundgesamtheit Als Grundgesamtheit bezeichnet man die Menge aller Objekte, Individuen oder Ereignisse, die bzgl. eines Merkmals untersucht werden. D. h. die Grundgesamtheit wird gebildet durch alle Objekte, Individuen oder Ereignisse, die überhaupt zur betrachteten Menge gehören können. Aus der Grundgesamtheit wird eine möglichst repräsentative **Stichprobe** ausgewählt, die dann bezüglich bestimmter Variablen untersucht wird. Sie stellt also nur einen Teil der „Wirklichkeit“ dar. Maßzahlen der Stichprobe bekommen lateinische Buchstaben ($\bar{x}$, s...) oder mit „Dach“ ($\hat{p}$, $\hat{\lambda}$), Maßzahlen der Grundgesamtheit hingegen bekommen griechische Buchstaben (μ, σ,...) oder ohne „Dach“ (p, λ).
Als **abhängige** Variable (response Variable) bezeichnet man diejenige Variable, deren Werte durch eine oder mehrere andere Variable bestimmt werden. Diese heißen entsprechend **unabhängige** Variablen (erklärende oder Prädiktorvariable).

H

Hauptfaktorenanalyse Im Gegensatz zur PCA unterstellt die Hauptfaktorenanalyse, daß man nur einen bestimmten Varianzanteil durch die Faktoren erklären kann der restliche Varianzanteil teilt sich sozusagen auf. Die Hauptfaktorenanalyse (Modell mit mehreren *gemeinsamen* Faktoren) nimmt an, daß die Varianz einer Variable zu zerlegen ist:

a) in den Anteil, den diese Variable mit den restlichen Variablen gemeinsam hat (gemeinsame Varianz) und

b) Anteil, der allein auf die spezifische Variable und den bei ihr auftretenden Meßfehler zurückzuführen ist (merkmalseigene Varianz).

Nicht die gesamte Varianz, sondern allein die gemeinsamen Varianzen der Variablen sollen durch das Modell der gemeinsamen Faktoren erklärt werden. Das Problem dabei ist die Schätzung der gemeinsamen Varianz. Ähnlich wie bei der Hauptkomponentenanalyse bezieht man nur die ersten k Hauptfaktoren in die Modellschätzung ein. Der Rest wird der Matrix zugerechnet. Die beiden Schritte der Hauptfaktorenanalyse, d.h. Kommunalitätenschätzung und Komponentenanalyse der Matrix, können auch iterativ wiederholt werden. Dazu werden die nach der ersten Schätzung erhaltenen Werte für die Matrix dazu verwendet, wieder eine reduzierte Matrix zu berechnen, die dann wiederum zu neuen Schätzungen für die Ladungsmatrizen führt. Die Iterationen werden solange fortgeführt, bis beim n-ten Schritt ein Abbruchkriterium erfüllt ist oder die voreingestellte Zahl von Iterationsschritten erreicht ist.
horseshoe effect Siehe arch effect.

I

Inertia ist ein Maß für die totale Varianz in einem Datensatz. Sie steht direkt in Beziehung zu dem physikalischen Konzept (auch in Ökosystemen so), daß ein Objekt, welches die Tendenz hat in Bewegung zu

[45] Die Varianzfunktion beschreibt den Einfluß des Erwartungswerts auf die Varianz der Responsevariablen.

sein auch in Bewegung bleiben möchte. Ebenso bei Objekten mit stehender Tendenz. Bei den unimodalen Ordinationstechniken (DCA & CCA) ist die Inertia eher als Spannweite der Art um ihren häufigsten Wert (Modalwert) oder Optimum im Ordinationsraum zu verstehen als die Varianz der Artenabundanz. http://www.okstate.edu/artsci/botany/ordinate/glossary.htm

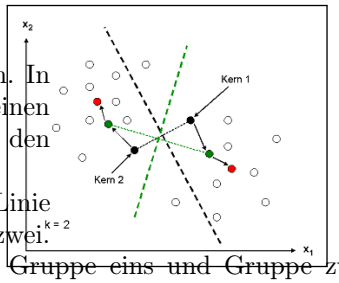
Intervallskala Intervallskalen sind metrische Skalen, in denen über den Unterschied zweier Meßwerte ausgesagt werden kann, ob er größer, gleich oder kleiner als der Unterschied zweier anderer Meßwerte ist. Das bedeutet: Skalenwerte einer Intervallskala können bezüglich ihrer Differenzen (und Summen) verglichen werden. Erst auf dem Niveau von Intervallskalen ist die Addition oder Subtraktion von Meßwerten sinnvoll und erlaubt. Beispiel: Temperatur in Grad Celsius : Die Differenz zwischen den Temperaturen 7 und 10 C ist genauso groß wie die Temperaturdifferenz zwischen 20 und 23 C°. Für viele psychologische Skalen wird Intervallskalenniveau angestrebt (z. B. Persönlichkeits- und Intelligenztests). Siehe auch Skalenniveau.

J

Jaccard Dies ist ein Index, in welchem gemeinsam fehlende Größen aus der Betrachtung ausgeschlossen werden. Übereinstimmungen und Nichtübereinstimmungen werden gleich gewichtet. – für Binärdaten, s. Distanzmaße

jackknife Es gibt noch ein „Gegenstück“ zum bootstrap, das jackknife. Dieses mißt die Güte eines Schätzers (z.B.: Mittelwert, Median – allg.: θ^*) anhand seiner Sensibilität gegenüber Datenveränderungen, indem die jackknife-Prozedur jeden Datenpunkt einzeln entfernt, und dann den Schätzer θ_i^* neu berechnet. Schließlich wird dann der jackknife-korrigierte Schätzer (θ_{jack}^*) berechnet (Dormann und Kühn 2004). In R gibt es für das jackknife eine eigene Funktion `jackknife` im package `bootstrap`.

K



k - means Bei diesem Verfahren wird die Anzahl der Cluster vorgegeben. In seiner einfachsten Version arbeitet das Verfahren wie folgt. Gegeben seinen $k \geq 1$ Cluster und n Datensätze, sowie eine Abstandsfunktion zwischen den Daten. Für $k = 2$ wird folgendermaßen vorgegangen:
1. Zwei (k) Kernpunkte werden zufällig ausgewählt (•). Die schwarze Linie (⋮) ist die geometrische Grenze zwischen Gruppe eins und Gruppe zwei.
2. Berechnen der Zentren bzw. des Durchschnittes (= *means* - Teil) von Gruppe eins und Gruppe zwei (•). Gruppengrenze ist jetzt (⋮). **3.** Neuer Kernpunkt ist •. Dazu wird neues Zentrum berechnet (•). Dieser Vorgang wird sooft wiederholt, bis sich alle k Kernpunkte stabilisieren, d.h. nicht mehr verschieben. (Quelle: http://www.bilyap.com/dwhd/kmeans.php?ottrid=zzQH7IoYEI)
Anmerkung: Größtes Problem von k-means ist die Wahl der Anzahl der Cluster. Die Wahl der Startwerte (Seeds) kann das Ergebnis des Algorithmus entscheidend beeinflussen. ⇒ Oft werden daher die Ergebnisse anderer Clustermethoden zur Bestimmung der Seeds benutzt. Durch die Mittelwertbildung können auch Cluster-Zentren entstehen, die mehr oder weniger weit von den eigentlichen Clustern liegen. ⇒ k-means ist nicht sehr robust. (s.a.Cluster Analyse Verfahren)

k - medoid PAM funktioniert ähnlich wie k - means. Es werden für eine gegebene Anzahl von k Clustern zunächst k Repräsentanten, sog. „medoids“, aus der Menge aller Daten gesucht. Die Medoids werden so ermittelt, daß die Summe der Abstände der Daten zu ihrem jeweils nächstgelegenen Medoid minimal ist. Nach Bestimmung der k Repräsentanten werden k Cluster gebildet, indem jeder Wert seinem nächstgelegenen Medoid zugeordnet wird. (Quelle: http://www.stat.uni-muenchen.de/~strimmer/publications/diplom-lampert.pdf). Dieses Verfahren wird als robuster beurteilt (aus R-Hilfe). siehe auch Cluster Analyse Verfahren

Kolmogorov-Smirnov-Test Mit dem Kolmogorov-Smirnov-Test `ks.test(...)` kann man testen, ob zwei numerische Datenvektoren X und Y von derselben Verteilung stammen. (Nullhypothese H_0 Kommen x und y aus derselben Verteilung?)

```
library(ctest)
x <- rnorm(1000)
y <- runif(1000)
ks.test(x,y) #
```

Was auf Grund der Wahl der Vektoren X und Y von vornherein schon klar war, wird auch durch den Test

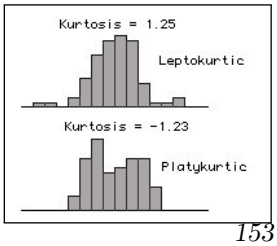
bestätigt: X und Y stammen offensichtlich nicht aus derselben Verteilung.
Andererseits kann man mit dem Kolmogorov-Smirnov-Test jedoch auch überprüfen, ob ein Datenvektor X aus einer ganz bestimmten Verteilung stammt. Dazu wird der Vektor Y durch die Bezeichnung der Verteilungsfunktion und die Paramter dieser Verteilung ersetzt. Bsp: Ist X gammaverteilt mit shape 1 und scale 2?

```
x<-rnorm(1000)
ks.test(x, "pgamma", 3,2)
```

In beiden Beispielen wurde ein zweiseitiger Test durchgeführt (Voreinstellung in R). Möchte man einen einseitigen Test durchführen, gibt es je nachdem die Optionen alternative="less"und alternative="greater", also z.B. `ks.test(x,y, alternative="greater")`. ☞ Da der Kolmogorov-Smirnov-Test für manche Situationen keine exakten p - Werte berechnet, da er Testparameter aus den Daten ermittelt, sollte man den Shapiro-Wilk Test auf Normalverteilung ($=H_0$) vorziehen.
Kommunalität Die Kommunalität gibt an, in welchem Ausmaß eine Variable i durch die Faktoren aufgeklärt bzw. erfaßt wird. Sie berechnet sich als Summe der quadrierten Ladungen einer Variablen.
Korrelation Maß des Zusammenhangs [„co-relation“] zwischen zwei oder mehr Variablen. Es existieren - in Abhängigkeit vom Meßniveau der zugrundeliegenden Daten eine Vielzahl von Korrelationskoeffizienten. Sie beschreiben Richtung und Stärke eines Zusammenhangs. Ohne theoretische Begründung sind Rückschlüsse auf die Verursachung eines Zusammenhangs nicht zulässig. Bsp.: Korrelation zwischen Störchennestern und Geburtenzahlen. s. auch Korrelationsmatrix
Korrelationsmatrix Die Korrelationsmatrix beschreibt die Abhängigkeit zwischen zwei Zufallsvariablen x_i und y_j . Im Gegensatz zur Kovarianz (Kovarianzmatrix) mißt die Korrelation auch die Stärke der Abhängigkeit. Zur Berechnung der Korrelation zwischen y_j und x_i werden die Deskriptoren standardisiert und danach wird die Kovarianz ermittelt. Damit kann eine Korrelation auch zwischen unterschiedlich dimensionierten Deskriptoren ermittelt werden. (Tabelle Pearsonscher Korrelationskoeffizient r und seine Einstufung). In R berechnet man die Korrelationsmatrix mit `cor(...)`.

Bereich von r	Beziehung zwischen x_i und y_j
-1, 1	perfekt negativ, positiv korreliert
> -1 bis -0.7	stark negativ korreliert
> -0.7 bis -0.3	schwach negativ korreliert
> -0.3 bis 0.0	nicht signifikant korreliert in Abhängigkeit von n
0.0 bis 0.3	nicht signifikant korreliert in Abhängigkeit von n
> 0.3 bis 0.7	schwach positiv korreliert
> 0.7 bis < 1	stark positiv korreliert

Kovariable Die Kovariable (auch Begleitvariable) bezieht sich auf die Variable (Umweltvariable), die bei der Berechnung ausgeklammert werden soll[46]. Dies ist entweder eine störende oder eine wichtige Variable, die bei einer Fragestellung nicht von unmittelbarem Interesse ist und herausgerechnet werden soll.
Kovarianzmatrix Die Kovarianz `cov(`x`,` y`)` beschreibt den Grad des miteinander Variierens (oder Kovariierens) zweier Meßwertreihen x und y. Die Kovarianz ist die Summe der gemittelten Abweichungsprodukte zweier Variablen. Nachteilig ist, daß sie abhängig ist von den Maßeinheiten der gemessenen Variablen. Positive Kovarianz: hohe x-Werte entsprechen hohen y-Werten, negative Kovarianz: hohe x-Werte entsprechen niedrigen y-Werten, keine Kovarianz: kein Zusammenhang zwischen x und y-Werten. – `cov(...)` – s.auch Korrelationsmatrix;
Kreuzvalidierung Form der Validitätsprüfung durch Replikation; d.h. die an einer Stichprobe gewonnenen Befunde werden zur Absicherung an einer zweiten, von der ersten unabhängigen Stichprobe erneut überprüft.
Kruskal - Wallis - Test Dieser Test führt eine einfaktorielle Varianzanalyse durch, um festzustellen, ob zwischen den k Faktorstufen signifikante Unterschiede auftreten, oder ob man davon ausgehen muß, daß alle Stichproben aus der gleichen Grundgesamtheit stammen. Für $k = 2$ kann man auch den Wilcoxon - Test verwenden. Im Gegensatz zur Varianzanalyse mit F - Test setzt man hier keine normalverteilten Grundgesamtheiten voraus, zudem genügen ordinalskalierte Daten. *Fragestellung:* entstammen die k Stichproben aus mindestens zwei verschiedenen Grundgesamtheiten? *Voraussetzungen:* die $k \geqq 3$ Grundgesamtheiten sollen stetige Verteilungen von gleicher Form haben, die Stichproben seien unabhängig und die Daten mindestens ordinalskaliert. H_0 : gleiche Grundgesamtheit. `kruskal.test(x, ...)` im Paket `ctest/stats`
Kurtosis Die Kurtosis ist ein Maß für die Art der Verteilung an den Rändern (Seiten). Verteilungen mit stark ausgeprägten Rändern (hohen Seitenwerten) werden



[46] Ist nicht dasselbe wie löschen!!

leptokurtic (leptos = dünn) genannt, Verteilungen mit wenig ausgeprägten Seiten heißen *platykurtic* (platys = breit). Eine Verteilung, die dieselbe Kurtosis wie die Normalverteilung aufweist, wird *mesokurtic* genannt. Die nebenstehenden Verteilungen haben dieselbe Varianz, ungefähr dieselbe Schiefe, aber eine ganz unterschiedliche Kurtosis. Eine Normalverteilung hat die Kurtosis von **0**.
In R mit dem Paket `fBasics` ab v1.9: `kurtosis(rnorm(1000)), kurtosis(runif(1000))`; (s.a.Skewness)

L

Likelihood-Funktion (engl.: Likelihood Function). Die im Rahmen der Maximum Likelihood-Schätzung verwendete Funktion, gibt an, welche(r) geschätzte(n) Parameter bei gegebenen Daten die größte Wahrscheinlichkeit aufweist, dem wahren Parameter in der Grundgesamtheit zu entsprechen. Die L. kann aber aber aus formalen Gründen nicht als Wahrscheinlichkeitsfunktion aufgefaßt werden, weshalb es auch sinnvoll ist, in der deutschen Sprache bei dem englischen Namen zu bleiben, um Verwechslungen vorzubeugen.
Da die L. ein Produkt vieler Einzelwahrscheinlichkeiten ist, ist sie numerisch schwer zu handhaben. Daher ist es sinnvoll, den Logarithmus der L., meist als LL (Log-Likelihood) abgekürzt, zu maximieren; häufig wird stattdessen auch -LL minimiert. Ich erwähne dies deshalb, weil man den Output von Computerprogrammen genau darauf hin prüfen muss, welcher Wert ausgegeben wird. Häufig ist dies auch nicht LL oder -LL, sondern 2 * LL oder 2 * -LL. Die Gründe dafür finden sich beim Likelihood-Verhältnis-Test. (Quelle: http://www.lrz-muenchen.de/~wlm/ilm_l2.htm)
Likelihood-Verhältnis-Test (auch: Likelihood-Quotienten-Test, Likelihood-Ratio Test [engl.]) Der L.-V.-T. ist ein relativ allgemein einsetzbares Verfahren zum Vergleich von Modellen auf der Grundlage der Maximum Likelihood-Schätzung. Verglichen werden jeweils zwei Modelle:
Ein Ausgangsmodell, welches i.a. mehrere Modellparameter enthält, und ein Vergleichsmodell, in welchem einem oder mehreren dieser Parameter Restriktionen auferlegt wurden. (Das Ausgangsmodell heißt daher auch unrestringiertes Modell, das Vergleichsmodell restringiertes Modell; diese Begriffe sind aber immer relativ zum jeweiligen Test-Ziel zu verstehen). Der Vergleich dient stets der Prüfung, ob das unrestringierte Modell tatsächlich (signifikant) „besser“ ist als das restringierte, d.h. einen besseren Fit aufweist. Ist das nicht der Fall, ist das restringierte Modell, weil einfacher (und dennoch hinsichtlich der Erklärungskraft nicht schlechter), vorzuziehen. Folgende „Restriktionen“ wären z. B. denkbar (dies dürften die wichtigsten praktischen Anwendungsbedingungen sein):

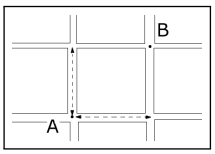
- Alle Parameter werden auf Null gesetzt. Dies ist eine Prüfung, ob das unrestringierte Modell insgesamt mehr „erklärt“ als rein durch Zufallsschwankungen (im Rahmen der Stichprobenziehung) zu erwarten wäre. Dieser Test entspricht dem F-Test auf Signifkanz des Gesamtmodells in der linearen Regressionsanalyse. Er wird von vielen Statistik-Paketen standardmäßig bei der Modellschätzung ausgegeben.
- Ein Parameter wird auf Null[47] gesetzt. Dies ist eine Prüfung, ob die betreffende Variable einen statistisch signifikanten Einfluß auf die abhängige Variable hat. Diese Prüfung ist anderen Statistiken (etwa mittels der Wald-Statistik oder der t-Statistik) überlegen.
- Mehrere Parameter werden auf Null gesetzt. Hier soll geprüft werden, ob eine Gruppe von Variablen einen statistisch signifikanten Einfluß auf die abhängige Variable hat.
- Zwei oder mehr Parameter sollen identisch sein (oder eine bestimmte, vorgegebene Differenz aufweisen). Hiermit kann geprüft werden, ob die Beträge zweier (oder mehrerer Parameter) sich in statistisch signifikanter Weise voneinander unterscheiden bzw. ihre Differenz einen bestimmten Betrag über- bzw. unterschreitet.

(Quelle: http://www.lrz-muenchen.de/~wlm/ilm_l7.htm)
Logarithmustransformation s. Symmetrisierung

M

[47] Man beachte: In der Praxis heißt „Parameter auf Null setzen“ nichts anderes als ein Modell zu schätzen, in welchem die entsprechenden Variablen weggelassen werden. Andere Restriktionen (wie die zuletzt genannte) sind nicht in allen Statistikpaketen standardmäßig implementiert.

Mahalanobisdistanz Einige Einschränkungen der Euklid - Distanz können durch die sog. Mahalanobisdistanz behoben werden. Insbesondere dann, wenn die Merkmale einen zu kleinen Maßstab haben und/oder hoch korreliert sind. Die Mahalanobisdistanz ein Maß, das angibt wie weit die unabhängigen Variablen vom Durchschnitt aller Klassen abhängen. Eine große Mahalanobisdistanz steht für die Fälle, die extreme Werte von *einer* oder von *mehreren* unabhängigen Variablen aufweist. Die Mahalanobisdistanz beseitigt einige Einschränkungen der Euklid - Distanz:
es berücksichtigt automatisch die Skalierung der Koordinatenachsen (es ist skaleninvariant), es behebt Korrelationen zwischen unterschiedlichen Merkmalen, es kann ebenso verwendet werden, wenn die Grenze zwischen den Merkmalen linear oder gekrümmt verläuft. Die Vorteile, die dieses Maß bietet, haben aber ihren Preis: die Kovarianzmatrix [48] kann schwer bestimmbar sein und der Speicherbedarf sowie der Zeitaufwand nehmen im quadratischen Maße zu, wenn die Anzahl der Merkmale steigt. Dieses Problem ist sicher unbedeutend, wenn nur wenige Merkmale geclustert werden sollen, verschärft sich aber bei vielen Merkmalen. In der Diskriminanzanalyse wird die Zuordnung eines Punktes zu einer bestimmten gegebenen Population unter anderem mit der Mahalanobis-Distanz bestimmt. s.a Distanzmaße



Manhattan-Metrik (auch City-Block-Metrik) Dies ist ein metrisches System, basierend auf einem Grid. Die Entfernung zwischen zwei Punkten wird definiert bezüglich eines rechtwinkligen Abstandes oder der Anzahl von Gridzellen in jeder Richtung. Bei diesem Distanzmaß bleiben Korrelationen zwischen den Merkmalen unberücksichtigt und hohe Unterschiede werden stark gewichtet. (Ist ein Spezialfall der sog. Minkowski -Distanz.), s.a Distanzmaße

Mann-Whitney-U-Test Besteht der Verdacht, daß die Voraussetzungen für einen t - Test verletzt sein könnten, kann am besten der U-Test von Mann und Withney berechnet werden.
(Quelle: http://www.wu-wien.ac.at/inst/ivm/strunk/pdf/StatistikGlossar.pdf)

Mantel - Test Der Manteltest vergleicht die Ähnlichkeit zweier Distanzmatritzen. Er kann in R mit `mantel(...)` aus dem package `vegan` durchgeführt werden. die Nullhypothese H_0 lautet: die Matrizen sind verschieden. Siehe auch Prokrustes-Test.

Maximum Likelihood-Schätzung ((von engl. maximale Wahrscheinlichkeit)) Statistisches Schätzverfahren, das eigentlich aus der Stochastik kommt. Die Logik ist etwa diese. Gegeben sind Daten einer Stichprobe und Annahmen über die Verteilung der relevanten Variablen. Wir prüfen nun, bei welchem (oder welchen) Parameter(n) in der Grundgesamtheit die gegebenen Daten am wahrscheinlichsten sind; der betreffende Wert gilt dann als bester Schätzer für den oder die Parameter. Es muß also das Maximum einer Funktion gefunden werden, die sich auf diese Wahrscheinlichkeiten bezieht, daher der Name Maximum Likelihood. Die betreffende Funktion heißt Likelihood-Funktion. Was sehr abstrakt klingt, hat manchmal praktische Folgen:

1. Die Likelihood-Funktion kann bei manchen Verteilungen mehrere (lokale) Maxima haben. Hier ist nicht sichergestellt, daß tatsächlich das absolute Maximum gefunden wird.

2. Manchmal hat (bei gegebenen Daten) die Likelihood-Funktion tatsächlich kein Maximum und die (iterative) Schätzung konvergiert nicht. Manche Programme teilen den Benutzern mit, wenn dieser Fall vorzuliegen scheint; andere (so SPSS) behelfen sich teilweise mit (nicht näher erläuterten) Tricks (man kann dann Parameter, die eigentlich gar nicht geschätzt werden können, an übergroßen Standardfehlern erkennen).

(Quelle: http://www.lrz-muenchen.de/~wlm/ilm_m3.htm)

Median Der Median oder auch Zentralwert einer Verteilung ist der Wert, der eine nach ihrer Größe geordnete Rangreihe halbiert. Der Median ist der Wert, von dem alle übrigen Werte so abweichen, daß die Summe der Absolutbeträge ein Minimum ergibt. Bei geradzahligem N liegt er zwischen den beiden Meßwerten. Der Median setzt mindestens Ordinalskalenniveau voraus. Der Median wird auch als „50. Zentil“ bezeichnet; er liegt immer zwischen dem arithmetischen Mittel (s. arithmetisches Mittel) und dem Modalwert, wenn er nicht mit ihnen zusammenfällt. Er eignet sich also auch gut bei sehr asymmetrischen Verteilungen, Verteilungen mit offenen Klassen und bei Ordinalskalierung (s. Ordinalskala).

Median - Clustering Diese Methode ähnelt dem Zentroid Clusteranalyseing, es besteht jedoch folgender Unterschied: Bei der Zentroid - Methode ergibt sich der Zentroid eines neuen Clusters als gewogenes Mittel aus den beiden Zentroiden der Ausgangs - Cluster, wobei die Fallzahlen der Ausgangscluster die Gewichte bilden.

[48] Streuungsmatrix der Zeilen- und Spaltenwerte

Beim Median - Clustering wird der Zentroid eines neuen Clusters dagegen als arithmetisches (ungewichtetes) Mittel der beiden Zentroide der Ausgangscluster berechnet. (s.a.Cluster Analyse Verfahren)
Medoid siehe k - medoid
Minimum Spanning Tree ... ist eigentlich so eine Art Minimalgerüst: so als ob ein Postbote verschiedene Orte abfahren muß, aber nur den kürzesten Weg zurücklegen darf.
Die Berechnung minimaler Spannbäume findet direkte Anwendungen in der Praxis, wenn man zum Beispiel kostengünstig zusammenhängende Netzwerke (z.B. Telefonnetzwerke, elektrische Netzwerke u.a.) herstellen will oder bei Computernetzwerken mit Redundanz, wo das Spanning Tree Protocol zur Anwendung kommt.
In der Graphentheorie selbst sind MST-Algorithmen häufig Grundlage komplexerer Algorithmen für schwierigere Probleme. Die Berechnung minimaler Spannbäume ist zum Beispiel Bestandteil von Approximationsalgorithmen für das Steinerbaum-Problem oder für das Problem des Handlungsreisenden (oft auch Traveling-Salesman-Problem genannt und TSP abgekürzt). nach http://de.wikipedia.org
MMDS Metrische Multidimensionale Skalierung[49] siehe PCoA. Es gibt 2 Typen: metrische und nichtmetrische MDS.

