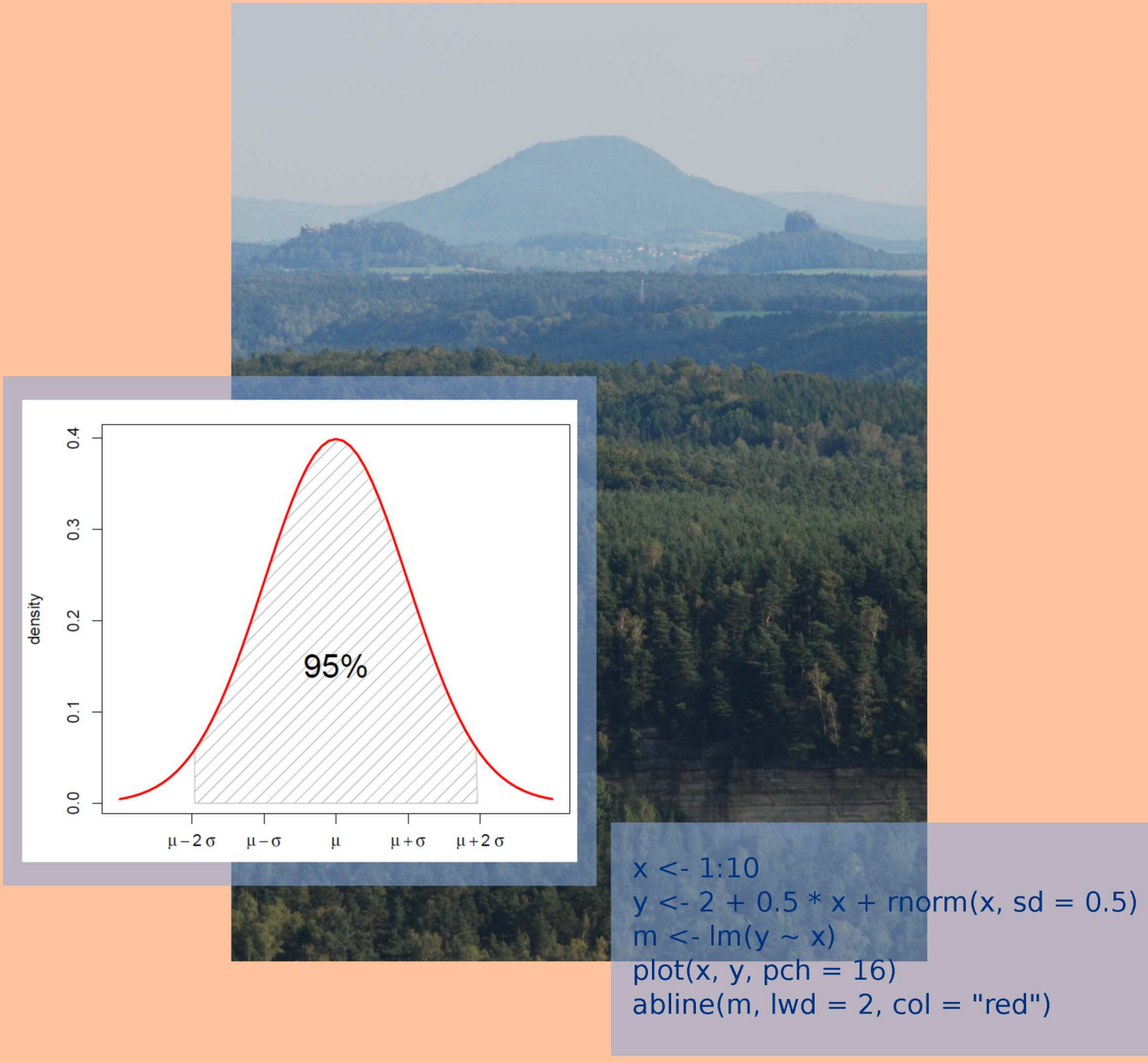
TECHNISCHE UNIVERSITÄT DRESDEN

# Datenanalyse mit R
# Ausgewählte Beispiele




```
x <- 1:10
y <- 2 + 0.5 * x + rnorm(x, sd = 0.5)
m <- lm(y ~ x)
plot(x, y, pch = 16)
abline(m, lwd = 2, col = "red")
```

Thomas Petzoldt

# Datenanalyse mit R
# Ausgewählte Beispiele

Thomas Petzoldt

17. Juli 2019

# Inhaltsverzeichnis

# Vorbemerkung und Dank

Dieses Tutorial befindet sich, genau so wie R in ständiger Weiterentwicklung. Im Regelfall erscheint jedes Jahr mindestens ein Update und es sollte immer die aktuellste Version heruntergeladen werden.

Aus diesem Grund ist es hiermit auch **ausdrücklich verboten** zusätzliche Kopien des Tutorials ins Internet zu stellen. Das Ausdrucken, Kopieren und Weitergeben an gute Freunde (auf USB-Stick) ist jedoch erlaubt, solange dieser Hinweis Bestandteil des Dokumentes bleibt.

Damit das Skript weiterentwickelt werden kann bitte ich sehr herzlich darum, Fehler und Verbesserungsvorschläge zu machen. Ich bemühe mich alle Vorschläge zu berücksichtigen, kleine Korrekturen sofort, größere später.

Ganz besonders bedanke ich mich bei Christof Bigler für die vielen hilfreichen Anmerkungen und Hinweise und empfehle wärmstens, sich das Tutorial von BIGLER and WUNDER (2003) zur Vertiefung von Kapitel 2.2 anzusehen.

# 1 Einleitung

Dieser Kurs setzt Grundbegriffe der angewandten Statistik auf Bachelor-Niveau voraus und versucht diese zu erweitern und zu vertiefen. Hierbei werden einige Dinge wiederholt, größere Wissenslücken müssen im Rahmen des Selbststudiums geschlossen werden. Darüber hinaus liegt ein wichtiger Schwerpunkt in der eigenständigen praktischen Anwendung statistischer Verfahren und dem verantwortungsvollen Umgang damit, d.h. der Vermittlung eines gewissen „Statistik-Gefühls“.

## 1.1 Literaturhinweise

Es existiert eine praktisch unübersehbare Fülle von Lehrbüchern und angewandter statistischer Literatur. Je nach vorhandenen Vorkenntnissen kann eines der folgenden Bücher sinnvoll sein:

- **Als gut verständliche Wiederholungsliteratur:** KÖHLER, W., G. SCHACHTEL und P. VOLESKE, 2007: Biostatistik. Eine Einführung in die Biometrie für Biologen und Agrarwissenschafter. Springer-Verlag, Berlin, 4. Auflage.
- **Als gut lesbares Einsteigerbuch in die Statistik mit R:** DALGAARD, P., 2008: Introductory Statistics with R, Springer-Verlag, New York.
- **Als hervorragend verständliche Einführung, speziell zu ANOVA-Verfahren:** CRAWLEY, M. J., 2012: The R Book. Wiley & Sons, Chichester.
- **Auch für den Ökologen wichtige Verfahren, enthält gut verständliches und praxisnahes Kapitel zur Zeitreihenanalyse:** KLEIBER, C. und ZEILEIS, A., 2008: Applied Econometrics with R. Springer-Verlag.
- **Als inzwischen klassische Referenz, auch für viele Spezialverfahren mit S-PLUS und R:** VENABLES, W. N. and B. D. RIPLEY, 2002: Modern Applied Statistics with S. Springer-Verlag.
- **Als praktische Einführung und Fallstudienbuch für die modernen Verfahren der gemischten Modelle:** ZUUR, A. F. et al. 2009: Mixed Effects Models and Extensions in Ecology with R. Springer-Verlag.
- **Als Nachschlagwerk zu R:** ADLER, J. 2010: R in a Nutshell. A Desktop Quick Reference, O’Reilley.
- **und generell** die zahlreichen Online-Dokumentationen zu statistischen Verfahren allgemein und zu R im besonderen[1].

Die Verfügbare Literatur zu „Statistik mit R“ ist unüberschaubar geworden. Eventuell empfehlenswerte zusätzliche Literatur wird bei den jeweiligen Verfahren genannt. Weitere Literaturhinweise (Lehrbücher und Originalpublikationen) finden sich in der Dokumentation zu R, in zahlreichen Büchern und auf den Online-Hilfe-Seiten der verschiedenen statistischen Verfahren.

[1] z.B. unter `www.r-project.org`

Viele R-Pakete beinhalten Kurzdokumentationen, Tutorials oder aktualisierte Publikationen in form von sogenannten „Vignetten“.

Die mathematische Sprache erlaubt es eigentlich, Zusammenhänge ohne den Bezug zu einer speziellen Software zu formulieren. Anders als vielleicht im Grundkurs müssen wir uns allerdings daran gewöhnen, dass die verschiedenen Publikationen unterschiedliche Konventionen und Schreibweisen verwenden. Um die praktische Anwendung zu erleichtern werden in diesem Skript recht häufig die Formeln weggelassen und stattdessen entsprechende R-Befehle angegeben. Falsch abgeschriebene Formeln und Programme sind ein jedoch großes Risiko und im Rahmen dieses Skriptes wird keinerlei Garantie übernommen. In Zweifelsfällen sind deshalb immer die Originalpublikationen oder Lehrbücher ausschlaggebend und die Verantwortung trägt jeder Wissenschaftler letztlich selbst.

Grundsätzlich lohnt es sich, für die Beschäftigung mit der statistischen Methodik einen gewissen Aufwand zu treiben, ansonsten besteht die Gefahr, die mit hohem Arbeitsaufwand gewonnenen Messergebnisse letztlich zu verschenken.

Der sehr berühmte Statistiker **Ronald Fisher** hat einmal gesagt:

> To consult the statistician after an experiment is finished is often merely to ask him to conduct a post mortem examination. He can perhaps say what the experiment died of.[2]

## 1.2 Aufgaben der Statistik

Statistische Verfahren werden also nicht erst am Ende einer Arbeit angewendet „um die Gutachter zu beruhigen“, sondern es ist wichtig, von vornherein die geplante Untersuchungsmethodik und die statistische Auswertung aufeinander abzustimmen. In diesem Zusammenhang besteht die Aufgabe der Statistik darin:

1. Hypothesen aufzustellen (beschreibende Statistik, Datenverdichtung, Mittelwert, Streuung, explorative Verfahren, Regression),
2. Versuche zu planen (Ermittlung von Effektstärke und zufälligem Fehler in Vorversuchen, Aufstellung eines geeigneten Versuchsdesigns, Ableitung des notwendigen Stichprobenumfanges),
3. Hypothesen zu prüfen (Schließende Statistik: klassische Tests, ANOVA, Korrelation, . . . ).

In diesem Zusammenhang kann man unterscheiden zwischen:

**beschreibender Forschung:** Beobachtung mit dem Ziel „Finden von Zusammenhängen“ (Korrelationen), das Untersuchungsobjekt wird nicht manipuliert.

**experimenteller Forschung:** Feststellen, ob sich ein vermuteter Effekt reproduzieren lässt.

- Manipulation einzelner Randbedingungen,
- Ausschalten von Störungen (Konstanthalten der anderen Randbedingungen),
- Versuchsansatz so einfach wie möglich.

Nur die experimentelle Forschung kann kausale Zusammenhänge schlüssig aufzeigen. Allerdings beginnt gute Wissenschaft immer mit Beobachtung, deshalb darf bobachtende Forschung keinesfalls abschätzig beurteilt werden.

[2] Presidential Address to the First Indian Statistical Congress, 1938. Sankhya 4, 14-17. siehe auch `https://en.wikiquote.org/wiki/Ronald_Fisher`

## 1.3 Strukturierung von Daten

Bei der Erfassung und Eingabe von Daten in den Computer ist es sehr wichtig, von vornherein eine Datenstruktur zu wählen, die nicht nur für eine spezielle Analyse sondern für viele verschiedene potentiell mögliche Auswertungen geeignet ist.

Hierbei ist in den allermeisten Fällen eine datenbankähnliche Struktur (Tab. 1.2) viel geeigneter als eine Kreuztabellenstruktur (Tab. 1.1).

Während Kreuztabellen automatisch aus Datenbanktabellen erzeugt werden können, erfordert der umgekehrte Weg meistens Handarbeit. Darüber hinaus arbeiten die meisten Statistikprogramme ebenfalls vorzugsweise mit Datenbank-Tabellen, z.B. bei der grafischen Darstellung, bei Mittelwertberechnungen oder bei ANOVAs, Tests und multivariater Statistik.

Ob eine Datenstruktur „gut“ (oder besser gesagt, allgemeingültig) ist, lässt sich anhand einiger Regeln relativ leicht feststellen:

- In einer Datenbanktabelle bezeichnen die **Spalten** die beobachteten Kriterien (Variablen), die **Zeilen** (Datensätze) enthalten die einzelnen Beobachtungen.
- Jeder Datensatz wird durch eine Reihe von **Identifikationsmerkmalen** (z.B. Datum, Wassertiefe, Gewässer, Replikat-Nr.) **eindeutig** identifiziert.
- Innerhalb einer Spalte (Variable) haben alle Einträge denselben Datentyp (z.B. Zahl oder Text) und eine einheitliche Maßeinheit.
- Bei Mehrfachbestimmungen gehören die **Replikate untereinander** und nicht nebeneinander.
- Erläuternde Zwischenüberschriften oder Leerzeilen sind nicht nur überflüssig, sondern behindern die (automatische) Auswertung.
- Ist die zeitliche oder räumliche Auflösung der Messwerte sehr unterschiedlich (z.B. für Phosphor in mehrere Tiefen, für Sichttiefe nur einmal am Tag) benutzt man für die entsprechenden Messwerte **separate Tabellen**. Diese lassen sich in Datenbankprogramen über **Relationen** miteinander verknüpfen.

Es lohnt sich durchaus, die Erstellung einer geeigneten Datenstruktur nicht dem Zufall zu überlassen, sondern sorgfältig zu planen. Dies gilt zwar besonders für größere Datenumfänge, lohnt sich aber manchmal auch bei kleineren Datensätzen, z.B. aus Laborexperimenten.

Tabelle 1.1: Beispiel für eine Kreuztabellenstruktur (**So ist es falsch!**)

| | | 0 m | | 3 m | | 8 m | |
|---|---|---|---|---|---|---|---|
| | Datum | Tw | Phyto | Tw | Phyto | Tw | Phyto |
| 1 | 17.01.1996 | 0.6 | 6.375 | 0.9 | 7.342 | 1.2 | 3.243 |
| 2 | 14.02.1996 | −0.1 | 11.655 | 1.6 | 16.446 | 1.9 | 13.161 |
| 3 | 13.03.1996 | 0.5 | 6.532 | 1.7 | 3.554 | 2.4 | 1.314 |
| 4 | 03.04.1996 | 3.2 | 10.015 | 2.8 | 4.652 | 2.3 | 3.58 |
| 5 | 24.04.1996 | 12.3 | | 11.8 | | 6.3 | |
| 6 | 08.05.1996 | 11.2 | 8.506 | 11.2 | 6.24 | 8 | 1.074 |
| 7 | 22.05.1996 | 13.9 | 4.777 | 13.9 | 4.01 | 13.6 | 1.313 |
| 8 | 05.06.1996 | 17.4 | 3.068 | 16.4 | 0.539 | 12.7 | 1.262 |
| 9 | 19.06.1996 | 19.4 | 5.203 | 19.4 | 5.715 | 13.5 | 7.407 |

Tabelle 1.2: Beispiel für eine Datenbank-Tabellenstruktur. (Auch nicht perfekt aber **besser**.)

| | Datum | PNNr | PNOrt | Tw | pH | PSRP | NH4N | NO3N | Phyto |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 17.01.1996 | 1.00 | Spree Zulauf | −0.20 | 7.66 | 0.08 | 1.26 | 5.46 | |
| 2 | 17.01.1996 | 2.00 | Vorsperre | 0.40 | 7.58 | 0.07 | 1.45 | 5.24 | 0.30 |
| 3 | 17.01.1996 | 3.00 | HS 0m | 0.60 | 7.99 | 0.04 | 0.40 | 2.52 | 6.38 |
| 4 | 17.01.1996 | 6.00 | HS 3m | 0.90 | 8.00 | 0.03 | 0.38 | 2.28 | 7.34 |
| 5 | 17.01.1996 | 8.00 | HS 5m | 0.90 | 7.97 | 0.04 | 0.35 | 2.30 | 5.65 |
| 6 | 17.01.1996 | 11.00 | HS 8m | 1.20 | 7.46 | 0.04 | 0.94 | 3.75 | 3.24 |
| 7 | 17.01.1996 | 13.00 | HS 10m | 1.90 | 7.13 | 0.04 | 0.52 | 1.54 | 0.35 |
| 8 | 14.02.1996 | 1.00 | Spree Zulauf | −0.30 | 7.58 | 0.16 | 2.58 | 5.72 | |
| 9 | 14.02.1996 | 2.00 | Vorsperre | −0.30 | 7.41 | 0.14 | 2.72 | 5.50 | |
| 10 | 14.02.1996 | 3.00 | HS 0m | −0.10 | 8.14 | 0.05 | 0.58 | 2.52 | 11.65 |
| 11 | 14.02.1996 | 4.00 | HS 1m | 1.50 | 8.64 | | | | |
| 12 | 14.02.1996 | 6.00 | HS 3m | 1.60 | 8.62 | 0.04 | 0.60 | 2.57 | 16.45 |
| 13 | 14.02.1996 | 8.00 | HS 5m | 1.60 | 8.65 | 0.04 | 0.27 | 2.58 | 13.96 |
| 14 | 14.02.1996 | 11.00 | HS 8m | 1.90 | 7.82 | 0.05 | 0.84 | 1.56 | 13.16 |
| 15 | 14.02.1996 | 13.00 | HS 10m | 3.90 | 7.02 | 0.05 | 0.92 | 1.04 | 7.21 |

# 2 Die Statistikumgebung R

## 2.1 Was ist R

R ist eine von IHAKA and GENTLEMAN (1996) entwickelte und inzwischen von einer erweiterten Gruppe von Wissenschaftlern (R Core Team) und einer sich darum scharenden Gemeinschaft ständig verbesserte und erweiterte Implementierung der vektororientierten Programmiersprache S. Neben grundlegenden Eigenschaften wie Programmierbarkeit und hoher Effizienz enthält R eine Sammlung hochentwickelter Statistik- und Grafikroutinen, die sich mit Hilfe von R auf einheitliche Weise benutzen und kombinieren lassen.

Die Sprache und das System S wurden ursprünglich von den AT&T Bell Laboratories entwickelt und war auch Basis des inzwischen nicht mehr separat erhältlichen kommerziellen Systems S-PLUS[1], zu dem neben dem S-Interpreter und den Statistikbibliotheken auch ein Menüsystem und hervorragende statistische Handbücher gehörten.

R ist Open Source-Software und wird unter der General Public License[2] vertrieben, einem Lizenzmodell das der wissenschaflichen Arbeitsweise entspricht. Alle Bestandteile des Systems sind offen und komplett einsehbar (Sourcecode in C, Fortran und R), die Quellen der Bibliotheken sind sauber zitiert und das System darf nicht nur kostenlos weiterkopiert werden, sondern es ist auch möglich, R selbst zu verändern und zu erweitern. Solche Versionen dürfen auch weitergegeben werden, müssen dann aber ebenfalls komplett offengelegt (publiziert) werden, so dass sich daraus ein Gewinn für alle Benutzer von R ergibt.

Das R-System ist auf allen wichtigen Betriebssystemen lauffähig, z.B. verschiedenen UNIX-Versionen, Linux, Microsoft Windows und Apple MacOS. Die kompletten Quellen, Binärdistributionen, die Dokumentation (z.B. VENABLES *et al.*, 2001; R CORE TEAM, 2014) und weiteres Material kann von der R-Homepage[3] oder dem weltweit verteilten CRAN[4] (Comprehensive R Archive Network) heruntergeladen werden. Darüberhinaus existieren Bücher, die statistische Themen speziell mit Hilfe von von S-PLUS oder R erläutern, z.B. VENABLES and RIPLEY (2002) oder CRAWLEY (2012).

Die Bedienung von R erfolgt in der Regel über Befehle und Funktionen, die entweder einzeln über eine Kommandozeile eingegeben oder als Programm (sogenanntes Script) gestartet werden können. Für den mit den typischen Windows-Programmen erfahrenen Nutzer mag dies auf den ersten Blick altmodisch wirken und einen „Kulturschock“ auslösen. In der praktischen Erfahrung zeigt sich jedoch, dass die Befehlssyntax durchaus effizienter und leistungsfähiger ist, als die Menüsysteme, wie z.B. der R-Commander. Dieser bietet eine Menü-Bedienung für häufig benötigte Verfahren. Es lohnt sich den R-Commander einmal auszubrobieren.

In Bezug auf das Menüsystem von S-PLUS, dem kommerziellen „Bruder“ von R schreibt CRAWLEY (2002):

[1] gehört jetzt zu TIBCO Spotfire `http://http://spotfire.tibco.com/`
[2] Free Software Foundation, `http://www.gnu.org`
[3] R Projekt, `http://www.r-project.org`
[4] CRAN, `http://cran.r-project.org`

> If you enjoy wasting time, you can pull down the menus and click in the dialog boxes to your hear's content. However, this takes about 5 to 10 times as long as writing the command line. Life is short. Use the command line.

Man kann die „R-Befehle“ auf unterschiedliche Weise eingeben: entweder man tippt sie direkt in die R--Kommandozeile ein oder man benutzt einen beliebigen Editor, für Windows z.B. einen in R eingebauten Editor oder den Editor Tinn-R[5].

Das Entwicklungsystem RStudio[6] ist besonders komfortabel. Es ist ebenfalls unter einer freien GNU Public License erhältlich, der AGPL v3[7]. RStudio kommuniziert direkt mit dem R-System, enthält einen Datenimport-Assistenten, erlaubt den direkten Zugriff auf die kontextsensitive Hilfe von R und enthält eine Reihe von Entwicklerwerkzeugen. RStudio gibt es für Linux, Windows und MacOS.

## 2.2 Programmstart und Hilfesystem

Die einfachste Möglichkeit um die Sprache „R“ zu lernen, ist das Verstehen und **kreative Abwandeln** von Beispielen, die Diagnose der dabei auftretenden Probleme und Fehler [8] und das Arbeiten mit R. Ab einem gewissen „Erfahrungslevel“ sollte man dann die offizielle R-Dokumentation (R CORE TEAM, 2014, An Introduction to R) oder z.B. die ersten Kapitel von VENABLES, W. N., and B. D. RIPLEY, 2002: „Modern Applied Statistics with S“ durcharbeiten.

Die ersten Abschnitte dieses „Crashkurses“ geben einen Überblick über einige Grundbausteine von R, mit denen man bereits erste praktische Aufgaben lösen kann.

Wir beginnen unsere erste Session mit dem Start von **RStudio**, einer integrierten Entwicklungsumgebung (IDE), die die Arbeit mit R enorm vereinfacht. RStudio teilt den Bildschirm in 3 (bzw. 4) Fenster auf, sogenannte "*panes*), von denen einige zusätzliche „Tabs“ besitzen um zwischen unterschiedlichen Ansichten zu wechseln.

In einer neuen RStudio-Session, zeigt ein Fenster die Haupt-Hilfeseite von R. Es ist demnach eine gute Idee, sich zunächst im sehr umfangreichen Hilfesystem umzuschauen. Am wichtigsten für das Selbsstudium sind das Einführungshandbuch “An Introduction to R”, die Suchmaschine (“Search Engine & Keywords”), das Verzeichnis der installierten Pakete(“Packages”) und die Antworten auf häufig gestellte Fragen (“FAQ – Frequently Asked Questions”), dazu eventuell das Handbuch “R Data Import/Export”.

Wir beginnen nun, das R-System selbst zu erkunden.

## 2.3 Erste Schritte

Die Eingabe von

```
2 + 4
```

[5] `http://sourceforge.net/projects/tinn-r/`
[6] `www.rstudio.org`
[7] `http://www.gnu.org/licenses/agpl-3.0.html`
[8] Keine Angst: Fehlermeldungen sind normal, es kann dadurch nichts kaputtgehen. Die Ausschüttung von Stresshormonen stimuliert bekanntlich das Gehirn und erhöht die Dauerhaftigkeit des Lerneffektes. Nur wer Fehler überwindet kann sich über den Erfolg richtig freuen.

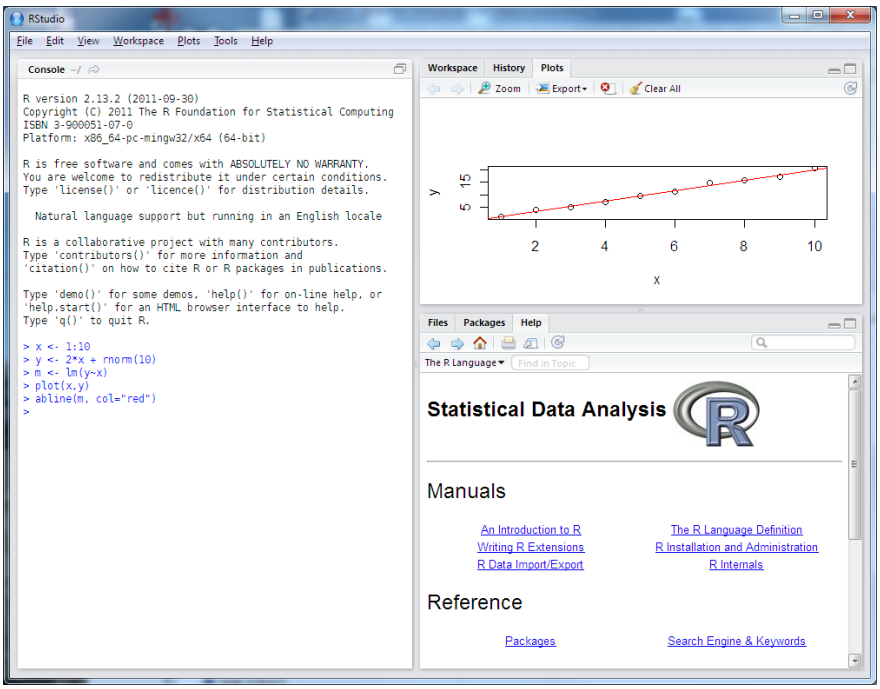



Abbildung 2.1: R Studio mit drei Fenstern. Hier sollte man sich zunächst das Hilfesystem anzuschauen. Mit **File – New File – New R Script** öffnet man das vierte Fenster für eigene Skripte (Sourcecodes).

ergibt als numerisches Rechenergebnis

```
[1] 6
```

Anstelle sich das Ergebnis sofort anzeigen zu lassen, können wir dieses auch in eine Variable abspeichern. Wir benutzen hierzu den Zuweisungsoperator „<−“.

```
a <- 2 + 4
```

Der Inhalt der Variablen `a` kann jederzeit mit

```
a
```

angezeigt werden und ergibt

```
[1] 6
```

Variablennamen beginnen in R immer mit einem Buchstaben, gefolgt von weiteren Buchstaben, Ziffern, Unterstrich oder Punkten. Hierbei wird strikt zwischen Groß- und Kleinschreibung unterschieden, d.h. die Variablen Wert und WERT können unterschiedliche Daten enthalten. Variablen dürfen keine reservierten Bezeichner überschreiben, z.B. `break, for, function, if, in, next, repeat, while` oder „`...`“ (drei Punkte). Andere Bezeichner wie `plot` können prinzipiell überschrieben werden, man sollte das aber möglichst vermeiden.

Im obigen Beispiel steht am Anfang der Zeile eine `[1]`, das bedeutet, dass es sich um das erste Element von `a` handelt. In R können Variablen nämlich grundsätzlich mehrere Werte enthalten (Vektoren, Listen, ...).

Der einfachste Weg der Dateneingabe nutzt den Verkettungsoperator `c` (combine):

```
werte <- c(2, 3, 5, 7, 8.3, 10)
werte
```

```
[1]  2.0  3.0  5.0  7.0  8.3 10.0
```

Sequenzen von Werten lassen sich mit

```
x <- 1:10
x
```

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

oder mit der Funktion `seq` erzeugen.

```
x <- seq(2, 4, 0.25)
x
```

```
[1] 2.00 2.25 2.50 2.75 3.00 3.25 3.50 3.75 4.00
```

Folgen gleicher Werte erhält man mit `rep`

```
x <- rep(2, 4)
x
```

```
[1] 2 2 2 2
```

Auf einzelne Werte kann man durch Indizes zugreifen, wobei der Index wiederum ein Vektor sein kann:

```
werte[5]
```

```
[1] 8.3
```

```
werte[2:4]
```

```
[1] 3 5 7
```

```
werte[c(1, 3, 5)]
```

```
[1] 2.0 5.0 8.3
```

Man kann einzelne oder alle Vektorelemente auch benennen, um mit Hilfe dieser Namen leichter auf diese Elemente zugreifen zu können:

```
benannt <- c(a = 1, b = 2.3, c = 4.5)
benannt
```

```
  a   b   c
1.0 2.3 4.5
```

```
benannt["a"]
```

```
a
1
```

Durch Angabe negativer Indizes lassen sich einzelne oder mehrere Werte eliminieren:

```
werte[-3]
```

```
[1]  2.0  3.0  7.0  8.3 10.0
```

Das Anhängen zusätzlicher Elemente ermöglicht ebenfalls der oben besprochene Verkettungsoperator `c`

```
c(1, 1, werte, 0, 0)
```

```
 [1]  1.0  1.0  2.0  3.0  5.0  7.0  8.3 10.0  0.0  0.0
```

Man kann sich jederzeit die Länge eines Vektors mit

```
length(werte)
```

```
[1] 6
```

anzeigen lassen. Es gibt auch Vektoren, die zwar existieren, aber keine Elemente enthalten:

```
werte <- NULL
werte
```

```
NULL
```

```
length(werte)
```

```
[1] 0
```

Leere Vektoren benötigen wir zuweilen, um einen „Container“ für das spätere Anhängen von Daten zu erzeugen:

```
werte <- NULL
werte
```

```
NULL
```

```
length(werte)
```

```
[1] 0
```

```
werte <- c(werte, 1)
werte
```

```
[1] 1
```

```
werte <- c(werte, 1.34)
werte
```

```
[1] 1.00 1.34
```

Soll eine Datenstruktur komplett gelöscht werden, so verwendet man die `remove` Funktion, z.B.:

```
rm(werte)
werte
```

```
Error: Object "werte" not found
```

Der komplette Arbeitsbereich lässt sich über das Menü von R oder Rstudio oder mit

```
rm(list = ls(all = TRUE))
```

löschen. Das R-System lässt sich über das Menü oder über den Befehl

```
> q()
```

beenden. Sie werden dann gefragt, ob Sie den „R workspace“ speichern und für eine spätere Sitzung erhalten wollen. Für eine Beispielsitzung wie diese ist das wahrscheinlich nicht sinnvoll.

## 2.4 Grafikfunktionen

Wir wollen nun eine Funktion grafisch darstellen, z.B. eine Sinus- und eine Kosinusfunktion im Bereich 0 bis 10. Hierzu erzeugen wir uns zunächst eine Wertetabelle, mit `x` und `y`-Werten. Damit wir später eine glatte Kurve erhalten, wählen wir für den Definitionsbereich ein Intervall von 0.1.

```
x <- seq(0, 10, 0.1)
y <- sin(x)
plot(x, y)
```

Soll anstelle von Plotsymbolen eine Linie gezeichnet werden, verwendet man ein optionales Argument `type="l"`. Achtung: das Symbol für `type` ist ein kleines L für „line“ und keine Eins! **Optionale Argumente werden in R immer mit Namen und Gleichheitszeichen angegeben und können in beliebiger Reihenfolge genannt werden**:

```
plot(x, y, type = "l")
```

Nun soll noch die Kosinusfunktion in einer anderen Farbe hinzugefügt werden. Zum Zufügen von Punkten oder Kurven zu einem existierenden Plot verwendet man `points` oder `lines`

```
y1 <- cos(x)
lines(x, y1, col = "red")
```

Mit Hilfe von `text` kann man beliebige Beschriftungen zu einer Grafik hinzufügen, z.B.

```
x1 <- 1:10
text(x1, sin(x1), x1, col = "green")
```

Soll die Grafik eine eigene Achsenbeschriftung und ein nutzerdefiniertes Koordinatensystem erhalten, so können wir weitere optionale Parameter verwenden.

```
plot(x, y, xlim = c(-10, 10), ylim = c(-2, 2),
    xlab = "x-Werte", ylab = "y-Werte", main = "Beispielgrafik")
```

Man kann einen R-Befehl auf den Folgezeilen fortsetzen, solange er nicht syntaktisch vollständig ist und z.B. die schließende Klammer fehlt. R zeigt eine solche Fortsetzungsmöglichkeit durch ein + an. Umgekehrt lassen sich auch mehrere Befehle auf eine Zeile schreiben, wenn man die Befehle durch Semikolon `;` trennt. Ein Doppelkreuz # bedeutet dagegen, dass es sich beim nachfolgenden Text um einen Kommentar handelt, der von R ignoriert werden soll.

Blättern Sie nun ein wenig in der Online-Hilfe mit `?plot` oder `?plot.default` und **experimentieren Sie mit den Plot-Parametern** `lty`, `pch`, `lwd`, `type`, `log` usw. Es existieren zahlreiche weitere Möglichkeiten, den Plot nach eigenem Willen zu gestalten, z.B. mit eigenen Achsen (`axis`), Legenden (`legend`) oder nutzerdefinierten Linien und Flächen (`abline`, `rect`, `polygon` usw.). Das grundsätzliche Aussehen der Grafiken (Schriftgrößen, Ränder usw.) lässt sich mit `par()` verändern.

Darüberhinaus existiert eine Reihe von „high level"-Grafiken für spezielle Zwecke. Um dies zu demonstrieren, erzeugen wir uns zunächst einmal 100 standardnormalverteilte Zufallszahlen (Mittelwert 0, Standardabweichung 1), plotten diese und zeichnen ein Histogramm. Damit vier Grafiken gleichzeitig dargestellt werden können, benutzen wir den Befehl `par(mfrow=c(2,2))`, das bedeutet zwei Zeilen mit zwei Spalten von Grafiken

```
par(mfrow = c(2, 2))
x <- rnorm(100)
plot(x)
hist(x)
```

Anschließend plotten wir noch einen *normal probability plot* sowie ein Histogramm der relativen Häufigkeiten (`probability=TRUE`) mit eingezeichneter Glockenkurve der Standardnormalverteilung.

```
qqnorm(x)
qqline(x, col="red")
hist(x, probability = TRUE)
xx <- seq(-3, 3, 0.1)
lines(xx, dnorm(xx, 0, 1),col = "red")
```

An dieser Stelle bietet sich ein wenig statistische Rechnung an, z.B. `mean(x)`, `var(x)`, `sd(x)`, `range(x)`, `summary(x)`, `min(x)`, `max(x)`, ...

Wir können auch prüfen, ob die `x`-Werte wirklich normalverteilt sind und benutzen dafür den Shapiro-Wilks-W-Test.

```
x <- rnorm(100)
shapiro.test(x)

        Shapiro-Wilk normality test

data:  x
W = 0.98662, p-value = 0.4126
```

Da es sich bei `x` um Zufallszahlen handelt, können die Werte natürlich auch anders aussehen. Wir wiederholen nun dieses Beispiel mit im Intervall (0,1) gleichverteilten Zufallszahlen:

```
par(mfrow=c(2,2))
y <- runif(100)
plot(y)
```

```
hist(y)
qqnorm(y)
qqline(y, col="red")
mean(y)
var(y)
min(y)
max(y)
hist(y,probability=T)
yy <- seq(min(y),max(y),length=50)
lines(yy,dnorm(yy,mean(y),sd(y)),col="red")
shapiro.test(y)
```

Zum Schluss vergleichen wir noch die beiden Zufallszahlenvektoren mit Hilfe eines Boxplots

```
par(mfrow=c(1, 1))
boxplot(x, y)
```

**Aufgabe:** Wiederholen Sie nun das Beispiel mit neuen Zufallszahlen (oder von Hand eingetippten Werten) und variieren Sie bei der Erzeugung der Zufallszahlen den Stichprobenumfang, den Mittelwert und die Standardabweichung.

Tabelle 2.1: Übersicht über die wichtigsten Datentypen in R

| Objekt | Mögliche Modi | Mixen von Modi möglich | Spalten gleicher Länge | Bemerkung |
|---|---|---|---|---|
| vector | numeric, character, complex, logical | nein | – | |
| factor | numeric, character | nein | – | kategorialer Datentyp, „Behandlung“, „Schlüsselfeld“, Achtung: häufige Fehlerquelle aber wichtig, z.B. für ANOVA. |
| array | numeric, character, complex, logical | nein | ja | ist eine mehrdimensionale Datenstruktur (Zeilen, Spalten, Ebenen, ...) |
| matrix | numeric, character, complex, logical | nein | ja | ist ein array mit nur 2 Dimensionen, kann immer in data.frame gewandelt werden |
| data.frame | numeric, character, complex, logical | ja | ja | entspricht einer Tabelle, kann in eine Liste oder eine Matrix umgewandelt werden |
| list | numeric, character, complex, logical, function, expression, formula | ja | nein | flexibelster Datentyp, kann einer Tabelle (data.frame) oder einer verschachtelten Baumstruktur entsprechen |

## 2.5 Datenstrukturen: Die R-Objekte

Zur Speicherung von Daten gibt es in R neben Vektoren weitere Objekte (Tab. 2.1). Diese Objekte besitzen immer die eingebauten Attribute `mode` (Datentyp) und `length` (Anzahl der Daten im Objekt).

Die einzelnen Objekttypen lassen sich unter gewissen Voraussetzungen ineinander umwandeln, z.B. mit `as.matrix`, `as.data.frame` usw. Bei Faktoren ist Vorsicht geboten, da hier die Inhalte als `levels` abgespeichert sind (andere Statistikpakete machen es übrigens genau so). Neben den hier genannten Objekttypen gibt es noch viele weitere.

## 2.6 Einlesen von Daten

Es existieren mehrere Möglichkeiten, Daten in R einzugeben. Diese sind umfassend und im Detail im Manual „R Data Import/Export“ beschrieben. An dieser Stelle sollen einige Möglichkeiten vorgestellt werden:

1. direkte Eingabe,
2. Einfügen über die Zwischenablage,
3. Einlesen aus einer Textdatei,
4. Datenimport-Assistent von RStudio.

Darüber hinaus existieren weitere Möglichkeiten, so z.B. der direkte Zugriff auf Datenbanken oder der Import von Dateien anderer Statistikprogramme wie SPSS, SAS, Stata oder Minitab (`library(foreign)`) oder sogar von ESRI-Shapefiles (`library(shapefiles)`).