- eine MDS, die auf gemessenen Näherungswerten[50] beruht wird Metrische Multidimensionale Skalierung genannt (hier MMDS) `cmdscale(`*stats*`)`
- eine MDS, die auf Beurteilungswerten[51] basiert, nennt man nichtmetrische Multidimensionale Skalierung (hier NMDS), da sie eben für nicht-metrische Werte verwendet wird. `sammon(`*MASS*`)`, `isoMDS(`*MASS*`)`

Bei der metrischen MDS gibt die räumliche Anordnung die Unähnlichkeit der Objekte wieder – je weiter weg, desto verschiedener –, während die nichtmetrische MDS die Ordnung der Ränge anhand ihrer Unähnlichkeit repräsentiert. Quelle: Guide to Advanced Data Analysis using IDAMS Software P.S. NAGPAUL, New Delhi (India) http://www.unesco.org/webworld/idams/advguide/TOC.htm
Modalwert Der Modus oder Modalwert ist der am *häufigsten* in einer Verteilung vorkommende Meßwert. Haben wir in einer Verteilung nicht einen, sondern zwei oder mehr Modalwerte, die nicht nebeneinander liegen, spricht man von einer bi - bzw. multimodalen Verteilung. Bei Häufigkeitsverteilungen mit Klassen ist der Modalwert die Mitte derjenigen Klasse, die am häufigsten vorkommt. Ein Vorteil des Modus: er kann leicht erkannt werden aus der Häufigkeitstabelle oder Graphik. Ein Nachteil: je nach Stichprobe fällt er unterschiedlich aus; auch innerhalb einer Stichprobe verändert er sich je nachdem, wie viele Klassen eingerichtet werden und wie breit diese sind. Der Modus kann für Daten jeden Skalenniveaus bestimmt werden.
Modell basiertes Clustering Frage: wieviele Cluster gibt es? Die Idee gründet auf der Annahme, daß die Daten aus k unabhängigen Populationen entstammen, deren Gruppenzuordnung jedoch nicht mehr bekannt ist. Wären die Gruppenbezeichner γ_i bekannt, und Gruppe i hätte Dichte $f_i(x_i, \theta)$, dann ist die Likelihood

$$\prod_{i=1}^{n} f_{\gamma_i}(x_i, \theta)$$

Da die Gruppenbezeichner γ_i unbekannt sind, und somit als Parameter angesehen werden müssen, wird die Likelihood-Funktion über (θ, γ) maximiert. (zu θ s.Anteilswert)
Quelle: http://stats.math.uni-augsburg.de/lehre/SS04/CA1.pdf
Monte - Carlo - Test Ein Synonym dafür ist auch Randomisationstest.
Multikolinearität Das Vorliegen einer gegenseitigen Abhängigkeit der erklärenden Variablen einer multiplen Korrelations- oder Regressionsgleichung, d.h. eine hohe Korrelation der erklärenden Variablen untereinander. Beispiel: Die Produktion von Kieselalgen in einem See hängt z.B. von den Faktoren Temperatur, pH - Wert, Carbonatgehalt, Sonnenscheindauer, Trübung, ... Vermutlich werden viele dieser Variablen zusammenhängen, das heißt, hoch miteinander korrelieren (= Multikolinearität).
Multiple lineare Regression Klassisches Regressionsverfahren bei denen mehr als eine Variable („multiple“) in die Kalibriergleichung aufgenommen wird. Die Auswahl der Variablen wird per Hand (step up) oder programmgesteuert (stepwise) vorgenommen.

[49] Anm.: hier herrscht etwas Konfusion, da manchmal sowohl die Metrisch Multidimensionale Skalierung als auch die Nichtmetrische Multidimensionale Skalierung mit MDS abgekürzt wird. Um dies zu vermeiden wurden hier die Abkürzungen MMDS und NMDS verwendet.
[50] von proximities übersetzt
[51] Ränge: z.B. 1, 2, 3, 4 od ja-nein

N

nearest neighbor – single linkage Aus jedem der beiden Cluster wird nur ein Objekt betrachtet. Es werden die beiden Objekte ausgewählt, zwischen denen die geringste Distanz besteht. Diese Distanz wird als Distanz zwischen den beiden Clustern angesehen. Nachteil dieses Verfahrens Verkettungseigenschaft und sensitiv gegenüber Ausreißern. (s.a.Cluster Analyse Verfahren)

nichtparametrisch Ein Test, der keine Verteilungsannahme der Daten braucht, um durchgeführt zu werden bezeichnet man als nichtparametrischen Test oder auch „verteilungsfrei“

NMDS Nichtmetrische Multidimensionale Skalierung[52] betrachtet die Ähnlichkeit bzw. Verschiedenheit von n Objekten und versucht diese in einem möglichst niederdimensionalen Raum (meist k = 1,2,3) so anzuordnen, daß die Ähnlichkeit bzw. Verschiedenheit möglichst gut wiedergegeben wird.
Bei der Bestimmung der Konfiguration der Punkte zueinander verwendet die NMDS einen iterativen Prozeß. Die Grundidee dieses Prozesses ist relativ simpel: alle Objekte werden zunächst mehr oder weniger willkürlich im Raum angeordnet. Im nächsten Schritt werden die Distanzen zwischen den Objekten mit den Ähnlichkeiten verglichen (wobei das Skalenniveau der Ähnlichkeiten berücksichtigt wird). Wenn nun zwei Objekte im Verhältnis zu ihrer Ähnlichkeit zu weit auseinanderliegen, werden sie aufeinander zu geschoben. Sollten zwei eher unähnliche Objekte zu nahe bei einander liegen, werden sie voneinander weg bewegt. Dieser Vorgang wird so lange fortgesetzt, bis die Konfiguration der Objekte die erhobenen Ähnlichkeiten zufriedenstellend widerspiegelt. Dabei muß vorher festgelegt werden, wieviel Dimensionen der Raum haben soll (http://www.wiwi.uni-wuppertal.de/kappelhoff/papers/mds.pdf, Ablauf s. Abb. 7 auf der nächsten Seite)

[52] Anm.: hier herrscht etwas Konfusion, da manchmal sowohl die Metrisch Multidimensionale Skalierung als auch die Nichtmetrische Multidimensionale Skalierung mit MDS abgekürzt wird. Um dies zu vermeiden wurden hier die Abkürzungen MMDS und NMDS verwendet. Meist ist mit MDS die NMDS gemeint.

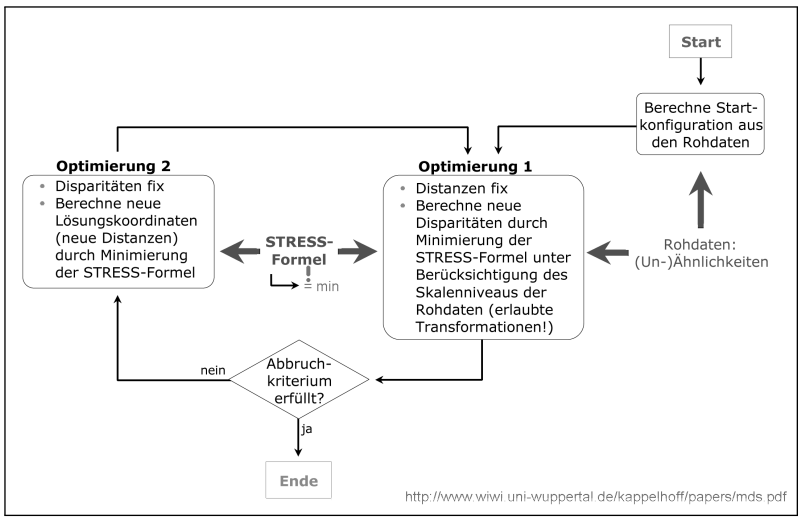
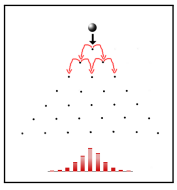


Abbildung 7: Iterationsprozess der NMDS im Überblick: **Optimierung 1:** Zunächst wird geprüft, wie gut die Konfiguration an die Ausgangsdaten angepaßt ist. Dabei werden die Rohdaten je nach Skalenniveau einer bestimmten Transformation unterzogen. Die daraus resultierenden Disparitäten[53]werden mit den Distanzen verglichen. Der Unterschied zwischen Disparitäten und Distanzen wird als STRESS ausgegeben. Wenn der STRESS der Konfiguration klein genug ist oder sich nicht mehr wesentlich verändert hat, wird nach Optimierungsschritt 1 die Iteration abgebrochen und das Ergebnis der MDS aus-gegeben. **Optimierung 2:** in diesem Optimierungsschritt werden die Objekte in der Konfiguration verschoben. Diese Verschiebung erfolgt wiederum auf Grundlage des Unterschiedes zwischen Disparitäten und Distanzen. Durch die resultierende verbesserte Anpassung der Distanzen an die Disparitäten wird der STRESS erneut verringert. Auf Optimierung 2 folgt wieder Optimierung 1, wodurch sich der STRESS in der Regel weiter verringert. Neben dem STRESS wird ein weiteres Maß als Gütekriterium für die Anpassung der Konfiguration an die Rohdaten betrachtet. Es handelt sich hierbei um R^2 (auch abgekürzt als RSQ). R^2 ist die quadrierte Korrelation der Distanzen mit den Disparitäten und stellt ein Maß für die lineare Anpassung der Disparitäten an die Distanzen dar. R^2 wird auch als Varianz der Disparitäten interpretiert, die durch die Distanzen erklärt wird. In der Praxis gelten R^2-Werte größer als 0,9 als akzeptabel. (http://www.wiwi.uni-wuppertal.de/kappelhoff/papers/mds.pdf)

Ein Unterschied zu den Eigenwertmethoden (PCA,PCoA, oder CA) besteht in der Weise, daß sie die Variabilität auf die Achsen maximieren. Beginnend mit der 1. Achse mit dem höchsten Erklärungsanteil an der Gesamtvariabilität. Bei der NMDS sind die Achsen hingegen beliebig. D.h. man kann die ganze Ordination drehen, zentrieren, invertieren.
In R gibt es die Funktion: `isoMDS(...)` – `MASS` - Paket.
Nominalskala Die Nominalskala setzt nur die Gleichheit oder Ungleichheit von Eigenschaften (z. B. Geschlecht) bzw. die Möglichkeit mehrklassiger Einteilungen (etwa in Berufe, Muttersprache, Haarfarbe, Studienrichtung...) in Kategorien voraus. Diese Kategorien müssen exakt definiert, sich gegenseitig ausschließend und erschöpfend sein. Die einzig erlaubte Rechenoperation ist Zählen, d. h. es wird festgestellt, ob eine Merkmalsausprägung überhaupt vorhanden ist und wenn ja, wie häufig sie auftritt. Siehe auch Skalenniveau.
Normalverteilung Die Bedeutung der Normalverteilung für die sozialwissenschaftliche empirische Forschung leitet sich aus der Tatsache ab, daß viele sozialwissenschaftliche, psychologische und biologische Merkmale zumindest annäherungsweise normalverteilt sind. Die Theorie, daß die Normalverteilung vieler Merkmale in einem Naturgesetz begründet liege, wird heute eher abgelehnt. Interessanterweise verteilen sich auch Zufallsverteilungen (beispielsweise das Galton - Brett Abb. rechts) oder Meßfehler normal, sofern eine genügend große Stichprobe zugrundeliegt. Aus diesen Beobachtungen leiten sich eine Reihe an statistischen Kennwerten und Prüfverfahren ab.



Anhaltspunkte für eine Normalverteilung sind gegeben, wenn das Verhältnis arithmetisches Mittel zu Median

[53] = Ungleichheiten, Unterschiede [lat. disparatum „abgesondert, getrennt“]

annähernd Eins ist bzw. wenn die Schiefe (Skewness) annähernd Null ist.
Die Dichtefunktion einer Normalverteilung mit Mittelwert 103 und Standardabweichung 2 ist (in R durch`hist(rnorm(1000, mean = 103, sd = 2), density = 30, freq = F); lines( density(103, 2), col = "red"))`

$$f(x) = \frac{1}{2 \cdot \sqrt{2 \cdot Pi}} \cdot e^{-\frac{(x-103)^2}{2 \cdot 2^2}}$$

man schreibt das mathematisch so auf: $N(103, 2)$ oder ganz allgemein: eine Zufallsvariable X ist normalverteilt mit dem Mittelwert μ und der Standardabweichung σ: $X \sim N = (\mu, \sigma)$
Test auf Normalverteilung: Shapiro-Wilk Test (`shapiro.test(x)` Paket `ctest/stats` für $n = 3...5000$, H_0 : die Streuung von x gleicht der, der Normalverteilung – Bsp.: $P = 0.004$, dann ist x NICHT normalverteilt) oder Kolmogorov-Smirnov-Test (testen, wenn 2 Variablen unabhängig sind, `ks.test(x, y)` Paket `ctest/stats`; H_0 : x und y sind aus der selben Verteilung).
Normierung Die Normierung dient der Relativierung von Datenreihen. Dabei werden die Daten in den Bereich zwischen Null und Eins gebracht, indem man sie z.B. auf ihren maximalen Wert skaliert. s.auch Datentransformation
Nullhypothese H_0 Ein statistischer Test ist ein Verfahren zur Überprüfung einer Annahme oder Hypothese über die Wahrscheinlichkeitsverteilung einer Zufallsvariable aufgrund einer Stichprobe. Dabei hilft der Test zu entscheiden, zwischen welchen beiden Hypothesen man sich guten Gewissens entscheiden darf[54]. Zeigt ein Test keine Signifikanz an, dann „behält“ man die Nullhypothese bei[55]. Zeigt er Signifikanz an (i.A. Alpha-Fehler = 5%=p), dann entscheidet man sich für die Alternativhypothese. Zum Beispiel heißt es beim Kolmogorov-Smirnov-Test: Test auf Gleichverteilung zweier Verteilungen. Das bedeutet H_0 = Gleichverteilung H_A ungleiche Verteilungen. Gibt der Test $p = 0.023$ aus, so entscheidet man sich für H_A, d.h. die zwei getesteten Verteilungen sind verschieden. Tipp: Ist es manchmal ganz unklar, was Nullhypothese oder Alternativhypothese ist, rechnet man einfach mit identischen Proben.

O

odds - ratio Die Odds und Odds Ratio sind eine Möglichkeit, Anteilswerte in Kreuztabellen auszudrücken und zu vergleichen. Man kann „Odds“ mit „Chancen“ und „Odds Ratio“ mit „relative Chancen“ übersetzen, es hat sich aber (bislang) auch in der deutschen Sprache eher der englische Begriff eingebürgert. Das ist auch deshalb sinnvoll, weil „relative Chancen“ leicht mit „relative Risiken“ verwechselt werden kann (was etwas anderes ist!). Betrachten wir die Tabelle: Übergewicht in Abhängigkeit vom Geschlecht. Wir können nun sagen: Die „Chancen“, daß eine Frau kein Übergewicht hat, betragen 60:40 oder 1,5. (Umgekehrt kann man auch sagen, daß die „Chancen“, Übergewicht aufzuweisen, 40:60 oder 0,66 betragen). Die „Chancen“ von Männern, kein Übergewicht aufzuweisen, betragen dagegen nur 30:70 oder 0,43. Grundsätzlich zeigt sich, daß ein Wert der Odds von genau 1 ein Verhältnis von 50:50 ausdrückt, Werte >1 drücken aus, daß die Kategorie im Zähler, Werte <1, daß diejenige im Nenner den größeren Anteil aufweist. Die Odds Ratio ist nun ein Maß für die Stärke des Unterschieds zwischen zwei Gruppen, hier Frauen und Männern. Die Odds Ratio setzt einfach die Odds der beiden Gruppen zueinander ins Verhältnis. Im Beispiel beträgt die Odds Ratio 1,5 : 0,43 = 3,5. D.h., die Chancen von Frauen, nicht übergewichtig zu sein, sind 3,5 mal so groß wie die von Männern. Die Odds Ratio kann daher als Zusammenhangsmaß aufgefaßt werden. Eine O.R. von 1 bedeutet, daß es keinen Unterschied in den Odds gibt, ist die O.R. >1, sind die Odds der ersten Gruppe größer, ist sie <1, sind sie kleiner als die der zweiten Gruppe. Odds und Odds Ratios lassen sich immer nur in Bezug auf zwei Ausprägungen ausdrücken. In größeren als 2x2-Tabellen können dementsprechend mehrere Odds und Odds Ratios berechnet werden.

	Frauen	Männer	alle
Kein Übergewicht	60%	30%	45%
Übergewicht	40%	70%	55%
N	100	100	200

(Quelle: http://www.lrz-muenchen.de/~wlm/ilmes.htm)
Ordinalskala Ordinalskalen (Rangskala) sind Skalen, in denen ausgesagt werden kann, welche Beziehung zwischen den Meßwerten bestehen, d. h. ordinalskalierte Meßwerte können bzgl. ihrer Größe in einer Rangreihe geordnet werden. Man nennt daher die Skalenwerte einer Ordinalskala auch Ränge. Die Operationen „größer“, „kleiner“ und „gleich“ sind hier erlaubt. Beispiel: Plazierungen/ Vgl.v. Rundenzeiten beim Sport; ungetestete

[54] auch als Testproblem bezeichnet
[55] Die Bezeichnung „behält“ steht hier deswegen, weil der Test immer von der Nullhypothese ausgeht

Fragebögen; Schulnoten, obwohl sie häufig wie intervallskalierte Daten behandelt werden. Siehe auch Skalenniveau.

Ordinationstechnik Ordinationstechniken versuchen Ordnung in das multivariate Chaos zu bringen ;-). Siehe Tabelle 10 und Abbildung 8 auf der nächsten Seite.

Tabelle 10: Zusammenfassung der Ordinationstechniken. Indirekte Analysen errechnen *hypothetische* Faktoren, die in den Daten liegen. Direkte Analysen beziehen z.B. Umweltdaten direkt in die Berechnung mit ein (daher direkt). Die direkten Methoden werden als „Redundancy Analysis" (RDA), „Canonical Correspondence Analysis" (CCA) und die „detrended" Variante: „Detrended Canonical Correspondence Analysis" bezeichnet. (verändert nach Legendre und Legendre 1998)

Responsemodell	Faktoranalysemethode			Distanzmaß	Variablen
	indirekt	direkt	partiell		
linear	PCA	RDA CAP	pRDA	Euklid - Distanz beliebig	quantitativ
Euklidraum	PCoA = MMDS			beliebig	quantitativ, semiquantitativ, qualitativ, oder gemischt
Euklidraum	NMDS			beliebig	quantitativ, semiquantitativ, qualitativ, oder gemischt
unimodal	CA	CCA	pCCA	Chi Quadrat (X^2) Distanz	nicht-negativ, dimensionshomogen Quantitativ- oder Binär-Daten, Abundanzen oder 0/1-Daten
unimodal - detrended	DCA	DCCA		Chi Quadrat (X^2) Distanz	- " -

P

p-2-seitig Das Ergebnis eines Signifikanztests ist im wesentlichen die Wahrscheinlichkeit dafür, daß sich zwei Meßwerte nicht voneinander unterscheiden. Da Wahrscheinlichkeit auf Englisch Probability heißt, wird sie mit dem Buchstaben „p" abgekürzt. p kann jedoch grundsätzlich auf zwei verschiedene Arten berechnet werden. p kann 1-seitig oder auch 2-seitig bestimmt werden. Welche der beiden Berechnungen im Einzelfall anzugeben ist, entscheidet sich durch die Fragestellung, die mit dem Signifikanztest beantwortet werden soll. Eine zweiseitige Fragestellung prüft, ob zwischen zwei Meßwerten ein Unterschied besteht, ohne genauer darauf einzugehen, welche Richtung der Unterschied hat (ob der eine Meßwert größer als der andere ist oder ob das Umgekehrte zu erwarten ist, wird nicht berücksichtigt). Eine einseitige Fragestellung prüft nicht nur, ob allgemein ein Unterschied besteht, sondern zudem, ob er in die erwartete Richtung geht. Der 2-seitige Wert wird also bei ungerichteten Signifikanztests angegeben. Er ist immer exakt doppelt so hoch wie der entsprechende 1-seitige Wert. Der 1-seitige Wert hat es damit „leichter" signifikant zu werden, erfordert aber die genauere Vorhersage.
(Quelle: http://www.wu-wien.ac.at/inst/ivm/strunk/pdf/StatistikGlossar.pdf)

parameterfreie Verfahren berücksichtigen nur die Rangfolge der Daten, daher können auf diesem Weg auch ordinalskalierte oder nicht normalverteilte, intervallskalierte Daten getestet werden. Bei den parameterfreien Verfahren gibt es voneinander unabhängige (unverbundene) Stichproben und voneinander abhängige (verbundene Stichproben). Zu den unverbundenen zählen der Kruskal - Wallis - Test und der Nemenyi - Test. (Köhler et. al 1996)

parametrisch Ein Test wird als parametrisch bezeichnet, wenn ihm eine Verteilung zugrunde liegt.

partielle Analysen Unter Partieller Analyse (z.B. Regression, Korrelation, ANOVA, Ordination) versteht man allgemein das Herausrechnen der Effekte von Kovariablen, um den Einfluß der übrigen Variablen besser zu sehen.

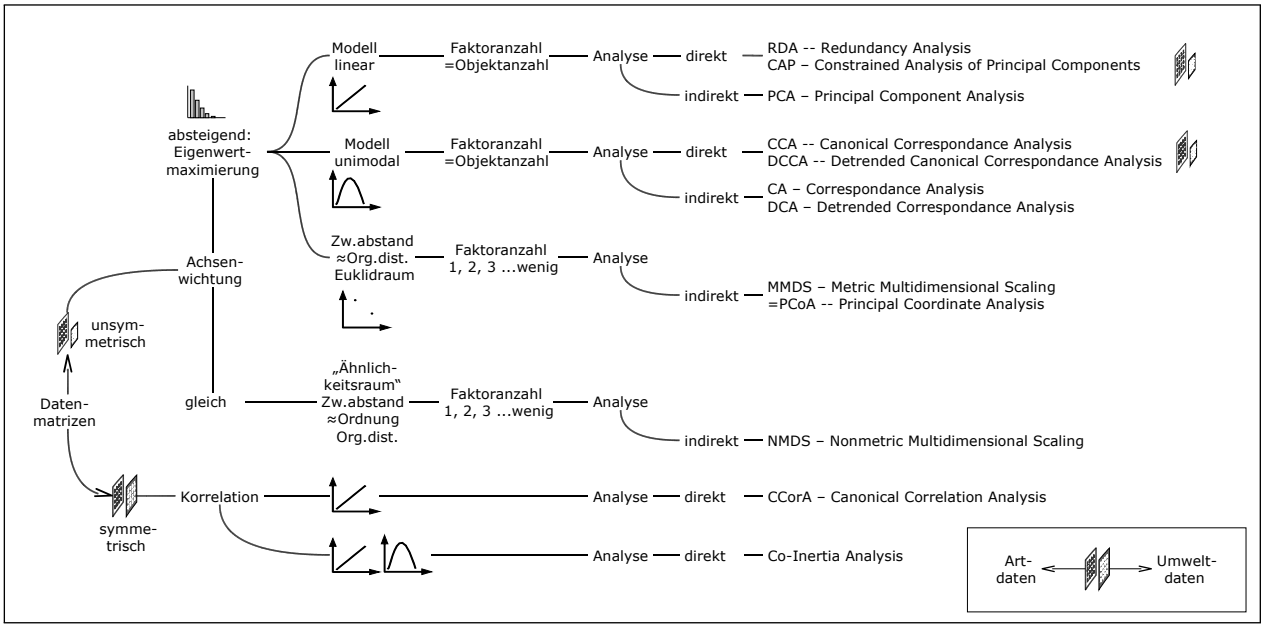


Abbildung 8: Übersicht über Ordinationstechniken und ihre Verwendung/Merkmale. Indirekte Analysen errechnen *hypothetische* Faktoren, die in den Daten liegen. Direkte Analysen beziehen z.B. Umweltdaten direkt in die Berechnung mit ein (daher direkt). Ist die Faktoranzahl gleich der Objektanzahl, dann werden die Faktoren als voneinander unabhängig betrachtet. Ist die Objektanzahl kleiner, dann geht das Modell davon aus, daß die Faktoren sich überlappen.

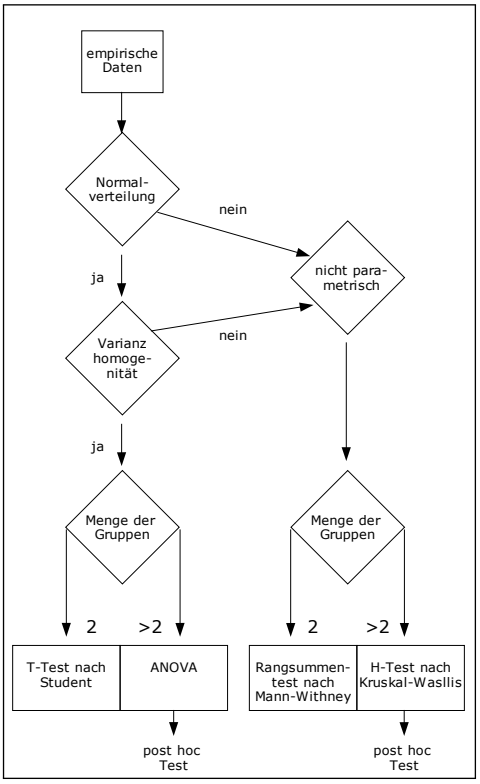
PCA Mit Hilfe der Faktorenanalyse oder PCA (principal component analysis) können große Variablensätze zu wenigen Variablengruppen geordnet werden. Die Faktorenanalyse ist also ein datenreduzierendes Verfahren. Der Zusammenhang zwischen Variablen soll dabei überschaubarer und interpretierbar gemacht werden. Wie gut eine einzelne Variable dann zu einer Variablengruppe paßt, dafür ist die korrelative Beziehung unter den Variablen verantwortlich. Einen Faktor bei der PCA kann man sich hierbei - ähnlich einer Regressionsgerade - als „besten Repräsentanten“ einer Variablengruppe vorstellen. Das Ergebnis der Faktorenanalyse sind wenige voneinander *unabhängige*, *hypothetische* Faktoren – man geht also davon aus, daß die Faktoren untereinander NICHT korrelieren. Wie die Ursprungsvariablen zu der errechneten Dimension beitragen, wird dabei aus den Faktorladungen deutlich: Eine Ladung von 1 bedeutet, die Variable ist mit dem Faktor identisch, eine Ladung von 0 bedeutet, die Variable ist von dem Faktor vollkommen unabhängig.
Interpretation der Variablen: wo die Vektoren hinzeigen nehmen die Werte linear zu, Vektoren rechtwinklig zueinander sind nicht korreliert, welche die in eine Richtung zeigen: hochkorreliert
pCCA Die pCCA ist ein unimodale, direkte Ordinationstechnik, bei der gezielt Variablen herausgerechnet werden, daher auch partiell. Das Ergebnis ist nicht dasselbe, wenn man gleich ohne die Variable(n) rechnet. Siehe auch CCA
PCoA Ausgangspunkt einer Hauptkoordinatenanalyse (engl.: principal coordinate analysis oder auch Metrisch Multidimensionale Skalierung) ist eine Distanzmatrix. Diese kann eine transformierte oder nicht-transformierte Assoziations-Matrix sein (manchmal auch direkt aus einer Datenmatrix berechnet). Die PCoA hilft uns die Struktur der Distanzmatrix näher zu beleuchten. Dabei wird die entstehende Grafik in einem Euklidischen Raum abgebildet (wie ein Kartesisches Koordinatensystem) wobei die relativen Distanzen der Arten untereinander so gut wie möglich beibehalten werden.
Im Gegensatz zur PCA werden weniger Dimensionen als Objekte durch eine ähnliche Prozedur wie bei der PCA ausgerechnet (die berechneten Faktoren überlappen sich also, d.h. sie werden nicht als unabhängig voneinander angesehen und vermischen sich quasi). So wie bei der PCA geben die Eigenwerte Information darüber, wieviel Varianz ein Faktor (=Dimension) die Daten erklärt. Im Gegensatz zur PCA können bei der Hauptkoordinatenanalyse auch Daten mit unterschiedlichem Skalenniveau verrechnet werden. s.a. NMDS (Legendre und Legendre 1998, http://myweb.dal.ca/~hwhitehe/BIOL4062/summ_11.htm)
Funktionen in R: `cmdscale(...)` im Paket `stats` (für die Güte der Anpassung wird ein Eigenwert-basiertes

„goodness of fit“ Maß `GOF` ausgegeben.); `dudi.pco(...)` im Paket `ade4`

Permutationstests Randomisationsmethoden sind Verfahren zum Testen von Hypothesen. Sie sind in der Literatur auch unter dem Namen Permutations-Tests zu finden. Bei konventionellen Tests wird ein Wert einer Teststatistik berechnet und mit einer statistischen Verteilung verglichen. Im Randomisations-Test wird eine statistische Referenzverteilung zufällig aus den Daten gezogen. Randomisations-Tests sind verteilungsfreie Verfahren, die Daten müssen jedoch unabhängig sein. Es wird immer auf die Nullhypothese H_0 getestet.