### 2.6.1 Direkte Eingabe

Die direkte Eingabe haben wir bereits oben benutzt, indem wir einen Vektor mit der `c` (combine)-Funktion erzeugt haben:

```
x <- c(1, 2, 5, 7, 3, 4, 5, 8)
```

Auf ähnliche Weise lassen sich auch andere Datentypen eingeben, z.B. ein Dataframe mit:

```
dat <- data.frame(f = c("a", "a", "a", "b", "b", "b"),
                  x = c(1,   4,   3,   3,   5,   7)
       )
```

Ein Dataframe kann aber auch mit Hilfe des R-Tabelleneditors eingegeben werden. Hierzu erzeugen wir zunächst einen leeren Dataframe und geben anschließend die Daten mit Hilfe der `fix`-Funktion ein:

```
dat <- data.frame()
fix(dat)
```

Hierbei können durch Klick auf den Tabellenkopf Variablennamen eingegeben werden. Am Ende der Eingabe muss das Fenster durch Klick auf die rechte obere Ecke (das X) geschlossen werden, damit R wieder in den normalen Kommandozeilenmodus zurückkehrt.

### 2.6.2 Einfügen aus der Zwischenablage

R ist in der Lage, Daten aus der Zwischenablage zu übernehmen, z.B. aus einer Tabellenkalkulation. Hierzu erzeugen wir uns in einer Tabellenkalkulation (z.B. Microsoft Excel oder OpenOffice Calc) eine Tabelle vom Typ „Datenbanktabelle“ (siehe oben). Die folgende Tabelle gibt chemische und morphometrische Größen einiger Seen aus Brandenburg und Mecklenburg an (Quelle: CASPER, 1985):

Wir kopieren nun diesen Datenbereich in der Tabellenkalkulation in die Zwischenablage (Bearbeiten, Kopieren) und können diese in R in eine Variable (z.B. `dat`) übernehmen:

```
dat <- read.table("clipboard", header=TRUE, dec=",")
```

Hierbei benennt `"clipboard"` die Zwischenablage als Datenquelle, `header=TRUE` gibt an, dass die erste Zeile die Variablennamen enthält und `dec=","` sagt, dass die Daten im deutschen Zahlenformat mit Komma als Dezimaltrennzeichen vorliegen.

Die Daten befinden sich nun als Dataframe in der Variablen `dat` und können entsprechend angezeigt oder ausgewertet werden:

```
dat
colMeans(dat[2:8])
boxplot(dat[2:8])
```

Tabelle 2.2: Morphometrische und chemische Größen einiger Seen (S=Stechlinsee, NN=Nehmitzsee Nord, NS=Nehmitzsee Süd, BL=Breiter Luzin, SL = Schmaler Luzin, DA = Dagowsee, HS = Feldberger Haussee; z=mittlere Tiefe (m), t=theoretische Verweilzeit (a), P=Phosphor-Konzentration ($\mu g\,l^{-1}$), N=Stickstoffkonzentration ($mg\,l^{-1}$), Chl=Chlorophyll-Konzentration ($\mu g\,l^{-1}$), PP=Jährliche Primärproduktion ($g\,C\,m^{-2}a^{-1}$), ST = Sichttiefe (m)), Daten: CASPER (1985)

| Nr | Seen | z | t | P | N | Chl | PP | ST |
|---|---|---|---|---|---|---|---|---|
| 1 | S | 23,70 | 40,00 | 2,50 | 0,20 | 0,70 | 95,00 | 8,40 |
| 2 | NN | 5,90 | 10,00 | 2,00 | 0,20 | 1,10 | 140,00 | 7,40 |
| 3 | NS | 7,10 | 10,00 | 2,50 | 0,10 | 0,90 | 145,00 | 6,50 |
| 4 | BL | 25,20 | 17,00 | 50,00 | 0,10 | 6,10 | 210,00 | 3,80 |
| 5 | SL | 7,80 | 2,00 | 30,00 | 0,10 | 4,70 | 200,00 | 3,70 |
| 6 | DA | 5,00 | 4,00 | 100,00 | 0,50 | 14,90 | 250,00 | 1,90 |
| 7 | HS | 6,30 | 4,00 | 1150,00 | 0,75 | 17,50 | 420,00 | 1,60 |

### 2.6.3 Einlesen aus einer Textdatei

Außer aus der Zwischenablage lassen sich mit `read.table` auch Daten aus Textdateien einlesen, z.B. von der Festplatte oder direkt aus dem Internet.

Gegebenenfalls müssen wir hierzu den vollständigen Pfad angeben oder über das Menü in den richtigen Pfad wechseln. Pfade lassen sich auch dem Befehl `setwd` wechseln, hierbei wird bei R auch unter Windows zweckmäßigerweise der normale slash „/“ und nicht der backslash „
“ verwendet:

```
setwd("x:/guest/stat/")
mydata <- read.table("hall.dat", header=TRUE)
```

oder über den Datei-Öffnen-Dialog:

```
mydata <- read.table(file.choose(), header=TRUE)
```

oder aus dem Internet:

```
mydata <- read.table("http://www.simecol.de/data/hall.dat", header=TRUE)
```

Da die Daten bereits im internationalen Format (mit Dezimalpunkt) vorliegen, ist die `dec`-Option nicht erforderlich. Wir schauen uns nun die eingelesenen Daten `mydata` näher an.

```
View(mydata)
```

öffnet (wenn es sich bei `mydata` wie hier um einen Dataframe handelt) die Tabellenansicht, die (im Gegensatz zu `fix`) kein Editieren gestattet und auch nicht unbedingt geschlossen werden muss. Der Datensatz `hall.dat` enthält die aus einer Publikation von HALL (1964) eingescannten Wachstumskurven von Daphnien in Abhängigkeit von Zeit (`day`), Temperatur (`temp`) und Futterkonzentration (in der etwas altertümlichen Maßeinheit „Klett-units“, `klett`).

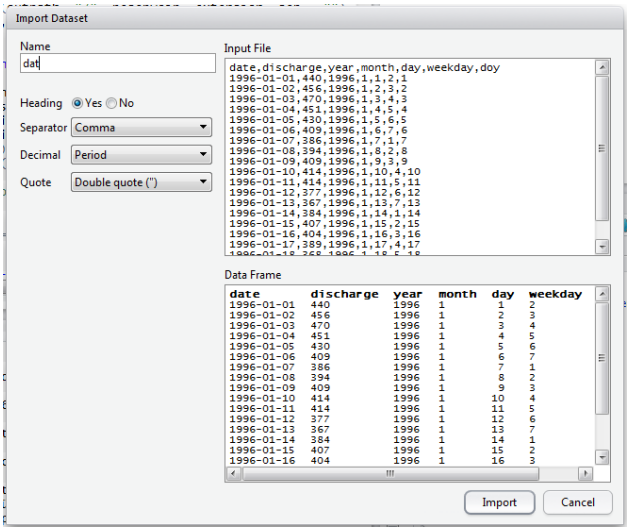



Abbildung 2.2: Datenimport-Assistent von RStudio.

### 2.6.4 Der Datenimport-Assistent von RStudio

RStudio enthält ein speziell für Gelegenheitsnutzer sehr nützliches Werkzeug, den Datenimport-Assistenten ("Import Dataset'). Im Kern dient dieser Assistent dazu, eine korrekte `read.table` oder `read.csv` - Funktion interaktiv zu erstellen. Das obere rechte Feld in Abb. 2.2 zeigt die originale Eingabedatei und das untere Fenster das Ergebnis. Nun stellt man die Optionen für Spaltentrennzeichen (*separator*), Dezimalzeichen (*decimal*) und, falls vorhanden, Anführungszeichen *quote* ein bzw. verändert diese, bis ein korrektes Ergebnis erscheint.

Hinweis: Man sollte nicht vergessen bei **Name** den gewünschten Namen für den Dataframe in R einzustellen (z.B. `dat`), ansonsten verwendet R den Dateinamen dafür.

Für erfahrene R-Nutzer bzw. wenn man einen Datensatz häufiger einlesen muss empfiehlt es sich, `read.table` oder `read.csv` selbst einzutippen oder zu kopieren und nicht jedes Mal den „Umweg" über den Datenimport-Assistenten zu nehmen.

### 2.6.5 Dateien von Tabellenkalkulationen

Es gibt mehrere Pakete in R, mit denen man direkt auf Exceltabellen zugreifen kann. Am einfachsten get dies mit dem Paket **readxl**:

```
library("readxl")
mydata <- read_excel("data/hall.xlsx", sheet=1)
```

Hierbei ist das erste Argument der Dateiname und das 2. Argument die Nummer oder der Name der einzulesenden Tabelle. Im Idealfall beginnt der Datenbereich in Excel oben links in Feld A1 oder zumindest in der A-Spalte ein paar Zeilen tiefer. In diesem Fall können zusätzliche Zeilen mit `skip` übersprungen werden, z.B. Kommentare am Anfang der Tabelle.

Auch Dateien aus Libre- bzw. Openoffice lassen sich so in R lesen, wenn sie als xlsx gespeichert wurden. Für den direkten Zugriff auf Libre/OpenOffice-ODS-Dateien (Open Document Spreadsheet) existiert außerdem ein Paket **readODS**.

### 2.6.6 Direkter Zugriff auf Datenbanken

Für größere Datenmengen empfiehlt sich das Anlegen einer Datenbank. Mit Hilfe verschiedener Datenbankschnittstellen kann aus R lesend und schreibend auf Datenbanken (z.B. SQLite, MySQL, MariaDB, PostgreSQL, Oracle, IBM DB2, Microsoft Access oder Microsoft SQL Server). Auch Excel oder Textdateien können zumindest lesend mit Hilfe der Datenbankschnittstellen und der Standard-Datenbankabfragesprache SQL angesprochen werden, z.B. mit dem Paket **RODBC**. Näheres dazu findet sich im Manual „R Data Import/Export“, das in jeder R-Installation enthalten ist. Besonders einfach ist die Benutzung des Datenbankformats SQLite mit dem paket **RSQlite**.

## 2.7 Arbeiten mit Dataframes

Besonders bei großen Datenstrukturen ist die volle Anzeige des Variableninhaltes nicht immer sinnvoll. Eine deutlich kompaktere Anzeige bietet hier die universelle Strukturansicht:

```
str(mydata)
```

Diese zeigt die Struktur, den Typ und den Umfang der Daten in einer Kurzansicht an und ist auch z.B. für Listen geeignet.

Weitere Diagnosemöglichkeiten sind:

```
names(mydata)
mode(mydata)
length(mydata)
```

und manchmal auch:

```
plot(mydata)
```

Auf die einzelnen Spalten eines Dataframe kann mit Hilfe von Indizes (mit `[]`) wie bei einem Vektor oder einer Matrix) oder mit Hilfe der Spaltennamen (mit `$`) zugegriffen werden, z.B.:

```
mean(mydata[4])
mean(mydata$leng)
plot(mydata$day, mydata$leng)
```

Besonders leistungsfähig ist die Möglichkeit, logische Ausdrücke als Indizes zu verwenden. Voraussetzung hierfür ist, dass alle „Spalten“ die gleiche Länge besitzen. Zur Vermeidung ständiger Wiederholung von `mydata$` empfielhlt es sich, die `plot`-Fuktion mit einer `with()`-Funktion zu umgeben:

```
with(mydata, plot(day[temp == 20], leng[temp == 20]))
unique(mydata$klett)
with(mydata, plot(day[temp == 20 & klett == 16], leng[temp == 20 & klett == 16])
```

Für einen logischen Vergleich muss in R immer ein doppeltes == angegeben werden. Bei logischen Verknüpfungen verwendet man ein `&` für UND und ein `|` für ODER. UND hat immer Vorrang vor ODER, es sei denn, es wird durch runde Klammern anders geregelt.

Benötigt man einen bestimmten Ausschnitt aus einem Dataframe häufiger, so kann man sich mit Hilfe von `subset` eine Teilmenge erzeugen

```
zwanziggrad <- subset(mydata, mydata$temp == 20)
View(zwanziggrad)
```

Auch hier sind wieder logische Verknüpfungen möglich.

Zum Schluss dieses Abschnittes wandeln wir den Hall-Datensatz noch in eine Matrix:

```
mydata   <- read.table("http://www.simecol.de/data/hall.dat", head = TRUE)
mymatrix <- as.matrix(mydata)
```

Das Element aus der 2. Zeile und der 4. Spalte erhält man mit:

```
mymatrix[2, 4]
```

```
[1] 0.5227271
```

die komplette 5. Zeile mit:

```
mymatrix[5, ]
```

```
    klett       temp        day       leng
0.2500000 20.0000000  9.2000000  0.9431816
```

und die Zeilen 5:10 der 4. Spalte (Länge) mit:

```
mymatrix[5:10, 4]
```

```
        5         6         7         8         9        10
0.9431816 0.9602271 1.1250000 1.2215910 1.3068180 1.3920450
```

Weitere Manipulationsmöglichkeiten von Matritzen, Dataframes und Listen sind in der R-Dokumentation erläutert.

#### Bildung von Mittelwerten für eine Faktorkombination

Zum Schluss sollen hier nur zwei Beispiele gezeigt werden, was man in R mit einer einzigen Zeile z.B. machen kann.

Die `aggregate`-Funktion erwartet als erstes Argument die Daten, als zweites Argument eine Liste der Kriterien, nach denen aggregiert werden soll und als 3. Argument eine Funktion, z.B. `mean`, `median`, `sd`, `max` usw.

```
aggregate(mydata, list(klett = mydata$klett, temp = mydata$temp), mean)
```

oder, da es nicht sinnvoll ist, auch Temperatur, Klett-units und Zeit zu mitteln:

```
aggregate(list(leng = mydata$leng),
          list(klett = mydata$klett, temp = mydata$temp), mean)
```

#### Kategoriale Boxplots

Die `split`-Funktion zerlegt einen Vektor in eine Liste von Vektoren mit Hilfe einer Liste von Kategorien:

```
boxplot(split(mydata$leng, list(klett = mydata$klett,
                                temp = mydata$temp)))
```

Noch einfacher geht es mit Hilfe des sogenannten „Formel-Interfaces“:

```
boxplot(leng ~ klett + temp, data = mydata)
```

Das Formel-Interface wird von vielen (aber nicht allen) R-Funktionen unterstützt und bedeutet im obigen Fall: `leng` *versus* `klett` *und* `temp`.

Eine noch bessere Darstellung können wir mit den speziell auf das Formelinterface zugeschnittenen Funktionen des `lattice`-Paketes erreichen. Lattice steht hierbei für „Gitter“ und bewirkt eine besonders übersichtliche und platzsparende Darstellung:

```
library(lattice)
xyplot(leng ~ day|temp * klett, data = mydata)
```

Grundsätzlich lohnt es, sich in die lattice-Grafiken einzuarbeiten. Allerdings geht es (wie immer) auch anders, z.B. indem man die Zerlegung einzeln mit Hilfe des „Datenbankzugriffs“ auf mehrere Zeilen verteilt (oder in eine Schleife einbaut):

```
attach(mydata)
gruppe1 <- leng[klett == 0.25 & temp == 11]
gruppe2 <- leng[klett == 1    & temp == 11]
gruppe3 <- leng[klett == 16   & temp == 11]
# ....
boxplot(gruppe1, gruppe2, gruppe3, names=c("0.25/11", "1/11", "16/11"))
detach(mydata)
```

## 2.8 Ausgabe der Ergebnisse

Die einfachste Möglichkeit besteht darin, die Textausgaben oder Grafiken von R einfach über die Zwischenablage in ein anderes Programm zu kopieren (z.B. OpenOffice, Microsoft Word oder Microsoft Powerpoint). Eine weitere Möglichkeit besteht darin, das gesamte Sitzungsprotokoll über „File - Save to File“ oder „File - Print“ auszugeben. Mit Hilfe von `sink` lassen sich Textausgaben anstelle auf den Bildschirm direkt in eine Protokolldatei umleiten, z.B.:

```
sink("d:/meinprotokoll.txt")
```

anschließend beendet man den Protokollmodus mit Hilfe von:

```
sink()
```

Darüber hinaus existieren weitere Möglichkeiten der Ausgabe von Grafiken und Protokollen, diese sind in der Onlinehilfe unter `print`, `print.table`, `cat`, `pdf`, `postscript`, (png), usw. oder im „R-Data Import/Export Manual“ erläutert. Das Package `xtable` enthält Möglichkeiten, formatierte LaTeX oder HTML-Tabellen zu erzeugen.

## 2.9 Beenden einer R-Sitzung

Wir können das R-Fenster einfach schließen, es über das Menü verlassen oder mit Hilfe von

```
q()
```

beenden. Wenn wir nun auf die Frage „Save workspace image“ mit Ja antworten, werden alle im R-Arbeitsspeicher enthaltenen Daten in eine Datei `.Rdata` im aktuellen Arbeitsverzeichnis gespeichert und stehen beim nächsten Start von R sofort wieder zur Verfügung, vorausgesetzt der Start erfolgt aus dem selben Verzeichnis. Alternativ kann der R-Arbeitsspeicher auch über das File-Menü eingelesen (Load Workspace) oder gespeichert (Save Workspace) werden.

### Übungen

1. Stellen Sie die Hall-Daten grafisch dar. Zeichnen Sie jeweils pro Temperaturlevel eine Grafik und unterscheiden Sie die Nahrungsvarianten durch Farben, Linientypen oder Plotsymbole. Beschriften Sie die Grafiken (Überschrift, Achsen).

2. Lesen Sie die Datei `lakeprofile.dat` ein (Datensatz: Studentenpraktikum vom 13.10.1992 am IGB Berlin). Zeichnen Sie Vertikalprofile für die dort enthaltenen Variablen.

   Hinweis: Maßeinheiten lassen sich mit der `expression`-Funktion auch mit griechischen Buchstaben oder Hoch- und Tiefstellung versehen. Das sieht zwar im Ergebnis druckreif aus, ist etwas gewöhnungsbedürftig, so dass wir uns den Aufwand an dieser Stelle sparen wollen (Näheres dazu siehe Anhang).

3. R enthält eine Reihe von Datensätzen für Testzwecke. Diese können mit der Funktion `data` geladen werden, z.B. `data(iris)`. Stellen Sie sich den Iris-Datensatz mit geeigneten Mitteln grafisch dar.

## 2.10 Bedingte Ausführung und Schleifen

### Gruppierte Ausdrücke

Mehrere Funktionen („Befehle“) können zu einer Gruppe zusammengefasst werden, wenn man sie in geschweifte Klammern einschließt. So ermittelt das folgende Beispiel die Ausführungszeit einer Gruppe von Funktionen mit Hilfe der Funktion `system.time`, die verschiedene Anteile der Rechenzeit des Systems ausgibt. Unter Windows ist diese Funktion etwas eingeschränkt, einige Werte sind hier nicht anwendbar und ergeben `NA`. Im folgenden Beispiel betrug die reine Rechenzeit 0.02s und die insgesamt verbrauchte Systemzeit 0.14s (auf einem Core i7 860 mit 2.9 GHz):

```
system.time({
  x <- rnorm(10000)
  plot(x)
})
```

```
   User      System verstrichen
   0.02        0.13        0.14
```

### Schleifen

Im Unterschied zu anderen Programmiersprachen benötigen wir in R nur selten eine Schleife, da in der Regel ja mit ganzen Vektoren und Matritzen gearbeitet wird.

Die Syntax der `for`-Schleife lautet:

```
for (name in expr1) expr2
```

Hiebei ist `name` die Laufvariable, `expr1\verb` ist im Regelfall ein Vektor und `expr2` meist eine Gruppenanweisung, die mehrfach ausgeführt werden soll. So druckt z.B. die Schleife

```
for (i in 1:10) {
  print(i)
}
```

die Zahlen von 1 bis 10.

Schleifen lassen sich schachteln. Zur besseren Lesbarkeit benutzt man sinnvollerweise Einrückungen.

So soll z.B. das folgende Beispiel 9 Grafiken für alle Kombinationen aus Temperatur und Futter auf den Bildschirm zeichnen:

```
halldata <- read.table("hall.txt", sep = " ", header = TRUE)
par(mfrow = c(3, 3))
for (klett.i in c(0.25, 1, 16)) {
  for (temp.i in c(11, 20, 25)) {
    cat(klett.i, temp.i, "\n")
    dat <- subset(halldata,
                  halldata$klett == klett.i &
                  halldata$temp == temp.i)
    plot(dat$day, dat$leng)
  }
}
```

Es gibt in R weitere Schleifentypen und Statements (`repeat`, `while`, `next`, `break`).

#### Bedingte Ausführung

Die bedingte Ausführung kann man z.B. zur Fallunterscheidung oder zur Ausnahmebehandlung einsetzen. Die Syntax lautet:

`if (`expr1`)` expr2 `else` expr3

Hierbei ist (expr1) ein logischer Ausdruck, der logisch wahr oder falsch sein kann.

Im vorhergehenden Beispiel gab es z.B. keine Daten für die Kombination aus `klett==0.25` und `temp==11`, die entsprechende Grafik führte deshalb zu einer Fehlermeldung. Mit Hilfe einer `if`-Konstruktion lässt sich hier eine Spezialbehandlung einfügen:

```
halldata<-read.table("hall.dat", sep = " ", header = TRUE)
par(mfrow=c(3, 3))
for (klett.i in c(0.25, 1, 16)){
  for (temp.i in c(11, 20, 25)) {
    dat <- subset(halldata,
                  klett == klett.i & temp == temp.i)
    if (nrow(dat) == 0) {                                    # keine Daten
      plot(0, axes = FALSE, type="n", xlab = "", ylab = "") # leerer plot
      box()                                                  # Rechteck
    } else {
      plot(dat$day, dat$leng)
    }
  }
}
```

...alles klar? Falls nicht, überspringen Sie das Beispiel vorerst und schauen es sich zu einem späteren Zeitpunkt nochmals an.

#### Vektorisierte ifelse-Konstruktion

In vielen Fällen kann man anstelle der `if`-Verzweigung die vektorisierte `ifelse`-Funktion benutzen. Im folgenden Beispiel werden z.B. alle Nullwerte mit einem kleinen Wert ($10^{-6}$) ersetzt. Hierzu wird im Falle `x == 0` der Wert `1e-6`, ansonsten (`else`) der ursprüngliche Wert von `x` verwendet und wieder in `x` abgespeichert:

```
x <- c(0, 0, 1, 3, 4, 2, 1, 7, 0, 2)
x <- ifelse(x == 0, 1e-6, x)
x
```

```
[1] 1e-06 1e-06 1e+00 3e+00 4e+00 2e+00 1e+00 7e+00 1e-06 2e+00
```

## 2.11 Definition eigener Funktionen

R lässt sich durch eigene Funktionen beliebig erweitern. Solche Funktionen schreibt man, wenn bestimmte Dinge häufiger benötigt werden oder wenn man komplizierte Dinge vereinfachen möchte. Die Syntax lautet

name <- function(arg1, arg2, arg3, ...) expression

Hierbei ist „name“ der name der Funktion (ein beliebiger Variablenname), „arg1, arg2, ...“ sind die Argumente der funktion und „expression“ sind die auszuführenden Befehle. Im Regelfall gibt eine Funktion einen Wert (oder mehrere Werte, z.B. als Liste) zurück. Das folgende Beispiel aus der „Introduction“ implementiert einen Zweistichproben t-Test.

```
twosam <- function(y1, y2) {
  n1 <- length(y1)
  n2 <- length(y2)
  yb1 <- mean(y1)
  yb2 <- mean(y2)
  s1 <- var(y1)
  s2 <- var(y2)
  s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
  tst <- (yb1 - yb2)/sqrt(s * (1/n1 + 1/n2))
  tst
}
```

Wir testen nun dieses Beispiel und vergleichen das Ergebnis mit der eingebauten Variante des t-Tests, die zusätzlich noch weitere Angaben, insbesondere zur Signifikanz liefert.

```
  x <- c(2,3,4,5,8)
  y <- c(1,3,5,9,9)
  twosam(x, y)
```

```
[1] -0.5255883
```

```
  t.test(x, y, var.equal=TRUE) # eingebauter t-Test
```

```
        Two Sample t-test

data:  x and y
t = -0.52559, df = 8, p-value = 0.6134
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -5.387472  3.387472
sample estimates:
mean of x mean of y
      4.4       5.4
```

Wir beachten in diesem Beispiel, dass die Variablennamen beim Funktionsaufruf (hier x und y) nicht mit den Namen in der Parameterzeile der Funktionsdefinition (hier y1 und y2) übereinstimmen müssen. **Es zählt lediglich die Reihenfolge und der passende Datentyp.** Das ist ganz wichtig und ermöglicht es, allgemeingültige Funktionen (Unterprogramme) zu schreiben. Eine weitere wichtige Eigenschaft ist die Lokalität der in der funktion benutzten Variablen. Diese sind nur innerhalb der Funktion gültig. So ergibt im Beispiel außerhalb der Funktion

```
yb1
```

```
Error: Object "yb1" not found
```

da diese lokale Variable nur innerhalb der Funktion gilt.

Wollen wir den Wert innerhalb einer Funktion wissen (z.B. zur Fehlersuche), so können wir diese entweder mit `print` ausdrucken lassen oder für die betreffende Funktion den Debugmodus einschalten `debug(twosam)`. Dieser Modus wird durch `undebug(twosam)` wieder ausgeschaltet.

Weitere Möglichkeiten , z.B. die Verwendung benannter optionaler Argumente mit Default-Werten, die Benutzung des `...`-Arguments (drei Punkte) globale Zuweisungen mit `<<-` oder das Einbinden von C- oder FORTRAN-Routinen sind der offiziellen R-Dokumentation zu entnehmen.

1. Implementieren und testen Sie die Ricker-Funktion auf Seite 69 von R for Beginners von E. Paradis[9].
2. Entwickeln (oder finden) Sie eine Funktion `circle`, die Kreise auf den Bildschirm zeichnet. Hinweis: Hierzu werden die Winkelfunktionen Sinus und Cosinus benötigt.

[9] `https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf`

# 3 Grundbegriffe

Bevor wir mit den eigentlichen statistischen Verfahren beginnen, sollen zunächst einige Begriffe kurz definiert und Grundprinzipien genannt werden.

## 3.1 Das Sparsamkeitsprinzip

Das Sparsamkeitsprinzip (engl. *principle of parsimony* oder *Occam's razor*) besagt:

> Wenn für ein Phänomen mehrere gleich gute Erklärungen existieren, dann ist die einfachste Erklärung die richtige Erklärung.

In Bezug auf statistische Modelle bedeutet das (CRAWLEY, 2002):

- Modelle sollen so wenige Parameter wie möglich besitzen,
- lineare Modelle sind gegenüber nichtlinearen Modellen zu bevorzugen,
- Experimente mit wenigen Annahmen sind komplizierten Experimenten vorzuziehen,
- Modelle sollten bis auf ein Minimum vereinfacht werden,
- einfache Erklärungen sind besser als komplizierte Erklärungen.

Das Sparsamkeitsprinzip ist eines der wichtigsten Grundprinzipien der statistischen Modellierung und aller Naturwissenschaften. Als anschauliche Einführung, was daraus folgen kann, lohnt es sich durchaus, sich einmal den Film „Contact" anzusehen.

## 3.2 Typen von Variablen

Variablen sind alles das, was gemessen oder das was bewusst vorgegeben oder eingestellt wird, z.B. Phosphat-Konzentration, Temperatur oder Abundanz. Parameter sind Werte, die mit Hilfe eines Modells berechnet werden, z.B. Mittelwert, Standardabweichung, Steigung einer Geraden.

Unabhängige Variablen (*explanation oder explanatory variables*) werden von uns vorgegeben oder resultieren aus nicht beeinflussbaren Randbedingungen. Abhängige Variablen (*response oder explained variables*) sind die Variablen, deren Reaktion wir untersuchen wollen.

Es existieren unterschiedliche Skalentypen:

**Binäre Variablen** oder Boolesche Variablen besitzen genau zwei Zustände: wahr oder falsch, Null oder Eins, vorhanden oder fehlend.

**Nominale Variablen** besitzen eine Bezeichnung, z.B. „rot", „gelb", „grün" aber keine Reihenfolge.

**Ordinale Variablen** besitzen eine Reihenfolge, z.B. die natürlichen Zahlen $(1, 2, 3, \ldots)$ oder die Trophiestufen der Gewässerklassifikation (oligotroph, mesotroph, eutroph, polytroph, hypertroph, nicht jedoch dystroph).

**Metrische Variablen** lassen sich kontinuierlich (stufenlos) messen. Hierbei unterscheidet man weiter die:

**Intervallskala:** Hierbei können sinnvolle Differenzen, nicht jedoch Quotienten gebildet werden. So ist es z.B. bei einer Temperatur von 20°C um 10K wärmer als bei 10°C aber nicht „doppelt so warm“.

**Verhältnisskala:** Bei einer Verhältnisskala existiert ein absoluter Nullpunkt und es sind auch Quotienten sinnvoll. So ist ein Baum mit 2 m doppelt so hoch wie mit 1 m.

Die „Wertigkeit“ der Variablen steigt von der binären Skala zur Verhältnisskala an. Es ist grundsätzlich immer möglich, eine „höherwertige“ Skala in eine „niederwertige“ umzuwandeln.

## 3.3 Wahrscheinlichkeitsbegriff

In der klassischen Definition ist die Wahrscheinlichkeit $p$ einfach der Quotient aus der Zahl der **günstigen Fälle** geteilt durch Zahl der **möglichen Fälle**. Bei einem Würfel beträgt z.B. die Wahrscheinlichkeit, eine bestimmte Zahl (z.B. eine 6) zu würfeln 1/6.

Da bei nicht abzählbaren Grundgesamtheiten jedoch das Problem auftreten kann, dass Zähler und/oder Nenner unendlich groß sind, benutzt man eine axiomatische Definition der Wahrscheinlichkeit:

**Axiom I:** $0 \leq p \leq 1$

**Axiom II:** für unmögliche Ereignisse gilt $p = 0$, für sichere Ereignisse gilt $p = 1$

**Axiom III:** für sich ausschließende Ereignisse $A$ und $B$, d.h. $A \bigcap B$ gilt: $p(A \bigcup B) = p(A) + p(B)$

## 3.4 Grundgesamtheit und Stichprobe

Die Objekte, von denen uns Messungen oder Beobachtungen vorliegen, bilden eine **Stichprobe**. Die **Grundgesamtheit** ist die Menge der Objekte, die die gleiche Chance hatten, in die Stichprobe zu gelangen. Die Grundgesamtheit wird also durch die Art definiert, wie die Stichprobe genommen wird (Repräsentativität). Bezüglich der Gewinnung der Stichprobe unterscheiden wir:

**Zufallsstichprobe (Random sampling):** Hierbei werden zufällige Objekte aus der gesamten Grundgesamtheit ausgewählt (Beispiele: Zufällige Auswahl der Probenahmestellen mittels Gitternetz; Lose, in welches Aquarium welcher Versuch angesetzt wird, zufällige Aufstellung von Kulturgefäßen).

**Geschichtete Stichprobe (Stratified sampling):** Hierbei wird die Grundgesamtheit in verschiedene Klassen oder Schichten (Strata) aufgeteilt. Diese werden getrennt beprobt und anschließend wird über ein gewichtetes Mittel aus den Strata auf die Grundgesamtheit hochgerechnet. Die geschichtete Probenahme erfordert Informationen über die Größe der einzelnen Strata. Beispiele: Wahlprognose, Tiefenschichten, Pelagial und Litoralstation, Mageninhalt von je 10 Fischen pro Größenklasse.

Zufällige Fehler lassen sich mit Hilfe der Statistik und ausreichend großer Stichproben schätzen und eliminieren (herausmitteln), systematische Fehler nicht. Die „wahren“ aber unbekannten Parameter der Grundgesamtheit werden mit griechischen Buchstaben bezeichnet ($\mu$, $\sigma$, $\gamma$, $\alpha$, $\beta$), die berechneten Parameter ($\bar{x}$,

$s$, $r^2 \ldots$) sind nur Schätzwerte. Ein einzelner Messwert $x_i$ einer Zufallsvariablen $X$ ergibt sich aus dem Erwartungswert $\mathbf{E}(X)$ der Zufallsvariablen und einem individuellen zufälligen Fehler $\varepsilon_i$. Der Erwartungswert des Fehlers ist Null:

$$x_i = \mathbf{E}(X) + \varepsilon_i \tag{3.1}$$

$$\mathbf{E}(\varepsilon) = 0 \tag{3.2}$$

## 3.5 Was ist ein statistischer Test?

Statistische Tests dienen zur Prüfung statistischer Hypothesen: aufgrund statistischer Modelle präzisierte Annahmen über teilweise unbekannte Verteilungen beobachteter Merkmale in der Grundgesamtheit, insbesondere über unbekannte **Parameter** dieser Verteilungen.

### Effektstärke und Signifikanz

Bei der Betrachtung statistischer Tests spielen die Begriffe **Effektstärke** und **Signifikanz** eine zentrale Rolle. Hierbei kennzeichnet die Effektstärke $\Delta$, wie groß ein beobachteter Effekt ist, z.B. den Unterschied zweier Mittelwerte $(\bar{x}_1 - \bar{x}_2)$ oder die Größe eines Korrelationskoeffizienten $(r^2)$. Oftmals ist die relative Effektstärke bedeutsam, d.h. das Verhältnis zwischen Effekt und zufälligem Fehler also das Verhältnis aus Signal und Rauschen, z.B. zwischen dem Unterschied zweier Mittelwerte und der Standardabweichung:

$$\delta = \frac{\bar{\mu}_1 - \bar{\mu}_2}{\sigma} = \frac{\Delta}{\sigma}$$

Der Begriff Signifikanz gibt an, dass ein beobachteter Effekt mit einer gewissen Wahrscheinlichkeit tatsächlich existiert und nicht allein durch zufällige Schwankungen erklärt werden kann.

Um zu prüfen, ob ein Effekt signifikant ist, stellt man statistische Hypothesen auf. Diese betreffen das Verhalten beobachtbarer Zufallsvariablen mit fester Wahrscheinlichkeitsverteilung.

$H_0$: Nullhypothese, dass 2 Grundgesamtheiten hinsichtlich eines Parameters oder einer bestimmten Eigenschaft übereinstimmen. Hierbei gilt die Annahme, dass der beobachtete Effekt rein zufälliger Art ist und die wirkliche Differenz Null ist.

Statistische Tests können nur Unterschiede zwischen den verglichenen Stichproben feststellen, d.h. $H_0$ wird i.d.R. aufgestellt, um verworfen zu werden. Es ist prinzipiell nicht möglich nachzuweisen, dass zwei Grundgesamtheiten gleich sind. Die Annahme der Nullhypothese bedeutet lediglich, dass der beobachtete Effekt auch durch zufällige Faktoren erklärt werden kann, d.h. es liegt entweder kein Effekt vor oder die Effektstärke ist zu klein, um sie beim vorhandenen Stichprobenumfang als signifikant anzuerkennen.
**Nicht signifikant bedeutet also: kein Effekt oder nicht genug gemessen!**

$H_a$: Die Alternativhypothese (experimentelle Hypothese) behauptet, dass ein bestimmter Effekt vorliegt. Eine Alternativhypothese ist niemals absolut sicher sondern grundsätzlich vorläufiger Art. Die Annahme der $H_a$ bedeutet nicht, dass ein Effekt nachgewiesen ist, sondern nur, dass die Nullhypothese unwahrscheinlich ist.

Tabelle 3.1: Fehler erster und zweiter Art

| | Wirklichkeit | |
|---|---|---|
| Entscheidung des Tests | $H_0$ wahr | $H_0$ falsch |
| $H_0$ beibehalten | richtig, $1-\alpha$ | Fehler 2. Art, $\beta$ |
| $H_0$ abgelehnt | Fehler 1. Art, $\alpha$ | Richtige Entscheidung mit Power $1-\beta$ |

In der Praxis gibt es immer Unterschiede und statistische Tests dienen dazu, einen Grenzwert festzulegen, ab wann der Unterschied nach menschlichem Ermessen groß genug sein soll um die $H_0$ abzulehnen. Um zu definieren, wie groß dieser Grenzwert sein darf, benutzt man die Irrtumswahrscheinlichkeit $\alpha$, die besagt, mit welcher Wahrscheinlichkeit ein als signifikant anerkannter Effekt irrtümlicherweise doch nur zufällig ist. Daraus folgt, dass es auch ein zufällig signifikantes Ergebnis geben kann.

Beim Prüfen einer $H_0$ sind 2 Fehlentscheidungen möglich (Tab 3.1):

1. $H_0$ unberechtigt abgelehnt (Fehler 1. Art $\alpha$)
2. $H_0$ unberechtigt beibehalten (Fehler 2. Art $\beta$

Die Irrtumswahrscheinlichkeit $\alpha$ wird vor dem Versuch festgelegt und beträgt in der Ökologie üblicherweise $\alpha = 0.05$. Die Irrtumswahrscheinlichkeit für die $H_a$ ($\beta$) ist meist unbestimmt, d.h. man gibt sich nur ein $\alpha$ vor und $\beta$ ergibt sich in Abhängigkeit von $\alpha$, $n$ und der Wirksamkeit (Power) des betreffenden Tests. Liegen Angaben zur Effektstärke und zum zufälligen Fehler aus Vorversuchen vor, kann man den notwendigen Stichprobenumfang $n$ durch eine sogenannte Poweranalyse abschätzen.

# 4 Statistische Maßzahlen

Grundsätzlich kann man Statistik auf zweierlei Art betreiben:

1. indem man die gemessenen Daten direkt gegenüber stellt (nichtparametrische oder verteilungsfreie Verfahren) oder
2. indem man Maßzahlen (Parameter) ermittelt, die die wesentlichen Eigenschaften der Verteilung einer Stichprobe charakterisieren.

Wenn statistische Parameter aus einer Stichprobe „berechnet“ werden, spricht man von einer Parameterschätzung und bezeichnet die geschätzten Werte mit lateinischen, die „wahren“ Werte der Grundgesamtheit dagegen mit griechischen Buchstaben. Oft existieren verschiedene **Schätzer** um einen Parameter der Grundgesamtheit anzunähern und man unterscheidet diese nach:

- Erwartungstreue (die Schätzung konvergiert bei steigendem Stichprobenumfang $n$ gegen den wahren Wert),
- Effizienz (man erreicht bereits bei einem kleinen $n$ eine gute Schätzung),
- Robustheit (die Schätzung wird wenig von Ausreißern oder von der Verletzung von Grundannahmen über die Verteilung beeinflusst).

Je nachdem, welche statistischen Eigenschaften von Stichproben man beschreiben will existieren unterschiedliche Parameter, z.B. Lageparameter (z.B. Mittelwert, Median), Streuungsparameter (z.B. Standardabweichung, Spannweite) oder Zusammenhangsparameter (Korrelationskoeffizienten).

## 4.1 Lageparameter

Das arithmetische Mittel einer Stichprobe ist die Summe aller Werte, geteilt durch den Stichprobenumfang:

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

das geometrische Mittel die Wurzel aus dem Produkt:

$$G = \sqrt[n]{\prod_{i=1}^{n} x_i}$$

wobei es meist günstiger ist, die logarithmische Form zu benutzen[1]:

[1] Ansonsten können sehr große Zahlen und rechentechnische Probleme entstehen.

$$G = \exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \ln x_i\right)$$

Das harmonisches Mittel ist der Kehrwert des Mittels der Kehrwerte:

$$\frac{1}{H} = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{x_i} \quad ; x_i > 0$$

In R berechnet man diese Mittelwerte mit Hilfe der `mean`-Funktion:

```
  x <- c(42, 43, 45, 51, 52, 55, 58, 61, 65, 67)
  mean(x)            # arithmetisches Mittel
```

```
[1] 53.9
```

```
  exp(mean(log(x))) # geometrisches Mittel
```

```
[1] 53.23059
```

```
  1/mean(1/x)        # harmonisches Mittel
```

```
[1] 52.56164
```

Ein Problem der genannten Lagemaßzahlen ist, dass diese stark durch Extremwerte beeinflusst werden können. Man verwendet deshalb oft gern den Median (oder Zentralwert), den nach der Reihenfolge mittleren Wert einer sortierten Stichprobe (bei $n$ ungerade) oder dem Mittelwert der beiden in der Reihenfolge mittleren Werte (bei $n$ gerade):

- $n$ ungerade:
$$Z = x_{(n+1)/2}$$
- $n$ gerade:
$$Z = \frac{x_{n/2} + x_{n/2+1}}{2}$$

Der getrimmte Mittelwert (*trimmed mean*) bildet eine Zwischenstufe zwischen arithmetischem Mittel und Median. Er wird berechnet, indem man einfach eine gewisse Anzahl von Extremwerten weglässt, z.B. 10% der höchsten und 10% der kleinsten Werte:

```
  median(x)            # Median
```

```
[1] 53.5
```

```
  mean(x, trim=0.1)    # getrimmter Mittelwert
```

```
[1] 53.75
```

```
  mean(x, trim=0.5)    # mit Median identisch
```

```
[1] 53.5
```

In vielen Fällen, insbesondere bei schiefen Verteilungen, ist es besser, den Median oder einen getrimmten Mittelwert anstelle des arithmetischen Mittels anzugeben. Der arithmetische Mittelwert sollte nur bei symmetrischen Verteilungen verwendet werden oder wenn die übliche Konvention dies explizit erfordert.

Der Modalwert (Dichtemittel $D$), wird meistens aus Klassenhäufigkeiten abgeleitet. Im einfachsten Fall wird der Klassenmittelwert der Klasse mit der höchsten Klassenhäufigkeit angegeben. Oft bildet man auch einen gewichteten Mittelwert aus der häufigsten Klasse und ihren unmittelbaren Nachbarn:

$$D = x_{uk} + \frac{f_k - f_{k-1}}{2f_k - f_{k-1} - f_{k+1}} \cdot b$$

Hierbei ist $f$ die Klassenhäufigkeit, $b$ die Klassenbreite, $k$ der Index der häufigsten Klasse und $x_{uk}$ deren untere Klassengrenze. Eine weitere Möglichkeit besteht darin, eine sogenannte Kernel-Dichteschätzung (*kernel density estimate*, Abb. 4.1) vorzunehmen und das Maximum als Modalwert anzugeben:

```
hist(x, probability = TRUE)
dens <- density(x)
lines(dens)
dens$x[dens$y == max(dens$y)]
```

```
[1] 54.22913
```

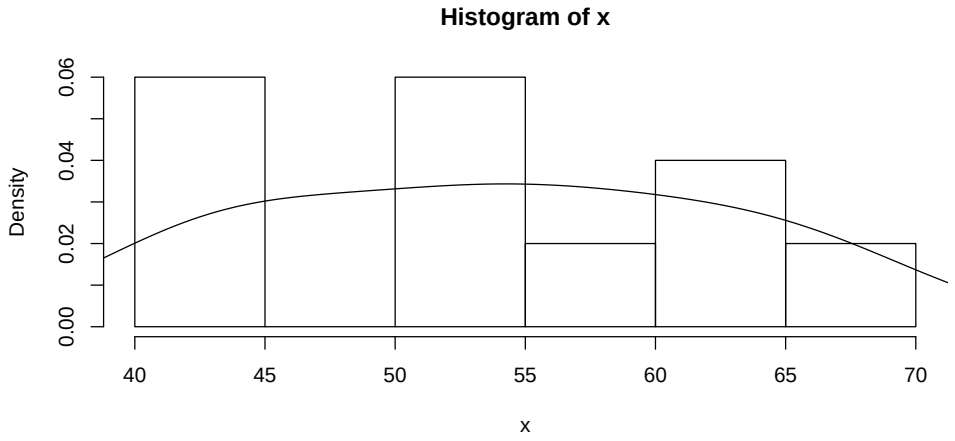



Abbildung 4.1: Histogramm und Dichteschätzung.

Eine Stichprobe kann durchaus mehrere Modalwerte besitzen, z.B. die Körpergröße von Fischen beim Vorliegen mehrerer Kohorten (Altersklassen). Das folgende Beispiel demonstriert eine bimodale Verteilung (Abb. 4.2):

```
library(simecol)  # enthält die peaks-Funktion
# Aus 2 Normalverteilungen zusammengesetzte bimodale Verteilung:
x <- c(rnorm(50, mean=10), rnorm(20, mean=14))
hist(x, prob=T)
dens <- density(x)
lines(dens)
peaks(dens, mode="max")$x # gibt die Modalwerte aus
```

```
[1]  9.997127 14.141254
```

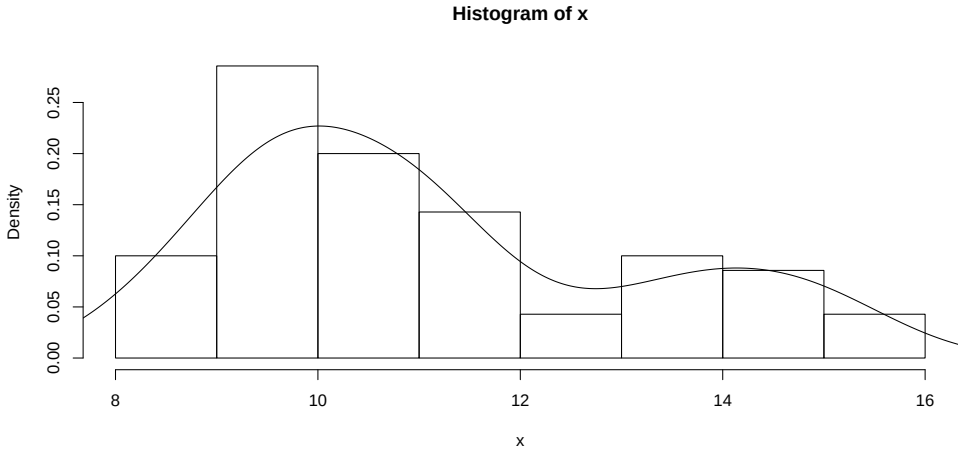



Abbildung 4.2: Histogramm und Dichteschätzung für eine bimodale Verteilung.

Tabelle 4.1: Hinweise zur Anwendung der versch. Maßzahlen, aus KÖHLER *et al.* (2002)

| | Lage | Streuung |
|---|---|---|
| Normalverteilung oder symmetrische Verteilung und mindestens Intervallskala | $\bar{x}$ | $s, s_{\bar{x}}$ |
| *cv* nur bei Verhältnisskala zulässig | | *cv* |
| eingipflig, schief, mindestens Ordinalskala | $Z, D$ | $V$ |
| $I_{50}$ nur bei $n \geq 12$ | $Q_1, Q_3$ | $I_{50}$ |
| mehrgipflig, mindestens Ordinalskala | $D_1, D_2$ | $V, I_{50}$ |
| $Z$ günstig bei offenen Randklassen | $Z$ | |
| Nominalskala | $D$ | |
| Zeitreihen | $H$ | |
| Verhältniszahlen | $G$ | |

## 4.2 Streuungsparameter

Streuungsmaßzahlen spielen in der statistischen Analyse eine zentrale Rolle. Sie geben an, wie groß die Variabilität in den gemessenen Daten ist und wie genau oder unsicher die Parameter sind, die aus diesen Daten geschätzt werden. Die am häufigsten verwendete Maßzahl ist die Varianz $s^2$:

$$s_x^2 = \frac{SQ}{FG} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

mit $SQ$ der Summe der Abweichungsquadrate vom Mittelwert $\bar{x}$ und $FG$ der Anzahl der Freiheitsgrade ($FG = n-1$).

Berechnet man $s^2$ mit dem Taschenrechner ist es meist zweckmäßiger, eine Gleichung auf Basis von Summen und Quadratsummen zu verwenden:

$$s_x^2 = \frac{\sum (x_i)^2 - (\sum x_i)^2/n}{n-1}$$

Die Wurzel der Varianz $s = \sqrt{s^2}$ bezeichnet man als Standardabweichung. Ihr Vorteil liegt darin, dass $s$ dieselbe Maßeinheit hat, wie $\bar{x}$, so dass man $s$ und $\bar{x}$ direkt miteinander vergleichen kann.

Als Variationskoeffizienten $cv$ bezeichnet man die relative Standardabweichung, bezogen auf den Mittelwert:

$$cv = \frac{s}{\bar{x}}$$

Er ist prinzipbedingt nur bei Daten mit einer Verhältnisskala anwendbar und kann auch in Prozent angegeben werden.

Die Spannweite (oder Variationsbreite, engl. *range*) kennzeichnet die Differenz zwischen Maximum und Minimum der Stichprobe:

$$V = x_{max} - x_{min}$$

allerdings wird diese sehr stark von Extremwerten und Ausreißern beeinflusst. Es ist deshalb günstiger, die jeweils 25% der kleinsten und er größten Werte zu streichen und den Interquartilbereich anzugeben (im allgemeinen erst bei $n \geq 12$ sinnvoll):

$$I_{50} = Q_3 - Q_1 = P_{75} - P_{25}$$

Hierbei sind $Q_3$ und $Q_1$ das erste und das dritte Viertel (Quartil) der aufsteigend sortierten Stichprobe oder das 25. bzw. 75. Perzentil ($P_{25}, P_{75}$).

Ist die zugrundeliegende Stichprobe normalverteilt, ergibt sich ein bestimmtes Verhältnis zwischen Interquartilbereich ($I_{50}$) und Standardabweichung:

$$\sigma = E(I_{50}/(2\Phi^{-1}(3/4))) \approx E(I_{50}/1.394)$$

wobei $\Phi^{-1}$ die Umkehrung der kumulativen Verteilungsfunktion der Normalverteilung (Quantilfunktion) ist. Ein im Vergleich zum Interquartilbereich noch robusteres Streuungsmaß ist die *median absolute deviation*:

$$MAD = median(abs(median - x_i))$$

Die MAD wird meist von vornherein mit dem oben genannten Wert $1/\Phi^{-1}(3/4)$ skaliert und konvergiert so bei normalverteilten Stichproben gegen den Wert der Standardabweichung $E(MAD) = \sigma$.

### Anwendung in R

Alle Streuungsmaße lassen sich in R sehr einfach ermitteln:

```
  x <- rnorm(100, mean=50, sd=10)  # 100 Zufallszahlen
  var(x)                           # Varianz
```

```
[1] 116.5235
```

```
  sd(x)                            # Standardabweichung
```

```
[1] 10.7946
```

```
  range(x)                            # Spannweite

[1] 19.89315 75.01822

  quantile(x, c(0.25, 0.75))          # Quartile

     25%      75%
42.08258 57.22386

  IQR(x)                              # Interquartilbereich

[1] 15.14128

  diff(quantile(x, c(0.25, 0.75))) # Interquartilbereich

     75%
15.14128

  mad(x)                              # Median Absolute Deviation

[1] 11.6922
```

## 4.3 Standardfehler des Mittelwertes

Im Gegensatz zu den bisher besprochenen Streuungsmaßzahlen kennzeichnet der Standardfehler des Mittelwertes nicht die Variabilität der Stichprobe, sondern schätzt die Streuung eines Mittelwertes in Abhängigkeit von der Stichprobenvarianz und dem Stichprobenumfang:

$$s_{\bar{x}} = \frac{s}{\sqrt{n}}$$

Er gibt an, wie genau ein Mittelwert ist und spielt deshalb eine zentrale Rolle bei der Angabe von Vertrauensintervallen oder in vielen statistischen Tests. Bei ausreichend großen Stichproben ($n > 30$) und annähernder Normalverteilung liegt der wahre Mittelwert $\mu$ mit 68%iger Wahrscheinlichkeit im Bereich $\bar{x} \pm s_{\bar{x}}$, bei kleinerem $n$ wird das Vertrauensintervall mit Hilfe der t-Verteilung geschätzt (Details siehe Abschnitt Verteilungen).

Werden in Grafiken Fehlerbalken angegeben, so handelt es sich meistens um den Standardfehler. Dies ist nicht nur optisch günstig (weil der Standardfehler kleiner ist als die Standardabweichung) sondern meistens auch deshalb sinnvoll, weil man in der Regel mehrere Stichproben oder zeitlich aufeinanderfolgende Mehrfachmessungen vergleichen will. Will man jedoch die Streuung der Einzelmessungen angeben, verwendet man die Standardabweichung oder (noch besser) den Interquartilbereich. Wegen dieser Mehrdeutigkeit der Fehlerbalken gehört eine Erläuterung (z.B. Fehlerbalken = $\pm$ Standardfehler) unbedingt in die Abbildungsunterschrift oder in eine Legende.

#### Anwendung in R

Fehlerbalken lassen sich am einfachsten mit Hilfe der Funktion `barplot2` aus dem Zusatzpaket `gplots` darstellen (Abb. 4.3):

```
library(gplots) # enthält die barplot2-Funktion
nx <- 2
ny <- 4
nz <- 10
x  <- rnorm(nx, mean=5, sd=1)
y  <- rnorm(ny, mean=4, sd=1)
z  <- rnorm(nz, mean=8, sd=2)
m  <- c(mean(x), mean(y), mean(z))
sx <- c(sd(x), sd(y), sd(z))/sqrt(c(nx, ny, nz))
barplot2(m, ci.l=m-sx, ci.u=m+sx, plot.ci=TRUE,
  names=c("x", "y", "z"), xlab="mean +/- se")
```

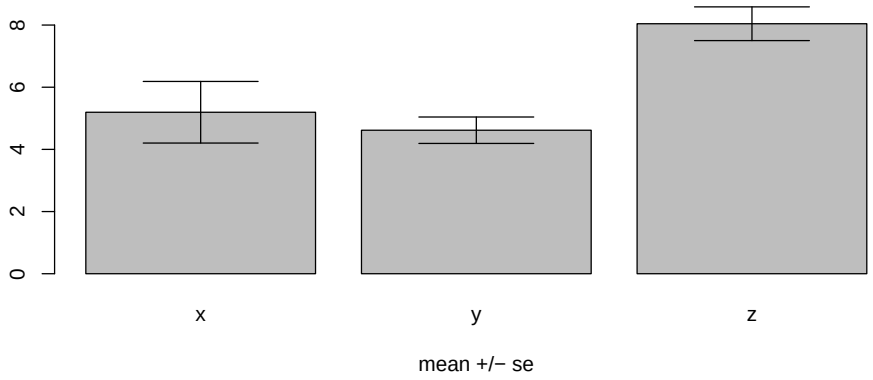



Abbildung 4.3: Der Standardfehler dient zum Vergleich von Mittelwerten

## 4.4 Übungen

1. Erzeugen Sie Zufallsstichproben mit $n = 30$ und ermitteln Sie die unterschiedlichen Lage- bzw. Streuungsparameter mit dem Taschenrechner bzw. mit einer Tabellenkalkulation und mit R.

2. Die Datei **prk_nit.txt** enthält Datensätze des individuellen Biovolumens (in $\mu\,\mathrm{m}^3$) von *Nitzschia acicularis*, einer Kieselalge, die im Rahmen von Studentenpraktika im Jahr 1985 bzw. 1990 bestimmt wurden.

   Laden Sie die Stichprobe `nit90` und drucken Sie ein Histogramm aus. Markieren Sie anschließend von Hand die unterschiedlichen Lage- und Streuungsparameter auf dem Ausdruck.

3. Der Iris-Datensatz ist ein klassischer Datensatz der Statistik. Er wurde bereits 1936 von R.A. Fisher zur Begründung der Diskriminanzanalyse eingesetzt (FISHER, 1936) und dient auch heute noch zur Veranschaulichung von Klassifikationsmethoden.

   Ermitteln Sie die geeignete Lage- und Streungsparameter für „Sepal.Length“ für die drei Arten des des Iris-Datensatzes (`data(iris)`). Wie genau sind diese Mittelwerte bestimmt? Kann man anhand dieser Mittelwerte die Arten sicher unterscheiden?

# 5 Verteilungen

Wahrscheinlichkeitsverteilungen sind das A und O der Statistik und viele Statistik-Kurse beginnen mit Münzen- oder Würfelexperimenten. Wir beginnen mit Zufallszahlengeneratoren. Auf diese Weise bekommen wir:

- ein Gefühl dafür, was Zufall ist und wie z.B. eine „echte Normalverteilung“ aussieht,
- ein Werkzeug zur Versuchsplanung: zum Kennenlernen statistischer Verfahren vor Durchführung des Experimentes und zum Testen der Trennschärfe (Power) dieser Verfahren.

So sollen z.B. gemessene Daten oft auf Normalverteilung geprüft werden. In der Praxis stellt sich aber fast immer heraus, dass mehr oder weniger starke Abweichungen vorliegen und es tritt die Frage auf, ob die Abweichungen relevant sind. Bei den im Folgenden verwendeten Zufallszahlen ist der Verteilungstyp der Grundgesamtheit aufgrund der Erzeugung mit einem bestimmten Typ von Zufallszahlengenerator bekannt.

Zur Darstellung benutzen wir zunächst die Funktion `hist()`, später lernen wir weitere Grafiktypen und Tests kennen.

## 5.1 Gleichverteilung

Gleichverteilte Zufallszahlen sind in einem bestimmten Intervall, z.B $(0,1)$ gleich wahrscheinlich. In R erzeugt man Zufallszahlen mit der Funktion `runif`. Hierbei steht `r` für *random* und `unif` für *uniform*. Das Argument in der Klammer gibt an, wie viele Zufallszahlen erzeugt (generiert) werden sollen, also wie groß der Stichprobenumfang sein soll. Zunächst generieren wir 10 Zufallszahlen und zeigen sie auf dem Bildschirm an:

```
runif(10)
```

```
[1] 0.52841136 0.46483478 0.59073531 0.01543587 0.67345019 0.19768644
[7] 0.28239957 0.67079991 0.41217242 0.08396927
```

und anschließend erzeugen wir noch einmal 400 Zufallszahlen und speichern sie in der Variablen `x`:

```
x <- runif(400)
```

Wir können die Zufallszahlen nun grafisch darstellen (Abb. 5.1):

```
par(mfrow=c(1,2))
plot(x)
hist(x)
```

Zum Vergleich, wie unterschiedlich gleichverteilte Zufallszahlen aussehen, kann man Zufallszahlenerzeugung und Darstellung durch “Verschachtelung” in einer Befehlszeile zusammenfassen:

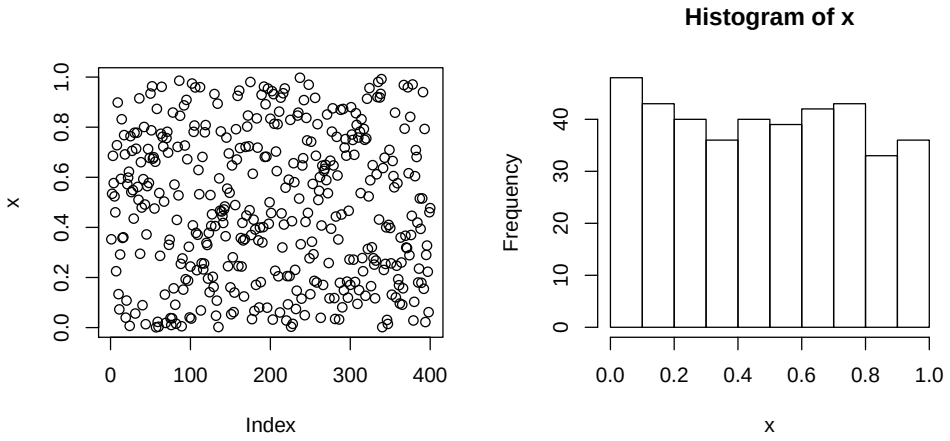



Abbildung 5.1: Gleichverteilte Zufallszahlen, links als Zahlenfolge, rechts als Histogramm.

```
hist(runif(x))
```

Sollen die Zufallszahlen nicht im Intervall $(0,1)$ liegen, sondern in einem anderen Intervall $(5,10)$, so benutzen wir entweder eine Normierung:

```
xmin <-5
xmax <- 10
runif(100)*(xmax-xmin) + xmin
```

oder wir übergeben Maximum und Minimum als zusätzliche Argumente an die `runif`-Funktion:

```
runif(100, xmin, xmax)
```

Das Histogramm (Abb. 5.1) zeigt, wie eine *empirische* Gleichverteilung aussieht, die mit Hilfe von Zufallszahlen gewonnen wurde, doch wie sieht der Idealfall, eine theoretische Gleichverteilung aus?

Hierzu benutzen wir die Funktion `dunif`, wobei das `d` für *density* oder Dichte steht. Zur Veranschaulichung plotten wir die Dichtefunktion über dem Intervall $(-0.2, 1.2)$ mit 100 (oder besser sogar 500) Stützstellen:

```
x <- seq(-0.2, 1.2, length=500)
plot(x, dunif(x), type="l",
                  xlab="Zufallsvariable X",
                  ylab="Wahrscheinlichkeitsdichte")
```

Die Dichtefunktion $f(X)$ wird oft als PDF (*probability density function*) bezeichnet. Die Fläche unter der Kurve ist Eins, d.h. zwischen $-\infty$ und $+\infty$ oder im Beispiel sogar im Intervall $(0,1)$ liegen 100% aller Ereignisse, also:

$$F(X) = \int_{-\infty}^{+\infty} f(X)dX = 1 \tag{5.1}$$

Wollen wir jedoch wissen, welcher Anteil der Ereignisse (hier: wieviele der erzeugten Zufallszahlen) kleiner als ein festgelegter Wert $x$ sind, benutzen wir die Verteilungsfunktion $F$ als bestimmtes Integral:

$$F(x) = \int_{-\infty}^{x} f(X)dX \tag{5.2}$$

Dieses Integral steht in R als Funktion `punif` (Wahrscheinlichkeitsfunktion oder *probability function*) zur Verfügung:

```
x <- seq(-0.2, 1.2, length=500)
plot(x, punif(x), type="l",
                  xlab="Zufallsvariable X",
                  ylab="Verteilungsfunktion")
```

#### Kumulative Häufigkeiten

Im Falle der Gleichverteilung ergibt sich eine im Intervall $(0,1)$ steigende Gerade. Für die empirische Verteilung entspricht dies der kumulativen Häufigkeit, die auch als kumulatives Histogramm dargestellt werden kann. Da die R-Funktion `hist` aber keine kumulative grafische Darstellung kennt, müssen wir hier etwas Handarbeit anlegen. Wir benutzen `hist` lediglich für die Klasseneinteilung, schalten aber die grafische Darstellung aus. Anschließend schauen wir uns das Objekt `h` mittels `str` an, bilden die kumulative Summe von `h$counts` und plotten das Ergebnis mit der allgemeineren Funktion `barplot`:

```
x <- runif(400)
hist(x)

h <- hist(x, plot=FALSE)  # nur Klassen, keine Grafik
hcum <- cumsum(h$counts)
barplot(hcum, names.arg=round(h$mids,2),col="white")
```

Bisher zeigten die Histogramme immer die absoluten Häufigkeiten, oftmals benötigen wir jedoch relative Häufigkeiten. Für die relativen Klassenhäufigkeiten existiert eine Option in `hist` und für die kumulativen Häufigkeiten erreichen wir dies einfach durch eine Division durch die Anzahl der Beobachtungen:

```
hcum <- cumsum(h$counts)/sum(h$counts)
barplot(hcum, names.arg=round(h$mids,2),
              col="white", ylab="Probability")
```

Als letzte wichtige Funktion in diesem Zusammenhang ist noch die Quantil-Funktion `qunif` zu erwähnen. Hierbei handelt es sich um die Inverse von `punif` mit deren Hilfe wir ermitteln können, bis zu welchem Wert ein entsprechender Anteil (Prozentsatz) der Ereignisse zu finden ist.

Beispiel: In welchem symmetrischen Bereich liegen 95% aller Werte bei einer Gleichverteilung $U(40,60)$:

```
qunif(c(0.025, 0.975), min=40, max=60)
```

```
[1] 40.5 59.5
```

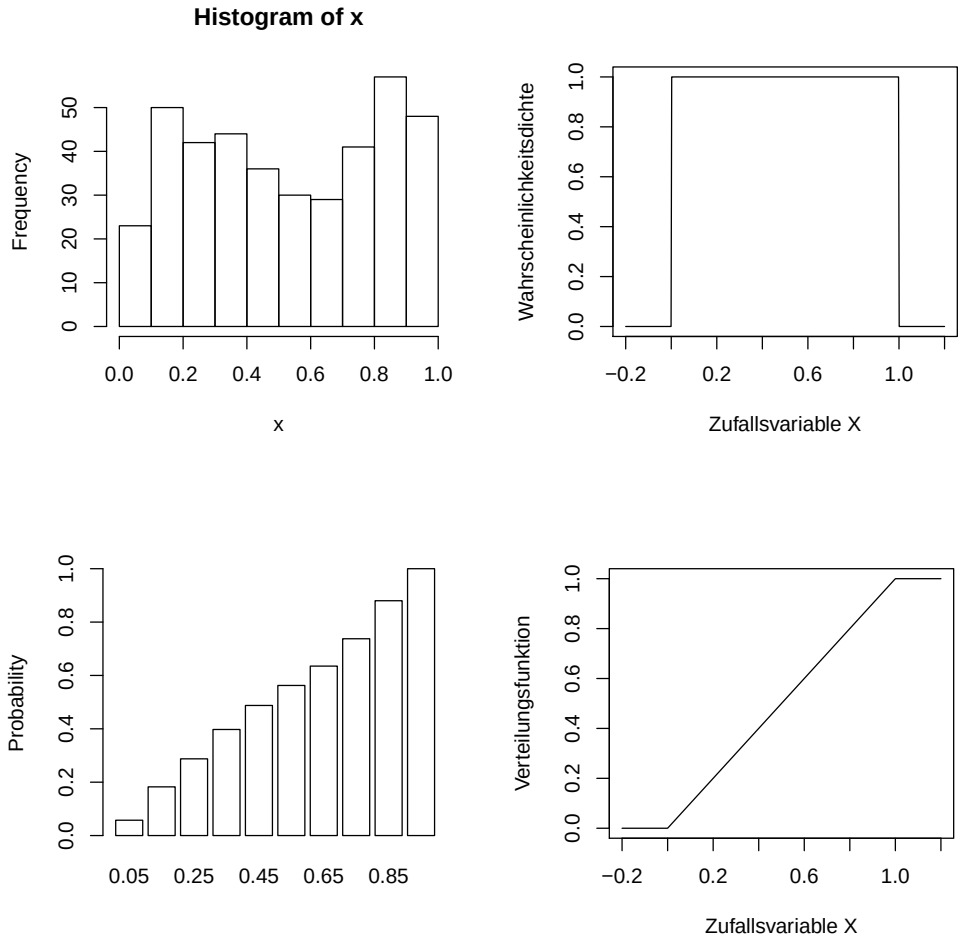



Abbildung 5.2: Gleichverteilung, oben absolute Häufigkeit und Dichtefunktion, unten relative kumulative Häufigkeit und Verteilungsfunktion.

## 5.2 Normalverteilung

Das Buch von CRAWLEY (2002) gibt eine sehr intuitive Einführung in das Verständnis der Normalverteilung. Ausgangspunkt ist die einfache Exponentialfunktion:

$$y = \exp(-|x|^m) \tag{5.3}$$

Zur Veranschaulichung plotten wir diese Funktion im Intervall $(-3,3)$ mit einem Exponenten von $m = 1,2,3,8$ (Abb. 5.3). Bei dieser Gelegenheit trainieren wir auch gleich die Definition einer benutzerdefinierten Funktion, die wir hier `f` nennen:

```
f <- function(x, m) {exp(-abs(x)^m)}  # die Exponentialfunktion
par(mfrow=c(2,2))                     # Platz fuer 4 Grafiken
x <- seq(-3, 3, length=100)           # Definitionsbereich
plot(x, f(x,1), type="l")
plot(x, f(x,2), type="l")
plot(x, f(x,3), type="l")
plot(x, f(x,8), type="l")
```

Die Funktion mit $m = 2$ besitzt auf Grund verschiedener Eigenschaften eine extrem hohe Bedeutung, wir müssen sie lediglich noch so skalieren, dass der Flächeninhalt unter der Kurve Eins wird und erhalten die Standard-Normalverteilung mit Mittelwert $\mu = 0$ und Varianz $\sigma^2 = 1$:

$$f(z) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-z^2/2} \tag{5.4}$$

Ausgehend von der Standardnormalverteilung erhalten wir andere Normalverteilungen mit einem beliebigen Mittelwert $\mu$ und einer beliebigen Varianz $\sigma^2$ durch eine weitere Skalierung:

$$z = \frac{x - \mu}{\sigma} \tag{5.5}$$

Hierbei verschiebt $\mu$ die gesamte Glockenkurve in $x$-Richtung und $\sigma$ führt zu einer Streckung bzw. Stauchung in $y$-Richtung. Man nennt diese Skalierung auch „Standardisierung“ oder $z$-Transformation.

### Implementation in R

In R existieren für die Normalverteilung Funktionen für Zufallszahlen (`rnorm`), die Dichtefunktion (`dnorm`), die Verteilungfunktion (`pnorm`) und die Quantile (`qnorm`).

Wir wollen nun 100 Zufallszahlen aus einer Normalverteinung mit $\mu = 50$ und $\sigma = 10$ erzeugen, stellen diese graphisch dar und ermitteln (schätzen) den Mittelwert $\bar{x}$ und die Standardabweichung $s$ der Stichprobe:

```
x <- rnorm(100, 50, 10)
hist(x, probability = TRUE)
mean(x)
```

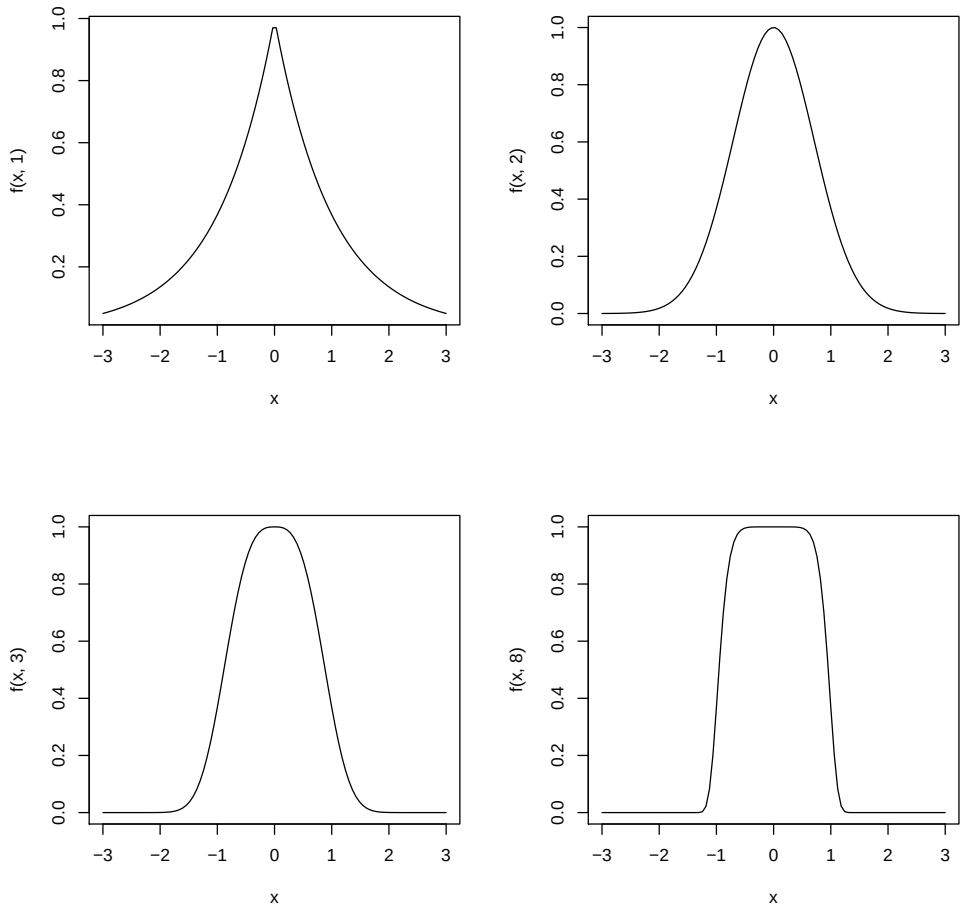



Abbildung 5.3: Verschiedene Exponentialfunktionen. Die Funktion mit dem Exponenten $m = 2$ kommt einer Normalverteilung recht nahe und muss nur noch so skaliert werden dass das Integral eins wird.

```
[1] 49.60832
```

```
 sd(x)
```

```
[1] 9.811248
```

Nun wollen wir eine Glockenkurve in das Diagramm (Abb. 5.4) einzeichnen und benutzen dazu zum einen eine allgemeine Dichteschätzung mit einem sogenannten Kernschätzer und zum anderen eine theoretische Normalverteilung unter Benutzung der Stichprobenparameter $\bar{x}$ und $s$:

```
 lines(density(x), col="blue")
 xnew <- seq(min(x), max(x), length=100)
 lines(xnew, dnorm(xnew, mean(x), sd(x)), col="red")
```

```
[1] 49.60832
```

Zwischen der Kernschätzer-Kurve (näheres siehe VENABLES and RIPLEY, 2002, oder die R-Online-Hilfe) und `dnorm` besteht ein wesentlicher Unterschied: Im ersten Fall machen wir lediglich eine allgemeine Dichteschätzung (also eine Glättung) ohne konkrete Annahmen über die zugrundeliegende Funktion, im zweiten

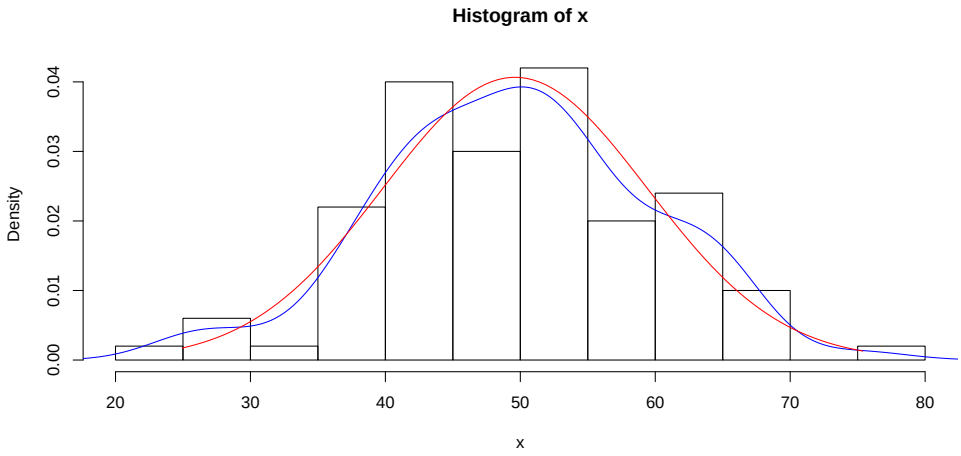



Abbildung 5.4: Normalverteilung: Klassen (Säulen), Dichteschätzung (blaue Linie) und theoretische Dichtefunktion (rote Linie).

Fall wenden wir bereits ein statistisches Modell an, d.h. die Annahme, dass es sich um eine Normalverteilung handelt.

## 5.3 t-Verteilung

Die t-Verteilung ist eine Prüfverteilung, d.h. sie beschreibt, wie sich verschiedene statistische Parameter verteilen. Weitere Prüfverteilungen sind die $\chi^2$-Verteilung und die F-Verteilung.

Im Prinzip ist die t-Verteilung der Normalverteilung sehr ähnlich. Sie besitzt zusätzlich zum Lageparameter $\mu$ und zum Streuungsparameter $\sigma$ einen weiteren Parameter für die Freiheitsgrade $df$ (*degrees of freedom*). Bei einer geringen Anzahl von Freiheitsgraden hat die t-Verteilung sehr breite „tails“, d.h. es besteht eine erhöhte Wahrscheinlichkeit von extremen Werten. Bei $df \to \infty$ oder praktisch bereits bei $df \approx 30$ geht die t-Verteilung in die Normalverteilung über.

### Beispiel

Als Beispiel plotten wir eine Standardnormalverteilung und mehrere t-Verteilungen (Abb. 5.5) mit unterschiedlicher Anzahl an Freiheitsgraden:

```
x <- seq(-3,3, length=100)
plot(x, dnorm(x), type="l", col="red")
lines(x,dt(x, df=1), col="cyan")
lines(x,dt(x, df=2), col="blue")
lines(x,dt(x, df=4), col="cyan")
lines(x,dt(x, df=8), col="blue")
lines(x,dt(x, df=16), col="blue")
lines(x,dt(x, df=32), col="green")
```

In einem zweiten Beispiel sehen wir uns die Abhängigkeit des häufig benötigten t-Wertes $t_{1-\alpha/2}$ mit $\alpha = 0.05$ (95%-Quantil bei zweiseitiger Fragestellung) von der Anzahl der Freiheitsgrade an (Abb. 5.6). Zum Vergleich dient das zweiseitige 5%-Standard-Normal-Quantil (als gestrichelte Linie):

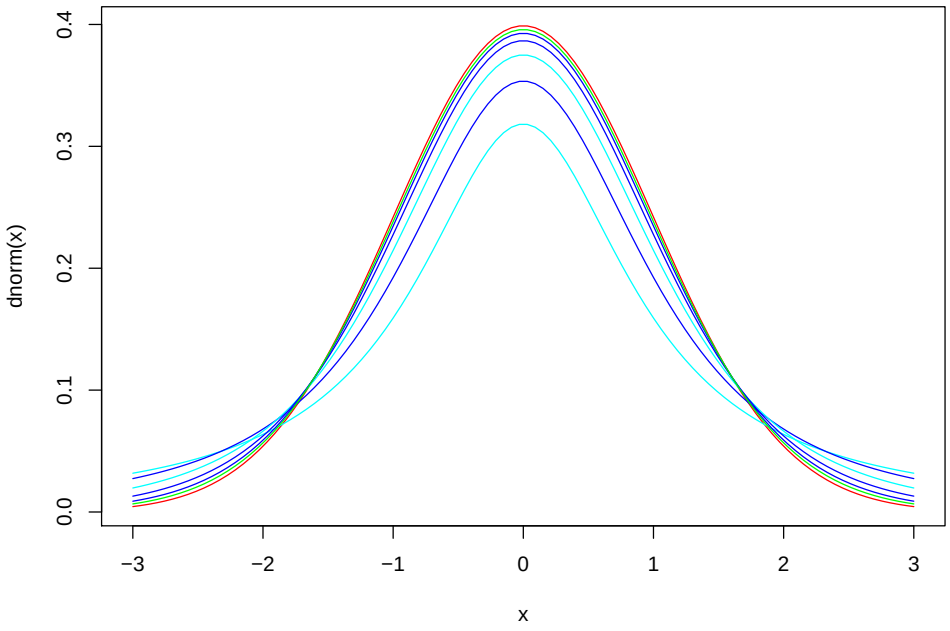



Abbildung 5.5: Dichteverteilung der Normalverteilung (rot) und von t-Verteilungen mit einer unterschiedlichen Anzahl von Freiheitsgraden.

```
plot(1:30, qt(0.975, 1:30), type="l",
   ylab="Students t", xlab="d.f.", ylim=c(0,15))
abline(h=qnorm(0.975), lty="dotted")
```

Die Grafik zeigt, dass wir bei Stichprobengrößen kleiner als 5 mit einer „besonderen Bestrafung“ rechnen müssen, d.h. zusätzlich zur Wirkung des Standardfehlers (s.u.) werden Vertrauensintervalle aufgrund der t-Verteilung sehr breit und statistische Tests verlieren dramatisch an Trennschärfe.