Poissonverteilung Die Poissonverteilung beschreibt die Verteilung die entsteht, wenn das Ereignis, welches eintritt, die Werte 0, 1,..., n annimmt (d.h. viele aber abzählbar und nicht ins Unendliche gehend). Als Beispiel hierzu wird oft der radioaktive Zerfall erwähnt: Aus einer sehr großen Anzahl von Atomen zerfällt in einer Zeiteinheit nur ein sehr kleiner Anteil der Atome. Dieser Zerfall ist rein zufällig und unabhängig von den schon zerfallenen Atomen. Dies ist eine wesentliche Voraussetzung für die Poisson - Verteilung. Da die Poissonverteilung nur diskrete Werte annehmen kann heißt sie auch *diskret*. Das Gegenteil wäre stetig, d.h. es können ∞ Werte angenommen werden. Siehe auch Verteilungen

post-hoc Tests auch a posteriori Tests – Es gibt verschiedene a posteriori Tests, die man ausführt, wenn sich Unterschiede in den Daten durch das Experiment nach der Versuchsplanung ergeben haben. Sie unterscheiden sich in den jeweils zugrundegelegten Gesichtspunkten für die Kompensation der Alpha-Fehler Kumulierung und damit im Ausmaß der Konservativität der Entscheidungen über die Ablehnung der Nullhypothese H_0. Konservativ ist eine Entscheidung über H_0 dann, wenn große Effekte signifikant werden, radikalere (= weniger konservative) Tests weisen bereits kleinere Effekte als signifikant aus. Nach Ihrer Konservativität geordnet gibt es die folgenden a posteriori Test:



Der **Scheffé-Test** ist der konservativste. Mit ihm ist es möglich mehr als 2 Mittelwerte zu vergleichen, indem sogen. lineare Kontraste gebildet werden – Scheffé verwendet die F - Verteilung. *Frage:* Es sollen Mittelwerte $\bar{x_1}, \bar{x_2}, ...\bar{x_k}$ bzw. Summen dieser Mittelwerte auf signifikante Unterschiede geprüft werden. *Voraussetzung:* die Varianzanalyse ergab eine Verwerfung der Nullhypothese H_0, d.h. es gibt Unterschiede in den Daten. Die Vgl. sind ungeplant. Es darf Unbilanziertheit vorliegen, d.h. ungleiche Stichprobenanzahlen. Nullhypothese H_0 es liegen keine linearen Kontraste vor. In R siehe Benutzerfunktion `scheffeCI(...)` auf Seite 91.

Der **Tukey Honest** *Fragestellung:* Es sollen Mittelwerte $\bar{x_1}, \bar{x_2}, ...\bar{x_k}$, bzw. Summen dieser auf signifikante Unterschiede geprüft werden. *Voraussetzung:* die Varianzanalyse ergab eine Verwerfung der Nullhypothese H_0, d.h. es gibt Unterschiede in den Daten. Die Vgl. sind ungeplant. Es liege Balanziertheit vor, d.h. die Anzahl Wiederholungen sei bei allen Faktorstufen gleich. Nullhypothese H_0 es liegen keine linearen Kontraste vor. In R mit `TukeyHSD(...)` Paket `ctest/stats` oder mit Benutzerfunktion `tukeyCI(...)` auf Seite 90.

Der **Newman-Keuls-Test** berücksichtigt nur die Tatsache, daß bei den der Größe nach geordneten Mittelwerten für benachbarte Mittelwerte schon kleinere Differenzen signifikant werden können als bei weiter auseinander liegenden Mittelwerten (=Range- Statistik). Für benachbarte Mittelwerte entspricht er genau dem t - Test zum Vergleich zweier Mittelwerte. *Frage:* Welche der k Stichprobenmittelwerte $\bar{x_1}, \bar{x_2}, ...\bar{x_k}$ unterscheiden sich signifikant? *Voraussetzung:* Die ANOVA ergab eine Verwerfung der Nullhypothese H_0, d.h. es gibt Unterschiede in den Daten. Die Vergleiche sind ungeplant. Es werden jeweils zwei Mittelwerte verglichen. Nullhypothese H_0 zwei verglichene MW sind gleich. Ähnliche Tests in R mit `pairwise.t.test(...)` Paket `ctest/stats` mit `p.adjust`

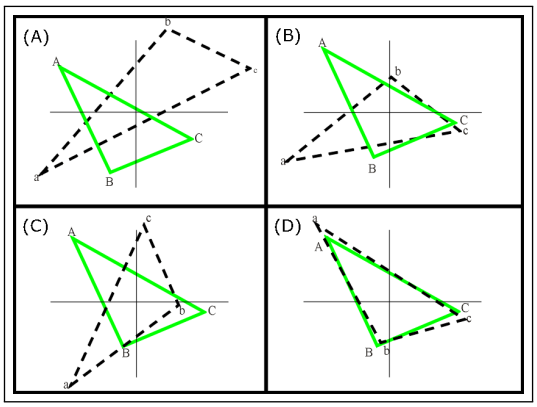
Der noch radikalere **Duncan-Test** entspricht dem Newman-Keuls- Test, legt aber für die weiter auseinander liegenden Mittelwerte engere Prüfverteilungen zu Grunde.

Der **Least Significant Differences Test** (LSD-Test) entspricht dem t - Test zum Vergleich zweier Mittelwerte. Er ist am wenigsten konservativ (=starkste Alpha-Fehler Kummulierung). Ähnliche Tests in R mit `pairwise.t.test(...)` Paket `ctest/stats` mit `p.adjust`.

(Quelle: Schema http://web.zoo.uni-heidelberg.de/Alternativmethoden/Skript%20Statistik.pdf; post hoc Tests http://eeglab.uni-trier.de/docs/vorlesung_va/10.pdf)

Potenztransformation s. Symmetrisierung

Prokrustes-Test In der griechischen Mythologie hat der Gastwirt Prokrustes Gästen, die zu klein für seine Betten waren, die Beine lang gezogen und zu großen Gästen wurden die Beine abgeschnitten, so daß sie in die Betten passten. Ähnlich verfährt auch der Prokrustes-Test. Er vergleicht keine Distanz- oder Ähnlichkeitsmatrizen, sondern zwei rechteckige Rohmatrizen. Haben die beiden Matrizen ungleich viele Variablen, wird die kleine Matrize durch Erweiterung mit Null-Werten auf die Größe der Größeren gebracht. Durch Rotation werden die zu vergleichenden Matrizen so gefitted, daß die Summe der quadratischen Abweichungen zwischen korrespondierenden Punkten der Matrizen minimiert wird. Es handelt sich also streng genommen um eine Ordinationstechnik. Diese Methode wird meist genutzt, um Ordinationsergebnisse zu vergleichen. (aus Dormann und Kühn 2004)

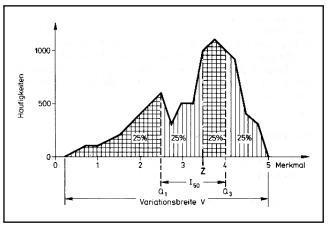


Was passiert beim Prokrustes Test? Ein Beispiel des Prokrustes Tests zeigt Abbildung (A) mit zwei einfachen Anordnungen. Die zusammengehörigen landmarks sind in GROß- und kleinbuchstaben gekennzeichnet. Das Ziel ist es die quadrierten Abweichungen (=Fehler und mit m^2 bezeichnet) zwischen den Landmarks zu minimieren durch Erweiterung, Drehung und Überführung ähnlich der anderen Anordnung, die sozusagen das Anpassungsziel darstellt. Abb. (B) zeigt die Anordnug nach Transformation, d.h. die Daten wurden centriert. Nach Rotation (Abb. C) werden die Daten durch Erweitern (scaling) angepaßt so daß m^2 minimal ist (Abb. D). Die Abweichungen zwischen den landmarks werden als Residialvektoren bezeichnet. Ein kleiner Residuenvektor zeigt große Übereinstimmung zwischen zusammenhängenden landmarks. Der Term m^2 basiert dabei auf der Summe der Abweichungsquadrate (Gower 1971b, http://www.zoo.utoronto.ca/jackson/pro1.html).

Q

Q - Modus Eine ökologische Datenmatrix kann aus zwei Haupt-Analysemodi betrachtet werden. Einmal aus Sicht der Deskriptoren (z.B.: Umweltvariablen Q - Modus) und zum anderen aus Sicht der Objekte (R - Modus). Das entscheidende hierbei ist, daß hier unterschiedliche Maße verwendet werden. Um Abhängigkeiten zwischen Deskriptoren zu beschreiben kommen Koeffizienten, wie der Pearsonsche Korrelationskoeffizient r vor, daher heißt dieser Modus R - Modus. Im Gegensatz dazu steht der Q - Modus, der die Abhängigkeiten der Objekte betrachtet. (Legendre und Legendre 1998)

Q - Q Plot Der Q - Q Plot oder Quantil - Quantil Plot (s.Quartil) bietet die grafische Möglichkeit, zu beurteilen, ob zwei Populationen aus einer gemeinsamen Verteilung stammen oder nicht. Dabei werden die Quantile beider Populationen oder einer Verteilungsfunktion gegen die einer Datenverteilung aufgetragen. Eine 45-Grad Linie wird i.d.R. auch mit eingezeichnet. Auf ihr liegen die Punkte nur dann, wenn die zwei Populationen aus der selben Verteilung stammen. Je größer die Abweichung, desto unterschiedlicher sind die Populationen/Werte hinsichtlich ihrer Verteilung. Die Vorteile des Q - Q Plots: die Probenanzahl muß nicht gleich sein; mehrere Aspekte sind auf den ersten Blick ersichtlich: Symmetrie, Ausreißer und Verschiebungen (z.B. Skalierung, Mittelwert).



Quartil Quartile (Q_1,...Q_4) teilen ein der Größe nach geordnetes Datenbündel in vier Teile. Das 25 %-Quartil (= 1. Quartil) gibt denjenigen Wert an, der das untere Viertel der Datenwerte von den oberen drei Vierteln trennt, usw. Das 50%-Quartil (2. Quartil) ist der **Median**. Der Abstand zwischen dem 25%-Quartil und dem 75%-Quartil (3. Quartil) wird als **Interquartilsabstand** bezeichnet. Es handelt sich bei Quartilen (mit Ausnahme des Medians) um Streuungsmaße. Die **Variationsbreite** umfaßt den Bereich vom kleinsten Meßwert bis zum größten.

R

R^2 auch R - squared, s. Bestimmtheitsmaß

R^2 adjusted s. Bestimmtheitsmaß

Rangkorrelationskoeffizienten Die Ermittlung von Korrelationen auf der Basis von Rängen bietet sich an, wenn die Deskriptoren nicht normal verteilt sind.

- Beim Rangkorrelationskoeffizienten nach Spearman r bekommen die Objekte einen Rang für ihre Deskriptoren zugeordnet. Der höchste Rang bekommt das Objekt mit dem höchsten Wert für y_1 bzw. y_2 . Die Ränge für y_1 und y_2 werden dann korreliert. Dazu kann der Pearson'sche Korrelationskoeffizient benutzt werden. Objekte können sich auch Ränge teilen. Dabei wird dann der durchschnittliche Rang gebildet. Der Signifikanztest für Spearman's r ist identisch mit Pearson's r.

- Beim Rangkoeffizienten nach Kendall τ werden die Ränge für y_1 und y_2 nach y_1 in aufsteigender Reihenfolge sortiert. Für y_2 werden Zahlen vergeben. Sind die Ränge zweier Objekte aufsteigend, so bekommen sie $+1$, sind sie absteigend, bekommen sie ein -1 zugeordnet. Die Summe der vergebenen Zahlen wird dann benutzt, um τ zu berechnen. Eine perfekte Korrelation ist dann vorhanden, wenn auch die Ränge von in aufsteigender Reihenfolge vorliegen. Kendall's τ kann nicht für geteilte Ränge berechnet werden.

RDA Die Redundanz - Analyse[56] hat das Ziel, die Varianz von Y $(n \times p)$ durch eine zweite Datenmatrix X $(n \times m)$ zu erklären. Die RDA setzt voraus, daß die Beziehungen innerhalb von Y linear sind. Prinzipiell ist die RDA eine Erweiterung der PCA, wobei die Hauptachsen eine Linearkombination der Regressoren in X sind.

reciprocal averaging Reciprocal averaging ist ein weit verbreiteter Algorhytmus für die Korrespondenzanalyse (CA). Die Korrespondenzanalyse selbst wird auch mit dem Begriff „reciprocal averaging" bezeichnet.

Redundanz ist das was überflüssig ist. In der Nachrichtentechnik z.B. ist das derjenige Teil der Mitteilung, der keinen Informationsgehalt hat.

Regressionsanalyse Die Regressionsanalyse ist eine Technik zur Modellierung einer Beziehung zwischen mindestens zwei Variablen. Man unterscheidet je nach Zusammenhang zwischen linearer und nichtlinearer Regression. Werden mehr als zwei Variablen zur Regressionsanalyse benutzt, so spricht man von multipler Regression. Das Ziel der Regressionsanalyse ist es, ein Modell zwischen einer oder mehreren unabhängigen und einer abhängigen Variablen zu finden, um das Verhalten der abhängigen Variablen zu prognostizieren. Diagnostische Plots in R :

- der Plot **Residuen vs. Fitted Values** bringt oft eine verbleibende unerklärte Struktur der Residuen zu Tage, wohingegen in einem guten Modell die Residuen zufällig um Null streuen sollten.

- der Plot der normalen Quantile der Residuen (**Normal Q-Q Plot**), erlaubt einen optischen Test der Normalverteilungsannahme der Fehler. Wenn die geordneten Residuen annähernd auf der Winkelhalbierenden liegen, hat man guten Grund zu der Annahme, daß die Fehler tatsächlich normalverteilt sind. Wie auch sortierte Zufallszahlen verdeutlichen können: `plot(sort(runif(100)))`

- der **Scale-Location Plot** zeigt die Wurzel der standardisierten Residuen gegen die Fitted Values; Punkte weit oben oder unten habe große Residuen

- der **Cook's Distance Plot**: Die Cook's Distance mißt den Einfluß einer einzelnen Beobachtung auf die Regressionskoeffizienten, d.h. eine Beobachtung mit einem großen Einfluß verändert die Regressionsebene stark, wenn die Beobachtung weggelassen wird. Eine Große Distanz steht für einen großen Einfluß auf den Regressionskoeffizienten.

Residue Als Residuen bezeichnet man die Differenzen zwischen den beobachteten Meßwerten y_i und den berechneten geschätzten Werten $\hat{y}_i$ eines Modells (Köhler et. al 1996). Die Residuen entsprechen dann dem Abstand der Punkte von der Modellgleichung: beobachtet $-$ geschätzt.

Riemann Distanz wird verwendet, wenn 2 Projektionen hinsichtlich Größe UND Form verglichen werden. So z.B. in der Biometrie bei der Prokrust Analyse (Prokrustes-Test). Der Wertebereich dieser Distanz ist 0 bis $\frac{\pi}{2}$ (1,5708).

R - Modus s. Q - Modus

robust Ein Verfahren der analytischen Statistik heißt robust, wenn es näherungsweise auch bei bestimmten Abweichungen (z.B. Ausreißern) von den Voraussetzungen, unter denen es abgeleitet wurde, gültig ist.

S

[56] lat. redundantia = überfluß

Shapiro-Wilk Test Dieser Test, testet auf Normalverteilung (= H_0). `shapiro.test(x)` Paket `ctest/stats` für $n = 3...5000$, H_0 : die Streuung von x gleicht der, der Normalverteilung – Bsp.: $P = 0.004$, dann ist x NICHT normalverteilt.
`shapiro.test(rnorm(100, mean = 5, sd = 3)) shapiro.test(runif(100, min = 2, max = 4))`
Shepard Diagramm Ein Shepard-Diagramm wird oft dazu benutzt, um zu sehen wie repräsentativ die errechnete (weniger dimensionierte) Ordination ist. Es werden dabei auf der x-Achse die Distanzen im multidimensionalen Raum (Datenraum) und auf der y-Achse die Distanzen aus der Berechnung (reduzierter Datenraum) aufgetragen. (Legendre und Legendre 1998)

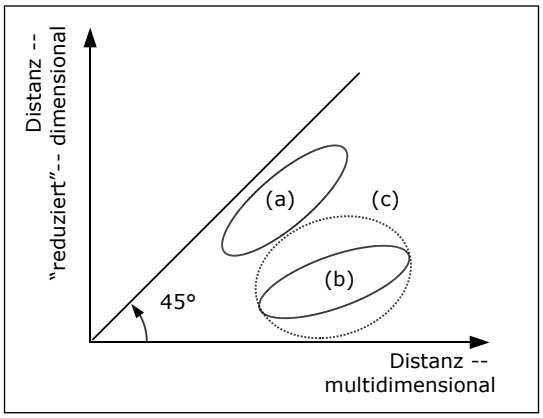


Abbildung 9: (a) der reduzierte Dimensionsraum repräsentiert einen großen Varianzanteil. (b) der Varianzanteil, den die reduzierte Dimension repräsentiert, ist kleiner. (c) dasselbe wie b, nur manche Distanzen werden gut und manche schlecht repräsentiert. Das Optimum wäre, wenn alle Punkte nahe 45 liegen.

Silhouette Plots sind eine graphische Darstellung des Ergebnisses einer hierarchischen Clusteranalyse und werden z.B. beim Fuzzy Clustering (s.Cluster Analyse Verfahren) mit der Funktion `fanny(...)` im Paket `cluster` ausgegeben. Um eine Silhouette zu erstellen benötigt man eine Distanzmatrix D und die Information, zu welcher Klasse das i-te Objekt gehört. Beim Berechnen der Distanzmatrix wird jedem „Distanz"-Objekt eine Zahl $s(i)$ zugeordnet, die angibt, wie gut das Objekt klassifiziert wurde. Dabei werden zwei Aspekte betrachtet. Einerseits wird durch eine Maßzahl beschrieben, wie nah ein Objekt an allen anderen Objekten seiner Klasse liegt, andererseits wird eine Maßzahl bestimmt, die die Nähe eines Objekts zu seiner nächsten Klasse beschreibt. Beide Maßzahlen werden zu einer Maßzahl zusammengefaßt. Die Werte von $s(i)$ liegen zwischen -1 und 1. Je höher der Wert von $s(i)$, desto mehr liegt Objekt i in seinem Cluster.

Beurteilung des Silhouettenkoefizienten SC	
0.71 bis 1.00	starke Struktur
0.51 bis 0.70	vernünftige Struktur
0.26 bis 0.50	schwache Struktur
0.00 bis 0.25	keine substantielle Struktur

skaleninvariant Skaleninvarianz ist ein Begriff aus der Mathematik. Gegeben ist eine Variable x mit einer Funktion $f(x)$. x wird mit einer Konstanten a multipliziert, also skaliert. Wenn dann $f(x) = f(ax)$, ist, nennt man die Funktion f skaleninvariant. Das bedeutet beispielsweise, daß die Funktion unverändert bleibt, wenn wenn man als Einheit Gramm statt Kilogramm verwendet. Skaleninvariant sind etwa: Korrelationsmatrix und Mahalanobisdistanz

Skalenniveau Das Skalenniveau wird bestimmt durch die Möglichkeit, vorhandene Objekte verschiedenen Kategorien oder Objektklassen zuzuordnen. Die Hauptfrage heißt hier also: wie kann ich das Beobachtete quantifizieren, also in Zahlen ausdrücken - und welche Beziehungen müssen die Zahlen untereinander haben, um die Wirklichkeit präzise widerzuspiegeln? Die unterschiedlichen Skalenniveaus müssen steigende Voraussetzungen erfüllen und haben so unterschiedliche Prüfverfahren und charakteristische Verteilungsmaße, und sie haben einen unterschiedlich großen Aussagewert. Siehe auch Intervallskala, Nominalskala, Ordinalskala, Verhältnisskala.

	Berechnungsmöglichkeiten							
	=	≠	<	>	+	−	÷	×
nominal	✓	✓						
ordinal	✓	✓	✓	✓				
Intervall	✓	✓	✓	✓	✓	✓		
Verhältnis	✓	✓	✓	✓	✓	✓	✓	✓

nichtmetrische Daten			**metrische** Daten	
nominal („namentlich“)	**binär**	**ordinal** („ordinale: eine Ordnung anzeigen“)	**intervallskaliert**	**verhälnisskaliert**
Geschlecht von Arten, Farben: dkgrün, grün, blau	vorhanden - nicht vorhanden	Schulnoten, wenig - mehr -viel, (!!keine Mittelwertbildung!!)	Differenzen: Temperaturskala [C]...	Gewicht, Körperlänge (32 cm ist 2× so lang, wie 16 cm ABER 32C ist ~~2×~~ 16C), K, Reaktionszeit, Meßwerte: O_2 [mg/l^{-1}]

Skewness Die Skewness (Schiefe) einer Verteilung gibt an, ob sich die Werte normal verteilen oder in eine Richtung der Skala tendieren. $\left(Skewness = \frac{\bar{x} - Modalwert}{StdAbw.}\right)$ (s.a. Kurtosis)
liegt vor wenn $Skewness < 0$
liegt vor, wenn $Skewness > 0$
liegt vor, wenn $Skewness = 0$
Eine Normalverteilung hat eine Skewness von **0**.
In R mit dem Paket `fBasics` ab v1.9: `skewness(rnorm(100)); skewness(log(rnorm(100)))`
Sørensen Dieses Distanzmaß wurde ursprünglich für presence - absence Daten entwickelt, aber es kann ebenso mit quantitativen Daten verwendet werden. Es ist meist verwendbar bei ökologischen Daten. Der Gebrauch dieser Distanz bei Intervalldaten ist eher empirisch zu rechtfertigen, als theoretisch ableitbar. Verglichen zur Euklid - Distanz reagiert es weniger sensitiv Ausreißern und heterogenen Daten gegenüber. – für Binärdaten, andere Namen: „Bray - Curtis“, „Lance und Williams (SPSS)“, s.a Distanzmaße
species score Eine Koordinate entlang einer Ordinationsachse, die die Position einer Art beschreibt, bezeichnet man als species score. Bei wichtenden Ordinationsmethoden wie CA, CCA und DCA repräsentieren die species scores das Zentroid der Art oder den Modalwert („höchste Abundanz“) der unimodal response - Kurve. Species Scores sind hilfreich bei der Interpretation der Ordinantionsachsen bei indirekten Ordinationsverfahren.
Standardabweichung Die Standardabweichung σ wird berechnet als die Quadratwurzel aus der Varianz – sie ist skaleninvariant. Berechnung mit `sd(...)`
Standardfehler Der Standardfehler (engl.: standard error) ist ein Maß für die Streuung einer Stichprobenstatistik über alle möglichen Zufallsstichproben vom Umfang n aus der Grundgesamtheit. Vereinfachend gesagt: Er ist ein Maß für die „durchschnittliche“ Größe des Stichprobenfehlers der Stichprobenstatistik (z.B. des arithmetischen Mittels oder des Anteilswertes). Der Standardfehler einer Stichprobenstatistik hängt von verschiedenen Faktoren ab, je nachdem, um welche Statistik es sich handelt. Ganz allgemein kann man jedoch sagen, daß ein Standardfehler um so kleiner wird, je größer der Stichprobenumfang ist. Größere Zufallsstichproben erlauben präzisere Schätzungen, weil der Stichprobenfehler kleiner wird.
Standardisierung Die Standardisierung dient dem Vergleich verschieden dimensionierter Variablen. Dabei wird von jedem Wert y_i einer Datenreihe deren Mittelwert abgezogen (Zentrieren) und dieser Wert durch die Standardabweichung dividiert:

$$y_{stan} = \frac{y_i - \overline{y}}{\sqrt{s_y^2}} \quad mit \quad s_y^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \overline{y})^2$$

Durch diese Operation verlieren Variablen ihre Dimensionen. Der Mittelwert der standardisierten Variable ist Null, die Varianz ist Eins. s.auch Datentransformation
statistische Power ist die Wahrscheinlichkeit, daß der Test eine falsche Nullhypothese H_0 verwirft. http://en.wikipedia.org/
sum of squares Mit „sum of squares“ sind die Quadratsummen gemeint: mittlere Abweichungsquadrate vom Mittelwert. ähnlich, aber nicht zu verwechseln mit Varianz.
Symmetrisierung Die Symmetrisierung einer Datenreihe dient der Annäherung an die Normalverteilung. Die Normalverteilung ist eine wichtige Voraussetzung für die Durchführung vieler statistischer Verfahren. Eine symmetrische Verteilung von Daten kann durch verschiedene Transformation der Daten erreicht werden, z.B. Logarithmus-Transformation,Wurzel-Transformation, Potenztransformation, usw. Die Art der Transformation richtet sich nach der Nicht-Symmetrie der Daten und nach ihren Eigenschaften (d.h. quantitative, kategorielle oder proportionale Daten). Die Transfor-

n	transformierte Werte	Bemerkungen
...	...	↑
3	y^3	linksschief
...	...	↑
2	y^2	↑
...	...	↑
1	y^1	ohne Effekt
...	...	↓
0.5	$\sqrt{y}$	↓
...	...	↓
log	log y	↓
...	...	↓
-0.5	$1/\sqrt{y}$	↓
...	...	↓
-1	$1/y$	rechtsschief
...	...	↓
-2	$1/y^2$	↓
...	...	↓

mation von Proportionen erfolgt am effektivsten durch die Winkeltransformation:

$$y_i' = arcsin\sqrt{y_i}$$

Durch die Transformation werden Daten an den Rändern dichter ins Zentrum plaziert, damit haben Extremwerte weniger Einfluß.
Bei Transformationen für quantitative, schiefe Daten (s.Skewness) unterscheidet man in linksschief und rechtsschief verteilte Daten. Eine geeignete Transformation rechtsschiefer Daten muß die Abstände zwischen größeren Werten stärker reduzieren als zwischen kleineren Werten. Alle Potenztransformationen mit $n < 1$ leisten das (s. Tabelle)
Einen Sonderfall stellt die Logarithmustransformation dar. Sie bewirkt zusätzlich, daß Werte nahe Null entzerrt werden.
Potenztransformationen bewirken, daß bei $n < 1$ große Werte stärker zusammenrücken. Potenzfunktionen mit negativem Exponenten haben dieselbe Wirkung wie die log-Transformation. Bei linksschief verteilten Daten muß die Transformation bewirken, dass höhere Werte stärker entzerrt werden. Dazu sind alle Potenztransformationen mit $n > 1$ geeignet. Eine Entscheidungshilfe zur Auswahl der geeigneten Transformation bietet die Transformationsleiter (s. Tabelle).
s.auch Datentransformation (Quelle http://www.bio.uni-potsdam.de/oeksys/vstatoek.pdf)

T

Testentscheidung Dieses Schema soll helfen sich in dem Wirrwarr der verschiedenen Tests zurechtzufinden.

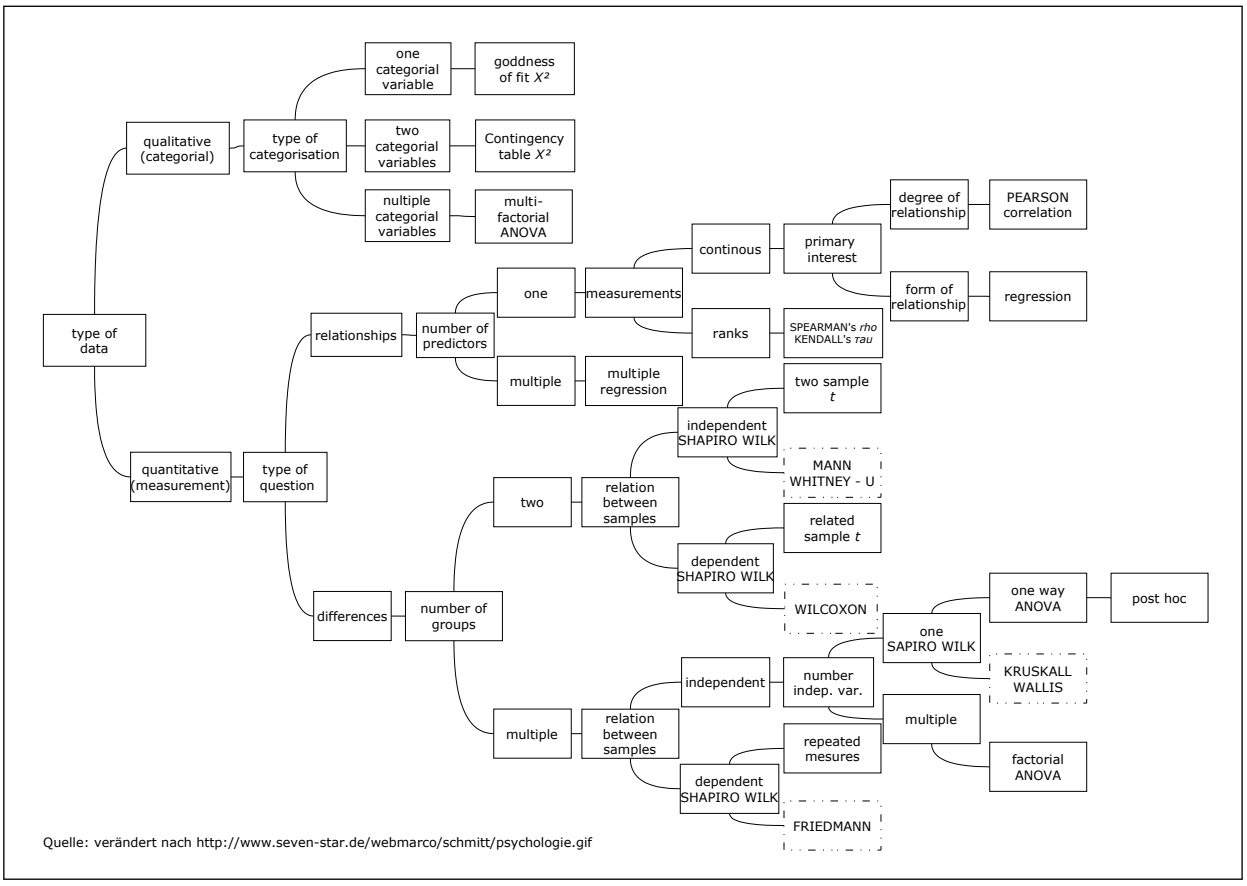


Abbildung 10: Testentscheidung – Testschema: – · · – nichtnormalverteilte Daten

TPS thin plate spline[57]: ist ein Interpolationsalgorithmus der aus der Physik entlehnt ist und in die Morphometrie durch Fred Bookstein eingebracht wurde. Der Algorithmus berechnet ein Deformationsgitter aus

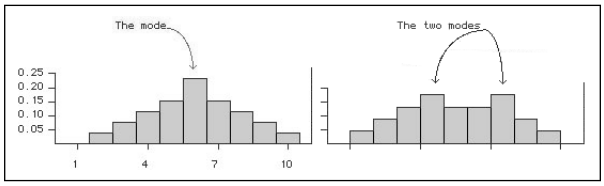
[57] spline ist eine Art Glättungskurve

dem Vergleich zweier Punkt-Konfigurationen (auch Landmarks), so daß die „Biegungsenergie“ der Deformation so gering als möglich ist. Bei der Spline-Berechnung wird dabei die quadrierte 2.Ableitung benutzt. Quelle: http://www.virtual-anthropology.com/virtual-anthropology/geometric-morphometrics

t - Test Ist ein besonders gebräuchlicher Signifikanztest für den Vergleich von zwei Mittelwerten. Der t-Test besitzt jedoch einige Voraussetzungen, die erfüllt sein müssen, damit er berechnet werden kann: die Mittelwerte müssen intervallskaliert sein (s.Skalenniveau) und normalverteilt sein. Die Nullhypothese H_0 ist: Mittelwert 1 und Mittelwert 2 sind gleich (kurz: $\mu_1 = \mu_2$). Die die Alternativhypothese H_A ist: $\mu_1 \neq \mu_2$. Der t-Test berechnet einen t-Wert $t_{Versuch}$. Dieser wird dann mit einem theoretischen t-Wert $t_Tabelle$ verglichen. Dieser Wert $t_Tabelle$ errechnet sich aus den Kenngrößen Anzahl derFreiheitsgrade (FG) und Alpha-Fehler (genau: Vgl. von 2 Mittelwerten $\Rightarrow FG-2$, Vgl. von einem Mittelwert mit einem theoretischen $\Rightarrow FG-1$). Man schreibt das $F(FG;\alpha)$. Ist $t_{Vers} \leqq t_{Tab}$ dann gilt H_0, ist $t_{Vers} > t_{Tab}$, dann gilt H_A. Test auf Normalverteilung: Shapiro-Wilk Test (`shapiro.test(x)` Paket `ctest/stats` für $n = 3...5000$, H_0 : die Streuung von x gleicht der, der Normalverteilung – Bsp.: $P = 0.004$, dann ist x NICHT normalverteilt) oder Kolmogorov-Smirnov-Test (testen, wenn 2 Variablen unabhängig sind, `ks.test(x, y)` Paket `ctest/stats`; $H_0 : x$ und y sind aus der selben Verteilung). Anm.: Die t-Test Statistik wird auch benutzt, um Regressionskoeffizienten zu testen. Deshalb taucht diese Statistik auch beim Output der Regressionsmodelle auf, wo die entsprechenden Koeffizientenwerte a b_1, b_2 der Geradengleichung $y = a + b_1x_1 + b_2x_2 + ...$ gegen 0 (=Nullhypothese H_0) getestet werden. Die F-Teststatistik hingegen, vergleicht die Streuungen (Varianzen) um die Mittelwerte.
(Quelle: http://www.wu-wien.ac.at/inst/ivm/strunk/pdf/StatistikGlossar.pdf)

U

unimodal Eine Häufigkeitsverteilung mit nur einem Gipfel und damit mit nur einem Modalwert wird als unimodal bezeichnet. Eine mit zwei häufigen Werten als bimodal. (s.a. Modalwert)



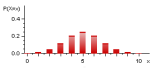
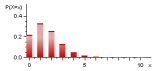
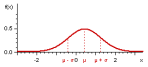
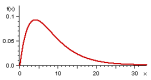
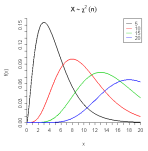
V

Varianz Die Varianz σ^2 (oder auch: Streuung; Dispersion) beschreibt die Verteilung der Merkmalsausprägung einer Variablen um den Mittelwert μ. Sie wird berechnet, indem die Summe der quadrierten Abweichungen aller Meßwerte vom arithmetisches Mittel geteilt wird durch die um 1 verminderte Anzahl der Messungen – die Varianz ist im Gegensatz zur Standardabweichung skalenabhängig und ist das Quadrat der Standardabweichung. Man muß sich dabei natürlich immer vor Augen halten, daß in der Varianz zwei Dinge „versteckt“ sind: zum einen die natürlich gegebene Streuung und zum anderen sind es die Meßfehler, die auch mit enthalten sind.

Varianzanalyse s.ANOVA

Verhältnisskala Bei der Verhältnis- oder Absolutskala muß es neben der definierten Maßeinheit auch einen absoluten Nullpunkt geben. Beispiel: die Celsius-Temperatur-Skala ist lediglich eine Intervallskala, da sie einen willkürlich festgelegten Nullpunkt (die Temperatur bei der Wasser gefriert) besitzt; die Kelvin-Skala hingegen ist eine Verhältnisskala, da ihr Nullpunkt (-273C, die Temperatur unter der keine Teilchenbewegung mehr erfolgen kann) von Natur aus vorgegeben ist und nicht unterschritten werden kann. Neben dem Addieren und Subtrahieren sind auch die Multiplikation und Division erlaubt.

Verteilungen

	Wertebereich	Eigenschaft		Beispiele
Binomialverteilung, $\frac{B(m,\pi)}{m}$	$0, \frac{1}{m}, ..., 1$	diskret		ja/nein Erfolge/Versuche
Poissonverteilung, $\frac{Po(\mu)}{\nu}$	$0, 1, ..., \infty$ (abzählbar)	diskret		Zähldaten; Varianz ist gleich dem MW
Normalverteilung, $N(\mu, \sigma^2)$	$-\infty, \infty$ (nicht zählbar)	stetig		
Gammaverteilung, $G(\mu, \nu)$	0,..., ∞ (nicht zählbar)	stetig		Blattfraß phytophager Insekten: viele Blätter nicht angefressen, andere nahezu vollständig; Varianz nimmt mit MW zu
Chi ^{2} - Verteilung Gamma Spezialfall	0, 1,..., ∞ (nicht zählbar)	stetig		
quasi Verteilung	0, 1,..., ∞ (nicht zählbar)	stetig	Wellenfunktion	

Verteilungsanpassungstest Eine Hilfe für Verteilungsanpassungstest mag die Tabelle geben. Anmerkung: alle Pakete, die hier angegeben sind gelten nur für ältere Versionen. Ab Version 2.0. Siehe auch Testentscheidung und post-hoc Tests befinden sie sich im Paket `stats`

Test	**Erläuterung**	**Funktion in R**	**Datenformat/ Annahmen**
	Verteilungsanpassungstest		
Kolmogorov-Smirnov-Test	Nullhypothese H_0: ob zwei numerische Datenvektoren X und Y von derselben Verteilung stammen. `library(ctest) ; x <- rnorm(1000); y <- rgamma(1000, 3,2); ks.test(x,y)`	`ks.test(x, y)`, package `ctest`	2 Spalten von gemessenen Daten/-
Shapiro-Wilk Test	Test einer Stichprobe auf Zugehörigkeit zur Normalverteilung	`shapiro.test(x)`, package `ctest`	Spalte mit gemessenen Daten /3 < n < 5000, H_0 : Streuung von x gleicht Normalverteilung
	Parametrische Tests		

...Fortsetzung umseitig

Test	Erläuterung	Funktion in R	Datenformat/ Annahmen
	...Fortsetzung Verteilungsanpassungstest		
t - Test	Der t - Test gibt eine Angabe über die Konsistenz zweier Mittelwerte. Er erlaubt eine Aussage, ob eine eigene Messung mit der eines ändern (aber auch eine frühere eigene Messung) konsistent ist. Damit kann getestet werden, ob eine Apparatur sich mit der Zeit verändert.	`t.test(x)`, `pairwise.t.test(x)`, package `ctest`	eine oder zwei Spalten mit gemessenen Daten/ Normalverteilung
F - Test	Der F - Test ist ein zum t - Test analoger Test, der die Konsistenz von Varianzen prüft.	`anova.lm(x,...,test="F")`, `anova.glm(x,...,test="F")`, package `base`	zwei oder mehr Spalten mit gemessenen Daten/ Normalverteilung
Chi^2 (χ^2) von Barlett	Test auf Homogenität von mehr als zwei Varianzen	`bartlett.test(x)`, package `ctest`	zwei oder mehr Spalten mit gezählten Daten/?
	verteilungsfreie, parameterfreie Verfahren		
Chi ^{2} - Test	Prüfung der Unabhängigkeit zweier qualitativer Merkmale. Er liefert ein allgemeines Kriterium für die Übereinstimmung der Grundgesamtheit mit der Stichprobe. Der χ^2 - Test taugt für jede Verteilungsfunktion, ist also modellfrei.	`chisq.test(x, y)`, package `ctest`	2 Spalten mit Zähldaten oder 1 getestet gegen eine Verteilungsfunktion/ n $>$ 30 große Datenmengen
Fischers exakter Test	Vergleich zweier Prozentzahlen/Häufigkeiten auf Gleichheit	`fisher.test(x)`, package `ctest/stats`	Prozentzahlen, Häufigkeite/n>3
Test von Cochran	Test auf Gleichverteilung mehrerer verbundener dichotomer Variablen	`mantelhaen.test(x)`, package `ctest/stats`	
	Test auf Korrelation von Rangreihen	`cor.test(x, y)`, package `ctest`	paarweise Zähl- oder Meßdaten/-
Wilcoxon - Test	Rangtest zum Vergleich zweier unabhängiger Verteilungen	`wilcox.exact(x, y)`, package `exactRankTests`	zwei Spalten von Zähl- oder Meßdaten/ n $>$ 7 gemeinsame stetige Verteilung unter der Nullhypothese vorausgesetzt.
Kruskal - Wallis - Test	Vergleich mehrerer Stichproben auf Gleichheit/ Ungleichheit der zugehörigen Verteilungen	`kruskal.test(x)`, package `ctest`	zwei oder mehr Spalten mit gezählten oder gemessenen Daten/-

W

Ward Clusteranalyse Beim Ward - Verfahren werden nicht die Objekte zusammengefaßt, die die geringste Distanz aufweisen, sondern die Objekte, die ein vorgegebenes Heterogenitätsmaß am wenigsten vergrößern. Als Heterogenitätsmaß wird ein Varianzkriterium verwendet: die Fehlerquadratsumme (engl. Sum of Squared).

Liefert im allgemeinen sehr gute Ergebnisse und bildet möglichst homogene Gruppen; ein kleiner Nachteil: ist ein Element zu einem Cluster hinzugefügt kann es später nicht mehr ausgetauscht werden; muß mit Euklid - Distanz berechnet werden.

Wilcoxon - Test *Fragestellung*: sind die Mediane zweier verbundener Stichproben X und Y signifikant verschieden? *Voraussetzungen* die beiden Grundgesamtheiten sollen stetige Verteilungen von gleicher Form haben, die Stichproben seien verbunden und die Daten mindestens intervallskaliert. H_0 : Mediane sind gleich. `wilcox.test(x, ...)` Paket `ctest/stats`

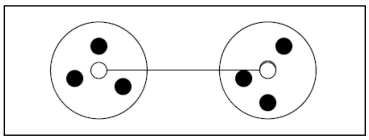
Winkeltransformation s. Symmetrisierung

Z

Zentrieren Beim Zentrieren von Daten werden diese auf ihren Mittelwert bezogen. Die neu berechneten Werte sind die Abweichungen vom Mittelwert. Es gilt:

$$y_{zen} = y_i - \overline{y} \quad mit \quad \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Zentrierte Daten haben einen neuen Mittelwert von Null. Die Abweichungen haben sowohl positive als auch negative Werte. s.auch Datentransformation



Zentroid Clusteranalyse Für jeden Cluster werden die arithmetischen Mittel aus den Werten berechnet, welche die in dem Cluster enthaltenen Objekte (Fälle) in den einzelnen Variablen aufweisen. Für jede Variable, auf die sich die Clusteranalyse stützt, ergibt sich somit (für jeden Cluster) ein Mittelwert. Die Distanz zwischen zwei Clustern wird anschließend in der gleichen Weise berechnet wie die Distanz zwischen zwei einzelnen Objekten, wobei anstatt einzelner Variablenwerte die jeweiligen arithmetischen Mittel zugrunde gelegt werden. (s.a.Cluster Analyse Verfahren)

Zufallsgröße Die Zufallsgröße X[58] ist eine Größe, die bei verschiedenen, unter gleichen Bedingungen durchgeführten Zufallsversuchen verschiedene Werte x_1, x_2,... annehmen kann. Eine *diskrete* Zufallsgröße kann in einem Intervall nur endlich viele Werte annehmen, eine *stetige* Zufallsgröße dagegen beliebig viele.

[58] wird immer GROßGESCHRIEBEN

Literatur

Amaral, G. J. A., I. L. Dryden und A. T. A. Wood (06/2007). „Pivotal bootstrap methods for k-sample problems in directional statistics and shape analysis“. Englisch. In: *Journal of the American Statistical Association* 102.478, S. 695–707 (siehe S. 120).

Bayes, T. (1763). „An Essay Toward Solving a Problem in the Doctrine of Chances. By the late Rev. Mr. Bayes, communicated by Mr. Price, in a letter to John Canton, M. A. and F. R. S.“ Englisch. In: *Philosophical Transactions of the Royal Society of London*, S. 370–418 (siehe S. 142).

Bookstein, F. L. (05/1986). „Size and shape spaces for landmark data in two dimensions“. Englisch. In: *Statistical Science* 1.2, S. 181–222 (siehe S. 121).

Christian, H. (2002). „Regression Fixed Point Clusters: Motivation, Consistancy and Simulations“. Englisch. In: *Journal of Classification* 19, S. 249–276 (siehe S. 149).

Clark, P. J. (1952). „An extension of the coefficient of divergence for use with multiple characters“. Englisch. In: *Copeia*, S. 61–64 (siehe S. 105).

Claude, J. (2008). *Morphometrics with R*. Englisch. Hrsg. von R. Gentleman, K. Hornik und G. Parmigiani. Use R! Springer, S. 316. ISBN: 978-0-387-77789-4. DOI: 10.1007/978-0-387-77790-0 (siehe S. 119, 185).

Coleman, B. D. u. a. (1982). „Randomness, area and species richness“. Englisch. In: *Ecology* 63, S. 1121–1133 (siehe S. 97).

Czekanowski, J. (1909). „Zur Differential Diagnose der Neandertalgruppe“. In: *Korrespondenz-Blatt deutsche Ges. Anthropol. Ethnol. Urgesch.* 40, S. 44–47 (siehe S. 105).

Dormann, C. F. und I. Kühn (07/2004). *Angewandte Statistik für die biologischen Wissenschaften*. 2. Auflage siehe Dormann und Kühn (2009), S. 1–233. URL: http://www.ufz.de/data/Dormann2004Statsskript1625.pdf (besucht am 03. 07. 2005) (siehe S. 143, 152, 163, 173).

– (10/2009). *Angewandte Statistik für die biologischen Wissenschaften*. 2. Aufl., S. 1–245. URL: http://www.ufz.de/data/deutschstatswork7649.pdf (besucht am 19. 01. 2010) (siehe S. 173).

Dryden, I. L. und K. V. Mardia (1998). *Statistical Shape Analysis*. Englisch. John Wiley & Sons, Chichester, S. 347 (siehe S. 120, 121).

Efron, B. und R. Tibshirani (1993). *An Introduction to the Bootstrap*. Englisch. Von Dormann und Kühn (2004) übernommen. Chapman & Hall (siehe S. 143).

Estabrook, G. F. und D. J. Roger (1966). „A gerneral method of taxonomic description for a computed similarity measure“. Englisch. In: *Bioscience* 16, S. 789–793 (siehe S. 104).

Faith, D. P. (1983). „Asymmetric binary similarity measures“. Englisch. In: *Oecologia (Berl.)* 57, S. 289–290 (siehe S. 105).

Frey, D. G. und E. S. Deevey (1998). „Numerical tools in palaeolimnology—Progress, potentialities, and problems“. Englisch. In: *Journal of Paleolimnology* 20, S. 307–332 (siehe S. 21).

Goodall, C. (1991). „Procrustes Methods in the Statistical Analysis of Shape“. Englisch. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 53.2, S. 285–339.

Goodall, D. W. (1964). „A probabilistic similarity index“. Englisch. In: *Nature (London)* 203, S. 1098 (siehe S. 105).

Gower, J. C. (1971a). „A general coefficient of similarity and some of its properties“. Englisch. In: *Biometrics* 27, S. 857–871 (siehe S. 104).

– (1971b). „Statistical methods of comparing different multivariate analyses of the same data“. Englisch. In: *Mathematics in the Archaeological and Historical Sciences*. Hrsg. von F. R. Hodson, D. G. Kendall und P. Tautu. Edinburgh University Press, Edinburgh, S. 138–149 (siehe S. 163).

Imbrie, J. und N. Kipp (1971). „A new micropaleontological method for quantitative paleoclimatology: application to a late Pleistocene Caribbean core“. Englisch. In: *The late Cenozoic glacial ages* 71, S. 77–181 (siehe S. 125).

Kaufman, L. und P. J. Rousseeuw (1990). *Finding groups in data*. Englisch. Wiley, New York (siehe S. 145).

Kuhl, F. P. und C. R. Giardina (1982). „Elliptic Fourier features of a closed contour“. Englisch. In: *Computer Graphics and Image Processing* 18.3, S. 236–258 (siehe S. VII, 119, 185).

Kulczynski, S. (1928). „Die Pflanzenassoziationen der Pieninen“. In: *Bull. Int. Acad. Pol. Sci. Lett. Cl. Sci. Math. Nat. Ser. B* Suppl. II. in Legendre und Legendre (1998) steht auch komischerweise: Suppl. II (1927), S. 57–203 (siehe S. 104).

Köhler, W., G. Schachtel und P. Voleske (1996). *Biostatistik*. 2. Aufl. Springer Verlag Berlin Heidelberg (siehe S. 143, 144, 149, 160, 164).

Legendre, P. und A. Chodorowski (1977). „A generalization of Jaccard's association coefficient for Q analysis of multi-state ecological data matrices“. Englisch. In: *Ekol. Pol.* 25, S. 297–308 (siehe S. 104).
Legendre, P. und E. D. Gallagher (2001). „Ecologically meaningful transformations for ordination of species data“. Englisch. In: *Oecologia* 129, S. 271–280 (siehe S. 146, 148).
Legendre, P. und L. Legendre (1998). *Numerical Ecology*. Englisch. 2. Aufl. Elsevier Science B.V., Amsterdam (siehe S. 101, 103–105, 144, 160, 161, 163, 165, 173).
Motyka, J. (1947). „O zadaniach i metodach badan geobotanicznych. Sur les buts et les méthodes des recherches géobotaniques“. Englisch. In: *Annales Universitatis Mariae Curie-Sklodowska (Lublin Polonia), Sectio C, Supplementum I*, S. 168 (siehe S. 104).
Ochiai, A. (1957). „Zoogeographic studies on the soleoid fishes found in Japan and its neighbouring regions“. Englisch. In: *Bull. Jpn. Soc. Sci. Fish.* 22.526 - 530 (siehe S. 104).
Oksanen, J. (17. 02. 2004). *Multivariate analysis in ecology – Lecture notes*. Englisch. URL: http://cc.oulu.fi/~jarioksa/opetus/metodi/notes.pdf (besucht am 28. 08. 2009) (siehe S. 107, 137).
– (2008). *Multivariate Analysis of Ecological Communities in R: vegan tutorial*. Englisch. version: May 20, 2008. URL: http://cc.oulu.fi/~jarioksa/opetus/metodi/vegantutor.pdf (siehe S. 92, 107).
Pearson, K. (1926). „On the coefficient of racial likeness“. Englisch. In: *Biometrika* 18, S. 105–117 (siehe S. 105).
Pruscha, H. (2006). *Statistisches Methodenbuch – Verfahren, Fallstudien, Programmcodes*. Universität München, Institut f. Mathematik, Theriseinstraße 39, 80333 München, Deutschland: Springer Verlag Berlin Heidelberg, S. 412 (siehe S. 93).
R Development Core Team: A language and environment for statistical computing (2005). Englisch. ISBN 3-900051-07-0, http://www.r-project.org. R Foundation for Statistical Computing. Vienna, Austria.
Racca, J. u. a. (2007). „PaleoNet: new software for building, evaluating and applying neural network based transfer functions in paleoecology“. Englisch. In: *Journal of Paleolimnology* 38.3. Software PaleoNet see http://www.cen.ulaval.ca/paleo/Paleonet/paleonet.html, S. 467–472. DOI: 10.1007/s10933-006-9082-x (siehe S. 122).
Rogers, D. J. und T. T. Tanimoto (1960). „A computer program for classifying plants“. Englisch. In: *Science (Washington D.C.)* 132, S. 1115–1118 (siehe S. 104).
Romain, F. (2006). *R graph gallery*. Englisch. http://addictedtor.free.fr/graphiques/ (siehe S. 75).
Russel, P. F. und T. R. Rao (1940). „On habitat and association of species of anopheline larvae in south-eastern Madras.“ Englisch. In: *J. Malar. Inst. India* 3, S. 153–178 (siehe S. 104).
Seyfang, L. (13. 10. 2005). *R Kurzbeschreibung*. Technische Universität Wien, S. 14. URL: http://www.statistik.tuwien.ac.at/public/filz/students/manual.pdf (siehe S. 1).
Sokal, R. R. und C. D. Michener (1958). „A statistical method for evaluating systematic relationships“. Englisch. In: *Univ. Kans. Sci. Bull.* 38, S. 1409–1438 (siehe S. 104).
Sokal, R. R. und P. H. A. Sneath (1963). *Principles of numerical taxonomy*. Englisch. W. H. Freeman, San Fransisco, S. 359 (siehe S. 104).
ter Braak, C. J. F. (1995). *Data analysis in community and landscape ecology*. Englisch. Hrsg. von R. H. Jongman, C. J. F. ter Braak und O. F. R. van Tongeren. 2. Aufl. Reprint from 1987. Cambridge University Press. Kap. 5 Ordination, S. 91–173 (siehe S. 148).
Vasko, K., H. T. T. Toivonen und A. Korhola (09/2000). „A Bayesian multinomial Gaussian response model for organism-based environmental reconstruction“. Englisch. In: *Journal of Paleolimnology* 24.3, S. 243–250. DOI: 10.1023/A:1008180500301 (siehe S. 123, 124, 126).
Watson, L., T. Williams und L. G. N. (1966). „Angiosperm taxonomy: a comparative study of some novel numerical techniques“. Englisch. In: *J Linn. Soc. Lond. Bot.* 59, S. 491–501 (siehe S. 105).
Whittaker, R. H. (1952). „Vegetation of the great smoky mountains“. Englisch. In: *Ecol. Monogr.* 26, S. 1–80 (siehe S. 105).