## 5.4 Logarithmische Normalverteilung

Viele in der Natur beobachtbare Prozesse sind nicht normalverteilt, sondern sie sind auf der linken Seite durch Null beschränkt, besitzen aber auf der rechten Seite große Extremwerte z.B. die Wasserführung eines Flusses, Nährstoffkonzentrationen in Gewässern oder die Phytoplanktonbiomasse in einem See. Für viele derartige Prozesse kann mit einigem Erfolg die logarithmische Normalverteilung angewendet werden.

```
x <- rlnorm(1000, meanlog=0, sdlog=0.5)
hist(x, probability=TRUE)
xnew <- seq(min(x), max(x), length=100)
lines(xnew, dlnorm(xnew, meanlog=mean(log(x)),
  sdlog=sd(log(x))), col="red")
```

Die Besonderheit der logarithmischen Normalverteilung ist, dass diese durch den Mittelwert und die Standardabweichung der Logarithmen definiert wird. Man bezeichnet die entsprechenden Stichprobenparameter mit $\bar{x}_L$ bzw. $s_L$ und in R als `meanlog` bzw. `sdlog`.

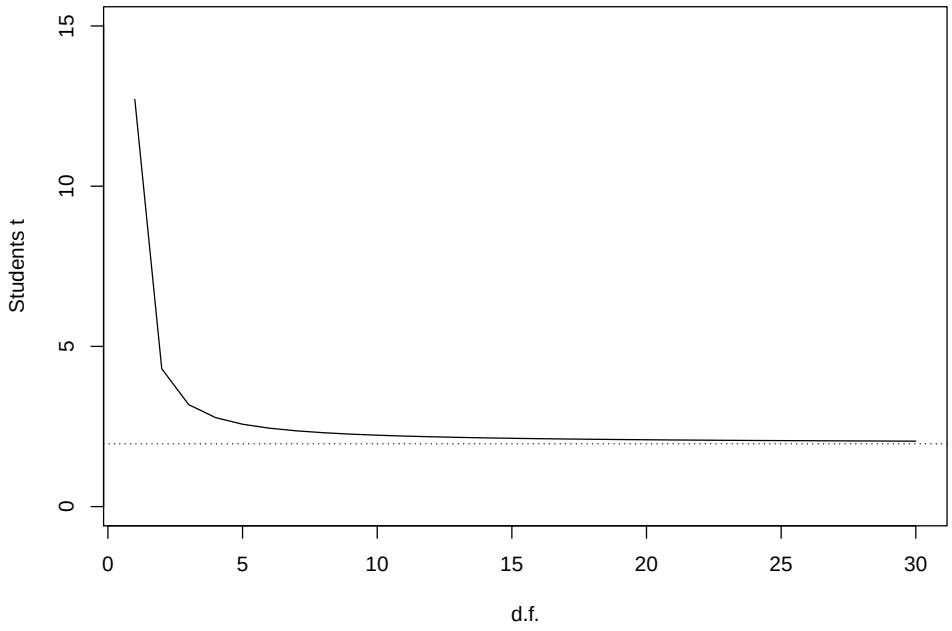



Abbildung 5.6: Abhängigkeit des t-Quantils von der Anzahl der Freiheitsgrade. Man erkennt das der $t$-Wert speziell bei $df < 5$ stark anwächst. Bei $df > 30$ erreicht der $t$-Wert ($t$ =2.04) annähernd das Quantil der Normalverteilung (1.96).

```
x <- rlnorm(1000, meanlog=0, sdlog=0.5)
mean(x); sd(x)
```

```
[1] 1.09834
```

```
[1] 0.5693962
```

```
mean(log(x)); sd(log(x))
```

```
[1] -0.03037786
```

```
[1] 0.5036964
```

Bei den Parametern $\bar{x}_L$ und $s_L$ handelt es sich demnach nicht um Mittelwert und Standardabweichung der Stichprobe selbst, sondern um die Parameter der sogenannten Elternverteilung. Logarithmiert man die aus einer logarithmischen Normalverteilung stammenden Werte, erhält man demnach eine Normalverteilung:

```
hist(log(x), probability=TRUE)
xnew <- seq(log(min(x)), log(max(x)), length=100)
lines(xnew, dnorm(xnew, mean=mean(log(x)), sd=sd(log(x))), col="red")
```

## 5.5 Gamma-Verteilung

Die Gammaverteilung ist ebenfalls eine rechtsschiefe Verteilung, die für viele praktische Probleme sehr nützlich ist, insbesondere im Zusammenhang mit den in letzter Zeit verstärkt angewendeten *Generalized*

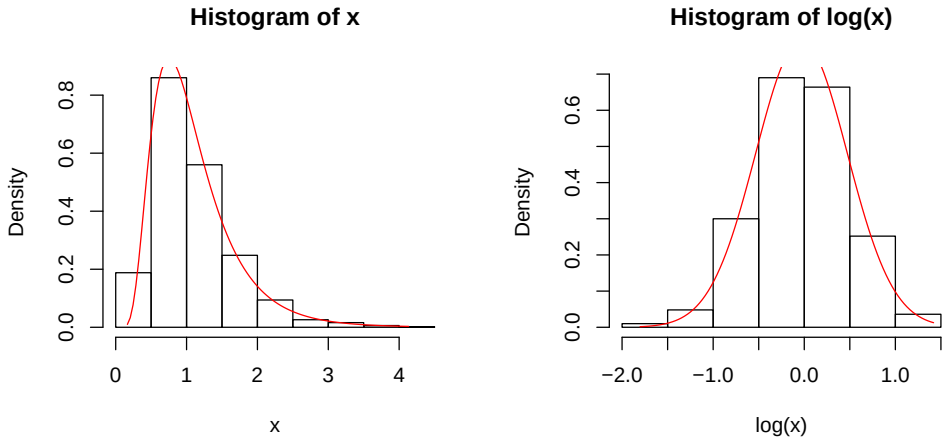



Abbildung 5.7: Logarithmische Normalverteilung (links) und dazugehörige Normalverteilung (*parent distribution*, rechts).

*Linear Models* (GLM), die z.B. Varianzanalysen für nichtnormalverteilte Daten ermöglicht. Die Gamma-Verteilung wird durch die beiden Parameter *shape* und *rate* oder alternativ durch *shape* und $scale = 1/rate$ beschrieben. Die Dichtefunktion lautet:

$$f(x) = \frac{1}{\beta^{\alpha}\Gamma(\alpha)} x^{\alpha-1}\mathbf{e}^{-x/\beta} \tag{5.6}$$

Hierbei ist $\alpha$ der *shape*-Parameter und $\beta$ ist der *scale*-Parameter. Interessanterweise ist $\alpha \cdot \beta$ der Mittelwert und $\alpha \cdot \beta^2$ ist die Varianz. Die $\chi^2$-Verteilung ist ein Spezialfall der Gamma-Verteilung mit $\alpha = df/2$, $\mu = df$ und $\sigma^2 = 2df$ und die Exponentialverteilung ist ein Spezialfall mit $\mu = \beta$, $\sigma = \beta^2$ und $\alpha = 1$.

Die Gamma-Verteilung ist demnach sehr vielfältig. Zur Visualisierung zeichnen wir uns einige Beispiele (Abb. 5.8):

```
x <- seq(0.01, 4, length=100)
par(mfrow=c(2, 2))
plot(x, dgamma(x, .5, .5), type="l")
plot(x, dgamma(x, .8, .8), type="l")
plot(x, dgamma(x, 2, 2), type="l")
plot(x, dgamma(x, 10, 10), type="l")
```

Anschließend generieren wir für diese Beispiele jeweils 1000 Zufallszahlen mit `rgamma` und berechnen den Mittelwert und die Varianz.

### Beispiel

Der Datensatz `prk_nit.txt` enthält individuelle Biomassen von *Nitzschia acicularis*-Zellen, die während zweier Praktika ermittelt wurden. Wir versuchen, eine Gamma-Verteilung anzupassen (Abb. 5.9):

```
dat <- read.table("http://www.simecol.de/data/prk_nit.txt",
  header=TRUE, sep="\t")
str(dat)
```

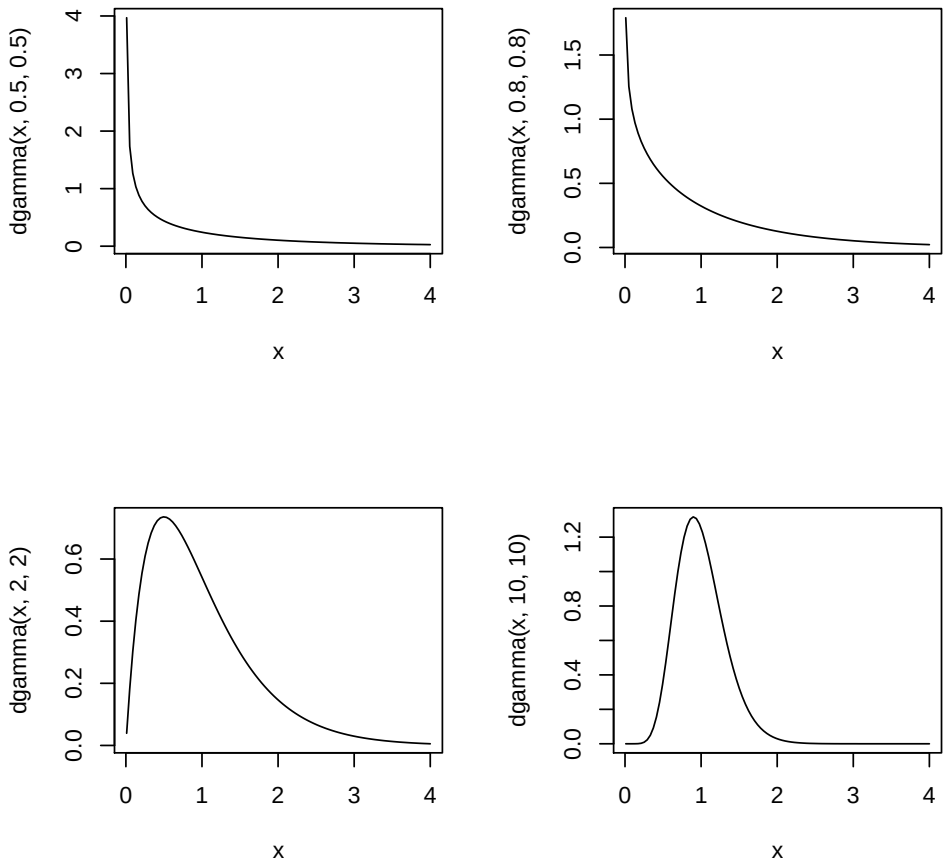



Abbildung 5.8: Beispiele für die Flexibilität der Gamma-Verteilung.

```
'data.frame':        74 obs. of  2 variables:
 $ Nit90: num  254 368 307 425 580 ...
 $ Nit85: num  438 319 554 289 404 ...
```

```
  rate <- mean(dat$Nit90)/var(dat$Nit90)
  shape <- rate * mean(dat$Nit90)
  xnew <- seq(0.01, max(dat$Nit90), length=100)
  hist(dat$Nit90, probability=TRUE)
  lines(xnew, dgamma(xnew, rate=rate, shape=shape), col="red")
```

## 5.6 Poisson-Verteilung

Die Poisson-Verteilung ist eine diskrete Verteilung. Sie findet Anwendung z.B. für Bakterien- und Planktonzählung oder für Warteschlangen und Ausfallsmodelle. Bei der Poissonverteilung gilt $\mu = \sigma^2$ und man bezeichnet diesen Mittelwertparameter mit $\lambda$. Das Vertrauensintervall hängt nur von der Anzahl der gezählten Einheiten ($k$) ab. Die Größe des Vertrauensintervalls einer Planktonzählung kann somit ganz einfach ermittelt werden:

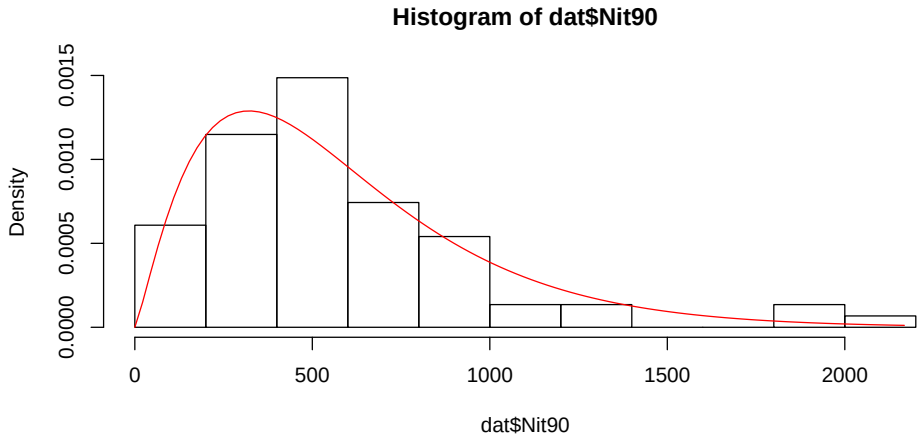



Abbildung 5.9: Histogramm des Nitzschia-Datensatzes und dazu geschätzte Gamma-Verteilung.

```
k <- 200 # gezählte Einheiten
qpois(c(0.025, 0.975), k)
```

```
[1] 173 228
```

Der Zählfehler beträgt bei $k = 200$ demnach etwa 15%, vorausgesetzt, die Organismenverteilung entspricht einer Poisson-Verteilung, d.h. es kommt nicht zur Verklumpung oder zu einer Überdispersion.

**Aufgabe:** Plotten Sie das Vertrauensintervall *relativ* zu $k$ im Bereich von $k = 5 \ldots 1000$.

## 5.7 Prüfen auf Verteilung und Transformation

Bei empirischen Daten interessiert uns oft, ob diese einer bestimmten Verteilung entsprechen, z.B. ob sie normalverteilt sind. Hierbei stoßen wir auf das allgemeine Problem, dass mit Hilfe statistischer Tests nicht auf Gleichheit oder Übereinstimmung, sondern immer nur auf signifikante Unterschiede getestet werden kann. Wird die Nullhypothese nicht abgelehnt so heißt dies lediglich, dass kein Unterschied gefunden werden konnte, d.h. dass entweder kein Unterschied existiert oder dass der Stichprobenumfang zu gering war.

Ein weiterer Aspekt besteht darin, dass es uns im allgemeinen ja eigentlich gar nicht um die „ideale Normalverteilung“ geht, sondern dass wir lediglich testen wollen, ob die Voraussetzungen für die nachfolgend geplanten Tests erfüllt sind. Bei solchen Vor-Tests benutzen wir praktisch die Irrtumswahrscheinlichkeit nur als Indikator für die Einhaltung der Testvoraussetzungen. Die gefundene Wahrscheinlichkeit ($p$-Wert) muss in solchen Fällen also größer als eine festgelegte Grenze sein, z.B. größer als 0.1. Zusätzlich werden graphische Verfahren angewendet.

### 5.7.1 Shapiro-Wilks-W-Test

Zur Prüfung auf Normalverteilung kann der Shapiro-Wilks-Test angewendet werden. Im Beispiel erzeugen wir uns 100 Zufallzahlen, die aus einer Normalverteilung stammen und prüfen, ob der Shapiro-Wilks-Test diese als normalverteilt duchgehen lässt:

```
x <- rnorm(100)
shapiro.test(x)

        Shapiro-Wilk normality test

data:  x
W = 0.99064, p-value = 0.7165
```

Im Beispiel ist der $p$-Wert größer als 0.1, d.h. von Seiten des Tests spricht nichts gegen die Annahme einer Normalverteilung. Wenn wir sagen würden, dass es sich nicht um eine Normalverteilung handelt, würden wir uns mit 72%iger Wahrscheinlichkeit irren. Allerdings können sich auf Grund des Zufallscharakters der Stichprobe (Zufallszahlengenerator) durchaus unterschiedliche Resultate ergeben.

### 5.7.2 Grafische Prüfung auf Normalverteilung

Bereits die einfache Boxplot-Darstellung (Abb. 5.10)gibt erste Hinweise, ob eine Normalverteilung wahrscheinlich ist, oder ob die Verteilung offensichtlich schief ist:

```
x1 <- rnorm(100, mean = 50, sd = 10)       # Normalverteilung
x2 <- runif(100, min = 30, max = 70)       # Gleichverteilung
x3 <- rlnorm(100, meanlog = 2, sdlog = 1) # Lognormalverteilung
boxplot(x1, x2, x3,
  names=c("Normalverteilung", "Gleichverteilung", "Lognormalverteilung"))
```

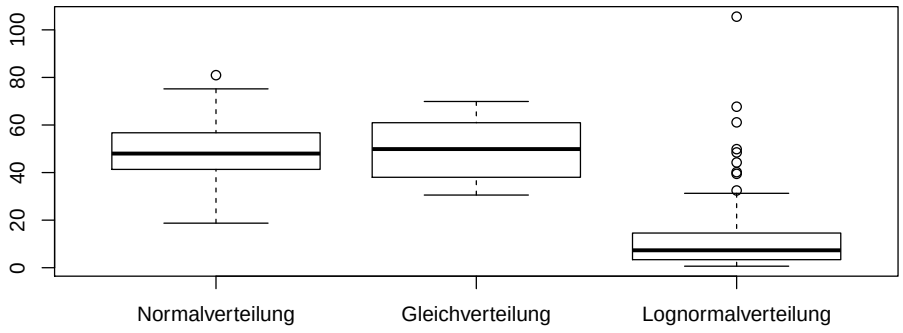



Abbildung 5.10: Vergleich unterschiedlicher Verteilungen mittels Boxplot.

Anstelle der bereits oben benutzten Häufigkeitsdiagramme (Histogramme) zur Darstellung von:

- absoluter Häufigkeit (Anzahl $f_i$ pro Klasse $i$)
- relativer Häufigkeit ($f_{i,rel} = \frac{f_i}{\sum f_i}$)
- kumulativer Häufigkeit (Summenhäufigkeit, $f_{i,cum} = \sum_{j=1}^{i} f_j$)
- oder relativer Summenhäufigkeit (Summenprozente, $f_{i,cum,rel} = \frac{f_{i,cum}}{\sum f_i}$

kann man die kumulative Häufigkeit einer beobachteten Verteilung gegen eine theoretische Verteilung (oder eine andere beobachtete Verteilung) auch ohne vorherige Klasseneinteilung mit Hilfe eines Quantil-Quantil-Plots darstellen (`qqplot`).

Speziell für die Normalverteilung existiert hierfür der „normal probability plot“ (`qqnorm`, Abb. 5.11):

```
qqnorm(x)
qqline(x, col="red")
```

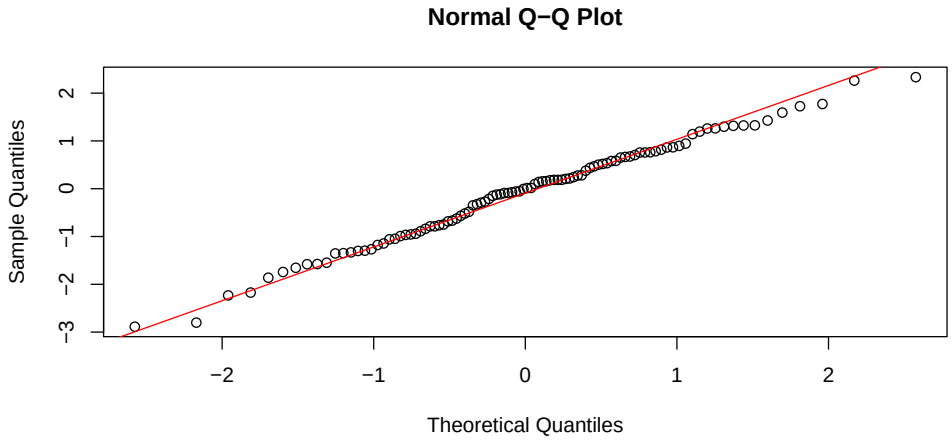



Abbildung 5.11: Quantilplot zur graphischen Prüfung auf Normalverteilung.

wobei wir dann von einer Normalverteilung ausgehen, wenn die Punkte annähernd auf einer Geraden liegen.

Diese Darstellung kann auch von Hand auf Wahrscheinlichkeitspapier gezeichnet werden. Bei großem Stichprobenumfang teilt man die Daten in Klassen ein und trägt auf der $x$-Achse die Klassenobergrenzen und auf der $y$-Achse die Summenprozente ein.

Bei kleinem Stichprobenumfang werden die Werte $x_i$ aufsteigend sortiert und für die y-Werte verwendet man zweckmäßiger Weise:

$$y_i = \frac{i - 0.3}{n + 0.4}$$

Hierbei dienen die Werte 0.3 und 0.4 zur Vermeidung von 100%, da dieses im Wahrscheinlichkeitsdiagramm im Unendlichen liegt.

**Aufgabe 1:** Erzeugen Sie eine Anzahl von Zufallszahlen mit $\mu = 50, \sigma = 10$ und stellen Sie diese mit Hilfe von R und von Hand auf Wahrscheinlichkeitspapier dar. Benutzen Sie dafür $10 \ldots 20$ Werte ohne Klasseneinteilung oder etwa 100 Werte mit Klasseneinteilung (10 Klassen).

**Aufgabe 2:** Prüfen Sie den Datendatz `nit90` graphisch und rechnerisch auf Normalverteilung.

### 5.7.3 Transformationen

Mit Hilfe einer Transformation kann man versuchen, die Daten in eine Form zu bringen, so dass die Voraussetzungen gängiger statistischer Verfahren erfüllt sind. Hierbei ist die Transformation kein fauler Trick

und keine unerlaubte Datenmanipulation. Sie dient lediglich dem Zweck, dass wir vorhandenes Wissen „recyceln“ können oder anders gesagt für unbekannte Probleme auf Bekanntes zurückgeifen können, also z.B. auf Verfahren, die für normalverteilte Daten entwickelt wurden. Eine Reihe nützlicher Transformationen finden wir in Statistikbüchern wie SACHS (1992) oder ZAR (1996) oder speziell für Wassergütevariablen in HÅKANSON and PETERS (1995). Hier einige Beispiele aus SACHS (1992):

- $x' = \ln(x), x' = \ln(x+a)$
- $x' = 1/x$ („sehr mächtig“, d.h. meist zu extrem)
- $x' = 1/\sqrt{x}$ (Mittelweg zw. ln und $1/x$)
- $x' = (x+a)^c$ ($a$ zwischen 0.5 und 1)
- $x' = a + bx^c$ (sehr allgemein, umfaßt Potenzen und Wurzeln)
- $x' = \sqrt{3/8 + x}$ (Zählungen: 1, 2, 3 $\rightarrow$ 0.61, 1.17, 1.54, 1.84, ...)
- $x' = \lg(x + 3/8)$
- $x' = \ln(\ln(x))$ für riesige Zahlen
- Prozentzahlen: Winkeltransformationen
  - $x' = \arcsin\sqrt{x/n}$
  - $x' = \arcsin\sqrt{\frac{x+3/8}{n+3/4}}$

Es ist sehr wichtig, dass die Transformationen monoton sind, d.h. dass sie die Reihenfolge der Daten unberührt lassen.

### 5.7.4 Box-Cox-Transformation

Eine besonders wichtige Form der Transformation sind die Potenzen und Logarithmen und oftmals benutzen Biologen die logarithmische Transformation rein intuitiv. Dies ist jedoch nicht immer angebracht (siehe KRAMBECK, 1995). Einen Weg, die optimale Potenz bzw. den Logarithmus zu bestimmen, ist die Box-Cox-Methode. Sie hilft bei Bestimmung der optimalen Potenz für die Transformation:

$$y' = y^{\lambda} \tag{5.7}$$

wobei ein $\lambda = 0$ bedeutet, dass die logarithmische Transformation verwendet werden muss. Die Funktion `boxcox` erwartet als Argument entweder ein `lm`- oder `aov`-Objekt (s.u.) oder eine Modellformel. Im untenstehenden Beispiel verwenden wir das sogenannte Null-Modell (`~1`), d.h. auf der rechten Seite nach der Tilde (~) steht keine Erklärungsvariable[1]:

```
library(MASS) # die Bibliothek zum Buch von Venables and Ripley
dat <- read.table("data/prk_nit.txt", header=TRUE)
boxcox(Nit90 ~ 1, data=dat)
```

[1] mehr zu Modellformeln, siehe unter Varianzanalyse

Die gepunkteten Linien zeigen den Vertrauensbereich für eine mögliche Transformation an. Man erkennt, dass entweder eine logarithmische Transformation ($\lambda = 0$) oder eine Potenztransformation mit einem Exponenten von ungefähr 0.15 angebracht ist. Wir können den Wert aus der Grafik oder den Maximalwert numerisch aus dem Boxcox-Objekt ablesen:

```
bc <- boxcox(Nit90 ~ 1, data = dat)
str(bc)
```

```
List of 2
 $ x: num [1:100] -2 -1.96 -1.92 -1.88 -1.84 ...
 $ y: num [1:100] -237 -233 -230 -226 -223 ...
```

```
bc$x[bc$y == max(bc$y)]
```

```
[1] 0.1818182
```

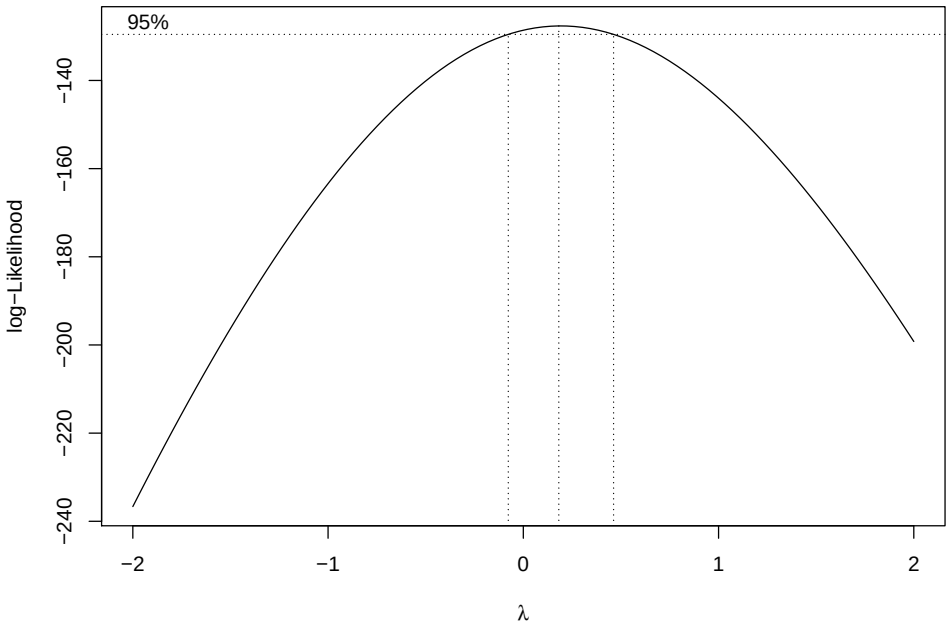



Abbildung 5.12: Prüfung auf Box-Cox-Transformation.

**Aufgabe**

Ermitteln Sie die optimale Transformation für beide Stichproben des Nitzschia-Datensatz und stellen Sie die gefundene Verteilung im Vergleich zur logarithmischen Transformation mit Hilfe eines Histogramms (mit eingezeichneter Glockenkurve) und als `qqnorm`-Plot dar. Prüfen Sie die Transformation auch mit dem Shapiro-Wilks-Test.

### 5.7.5 Rang-Transformation

Auch die Umwandlung in Ränge ist eine Transformation. Hierbei wird allerdings aus einer metrischen Variablen eine ordinale Variable, d.h. es geht Information verloren. Auf der anderen Seite sind jedoch viele

rangbasierte Tests sehr robust und trotzdem noch recht leistungsfähig.

Die Rang-Transformation wird einfach mit Hilfe von `rank` durchgeführt und auf diese Weise lässt sich auch der Spearman-Rangkorrelationskoeffizient berechnen:

```
x <- rnorm(10)
y <- rnorm(10) + x
plot(x, y)
cor(rank(x), rank(y))
```

```
[1] 0.6848485
```

## 5.8 Der zentrale Grenzwertsatz

Der Zentrale Grenzwertsatz (ZGWS) der Statistik besagt, dass die Summe einer großen Anzahl identisch verteilter und voneinander unabhängiger Zahlen normalverteilt ist, unabhängig davon, wie die Verteilung dieser Zahlen war. In der Praxis bedeutet das, dass dann eine Normalverteilung entsteht, wenn ein Prozess aus mehreren Teilprozessen mit ungefähr gleichem Fehler besteht. Dominiert jedoch ein einzelner Faktor einen Prozess (z.B. der Niederschlag den Durchfluss in einem Bach), dann erhält man meist keine Normalverteilung. Bezüglich der „großen Zahl“ ist zu sagen, dass diese Anzahl davon abhängt, wie „gutmütig“ oder symmetrisch die Originalverteilung war.

Für statistische Fragestellungen folgt aus dem ZGWS auch, dass bei Tests mit großen Stichprobenumfängen manchmal auch dann von Normalverteilung ausgegangen werden kann, wenn bei den Originadaten nachweislich Abweichungen von der Normalverteilung existieren.

Zur Veranschaulichung des ZGWS führen wir ein Simulationsexperiment durch. Wir erzeugen uns eine Matrix mit gleichverteilten Zufallszahlen in 12 Zeilen und 100 Spalten, bilden die Spaltensummen und schauen uns die Histogramme der Originaldaten und der Summen an (5.13):

```
par(mfrow=c(1,2))
x  <- matrix(runif(12*100), nrow=12)
xs <- colSums(x)
hist(x)
hist(xs)
shapiro.test(x)
```

```
        Shapiro-Wilk normality test

data:  x
W = 0.95368, p-value < 2.2e-16
```

```
shapiro.test(xs)
```

```
        Shapiro-Wilk normality test

data:  xs
W = 0.97475, p-value = 0.05161
```

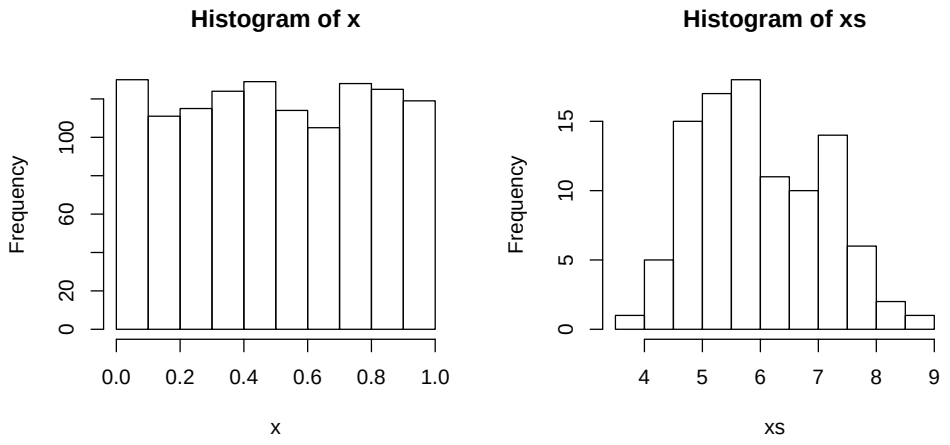



Abbildung 5.13: Histogramm einer gleichverteilten Stichprobe mit $N = 1200$ (links) sowie von 100 Mittelwerten aus je 12 Einzelwerten diesr Stichprobe (rechts). Man erkennt, dass sich die Mittelwerte schon „einigermaßen einer Normalverteilung“ annähern, obwohl die Verteilung der Stichprobe eine Gleichverteilung ist.

#### Aufgabe

Bilden Sie 100 zufällige Teilstichproben mit $n = 20$ aus dem Datensatz `Nit90` und prüfen Sie die 100 Mittelwerte graphisch und rechnerisch auf Normalverteilung. Stellen Sie außerdem die Originalstichprobe und die Mittelwerte in zwei Histogrammen mit identischer Skalierung dar.

Hinweis: Zur Teilstichprobenbildung kann die `sample`-Funktion verwendet werden.

#### Lösung

Es sind sowohl Lösungen mit Hilfe von Matrizen als auch mit Hilfe von Schleifen denkbar. Es folgt eine Lösung mit einer `for`-Schleife:

```
dat <- read.table("data/prk_nit.txt", header=TRUE)
m <- NULL
for (i in 1:100) {
  m <- c(m, mean(sample(dat$Nit90, 20)))
}
par(mfrow = c(1, 2))
hist(dat$Nit90, xlim = c(0, 2000), ylim = c(0, 25))
hist(m, xlim = c(0, 2000), ylim = c(0, 25))
```

Wiederholen Sie anschließend das Beispiel mit Stichprobenumfängen von $n = 10$ und $n = 40$ und führen Sie den Shapiro-Wilks-Test durch.

## 5.9 Vertrauensintervalle für den Mittelwert

Wie wir im vorigen Beispiel gesehen haben, ist die Streuung der Mittelwerte kleiner als die Streuung der Original-Stichprobe, wobei die Streuung der Mittelwerte umso kleiner wird, je größer der Stichprobenum-

fang ist. Der statistische Parameter, der die Genauigkeit eines Mittelwertes beschreibt, ist der Standardfehler des Mittelwertes:

$$s_{\bar{x}} = \frac{s}{\sqrt{n}} \tag{5.8}$$

Das heißt, die Streuung des Mittelwertes halbiert sich, wenn man den Stichprobenumfang vervierfacht. Man kann nun das Intervall angeben, in dem sich 95% der Mittelwerte befinden, das 95%-Konfidenzintervall:

$$CI_{95\%} = (\bar{x} - z_{0.975} \cdot \frac{s}{\sqrt{n}}, \bar{x} + z_{0.975} \cdot \frac{s}{\sqrt{n}}) \tag{5.9}$$

mit $z_{1-\alpha/2} = z_{0.975} = 1.96$. Für kleine Stichproben ($n \lessapprox 30$) oder auch ganz allgemein wird anstelle des Normalverteilungsquantils $z$ das t-Quantil mit $n-1$ Freiheitsgraden benutzt. Das folgende Beispiel zeigt die Schätzung des Konfidenzintervalls für eine normalverteilte Zufallsstichprobe mit $\mu = 50$ und $\sigma = 10$:

```
n <- 10
x <- rnorm(n, 50, 10)
m <- mean(x)
s <- sd(x)
se <- s/sqrt(n)
# untere und obere Vertrauensgrenze
m + qt(c(0.025, 0.975), n-1) * se
```

```
[1] 44.53931 59.23066
```

Bei der Angabe von Vertrauensintervallen für reale Daten müssen wir beachten, dass die Originalverteilung möglicherweise nicht normal ist und sich deshalb auch für die Mittelwerte (insbesondere bei kleinem $n$, siehe ZGWS) Abweichungen von der Normalverteilung ergeben. Eine Lösung des Problems besteht darin, zunächst die Vertrauensintervalle für eine transformierte (z.B. logarithmierte) Stichprobe zu schätzen und diese anschließend zurückzutransformieren:

```
x <- log(dat$Nit90)
m <- mean(x)
s <- sd(x)
n <- length(x)
se <- s/sqrt(n)
ci <- m + qt(c(0.025, 0.975), n-1) * se
exp(m)   # ergibt den geometrischen Mittelwert
```

```
[1] 475.8295
```

```
exp(ci)  # und dessen asymetrisches Vertrauensintervall
```

```
[1] 407.8510 555.1383
```

### 5.9.1 Ausreißer

Extrem hohe oder niedrige Werte innerhalb einer Reihe, von denen fraglich ist, ob sie überhaupt möglich sind, dürfen unter gewissen Voraussetzungen vernachlässigt werden.

Allerdings können Ausreißer auch auf neue Phänomene hindeuten, deshalb ist eine statistisch saubere Entscheidung, ob ein Wert ein Ausreißer ist, sehr schwierig. Aus diesem Grunde finden sich in Lehrbüchern auch sehr verschiedene Ausreißerregeln. Eine Möglichkeit ist die sogenannte $4\sigma$-Regel, wobei $\bar{x}$ und s ohne Ausreißer berechnet werden und $n \geq 10$ gelten sollte. Einen weiteren Ausreißertest gibt SACHS (1992) für $n \geq 25$ an. Hierzu berechnet man einen Wert $T_1$ mit:

$$T_1 = \left| \frac{x_1 - \mu}{\sigma} \right|$$

und schlägt dann in einer entsprechenden Tabelle nach. Ausreißer bzw. ihre Anzahl dürfen nicht verschwiegen werden. Für lineare Modelle und GLMs findet sich ein geeigneter Ausreißertest (Bonferroni-Ausreißertest, `outlierTest`) im Paket `car`.

```
library(car)
x <- c(rnorm(20), 12) # die 12 ist ein Ausreißer
outlierTest(lm(x~1)) # x~1 ist ein sogenanntes "Nullmodell"
```

```
   rstudent unadjusted p-value Bonferonni p
21 10.19573          3.848e-09   8.0808e-08
```

Der 21. Wert (also die 12) wird als Ausreißer erkannt.

Anstelle Ausreißer zu eliminieren, kann man auch solche Verfahren anwenden, die von vornherein unempfindlich (robust) gegenüber Ausreißern sind, z.B. Median oder getrimmte Mittelwerte anstelle des arithmetischen Mittels, Rangtests, robuste Regression oder Bootstrap-Verfahren.