Anhang

Funktion 1: Zum Zeichnen von Tiefendiagrammen, wie bei Bohrkernen, Pollendiagrammen etc. Beispiele s. 56. Funktion kann in separater Datei mit `source("Pfad/plotDepth.R")` eingelesen werden. Aufpassen mit Zeilenumbruch beim Kopieren.

```
########################################
# see also below at function's arguments #
########################################
# data                dataset as a data.frame or matrix: with first column as depth
# yaxis.first=TRUE    TRUE/FALSE does first column contain depth datas?
# yaxis.num="n"       switch on/off numbers at remaining y-axes on="s" off="n"
# xaxes.equal=TRUE,   equal scaling of xaxes; can be set individually by  c(...)
# xaxis.ticks.minmax=FALSE, only minmax is drawn; can be set individually by  c(...)
# xaxis.num="s",      switch on/off numbers+ticks at x-axis on="s" off="n"
# bty="L"             boxtype as in plot: L, c, o ...; can be set individually by  c(...)
# l.type="solid"      line type default; can be set individually by  c(...)
# l.width=1,          line width; can be set individually by  list(...) or nested with c()
# lp.color="black"    line color; can be set individually by  c(...)
# plot.type="o"       type of plot - as in plot(); can be set individually by  c(...)
#    possible: o, b, c, n, h, p, l, s, S
#    "p" for points,
#    "l" for lines,
#    "b" for both,
#    "c" for the lines part alone of "b",
#    "o" for both "overplotted",
#    "h" for "histogram" like horizontal lines,
#    "s" or "S" for stair steps,
#    "n" for no plotting.
# plot.before=NULL   evaluate/draw before plotting
#                    eg.: grid() as expression(); nested: 'expression(grid())'
#                    can be set individually by  list(...) or nested with expression()
# plot.after=NULL    evaluate/draw after plotting
#                    additional graphics eg.: points(), lines() as expression()
#                    expression(lines(...)) - can be set individually by  list(...)
#                    or nested with expression()
# yaxis.lab=FALSE     no additional labels on remaining y-axes
# yaxis.ticks=TRUE    add y-ticks to graph ?
# axis.top=list(c(FALSE, FALSE)) -- x-axis also on top?
#                    call for axis and labels as c(axis=TRUE, labels=TRUE)
#                    can be nested with list( c(T,F), c(T,T), ...)
# nx.minor.ticks=5   number of intervals at x-axis if package Hmisc loadable
#                    can be set individually by  c(...)
# ny.minor.ticks=5   number of intervals at y-axis if package Hmisc loadable
#                    can be set individually by  c(...)
# mar.outer=c(1,6,4,1) --  margin at outer side: c(bottom, left , top, right)
# mar.top=9          margin at the top
# mar.bottom=5       margin at the bottom
# txt.xadj=0.1       align text at plot-top in x-axis direction: 0...1 left...right
# txt.yadj=0.1       align text at plot-top in y-axis direction: in scalenumbers
#                    + -> to the top - -> to the bottom
# colnames=TRUE      can be set individually by  c(...)
# rotation=60        text rotation: can be set individually by  c(...)
# p.type=21          type of points like pch in points()
#                    can be set individually by list(...) also nested
# p.bgcolor="white"  point background color: can be set individually by c(...)
# p.size = 1         point size: can be set individually by list(...) also nested
# subtitle=""        subtitle: can be set individually by list(...)
```

```
# xlabel=""          x-labeling: can be set individually by list(...)
# main=""            titel of individual plots: can be set individually by list(...)
# polygon=FALSE      plot polygon on/off: can be set individually by  c(...)
# polygon.color="gray" -- color of polygon plot; can be set individually by  c(...)
# show.na=TRUE       show NA values as red cross
# min.scale.level=0.2
#                   0...1 if data are less than 0.2(=20%) from maximum of the data
#                   than draw raltive 'min.scale.rel'-width for the plot
# min.scale.rel=0.5,
#                   0...1 relative space for minimal data
#                   1 means maximal width
# min.scaling=FALSE --  add upscaling plots to rare data; can be set individually by  c(...)
# color.minscale="gray95" -- color for rare scaled data; can be set individually by list(...)
# wa.order="none", sort variables according to the weighted average with y
#                  "bottomleft", "topleft" from strat.plot(palaeo - pkg)
# ...  passed to function 'lines()'
#
##############################
plot.depth <- function(
    data,# data.frame assumed with first column as y-axis
  # axis settings
    yaxis.first=TRUE, # is 1st data-column 1st y-axis?
    yaxis.num="n",    # supress labelling at remaining y-axis
    xaxis.num="s",    # show labelling at x-axis
    xaxes.equal=TRUE, # equal scaling of all x-axes
    cex.x.axis=par("cex.axis")*0.8,# size x-axis labels
    cex.y.axis=par("cex.axis")*0.8,# size y-axis labels
  # ticks/labels
    xaxis.ticks.minmax=FALSE, # only min-max?
    xaxes.adjustlabels=if(xaxis.ticks.minmax) TRUE else FALSE, # adjust labels of x-axes
    yaxis.lab=FALSE,  # axis labels on remaining y-axis?
    yaxis.ticks=TRUE, # add y-ticks to graph?
    axis.top=list(c(FALSE, FALSE)),# axis on top? c(axis=TRUE, labels=TRUE)
    nx.minor.ticks=if(xaxis.ticks.minmax) 0 else 5,# number intervals for minor ticks; 0 hides them
    ny.minor.ticks=5, # number intervals for minor ticks; 0 hides them
    bty="L",          # boxtype
  # plotting types: "o", "b", "c", "n", "h", "p", "l", "s", "S" see example(points)
    plot.type="o",    # point-plot type
    plot.before=NULL, # something to plot BEFORE the graph is drawn?
    plot.after=NULL,  # something to plot AFTER the graph is drawn?
  # lines
    l.type="solid",     # line type
    l.width=1,        # line width
    lp.color="black", # line/point color
  # points
    p.type=21,          # point type
    p.bgcolor="white",# point background color
    p.size = 1,       # point size
  # polygon
    polygon=FALSE, # plot Polygon?
    polygon.color="gray", # color polygon
  # NA - data (not available)
    show.na=TRUE,        # show missing values?
  # margins
    mar.outer=c(1,6,4,1),# outer margin of whole plot
    mar.top=   9,# margin on the top
    mar.bottom=5,# margin on the bottom
    mar.right= 0,# margin on the right side
  # minimum sclaing: for minimum data
```

```
  min.scaling=FALSE,
  color.minscale="gray95",# color for minimum scaled data
  min.scale.level=0.2,    # 0...1 plot's width: if data take less than 0.2(=20%)
  min.scale.rel  =0.5,    # 0...1 relative space for all minimal data
# texts, labels, subtitles
  txt.xadj=0.1,  # at plot-top: x-adjusting text in x-axis direction
  txt.yadj=0.1,  # at plot-top: y-adjusting text in y-axis direction
  colnames=TRUE, # add columnames
  rotation=60,   # columnames rotation
  subtitle="",   # below every plot
    xlabel="",   # x-labels
    ylabel="",   # first y-label
    main  ="",   # title for each plot
# weighted averageing of data
  wa.order="none", # "bottomleft", "topleft"
  ... # other arguments passed to other functions
){
# ------8<---- function minor.tick start
# from Hmisc package added: axis=c(1,2) + '...' for axis( , ...)
# axis=c(3,4) draws also ticks on top or right
minor.tick <- function (nx = 2, ny = 2, tick.ratio = 0.5, axis=c(1,2), ...)
{
    ax <- function(w, n, tick.ratio) {
        range <- par("usr")[if (w == "x")
            1:2
        else 3:4]
        tick.pos <- if (w == "x")
            par("xaxp")
        else par("yaxp")
        distance.between.minor <- (tick.pos[2] - tick.pos[1])/tick.pos[3]/n
        possible.minors <- tick.pos[1] - (0:100) * distance.between.minor
        low.minor <- min(possible.minors[possible.minors >= range[1]])
        if (is.na(low.minor))
            low.minor <- tick.pos[1]
        possible.minors <- tick.pos[2] + (0:100) * distance.between.minor
        hi.minor <- max(possible.minors[possible.minors <= range[2]])
        if (is.na(hi.minor))
            hi.minor <- tick.pos[2]
        if (.R.)
            axis(if (w == "x")
                axis[1]
            else axis[2], seq(low.minor, hi.minor, by = distance.between.minor),
                labels = FALSE, tcl = par("tcl") * tick.ratio, ...)
        else axis(if (w == "x")
            axis[1]
        else axis[2], seq(low.minor, hi.minor, by = distance.between.minor),
            labels = FALSE, tck = par("tck") * tick.ratio, ...)
    }
    if (nx > 1)
        ax("x", nx, tick.ratio = tick.ratio)
    if (ny > 1)
        ax("y", ny, tick.ratio = tick.ratio)
    invisible()
}
# ------8<---- function minor.tick end
# check data
if(!is.data.frame(data) & !is.matrix(data))
  stop(paste("\nFunction \'plot.depth(data, ...)\' expect a data.frame or matrix!\n Your data ↩
  are: \'",mode(data),"\'.\n  Use \"as.data.frame(depthdata) -> depthdata\" or ↩
```

```
    \"as.matrix(depthdata) -> depthdata\".", sep=""))
  if(ncol(data) < 2)
    stop("\nStop: at least 2 columns in the data!")
  if(ncol(data) > 50 && yaxis.first==FALSE){
    warning("\nOnly the first 50 columns will be drawn.")
    data <- data[,1:50]
  }
  if(ncol(data) > 51 && yaxis.first==TRUE){
    warning("\nOnly the first 50 columns will be drawn.")
    data <- data[,1:51]
  }
  nc <- ncol(data) # number of columns
  nr <- nrow(data) # number of rows
  if(yaxis.first==TRUE){# if 1st column is first y-axis
    nc.data <- nc-1 # number of columns for drawing
    draw <- 2:nc # what should be drawn
    y.depth <- data[,1] # depth scale
    y.axfirst.type ="s"
    warning("plot.depth() assumes yaxis.first=TRUE\n")
  }
  else{# no first y-axis
    nc.data <- nc# number of columns for drawing
    draw <- 1:nc # what should be drawn
    y.depth <- (1:nr)*(-1) # depth scale
    warning("Your data will be drawn as category numbers (=number of rowname)\n")
    y.axfirst.type ="n"
  }
  # weighted averageing order
  # (from package paleo http://www.campus.ncl.ac.uk/staff/Stephen.Juggins/analysis.htm)
  if (wa.order == "topleft" || wa.order == "bottomleft") {
      colsum <- colSums(data[,draw])
      opt <- (t(data[,draw]) %*% y.depth)/colsum
      if (wa.order == "topleft")
          opt.order <- rev(order(opt))
      else opt.order <- order(opt)
      draw <- opt.order
      cat("Column Index (wa.order):",draw,"\n")
      # data <- data[, opt.order]
  }

  x.maximum <- max(apply(data[,draw],2,max, na.rm=TRUE))
  x.maxima <- apply(data[,draw],2,max, na.rm=TRUE)
  # cat(x.maximum) control
  x.max <- apply(data[,draw],2,max, na.rm=TRUE)
  stopifnot(0 <= min.scale.level && min.scale.level <=1)
  stopifnot(0 <= min.scale.rel && min.scale.rel <=1)
  par(no.readonly=TRUE) -> original # save graphical settings
  # ---8<--- get settings for layout
  # maxima from each column
  apply(data[,draw],2,max, na.rm=TRUE) -> x.widths
  xwidths <- NULL # temporary vector
  for(i in 1:length(x.widths)){# for each maximum
    # allow individual settings for plots via index
    ifelse(length(xaxes.equal)==nc.data, equal.i <- i, equal.i <- 1)
    ifelse(x.widths[i]/max(x.widths) <= min.scale.level,
      {# x.widths/max <= 0.5
        xwidths[i] <- min.scale.rel # 0...min.scale.rel
        # maximum for x-axis
        ifelse(xaxes.equal[equal.i]==FALSE,
```

```
        {# draw xaxes-equal FALSE:
          x.max[i] <- max(data[,draw[i]], na.rm=TRUE) # maximum of column
        }, {#  draw xaxes-equal TRUE
          x.max[i] <- x.maximum * min.scale.rel # maximum of all data
        }
      ) # xaxes.equal
    },{# x.widths/max > 0.5
      xwidths[i] <- x.widths[i]/max(x.widths) # 0...1
      # maximum for x-axis
      ifelse(xaxes.equal[equal.i]==FALSE,
        {# FALSE:
          x.max[i] <- max(data[,draw[i]], na.rm=TRUE) # maximum of column
        },{
          x.max[i] <- x.maxima[i] # maximum of all data
        }
      ) # xaxes.equal end
    }
  ) # minscale.level end
}
# set layout
x.widths <- xwidths
layout(matrix(1:nc.data,1 , nc.data), widths=x.widths)
# ---8<--- end get settings for layout
    par(mar=c(
      mar.bottom, # bottom
      0, # left
      mar.top, # top
      ifelse(yaxis.num=="s", 1.5 + mar.right, mar.right) # right
      )+0.1,
      xpd=NA # NA to get no overplotted text
    )
  for(i in 1:length(draw)){# draw each plot
  #cat(i,"\n")
    # check for lists in list() or c() in differrent options
    ifelse(length(plot.type)          == nc.data, n.i <- i,      n.i <- 1)
    ifelse(length(ny.minor.ticks)     == nc.data, ny.i <- i,     ny.i <- 1)
    ifelse(length(nx.minor.ticks)     == nc.data, nx.i <- i,     nx.i <- 1)
    ifelse(length(polygon)            == nc.data, p.i <- i,      p.i <- 1)
    ifelse(length(min.scaling)        == nc.data, min.i <- i,    min.i <- 1)
    ifelse(length(l.type)             == nc.data, lt.i <-i,      lt.i <- 1)
    ifelse(length(lp.color)           == nc.data, lc.i <-i,      lc.i <-1)
    ifelse(length(l.width)            == nc.data, lw.i <-i,      lw.i <- 1)
    ifelse(length(p.type)             == nc.data, pt.i <-i,      pt.i <- 1)
    ifelse(length(p.size)             == nc.data, pw.i <- i,     pw.i <- 1)
    ifelse(length(p.bgcolor)          == nc.data, pbg.i <- i,    pbg.i <- 1)
    ifelse(length(colnames)           == nc.data, col.i <- i,    col.i <- 1)
    ifelse(length(rotation)           == nc.data, r.i <- i,      r.i <- 1)
    ifelse(length(xlabel)             == nc.data, xlab.i <- i,   xlab.i <- 1)
    ifelse(length(subtitle)           == nc.data, sub.i <- i,    sub.i <- 1)
    ifelse(length(main)               == nc.data, main.i <- i,   main.i <- 1)
    ifelse(length(plot.before)        == nc.data, before.i <- i, before.i <- 1)
    ifelse(length(plot.after)         == nc.data, after.i <- i,  after.i <- 1)
    ifelse(length(axis.top)           == nc.data, axtop.i <- i,  axtop.i <- 1)
    ifelse(length(xaxis.num)          == nc.data, xnum.i <- i,   xnum.i <- 1)
    ifelse(length(xaxis.ticks.minmax) == nc.data, xminmax.i <- i,xminmax.i <- 1)
    # margins of x-axis
    if(i==1) par(oma=mar.outer, xaxt=xaxis.num[xnum.i])
    else par(xaxt=xaxis.num[xnum.i])
    # axis ticks and labelling
```

```
par(
  mgp=c(3, ifelse(yaxis.num=="s" && i > 1, 0.3, 1), 0)
)
# minimum
ifelse(
  min(data[,draw[i]], na.rm=TRUE) > 0,
  x.min <- 0,# 0... max
  x.min <- min(data[,draw[i]], na.rm=TRUE) # min...max
)
# draw plot()
par(xpd=FALSE)# to draw also ylabel
plot(data[,draw[i]], y.depth,
  ann=ifelse(i==1,TRUE, FALSE),# nichts an Achse
  type="n",# Punkttyp
  yaxt=ifelse(i==1,y.axfirst.type, yaxis.num),# y-Achse an/aus
  xlim=c(x.min,x.max[i]),
  bty=ifelse(length(bty)==nc.data, bty[i], bty),
  xlab=ifelse(length(xlabel)==nc.data, xlabel[i], xlabel),
  ylab=ylabel,#ifelse(i==1, ylabel, ""),
  panel.first = eval(plot.before[[before.i]]),
  xaxt = "n" # no x-axis
)
par(xpd=FALSE)
if(i==1 && y.axfirst.type=="n"){ # draw extra first y-axis
  axis(side=2, labels=rownames(data),
    at=(1:nr)*(-1), cex.axis=cex.y.axis
  )
  box(bty=bty)
}
# draw x-axis
axTicks(1,
  axp=if(xaxis.ticks.minmax[xminmax.i]==TRUE) {c(par()$xaxp[1:2], 1)} else NULL
) -> x.axis
for(itemp in seq.int(length(x.axis) -> nx))
  text(
    x=seq(from=x.axis[1] , to=x.axis[nx], length.out=nx)[itemp],
    y=par("usr")[3] - par()$cxy[2], # coordinates - character hight
    adj = if(xaxes.adjustlabels) seq(from=0,to=1, length.out=nx)[itemp] else 0.5,
    labels = x.axis[itemp],
    cex=cex.x.axis,
    xpd=NA
  )
  axis(1, at=x.axis, labels=FALSE)

# minor ticks if package Hmisc is installed
if(yaxis.ticks==FALSE && i > 1) ny.minor.ticks[ny.i] <- 0

if(require(Hmisc)) {minor.tick(
    ny=ifelse(i==1 && y.axfirst.type=="n", 0, ny.minor.ticks[ny.i]),
    nx=nx.minor.ticks[ny.i]
  )
}
else warning("Install package 'Hmisc' to add minor ticks on axes")

# y-axis for remainig axes
if(i > 1) { axis(side=2,
  labels=yaxis.lab,
  tick=yaxis.ticks,
  cex.axis=cex.y.axis
```

```
    )
  }
  # x-axis top
  if(length(axis.top[[axtop.i]])==2){
    if(axis.top[[axtop.i]][1]==TRUE){
      axis(side=3, labels=axis.top[[axtop.i]][2], tick=TRUE, tcl=0.5, cex.axis=cex.x.axis)
      minor.tick(ny=0, nx=nx.minor.ticks[ny.i], axis=c(3,4), tcl=0.25)
    }
  }
  else warning("Option 'axis.top' wants 2 arguments as list(...):",
  "\n2nd argument is for numbers on axis, so eg.: axis.top=list(c(T, F))")
  # labelling of columns
  if(colnames[col.i]==TRUE){
    min(par()$usr[1:2]) -> x.text
    abs(max(par()$usr[1:2])-x.text)*txt.xadj -> x.adj # %-width of x-axis
    max(par()$usr[3:4]) -> y.text
    par(xpd=NA) # NA to get no overplotted text
      text(x.text+x.adj, y.text+txt.yadj, labels=colnames(data)[draw[i]], adj=0, ↩
srt=rotation[r.i] )
    par(xpd=FALSE)
  }
  # title subtitle, xlabels
  title(
    sub=subtitle[[sub.i]],
    xlab=xlabel[[xlab.i]],
    main=main[[main.i]]
  )

  # pseudo histograms; width can be set with option 'l.width'
  if( plot.type[n.i] =="h"){
    for(n in 1:nr){
      x <- c(0,data[n,draw[i]])
      y <- c(y.depth[n], y.depth[n])
      par(lend="butt") # line-End
      lines(x,y,
        lty=l.type[[lt.i]],
        lwd=l.width[[lw.i]],
        col=ifelse(length(lp.color[[lc.i]])==nr, lp.color[[lc.i]][n], lp.color[[lc.i]]),
      )
      par(lend="round")
    }
  }
  # Polygonplot
  if (polygon[p.i]==TRUE){
    # add zero values to margins where NA values occur
    # eg.: NA NA 23  7 34 84 NA NA
    #      -1 -2 -3 -4 -5 -6 -7 -8
    # to: NA NA| 0| 23  7 34 84 | 0| NA NA
    #   -1 -2 |-3| -3 -4 -5 -6 |-6| -7 -8
    data.null <- data.frame(
      rbind(
        if(!is.na(data[1, draw[i]])) cbind(y.depth[1], 0),
        cbind(y.depth[1], data[1,draw[i]])
      )
    )
    for(r in 2:nr){
      data.null <- rbind(
        as.matrix(data.null),
        # r-1==NA && r!=NA -> 0r
```

```
        if(is.na(data[r-1, draw[i]]) && !is.na(data[r, draw[i]])) cbind(y.depth[r], 0),
        # r-1!=NA && r==NA -> 0r-1
        if(!is.na(data[r-1, draw[i]]) && is.na(data[r, draw[i]])) cbind(y.depth[r-1], 0),
        as.matrix( cbind(y.depth[r], data[r, draw[i]]) ),
        # r==nr -> 0r
        if(r==nr && !is.na(data[r, draw[i]])) cbind(y.depth[r], 0)
      )
    }

    # min.scaling
    if (min.scaling[min.i]==TRUE || min.scaling[min.i] > 0){
      # default 5-scaled
      if(min.scaling[min.i]==TRUE) min.scaling[min.i] <- 5
      polygon(
        data.null[, 2]*min.scaling[min.i] ,
        data.null[, 1],
        col=ifelse(length(color.minscale)==nc.data,color.minscale[[i]],color.minscale[1]),
        xpd=FALSE
      )
      # scaling as message message
      message(paste("Column \'", colnames(data)[draw[i]],"\' is scaled ",
        min.scaling[min.i], "-times to original data.", sep="")
      )
    }# end min.scaling
    # default polygon
    polygon(
      data.null[, 2],
      data.null[, 1],
      col=ifelse(length(polygon.color)==nc.data, polygon.color[i], polygon.color),
      xpd=FALSE
    )
    # warning/recommendation, if NA in data
    if(any(is.na(data[,draw[i]]))) {warning("Column \'",
      colnames(data)[draw[i]], "\' contain NA-values.",
      "\nOther possibility to draw: switch off drawing polygon with option \'polygon=c(T, T, ↩
F, ...)\'",
      "\nand set the column to \'F\' (FALSE) than draw histogram-like lines with the ↩
following two options:",
      "\n  plot.type=c(...,\"h\",...),\n  l.width=c(..., 15, ...), ",call. = FALSE)
    }
  }# polygon end
  if(show.na==TRUE){# draw red cross, at NA-value position
    which(is.na(data[,draw[i]])) -> na.index
    # add red 'x'
    points(y=y.depth[na.index], x=rep(0, length(na.index)), pch=4, col="red")
    if(length(na.index) > 0) {
      message("With option 'show.na=FALSE' you can switch off red crosses.")
    }
  }
  # points lines ....
  lines(data[,draw[i]], y.depth,
    ann=FALSE,# nichts an Achse
    type=ifelse(plot.type[n.i]=="h", "n",plot.type[n.i]),# type of points
    lty=l.type[[lt.i]],
    lwd=l.width[[lw.i]],
    pch=p.type[[pt.i]],
    col=lp.color[[lc.i]],
    bg=p.bgcolor[[pbg.i]],
    panel.last = eval(plot.after[[after.i]]),
```

```
        cex = p.size[[pw.i]],
        ...
      )
    }# end for i
  if(xaxis.ticks.minmax && xaxes.adjustlabels) cat("x-labels are adjusted...\n")
  par(original)
}# end plot.depth
cat("#########################################################################\n# read userfunction ↩
    plot.depth(mydata, yaxis.first=TRUE):       #\n#   * plots up to 50 species as abundances for ↩
    drilling cores   #\n#   * assumes (by default) first column in mydata being y-axis  ↩
    #\n#########################################################################\n")
```

Funktion 2: `line.labels.add()` ist zum Hinzufügen einer wagerechten/senkrechten oder freien Markierungsline mit bzw. ohne Text z.B. um gewisse Abschnitte in Grafiken zu markieren. Linie wird mit Maus gesetzt. Beispiele s. auf Seite 56. Funktion kann in separater Datei mit `source("Pfad/LinieLabelsAdd.R")` eingelesen werden. Aufpassen mit Zeilenumbruch beim Kopieren.

```
line.labels.add <- function(
  wieoft=1, # wieviele Linien
  color="darkred", # Farbe Linie + Text; Liste mit c(...) moeglich
  color.bg="white", # Box-Hintergrund; Liste mit c(...) moeglich
  l.type="solid", # Linientyp; Liste mit c(...) moeglich
  l.width=1, # Linienbreite; Liste mit c(...) moeglich
  orientation="h", # h-horizontal, v-vertical n-keine
  text=FALSE, # zusaetzlicher Text; Liste mit c(...) moeglich
  border=FALSE,# Rahmen um Text; Liste mit c(...) moeglich
  xpad=0.6, # padding Abstand Boxrand-Text; Liste mit c(...) moeglich
  ypad=0.6, # padding Abstand Boxrand-Text; Liste mit c(...) moeglich
  text.scale=1, # Skalierung von Text; Liste mit c(...) moeglich
  ... # fuer text(...)
  ){
  par(xpd=T) -> original # Grafikparameter speicehrn
  for(i in 1:wieoft){ # fuer jedes 'wieoft'
    # Schalter fuer eventuelle Optionen als Liste
    ifelse(length(color)==wieoft, c.i <- i, c.i <- 1)
    ifelse(length(l.type)==wieoft, lt.i <- i, lt.i <- 1)
    ifelse(length(l.width)==wieoft, lw.i <- i, lw.i <- 1)
    ifelse(length(text)==wieoft, t.i <- i, t.i <- 1)
    ifelse(length(color.bg)==wieoft, cb.i <- i, cb.i <- 1)
    ifelse(length(border)==wieoft, b.i <- i, b.i <- 1)
    ifelse(length(orientation)==wieoft, o.i <- i, o.i <- 1)
    ifelse(length(xpad)==wieoft, xpad.i <- i, xpad.i <- 1)
    ifelse(length(ypad)==wieoft, ypad.i <- i, ypad.i <- 1)
    ifelse(length(text.scale)==wieoft, tscale.i <- i, tscale.i <- 1)
    ifelse(text!=FALSE , info <- paste("fuer Text: \"",text,"\" ", sep=""), info <- "")
    message("!> Setze mit der Maus ",info,"Beginn und Ende der Linie (",
      i ,
      " von ",wieoft,")")
    locator(2) -> wo
    if(orientation[o.i]=="h" || orientation[o.i]=="v" ){
      if(orientation[o.i]=="v") mean(wo$x) -> wo$x[1:2]
      if(orientation[o.i]=="h") mean(wo$y) -> wo$y[1:2]
    message("!> Die Funktion \'line.labels.add()\' nimmt fuer waagerechte/senkrechte",
      "\n!> Linien immer das Mittel zweier Mauspunkte.")
    }
    # Linie zeichnen
    lines(wo$x, wo$y,
      col=color[c.i], # Farbe
```

```
      lty=l.type[lt.i], # Linientyp
      lwd=l.width[lw.i] #Linienbreite
    )
    # Textausgabe
    if(text!=FALSE){
      # angepasst aus Paket boxed.labels(plotrix)
      widths <- strwidth(text, cex=text.scale[tscale.i])
      heights <- strheight(text, cex=text.scale[tscale.i])
      x.center <-  mean(wo$x)
      y.center <- mean(wo$y)
      ifelse(orientation[o.i]=="v",
        {
          x1 <- x.center - heights * xpad[xpad.i]
          x2 <- x.center + heights * xpad[xpad.i]
          y1 <- y.center - widths * ypad[ypad.i]
          y2 <- y.center + widths * ypad[ypad.i]
        },{
          x1 <- x.center - widths * xpad[xpad.i]
          x2 <- x.center + widths * xpad[xpad.i]
          y1 <- y.center - heights * ypad[ypad.i]
          y2 <- y.center + heights * ypad[ypad.i]
        }
      )
      rect(x1, y1, x2, y2, col = color.bg[cb.i], border = border[b.i])
      text(x.center, y.center,
        col=color[c.i], # Farbe
        adj=0.5,
        labels=text[t.i],
        srt=ifelse(orientation[o.i]=="v",90,0),
        cex=text.scale[tscale.i],
        ...
      )
    }# end if 'text'
  }# end for 'wieoft'
  par(original)# Grafikparameter wieder zurueck
}# end line.labels.add()
```

Funktion 3: `listExpressions()` – Listet sogenannte `expression()` auf (s.S. 41). Die `expression()` muß als Zeichenkette vorliegen innerhalb einer Liste oder Vektors. Aufpassen mit Zeilenumbruch beim Kopieren.

```
# expression auflisten
listExpressions <- function(
  x=0,          # x-Position
  y=0,          # y-Position
  title = NULL, # möglicher Titel
  expressions,  # expression z.B.: "expression('°'*C[paste(H[2]*O)])"
  vspace = 1.1, # zusätzlicheer vertikaler Platz
  cex=par("cex"), # Skalierung
  ...           # zusätzliche Argumente nach text()
  ){
  if(!is.null(title) && !is.null(expressions))
  text(
    x=x,
    y=y+strheight(title, cex=par("cex"))*1.1, # minus i-mal Buchstabenhöhe * 1.3
    labels=title,
    cex=cex
  )
  if(is.null(expressions)){
```

```
    warning("No expressions given in listExpressions(...)
    \n")
  }else{
    # maximale Buchstabenhöhe bestimmen
    maxString <- 0
    for(i in 1:length(expressions)){
      maxString[i] <- strheight(eval(parse(text=expressions[[i]])), cex=cex)
    }
    maxString <- max(maxString)
    # Ausgabe Liste
    for(i in 1:length(expressions)){
      text(
        x=x,
        y=y-maxString * vspace *i, # minus i-mal Buchstabenhöhe
        # Zeichenkettenauswerten:
        labels=eval(parse(text=expressions[[i]])),
        cex=cex,
        ... # zusätzliche Argumente wie adj, cex usw.
      )
    }
  }
}# Ende listExpressions()
```

Funktion 4: Funktionen für Umrißanalyse mittels normalisierter elliptischer Fourier Transformation nach Kuhl und Giardina (1982) und Programm SHAPE http://life.bio.sunysb.edu/morph/ > Outlines > Shape. Aufpassen mit Zeilenumbruch beim Kopieren. Siehe auch Claude (2008).

```
#####################################
# Functions for image and outline analysis using
# (normalisierter) elliptical Fourier analysis
# make functions work with:
#    source("path/to/this/ASCII-nonHTMLfile.R")
# Date: 2009-02-20 19:27:59
# Author: Andreas Plank (andreas.plank@web.de)
# Functions for Outline Programm SHAPE:
# http://life.bio.sunysb.edu/morph/ >  Outlines > Shape
# Inhalt:
#   plotchain()       - chain plot after Freemann 1974
#   readchain()       - read chain data from program SHAPE  ( *.chc files)
#   readnef()         - read nef data from program SHAPE ( *.nef files)
#   plotnef()         - plot nef data obtained from readnef()
#   chainToXY()       - convert a chain to x-y coordinates
#   harmonic.simple() - calculate harmonics
#   getharmonics()    - getharmonics from a chain
########


#####################################
# plot chain data after Herbert Freemann 1974:
#   Computer processing of line-drawing images (Computing Surveys, VoL 6, No. 1, March 1974)
# directions:
# 3 2 1
#  \|/
# 4- -0
#  /|\
# 5 6 7
# [Sample name]_[Number] [X] [Y] [Area (mm2) per pixel] [Area (pixels)]
# [Chain code] -1
```

```
plotchain <- function(
  chain,          # string from chain coding file file *.chc
  originsfift = c(0,0), # x, y shift from default
  print=FALSE,    # print coordinates
  pch=".",        # type of point: (p)lotting (ch)aracter
  polygon=FALSE, # plot polygon?
  legend = TRUE,
  main="chain plot after Freemann 1974\nComputer Processing of Line-Drawing Images",
  ...             # additional arguments to plot(...)
  ) {
  if(!is.character(chain))
  stop("\n#> 'chain' should be a character string like \"Sample1_1 121 101 1.123525 5178 5 4 4 ↩
  5 4 6 5 ... -1\"!")
  if(length(originsfift)!=2)
  stop("\n#> 'originsfift' needs 2 coordinates as 'c(x,y)'!")
  # split the chian code
  chainsplit <- strsplit(chain," ")
    #save parameters
    chainsplit[[1]][1] -> sample # sample name
    (as.numeric(chainsplit[[1]][2]) -> x)+originsfift[1] -> xstart # x
    (as.numeric(chainsplit[[1]][3]) -> y)+originsfift[2] -> ystart # y
    as.numeric(chainsplit[[1]][4]) -> areamm2  # Area (mm2) per pixel
    as.numeric(chainsplit[[1]][5]) -> areapx   # Area (pixels)
  chain <- as.numeric(chainsplit[[1]][6:(length(chainsplit[[1]])-1)]) # chain
  n.chain <- length(chain)
  # directions
  dir = c(0,1,2,3,4,5,6,7)
    # 3 2 1  directions
    #  \|/
    # 4- -0
    #  /|\
    # 5 6 7
  n.dir <- length(dir)       # number of directions
  # pi*0.0  = 0    # 0
  # pi*0.25 = 45   # 1
  # pi*0.5  = 90   # 2
  # pi*0.75 = 135  # 3
  # pi*1.0  = 180  # 4
  # pi*1.25 = 225  # 5
  # pi*1.5  = 270  # 6
  # pi*1.75 = 315  # 7
  # pi*2.0  = 360  # length(dir)+1
  # (dir.radangles   = pi*2.0/(length(dir)+1))
  (dir.radangles <- seq(0,2*pi, length.out=n.dir+1))
  chain.df <- cbind(chain,
    "x.dir" = round(cos(dir.radangles[chain+1]),0), # round to 1
    "y.dir" = round(sin(dir.radangles[chain+1]),0)  # round to 1
  )
  chain.df <- cbind(chain.df,
    "x" = cumsum(chain.df[,"x.dir"]), # cumulate x-coordinates
    "y" = cumsum(chain.df[,"y.dir"])  # cumulate y-coordinates
  )
  chain.x <- chain.df[,"x"] + xstart
  chain.y <- chain.df[,"y"] + ystart
#   print(cbind(chain.x,chain.df[,"x"]))
  plot(chain.x,chain.y,
    asp=1,
    pch=pch,main=main,...)
  if(polygon==TRUE) polygon(chain.x,chain.y,...)
```

```
    if(legend==TRUE) {
      legend("topleft",
        title=sample,
        legend=c(
          paste("start-xy: ",xstart," ",ystart,sep=""),
          paste("area: ",areapx,"px",sep=""),
          paste("area: ",areamm2,"mm^2",sep=""),
          paste("n =",n.chain)
        ),
        bty="n",
        cex = 0.8
      )
    }
  # points(chain.x[c(1,n.chain-1)],chain.y[c(1,n.chain-1)],col=c("red","blue"),pch=c(0,3))
    if(print==TRUE) {
      return(
        list(
        "coordinates" = chain.df,
        "start" = c(xstart,ystart),
        "areamm2" = areamm2,
        "areapx" = areapx,
        "n" = n.chain
        )
      )
    }
  }

#######################################
# read chain data ( *.chc files) after Herbert Freemann 1974:
#   Computer processing of line-drawing images (Computing Surveys, VoL 6, No. 1, March 1974)
# directions:
# 3 2 1
#  \|/
# 4- -0
#  /|\
# 5 6 7
# [Sample nema]_[Number] [X] [Y] [Area (mm2) per pixel] [Area (pixels]
# [Chain code] -1
  readchain <- function(
    file,    # *.chc files
    sep="\n" # \n -> line end is seperator
    ){
    data <- read.table(file,sep=sep)
    data <- as.character(data$V1)
    (data <- sub(",",".",data)) # print data as charcters
  }


#######################################
# read normalized eliptic Fourier datasets from *.nef file
  readnef <- function(
      file,      # from *.nef file
      nharmo=20, # number of harmonics
      ndata=1    # number of datasets
      ){
    for(i in 1:ndata){
      if(i==1){
        name <- read.fwf(file,
          widths=30,
```

```
          skip=2,
          n=1,
          col.names="name"
        )
        data <- read.fwf(file,
          widths=rep(15,4),
          skip=3,
          n=nharmo,
          col.names=c("a","b","c","d")
        )
        dataframe <- list(
          list(
            "nharmo" = nharmo,
            "sample" = name,
            "nefabcd" = data
          )
        )
      }
      else {
        name <- read.fwf(file,
          widths=30,
          skip=2+(nharmo+1)*(i-1),
          n=1,
          col.names="name"
        )
        data <- read.fwf(file,
          widths=rep(15,4),
          skip=3+(nharmo+1)*(i-1),
          n=nharmo,
          col.names=c("a","b","c","d")
        )
        dataframe[[i]] <- list(
          "nharmo" = nharmo,
          "sample" = name,
          "nefabcd" = data
        )
      }
    }
    dataframe # print data
  }

########################################
# function from rhelp list
# Thomas Petzoldt (Sat 19 Jan 2002 - 23:41:28 EST)
# calculate harmonics
  harmonic.simple <- function(x, a0, a, b, t, ord) {
    y <- a0
    for (p in ord) {
      k <- 2 * pi * p * x/t
      y <- y + a[p] * cos(k) + b[p] * sin(k)
    }
    y
  }# end harmonic.simple()


#####################################
# plot normalized eliptic Fourier datasets from *.nef file
  plotnef <- function(nefdf, # normalized elliptic fourier data frame
      cex=0.7,      # character expansion
```

```
      orig =c(0,0), # origin
      print=FALSE,  # print results?
      main = "normalized elliptic Fourier shape",
      linecol = "black", # line color
      lty= "dotted",     # line type
      ...           # arguments to plot()
    ){
    n <- length(nefdf$nefabcd[,1])
    x <- harmonic.simple(x=1:n,a0=0,a=nefdf$nefabcd[,"a"],b=nefdf$nefabcd[,"b"],n,1:n)#;
    y <- harmonic.simple(x=1:n,a0=0,a=nefdf$nefabcd[,"c"],b=nefdf$nefabcd[,"d"],n,1:n)#;

  #   print(nefdf$nefabcd[,1])
  #   cat(x,sep="\n")
  #   orig=c("x"=0,"y"=0)
    # angle=90
    a <- x - orig[1]
    b <- y - orig[2]
    c <- sqrt(a^2 + b^2)
    # sin(alpha) y-Komponente
    (ysin <- b/c)
    # cos(alpha) x-Komponente
    (xcos <- a/c)

    plot(x,y,
      type="n",
      asp=1,
      main=main,
      ...
    )
    lines(c(x,x[1]), c(y,y[1]),col=linecol,lty=lty)
    text(x,y,
    labels=1:n,
    cex=cex
    )
  #   lines(aspline(x,y),col="red")
  #   segments(
  #       rep(0,nx),
  #       rep(0,ny),
  #       xcos#c,
  #       ysin*c,
  #       lty="28"
  #   )

  #   abline(h=orig[1],col="gray20")
  #   abline(v=orig[0],col="gray20")
    if(print==TRUE) cbind(x,y)
  }# end plotnef()



########################################
# convert a chain to x-y coordinates
#     n <- length(nefdf$nefabcd[,1])
#     x <- harmonic.simple(x=1:n,a0=0,a=nefdf$nefabcd[,"a"],b=nefdf$nefabcd[,"b"],n,1:n)#;
#     y <- harmonic.simple(x=1:n,a0=0,a=nefdf$nefabcd[,"c"],b=nefdf$nefabcd[,"d"],n,1:n)#;
#  e.g. file 'chainfile.txt' contains name and chaincode as follows:
  # ------------8<-----------
  # NameOfPicture
  # 766776770777077070707070070070070707
```

```
# 70707707000001101010100101010101100101
# 70707070707070070707070707070070707
# 00100100100100010010010010001001001000
# 23222323232322322332223223223222222232
# 22322323232323334565665666665666553222
# ....
# ------------8<-----------
#### than do for instance:
# test <- chainToXY(file="/path/to/a/chainfile.txt")
# names(test)
# # [1] "name" "chaincode"
# harmo <- getharmonics(test$chain,n=100,verbose=FALSE)
# plot(test$x, test$y,
#   main=paste("Raw outline of:",test$name), pch=".", asp=1)
# plot(harmo$normalized$x, harmo$normalized$y,
#   asp=1, pch=".",  main=paste("Normalized elliptic Fourier reconstruction:\n",main=test$name))
# see also R-package 'shapes'

chainToXY <- function(
  chain="",
  name="givenName" ,
  file=NULL
  ){
  if (missing(chain) && is.null(file)){
    stop("Stop: chain data needed with directions:\n3   2   1\n  \\\ | /  \n4 - X - 0\n  / | ↩
  \\\  \n5   6   7\n")
  }
  if(is.character(chain)){
    if(nchar(chain)< 4 && is.null(file)) stop("Stop: length of chain must have at least 4 ↩
  elements.")
  }else if(length(chain)< 4 && is.null(file)) stop("Stop: length of chain must have at least 4 ↩
  elements.")
  if(!is.null(file)){## catch from file like:
    #PsclPs_barb_men_Tsa1999_r
    #66070770070007...
    #00070000000000...
    #
    # ^^
    # With a last line blank (=return). If it is missing only a warning will appear.
    data <- readLines(con=file, n=-11)# read all lines, first with name
    for(i in 2:length(data)){
      tmp <- gsub("(.)","\\1 ",data[i])
      tmp <- strsplit(tmp, " ")
      if(i==2) chaincode <- tmp else chaincode <- append(chaincode, tmp)
    }
    chaincode <- as.numeric(unlist(chaincode))
    chain <- list(
      name=data[1],
      chaincode = chaincode
    )
  }# end catch from file
  if(is.character(chain)){## chain given as character
    for(i in 1:length(chain)){
      tmp <- gsub("(.)","\\1 ",chain[i])
      tmp <- strsplit(tmp, " ")
      if(i==1) chaincode <- tmp else chaincode <- append(chaincode, tmp)
    }
    chaincode <- as.numeric(unlist(chaincode))
    chain <- list(
```

```
        name=name,
        chaincode = chaincode
      )
    }
    if(is.numeric(chain)){
      chaincode <- as.vector(unlist(chain))
      chain <- list(
        name=name,
        chaincode = chaincode
      )
    }
    distXY  <- 1+( (sqrt(2) -1)/2 ) * (1-(-1)^chain$chaincode)
    distXYcumsum <- cumsum(distXY)
    xdelta <- sapply(chain$chaincode,function(c){switch(as.character(c),
        '0'= 1, '1'= 1, '2'= 0, '3'= -1, '4'= -1, '5'= -1, '6'= 0, '7'= 1
      )}# end switch(cchaincode)
    )
    ydelta <- sapply(chain$chaincode,function(c){switch(as.character(c),
        '0'= 0, '1'= 1, '2'= 1, '3'= 1, '4'= 0, '5'= -1, '6'=-1, '7'=-1
      )}# end switch(chaincode)
    )
    chain <- list(
      name=chain$name,
      chain=chain$chaincode,
      distXY=distXY,
      distXYcumsum=distXYcumsum,
      xdelta = xdelta,
      ydelta = ydelta,
      x = cumsum(xdelta),
      y = cumsum(ydelta)
    )
    # output
    return(chain)
  } # end chainToXY()


########################################
# get harmonics to reconstruct an image outline by
# (normalized) elliptic Fourier transformation
#
  getharmonics <- function(
    chain,# chaincode
    n=20, # number of harmonics
    verbose=TRUE,
    debug=FALSE){
#     if (missing(chain)){
#       stop("Stop: chain data needed with directions:\n3   2   1\n  \\\ | /  \n4 - X - 0\n  / | ↩
    \\\  \n5   6   7\n")
#     }
#     if(nchar(chain)< 4 ) stop("Stop: length of chain must have at least 4.")
    chain <- chainToXY(chain)
    tp <- append(chain$distXYcumsum, values=0, after=0)
    # normally tp[p] and tp[p-1] but here 0 value is prepended, because tp[p-1] cause an error
    # therefore here is tp[p+1] and tp[p] used
    dT <- chain$distXY
    x  <- chain$x
    y  <- chain$y
    dx <- chain$xdelta
    dy <- chain$ydelta
```

```
    T  <- sum(chain$distXY)
    K  <- length(dx)
    Asums=numeric(n);
    Bsums=numeric(n);
    Csums=numeric(n);
    Dsums=numeric(n);
    An=numeric(n);
    Bn=numeric(n);
    Cn=numeric(n);
    Dn=numeric(n);
    ndig <- ceiling(log10(n))
    an =0
    an <- T/(2*(1:n)^2*pi^2)
    t1 <- Sys.time()
    for(ni in 1:n){
      if(ni==1 && verbose)   cat("Calculate Fourier descriptors for (",n," harmonics)\n", sep="")
      #cat(sprintf(paste("\n%1$",ndig,"s: ", sep=""), n-ni), sep="")
      if(debug && ni==1)## some information
        cat("
+------- debug information ------+
|                    .-\"-.       |
|     .-\"-.       @@/     \\      |
|    /     \\@@    Y '-<<<-'      |
|    '->>>-' Y        '''         |
| jgs (http://www.ascii-art.de/) |
+--------------------------------+\n",
          "T=",T," (total length)\nK=",K," (length of chain)",
          sprintf("\n\nFor each Harmonic number 1...%2$s run through points 1...%3$s:",ni,n,K),
          "\n",sep=""
        )

      p <- 1:K
        Asums[ni] <- sum(dx[p]/dT[p] * ( cos( 2 * ni * pi * tp[p+1] / T) - ( cos( 2 * ni * pi * ↩
    tp[p] / T ) ) ) )
        Bsums[ni] <- sum(dx[p]/dT[p] * ( sin( 2 * ni * pi * tp[p+1] / T) - ( sin( 2 * ni * pi * ↩
    tp[p] / T ) ) ) )
        Csums[ni] <- sum(dy[p]/dT[p] * ( cos( 2 * ni * pi * tp[p+1] / T) - ( cos( 2 * ni * pi * ↩
    tp[p] / T ) ) ) )
        Dsums[ni] <- sum(dy[p]/dT[p] * ( sin( 2 * ni * pi * tp[p+1] / T) - ( sin( 2 * ni * pi * ↩
    tp[p] / T ) ) ) )

        if(debug && ni==1){## some information
          dig=ceiling(log10(T))
          cat(
            sprintf(
              paste("p %1$",dig,
                "s: dX dY(%2$2s,%3$2s), differences XY: dT(%4$1.3f) and cumsum XY: tp tp-1(%6$",
                (dig+4),
                ".3f, %5$",
                (dig+4),
                ".3f)", sep=""
              ),
              p , dx[p], dy[p], dT[p], if(p==1) 0 else tp[p-1], tp[p]
            ),"\n"
          )## end cat()
        }## end debug
      An[ni] <- an[ni] * Asums[ni]
      Bn[ni] <- an[ni] * Bsums[ni]
      Cn[ni] <- an[ni] * Csums[ni]
```

```
  Dn[ni] <- an[ni] * Dsums[ni]
  if(verbose) cat(".", if(ni %% 50 ==0) "\n" else "", sep="")
  if(ni==n && verbose) cat("done\n")
}# end for 1:n
aNorm = numeric(n); bNorm = numeric(n); cNorm = numeric(n); dNorm = numeric(n);
aStar = numeric(n); bStar = numeric(n); cStar = numeric(n); dStar = numeric(n);
# normA = numeric(n)
# normB = numeric(n)
# normC = numeric(n)
# normD = numeric(n)
theta = numeric(n)
for(ni in 1:n) {##calculate matrices
    # theta
  if(ni==1 && verbose)      cat("Calculate normalization\n")
  if(verbose) cat(".", if(ni %% 50 ==0) "\n" else "", sep="")
  if(ni==n && verbose) cat("done\n")

    theta[ni]  =
        0.5 *
          atan(
            2*( An[ni]*Bn[ni] + Cn[ni]*Dn[ni] )
              /
            ( An[ni]^2 + Cn[ni]^2 - Bn[ni]^2 - Dn[ni]^2 )
          )
      ;
    aStar[ni] = (An[1] *        cos( theta[ni] )) + (Bn[1] * sin( theta[ni] ));
    bStar[ni] = (An[1] * (-1) * sin( theta[ni] )) + (Bn[1] * cos( theta[ni] ));
    cStar[ni] = (Cn[1] *        cos( theta[ni] )) + (Dn[1] * sin( theta[ni] ));
    dStar[ni] = (Cn[1] * (-1) * sin( theta[ni] )) + (Dn[1] * cos( theta[ni] ));
  if(ni==1){
    E_Star = sqrt( (aStar[1]^2  + cStar[1]^2)  );
    psi    =   atan( (cStar[1] / aStar[1]) );
      ## returns '+' but PHP returns '-' but range after Kuhl and Giardina 1982 0 <= psi_1 <= 2PI
  }
  if(debug){
    cat(
      sprintf(
        "theta[ni=%8$2s]:%1$32.50f psi=%2$8.5f E*=%3$8.5f\n  a*=%4$+-10.5e b*=%5$+-10.5e  ↩
c*=%6$+-10.5e d*=%7$+-10.5e cos=%9$8.5f sin=%10$8.5e",
         theta[ni] , psi, E_Star, aStar[1], bStar[1], cStar[1], dStar[1], ni, cos( theta[ni] ↩
), sin( theta[ni] )
      ),
      "\n", sep=""
    );
    cat(
      "  a*=",An[1] ,"*", cos( theta[ni] ) ,"+", Bn[1] ,"*", sin( theta[ni] ),
      "\n"
    )
  }
  # m12_11 =  cos(psi)*An[ni] + sin(psi)*Cn[ni]; m12_12 =  cos(psi)*Bn[ni] + sin(psi)*Dn[ni];
  # m12_21 = -sin(psi)*An[ni] + cos(psi)*Cn[ni]; m12_22 = -sin(psi)*Bn[ni] + cos(psi)*Dn[ni];
  #
  # m23_11 = m12_11 * cos(ni*theta[1]) + m12_12 * sin(ni*theta[1]);  m23_12 = m12_11 * (-1) * ↩
sin(ni*theta[1]) + m12_12 * cos(ni*theta[1]);
  # m23_21 = m12_21 * cos(ni*theta[1]) + m12_22 * sin(ni*theta[1]);  m23_22 = m12_21 * (-1) * ↩
sin(ni*theta[1]) + m12_22 * cos(ni*theta[1]);
  #
  # aNorm[ni] = 1 / E_Star * m23_11;       bNorm[ni] = 1 / E_Star * m23_12;
  # cNorm[ni] = 1 / E_Star * m23_21;       dNorm[ni] = 1 / E_Star * m23_22;
```

```
    m1 <- matrix(c(       cos(psi),       sin(psi),
                         -sin(psi),       cos(psi)), 2, 2, byrow=TRUE)
    m2 <- matrix(c(       An[ni],       Bn[ni],
                          Cn[ni],       Dn[ni]), 2, 2, byrow=TRUE)
    m3 <- matrix(c( cos(ni*theta[1]), -sin(ni*theta[1]),
                    sin(ni*theta[1]),  cos(ni*theta[1])), 2, 2, byrow=TRUE)
    aNorm[ni] <- (1/E_Star *(m1 %*% m2 %*% m3))[1,1]
    bNorm[ni] <- (1/E_Star *(m1 %*% m2 %*% m3))[1,2]
    cNorm[ni] <- (1/E_Star *(m1 %*% m2 %*% m3))[2,1]
    dNorm[ni] <- (1/E_Star *(m1 %*% m2 %*% m3))[2,2]

  }## end calculate matrices
## return reconstructed x y
if(verbose) cat("Reconstruct x-y values...\n")
xNorm <- harmonic.simple(x=1:n,a0=0,a=aNorm,b=bNorm,n,1:n)#;
yNorm <- harmonic.simple(x=1:n,a0=0,a=cNorm,b=dNorm,n,1:n)#;
xnotNorm <- harmonic.simple(x=1:n,a0=0,a=An,b=Bn,n,1:n)#;
ynotNorm <- harmonic.simple(x=1:n,a0=0,a=Cn,b=Dn,n,1:n)#;
    #cat(time,units(time)," ")
  harmo <- list(
    nharmo = n,
    notnormalized = data.frame(
      a = An,
      b = Bn,
      c = Cn,
      d = Dn,
      x = xnotNorm,
      y = ynotNorm
    ),
    normalized = data.frame(
      a = aNorm,
      b = bNorm,
      c = cNorm,
      d = dNorm,
      x = xNorm,
      y = yNorm
    )
  )
    t2 <- Sys.time()
    time <- round(difftime(t2,t1), 3)
  if(verbose) cat("Normalized and not normalized values returned\nDone calculation in ", time," ↩
  ",units(time),".\n", sep="")
  return(harmo) # output values
} # end getharmonics()
```

Funktion 5: Funktion `modelEquation(lm-modell, ndigits, format)` für lineare Modellgleichungen. Wird als `expression()` ausgegeben.

```
# return equation y = b * x +a as expression
modelEquation <- function(
  mod,
  ndigits=4, # number of digits
  format="f" # sprintf - format:
    #'e' -> scientific
    #'f' -> float
  ){
  if (missing(mod))
    stop("Stop here: model needed.")
```

```
  if(class(modLLSESP)!="lm")# no linear model
    stop("Stop here: only for linear models.")
  # prepare y = slope * var + intercept
    mod.coef <- coef(mod) # coefficients of model
    mod.formula <-
      if(length(mod.coef)==2) # y = slope * var + intercept
        paste(
          "y ==",
          # slope
          sprintf(paste("%1$ .",ndigits,format, sep=""), mod.coef[2]),
          "%.%",
          # variable
          names(mod.coef[2]),
          # intercept
          sprintf(paste("%1$+.",ndigits,format, sep=""), mod.coef[1])
        ) else paste( # y = slope * var
          "y ==",
          # slope
          sprintf(paste("%1$+.",ndigits,format, sep=""), mod.coef[1]),
          "%.%",
          # variable
          names(mod.coef[1])
        )
    mod.formula <- parse(text=eval(mod.formula))
  return(mod.formula) # return as expression
} # end modelEquation()
# USAGE:
# ?cars # dataset cars
# speed <- cars$speed
# mod.lm <- lm(cars$dist~speed)
# plot(cars$dist~cars$speed)
# abline(mod.lm) # model line
# legend(
#   "topleft",
#   legend=modelEquation(mod.lm),
#   bty="n"
# )
```

Funktion 6: Funktion `textWithBGColor(text, type, ...)` ist für Internettexte, die etwas mit Hintergrundfarbe ausgefüllt werden. Farben sind: Regenbogenfarben: Farbe?, Grauwerte: Grau? oder schwarz/weiß: SW .