#### Extremwerte in Boxplots

In der Boxplot-Funktion `boxplot` werden standardmäßig die Werte als Extremwerte markiert, die das mehr als 1,5-fache der Interquartil-Boxbreite von der Box entfernt liegen (`range=1.5`). Hierbei ist jedoch der Begriff „Extremwert“ nicht unbedingt mit „Ausreißer“ gleichzusetzen. Da dieser Sachverhalt manchmal nur schwer zu erläutern ist, empfiehlt es sich unter Umständen, die Whiskers bis zu den Extremwerten zeichnen zu lassen (`range=0`):

```
par(mfrow=c(1,2))
x <- c(1,2,3,4,5,6,12)
boxplot(x)
boxplot(x, range=0)
```

### 5.9.2 Vertrauensintervalle durch Bootstrapping

Die oben durchgeführte Schätzung des Vertrauensintervalls beruht auf der Annahme normalverteilter Mittelwerte und wie wir wissen, ist das nicht immer der Fall. Eine Möglichkeit, sich bei diesem Problem wie Münchhausen „an den eigenen Schnürsenkeln“ aus dem Sumpf zu ziehen, ist das Bootstrapping. Dieses beruht darauf, die Verteilung durch eine mehrmalige Teilstichprobenahme (mit Zurücklegen) zu approximieren und das Vertrauensintervall daraus direkt abzuleiten. Das folgende Beispiel demonstriert das Resampling:

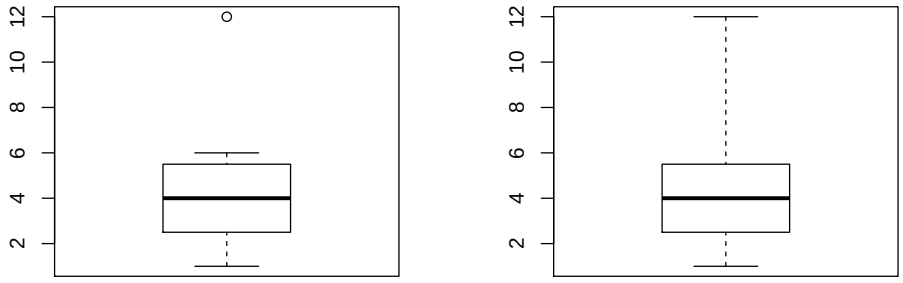



Abbildung 5.14: Boxplots mit separat ausgewiesenen Extremwerten (links) und mit Whiskers die die Extremwerte einschließen (rechts).

```
x <- 1:10
x
```

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

```
xx <- sample(x, replace=TRUE)
xx
```

```
 [1]  6  6  1 10  2  9  1  7  4  4
```

Wir wiederholen dies nun mehrmals (z.B. 1000 mal) mit dem `Nit90`-Datensatz und merken uns die Mittelwerte:

```
dat <- read.table("data/prk_nit.txt", header=TRUE)
m <- NULL
for (i in 1:1000) {
  m <- c(m, mean(sample(dat$Nit90, replace=TRUE)))
}
```

Anschließend stellen wir die 1000 Mittelwerte (`m`) als Histogramm dar und lesen die empirischen Quantile (95% zweiseitig) aus der Verteilung der Mittelwerte ab:

```
hist(m)
quantile(m, c(0.025, 0.975))
```

```
   2.5%    97.5%
501.8153 675.9295
```

Eine Visualisierung erreichen wir mit:

```
plot(sort(m), 1:length(m), type="l")
abline(v=mean(dat$Nit90), col="red")
abline(v=quantile(m, c(0.025, 0.975)), lty="dotted", col="green")
```

Zum Vergleich ermitteln wir nun noch das Vertrauensintervall für den arithmetischen Mittelwert mit Hilfe des Standardfehlers:

```
mm <- mean(dat$Nit90)
n <- length(dat$Nit90)
se <- sd(dat$Nit90)/sqrt(n)
abline(v=mm, col="blue", lty="dotted")
abline(v=c(mm - qt(0.975, n-1) * se,
           mm + qt(0.975, n-1) * se),
           col="blue", lty="dotted")
```

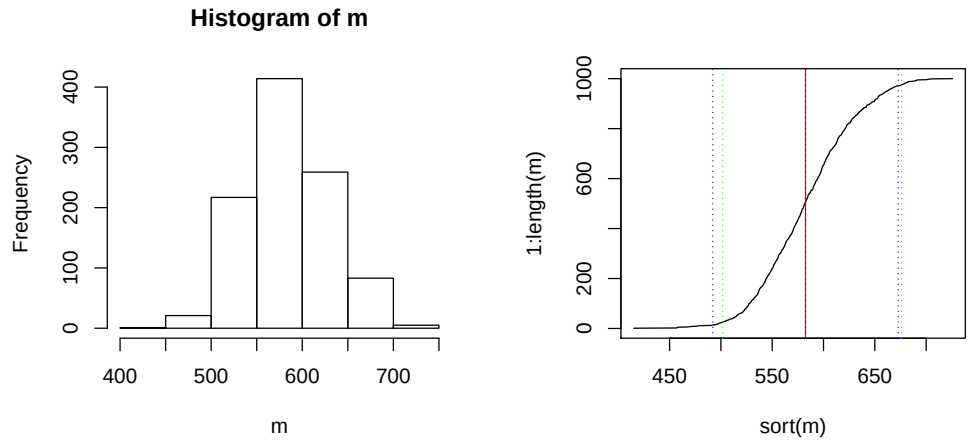



Abbildung 5.15: Nitzschia-Datensatz und Vertrauensintervalle mittels Bootstrapping (grün) bzw. parametrisch mit t-Quantil und Standardfehler (blau).

Wie wir sehen (Abb. 5.15), entspricht das Intervall ungefähr dem mit dem Bootstrap ermittelten Intervall. Dies muss nicht immer so sein, insbesondere nicht bei nicht-normalverteilten Daten. Im vorliegenden Fall resultiert die gute Übereinstimmung jedoch aus dem großen Stichprobenumfang bei der Mittelwertbildung ($n = 74$) und dem Wirken des ZGWS.

Da die Nitzscia-Daten annähernd lognormalterteilt sind, könnte man alternativ auch ein Vertrauensintervall für den Mittelwert der Logarithmen ermitteln und diese zurücktransformieren. Das Ergebnis wäre das (asymmetrische) Vertrauensintervall des geometrischen Mittels.

Das Bootstrap-Verfahren ist selbstverständlich keine Wunderwaffe, kann aber, vorausgesetzt der Stichprobenumfang ist nicht zu klein und die Verteilung ist nicht allzu extrem, ganz allgemein für die Schätzung von Vertrauensintervallen angewendet werden, nicht nur für den Mittelwert, auch für die Streuung, Korrelationskoeffizienten oder Standardfehler. Es existieren unterschiedliche Varianten des Bootstrap, davon sind mehrere in R verfügbar. Im folgenden benutzen wir BCa-Bootstrap (nonparametric, bias corrected and adjusted). Hierbei ist der Bias die Differenz aus dem normalen Mittelwert und dem Bootstrap-Mittelwert.

```
library(bootstrap)
bca <- bcanon(dat$Nit90, 1000, mean)
str(bca)
```

```
List of 5
 $ confpoints: num [1:8, 1:2] 0.025 0.05 0.1 0.16 0.84 ...
```

```
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "alpha" "bca point"
 $ z0         : num 0.0904
 $ acc        : num 0.0361
 $ u          : num [1:74] 587 585 586 585 582 ...
 $ call       : language bcanon(x = dat$Nit90, nboot = 1000, theta = mean)
```

führt einen BCa-Bootstrap mit 1000 Wiederholungen durch. Die Ergebnisse sind in einer Liste `bca` enthalten. Aus dieser Liste interessiert uns vor allem `bca$confpoints`. Eine Anzahl von 1000 Replikaten reicht im allgemeinen aus, mehr als 10000 sind nur selten sinnvoll. Mit Hilfe des Parameters `alpha` kann angeben werden, welche Quantile ausgegeben werden sollen, z.B. 0.025, 0.975 und der Bootstrap-Mittelwert selbst mit `alpha`=0.5:

```
bcanon(dat$Nit90, 1000, mean, alpha=c(0.025,0.5,0.975))$conf
```

```
     alpha bca point
[1,] 0.025  507.2676
[2,] 0.500  584.8797
[3,] 0.975  686.7378
```

Zum Abschluss dieses Teils ermitteln wir noch den Median (`median`) des `Nit90`-Datensatzes und sein Bootstrap-Vertrauensintervall. Welchem anderen statistischen Parameter der Stichprobe liegen der Median und sein Vertrauensintervall am nächsten und wie könnte das zu erklären sein?

## 5.10 Aufgaben

1. Plotten Sie mehrere Dichtefunktionen mit unterschiedlichem $\mu$ und $\sigma$ in ein gemeinsames Diagramm.
2. Plotten Sie ein kumulatives Histogramm für eine Normalverteilung mit $\bar{x} = 50, s = 10$ und zeichnen Sie die Verteilungsfunktion ein.
3. Wieviel Prozent der Messwerte liegen im Intervall $(\bar{x}-s, \bar{x}+s)$?
4. In welchem um $\bar{x}$ symmetrischen Intervall liegen 95% der Messwerte?
5. Untersuchen Sie das Blütenmerkmal „Sepal.Length“ aus dem Iris-Datensatz (`data(iris)`) graphisch und numerisch hinsichtlich seiner Verteilungseigenschaften. Kann a) für den Gesamtdatensatz und b) für die Einzelarten von angenäherter Normalverteilung ausgegangen werden?

# 6 Klassische Tests

Nicht immer ist eine trickreiche Varianzanalyse oder eine aufwändige multivariate Analyse erforderlich. In vielen Fällen ist bereits ein sogenannter klassischer Test ausreichend. Das Sparsamkeitsprinzip gilt auch hier: es sollte immer das einfachste Verfahren bevorzugt werden.

Im folgenden wird davon ausgegangen, dass die meisten Tests bereits mehr oder weniger bekannt sind. Die Darstellung beschränkt sich deshalb auf kurze Beispiele und eine Erläuterung der Anwendung in R.

## 6.1 Prüfung der Varianzgleichheit

Die Prüfung auf Varianzgleichheit benötigt man bei vielen Verfahren als Vortest, z.B. für Mittelwertvergleiche und zum zweiten als Kern weitergehender Verfahren, z.B. der Varianzanalyse. Der klassische auf dem Quotienten zweier Varianzen beruhende F-Test steht in R als `var.test` zur Verfügung. Für den Vergleich von mehr als zwei Varianzen kann entweder der parametrische Bartlett-Test oder der nichtparametrische Fligner-Killeen-Test angewendet werden. Letzterer gilt als sehr robust gegen Abweichungen von der Normalverteilung. Zur Demonstration erzeugen wir uns drei normalverteilte Datensätze mit gleicher (`x1, x3`) bzw. unterschiedlicher Varianz (`x2`) und wenden die Varianztests darauf an. Zur Visualisierung zeichnen wir einen Boxplot (Abb. 6.1):

```
x1 <- rnorm(10, 5,  1)
x2 <- rnorm(10, 5,  2)
x3 <- rnorm(10, 10, 1)
boxplot(x1, x2, x3)
```

Zum Vergleich von Varianzen stehen uns mehrere Tests zur Verfügung, der klassische $F$-Test:

```
var.test(x1, x2)
```

```
        F test to compare two variances

data:  x1 and x2
F = 0.21105, num df = 9, denom df = 9, p-value = 0.02989
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.05242167 0.84968342
sample estimates:
ratio of variances
         0.2110493
```

der Bartlett-Test, der auch für den Vergleich von mehr als 2 Varianzen geeignet ist:

```
bartlett.test(list(x1, x2, x3))
```

```
        Bartlett test of homogeneity of variances

data:  list(x1, x2, x3)
Bartlett's K-squared = 7.7136, df = 2, p-value = 0.02114
```

oder als nichtparametrische Alternative der Fligner-Killeen-Test:

```
 fligner.test(list(x1, x2, x3))

        Fligner-Killeen test of homogeneity of variances

data:  list(x1, x2, x3)
Fligner-Killeen:med chi-squared = 2.2486, df = 2, p-value = 0.3249
```

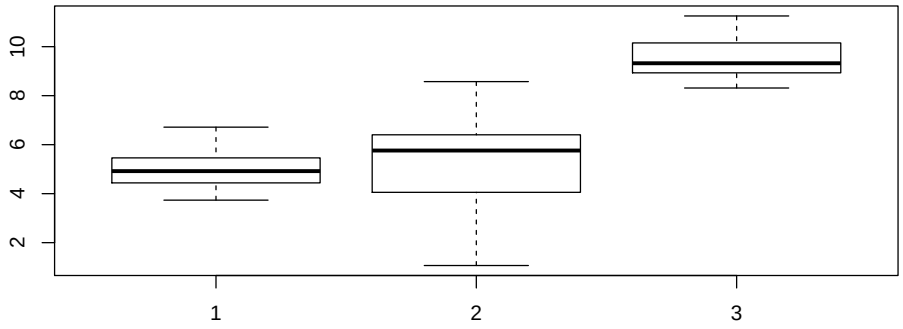



Abbildung 6.1: Boxplots für 3 Stichproben aus normalverteilten Grundgesamtheiten.

## 6.2 Prüfung von Mittelwertunterschieden

Zur Prüfung auf Mittelwertunterschiede kann der klassische t-Test angewendet werden. Der Einstichproben-Test prüft, ob eine Stichprobe aus einer Grundgesamtheit mit einem bestimmten Mittelwert $\mu$ stammt:

```
 x <- rnorm(10, 5, 1)# normalverteilte Stichprobe mit mu=5, s=1
 t.test(x, mu=5)     # stammt x aus einer Stichpr. mit mu=5?

        One Sample t-test

data:  x
t = 1.9313, df = 9, p-value = 0.08549
alternative hypothesis: true mean is not equal to 5
95 percent confidence interval:
 4.944835 5.699254
sample estimates:
mean of x
 5.322045
```

Im Zweistichproben-Fall vergleichen wir zwei unabhängige Stichproben:

```
x1 <- rnorm(10, 5, 1)
x2 <- rnorm(10, 5.5, 1)
t.test(x1, x2)
```

```
        Welch Two Sample t-test

data:  x1 and x2
t = -1.6733, df = 17.095, p-value = 0.1125
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.6509507  0.1901477
sample estimates:
mean of x mean of y
 4.991284  5.721686
```

Das bedeutet, die beiden Stichproben sind signifikant verschieden ($p < 0.05$). Allerdings hat R nicht den „normalen“ t-Test durchgeführt sondern den Welch-Test (auch heteroskedastischer t-Test genannt), bei dem die Varianzen der beiden Stichproben nicht gleich sein müssen. Haben wir jedoch geprüft (z.B. mit `var.test`), dass die Varianzen gleich sind:

```
var.test(x1, x2)  # F-Test
```

```
        F test to compare two variances

data:  x1 and x2
F = 1.5976, num df = 9, denom df = 9, p-value = 0.4962
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.3968243 6.4319774
sample estimates:
ratio of variances
          1.597612
```

...können wir auch den „normalen“ t-Test benutzen:

```
t.test(x1, x2, var.equal=TRUE) # klassischer t-Test
```

```
        Two Sample t-test

data:  x1 and x2
t = -1.6733, df = 18, p-value = 0.1116
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.647459  0.186656
sample estimates:
mean of x mean of y
 4.991284  5.721686
```

Liegen die Daten paarweise vor (z.B. Allergietest am linken und am rechten Oberarm), kann der gepaarte t-Test verwendet werden. Der Vorteil ist, dass hierbei der Einfluss der individuellen Unterschiede (also die Kovariate) eliminiert wird, der Nachteil, dass sich die Anzahl der Freiheitsgrade verringert. Wenn die Daten wirklich paarweise vorliegen ist es auf jeden Fall vorteilhaft, diese Information zu verwenden:

```
x1 <- c(2, 3, 4, 5, 6)
x2 <- c(3, 4, 7, 6, 8)
t.test(x1, x2, var.equal=TRUE) # ergibt p=0.20  n.s.
```

```
        Two Sample t-test

data:  x1 and x2
t = -1.372, df = 8, p-value = 0.2073
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.28924  1.08924
sample estimates:
mean of x mean of y
      4.0       5.6
```

```
t.test(x1, x2, paired=TRUE)    # ergibt p=0.016 also signifikant
```

```
        Paired t-test

data:  x1 and x2
t = -4, df = 4, p-value = 0.01613
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.710578 -0.489422
sample estimates:
mean of the differences
                  -1.6
```

Man erkennt, dass der paarweise t-Test eine höhere Trennschärfe besitzt.

#### Rangsummentests

Sind die Daten nicht annähernd normalverteilt, können sie eventuell transformiert werden. Als Alternative kann auch ein nichtparametrischer auf Rängen basierender Test (Wilcoxon-Test bzw. Mann-Whitney-U-Test) durchgeführt werden, die in R beide unter `wilcox.test` zusammengefasst sind.

```
dat <- read.table("data/prk_nit.txt", header=TRUE)
with(dat, boxplot(Nit90, Nit85))
with(dat, wilcox.test(Nit90, Nit85))
```

```
        Wilcoxon rank sum test with continuity correction

data:  Nit90 and Nit85
W = 3007, p-value = 3.194e-07
alternative hypothesis: true location shift is not equal to 0
```

Ein analoger Test für mehr als zwei Stichproben ist der Kruskal-Wallis-Rangsummen-Test (`kruskal.test`).

Das Paket `exactRankTests`[1] enthält verbesserte Alternativen zum Wilcoxon-Test, zum einen `wilcox.exact`, der auch Bindungen (doppelte Werte) berücksichtigt und zum anderen den Permutationstest (`perm.test`) der die exakte Wahrscheinlichkeit auf Basis einer vollständigen Permutation aller Möglichkeiten berechnet. Der Permutationstest ist vergleichsweise rechenaufwändig, bereitet aber auf heutigen PCs keinerlei Probleme. Letztlich ist der Wilcoxon-Test nur eine Approximation des Permutationstests.

Beim Permutationstest müssen wir in R darauf achten, eventuell fehlende Werte (`NA`-Werte) zu eliminieren und ggf. die kleinere Stichprobe zuerst anzugeben:

```
library(exactRankTests)
with(dat, wilcox.exact(Nit90, Nit85))
```

```
        Asymptotic Wilcoxon rank sum test

data:  Nit90 and Nit85
W = 3007, p-value = 3.153e-07
alternative hypothesis: true mu is not equal to 0
```

```
with(dat, perm.test(na.omit(Nit85), Nit90))
```

```
        Asymptotic 2-sample Permutation Test

data:  na.omit(Nit85) and Nit90
T = 16828, p-value = 7.744e-06
alternative hypothesis: true mu is not equal to 0
```

## 6.3 Prüfung auf einen Zusammenhang

#### Kontingenztafeln für nominale Variablen

Kontingenztafeln können verwendet werden, um Zusammenhänge zwischen nominalen (oder kategorialen oder qualitativen) Daten aufzudecken, z.B. zwischen Augen- und Haarfarbe, der Art einer Behandlung und der Anzahl der Heilungserfolge (geheilt/nicht geheilt) oder z.B. einem Daphnien-Klon in einem See und dem Aufenthaltsort (Tab. 6.1). Hierbei ist es wichtig, die absolute Anzahl der erfassten Individuen anzugeben (z.B. untersuchte Versuchstiere) und nicht etwa Prozentzahlen oder „Individuenzahl pro Liter“.

Zum Test erzeugen wir uns zunächst eine Matrix mit 3 Zeilen und 2 Spalten

```
x <- matrix(c(50, 37, 72, 87, 78, 45), ncol=2)
x
```

```
     [,1] [,2]
[1,]   50   87
[2,]   37   78
[3,]   72   45
```

[1] Dieses Paket funktioniert nach wie vor, wird aber nicht mehr weiterentwickelt. Stattdessen enthält das neue Paket `coin` einen verallgemeinerten Ansatz für diese und weitere nichtparametrische Tests.

Tabelle 6.1: Aufenthaltsort von Daphnien-Klongruppen im Steinbruchrestsee Gräfenhain (Klon A, Klon B, Klon C, epilimnisch=oben, hypolimnisch=unten, Daten: Matthes, Marco, unveröff.)

| | Epilimnion | Hypolimnion |
|---|---|---|
| Klon A | 50 | 87 |
| Klon B | 37 | 78 |
| Klon C | 72 | 45 |

und führen anschließend einen $\chi^2$-Test oder alternativ einen exakten Test nach Fisher (*Fisher's exact test*) durch:

```
 chisq.test(x)

        Pearson's Chi-squared test

data:  x
X-squared = 24.255, df = 2, p-value = 5.408e-06
```

oder:

```
 fisher.test(x)

        Fisher's Exact Test for Count Data

data:  x
p-value = 5.807e-06
alternative hypothesis: two.sided
```

Als Ergebnis erhalten wir, dass es einen Zusammenhang zwischen Klon und Aufenthaltsort gibt, also dass sich der Aufenthaltsort der Klone unterscheidet.

#### Korrelationskoeffizient für metrische Variablen

Für metrische Daten wird meist der Korrelationskoeffizient nach Pearson, der Rang-Korrelationskoeffizient nach Spearman oder alternativ auch Kendall's-Tau verwendet. Die entsprechenden Signifikanztests können in R mit Hilfe von `cor.test` durchgeführt werden:

```
 x <- c(1,2,3,5,7,9)
 y <- c(3,2,5,6,8,11)
 cor.test(x, y, method="pearson")

        Pearson's product-moment correlation

data:  x and y
t = 7.969, df = 4, p-value = 0.001344
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.7439930 0.9968284
sample estimates:
```

```
      cor
0.9699203
```

Bestehen Zweifel an der Linearität des Zusammenhanges oder an der Normalverteiltheit der Residuen, kann man einen Rang-Korrelationstest durchführen. Meistens wir der Korrelationskoeffizient nach Spearman verwendet:

```
 cor.test(x, y, method="spearman")

        Spearman's rank correlation rho

data:  x and y
S = 2, p-value = 0.01667
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.9428571
```

... und manchmal auch „Kendall's Tau“:

```
 cor.test(x, y, method="kendall")

        Kendall's rank correlation tau

data:  x and y
T = 14, p-value = 0.01667
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.8666667
```

Sollen lediglich die Korrelationskoeffiezienten berechnet werden, z.B. für eine ganze Matrix oder alle Variablen eines Dataframe, kann die Funktion `cor` verwendet werden.

## 6.4 Ermittlung der Power eines statistischen Tests

Die Trennschärfe eines statistischen Tests hängt neben der Struktur des verwendeten experimentellen Designs und spezifischen Eigenschaften des verwendeten Tests vor allem von:

- der relativen Effektstärke (Effekt/Standardabweichung, $\delta = \frac{(\bar{x}_1 - \bar{x}_2)}{s}$,
- dem Stichprobenumfang $n$,
- und der festgesetzten Irrtumswahrscheinlichkeit $\alpha$ ab.

Dabei gilt die Regel: je kleiner $\alpha$, $n$ und $\Delta$ sind, umso größer ist der Fehler 2. Art $\beta$, d.h. die Wahrscheinlichkeit, Effekte zu übersehen, obwohl sie existieren. Es ist deshalb angeraten, den Stichprobenumfang vor der Durchführung eines Experimentes festzulegen und das anzuwendende statistische Verfahren zu testen. Die Ermittlung des Stichprobenumfanges in Abhängigkeit von $\alpha$, $\beta$ und $\Delta$ oder umgekehrt die Abschätzung von $\beta$ bei feststehendem $n$ bezeichnet man als Poweranalyse.

Im Einstichprobenfall lässt sich der minimal notwendige Stichprobenumfang wie folgt ermitteln:

$$n = \left(\frac{z_\alpha + z_{1-\beta}}{\Delta}\right)^2$$

wobei $z$ die Quantile (`qnorm`) der Standardnormalverteilung für $\alpha$ (Irrtumswahrscheinlichkeit), $1-\beta$ die Power und $\Delta = \delta/s$ die relative Effektstärke sind.

Für den zweiseitigen Fall $\alpha = 0.025$ und $\beta = 0.2$ betragen die beiden $z$-Quantile 1.96 bzw. 0.84. Somit ergibt sich die Faustregel:

$$n = (1.96 \pm 0.84)^2 \cdot 1/\Delta^2 \approx 8 \cdot 1/\Delta^2$$

Diese Faustregel ermöglicht eine gewisse Abschätzung, gilt jedoch nur für das Einstichprobenproblem. Für andere Versuchsdesigns und Tests existieren entsprechende Nomogramme oder Formeln, z.B. in ZAR (1996).

### 6.4.1 Power eines t-Tests

Für die t-Tests kann die Power oder alternativ der notwendige Stichprobenumfang oder die minimale Effektstärke mit Hilfe der Funktion `power.t.test()` ermittelt werden. So ergibt:

```
power.t.test(n=5,delta=0.5,sig.level=0.05)

     Two-sample t test power calculation

              n = 5
          delta = 0.5
             sd = 1
      sig.level = 0.05
          power = 0.1038399
    alternative = two.sided

NOTE: n is number in *each* group
```

nur eine erschreckend niedrige Power von 0.10, d.h. bei $n = 5$ findet man einen in Wirklichkeit vorhandenen Effekt in der Größe einer halben Standardabweichung nur in einem von 10 Fällen.

Um mit $n = 5$ eine Power von 80% zu erreichen, ist eine relative Effektstärke von mindestens 2 erforderlich:

```
power.t.test(n=5,power=0.8,sig.level=0.05)

     Two-sample t test power calculation

              n = 5
          delta = 2.024438
             sd = 1
      sig.level = 0.05
          power = 0.8
```

```
    alternative = two.sided

NOTE: n is number in *each* group
```

Um einen relativ schwachen Effekt von $0.5s$ zu finden, benötigt man mindestens einen Stichprobenumfang von $n = 64$:

```
  power.t.test(delta=0.5,power=0.8,sig.level=0.05)

     Two-sample t test power calculation

              n = 63.76576
          delta = 0.5
             sd = 1
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

Es zeigt sich hier sehr klar, dass entweder ein großer Stichprobenumfang oder eine große Effektstärke nötig sind, um einen vorhandenen Effekt auch finden zu können. Anderenfalls wird die Power so niedrig, dass sich der Aufwand für ein solches Experiment eigentlich nicht lohnt.

Derzeit sind in R außerdem Funktionen für die Poweranalyse von balancierten Ein-Wege-ANOVA-Designs und für den Proportions-Test (`prop.test`) vorhanden. Für andere Fragestellungen kann die Simulationsmethode verwendet werden.

### 6.4.2 Simulationsmethode

Alternativ kann man die Power auch mittels Simulation ermitteln. Dies ist zwar vergleichsweise rechenaufwändig und liefert nur Näherungswerte, funktioniert aber prinzipiell für jedes beliebige Testdesign. Im folgenden Beispiel führen wir 1000 t-Tests mit Daten aus festgelegten Grundgesamtheiten und bekanntem Mittelwert-Unterschied durch und prüfen, in wieviel Prozent der Fälle sich ein signifikantes Resultat ergibt und zählen dies in den Variablen `a` und `b`:

```
### Simulierte Power eines t-Tests ###

# Parameter der Grundgesamtheiten
 n <- 10
 xmean1 <- 50
 xmean2 <- 55
 xsd1   <- xsd2 <- 10
 alpha  <- 0.05
# Anzahl der Simulationslaeufe
 nn <- 1000
 a <- b <- 0
 for (i in 1:nn) {
  # Erzeugen von Zufallszahlen
```

```
  x1 <- rnorm(n, xmean1, xsd1)
  x2 <- rnorm(n, xmean2, xsd2)
  # Ergebnisse des t-Testes
  p <- t.test(x1,x2,var.equal=TRUE)$p.value
  if (p < alpha) {
     a <- a+1
     } else {
     b <- b+1
  }
 }
 print(paste("a=", a, ", b=", b, ", a/n=", a/nn, ", b/n=", b/nn))
```

Wir erhalten (in etwa, da Zufallsexperiment):

```
[1] "a= 194 , b= 806 , a/n= 0.194 , b/n= 0.806"
```

Hierbei ist $a/n$ der Anteil der signifikanten Resultate, das heißt, trotz des nachweislich vorhandenen Unterschiedes zwischen den beiden Mittelwerten erhalten wir nur in etwa 20% ein signifikantes Ergebnis. Die Power $(1-\beta)$ beträgt also nur 20%.

## 6.5 Aufgaben

1. Vergleichen Sie die Mittelwerte des Blütenmerkmals „Sepal.Length“ a) von *Iris setosa* und *Iris versicolor* bzw. b) von *Iris versicolor* und *Iris virginica* des Iris-Datensatzes (`data(iris)`). Wählen Sie einen geeigneten Test und interpretieren Sie das Ergebnis.
2. Wiederholen Sie den Test mit Teilstichproben der jeweiligen Arten (z.B. n=5 oder n=10).
3. Wie groß muss generell die Effektstärke sein, um bei einem Zweistichproben-t-Test einen signifikanten Unterschied bei $n = 3, \alpha = 0.05$ und $1-\beta = 0.8$ zu erhalten?
4. Untersuchen Sie die Power eines Wilcoxon-Tests im Vergleich zu einem t-Test mit der Simulationsmethode.

# 7 Korrelation und Regression

## 7.1 Überblick

Korrelations und Regressionsanalyse dienen dazu, sich mit der Frage der Abhängigkeit zwischen zwei oder mehreren Zufallsvariablen zu beschäftigen. Die beiden Verfahren werden meist im Zusammenhang diskutiert und von der verwendeten Software oft auch in einem Schritt durchgeführt. Es existieren jedoch wichtige Unterschiede:

- Die *Korrelationsanalyse* prüft, ob *überhaupt* ein Zusammenhang existiert und wie stark dieser ist (*Signifikanztest*).

  Bei Vorliegen von zwei Stichproben betrachtet die Korrelationsanalyse beide als Zufallsgröße (Modell II), d.h. es wird nicht zwischen unabhängiger und abhängiger Größe unterschieden. Als Maß für den Zusammenhang dient der Korrelationskoeffizient $\rho$, der durch $r$ geschätzt wird.

- Die *Regressionsanalyse* versucht, Zusammenhänge durch Funktionen zu beschreiben.

  Es wird normalerweise von Modell I ausgegangen, das streng zwischen abhängigen und unabhängigen Größen unterscheidet, d.h. es wird angenommen, dass die „unabhängigen" Größen festgelegt sind und keinen Fehler besitzen. Bei einer Kalibrierkurve für ein Fotometer in der chemischen Analytik geht man z.B. davon aus, dass die Einwage der nachzuweisenden Chemikalie festgelegte Stufen besitzt und alle Fehler der Analyse (Reaktionszeit, Verunreinigungen, Messfehler des Fotometers und auch der Fehler der Einwage) als Fehler der abhängigen Variablen (also der Extinktion) erscheinen.

## 7.2 Korrelationsanalyse

### 7.2.1 Produkt-Moment-Korrelationskoeffizient nach PEARSON

Der PEARSONsche Korrelationskoeffizient ist der übliche Korrelationskoeffizient, wie wir ihn alle kennen. Mit ihm wird geprüft, ob ein *linearer* Zusammenhang vorliegt.

**Berechnung:**

$$r_P = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}}$$

**günstiger:**

$$r_P = \frac{\sum XY - \sum X \sum Y / n}{\sqrt{(\sum X^2 - (\sum X)^2/n)(\sum Y^2 - (\sum Y)^2/n)}}$$

Der Korrelationskoeffizient liegt immer im Wertebereich: $-1 \leq r_P \leq +1$.

Hierbei bedeutet:

| | |
|---|---|
| 0 | kein Zusammenhang |
| $+1$ bzw. $-1$ | funktionaler positiver bzw. negativer Zusammenhang |
| $0 < \|r_P\| < 1$ | positiver bzw. negativer Zusammenhang |

In R erhalten wir den Korrelationskoeffizienten für zwei Zufallsvariablen $x$ und $y$ mit Hilfe von:

```
x <- c(1, 2, 3, 4, 5, 4, 3)
y <- c(1, 3, 4, 4, 6, 7, 8)
cor(x,y)
```

```
[1] 0.677408
```

und einen Signifikanztest (Nullhypothese $\rho = 0$) einfach mit:

```
cor.test(x, y)
```

```
        Pearson's product-moment correlation

data:  x and y
t = 2.0592, df = 5, p-value = 0.09453
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.1544282  0.9472485
sample estimates:
     cor
0.677408
```

Zur Ermittlung „von Hand“ existieren unterschiedliche Möglichkeiten, z.B.:

1. Nachschlagen in einer Tafel „Zufallshöchstwerte des Korrelationskoeffizienten“, siehe Tabelle 7.1 oder GRIMM and RECKNAGEL (1985).
2. Approximation über das $t$-Quantil:

$$\hat{t}_{\alpha/2;n-2} = \frac{|r_P|\sqrt{n-2}}{\sqrt{1-r_P^2}}$$

3. F-Test, siehe z.B. SACHS (1992).

Der Signifikanztest dient dazu zufällige Korrelationen von nicht zufälligen (signifikanten) Korrelationen zu unterscheiden. Je mehr gemessen wurde, umso kleinere Zusammenhänge können als signifikant erkannt werden. Stehen nur zwei Wertepaare zur Verfügung, kann nicht auf Signifikanz geprüft werden, da sich durch zwei Punke immer eine Gerade legen läßt (keine Freiheitsgrade).

#### Problemfälle

Die Berechnung von $r_P$ ist zwar immer erlaubt, der Test setzt jedoch bivariate Normalverteilung, stetige Zufallsvariablen, unabhängige Beobachtungspaare und einen linearen Zusammenhang voraus. Monotone, nichtlineare Zusammenhänge verfälschen $r_P$ und auch durch Nichtnormalität und Ausreißer kann $r_P$ ganz erheblich verfälscht werden. Es empfiehlt sich deshalb auf jeden Fall eine grafische Kontrolle und gegebenenfalls eine Transformation, z.B. durch Logarithmieren.

Tabelle 7.1: Einige ausgewählte Zufallshöchstwerte ($r_{krit}$) für den Korrelationskoeffizienten (Nullhypothese$H_0 : \rho = 0$)

| $n$ | FG | $t$ | $r_krit$ |
|---|---|---|---|
| 3 | 1 | 12.706 | 0.997 |
| 5 | 3 | 3.182 | 0.878 |
| 10 | 8 | 2.306 | 0.633 |
| 20 | 18 | 2.101 | 0.445 |
| 50 | 48 | 2.011 | 0.280 |
| 100 | 98 | 1.984 | 0.197 |
| 1000 | 998 | 1.962 | 0.062 |

### 7.2.2 Rang-Korrelationskoeffizient nach SPEARMAN

Im Gegensatz zum PEARSONschen Korrelationskoeffizienten prüft der Rang-Korrelationskoeffizient nach SPEARMAN nicht auf einen linearen Zusammenhang, sondern auf einen beliebigen monoton steigenden oder monoton fallenden Zusammenhang.

Anstelle der Meßwerte werden Ränge verwendet und so ergibt sich:

$$r_S = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

wobei $d_i$ die Rangdifferenz eines Messwertpaares $d_i = x_i - y_i$ ist. Es gilt auch hier: $-1 \leq r_S \leq +1$

Zum Test kann auch hier wieder die Tabelle der Zufallshöchstwerte (siehe GRIMM and RECKNAGEL, 1985) verwendet werden.

Zufallshöchstwerte für den Rangkorrelationskoeffizienten

| $\alpha$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|
| 0.05 | 1.000 | 0.900 | 0.829 | 0.745 | 0.690 | 0.683 | 0.636 | 0.609 | 0.580 |
| 0.01 | | | 0.943 | 0.893 | 0.857 | 0.817 | 0.782 | 0.754 | 0.727 |

Ein anderer Test für kleine Stichproben ($10 \leq n \leq 20$) nutzt die $t$-Verteilung

$$\hat{t}_{1-\frac{\alpha}{2};n-2} = \frac{|r_S|}{\sqrt{1-r_S^2}}\sqrt{n-2}$$

oder für $20 \leq n$ eine Normalapproximation:

$$\hat{z}_{1-\frac{\alpha}{2}} = |r_S|\sqrt{n-1}$$

**Beispiel:** Berechnung des $r_S$ von Hand:

| $x$ | $y$ | $R_x$ | $R_y$ | $d$ | $d^2$ |
|---|---|---|---|---|---|
| 1 | 2.7 | 1 | 1 | 0 | 0 |
| 2 | 7.4 | 2 | 2 | 0 | 0 |
| 3 | 20.1 | 3 | 3 | 0 | 0 |
| 4 | 500.0 | 4 | 5 | -1 | 1 |
| 5 | 148.4 | 5 | 4 | +1 | 1 |
| | | | | | 2 |

$$r_S = \frac{6 \cdot 2}{5 \cdot (25-1)} = \frac{12}{120} = 0.9$$

Zum Vergleich: $r_P = 0.58$

**Hinweise:**

Treten häufig Bindungen (*ties*) auf so werden für identische Werte mittlere Ränge gebildet. Anschließend schätzt man $r_S$ indem man den $r_P$ der Ränge berechnet.

**Vorteile des $r_S$:**

- verteilungsunabhängig,
- prüft auf beliebige *monotone* Zusammenhänge,
- wird wenig von Ausreißern beeinflußt.

**Nachteile:**

- gewisser Informationsverlust durch Rangbildung,
- keine direkte Beziehung zum Bestimmtheitsmaß.

**Schlußfolgerung:** $r_S$ ist trotzdem sehr zu empfehlen!

### 7.2.3 Schätzung und Prüfung von $r_S$ mit R

```
x <- c(1,2,3,4,5,4,3)
y <- c(1,3,4,4,6,7,8)
cor.test(x,y, method="spearman")
```

```
        Spearman's rank correlation rho

data:  x and y
S = 21.063, p-value = 0.1343
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.6238795
```

Sind Bindungen (ties) vorhanden, gibt R eine entsprechende Warnung aus. Man schätzt in einem solchen Fall den Rangkorrelationskoeffizienten über den PEARSONschen Korrelationskoeffizienten der Ränge[1]:

```
cor.test(rank(x), rank(y))