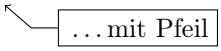
```
# Creating web colored text with a background color:
# rainbow colors, gray scale colors or black/white
textWithBGColor <- function(
  text, # the text
  type="rainbow", # types: "rainbow", "gray", "bw"
  addWhiteSpace=TRUE,
   ... # pass along to other functions
  ){
  if(missing(text)) # stop here with example
    stop("Stop here 'text = \"some text\"' is missing. Usage:\ntextWithBGColor(text='My ↩
    text.')\ntextWithBGColor(text='My text.', type=\"gray\", s=0.3)\n # see also Help pages for ↩
    ?rainbow() and ?gray.colors() (-> for additionals: saturation etc.)")
    if(addWhiteSpace){# addWhiteSpace if needed
      text <- paste(" ", text, " ", sep="")
    }
    ntext <- nchar(text) # number of characters
```

```
    ncolrange <- 1:ntext # 1, 2, ..., ntext
    for(icol in ncolrange){# run for all index-colors
      cat(
        sprintf(# compose HTML-tag <span>...</span>
          '<span style="background-color:%1$7.7s;%3$s">%2$s</span>',
          switch(type, # first argument %1
            rainbow = rainbow(ntext, ...)[icol],
            gray = gray.colors(ntext, ...)[icol],
            grey = gray.colors(ntext, ...)[icol],
            bw = if(icol <= ntext %/% 2 ) "#000000" else "#FFFFFF",
            rainbow(ntext, ...)[icol] # default
          ),# end switch(type)
          # second argument %2
          sub(" ", " ", substring(text, icol, icol )),
          # third argument %3
          switch(type, bw = if(icol <= ntext %/% 2 ) " color:#FFFFFF;" else " color:#000000;", "")
        ),# end compose HTML-tag <span>...</span>
        sep="" #no space here
      )# end cat print properly to prompt
    }# end for 1:ntext
    cat("\n") # add a line break
} # end textWithBGColor()
textWithBGColor(text='Farbe?', s=0.7) # Farbe?
textWithBGColor(text='Grau?', type="gray", s=0.3) # Grau?
textWithBGColor(text='SW', type="bw") # SW
```

Funktion 7: Funktion `asking(question, answerNo, answerYes)` ist fordert den Benutzer auf, eine Frage zu beantworten. Als Entscheidungsweiche quasi.

```
## ask user what to do
asking <- function(
    question="Are you a satisfied R user?",
    answerNo="This is impossible. YOU LIED!",
    answerYes="I knew it.",
    prompt="n",
    stop = FALSE # can force to stop the script
  ) {
  question <- paste(question,"(y/n)\n")
  cat("\a") # alarm for Linux
  ANSWER <- readline(question)
  #   cat(ANSWER)
  if (substr(ANSWER, 1, 1) == prompt){# no answer
    cat(answerNo,"\n")
    return(FALSE)# returns FALSE
    if(stop)
      stop("Breake here - user stopped.")
  }else{
    cat(answerYes,"\n") # green
    return(TRUE) # returns TRUE
  }
}
# asking() # after an idea of R-example ?readline
# asking("Continue? It takes long time: ","Stop here.","OK, continue ...")
# if(asking(...)) {then do a lot o R stuff} else {do another lot o R stuff}
cat("Read asking(question, no, yes): asks the user for input...\n")
```

Funktion 8: Die Funktion `arrowLegend()` plaziert eine Legende mit der Maus an eine Stelle und zusätzlich mit einem Pfeil.




```
# draw a legend at given points additionally indicated by an arrow
arrowLegend <- function(
  text,      # legend text
  npoints=2, # pick 1-3 points with mouse
  lineHoirz=TRUE, # 2nd → 3rd point: draws a horizontal line
  adj=c(0, 0.5), # adjusting text
  ... # further args from legend()
  ){
  if(missing(text))
    stop("Stop here 'text' missing. Usage: arrowLegend(text='my text')")
    if(npoints >= 1 &  npoints <= 3){
      xjust = 0 # xalign
      cat(paste("Please select", npoints, "points for the arrow. Last point places the ↩
    legend...\n"))
      xy <- locator(npoints)
      if(npoints!=1){# 2 or 3 points
        # adjust Box+Text to the given x-coordinates
        xjust <- if(xy$x[npoints-1] < xy$x[npoints]) 0 else 1
        if(npoints ==3) {# 3 points? add segments(...)
          if(lineHoirz){
            xy$y[2:3]  <- mean(xy$y[2:3])
          }
          segments( xy$x[2], xy$y[2], xy$x[3], xy$y[3] )
        }
        arrows( xy$x[1], xy$y[1], xy$x[2], xy$y[2] , code = 1, length = 0.12)
      }# 2 or 3 points
      legend(
        x = xy$x[npoints] , y = xy$y[npoints],
        legend = text,
        yjust=0.5,
        # do x-adjusting
        xjust = xjust, adj = adj,
        ...
      )
    }else {# all other points
      cat(paste(npoints, "points are not allowed...\n"))
    }
    cat('xjust:',xjust,"\n")
} # end arrowLegend()
cat("Read arrowLegend(text): draw a legend at given mouse click points additionally indicated by ↩
    an arrow...\n")
```

Funktion 9: Die Funktion `grDeviceUserSize()` ist voreingestellt auf „Querformat“, d.h. ein Verhältnis von 12/7. Andere Proportionen können auch angegeben werden.

```
## defines size of graphic device on Linux/Windows Apple?
grDeviceUserSize <- function(scale=12/7, dinMin = 6.9, dinMax = 7.1){
  # some information prompt
  cat("w:",mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),
      "h:",mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1), 'dev.cur():',dev.cur(),"\n")
  if(dev.cur()==1){ # wenn kein device (also NULL) dann:
    # neues Grafikfenster
    switch(tolower(Sys.info()["sysname"]),
      linux     = {
```

```
        X11(
          width = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),
          height= mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      },# end Linux
      windows = {
        windows(
          width = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),
          height= mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      },# end Windows
      quartz   = {
        quartz(
          width = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),
          height= mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      } # end MacOS
    )# end switch(Sys.info())
  }else if(
    any(
      par()$din[1] < dinMin * ifelse(scale>=1, scale, 1) ||
        par()$din[1] > dinMax * ifelse(scale>=1, scale, 1),
      par()$din[2] < dinMin * ifelse(scale<1, 1/scale, 1) ||
        par()$din[2] > dinMax * ifelse(scale<1, 1/scale, 1)
    )
  ){ # andernfalls
    #cat(par()$din,"\n")
    dev.off() # device off = close graphic device
    # neues Grafikfenster
    switch(Sys.info()["sysname"],
      linux     = {
        X11(
          width  = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),#*scale,
          height = mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      },# end Linux
      windows = {
        windows(
          width  = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),#*scale,
          height = mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      },# end Windows
      quartz   = {
        quartz(
          width  = mean(c(dinMin,dinMax)) * ifelse(scale>=1, scale, 1),#*scale,
          height = mean(c(dinMin,dinMax)) * ifelse(scale<1, 1/scale, 1)
        )
      } # end MacOS
    )# end switch(Sys.info())
  }
} # end grDeviceUserSize()
cat("Read grDeviceUserSize(): defines size of graphic device. Default is landscape...\n")
```

Die folgende Kurzreferenz ist von Tom Short s. unter http://www.Rpad.org Version vom 2005-07-12

R Reference Card

by Tom Short, EPRI Solutions, Inc., tshort@eprisolutions.com 2005-07-12

Help and basics

Most R functions have online documentation.

help(topic) documentation on topic
?topic id.
help.search("topic") search the help system
apropos("topic") the names of all objects in the search list matching the regular expression "topic"
help.start() start the HTML version of help
str(a) display the internal *str*ucture of an R object
summary(a) gives a "summary" of a, usually a statistical summary but it is *generic* meaning it has different operations for different classes of a
ls() show objects in the search path; specify pat="pat" to search on a pattern
ls.str() str() for each variable in the search path
dir() show files in the current directory
methods(a) shows S3 methods of a
methods(class=class(a)) lists all the methods to handle objects of class a
options(...) set or examine many global options; common ones: width, digits, error
library(x) load add-on packages; library(help=x) lists datasets and functions in package x.
attach(x) database x to the R search path; x can be a list, data frame, or R data file created with save. Use search() to show the search path.
detach(x) x from the R search path; x can be a name or character string of an object previously attached or a package.

Input and output

load() load the datasets written with save
data(x) loads specified data sets
read.table(file) reads a file in table format and creates a data frame from it; the default separator sep="" is any whitespace; use header=TRUE to read the first line as a header of column names; use as.is=TRUE to prevent character vectors from being converted to factors; use comment.char="" to prevent "#" from being interpreted as a comment; use skip=n to skip n lines before reading data; see the help for options on row naming, NA treatment, and others
read.csv("filename",header=TRUE) id. but with defaults set for reading comma-delimited files
read.delim("filename",header=TRUE) id. but with defaults set for reading tab-delimited files
read.fwf(file,widths,header=FALSE,sep="",as.is=FALSE) read a table of *f*ixed *w*idth *f*ormatted data into a 'data.frame'; widths is an integer vector, giving the widths of the fixed-width fields
save(file,...) saves the specified objects (...) in the XDR platform-independent binary format
save.image(file) saves all objects
cat(..., file="", sep=" ") prints the arguments after coercing to character; sep is the character separator between arguments
print(a, ...) prints its arguments; generic, meaning it can have different methods for different objects
format(x,...) format an R object for pretty printing
write.table(x,file="",row.names=TRUE,col.names=TRUE, sep=" ") prints x after converting to a data frame; if quote is TRUE, character or factor columns are surrounded by quotes ("); sep is the field separator; eol is the end-of-line separator; na is the string for missing values; use col.names=NA to add a blank column header to get the column headers aligned correctly for spreadsheet input
sink(file) output to file, until sink()

Most of the I/O functions have a file argument. This can often be a character string naming a file or a connection. file="" means the standard input or output. Connections can include files, pipes, zipped files, and R variables.

On windows, the file connection can also be used with description = "clipboard". To read a table copied from Excel, use

```
x <- read.delim("clipboard")
```

To write a table to the clipboard for Excel, use

```
write.table(x,"clipboard",sep="\t",col.names=NA)
```

For database interaction, see packages RODBC, DBI, RMySQL, RPgSQL, and ROracle. See packages XML, hdf5, netCDF for reading other file formats.

Data creation

c(...) generic function to combine arguments with the default forming a vector; with recursive=TRUE descends through lists combining all elements into one vector
from:to generates a sequence; ":" has operator priority; 1:4 + 1 is "2,3,4,5"
seq(from,to) generates a sequence by= specifies increment; length= specifies desired length
seq(along=x) generates 1, 2, ..., length(x); useful for for loops
rep(x,times) replicate x times; use each= to repeat "each" element of x each times; rep(c(1,2,3),2) is 1 2 3 1 2 3; rep(c(1,2,3),each=2) is 1 1 2 2 3 3
data.frame(...) create a data frame of the named or unnamed arguments; data.frame(v=1:4,ch=c("a","B","c","d"),n=10); shorter vectors are recycled to the length of the longest
list(...) create a list of the named or unnamed arguments; list(a=c(1,2),b="hi",c=3i);
array(x,dim=) array with data x; specify dimensions like dim=c(3,4,2); elements of x recycle if x is not long enough
matrix(x,nrow=,ncol=) matrix; elements of x recycle
factor(x,levels=) encodes a vector x as a factor
gl(n,k,length=n*k,labels=1:n) generate levels (factors) by specifying the pattern of their levels; k is the number of levels, and n is the number of replications
expand.grid() a data frame from all combinations of the supplied vectors or factors
rbind(...) combine arguments by rows for matrices, data frames, and others
cbind(...) id. by columns

Slicing and extracting data

Indexing lists

x[n]	list with elements n
x[[n]]	n^{th} element of the list
x[["name"]]	element of the list named "name"
x$name	id.

Indexing vectors

x[n]	n^{th} element
x[-n]	all *but* the n^{th} element
x[1:n]	first n elements
x[-(1:n)]	elements from n+1 to the end
x[c(1,4,2)]	specific elements
x["name"]	element named "name"
x[x > 3]	all elements greater than 3
x[x > 3 & x < 5]	all elements between 3 and 5
x[x %in% c("a","and","the")]	elements in the given set

Indexing matrices

x[i,j]	element at row i, column j
x[i,]	row i
x[,j]	column j
x[,c(1,3)]	columns 1 and 3
x["name",]	row named "name"

Indexing data frames (matrix indexing plus the following)

x[["name"]]	column named "name"
x$name	id.

Variable conversion

as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ... convert type; for a complete list, use methods(as)

Variable information

is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ... test for type; for a complete list, use methods(is)
length(x) number of elements in x
dim(x) Retrieve or set the dimension of an object; dim(x) <- c(3,2)
dimnames(x) Retrieve or set the dimension names of an object
nrow(x) number of rows; NROW(x) is the same but treats a vector as a one-row matrix
ncol(x) and **NCOL(x)** id. for columns
class(x) get or set the class of x; class(x) <- "myclass"
unclass(x) remove the class attribute of x
attr(x,which) get or set the attribute which of x
attributes(obj) get or set the list of attributes of obj

Data selection and manipulation

which.max(x) returns the index of the greatest element of **x**
which.min(x) returns the index of the smallest element of **x**
rev(x) reverses the elements of x

sort(x) sorts the elements of x in increasing order; to sort in decreasing order: rev(sort(x))
cut(x,breaks) divides x into intervals (factors); breaks is the number of cut intervals or a vector of cut points
match(x, y) returns a vector of the same length than x with the elements of x which are in y (NA otherwise)
which(x == a) returns a vector of the indices of x if the comparison operation is true (TRUE), in this example the values of i for which x[i] == a (the argument of this function must be a variable of mode logical)
choose(n, k) computes the combinations of k events among n repetitions = $n!/[(n-k)!k!]$
na.omit(x) suppresses the observations with missing data (NA) (suppresses the corresponding line if x is a matrix or a data frame)
na.fail(x) returns an error message if x contains at least one NA
unique(x) if x is a vector or a data frame, returns a similar object but with the duplicate elements suppressed
table(x) returns a table with the numbers of the differents values of x (typically for integers or factors)
subset(x, ...) returns a selection of x with respect to criteria (..., typically comparisons: x$V1 < 10); if x is a data frame, the option select gives the variables to be kept or dropped using a minus sign
sample(x, size) resample randomly and without replacement size elements in the vector x, the option replace = TRUE allows to resample with replacement
prop.table(x,margin=) table entries as fraction of marginal table

Math

sin,cos,tan,asin,acos,atan,atan2,log,log10,exp
max(x) maximum of the elements of x
min(x) minimum of the elements of x
range(x) id. then c(min(x), max(x))
sum(x) sum of the elements of x
diff(x) lagged and iterated differences of vector x
prod(x) product of the elements of x
mean(x) mean of the elements of x
median(x) median of the elements of x
quantile(x,probs=) sample quantiles corresponding to the given probabilities (defaults to 0,.25,.5,.75,1)
weighted.mean(x, w) mean of x with weights w
rank(x) ranks of the elements of x
var(x) or cov(x) variance of the elements of x (calculated on $n-1$); if x is a matrix or a data frame, the variance-covariance matrix is calculated
sd(x) standard deviation of x
cor(x) correlation matrix of x if it is a matrix or a data frame (1 if x is a vector)
var(x, y) or cov(x, y) covariance between x and y, or between the columns of x and those of y if they are matrices or data frames
cor(x, y) linear correlation between x and y, or correlation matrix if they are matrices or data frames
round(x, n) rounds the elements of x to n decimals
log(x, base) computes the logarithm of x with base base
scale(x) if x is a matrix, centers and scales the data; to center only use the option scale=FALSE, to scale only center=FALSE (by default center=TRUE, scale=TRUE)
pmin(x,y,...) a vector which ith element is the minimum of x[i], y[i], ...
pmax(x,y,...) id. for the maximum
cumsum(x) a vector which ith element is the sum from x[1] to x[i]
cumprod(x) id. for the product
cummin(x) id. for the minimum
cummax(x) id. for the maximum
union(x,y), **intersect(x,y)**, **setdiff(x,y)**, **setequal(x,y)**, **is.element(el,set)** "set" functions
Re(x) real part of a complex number
Im(x) imaginary part
Mod(x) modulus; abs(x) is the same
Arg(x) angle in radians of the complex number
Conj(x) complex conjugate
convolve(x,y) compute the several kinds of convolutions of two sequences
fft(x) Fast Fourier Transform of an array
mvfft(x) FFT of each column of a matrix
filter(x,filter) applies linear filtering to a univariate time series or to each series separately of a multivariate time series

Many math functions have a logical parameter na.rm=FALSE to specify missing data (NA) removal.

Matrices

t(x) transpose
diag(x) diagonal
%*% matrix multiplication
solve(a,b) solves a %*% x = b for x
solve(a) matrix inverse of a
rowsum(x) sum of rows for a matrix-like object; **rowSums(x)** is a faster version
colsum(x), **colSums(x)** id. for columns
rowMeans(x) fast version of row means
colMeans(x) id. for columns

Advanced data processing

apply(X,INDEX,FUN=) a vector or array or list of values obtained by applying a function FUN to margins (INDEX) of X
lapply(X,FUN) apply FUN to each element of the list X
tapply(X,INDEX,FUN=) apply FUN to each cell of a ragged array given by X with indexes INDEX
by(data,INDEX,FUN) apply FUN to data frame data subsetted by INDEX
ave(x,...,FUN=mean) subsets of x are averaged (or other function specified by FUN), where each subset consist of those observations with the same factor levels
merge(a,b) merge two data frames by common columns or row names
xtabs(a b,data=x) a contingency table from cross-classifying factors
aggregate(x,by,FUN) splits the data frame x into subsets, computes summary statistics for each, and returns the result in a convenient form; by is a list of grouping elements, each as long as the variables in x
stack(x, ...) transform data available as separate columns in a data frame or list into a single column
unstack(x, ...) inverse of stack()
reshape(x, ...) reshapes a data frame between 'wide' format with repeated measurements in separate columns of the same record and 'long' format with the repeated measurements in separate records; use (direction="wide") or (direction="long")

Strings

paste(...) concatenate vectors after converting to character; sep= is the string to separate terms (a single space is the default); collapse= is an optional string to separate "collapsed" results
substr(x,start,stop) substrings in a character vector; can also assign, as substr(x, start, stop) <- value
strsplit(x,split) split x according to the substring split
grep(pattern,x) searches for matches to pattern within x; see ?regex
gsub(pattern,replacement,x) replacement of matches determined by regular expression matching sub() is the same but only replaces the first occurrence.
tolower(x) convert to lowercase
toupper(x) convert to uppercase
match(x,table) a vector of the positions of first matches for the elements of x among table
x %in% table id. but returns a logical vector
pmatch(x,table) partial matches for the elements of x among table
nchar(x) number of characters
strwidth(s, units = "user", cex = NULL), strheight(...) width or height of characters

Dates and times

The class Date has dates without times. POSIXct has dates and times, including time zones. Comparisons (e.g. >), seq(), and difftime() are useful. Date also allows + and −. ?DateTimeClasses gives more information. See also package chron.

as.Date(s) and **as.POSIXct(s)** convert to the respective class; format(dt) converts to a string representation. The default string format is "2001-02-21". These accept a second argument to specify a format for conversion. Some common formats are:

%a, %A Abbreviated and full weekday name.
%b, %B Abbreviated and full month name.
%d Day of the month (01–31).
%H Hours (00–23).
%I Hours (01–12).
%j Day of year (001–366).
%m Month (01–12).
%M Minute (00–59).
%p AM/PM indicator.
%S Second as decimal number (00–61).
%U Week (00–53); the first Sunday as day 1 of week 1.
%w Weekday (0–6, Sunday is 0).
%W Week (00–53); the first Monday as day 1 of week 1.
%y Year without century (00–99). Don't use.
%Y Year with century.
%z (output only.) Offset from Greenwich; -0800 is 8 hours west of.
%Z (output only.) Time zone as a character string (empty if not available).

Where leading zeros are shown they will be used on output but are optional on input. See `?strftime`.