        Pearson's product-moment correlation

data:  rank(x) and rank(y)
t = 1.785, df = 5, p-value = 0.1343
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.2436494  0.9368085
sample estimates:
      cor
0.6238795
```

Im vorliegenden Beispiel ergeben beide Verfahren identische Ergebnisse und keinen signifikanten Zusammenhang.

### 7.2.4 Multiple Korrelation

**Beispiel:** Chl-a=$f(X_1, X_2, X_3, X_4, \dots)$, wobei $X_i$ die Biomasse der $i$-ten Phytoplanktonart ist.

Es wird unterschieden zwischen multiplem Korrelationskoeffizient und partiellen Korrelationskoeffizienten. Multiple Korrelationsanalyse erscheint auf den ersten Blick als sehr leistungsfähig, in der Praxis treten jedoch ernsthafte Probleme auf, z.B.

1. wenn die unabhängigen Variablen untereinander korreliert sind führt dies zu Verfälschung des multiplen $r$ (Multikollinearität)
2. Nichtlinearitäten lassen sich bei der multiplen Regression noch schwerer handhaben als im Zweistichprobenfall.

Als Ausweg wird deshalb empfohlen, sich zunächst mit Hilfe multivariater Methoden (z.B. MDS) einen Überblick zu verschaffen und anschließend das Problem mit viel Fingerspitzengefühl und Hintergrundwissen über die beobachteten Prozesse anzugehen.

Nähere Hinweise dazu findet man in HARRELL (2001) oder QIAN (2009).

## 7.3 Lineare Regression

Mittels Regressionsanalyse wird versucht, die Art des Zusammenhanges zweier Größen (*einfache* Regression) oder mehrerer Größen (*multiple* Regression) durch eine Funktion zu beschreiben. Wenn nicht nur mehrere unabhängige Größen ($x$-Variablen) sondern aucn mehrere abhängige Größen ($y$-Variablen) untersucht werden sollen, spricht man von einer *multivariaten* Regression. Im Gegensatz zur Korrelationsanalyse, die prüft, ob überhaupt ein Zusammenhang existiert, besteht hier der Schwerpunkt in der quantitativen

[1] Man schreibt dann einfach: „Wegen Vorliegen von Bindungen wurde der Rangkorrelationskoeffizient mit Hilfe des PEARSONschen Korrelationskoeffizienten der Ränge geschätzt."

Beschreibung des Zusammenhanges durch eine Funktion. Einige Unterschiede bestehen auch in den zugrundeliegenden statistischen Annahmen. Während man bei der Korrelationsanalyse davon ausgeht, dass alle untersuchten Variablen fehlerbehaftete Zufallsvariablen sind (Modell II), unterscheidet man bei der Regressionsanalyse im allgemeinen unabhängige Größen (ohne Fehler) und abhängige Größen (mit Fehler) und bezeichnet das als Modell I (näheres hierzu z.B. in SACHS, 1992; ZAR, 1996).

### 7.3.1 Grundlagen

Lineare Modelle allgemein bilden die Basis einer Vielzahl von statistischen Methoden, z.B. linearer Regression oder Varianzanalyse. Die Gleichung eines multiplen linearen Modells lautet

$$y_i = \alpha + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p} + \varepsilon_i \tag{7.1}$$

Hierbei berechnet sich jeder Wert $y_i$ der abhängigen Variablen aus den entsprechenden Werten $x_{i,j}$ der unabhängigen Variablen, den Parametern $\alpha$ (Schnittpunkt mit der y-Achse) und $\beta_j$ (Regressionskoeffizient oder Steigung) des Modells und einem normalverteilten Fehlerterm $\varepsilon_i$ mit dem Mittelwert Null.

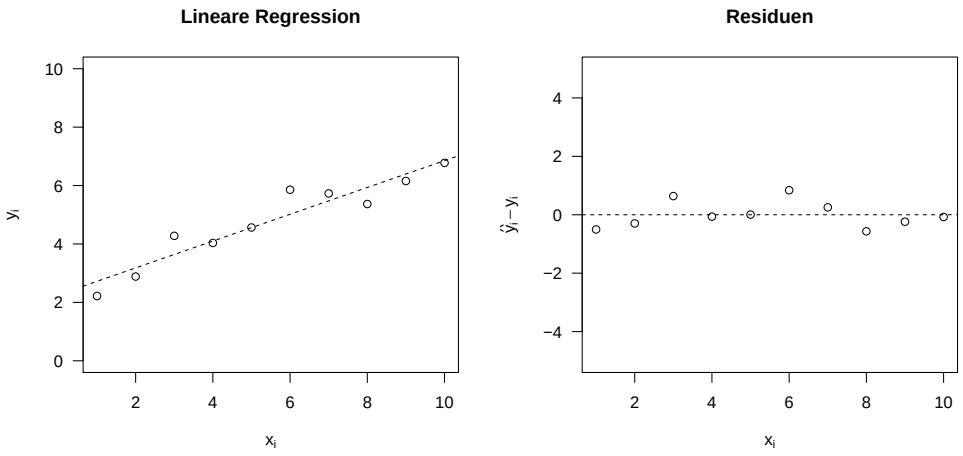



Abbildung 7.1: Lineare Regression (links) und Residuen (rechts).

Passt man z.B. ein einfaches Regressionsmodell (Ausgleichsgerade, Abb. 7.1) mit einer abhängigen Variablen $y$ und nur einer unabhängigen Variablen $x$ an, so schreibt man häufig:

$$\hat{y}_i = a + b \cdot x_i \tag{7.2}$$

Hierbei bezeichnet $\hat{y}$ die geschätzten Werte der abhängigen Variablen, d.h. die Werte auf der Geraden und $a$ bzw. $b$ die geschätzten Werte für die wahren Modellparameter $\alpha$ bzw. $\beta$.

Für die Ermittlung von $a$ und $b$ benötigt man ein Optimierungskriterium. In den meisten Fällen wird hierfür die Summe der Abweichungsquadrate $SQ = \sum(\hat{y}_i - y_i)^2$ verwendet. Um das Minimum $SQ \rightarrow Min$ zu ermitteln, setzt man die partiellen ersten Ableitungen von SQ bezogen auf die Parameter gleich Null:

$$\frac{\partial \sum(\hat{y}_i - y_i)^2}{\partial a} = \frac{\partial \sum(a + b \cdot x_i - y_i)^2}{\partial a} = 0 \tag{7.3}$$

$$\frac{\partial \sum(\hat{y}_i - y_i)^2}{\partial b} = \frac{\partial \sum(a + b \cdot x_i - y_i)^2}{\partial b} = 0 \tag{7.4}$$

Nach Lösung des resultierenden Gleichungsystems erhält man die Bestimmungsgleichungen:

$$b = \frac{\sum x_i y_i - \frac{1}{n}(\sum x_i \sum y_i)}{\sum x_i^2 - \frac{1}{n}(\sum x_i)^2} \tag{7.5}$$

$$a = \frac{\sum y_i - b \sum x_i}{n} \tag{7.6}$$

Die Ausgleichsgerade lässt sich demnach durch ein Gleichungssystem direkt aus den gegebenen Werten für $x$ und $y$ berechnen, man bezeichnet das deshalb als analytische Lösung.

#### Residuen und Bestimmtheitsmaß

Die Differenzen $\varepsilon_i = y_i - \hat{y}_i$ zwischen den gemessenen Werten der abhängigen Variablen und den mit Hilfe des Modells vorhergesagten Variablen bezeichnet man als Residuen (Abb. 7.1 rechts). Es ist offensichtlich, dass die Residuen weniger stark streuen als die ursprünglichen Messwerte $y_i$. Die Varianz der Residuen nennt man Restvarianz. Das bedeutet, dass ein Teil der ursprünglichen Streuung nun in der Geraden enthalten ist, d.h. durch das Modell erklärt wird. Der durch das Modell erklärte Anteil der Varianz an der ursprünglichen Varianz ist das Bestimmtheitsmaß $B$ (Determinationskoeffizient), das bei der *linearen* Regression gleich dem Quadrat des Korrelationskoeffizienten $r^2$ (nach Pearson) ist.

Allgemein gilt:

$$B = \frac{\text{erklärte Varianz}}{\text{ursprüngliche Varianz}} = \frac{\text{ursprüngliche Varianz} - \text{Restvarianz}}{\text{ursprüngliche Varianz}} \tag{7.7}$$

oder

$$B = \frac{s_y^2 - s_{y_i - \hat{y}_i}^2}{s_y^2} \tag{7.8}$$

und im Falle der linearen Regression

$$B = r^2 = \frac{\sum(\hat{y} - \bar{y})^2}{\sum(y - \bar{y})^2} \tag{7.9}$$

Das Bestimmtheitsmaß hat immer einen Wert zwischen 0 und 1, ein $B = 0.85$ bedeutet z.B., dass das Modell 85% der ursprünglichen Varianz erklärt.

#### Signifikanztest

Neben der Höhe des Bestimmtheitsmaßes und der grafischen Kontrolle gehört zu jeder Regressionsrechnung ein Signifikanztest. Je größer der Stichprobenumfang $n$ ist, umso kleinere Werte von $B$ können noch als signifikant erkannt werden. Zur Prüfung der Steigung $b$ benutzt man hierzu einen F-Test mit:

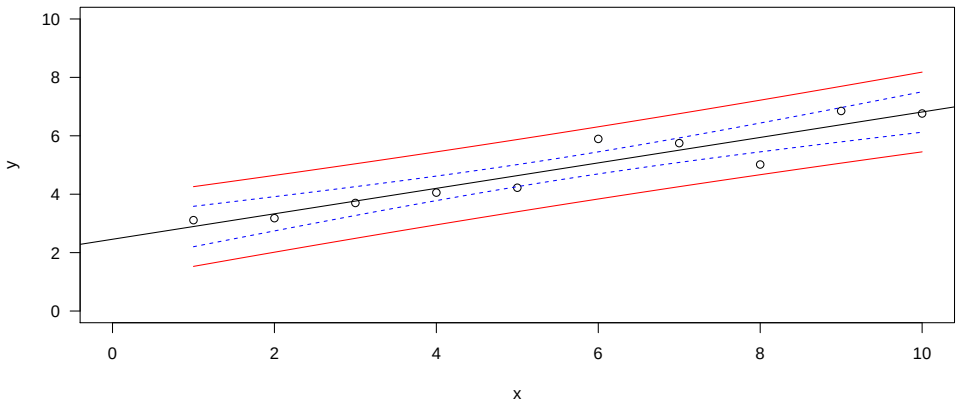



Abbildung 7.2: Lineare Regression mit Konfidenzintervall der Geraden (gestrichelt) und Vorhersageintervall (durchgezogen).

$$\hat{F}_{1;n-2;\alpha} = \frac{s^2_{erklrt}}{s^2_{Rest}} = \frac{r^2(n-2)}{1-r^2} \tag{7.10}$$

### Vertrauensintervalle

Es lassen sich Vertrauensintervalle für die Parameter (z.B. $a$ und $b$), für die Regressionsgerade selbst und auch für zukünftige Beobachtungen $Y_i$ an der Stelle $X_i$ angeben (Vorhersageintervall). Für die Parameter $a$ und $b$ erhält man die Vertrauensintervalle einfach mit Hilfe ihrer Standardfehler $s_a$ bzw. $s_b$ und dem $t$-Quantil:

$$a \pm t_{1-\alpha/2,n-2} \cdot s_a \tag{7.11}$$
$$b \pm t_{1-\alpha/2,n-2} \cdot s_b \tag{7.12}$$

Oftmals werden in der graphischen Darstellung (z.B. Abb. 7.2) hyperbolische „Konfidenzintervalle“ angegeben. Hierbei handelt es sich zum einen um das eigentliche Konfidenzintervall der Regressionsgeraden. Die hyperbolische Form setzt sich aus einer Verschiebung (Vertrauensintervall des Parameters $a$) und einer Drehung (Vertrauensintervall von $b$) zusammen. Bei einer angenommenen Irrtumswahrscheinlichkeit von $\alpha = 0.05$ liegt die wahre Regressionsgerade demnach mit 95%iger Wahrscheinlichkeit zwischen diesen Intervallgrenzen.

Das Vorhersageintervall hat dagegen eine völlig andere Bedeutung. Es sagt aus, in welchem Bereich sich ein für ein gegebenes $x$ vorhergesagter Wert $y$ befindet. Während das Konfidenzintervall demnach den mittleren Verlauf der Regressionsgeraden charakterisiert, macht das Vorhersageintervall eine Aussage über zu erwartende Einzelwerte.

Nach (SACHS, 1992) bzw. (ZAR, 1996) erhalten wir diese Intervalle wie folgt. Basierend auf der Summe der Abweichungsquadrate $Q_x$ für die $x$-Werte:

$$Q_x = \sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - \frac{1}{n}\sum_{i=1}^{n} x_i \tag{7.13}$$

und des Standardfehlers der Vorhersage (also der Standardabweichung der $\hat{y}$-Werte) für ein gegebenes $x$ ($s_{y|x}$ gelesen als „s y für x“)

$$s_{y|x} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n-2}} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - a - b \cdot x_i)}{n-2}} \tag{7.14}$$

berechnen wir zunächst die Standardabweichung für einen geschätzten Mittelwert $\underline{\hat{y}}$ und die Standardabweichung für einen vorausgesagten Einzelwert $\hat{y}_.$ an der Stelle $x$

$$s_{\underline{\hat{y}}} = s_{y|x} \cdot \sqrt{\frac{1}{n} + \frac{(x-\bar{x})^2}{Q_x}} \tag{7.15}$$

$$s_{\hat{y}_.} = s_{y|x} \cdot \sqrt{1 + \frac{1}{n} + \frac{(x-\bar{x})^2}{Q_x}} \tag{7.16}$$

und erhalten das Konfidenzintervall der Regressionsgerade als:

$$\hat{y} \pm \sqrt{2 \cdot F_{(2,n-2)} \cdot s_{\underline{\hat{y}}}} \tag{7.17}$$

und das Vorhersageintervall als:

$$\hat{y} \pm t_{(n-2)} s_{\hat{y}_.} \tag{7.18}$$

wobei $F$ und $t$ die entsprechenden Quantile der $F$- und der $t$-Verteilung sind.

#### Voraussetzungen der linearen Regression

Die Parameter $(a, b)$ werden nur dann unverzerrt geschätzt und der Signifikanztest ist nur dann zulässig, wenn die folgenden Voraussetzungen eingehalten wurden (SACHS, 1992):

1. Es gilt Modell I ($x$ ist festgelegt, $y$ ist eine Zufallsvariable).
2. Für jeden Wert von $x$ ist $y$ eine Zufallsvariable mit dem Mittelwert $\mu_{y|x}$ und der Varianz $\sigma^2_{y|x}$.
3. Die $y$-Werte sind unabhängig und identisch verteilt (keine Autokorrelation, überall gleiches $\sigma^2$)
4. Bei multipler Regression dürfen die $x_j$ nicht untereinander korreliert sein (Multikollinearitätsbedingung).
5. Die Residuen $e$ und die $y$-Werte müssen normalverteilt sein.

Hierbei ist es besonders wichtig, dass die Varianz der Residuen über den gesamten Bereich von $x$ homogen ist, d.h. die Streuung der Residuen darf nicht zu- oder abnehmen und es dürfen keine systematischen Muster erkennbar sein.

Weitere Hinweise über Grundlagen und Durchführung von Regressionsanalysen sind in KÖHLER *et al.* (2002), SACHS (1992), ZAR (1996) oder anderen Statistikbüchern zu finden, z.B. zur Berechnung von Vertrauensintervallen, zur Testung der Signifikanz von $a$ oder zu Alternativen zur Methode der kleinsten Quadrate.

### 7.3.2 Implementation in R

#### Modellformeln in R

In R existiert eine spezielle Formelsyntax zur Beschreibung von statistischen Modellen. Ein einfaches lineares Modell (y versus x) beschreibt man z.B. mit:

```
y ~ x + 1
```

oder kurz ( + 1 kann weggelassen werden):

```
y ~ x
```

eine lineare Regression, die durch den Ursprung geht (also ohne Achsenabschnitt) mit

```
y ~ x - 1
```

und eine doppelt logarithmisch transformierte Funktion mit:

```
log(y) ~ log(x)
```

Die komplette Formelsyntax und weitere Beispiele sind in der R-Dokumentation enthalten, z.B. in VENABLES *et al.* (2001).

#### Ein einfaches Beispiel

Wir erzeugen zunächst einen Vektor mit 10 `x`-Werten:

```
x <- 1:10
```

und einen davon abhängigen Vektor von `y`-Werten

```
y <- 2 + 0.5 * x + rnorm(x)
```

hierbei liefert `rnorm(x)` einen Vektor mit der gleichen Anzahl von Zufallszahlen, wie `x` Werte enthält. Die Zufallszahlen entstammen einer standardnormalverteilten Grundgesamtheit mit einem Mittelwert $\mu = 0$ und einer Standardabweichung $\sigma = 1$. Zunächst plotten wir die Daten mit:

```
plot(x, y)
```

Zur Anpassung eines linearen Modells kann die Funktion `lm()` verwendet werden:

```
reg <- lm(y ~ x)
```

Das von `lm` zurückgelieferte R-Objekt (in unserem Fall `reg` genannt) enthält die kompletten Regressionsergebnisse, Signifikanztests, Residuen und weitere Informationen, die durch spezielle R-Funktionen extrahiert werden können. So liefert:

```
summary(reg)
```

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

Residuals:
     Min       1Q   Median       3Q      Max
-1.33483 -0.21043 -0.03764  0.59020  1.04427

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.77581    0.59426   1.306 0.228008
x            0.64539    0.09577   6.739 0.000147 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8699 on 8 degrees of freedom
Multiple R-squared:  0.8502,        Adjusted R-squared:  0.8315
F-statistic: 45.41 on 1 and 8 DF,  p-value: 0.0001467
```

die wichtigsten Ergebnisse der Regression, z.B. die Koeffizienten (Coefficients), wobei der *Intercept* dem Parameter $a$ entspricht und die die Abhängigkeit von $x$ dem Steigungsparameter $b$. Das Bestimmtheitsmaß findet sich als *Multiple R-squared*, weiterhin werden Standardfehler der Parameter und Signifikanzniveaus ausgegeben.

Zur Vervollständigung der Grafik übergibt man das Ergebnisobjekt der Regression, in unserem Fall also `reg` an die Universal-Linien-Zehchenfunktion von R, `abline`:

```
abline(reg)
```

und erhält die Ausgleichsgerade. Mit Hilfe von `predict()` lassen sich für gegebene neue x-Werte die y-Werte berechnen:

```
x1 <- c(1.5, 2.5, 7.5)
y1 <- predict(reg, data.frame(x=x1))
points(x1, y1, col="green")
```

Hierbei ist zu beachten, dass `predict` die neuen Daten in einem Dataframe mit den gleichen Variablennamen erwartet, wie sie beim zugehörigen Aufruf von `lm` verwendet wurden. Mit Hilfe von `predict` lassen sich auch die in Abb 7.2 dargestellten Vertrauens- und Vorhersageintervalle erzeugen:

```
x <-1:10
y <- 2 + 0.5*x + 0.5*rnorm(x)
reg <- lm(y ~ x)
plot(x,y, ylim=c(0,10), xlim=c(0,10))
abline(reg)
newdata <- data.frame(x=seq(min(x), max(x), length=100))
```

```
conflim <- predict(reg, newdata=newdata, interval="confidence")
predlim <- predict(reg, newdata=newdata, interval="prediction")
lines(newdata$x, conflim[,2], col="blue", lty="dashed")
lines(newdata$x, conflim[,3], col="blue", lty="dashed")
lines(newdata$x, predlim[,2], col="red")
lines(newdata$x, predlim[,3], col="red")
```

Es existieren zahlreiche weitere Möglichkeiten, z.B. liefert `coef(reg)` die Koeffizienten und `resid(reg)` die Residuen für ihre weitere Verwendung. Die Funktion `plot(reg)` zeigt wichtige diagnostische Plots und mit Hilfe von `str(reg)` lassen sich die in `reg` enthaltenen Elemente oder eine Zusammenfassung der kompletten Datenstruktur anzeigen.

### 7.3.3 Übung: Beziehung zwischen Chlorophyll und Phosphat

#### Problem

Die Datei `oecd.dat` enthält aus einer Grafik aus VOLLENWEIDER and KEREKES (1980) digitalisierte Jahresmittel von Gesamtphosphat (TP $\mu g l^{-1}$) und Chlorophyll a (CHLa in $\mu g l^{-1}$) von insgesamt 92 Seen[2]. Die Daten sollen grafisch dargestellt und eine geeignete Regressionsgerade angepasst werden.

#### Lösung

Zunächst werden die Daten aus einer Textdatei eingelesen, wobei die erste Zeile den Variablennamen enthält (`header=TRUE`). Anschließend plotten wir die Daten und fitten ein lineares Modell:

```
mydata <- read.table("oecd.dat", header=TRUE)
plot(CHLa ~ TP, data = mydata)
reg <- lm(CHLa~TP, data = mydata)
abline(reg)
summary(reg)
```

Die Darstellung deutet darauf hin, dass die Voraussetzungen der linearen Regression grob verletzt wurden und sowohl die x- als auch die y-Achse transformiert werden muss. Wie in der Originalarbeit verwenden wir eine logarithmische Transformation. Hierbei benutzt R, wie die meisten anderen Programmiersprachen auch, `log()` für den natürlichen Logarithmus. Um Verwechslungen zu vermeiden, ist es im Regelfall besser, in Texten „ln“ anstelle von „log“ zu schreiben. Zur Anpassung einer Geraden an die transformierten Daten kann der Logarithmus direkt in der Modellformel innerhalb von `lm` verwendet werden:

```
plot(log(CHLa) ~ log(TP), data = mydata)
reg <- lm(log(CHLa)~log(TP), data = mydata)
abline(reg)
summary(reg)
```

### 7.3.4 Weitere Aufgaben

1. Wandeln Sie die Gleichung $\ln(CHLa) = a + b \cdot \ln(TP)$ in die Form $CHLa = a' \cdot TP^b$ um.

[2] Da einzelne Punkte auf der Grafik übereinanderliegen, fehlen 2 der ursprünglichen 94 Seen.

2. Entfernen Sie die Daten, bei denen nicht Phosphor (P), sondern Stickstoff (N) oder Licht (I) der limitierende Faktor ist und rechnen Sie die Regression neu. Stellen Sie alle Geraden in einer gemeinsamen Abbildung dar.
3. Berechnen Sie das Bestimmtheitsmaß mittels Gleichung 7.8 und vergleichen Sie diesen Wert mit dem von R ausgegebenen Ergebnis.

## 7.4 Polynome

### 7.4.1 Grundlagen

Polynome lassen sich als lineare Modelle ansehen, auch wenn die Funktion

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots \beta_p x_i^p + \varepsilon \tag{7.19}$$

und die dazugehörigen Kurven scheinbar nichtlinear sind. Der Grund hierfür ist, dass sich die Parameter $\beta_j$ über ein lineares Gleichungssystem mit $p+1$ Gleichungen und $p+1$ Unbekannten analytisch lösen (direkt ausrechnen) lassen. Hierbei nennt man $p$ die Ordnung des Polynoms. Dieses Gleichungssystem wird, genauso wie bei der linearen Regression, mit Hilfe der partiellen Ableitungen aufgestellt und ist in gängigen Statistikprogrammen bereits enthalten.

Mit Polynomen lassen sich in der Praxis nahezu alle nichtlinearen Zusammenhänge zwischen zwei oder auch mehreren Variablen (z.B. polynomiale Flächen) beschreiben. Hierbei ist folgendes zu beachten:

1. Obwohl die Datenpunkte meist sehr gut getroffen werden, ergeben sich insbesondere bei höheren Polynomordnungen Schwierigkeiten, da die Kurven zwischen den Stützstellen zu „Schwingungen“ neigen und außerhalb des Definitionsbereiches „weglaufen“ können. Polynome dürfen deshalb immer nur zur Interpolation und nicht zur Extrapolation verwendet werden.
2. Bei höheren Polynomordnungen treten häufig numerische Probleme auf, d.h. die numerische Genauigkeit des Computers reicht nicht aus, um die Koeffizienten der höheren Polynomgrade zu berechnen.[3] Die Polynomkoeffizienten (insbesondere die höheren Grade) müssen oft mit sehr vielen Stellen angegeben werden.
3. Bei $n$ Datenpunkten ist eine maximale Polynomordnung von $p = n-1$ möglich. Falls keine numerischen Probleme auftreten, geht die Kurve in diesem Fall exakt durch alle Punkte.

Es existieren zahlreiche Anwendungsbeispiele. So basieren z.B. die Formeln für die Sauerstoffsättigung des Wassers oder die Dichte des Wassers auf Polynomen (FOFONOFF, 1983; AMERICAN PUBLIC HEALTH ASSOCIATION, 1992). Allerdings sind Polynome meist lediglich „empirische Interpolationsformeln“, d.h. ohne biologische, chemische oder physikalische Begründung. Wenn immer möglich, sollten anstelle von Polynomen solche Funktionstypen verwendet werden, für die eine den Daten entsprechende naturwissenschaftliche Begründung existiert (analytische Funktionen).

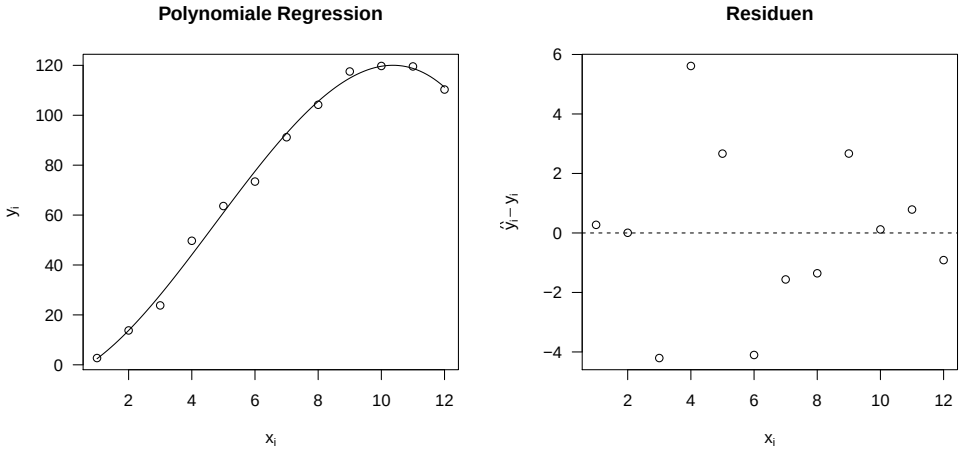



Abbildung 7.3: Polynomiale Regression (links) und Residuen (rechts).

### 7.4.2 Implementation in R

Zunächst einmal erzeugen wir einen Testdatensatz mit Hilfe von Zufallszahlen, im Beispiel ein Polynom 3. Grades mit einem normalverteilten Zufallsfehler und einer Standardabweichung von $\sigma = 3$:

```
x <- seq(1,12,1)
y <- 2*x + 3*x^2 - 0.2*x^3 + 3*rnorm(x)
plot(x, y)
```

Zur Ermittlung der Parameter („fitten“ des Modells) verwenden wir die Funktion `lm()` und die Modellformel:

```
y ~ 1 + I(x) + I(x^2) + I(x^3)}
```

Hierbei ist `I()` eine spezielle Schreibweise (*as is*), die bewirkt, dass der in der Klammer stehende Term arithmetisch (also als Potenz) und nicht „symbolisch“ im Sinne einer R-Modellformel ausgewertet wird (siehe `help(I)`). Die `1` als Symbol für den Achsenabschnitt kann auch weggelassen werden.

```
reg <- lm(y ~ x + I(x^2) + I(x^3))
summary(reg)
lines(x,reg$fitted.values, col="red")
```

Eine glattere Kurve erhalten wir, wenn wir eine höhere Anzahl (z.B. 100) Stützstellen verwenden

```
x1 <- seq(min(x), max(x), length=100)
y1 <- predict(reg, data.frame(x=x1))
lines(x1, y1, col="green")
```

Je nach den verwendeten Zufallszahlen erhalten wir dann ein Ergebnis ähnlich Abbildung 7.3.

[3] Als Alternative können auch orthogonale Polynome (`y ~ poly(x, degree)`) verwendet werden, hierbei treten jedoch andere praktische Probleme auf, z.B. funktioniert die Funktion `predict` anders als erwartet.

### 7.4.3 Übung

#### Problem

Für die Anwendung von Seenmodellen auf konkrete Gewässer oder für die Berechnung von Massenbilanzen benötigt man oft Funktionen, die die Abhängigkeit von Seefläche bzw. Volumen von der Wassertiefe oder dem Stauspiegel angeben (hypsografische Funktion oder Schlüsselkurve genannt). Es soll ein solches Polynom für die Abhängigkeit des Flächeninhaltes vom Wasserspiegel für den Datensatz `hypso.dat` gefunden werden.

#### Lösung

Zunächst sehen wir uns die Variablennamen in der Datendatei an und passen das obige Beispiel entsprechend an:

```
mydata <- read.table("hypso.dat", header=TRUE)
plot(Area ~ Level, data=mydata)
reg <- lm(Area ~ I(Level) + I(Level^2), data=mydata)
summary(reg)
lines(mydata$Level, reg$fitted.values, col="red")
x1 <- seq(0, 80, length=100)
y1 <- predict(reg, data.frame(Level=x1))
lines(x1, y1, col="green")
```

Anschließend testen wir auch andere Polynomgrade (z.B. $1 \ldots 4$) und sehen uns das Bestimmtheitsmaß und den Kurvenverlauf näher an.

#### Hinweise

Bei der Anpassung hypsografischer Funktionen ist zu beachten, dass die Flächen- und die Volumenkurven nicht voneinander unabhängig sind, da sich das Volumen als Integral der Fläche über die Tiefe ergibt (schichtenweise Aufsummierung). Stimmt dieser Zusammenhang nicht, können Bilanzfehler auftreten, d.h. Masse entsteht aus dem Nichts bzw. verschwindet spurlos. In der Praxis ermittelt man im Regelfall die Flächen aus einer Karte oder durch Lotung und integriert dann analytisch (was bei Polynomen besonders einfach ist) oder numerisch (was zu einem gewissen Fehler führt).

Wichtig ist auch der Verlauf der Kurve in der Nähe des Nullpunktes, besonders wenn mit hypolimnischen Volumina gerechnet wird. Viele in der Praxis verwendete Kurven gehen nicht durch den Nullpunkt (sogenannter Totraum) und führen zu entsprechenden Problemen.

### 7.4.4 Weitere Aufgaben

1. Berechnen Sie ein Regressionspolynom für das Volumen und vergleichen Sie dieses mit der analytisch integrierten Form.
2. Bilden Sie die erste Ableitung des Volumenpolynoms und vergleichen Sie dieses mit der Flächenkurve.
3. Bestimmen Sie Regressionspolynome mit einer Ordnung von 1 bis zur maximal möglichen Ordnung für den folgenden Datensatz und interpretieren Sie das Ergebnis

```
x <- c(1, 2, 3, 6, 8, 9, 10)
y <- c(1, 3, 2, 3, 9, 8, 7)
```

4. Extrapolieren Sie die erhaltenen Polynome über den Definitionsbereich hinaus (z.B. von -50 bis + 50), hierbei müssen eventuell die Definitions- und Wertebereiche der Grafik angepasst werden (`plot(x, y, xlim=c(-50,50))`).

## 7.5 Nichtlineare Regression

### 7.5.1 Grundlagen

Außer den Polynomen gibt es noch weitere Regressionsfunktionen, für die analytische Lösungen nach der Methode der kleinsten Quadrate existieren. Für einige weitere Funktionen existieren Linearisierungen, d.h. eine Transformation, um die nichtlineare Funktion in eine lineare Funktion umzuwandeln. So entspricht z.B.:

$$y = a \cdot x^b \tag{7.20}$$

der Funktion

$$\ln(y) = \ln(a) + b \cdot \ln(x) \tag{7.21}$$

Hierbei ist zu beachten, dass eine solche Transformation auch die Residuen transformiert. Das ist richtig und notwendig, wenn dadurch die statistischen Voraussetzungen für die Regressionsanalyse erst hergestellt werden, z.B. im obigen Beispiel der Abhängigkeit des Chlorophylls vom Gesamtphosphat.

In vielen Fällen (z.B. sehr oft bei der Monodfunktion) führt jedoch die Linearisierung zu einer unzulässigen Veränderung der Residuen mit dem Ergebnis, dass die Varianzhomogenität der Residuen verletzt wird und die Regressionsparameter verfälscht werden (bias).

Zur Lösung dieses Problems existieren unterschiedliche Ansätze, z.B. die Verwendung einer Wichtung (siehe ENGELN-MÜLLGES and REUTTER, 1996) oder die direkte numerische Anpassung der Funktion (numerische Optimierung).

#### Numerische Optimierungsverfahren

In Fällen, in denen keine analytische Lösung verfügbar ist, wenn keine Linearisierung existiert oder wenn durch die Linearisierung die Voraussetzungen der Regressionsrechnung verletzt werden, muss ein numerisches Optimierungsverfahren angewendet werden („echt nichtlineare“ Regression)[4]. Das bedeutet, dass sich die Parameter der Regressionsgleichung nicht „direkt“, d.h. **analytisch** berechnen lassen, sondern dass sie durch ein Iterationsverfahren schrittweise, d.h. **numerisch** angenähert werden müssen.

Wie auch bei der linearen Regression, wird die Güte der Anpassung meist durch die Summe der Abweichungsquadrate $SQ$ gemessen:

[4] Manchmal werden Optimierungsverfahren auch aus Bequemlichkeit angewendet, obwohl eigentlich eine analytische Lösung existiert.

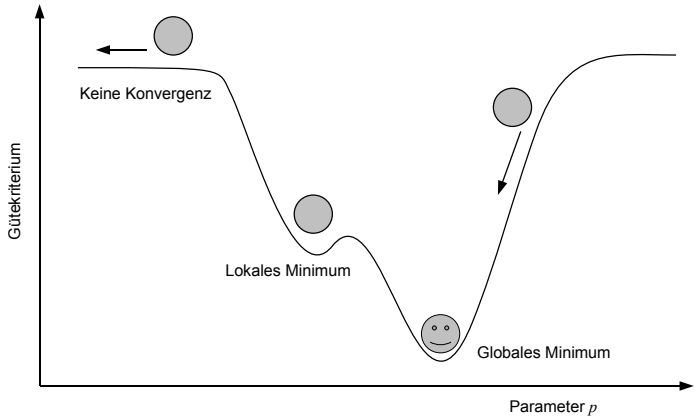



Abbildung 7.4: Numerisches Optimierungsproblem: Ziel ist das Finden des globalen Minimums einer Funktion, z.B. der Summe der kleinsten Quadrate als Gütekriterium. Sind mehrere Parameter $p_i$ zu optimieren, ergibt sich ein mehrdimensionales „Gütegebirge“.

$$SQ = \sum (y_i - f(\mathbf{x}_i, \mathbf{p}))^2 = \min! \tag{7.22}$$

wobei $y$ die abhängige Variable, $\mathbf{x}$ die unabhängigen Variablen und $\mathbf{p}$ der Parametervektor sind.

Es existieren viele unterschiedliche Optimierungsverfahren und viele Originalpublikationen und Lehrbücher (z.B. EVERITT, 1987; BATES and WATTS, 1988; PRESS *et al.*, 1992; ENGELN-MÜLLGES and REUTTER, 1996). Die einzelnen Verfahren unterscheiden sich z.B. dadurch, ob die Ableitungen der Funktionen benötigt werden oder ob die Suche ohne Ableitungen auskommt (ableitungsfrei), nach ihrer Effizienz und ihrem Verhalten bei numerisch schwierigen Problemen.

Bei den Verfahren vom Newton-Typ (z.B. Gauss-Newton-Verfahren, Newton-Raphson-Verfahren, Quasi-Newton-Verfahren) sind die partiellen zweiten Ableitungen (Hesse-Matrix) erforderlich oder werden intern numerisch geschätzt, dadurch sind diese Verfahren sehr effizient und konvergieren schnell.

Bei den Gradienten- oder Simplexverfahren wird die Richtung des steilsten Abstieges verfolgt. Die Verfahren sind weniger effizient, empfehlen sich aber z.B. zur Gewinnung von Startwerten beim Vorliegen lokaler Minima.

In sehr schwierigen Fällen (z.B. für die Kalibrierung komplexer Modelle) werden stochastische Verfahren z.B. Monte-Carlo-Methoden oder sogenannte „Evolutionsstrategien“ angewendet.

Die allgemeine Verfügbarkeit schneller Rechner und leistungsfähiger Algorithmen in Statistikpaketen und Tabellenkalkulationen hat dazu geführt, dass Optimierungsverfahren heute in vielen Fällen recht einfach angewendet werden können. Trotzdem sind eine gewisse Vorsicht und Fingerspitzengefühl immer angebracht.

#### Wahl eines geeigneten Modells

Die Auswahl eines geeigneten Regressionsmodells (Regressionsfunktion) für ein gegebenes Problem oder einen gegebenen Datensatz kann nicht vom Optimierungsalgorithmus erwartet werden, sondern muss vom

Anwender vorgenommen werden. Im Idealfall führen physikalische, chemische, biologische oder andere theoretische Betrachtungen zu einem mechanistischen Modell (BATES and WATTS, 1988). Die Modellauswahl ist naturgemäß die Aufgabe des jeweiligen, mit seiner Materie am besten vertrauten Anwenders und deshalb für uns als Naturwissenschafter ein wichtiger wissenschaftlicher Bestandteil der Modellbildung.

Bei der Auswahl des Regressionsmodells helfen Erfahrungswerte, ein entsprechendes Literaturstudium und geeignete Funktionsbibliotheken. Es sollten möglichst einfache, analytisch begründete Funktionen ausgewählt werden. Hinter guten Regressionsfunktionen (z.B. der logistischen Funktion ) verbirgt sich oft die analytische Lösung eines Differentialgleichungsmodells. Darüberhinaus ist es möglich, auch als Differentialgleichung geschriebene Kompartimentmodelle zu optimieren.

Man beginnt immer mit einem einfachen Modell und baut dieses dann gegebenenfalls durch Hinzufügen weiterer Terme und Parameter schrittweise auf.

Ein Problem kann dadurch auftreten, dass einzelne Parameter des Modells zu stark voneinander abhängig sind, d.h. dass sie sich gegenseitig kompensieren. Im trivialen Fall der Gleichung:

$$y = a + \frac{b}{c} \cdot x \tag{7.23}$$

ist es völlig offensichtlich, dass $b$ und $c$ nicht gleichzeitig bestimmt werden können, da bei einer entsprechenden Vergrößerung von $a$ und $b$ der Quotient konstant bleibt. Man spricht in einem solchen Fall davon, dass die Parameter unbestimmbar sind.

Oftmals ist der Zusammenhang nicht so offensichtlich oder auch weniger streng. Vereinfacht gesagt, hängt die Bestimmbarkeit von der Anzahl der Parameter, der Anzahl der Datenpunkte und der Varianz der Residuen ab, sowie davon, wie streng die Parameter voneinander abhängen. Als Maß dafür dient der Korrelationskoeffizient der Parameter, der möglichst gering sein sollte. Sind einzelne oder alle Parameter schwer oder nicht bestimmbar, so muss man das Modell vereinfachen und Parameter zusammenfassen, den Datensatz vergrößern oder den Messfehler verringern oder einen der fraglichen Parameter in einem zusätzlichen Experiment separat bestimmen.

#### Festlegung der Startwerte

Allen numerischen Verfahren ist gemeinsam, dass zunächst Startwerte oder zu durchsuchende Parameterbereiche festgelegt werden müssen. Der Optimierungsalgorithmus versucht nun, ein globales Minimum der Gütefunktion (Abb. 7.4) zu finden und stoppt, wenn ein Minimum gefunden wurde oder wenn nach einer vorher festgelegten Rechenzeit oder Anzahl an Iterationsschritten noch immer keine Konvergenz erreicht ist. Die Konvergenz- und Toleranzparameter können vom Anwender festgelegt werden.

Da oftmals nicht nur ein globales Minimum sondern zusätzlich noch lokale Minima existieren können, gibt es grundsätzlich keine Gewähr dafür, in endlicher Rechenzeit das globale Minimum zu finden. Aus diesem Grunde sollten stets mehrere Optimierungsläufe mit unterschiedlichen Startwerten durchgeführt werden. Man erhält gute Startwerte meist durch Überlegen, durch manuelles Ausprobieren oder durch Näherungslösungen, z.B. mit Hilfe einer linearisierenden Transformation. Für einige Funktionen existieren spezielle Verfahren zur automatischen Ermittlung von Startwerten.

#### Bewertung der gefundenen Funktion

Bei der nichtlinearen Regression ist eine grafische Überprüfung besonders wichtig. Neben der Darstellung von Messwerten und Ausgleichskurve sollten auch die Residuen angeschaut werden. Hier darf sich kein Muster und keine systematische Abweichung abzeichnen und die Varianz muss homogen sein.

Ein wichtiges Kriterium ist auch das Bestimmtheitsmaß, das bei der nichtlinearen Regression nicht einfach aus dem Quadrat des Korrelationskoeffizienten abgeleitet werden kann. Stattdessen gilt die allgemeine Formel nach Gleichung 7.8, also:

$$B = 1 - \frac{s_\varepsilon^2}{s_y^2} \tag{7.24}$$

wobei $s_\varepsilon^2$ die Varianz der Residuen ($\varepsilon = \hat{y} - y$) und $s_y^2$ die Varianz der abhängigen Variablen ist. In bestimmten Fällen kann es durchaus vorkommen, dass das Bestimmtheitsmaß negativ wird, was bedeutet, dass die Residuen eine größere Varianz besitzen als die originalen Messwerte. Der Grund dafür ist eine fehlgeschlagene Optimierung. In der grafischen Darstellung erkennt man dies sofort, da die gefittete Kurve neben den Punkten liegt. Zur Lösung des Problems kann man versuchen, die Optimierung mit neuen Startwerten zu wiederholen oder man wählt einen anderen Funktionstyp für die Regressionsgleichung.

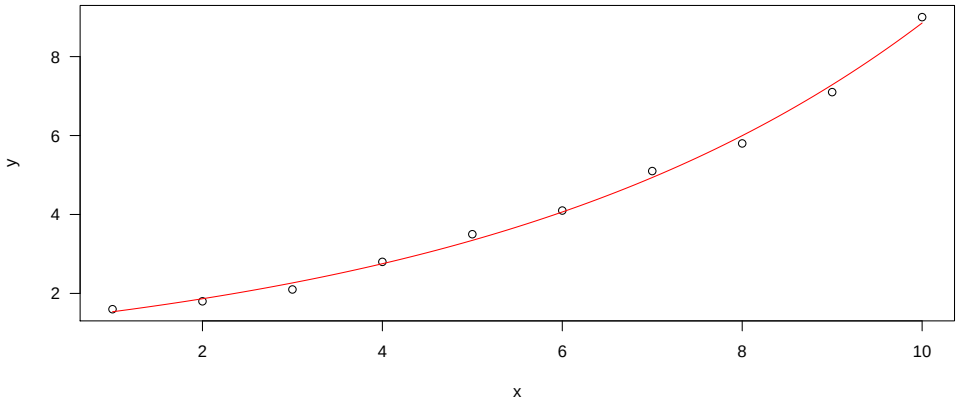



Abbildung 7.5: Exponentieller Zusammenhang.

### 7.5.2 Implementation in R

In R existieren mehrere Bibliotheken zur Optimierung nichtlinearer Probleme. Für die nichtlineare Regression mit dem Verfahren der Summe der kleinsten Quadrate wird die Funktion `nls` benutzt, die standardmäßig einen Gauss-Newton-Algorithmus verwendet. Die Funktion `nls` erwartet eine Regressionsfunktion sowie die Daten und die Startwerte als Listen.

Die zu optimierende Regressionsfunktion kann in der Modellschreibweise (z.B. eine exponentielle Wachstumsfunktion als `y~a*exp(b*x)`) angegeben werden, wobei bei der nichtlinearen Regression im Unterschied zur linearen Regression alle Symbole (z.B. `^`) immer „arithmetisch“ interpretiert werden. Noch allgemeiner ist es, die Funktion als R-Funktion zu definieren:

```
f <- function(x, a, b) {
  a * exp(b * x)
}
```

Als dritte Möglichkeit können bereits vordefinierte Autostart-Funktionen genutzt werden, bei denen auch die Startwerte automatisch ermittelt werden, z.B. `SSlogis` für das logistische Modell oder `SSmicmen` für das Michaelis-Menten-Modell.

Die Vorgehensweise soll am Beispiel eines exponentiellen Modells $y = a \cdot \exp(bx)$ erläutert werden. Dieses Modell ist sehr universell und findet sich z.B. bei der Beschreibung des exponentiellen Wachstums oder als Zerfalls- oder Absorptionsgesetz wieder. Zunächst definieren wir eine R-Funktion `f`, stellen die Daten (`x`, `y`) bereit und belegen `pstart` mit den Startwerten für das Regressionsmodell. Mit der optionalen Angabe `trace=TRUE` erreichen wir, dass die Zwischenwerte der Optimierung angezeigt werden. Zum Schluss werden die Ergebnisse auf den Bildschirm ausgedruckt und grafisch dargestellt. Für die Funktionsauswertung (zum Beispiel für die grafische Darstellung) kann wieder die Funktion `predict` verwendet werden.

```
f <- function(x, a, b) {a * exp(b * x)}
x <- 1:10
y <- c(1.6, 1.8, 2.1, 2.8, 3.5, 4.1, 5.1, 5.8, 7.1, 9.0)
pstart <- list(a=1, b=1)
plot(x, y)
aFit <- nls(y~f(x,a,b), start=pstart, trace=FALSE)
x1 <- seq(1, 10, 0.1)
y1 <- predict(aFit, list(x=x1))
lines(x1, y1, col="red")
```

Wenn man `trace=TRUE` setzen würde könnte man sehen, wie der Algorithmus iterativ die Parameter annähert. Als Ergebnis erhalten wir die Parameterwerte mit Angaben zur Signifikanz:

```
summary(aFit, correlation=TRUE)
```

```
Formula: y ~ f(x, a, b)

Parameters:
  Estimate Std. Error t value Pr(>|t|)
a 1.263586   0.049902   25.32 6.34e-09 ***
b 0.194659   0.004716   41.27 1.31e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1525 on 8 degrees of freedom

Correlation of Parameter Estimates:
  a
b -0.97

Number of iterations to convergence: 13
Achieved convergence tolerance: 2.504e-08
```

Mit der Option `correlation=TRUE` erreicht man, dass Korrelationskoeffizienten für die wechselseitige Abhängigkeit der Modellparameter (hier $a$ und $b$) ausgegeben werden.

Dieser Korrelationskoeffizient beträgt im vorliegenden Fall rund $-0.97$, hat also einen recht hohen Betrag. Das bedeutet, dass eine gewisse Wechselwirkung zwischen den Parametern besteht, die aber im vorliegenden Fall wegen der kleinen Residuen (also der „gute Daten“) kein großes Problem darstellt. Man darf diesen

Korrelationmskoeffizienten keinesfalls mit dem Korrelationskoeffizienten (oder besser, dem Bestimmtheitsmaß) des Regressionsmodells selbst verwechseln, welches sich nach der nach Gleichung 7.24 berechnen lässt:

```
1 - var(residuals(aFit))/var(y)
```

```
[1] 0.9965644
```

Das nichtlineare Bestimmtheitsmaß beträgt demnach 0.9966, d.h. das verwendete Potenzmodell erklärt 99.66% der Varianz der y-Werte.

### 7.5.3 Übung

#### Problem

Anlässlich eines Praktikums[5], wurde das heterotrophe Potential (Glucose-Aufnahmerate $AR$ in $\mu$g C l$^{-1}$h$^{-1}$) in Abhängigkeit vom Substratangebot (Glucosekonzentration $S$, in $\mu$gl$^{-1}$) bestimmt. Für eine Probe aus 2,5m Wassertiefe des Südwestbeckens der Fuchskuhle ergaben sich folgende Werte:

```
# Substrat ug C /l
S  <- c(25, 25, 10, 10, 5, 5, 2.5, 2.5, 1.25, 1.25)
# Aufnahmerate ug C /(l*h)
AR <- c(0.0998, 0.0948, 0.076, 0.0724, 0.0557,
        0.0575, 0.0399, 0.0381, 0.017, 0.0253)
```

Gesucht sind die Parameter $K$ und $V_m$ einer Michaelis-Menten-Kinetik:

$$AR = \frac{V_m \cdot S}{K + S} \tag{7.25}$$

#### Lösung 1

Oft wird die Michaelis-Menten-Formel über Linearisierungen angepasst, meist ist für dieses Problem jedoch eine nichtlineare Optimierung vorzuziehen. Wir wandeln das `nls`-Beispiel ab und erhalten:

```
f <- function(S, Vm, K) {
  Vm * S/(K + S)
}
pstart <- list(Vm=max(AR), K=5)
aFit   <- nls(AR ~ f(S, Vm, K), start=pstart, trace=TRUE)
plot(S, AR, xlim=c(0, max(S)), ylim=c(0, max(AR)))
x1 <- seq(0, 25, length=100)
lines(x1, predict(aFit,list(S=x1)), col="red")
summary(aFit)
Rsquared <- 1 - var(residuals(aFit))/var(AR)
paste("r^2=", round(Rsquared, 4))
```

[5] von Studenten der TU Dresden im September 2001 am Institut für Gewässerökologie und Binnenfischerei, Abt. Geschichtete Seen

Damit wir eine möglichst glatte Kurve erhalten, verwenden wir zur graphischen Darstellung der Kurve wieder einen Vektor `x1` mit 100 Werten über den Definitionsbereich von 0 bis 25.

#### Lösung 2

Es existiert eine noch einfachere Lösung, die Verwendung der Autostart-Funktion `SSmicmen`, bei der wir die Definition des Modells und die Angabe von Startwerten einsparen können:

```
aFit <- nls(AR ~ SSmicmen(S, Vm, K), trace=TRUE)
plot(S, AR, xlim=c(0, max(S)), ylim=c(0, max(AR)))
x1 <- seq(0, 25, length=100)
lines(x1, predict(aFit,list(S=x1)), col="red")
summary(aFit)
paste("r^2=", round(1 - var(residuals(aFit))/var(AR), 4))
```

### 7.5.4 Weitere Aufgaben

1. Linearisieren Sie das Implementationsbeispiel (exponentielles Modell), passen Sie ein lineares Modell an und vergleichen Sie die Ergebnisse.
2. Passen Sie ein geeignetes Modell an die Daten eines Batchversuches mit dem *Microcystis aeruginosa*-Stamm PCC 7806 (JÄHNICHEN *et al.*, 2001) an:

```
# Zeit (t)
x <- c(0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20)
# Zellen (pro ml)
y <- c(0.88, 1.02, 1.43, 2.79, 4.61, 7.12,
     6.47, 8.16, 7.28, 5.67, 6.91) * 1e6
```

# 8 Varianzanalyse (ANOVA)

Die unterschiedlichen Varianzanalyseverfahren sind ein ganz entscheidender der Kern der Statistik und auch von R. So handelt z.B. das Buch von CRAWLEY (2002) mit seinen 761 Seiten zu einem großen Teil (sicher mehr als die Hälfte der Seiten) von Varianzanalyseverfahren. Darüber hinaus existiert eine Online-Anleitung von FARAWAY (2002) zur Varianzanalyse mit R auf dem CRAN-Server[1]. Eine besonders anschauliche Einführung in die ANOVA bietet das online erhältliche Einleitungskapitel[2] für das Buch von GRAFEN and HAILS (2002).

Auf Grund des guten Angebots an gut verständlichen Einführungen zum Thema konzentrieren sich die folgenden Abschnitte auf ausgewählte Beispiele und Anwendungsaspekte mit R. Voraussetzung zum Verständnis ist aber unbedingt ein begleitendes Selbststudium mit einem guten Lehrbuch, z.B. DALGAARD (2002), GRAFEN and HAILS (2002), RUDOLF and KUHLISCH (2008) oder CRAWLEY (2002).

## 8.1 Ein einfaches Beispiel

In einem Batchversuch wurden für das Cyanobakterium *Pseudanabaena* Wachstumsraten bestimmt (Tab. 8.1). Dies erfolgte in einem Standard-Nährmedium (Kontrolle), unter Zugabe von einer toxischen Substanz (Microcystin) die von einer anderen Cyanobakterienart (*Microcystis aeruginosa*) produziert wird sowie unter Zugabe eines Microcystinderivates (Substanz A). Die Fragestellung ist nun, ob Microcystin oder Substanz A das Wachstum von *Pseudanabaena* beinflusst (nähere Informationen zum experimentellen Ansatz, siehe JÄHNICHEN *et al.*, 2001, 2008).

Man könnte auf die Idee kommen, den Einfluss der Substanzen durch mehrfache Anwendung eines t-Tests zu prüfen. Das ist jedoch nicht ohne weiteres zulässig, da sich bei mehrfacher Anwendung von Tests die Irrtumswahrscheinlichkeit für das Gesamtproblem erhöhen würde (Mehrfachtestproblem). Bei $\alpha = 0.05$ können bekanntlich 5% bzw. 1/20 aller Tests falsch positiv sein. Im vorliegenden Fall wären aber gleich drei t-Tests durchzuführen, d.h. die Irrtumswahrscheinlichkeit könnte maximal 3 mal 0.05 betragen und demzufolge auf einen Wert von bis zu 0.15 ansteigen. Man bezeichnet dies als Bonferroni-Regel, die besagt dass sich die Irrtumswahrscheinlichkeit $\alpha'$ beim Testen von $m$ paarweisen Hypothesen zwischen $\alpha$ und $m \cdot \alpha$ bewegt:

$$\alpha \leq \alpha' \leq m \cdot \alpha \tag{8.1}$$

Um zu verhindern, dass die Irrtumswahrscheinlichkeit $\alpha'$ größer als die vorher festgelegte Schranke, hat man zwei Möglichkeiten. Eine Möglichkeit besteht darin, $\alpha$ entsprechend zu verringern, eine andere besteht darin, den Mehrfachtest zu vermeiden und stattdessen einen Test für das Gesamtproblem durchzuführen.

Die Varianzanalyse (ANOVA) ist eine solche Möglichkeit, d.h. anstelle der Einzeltests testen wir, ob die Substanzen insgesamt einen Effekt haben.

[1] `http://cran.r-project.org`
[2] `url://www.oup.com/grafenhails/`

Tabelle 8.1: Wachstumsraten von *Pseudanabaena* in einem Batchversuch.

| Kontrolle | MCYST | Substanz A |
|---|---|---|
| 0.086 | 0.092 | 0.095 |
| 0.101 | 0.088 | 0.102 |
| 0.086 | 0.093 | 0.106 |
| 0.086 | 0.088 | 0.106 |
| 0.099 | 0.086 | 0.106 |

### ANOVA mit R

Zunächst einmal müssen die Daten eingelesen werden oder, wie im vorliegenden Fall, direkt als R-Variablen angegeben werden:

```
treat <- factor(rep(c("BW", "MCYST", "Subst A"), each = 5))
mu <- c(0.086, 0.101, 0.086, 0.086, 0.099,
        0.092, 0.088, 0.093, 0.088, 0.086,
        0.095, 0.102, 0.106, 0.106, 0.106)
```

Hierbei ist `treat` die unabhängige Variable (Behandlung) und `mu` die abhängige Variable (Wachstumsrate). Generell ist es sinnvoll, sich zunächst einmal eine graphische Vorstellung zu verschaffen:

```
boxplot(mu ~ treat)
```

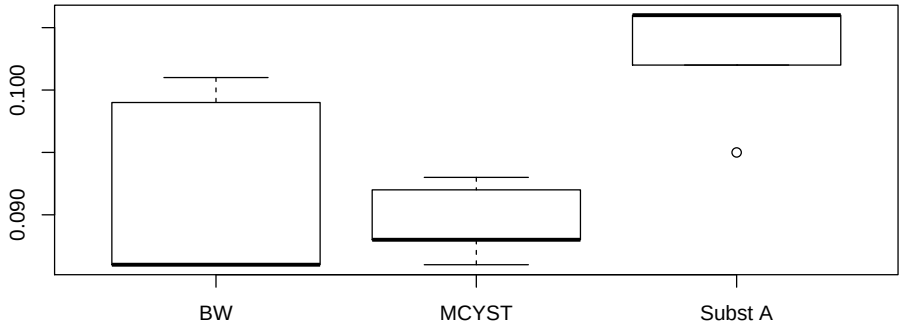



Abbildung 8.1: Boxplots für die Microcystis-Wachstumsraten.

Für die Durchführung einer ANOVA existieren grundsätzlich zwei Rechenwege, entweder die Anwendung spezieller ANOVA-Formeln zur Berechnung der Quadratsummen (z.B. in RUDOLF and KUHLISCH, 2008), oder der sehr allgemeingültige Weg über die Anpassung linearer Modelle. R geht generell den zweiten Weg und schafft dadurch eine einheitliche Herangehensweise, auch für die verallgemeinerten Spielarten der ANOVA. Allerdings unterscheiden sich die beiden Wege nur im „äußerlichen Aussehen“ der Formeln, die erhaltenen Ergebnisse sind identisch. Konkret heißt das, dass für das Microcystin-Beispiel zunächst ein lineares Modell gefittet wird, genauso wie bei einer Regression:

```
m <- lm(mu ~ treat)
```

Man kann sich nun dieses Modell ansehen:

```
summary(m)
```

```
Call:
lm(formula = mu ~ treat)

Residuals:
    Min      1Q  Median      3Q     Max
-0.0080 -0.0045 -0.0010  0.0030  0.0094

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.091600   0.002464  37.169 9.23e-14 ***
treatMCYST   -0.002200   0.003485  -0.631  0.53972
treatSubst A  0.011400   0.003485   3.271  0.00669 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.005511 on 12 degrees of freedom
Multiple R-squared:  0.5939,        Adjusted R-squared:  0.5262
F-statistic: 8.775 on 2 and 12 DF,  p-value: 0.004485
```

Man erhält im vorliegenden Fall insgesamt 3 Modellparameter, einen Achsenabschnitt (*Intercept*, dessen p-Wert hier lediglich angibt dass die Werte größer als Null sind), hier ganz konkret den Mittelwert des Blindwertes, und zwei Steigungsparameter für die Behandlungen.

Die eigentliche Varianzanalysetabelle erhält man mit der Funktion `anova`:

```
anova(m)
```

```
Analysis of Variance Table

Response: mu
          Df     Sum Sq    Mean Sq F value   Pr(>F)
treat      2 0.00053293 2.6647e-04   8.775 0.004485 **
Residuals 12 0.00036440 3.0367e-05
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Wir sehen, dass wir in unserem Design einen signifikanten Effekt mit $p = 0.0045$ vorliegen haben. Wir erhalten außerdem Informationen über die Quadratsummen (Sum Sq) und die Freiheitsgrade (Df). Was zunächst fehlt ist jedoch eine Information, welche der beiden Behandlungen (Microcystin oder Substanz A), denn einen Effekt hatte, doch dieser Test soll erst zu einem späteren Zeitpunkt nachgeholt werden.

## 8.2 Modellformeln mit R

Ein großer Teil der Flexibilität der ANOVA-Verfahren in R resultiert aus dem Konzept der Modellformeln. So lassen sich bereits mit den ANOVA-Grundfunktionen `lm` und `aov` ganz erstaunlich komplexe Probleme analysieren. Zusätzlich existieren zahlreiche weitere Funktionen mit ANOVA-Funktionalität, z.B. `gam`,

Tabelle 8.2: Beispiele für Modellformeln und R, weitere Beispiele, siehe CRAWLEY (2002)

| | | |
|---|---|---|
| Nullmodell | `y ~ 1` | nur Achsenabschitt (Konstante, Mittelwert) |
| ANOVA | `y ~ f` | $f$ als unabhängige Variable (Faktor) |
| | `y ~ f1 + f2` | Faktoren $f_1$ und $f_2$ ohne Interaktion |
| | `y ~ f1 + f2 + f1:f2` | Faktoren $f_1$ und $f_2$ mit Interaktion |
| | `y ~ f1 * f2` | Faktoren $f1$ und $f_2$ mit Interaktion |
| | `y ~ f1 * f2 - f1:f2` | Faktoren $f_1$ und $f_2$ ohne Interaktion |
| Lineare Regression | `y ~ x + 1` | $x$ als unabhängige Variable, Steigung und Achsenabschitt |
| | `y ~ x` | identisch zu `y~x+1` |
| | `y ~ x - 1` | Nur Steigung (Konstante=0) |
| Multiple Regression | `y ~ x1 + x2` | unabhängige Variablen $x_1$ und $x_2$ |
| | `y ~ x1 * x2` | unabhängige Variablen $x_1$ und $x_2$ sowie Interaktionsterm |
| ANCOVA | `y ~ f + x` | $f$ als Faktor, $x$ als Covariate, gemeinsame Steigung, separater Schnittpunkt pro Faktorstufe |
| | `y ~ f * x` | $f$ separate Steigung und separater Schnittpunkt pro Faktorstufe |
| Nested ANOVA | `y ~ a/b/c` | Faktor $c$ innerhalb $b$ und wiederum innerhalb $a$ verschachtelt |
| Mixed Effects ANOVA | `y ~ f + (1 \| g)` | zufälliger Effekt in Klammern mit additivem Term für jede Gruppe $g$ |
| | `y ~ f1 + Time * f2 + (Time \| g)` | fester Effekt von $f1$, fester Effekt von $f_2$ und Interaktion mit der Zeit, zufälliger Effekt mit Achsenabschnitt und Steigung (bzgl. Zeit) für jede Gruppe $g$. |

`lme`, `nls`, `nlme` oder `loess`. Eine Modellformel, z.B. `y~1+x*y` sieht auf den ersten Blick wie eine Rechenformel aus, wird aber symbolisch interpretiert.

Jeweils links von der Tilde steht die abhängige Variable (*response oder explained variable*) und rechts stehen die unabhängigen Variablen (*explanatory oder predictor variables*). Die Rechenzeichen werden im allgemeinen nicht algebraisch verwendet, sondern spezifizieren z.B., welche Interaktionen analysiert werden sollen oder wie das Versuchsdesign ist (z.B. *nested* oder *split-plot*).

Die unabhängigen Variablen können nominal (d.h. Faktoren) oder metrisch skaliert sein. Im Fall von ausschließlich metrischen Werten rechnet R eine (ggf. multiple) Regression, im Fall einer Mischung metrischer und nominaler Variablen eine ANCOVA. Soll stattdessen eine ANOVA gerechnet werden, müssen eventuell metrische Variablen mit `as.factor` in eine Faktorvariable umgewandelt werden.

Innerhalb von Modellformeln können auch Transformationen verwendet werden, z.B. `log(y) ~ log(x)`. Hierbei ist aber immer zu beachten, dass es sich bei Modellformeln nicht um algebraische Gleichungen sondern um eine symbolische Schreibweise handelt. In bestimmten Fällen muss durch den Ausdruck `I()` (as is) eine algebraische Interpretation erzwungen werden. So würde z.B. `y ~ x^2` als `y ~ x + x + x:x` interpretiert, d.h. da eine Interaktion einer Variablen $x$ mit sich selbst sinnlos ist lediglich als Abhängigkeit

$y$ von $x$ mit Achsenabschnitt und Steigung. Wenn wirklich eine quadratische Abhängigkeit gewünscht wird, muss `y ~ I(x^2)` geschrieben werden.

## 8.3 Zweifache Klassifikation

Die besondere Stärke der ANOVA besteht darin, dass sich der Einfluss mehrerer Einflussfaktoren simultan testen lässt. Das erspart nicht nur Arbeit beim Testen, sondern hat vor allem den Vorteil, dass auch Wechselwirkungen zwischen den einflussfaktoren identifiziert werden können. Der folgende Datensatz aus CRAWLEY (2002)[3] soll die Philosophie kurz demonstieren:

```
hams <- read.table(
  "http://www.bio.ic.ac.uk/research/mjcraw/statcomp/data/factorial.txt",
   header=TRUE)
par(mfrow=c(1,2))
boxplot(growth~diet, data=hams, xlab="diet", ylab="growth")
boxplot(growth~coat, data=hams, xlab="coat")
```

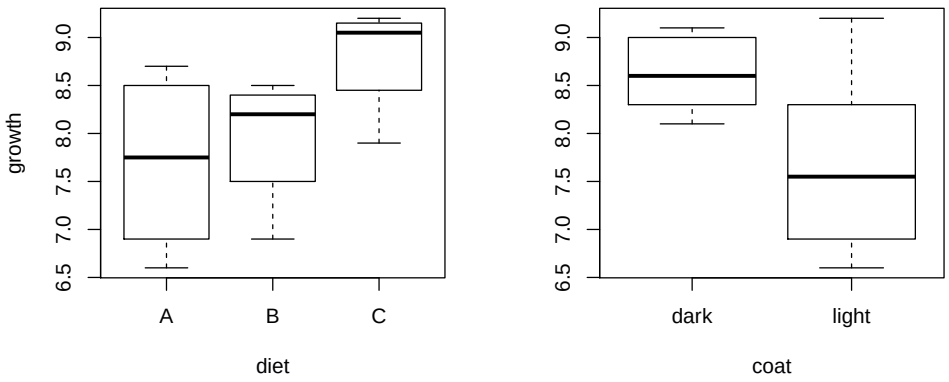



Abbildung 8.2: Boxplots für den Hamster-Datensatz (CRAWLEY, 2002).

Es fragt sich nun, ob das Wachstum der Hamster vom Futter (`diet`) oder der Hamsterrasse (`coat`) abhängt, oder ob vielleicht die unterschiedlichen Rassen verschiedene Futteransprüche haben. Als Nullhypothesen ($H_0$) formuliert heißt das:

1. Das Futter beeinflusst das Wachstum nicht.
2. Die beiden Hamsterrassen zeigen tatsächlich ein unterschiedliches Wachstum.
3. Es gibt keine Wechselwirkung (Interaktion) zwischen Futter und Rasse bezüglich Wachstum.

Auch hier können wir wieder ein Modell fitten, im vorliegenden Fall das Gesamtmodell inklusive Interaktionen:

```
m <- lm(growth ~ diet + coat + diet:coat, data=hams)
```

[3] Die Datensätze zu CRAWLEY (2002) finden sich unter `http://www.bio.ic.ac.uk/research/mjcraw/statcomp/`

wobei `diet:coat` der Interaktionseffekt ist, oder abgekürzt:

```
m <- lm(growth ~ diet * coat, data=hams)
```

Die ANOVA-Tabelle erhalten wir wieder mit:

```
anova(m)
```

```
Analysis of Variance Table

Response: growth
          Df  Sum Sq Mean Sq F value  Pr(>F)
diet       2 2.66000 1.33000  3.6774 0.09069 .
coat       1 2.61333 2.61333  7.2258 0.03614 *
diet:coat  2 0.68667 0.34333  0.9493 0.43833
Residuals  6 2.17000 0.36167
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Zunächst schaut man sich immer den Interaktionseffekt an. Falls ein solcher vorliegt, ist das oft ein interessantes wissenschaftliches Ergebnis, z.B. ob zwei Genotypen unterschiedlich auf ein Antibiotikum reagieren. Man sollte jedoch beachten, dass sich bei Vorliegen von Interaktionen die Einzeleffekte meist nicht mehr interpretieren lassen.

Im vorliegenden Beispiel finden wir keine Interaktion, deshalb kann man sich die Einzeleffekte ansehen. Im vorliegenden Fall ist nur `coat` signifikant, d.h. die Hamsterrassen besitzen unterschiedliches Wachstum, das aber nicht vom Futter abhängt[4].

Möglich wäre auch eine traditionelle ANOVA mit einem identischen Ergebnis:

```
av <- aov(growth ~ diet * coat, data=hams)
summary(av)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
diet         2 2.6600  1.3300   3.677 0.0907 .
coat         1 2.6133  2.6133   7.226 0.0361 *
diet:coat    2 0.6867  0.3433   0.949 0.4383
Residuals    6 2.1700  0.3617
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Achtung: die obigen F-Tabellen sind bei einer Zwei- oder Mehrwege-ANOVA nur dann interpretierbar, wenn das Design balanciert ist, d.h. wenn jede Faktorkombination gleich viel Parallelen enthält. Im nichtbalancierten Fall verwendet man entweder die in Kapitel 8.6 vorgestellte Methode oder sogenannte „Typ-II-Tests“, mit der Funktion `Anova` (mit Großbuchstaben A) aus dem Paket **car**.

## 8.4 Voraussetzungen der ANOVA

Zurück zum linearen Modell: Nachdem wir uns die einzelnen Effekte (`diet`, `coat`) und deren Interaktion (`diet:coat`) auf ihren Betrag (Sum of Squares) und ihr Signifikanzniveau angesehen haben, ist es eine

[4] In Wirklichkeit ist das Beispiel noch etwas subtiler, Details siehe CRAWLEY (2002)

gute Idee, die Einhaltung der Voraussetzungen der Varianzanalyse zu prüfen, insbesondere:

1. Unabhängigkeit der Stichproben,
2. Varianzhomogenität,
3. Normalverteiltheit der Residuen.

Die erste Voraussetzung „Unabhängigkeit der Stichproben“ ist die wichtigste. Sie wird durch das Design bestimmt und kann nicht durch einen Test geprüft werden. Am zweitwichtigsten ist das Kriterium „Varianzhomogenität“ und erst an dritter Stelle kommt das Kriterium Normalverteilung. Wichtig dabei, es geht um die Normalverteilung der **Residuen**, nicht etwa der Originaldaten. Das bedeutet, man muss zuerst die ANOVA rechnen (das Modell fitten), damit man die Residuen erhält und kann erst danach prüfen.

Zur Prüfung auf Varianzhomogenität kann man den Bartlett-Test oder z.B. den nichtparametrischen Fligner-Killeen-Test benutzen[5]:

```
bartlett.test(growth ~ as.factor(diet:coat), data=hams)

        Bartlett test of homogeneity of variances

data:  growth by as.factor(diet:coat)
Bartlett's K-squared = 4.4287, df = 5, p-value = 0.4895
```

Der $p$-Wert ist nicht signifikant, d.h. der Test liefert keinen Verdacht auf eine Veletzung der Varianzhomogenitätsvoraussetzung. Anschließend prüfen wir auf Normalverteilung der Residuen:

```
qqnorm(residuals(m))
qqline(residuals(m))
shapiro.test(residuals(m))

        Shapiro-Wilk normality test

data:  residuals(m)
W = 0.9596, p-value = 0.7781
```

Auch hier finden wir keinen Widerspruch, was aber in Anbetracht des kleinen $N$ nicht besonders überraschend ist. Eine wichtige Diagnose ist auch, ob eine Abhängigkeit der Residuen von den Mittelwerten besteht:

```
plot(resid(m) ~ predict(m))
```

Sollte hierbei eine Abhängigkeit sichtbar sein, ist entweder eine Transformation, ein GLM-Ansatz[6] oder ein nichtparametrisches Verfahren in Betracht zu ziehen.

Wir haben nun die Diagnostik weitgehend „von Hand“ durchgeführt. Eine automatische Schnelldiagnose ist jedoch auch ganz einfach mit Hilfe von:

```
plot(m)
```

[5] Der von anderen Statistikprogrammen (SPSS, Statistica) benutzte Levene-Test ist im Paket `car` enthalten. Es handelt sich hierbei um eine Variante die von SPSS etwas abweichende Ergebnisse liefert. Darüber hinaus ist dieser Test R-Community umstritten.

[6] Eine kurze Einführung hierzu bietet das Online-Kapitel 31 zum Buch von CRAWLEY 2002, siehe `http://www.bio.ic.ac.uk/research/mjcraw/statcomp/chapters.htm`

möglich, das eine Reihe diagnostischer Plots erzeugt. Generell gilt, dass besonders bei kleinem $N$ graphische Tests und Verantwortungsgefühl viel wichtiger sind als stures Abarbeiten der Vortests für Varianzhomogenität und Verteilung.

## 8.5 Post-hoc-Tests

Post-hoc-Tests gehören nicht unbedingt zur „reinen Lehre“ der Varianzanalyse, da sie die Verlässlichkeit des $\alpha$-Wertes untergraben und die Wahrscheinlichkeit eines Fehlers 1. Art erhöhen können. Andererseits ist gerade dies die Methode, wie oft vorgegangen wird: „Erst messen, dann Effekte suchen“, und so sollten wir wenigstens die Verfahren verwenden, die das Problem des multiplen Tests (siehe Gleichung 8.1) explizit berücksichtigen und entweder entsprechende Differenzen berechnen oder die Irrtumswahrscheinlichkeit entsprechend verschärfen. Eine oft verwendete Möglichkeit hierfür ist der Tukey-HSD-Test (*Honest Significant Difference*), für den R auch eine grafische Visualisierung bietet.

Zur Erläuterung, wie dieser Test funktioniert, verwenden wir wieder das Microcystin-Beispiel:

```
treat <- factor(c(rep("BW", 5), rep("MCYST", 5), rep("Subst A", 5)))
mu <- c(0.086, 0.101, 0.086, 0.086, 0.099,
        0.092, 0.088, 0.093, 0.088, 0.086,
        0.095, 0.102, 0.106, 0.106, 0.106)
```

Man beachte, dass der Tukey-Test ein „traditionelles“ ANOVA-Objekt (Funktion `aov`) benötigt:

```
av <- aov(mu ~ treat)
TukeyHSD(av, conf.level=0.95, ordered=TRUE)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level
    factor levels have been ordered

Fit: aov(formula = mu ~ treat)

$treat
                 diff          lwr         upr     p adj
BW-MCYST       0.0022 -0.007098057 0.01149806 0.8060911
Subst A-MCYST 0.0136  0.004301943 0.02289806 0.0055096
Subst A-BW    0.0114  0.002101943 0.02069806 0.0170476
```

Man erkennt, dass sich die Wachstumsrate bei Zusatz von Substanz A sowohl vom Blindwert als auch vom Microcystin-Treatment signifikant unterscheidet. Ein signifikanter Effekt der Microcytin-Zugabe ist jedoch nicht feststellbar.

#### Korrektur des p-Wertes

Anstelle eines speziellen post-hoc-Tests kann man auch einen normalen Test verwenden und die p-Werte nachträglich korrigieren. In der Ökologie wird hierzu gern die sequenzielle Prozedur nach HOLM (1979) verwendet. Diese ist in R nutzerfreundlich implementiert und kann direkt für den multiplen t-Test (`pairwise.t.test`) oder für den Wilcoxon-Test (`pairwise.wilcox.test`) oder ganz allgemein (`p.adjust`) verwendet werden:

```
 pairwise.t.test(mu, treat)

        Pairwise comparisons using t tests with pooled SD

data:  mu and treat

        BW     MCYST
MCYST   0.5397 -
Subst A 0.0134 0.0063

P value adjustment method: holm
```

Das ist im vorliegenden Fall weniger empfehlenswert als der Tukey-Test, allerdings auch breiter anwendbar, denn man könnte das auch mit einem anderen Test machen, z.B. dem Wilcoxon-Test (bitte selbst ausprobieren):

```
 pairwise.wilcox.test(mu, treat)
```

oder, ganz allgemein Schritt für Schritt, was auch noch den Vorteil bietet dass man nur die Tests machen muss die relevant sind und dadurch Trennschärfe gewinnt:

```
 library(exactRankTests)
 wilcox.exact(mu[treat == "BW"], mu[treat == "Subst A"])

        Exact Wilcoxon rank sum test

data:  mu[treat == "BW"] and mu[treat == "Subst A"]
W = 2, p-value = 0.03175
alternative hypothesis: true mu is not equal to 0

 wilcox.exact(mu[treat == "BW"], mu[treat == "MCYST"])

        Exact Wilcoxon rank sum test

data:  mu[treat == "BW"] and mu[treat == "MCYST"]
W = 11.5, p-value = 0.9048
alternative hypothesis: true mu is not equal to 0

 p.adjust(c(0.03175, 0.9048))

[1] 0.0635 0.9048
```

Man erkennt, dass der Wilcoxon-Test trotz verringerter Anzahl Tests (zwei statt drei) zu wenig trennsscharf ist um den signifikanten Effekt der Substanz A zu detektieren.

#### Kontraste

Eine weitere Alternative zu diesen Tests bieten Kontraste, die es erlauben, Gruppen von Mittelwerten miteinander zu vergleichen. Auch hierzu bietet R eine hervorragende Unterstützung und CRAWLEY (2002) widmet diesem Problem ein ganzes Kapitel. Unter Ausnutzung solcher Kontraste bietet das Zusatzpaket `multcomp` eine Reihe von besonders trennscharfen post-hoc-Tests:

```
 m <- lm(mu ~ treat)
 anova(m)

Analysis of Variance Table

Response: mu
          Df     Sum Sq    Mean Sq F value   Pr(>F)
treat      2 0.00053293 2.6647e-04   8.775 0.004485 **
Residuals 12 0.00036440 3.0367e-05
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

 library(multcomp)
 posthoc <- glht(m, linfct = mcp(treat = "Tukey"))
 summary(posthoc)

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = mu ~ treat)

Linear Hypotheses:
                       Estimate Std. Error t value Pr(>|t|)
MCYST - BW == 0      -0.002200   0.003485  -0.631  0.80616
Subst A - BW == 0     0.011400   0.003485   3.271  0.01703 *
Subst A - MCYST == 0  0.013600   0.003485   3.902  0.00542 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

Die Funktion `glht` ist sehr allgemeingültig und erlaubt Mehrfachtests neben den „normalen“ linearen Modellen auch für GLMs, lineare Modelle mit gemischten Effekten und Überlebensmodelle.

## 8.6 Modellselektion und Modellvereinfachung

Bei komplizierteren Modellen hängt das Ergebnis (die einzelnen Signifikanzlevel) davon ab, welche Erklärungsvariablen und welche Interaktionen benutzt wurden. In solchen Fällen ist das wechselweise Ausschalten von Einflussfaktoren und der Vergleich mehrerer ANOVA-Modelle sinnvoll.

Hierbei hängt der Vergleich nicht allein von der Güte der Anpassung, sondern auch von der Anzahl der benutzten Parameter ab. Ein Kriterium, das diesen *trade-off* berücksichtigt, ist das AIC (Akaike Information Criterion oder „penalised log likelihood“). Dieser Wert beinhaltet die Anpassungsgüte in Form des *log likelihood* ($ln(L)$), vergrößert durch $2k$, wobei $k$ die Anzahl der geschätzten Parameter ist:

$$AIC = -2ln(L) + 2k$$

Für den Modellvergleich gilt, dass AIC möglichst klein sein soll, d.h. dass die Modellanpassung (gemessen am *log likelihood*) die Anzahl der benötigten Parameter rechtfertigt.