Graphics devices

`x11(), windows()` open a graphics window
`postscript(file)` starts the graphics device driver for producing PostScript graphics; use `horizontal = FALSE, onefile = FALSE, paper = "special"` for EPS files; `family=` specifies the font (AvantGarde, Bookman, Courier, Helvetica, Helvetica-Narrow, NewCenturySchoolbook, Palatino, Times, or ComputerModern); `width=` and `height=` specifies the size of the region in inches (for `paper="special"`, these specify the paper size).
`ps.options()` set and view (if called without arguments) default values for the arguments to `postscript`
`pdf, png, jpeg, bitmap, xfig, pictex`; see `?Devices`
`dev.off()` shuts down the specified (default is the current) graphics device; see also `dev.cur`, `dev.set`

Plotting

`plot(x)` plot of the values of `x` (on the *y*-axis) ordered on the *x*-axis
`plot(x, y)` bivariate plot of `x` (on the *x*-axis) and `y` (on the *y*-axis)
`hist(x)` histogram of the frequencies of `x`
`barplot(x)` histogram of the values of `x`; use `horiz=FALSE` for horizontal bars
`dotchart(x)` if `x` is a data frame, plots a Cleveland dot plot (stacked plots line-by-line and column-by-column)
`pie(x)` circular pie-chart
`boxplot(x)` "box-and-whiskers" plot
`sunflowerplot(x, y)` id. than `plot()` but the points with similar coordinates are drawn as flowers which petal number represents the number of points
`stripplot(x)` plot of the values of `x` on a line (an alternative to `boxplot()` for small sample sizes)
`coplot(x~y | z)` bivariate plot of `x` and `y` for each value or interval of values of `z`
`interaction.plot (f1, f2, y)` if `f1` and `f2` are factors, plots the means of `y` (on the *y*-axis) with respect to the values of `f1` (on the *x*-axis) and of `f2` (different curves); the option `fun` allows to choose the summary statistic of `y` (by default `fun=mean`)
`matplot(x,y)` bivariate plot of the first column of `x` *vs.* the first one of `y`, the second one of `x` *vs.* the second one of `y`, etc.
`fourfoldplot(x)` visualizes, with quarters of circles, the association between two dichotomous variables for different populations (`x` must be an array with `dim=c(2, 2, k)`, or a matrix with `dim=c(2, 2)` if $k = 1$)
`assocplot(x)` Cohen–Friendly graph showing the deviations from independence of rows and columns in a two dimensional contingency table
`mosaicplot(x)` 'mosaic' graph of the residuals from a log-linear regression of a contingency table
`pairs(x)` if `x` is a matrix or a data frame, draws all possible bivariate plots between the columns of `x`
`plot.ts(x)` if `x` is an object of class `"ts"`, plot of `x` with respect to time, `x` may be multivariate but the series must have the same frequency and dates
`ts.plot(x)` id. but if `x` is multivariate the series may have different dates and must have the same frequency
`qqnorm(x)` quantiles of `x` with respect to the values expected under a normal law
`qqplot(x, y)` quantiles of `y` with respect to the quantiles of `x`
`contour(x, y, z)` contour plot (data are interpolated to draw the curves), `x` and `y` must be vectors and `z` must be a matrix so that `dim(z)=c(length(x), length(y))` (`x` and `y` may be omitted)
`filled.contour(x, y, z)` id. but the areas between the contours are coloured, and a legend of the colours is drawn as well
`image(x, y, z)` id. but with colours (actual data are plotted)
`persp(x, y, z)` id. but in perspective (actual data are plotted)
`stars(x)` if `x` is a matrix or a data frame, draws a graph with segments or a star where each row of `x` is represented by a star and the columns are the lengths of the segments
`symbols(x, y, ...)` draws, at the coordinates given by `x` and `y`, symbols (circles, squares, rectangles, stars, thermometres or "boxplots") which sizes, colours ... are specified by supplementary arguments
`termplot(mod.obj)` plot of the (partial) effects of a regression model (`mod.obj`)

The following parameters are common to many plotting functions:
`add=FALSE` if `TRUE` superposes the plot on the previous one (if it exists)
`axes=TRUE` if `FALSE` does not draw the axes and the box
`type="p"` specifies the type of plot, `"p"`: points, `"l"`: lines, `"b"`: points connected by lines, `"o"`: id. but the lines are over the points, `"h"`: vertical lines, `"s"`: steps, the data are represented by the top of the vertical lines, `"S"`: id. but the data are represented by the bottom of the vertical lines
`xlim=, ylim=` specifies the lower and upper limits of the axes, for example with `xlim=c(1, 10)` or `xlim=range(x)`
`xlab=, ylab=` annotates the axes, must be variables of mode character
`main=` main title, must be a variable of mode character
`sub=` sub-title (written in a smaller font)

Low-level plotting commands

`points(x, y)` adds points (the option `type=` can be used)
`lines(x, y)` id. but with lines
`text(x, y, labels, ...)` adds text given by `labels` at coordinates (`x`,`y`); a typical use is: `plot(x, y, type="n"); text(x, y, names)`
`mtext(text, side=3, line=0, ...)` adds text given by `text` in the margin specified by `side` (see `axis()` below); `line` specifies the line from the plotting area
`segments(x0, y0, x1, y1)` draws lines from points (`x0`,`y0`) to points (`x1`,`y1`)
`arrows(x0, y0, x1, y1, angle= 30, code=2)` id. with arrows at points (`x0`,`y0`) if `code=2`, at points (`x1`,`y1`) if `code=1`, or both if `code=3`; `angle` controls the angle from the shaft of the arrow to the edge of the arrow head
`abline(a,b)` draws a line of slope `b` and intercept `a`
`abline(h=y)` draws a horizontal line at ordinate `y`
`abline(v=x)` draws a vertical line at abcissa `x`
`abline(lm.obj)` draws the regression line given by `lm.obj`
`rect(x1, y1, x2, y2)` draws a rectangle which left, right, bottom, and top limits are `x1`, `x2`, `y1`, and `y2`, respectively
`polygon(x, y)` draws a polygon linking the points with coordinates given by `x` and `y`
`legend(x, y, legend)` adds the legend at the point (`x`,`y`) with the symbols given by `legend`
`title()` adds a title and optionally a sub-title
`axis(side)` adds an axis at the bottom (`side=1`), on the left (`2`), at the top (`3`), or on the right (`4`); `at=vect` (optional) gives the abcissa (or ordinates) where tick-marks are drawn
`box()` draw a box around the current plot
`rug(x)` draws the data `x` on the *x*-axis as small vertical lines
`locator(n, type="n", ...)` returns the coordinates (x, y) after the user has clicked `n` times on the plot with the mouse; also draws symbols (`type="p"`) or lines (`type="l"`) with respect to optional graphic parameters (`...`); by default nothing is drawn (`type="n"`)

Graphical parameters

These can be set globally with **`par(...)`**; many can be passed as parameters to plotting commands.
`adj` controls text justification (`0` left-justified, `0.5` centred, `1` right-justified)
`bg` specifies the colour of the background (ex. : `bg="red"`, `bg="blue"`, ... the list of the 657 available colours is displayed with `colors()`)
`bty` controls the type of box drawn around the plot, allowed values are: `"o"`, `"l"`, `"7"`, `"c"`, `"u"` ou `"]"` (the box looks like the corresponding character); if `bty="n"` the box is not drawn
`cex` a value controlling the size of texts and symbols with respect to the default; the following parameters have the same control for numbers on the axes, `cex.axis`, the axis labels, `cex.lab`, the title, `cex.main`, and the sub-title, `cex.sub`
`col` controls the color of symbols and lines; use color names: `"red"`, `"blue"` see `colors()` or as `"#RRGGBB"`; see `rgb()`, `hsv()`, `gray()`, and `rainbow()`; as for `cex` there are: `col.axis`, `col.lab`, `col.main`, `col.sub`
`font` an integer which controls the style of text (`1`: normal, `2`: italics, `3`: bold, `4`: bold italics); as for `cex` there are: `font.axis`, `font.lab`, `font.main`, `font.sub`
`las` an integer which controls the orientation of the axis labels (`0`: parallel to the axes, `1`: horizontal, `2`: perpendicular to the axes, `3`: vertical)
`lty` controls the type of lines, can be an integer or string (`1`: `"solid"`, `2`: `"dashed"`, `3`: `"dotted"`, `4`: `"dotdash"`, `5`: `"longdash"`, `6`: `"twodash"`, or a string of up to eight characters (between `"0"` and `"9"`) which specifies alternatively the length, in points or pixels, of the drawn elements and the blanks, for example `lty="44"` will have the same effect than `lty=2`
`lwd` a numeric which controls the width of lines, default `1`
`mar` a vector of 4 numeric values which control the space between the axes and the border of the graph of the form `c(bottom, left, top, right)`, the default values are `c(5.1, 4.1, 4.1, 2.1)`
`mfcol` a vector of the form `c(nr,nc)` which partitions the graphic window as a matrix of `nr` lines and `nc` columns, the plots are then drawn in columns

mfrow id. but the plots are drawn by row
pch controls the type of symbol, either an integer between 1 and 25, or any single character within ""

1 ○ 2 △ 3 + 4 × 5 ◇ 6 ▽ 7 ⊠ 8 ✳ 9 ⊕ 10 ⊕ 11 ⊠ 12 ⊞ 13 ⊠ 14 ⊠ 15 ■
16 ● 17 ▲ 18 ◆ 19 ● 20 ● 21 ○ 22 □ 23 ◇ 24 △ 25 ▽ * ⋆ . · X X a a ? ?

ps an integer which controls the size in points of texts and symbols
pty a character which specifies the type of the plotting region, "s": square, "m": maximal
tck a value which specifies the length of tick-marks on the axes as a fraction of the smallest of the width or height of the plot; if tck=1 a grid is drawn
tcl a value which specifies the length of tick-marks on the axes as a fraction of the height of a line of text (by default tcl=-0.5)
xaxs, yaxs style of axis interval calculation; default "r" for an extra space; "i" for no extra space
xaxt if xaxt="n" the *x*-axis is set but not drawn (useful in conjunction with axis(side=1, ...))
yaxt if yaxt="n" the *y*-axis is set but not drawn (useful in conjonction with axis(side=2, ...))

Lattice (Trellis) graphics

xyplot(y~x) bivariate plots (with many functionalities)
barchart(y~x) histogram of the values of y with respect to those of x
dotplot(y~x) Cleveland dot plot (stacked plots line-by-line and column-by-column)
densityplot(~x) density functions plot
histogram(~x) histogram of the frequencies of x
bwplot(y~x) "box-and-whiskers" plot
qqmath(~x) quantiles of x with respect to the values expected under a theoretical distribution
stripplot(y~x) single dimension plot, x must be numeric, y may be a factor
qq(y~x) quantiles to compare two distributions, x must be numeric, y may be numeric, character, or factor but must have two 'levels'
splom(~x) matrix of bivariate plots
parallel(~x) parallel coordinates plot
levelplot(z~x*y|g1*g2) coloured plot of the values of z at the coordinates given by x and y (x, y and z are all of the same length)
wireframe(z~x*y|g1*g2) 3d surface plot
cloud(z~x*y|g1*g2) 3d scatter plot
In the normal Lattice formula, y x|g1*g2 has combinations of optional conditioning variables g1 and g2 plotted on separate panels. Lattice functions take many of the same arguments as base graphics plus also data= the data frame for the formula variables and subset= for subsetting. Use panel= to define a custom panel function (see apropos("panel") and ?llines). Lattice functions return an object of class trellis and have to be print-ed to produce the graph. Use print(xyplot(...)) inside functions where automatic printing doesn't work. Use lattice.theme and lset to change Lattice defaults.

Optimization and model fitting

optim(par, fn, method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN") general-purpose optimization; par is initial values, fn is function to optimize (normally minimize)
nlm(f,p) minimize function f using a Newton-type algorithm with starting values p
lm(formula) fit linear models; formula is typically of the form response termA + termB + ...; use I(x*y) + I(x^2) for terms made of non-linear components
glm(formula,family=) fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution; family is a description of the error distribution and link function to be used in the model; see ?family
nls(formula) nonlinear least-squares estimates of the nonlinear model parameters
approx(x,y=) linearly interpolate given data points; x can be an xy plotting structure
spline(x,y=) cubic spline interpolation
loess(formula) fit a polynomial surface using local fitting
Many of the formula-based modeling functions have several common arguments: data= the data frame for the formula variables, subset= a subset of variables used in the fit, na.action= action for missing values: "na.fail", "na.omit", or a function. The following generics often apply to model fitting functions:
predict(fit,...) predictions from fit based on input data
df.residual(fit) returns the number of residual degrees of freedom
coef(fit) returns the estimated coefficients (sometimes with their standard-errors)
residuals(fit) returns the residuals
deviance(fit) returns the deviance
fitted(fit) returns the fitted values
logLik(fit) computes the logarithm of the likelihood and the number of parameters
AIC(fit) computes the Akaike information criterion or AIC

Statistics

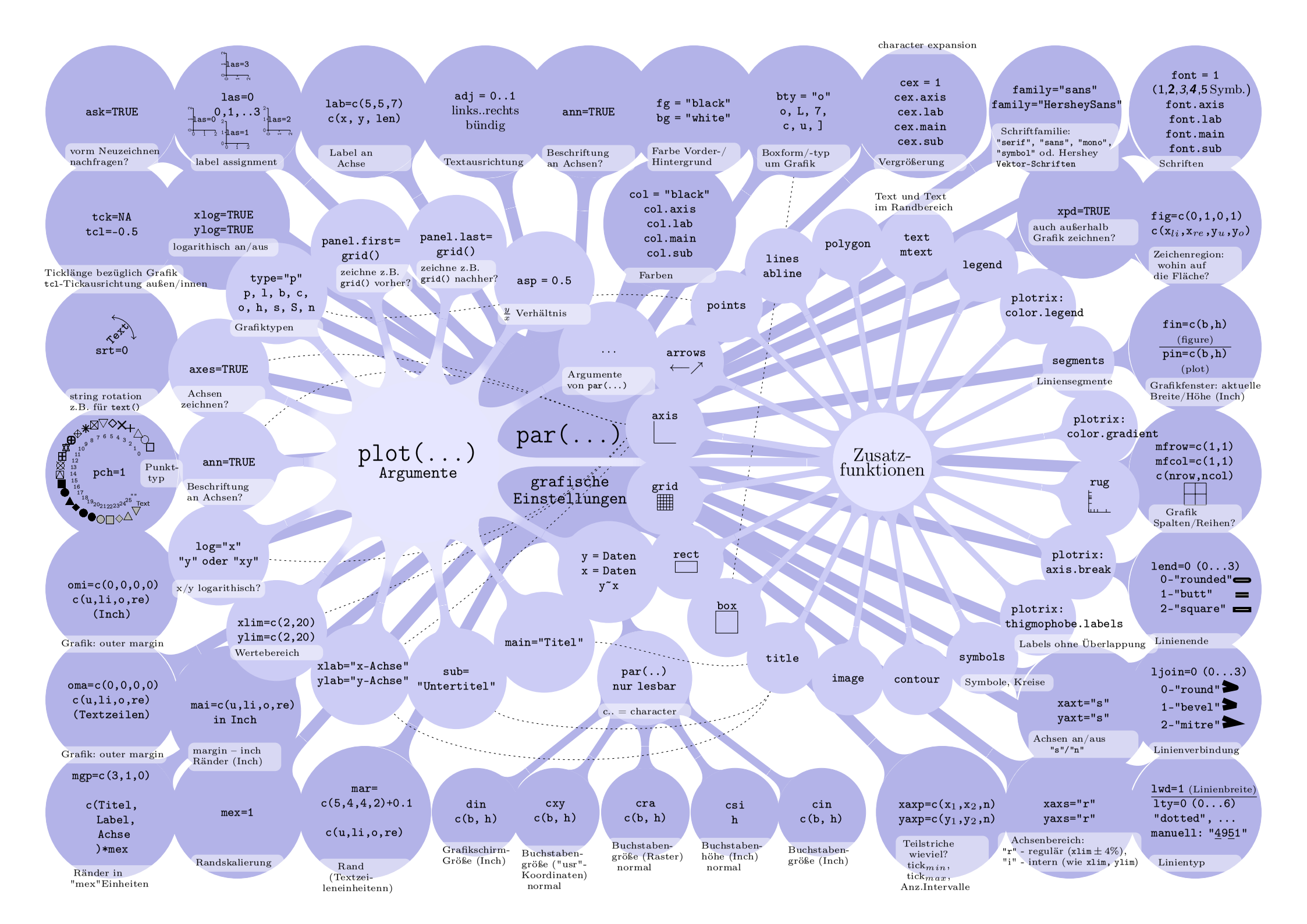
aov(formula) analysis of variance model
anova(fit,...) analysis of variance (or deviance) tables for one or more fitted model objects
density(x) kernel density estimates of x
binom.test(), **pairwise.t.test()**, **power.t.test()**, **prop.test()**, **t.test()**, ... use help.search("test")

Distributions

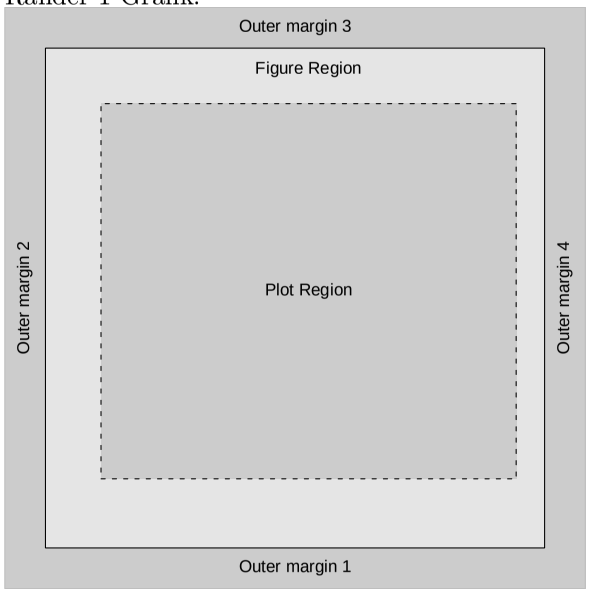
rnorm(n, mean=0, sd=1) Gaussian (normal)
rexp(n, rate=1) exponential
rgamma(n, shape, scale=1) gamma
rpois(n, lambda) Poisson
rweibull(n, shape, scale=1) Weibull
rcauchy(n, location=0, scale=1) Cauchy
rbeta(n, shape1, shape2) beta
rt(n, df) 'Student' (*t*)
rf(n, df1, df2) Fisher–Snedecor (*F*) (χ^2)
rchisq(n, df) Pearson
rbinom(n, size, prob) binomial
rgeom(n, prob) geometric
rhyper(nn, m, n, k) hypergeometric
rlogis(n, location=0, scale=1) logistic
rlnorm(n, meanlog=0, sdlog=1) lognormal
rnbinom(n, size, prob) negative binomial
runif(n, min=0, max=1) uniform
rwilcox(nn, m, n), rsignrank(nn, n) Wilcoxon's statistics
All these functions can be used by replacing the letter r with d, p or q to get, respectively, the probability density (d*func*(x, ...)), the cumulative probability density (p*func*(x, ...)), and the value of quantile (q*func*(p, ...), with $0 < p < 1$).

Programming

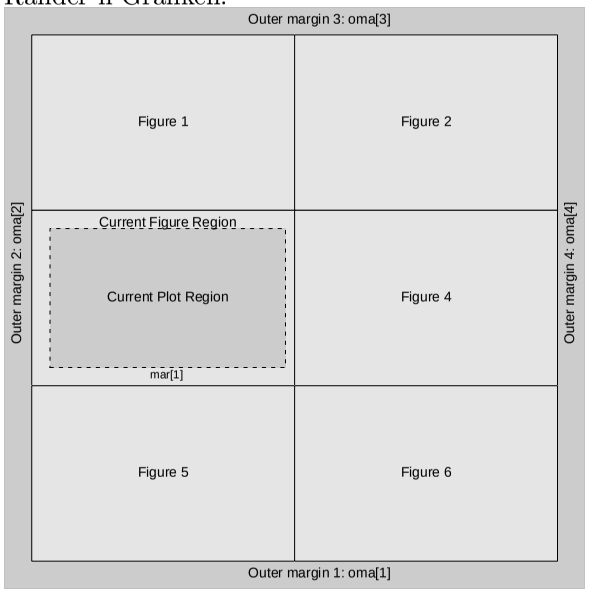
function(arglist) expr function definition
return(value)
if(cond) expr
if(cond) cons.expr else alt.expr
for(var in seq) expr
while(cond) expr
repeat expr
break
next
Use braces {} around statements
ifelse(test, yes, no) a value with the same shape as test filled with elements from either yes or no
do.call(funname, args) executes a function call from the name of the function and a list of arguments to be passed to it
readline(prompt = "") reads a line from the terminal, eg. user input


plot(...)
Argumente
par(...)
grafische Einstellungen
Zusatz-funktionen
ask=TRUE
vorm Neuzeichnen nachfragen?
las=0
0,1,..3
label assignment
lab=c(5,5,7)
c(x, y, len)
Label an Achse
adj = 0..1
links..rechts bündig
Textausrichtung
ann=TRUE
Beschriftung an Achsen?
fg = "black"
bg = "white"
Farbe Vorder-/ Hintergrund
bty = "o"
o, L, 7, c, u,]
Boxform/-typ um Grafik
character expansion
cex = 1
cex.axis
cex.lab
cex.main
cex.sub
Vergrößerung
family="sans"
family="HersheySans"
Schriftfamilie:
"serif", "sans", "mono", "symbol" od. Hershey Vektor-Schriften
font = 1
(1,2,3,4,5 Symb.)
font.axis
font.lab
font.main
font.sub
Schriften
tck=NA
tcl=-0.5
Ticklänge bezüglich Grafik
tcl-Tickausrichtung außen/innen
xlog=TRUE
ylog=TRUE
logarithisch an/aus
panel.first= grid()
zeichne z.B. grid() vorher?
panel.last= grid()
zeichne z.B. grid() nachher?
asp = 0.5
y/x Verhältnis
col = "black"
col.axis
col.lab
col.main
col.sub
Farben
Text und Text im Randbereich
xpd=TRUE
auch außerhalb Grafik zeichnen?
fig=c(0,1,0,1)
c(xli,xre,yu,yo)
Zeichenregion: wohin auf die Fläche?
srt=0
string rotation z.B. für text()
type="p"
p, l, b, c, o, h, s, S, n
Grafiktypen
axes=TRUE
Achsen zeichnen?
...
Argumente von par(...)
arrows
axis
grid
rect
box
points
lines
abline
polygon
text
mtext
legend
plotrix: color.legend
segments
Liniensegmente
fin=c(b,h)
(figure)
pin=c(b,h)
(plot)
Grafikfenster: aktuelle Breite/Höhe (Inch)
plotrix: color.gradient
rug
mfrow=c(1,1)
mfcol=c(1,1)
c(nrow,ncol)
Grafik Spalten/Reihen?
pch=1
Punkt-typ
ann=TRUE
Beschriftung an Achsen?
log="x"
"y" oder "xy"
x/y logarithisch?
y = Daten
x = Daten
y~x
plotrix: axis.break
plotrix: thigmophobe.labels
Labels ohne Überlappung
lend=0 (0...3)
0-"rounded"
1-"butt"
2-"square"
Linienende
omi=c(0,0,0,0)
c(u,li,o,re)
(Inch)
Grafik: outer margin
xlim=c(2,20)
ylim=c(2,20)
Wertebereich
xlab="x-Achse"
ylab="y-Achse"
sub= "Untertitel"
main="Titel"
par(..)
nur lesbar
c.. = character
title
image
contour
symbols
Symbole, Kreise
ljoin=0 (0...3)
0-"round"
1-"bevel"
2-"mitre"
Linienverbindung
oma=c(0,0,0,0)
c(u,li,o,re)
(Textzeilen)
Grafik: outer margin
mai=c(u,li,o,re)
in Inch
margin – inch Ränder (Inch)
xaxt="s"
yaxt="s"
Achsen an/aus "s"/"n"
mgp=c(3,1,0)
c(Titel, Label, Achse)*mex
Ränder in "mex"Einheiten
mex=1
Randskalierung
mar=
c(5,4,4,2)+0.1
c(u,li,o,re)
Rand (Textzei-leneinheitenn)
din
c(b, h)
Grafikschirm-Größe (Inch)
cxy
c(b, h)
Buchstaben-größe ("usr"-Koordinaten) normal
cra
c(b, h)
Buchstaben-größe (Raster) normal
csi
h
Buchstaben-höhe (Inch) normal
cin
c(b, h)
Buchstaben-größe (Inch)
xaxp=c(x1,x2,n)
yaxp=c(y1,y2,n)
Teilstriche wieviel? tick_min, tick_max, Anz.Intervalle
xaxs="r"
yaxs="r"
Achsenbereich:
"r" - regulär (xlim ± 4%),
"i" - intern (wie xlim, ylim)
lwd=1 (Linienbreite)
lty=0 (0...6)
"dotted", ...
manuell: "4951"
Linientyp

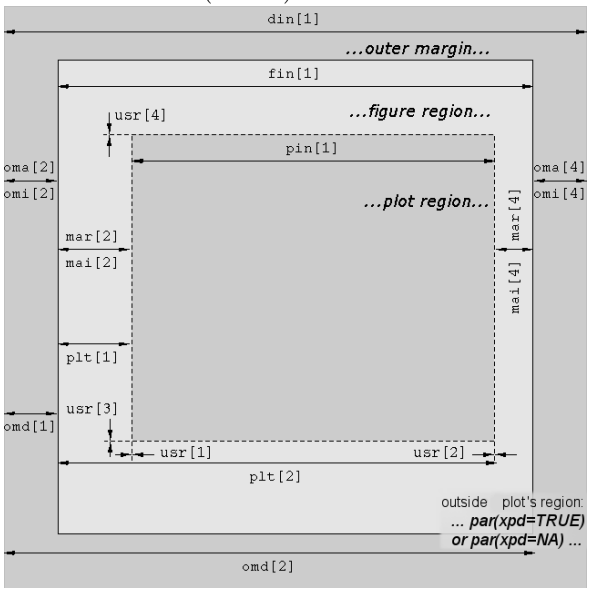
Ränder 1 Grafik:



Ränder n Grafiken:

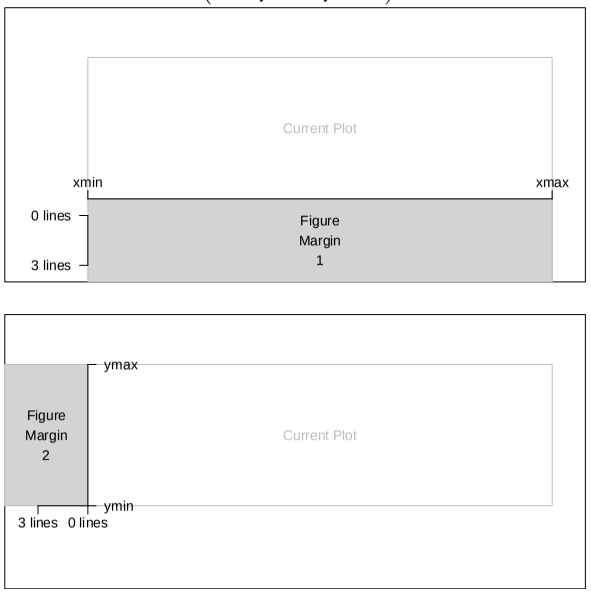


Ränder 1 Grafik (Detail):

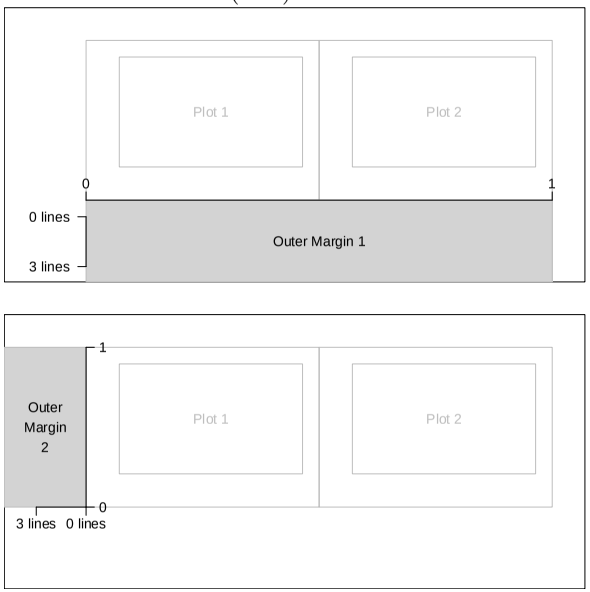


Diese Grafiken sind von Paul Murrell http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html

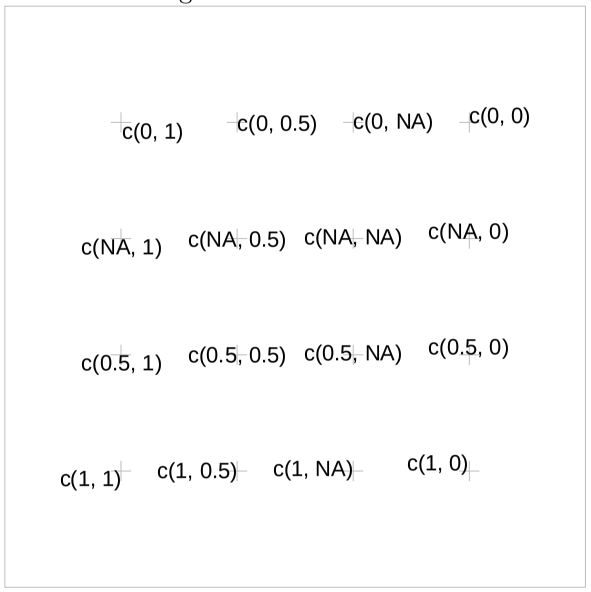
Ränder 1 Grafik (`mar, mai, mex`):



Ränder n Grafiken (`oma`):



Textausrichtung:



Index

Alle Stichwörter mit (G) finden sich im Glossar wieder.

E

F

G

N

O

P

W

Z