Man kann mehrere Modelle direkt miteinander vergleichen, indem man diese separat fittet oder aus einem Grundmodell mit Hilfe von `update` ableitet:

```
 hams <- read.table(
   "http://www.bio.ic.ac.uk/research/mjcraw/statcomp/data/factorial.txt",
   header=TRUE)
 m1 <- lm(growth ~ diet * coat, data=hams)
 m2 <- lm(growth ~ diet + coat, data=hams)
 anova(m1, m2)
```

```
Analysis of Variance Table

Model 1: growth ~ diet * coat
Model 2: growth ~ diet + coat
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1      6 2.1700
2      8 2.8567 -2  -0.68667 0.9493 0.4383
```

```
 AIC(m1, m2)
```

```
   df      AIC
m1  7 27.53237
m2  5 26.83151
```

Wir erkennen, dass das Modell mit Interaktionseffekt (m1) nicht signifikant besser ist als das Modell ohne Interaktionseffekt (2), außerdem ist auch der AIC kleiner.

Ein automatisierter Vergleich aller sich aus einer Grundmodellformel ergebenden Vereinfachungen und der Vergleich des AIC ist mit Hilfe von `step` möglich, wobei man selbstverständlich das kompliziertere Modell (das *full model*) zur Vereinfachung übergibt:

```
 step(m1)
```

```
Start:  AIC=-8.52
growth ~ diet * coat

            Df Sum of Sq    RSS     AIC
- diet:coat  2   0.68667 2.8567 -9.2230
<none>                   2.1700 -8.5222

Step:  AIC=-9.22
growth ~ diet + coat

       Df Sum of Sq    RSS     AIC
<none>              2.8567 -9.2230
- diet  2    2.6600 5.5167 -5.3256
- coat  1    2.6133 5.4700 -3.4275
```

```
Call:
lm(formula = growth ~ diet + coat, data = hams)

Coefficients:
(Intercept)         dietB         dietC     coatlight
     8.1667        0.2500        1.1000       -0.9333
```

Wir erkennen, das das beste Modell das Modell mit beiden Haupteffekten, aber ohne Interaktion ist (AIC = -9.2230).

## 8.7 Aufgabe

Als weitere Übung benutzen wir ein zweites Beispiel aus (CRAWLEY, 2002, S. 265ff). Hierbei geht es um eine Untersuchung des Einflusses des Wassers aus 2 Flüssen (Tyne, Wear), 4 unterschiedlichen Detergenzien (BrandA, BrandB, BrandC, BrandD) und unterschiedlichen Klonen (Clone1, Clone2, Clone3) auf die Wachstumsrate von *Daphnia*. Es handelt sich also um eine ANOVA mit dreifacher Klassifikation. Da das Design, so wie beim Hamsterbeispiel auch, nicht nur die Analyse aller Haupteffekte sondern auch die aller Wechselwirkungen zulässt, spricht man von einer „voll-faktoriellen“ oder auch kurz „faktoriellen ANOVA“.

### Lösung

Neben der eigentlichen ANOVA sollte immer auch eine entsprechende Visualisierung und ggf. Berechnung von Mittelwerten vorgenommen werden. Im Beispiel werden dazu die Mittelwerte als Kreuztabelle in Abhängigkeit von Detergenz und Daphnia-Klon numerisch ausgegeben. Die Funktion `interaction.plot` stellt die Interaktionen dar:

```
daphnia <- read.table(
  "http://www.bio.ic.ac.uk/research/mjcraw/statcomp/data/daphnia.txt",
  header = TRUE)
par(mfrow=c(1, 3))
plot(Growth.rate ~ Water, data = daphnia, col = "blue")
plot(Growth.rate ~ Detergent, data = daphnia, col = "yellow")
plot(Growth.rate ~ Daphnia, data= daphnia, col = "orange")
m <- lm(Growth.rate ~ Water * Detergent * Daphnia, data = daphnia)
anova(m)
with(daphnia,
  tapply(Growth.rate, list(Detergent, Daphnia), mean)
)
par(mfrow=c(1, 1))
with(daphnia,
  interaction.plot(Detergent, Daphnia, Growth.rate)
)
```

Der Interaktionsplot visualisiert eventuelle Wechselwirkungen. Verlaufen die Linien „ungefähr parallel“[7], dann liegt keine Wechselwirkung vor.

Zusätzlich können weitere diagnostische Grafiken und Tests (z.B. Bartlett-Test, Q-Q-Plot) angewendet werden.

[7] Was parallel ist und was nicht, das sagen uns die p-Werte für die Interaktionsterme.

## 8.8 ANCOVA

In den vorherigen Beispielen waren die Faktoren immer nominale Variablen, also Kategorien. Will man den Einfluss metrischer (also kontinuierlicher) Variablen prüfen, dann spricht man von Covarianzanalyse (ANCOVA). In R wird die ANCOVA exakt genau so durchgeführt wie eine ANOVA, lediglich der Datentyp der Erklärungsvariablen (numerisch oder factor) bestimmt, ob eine ANOVA oder eine ANCOVA gerechnet wird. Außerdem sind beliebige Kombinationen möglich.

Als Beispiel dient ein Datensatz von DOBSON (1983) des Geburtsgewichts australischer Babies in Abhängigkeit vom Geschlecht und der Schwangerschaftswoche bei der Geburt. Als erstes plotten wir die Daten getrennt für Mädchen und für Jungen und passen separate Geraden an:

```
dat <- read.table("data/dobson.txt", head=TRUE)
names(dat)
```

```
[1] "sex"    "week"   "weight"
```

```
plot(weight ~ week, data = dat, col=c("red", "blue")[as.numeric(sex)])
fem <- lm(weight ~ week, data = dat, subset = sex=="F")
mal <- lm(weight ~ week, data = dat, subset = sex=="M")
abline(fem, col="red")
abline(mal, col="blue")
```

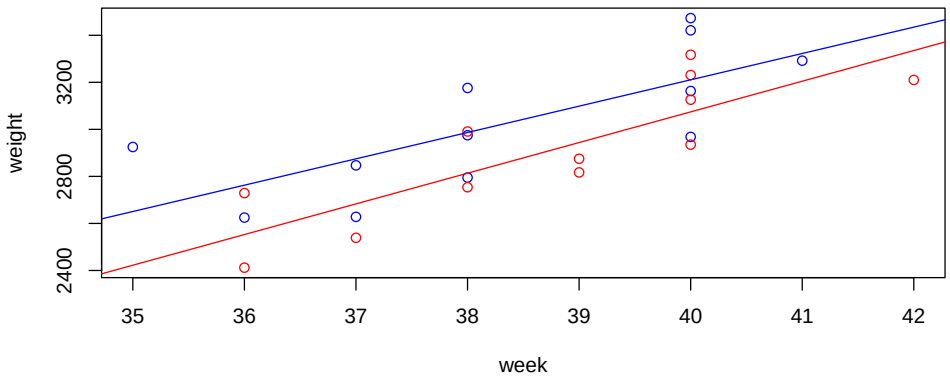



Abbildung 8.3: Abhängigkeit des Geburtsgewichtes vom Geburtszeitpunkt (rot: Mädchen, blau:Jungen).

Die Frage ist nun, ob die Geraden:

- signifikant von der Parallelität abweichen (Steigung unterschiedlich) und ob sie
- einen von Null verschiedenen Abstand besitzen (Achsenabschnitt verschieden).

Die erste Idee ist es, die Modelle direkt zu vergleichen, so wie oben:

```
anova(fem, mal)
```

```
Analysis of Variance Table
```

```
Model 1: weight ~ week
Model 2: weight ~ week
  Res.Df    RSS Df Sum of Sq F Pr(>F)
1     10 248726
2     10 403699  0   -154973
```

Aber das ist **falsch**, da jeweils unterschiedliche Daten benutzt wurden. Die richtige Vorgehensweise besteht darin, ein lineares Modell an den gesamten Datensatz anzupassen. Hierbei verwendet R automatisch nur Variablen vom Typ `factor` als Behandlungen (Faktoren) im Sinne der ANOVA und alle numerischen Variablen (im Beispiel also `week`) als Kovariate:

```
 str(dat) # welche Variablen sind Faktoren?
```

```
'data.frame':        24 obs. of  3 variables:
 $ sex   : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ week  : int  40 38 40 35 36 37 41 40 37 38 ...
 $ weight: int  2968 2795 3163 2925 2625 2847 3292 3473 2628 3176 ...
```

```
 anc1 <- lm(weight ~ week * sex, data = dat)
 anova(anc1)
```

```
Analysis of Variance Table

Response: weight
          Df  Sum Sq Mean Sq F value    Pr(>F)
week       1 1013799 1013799 31.0779 1.862e-05 ***
sex        1  157304  157304  4.8221   0.04006 *
week:sex   1    6346    6346  0.1945   0.66389
Residuals 20  652425   32621
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Es fällt auf, dass sowohl die Woche einen Einfluss hat (was ja erwartet werden kann) und dass das Geschlecht einen geringen Einfluss hat. Im Gegensatz dazu existiert keine Interaktion, d.h. die oben gezeichneten Geraden verlaufen parallel. Wir können nun z.B. auch noch mit `step(anc1)` das AIC ansehen und stellen fest, dass auch dieser das Weglassen der Interaktion nahelegt. Anschließend prüfen wir den Achsenabschnitt separat, indem wir den Interaktionsterm weglassen. Hierbei ist die Reihenfolge der abhängigen Variablen entscheidend. Die Covariaten, d.h. die Variablen deren Einfluss „herausgerechnet“ werden sollen (gemeinsame Steigung in Abhängigkeit von der Woche), müssen vor der eigentlichen Erklärungsvariable (Geschlecht) stehen:

```
 anc2 <- lm(weight ~ week + sex, data = dat)
 anova(anc2)
```

```
Analysis of Variance Table

Response: weight
          Df  Sum Sq Mean Sq F value    Pr(>F)
week       1 1013799 1013799 32.3174 1.213e-05 ***
sex        1  157304  157304  5.0145   0.03609 *
```

```
Residuals 21  658771   31370
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Wir erkennen deutlich, welche australischen Babies aus dem Testdatensatz bei ihrer Geburt das höhere Gewicht hatten.

## 8.9 Weitere Hinweise

Die vorangegangenen Ausführungen sollen einen Einstieg bieten und ermutigen. Trotzdem ist weiteres Selbststudium und Diskussion der eigenen Ergebnisse mit „Statistik-Experten“ oder Kollegen immer wichtig.

- Das Grundprinzip der Unabhängigkeit bei der Gewinnung der Stichproben ist am wichtigsten, wird aber auch am häufigsten verletzt, Stichwort Pseudoreplikate. Hier hilft am besten diskutieren des Designs in der Arbeitsgruppe. Manchmal können auch Pseudoreplikate nützlich sein, wenn man wenigstens ein Mindestmaß an echten Replikaten hat.
- Ein gutes Design hilft, Parallelen einzusparen. Unter bestimmten Bedingungen ist es sogar möglich (und manchmal sogar besser), scheinbar „ohne Parallelen“ auszukommen, z.B. bei einem Gradientendesign.
- Bei Designs mit mehreren Faktoren (mehrfache Klassifikation) gelten die nach der oben vorgestellten Methode erhaltenen ANOVA-Tabellen nur für balancierte Designs, also bei gleicher Stichprobenzahl für alle Faktorkombinationen. Bei unbalancierten Designs muss man entweder paarweise Modellvergleiche bzw. den AIC benutzen (s.o.), oder man benutzt sogenannte Typ II-Tests. Diese sind mit Hilfe der Funktion `Anova` aus dem Paket **car** auch in R verfügbar. Vor den Typ-III-Tests wird ausdrücklich gewarnt (VENABLES, 1998).

# 9 Etwas Zeitreihenanalyse

Die Analyse von Zeitreihendaten wirft eine Reihe spezieller Probleme auf, zu deren Behandlung ebenfalls spezifische Ansätze existieren. Das folgende Kapitel demonstriert das grundsätzliche Vorgehen für ausgewählte Ansätze. Zur näheren Vertiefung empfiehlt sich ein Blick in das sehr gut verständliche Kapitel 6 aus KLEIBER and ZEILEIS (2008) in ein spezifisches Lehrbuch der Zeitreihenanalyse, z.B. SHUMWAY and STOFFER (2006).

Der Konvention verschiedener Lehrbücher zur Zeitreihenanalyse entsprechend bezeichnen wir im Folgenden die Variablen nicht mit $x$ und $y$ sondern die unabhängige Variable mit $t$ und die abhängigen Variablen mit $x$.

## 9.1 Stationarität

Die Stationarität von Zeitreihen ist einer der zentralen Begriffe der Zeitreihenanalyse. Ein stochastischer Prozess $(x_t)$ heißt stark stationär, wenn die Verteilung von $(x_{s+t})$ nicht vom Index $s$ abhängt. Unter schwacher Stationarität versteht man hingegen die Bedingung, dass wenigstens die Momente erster und zweiter Ordnung, also Mittelwert, Varianz und Covarianz über die Zeit konstant sind.

Zur Verdeutlichung vergleichen wir die folgenden beiden Zeitreihen:

$$x_t = \beta_0 + \beta_1 t + u_t \tag{9.1}$$

$$x_t = x_{t-1} + c + u_t \tag{9.2}$$

Hierbei ist $t$ die Zeit, $\beta_0, \beta_1, c$ sind Konstanten und $u_t$ ist ein Zufallsprozess (sogenanntes *white noise*). Man erkennt, dass die Zeitreihe nach Gleichung 9.1 einem linearen Regressionsmodell ähnelt, dagegen entspricht die Zeitreihe nach Gleichung 9.2 einerm *random walk* mit einer Driftkonstante $c$.

#### Zwei Beispieldatensätze

Zur Veranschaulichung generieren wir uns zwei entsprechende Zeitreihen:

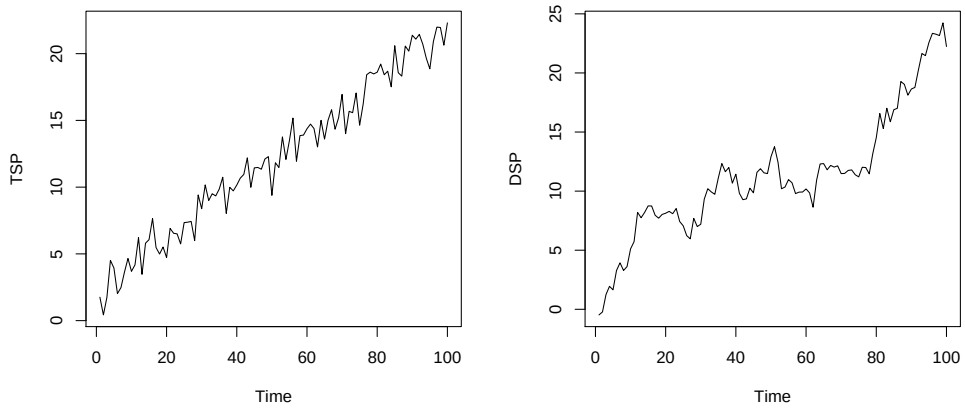
```
set.seed(1237) # für reproduzierbare Zeitreihen im Skript
time <- 1:100
## lineares Regressionsmodell
TSP <- 2 + 0.2 * time + rnorm(time)
## random walk
DSP <- numeric(length(time))
DSP[1] <- rnorm(1)
for (tt in time[-1]) DSP[tt] <- DSP[tt-1] + 0.2 + rnorm(1)
```

In R existieren spezielle Klassen für Zeitreihendaten, die wichtigsten sind `ts` für äquidistante Daten und `zoo` für nicht-äquidistante Daten. Zur Konvertierung eines Vektors in eine äquidistante Zeitreihe benutzen wir die Funktion `ts`:

```
TSP <- ts(TSP)
DSP <- ts(DSP)
```

Hierbei werden nur die $x$-Werte der Zeitreihe selbst abgespeichert. Die Zeit selbst ist als Anfang, Ende und Frequenz enthalten, sie kann mit Hilfe der Funktion `time()` wieder extrahiert werden. Eine weitere sehr nützliche Funktion ist `tsp()` (*time series properties*).

```
par(mfrow=c(1,2))
plot(TSP)
plot(DSP)
```




Man erkennt leicht, dass beide Zeitreihen offensichtlich einen Trend besitzen, nur wie prüft man ob dieser Trend auch signifikant ist? Die naheliegendste Methode wäre sicher eine lineare Regression von $x_t$ gegen die Zeit, aber ist das korrekt?

### Ein Simulationsexperiment

In einem Simulationsexperiment soll auf Signifikanz eines linearen Trends für die Zeitreihen nach Gleichung 9.1 und 9.2 geprüft werden. Zur Vereinfachung definieren wir uns dazu zwei Funktionen zur Erzeugung von Zeitreihen vom Typ „TSP“ und „DSP“ mit nutzerspezifisch einstellbaren Parametern $\beta_0, \beta_1$ und $c$:

```
genTSP <- function(time, beta0, beta1)
  as.ts(beta0 + beta1 * time + rnorm(time))
```

sowie:

```
genDSP <- function(time, c) {
  DSP <- numeric(length(time))
  DSP[1] <- rnorm(1)
  for (tt in time[-1]) DSP[tt] <- DSP[tt-1] + c + rnorm(1)
  as.ts(DSP)
}
```

Nun testen wir die Anzahl der signifikanten F-Tests für die lineare Regression durch Simulation für die beiden Typen von Zeitreihen. Hierbei wird der Trend auf Null gesetzt, d.h. der Rückgabewert der Funktion `countSignif(a)` zählt die falsch positiven Ergebnisse innerhalb einer Simulationsschleife

```
count.signif <- function(N, time, FUN, ...) {
  a <- 0
  for (i in 1:N) {
    x <- FUN(time, ...)
    m <- summary(lm(x  ~ time(x)))
    f <- m$fstatistic
    p.value <- pf(f[1], f[2], f[3], lower=FALSE)
    # cat("p.value", p.value, "\n")
    if (p.value < 0.05) a <- a + 1
  }
  a
}
```

Die obige Funktion erscheint möglicherweise für manchen Leser dieses Skriptes als zu trickreich. Letztlich sind die Feinheiten (Übergabe einer Funktion `FUN`, Durchreichen optionaler Argumente mit `...` oder Ermittlung des p-Wertes über die Verteilungsfunktion der F-Verteilung `pf`, siehe `https://stat.ethz.ch/pipermail/r-help/2009-April/194121.html`) nicht so sehr entscheidend.

Wichtig ist was die Funktion tut, sie simuliert viele (z.B. 1000) Zeitreihen, mit Hilfe einer gegebenen Funktion `FUN` und zählt wie oft ein signifikantes Ergebnis mit $p < 0.05$ gefunden wird. Für die Funktion `genTSP` beträgt der Anteil der falsch positiven etwa 5%:

```
Nruns <- 100 # besser 1000 !!!
count.signif(N=Nruns, time=time, FUN=genTSP, beta0=0, beta1=0) / Nruns
```

```
[1] 0.05
```

Für den Prozess `genDSP` ist dieser Anteil viel größer als die erwarteten 5%:

```
count.signif(N=Nruns, time=time, FUN=genDSP, c=0) / Nruns
```

```
[1] 0.91
```

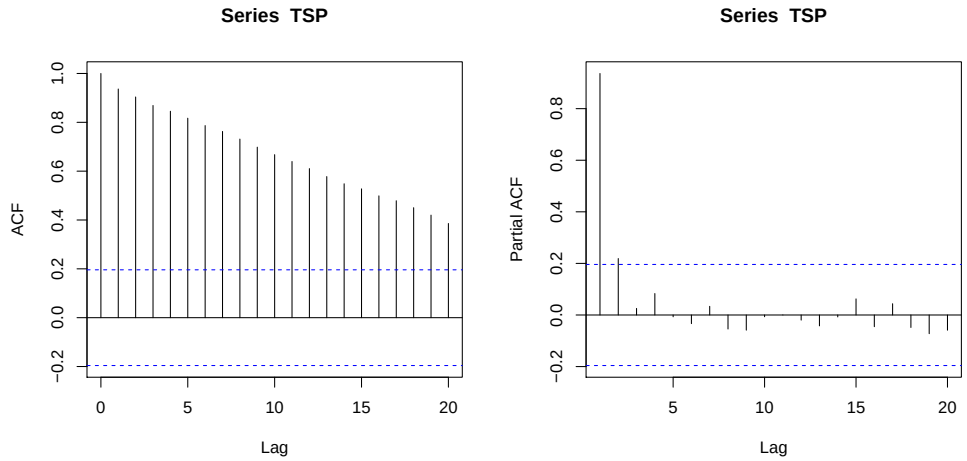
Man findet hier also zu häufig einen scheinbaren Trend, obwohl keiner vorhanden ist. Man nennt dieses Phänomen „spurious regression“. Die Ursache dafür ist, dass es sich nur bei TSP um einen Prozess mit deterministischem Trend handelt (trendstationärer Prozess). Dagegen ist DSP ein sogenannter differenzenstationärer Prozess, der nicht durch Subtraktion eines (hier linearen) Trends, sondern durch Differenzenbildung stationär gemacht werden muss.

## 9.2 Autokorrelation

Unter Autokorrelation (`acf`) versteht man eine Korrelation einer Zeitreihe mit einer zeitverschobenen Version der selben Zeitreihe. Hierbei varriiert man die Verschiebung (Lag genannt) und stellt das Ergebnis tabellarisch oder grafisch dar (Autokorrelogramm). Die Autokorrelation bei einem $lag = 0$ ist Eins, die anderen Werte geben an, wie sehr ein bestimmter Wert zum Zeitpunkt $x_t$ von seinen Vorgängern ( $x_{t-1}, x_{t-2}, ...$

abhängt. Die Abhängigkeit kann allerdings indirekt sein, d.h. $x_t$ hängt nur deshalb auch von $x_{t-2}$ $x_{t-1}$ seinerseits von $x_{t-2}$ abhängig ist. Möchte man nur die direkte Abhängigkeit ohne den indirekten Einfluss darstellen, verwendet man die partielle Autokorrelation (`pacf`).

Für den Datensatz `TSP` erkennt man im Autokorrelogramm eine abnehmende serielle Abhängigkeit der Beobachtungen, die partielle Autokorrelationsfunktion zeigt jedoch, dass nur direkt aufeinanderfolgender Werte ($lag = 1$) signifikant korreliert sind ist:

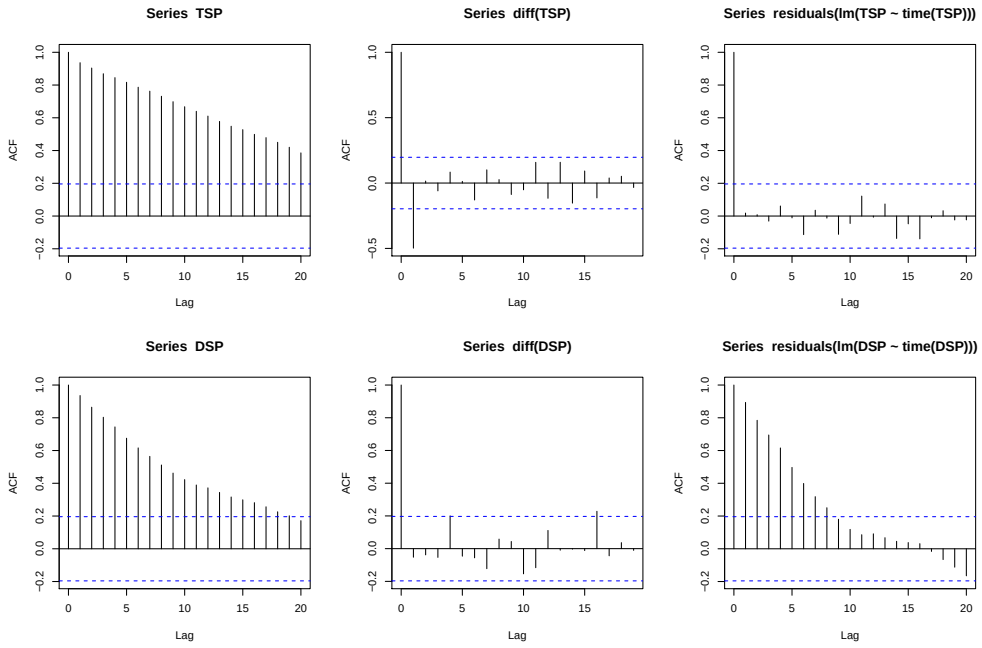



Mit Hlfe einer Kreuzkorrelation (`ccf`) zwischen zwei Variablen kann man deren deren wechselseitige Abhängigkeit beschreiben, wobei der Lag hier positiv oder negativ sein kann. So bestimmt z.B. die Globalstrahlung die Wassertemperatur in einem See und nicht umgekehrt.

Mit Hilfe typischer Muster der Autokorrelationsfunktion ist es möglich, unterschiedliche Typen von Zeitreihen voneinander zu unterscheiden, z.B. die Zeitreihen vom Typ TSP und DSP nach Gleichung 9.1 bzw. 9.2.

Die Autokorrelogramme der Originalzeitreihen (links) sehen noch vergleichsweise ähnlich aus. Bei den differenzierten Zeitreihen (Mitte) bzw. den durch Subtraktion der Regressionsgerade erhaltenen Reihen sicht man jedoch deutliche Unterschiede:

```
par(mfrow=c(3,3))
acf(TSP)
acf(diff(TSP))
acf(residuals(lm(TSP~time(TSP))))
acf(DSP)
acf(diff(DSP))
acf(residuals(lm(DSP~time(DSP))))
```




Nach dem Differenzieren der Zeitreihe „TSP“ erkennt man deutlich eine negative Korrelation bei $lag = 2$ (Abb. Mitte), nach Subtraktion der Regressionsgerade erhält man für die Residuen erwartungsgemäß nur noch weißes Rauschen (Rechts). Beim Beispiel „DSP“ ist es genau anders. Hier erhält man nach dem Differenzieren weißes Rauschen (Mitte), die „trendbereinigte“ Zeitreihe zeigt jedoch eine starke Autokorrelation (Rechts).

## 9.3 Einheitswurzeltests (unit-root test)

Um festzustellen, ob eine Zeitreihe vom Typ DSP (differenzstationärer Prozess) ist, werden sogenannte Einheitswurzeltests verwendet, auf die dahinter stehende mathematische Theorie kann hier nicht weiter eingegangen werden. In der Praxis wird oft der ADF-Test (der augmented Dickey-Fuller-Test) angewendet. Er ist in R im Paket **tseries** enthalten:

```
library(tseries)
adf.test(TSP)

        Augmented Dickey-Fuller Test

data:  TSP
```

```
Dickey-Fuller = -4.0145, Lag order = 4, p-value = 0.01134
alternative hypothesis: stationary
```

```
 adf.test(DSP)
```

```
        Augmented Dickey-Fuller Test

data:  DSP
Dickey-Fuller = -2.4355, Lag order = 4, p-value = 0.3962
alternative hypothesis: stationary
```

Man erkennt dass die Serie TSP durch Subtraktion eines linearen Trends (das macht der Test automatisch) stationär gemacht werden.

Bei der DSP-Serie kann dagegen die Nullhypothese einer Einheitswurzel nicht abgelehnt werden, die Zeitreihe ist also instationär. Nach Differenzieren spricht nach diesem Test nichts mehr gegen eine Stationarität:

```
 adf.test(diff(DSP))
```

```
        Augmented Dickey-Fuller Test

data:  diff(DSP)
Dickey-Fuller = -3.9902, Lag order = 4, p-value = 0.01261
alternative hypothesis: stationary
```

Ein direkter Test auf Stationarität, der KPSS-Test (Kwiatkowski-Phillips-Schmidt-Shin Test) testet direkt auf Stationarität oder Trendstationarität:

```
 kpss.test(TSP)                # instationär
```

```
        KPSS Test for Level Stationarity

data:  TSP
KPSS Level = 2.0842, Truncation lag parameter = 4, p-value = 0.01
```

```
 kpss.test(TSP, null="Trend")  # nach Trendbereinigung stationär
```

```
        KPSS Test for Trend Stationarity

data:  TSP
KPSS Trend = 0.068349, Truncation lag parameter = 4, p-value = 0.1
```

```
 kpss.test(DSP)                # instationär
```

```
        KPSS Test for Level Stationarity

data:  DSP
KPSS Level = 1.6951, Truncation lag parameter = 4, p-value = 0.01
```

```
 kpss.test(DSP, null="Trend")  # immer noch instationär
```

```
        KPSS Test for Trend Stationarity

data:  DSP
KPSS Trend = 0.22627, Truncation lag parameter = 4, p-value = 0.01
```

#### Trend Tests

Trendtests mit Hilfe der „normalen“ Korrelations- und Regressionsanalyse sind nur für trendstationäre Zeitreihen, nicht aber für differenzstationäre Zeitreihen möglich, da in letzteren die Residuen autokorreliert sind. Das gilt auch für den in den Umweltwissenschaften oft angewendeten Mann-Kendall-Test, der in seiner Standardformulierung auch nur für trendstationäre Zeitreihen anwendbar ist:

```
 library("Kendall")
 MannKendall(TSP)

tau = 0.894, 2-sided pvalue =< 2.22e-16

 MannKendall(DSP)

tau = 0.755, 2-sided pvalue =< 2.22e-16
```

## 9.4 Zerlegung in Mittelwert, Trend und saisonale Komponente

Die traditionelle Herangehensweise der Zeitreihenanalyse basiert darauf, eine Zeitreihe in unterschiedliche Komponenten zu zerlegen (klassisches Komponentenmodell), speziell in die Komponenten:

1. Trendkomponente,
2. saisonale oder zyklische Komponente und
3. zufällige Komponente.

Viele der Zerlegungsverfahren erfordern Normalverteilung der Residuen. Ist das nicht der Fall oder wenn sich z.B. die Varianz einer Zeitreihe proportional zu einem Trend verändert, kann eventuell eine Transformation helfen. Für Biomassedaten ist wegen der in vielen Fällen vorherrschenden linksgipfligen Verteilung oft eine logarithmische Transformation hilfreich.

### 9.4.1 Glättungsverfahren

Die Ermittlung bzw. Elimination eines Trends kann durch Anpassung von Kurven oder durch sogenannte Filter vorgenommen werden. So kann man z.B. einen linearer Trend durch eine lineare Regression gegen die Zeit ermitteltn. Die Residuen entsprechen dann der trendbereinigten Zeitreihe. Das funktioniert nicht nur bei trendstationären sondern im Prinzip auch bei differenzstationären Zeitreihen. Nur sind dann, wie oben ausgeführt, die normalerweise für de lineare Regression verwendeten Signifikanztests wegen der Autokorrelation der Residuen nicht anwendbar.

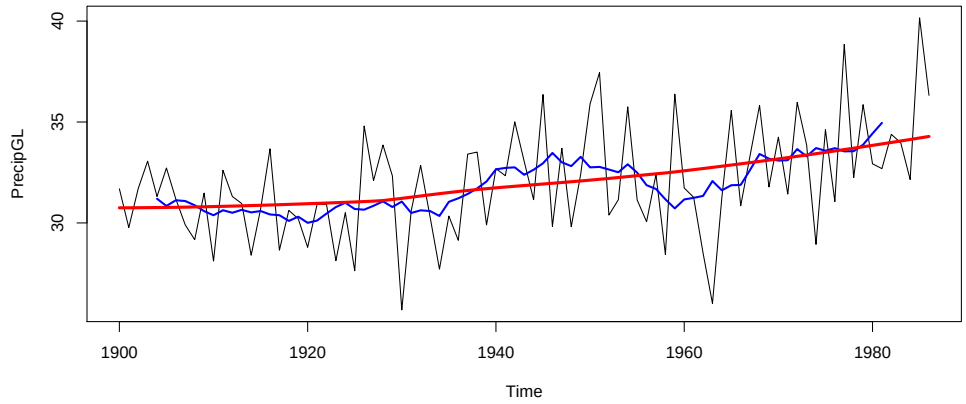
Alternativ können Trends durch Anwendung von gleitenden Mittelwerten (lineare Filter) oder durch exponentielles Glätten aufgedeckt werden. Auch Differenzenbildung (s.o.) führt zur Elimination von Trends. Zur Veranschaulichung eines linearen Filters benutzen wir einen Datensatz über den jährlichen Niederschlag im

Gebiet der Great Lakes von 1900-1986, dieser ist im Paket **Kendall** (McLeod, 2009) enthalten. Zum Vergleich ist eine weitere Möglichkeit gezeigt, der in der modernen Datenanalyse sehr beliebte LOWESS-Filter (Cleveland, 1981):

```
library(Kendall)
data(PrecipGL)
tsp(PrecipGL)
```

```
[1] 1900 1986    1
```

```
plot(PrecipGL)
kernel <- rep(1, 10)  # Rechteck-Filter; Variieren Sie die Bandbreite!
lines(filter(PrecipGL, kernel/sum(kernel)), lwd=2, col="blue")
lines(lowess(time(PrecipGL),PrecipGL),lwd=3, col=2)
```




Eine auf diese Weise trendbereinigte Zeitreihe erhält man dann durch Subtraktion des Trends, z.B.:

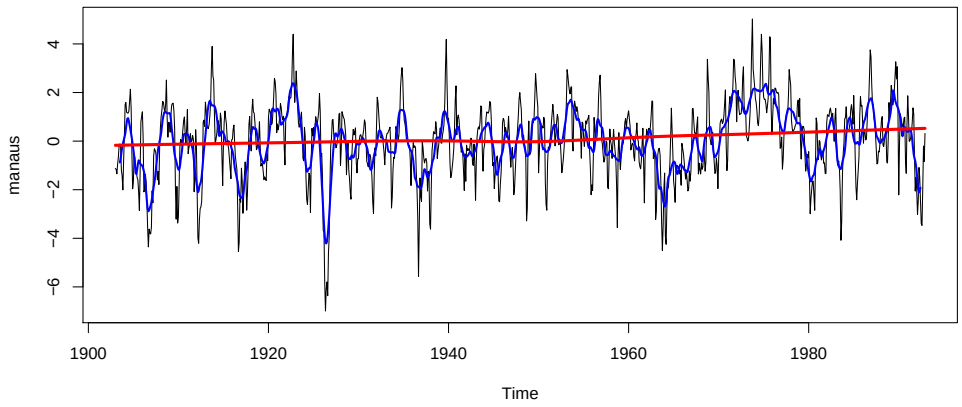
```
smooth <- filter(PrecipGL, kernel/sum(kernel))
## oder
# smooth <- lowess(time(PrecipGL), PrecipGL)$y
res <- PrecipGL - smooth
```

Für eine saisonale Zeitreihe mit Monatswerten empfehlen Kleiber and Zeileis (2008) einen Filter mit 13 Koeffizienten. Der im folgenden Beispiel verwendete Datensatz beschreibt den Wasserspiegel des Rio Negro bei Manaus 18 km vor dem Zusammenfluss mit dem Amazonas. Der Datensatz (Sternberg, 1987, 1995) findet sich im Paket **boot** (Canty and Ripley, 2009):

```
library(boot)
data(manaus)
tsp(manaus)
```

```
[1] 1903.000 1992.917   12.000
```

```
plot(manaus)
lines(filter(manaus, c(0.5, rep(1, 11), 0.5)/12), lwd=2, col="blue")
lines(lowess(time(manaus),manaus),lwd=3, col=2)
```




Die Stärke der Glättung kann für den linearen Filter und selbstverständlich auch für den LOWESS-Glätter variiert werden. Ergänzend zu LOWESS existiert in R noch ein verbesserter Algorithmus (LOESS, siehe auch Abschnitt 11.3). Darüber hinaus exitieren Verfahren, die versuchen eine optimale Glättung automatisch zu erreichen (z.B. durch GCV, *generalized cross validation*). Auch die in vielen Disziplinen beliebten GAM (*generalized addidive models*) gehören in die Klasse der Glättungsmodelle. Einen ausgezeichneten Überblick hierzu gibt WOOD (2006), weitere Tutorials finden sich auf der Homepage[1] des Autors.

### 9.4.2 Automatische Zeitreihendekomposition

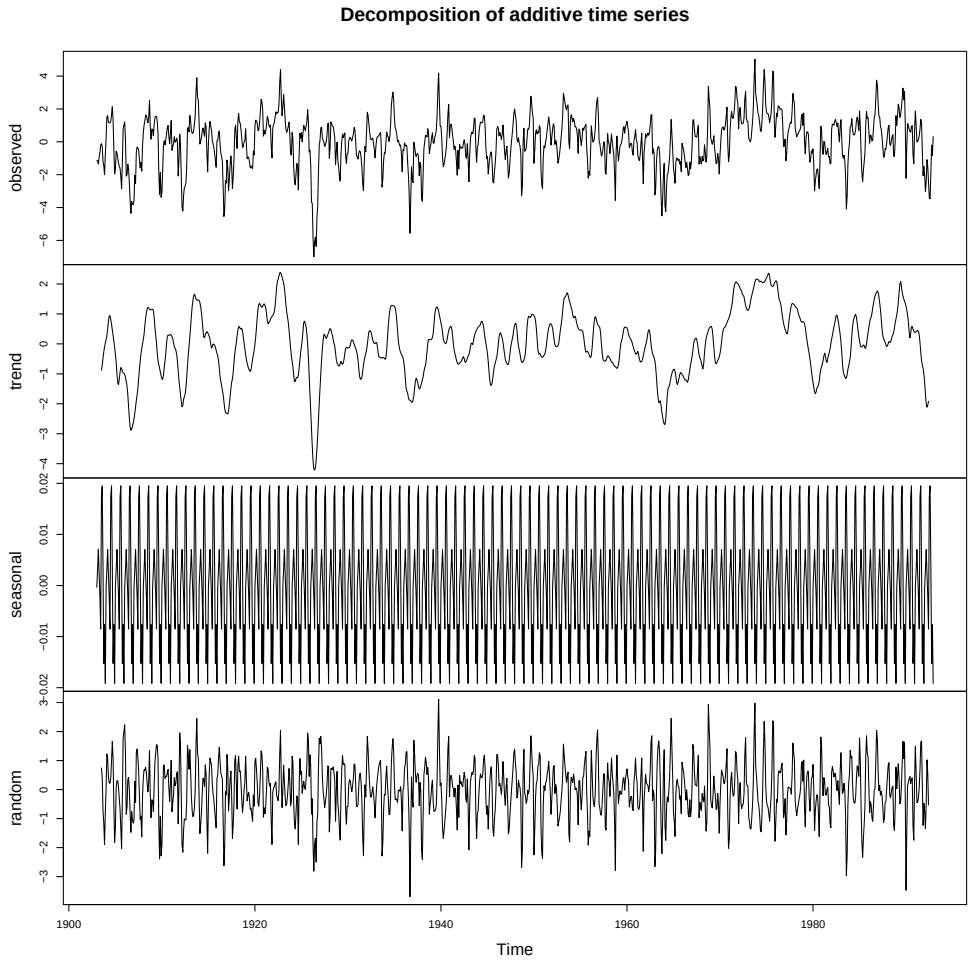
Die Funktion `decompose` implementiert den klassischen Ansatz der Zeitreihendekomposition mit Hilfe von einfachen symmetrischen Gleitmittelfiltern:

```
manaus_dec <- decompose(manaus)
str(manaus_dec)
```

```
List of 6
 $ x       : Time-Series [1:1080] from 1903 to 1993: -1.124 -1.164 -1.349 -0.945 
 $ seasonal: Time-Series [1:1080] from 1903 to 1993: -0.00036 0.00312 0.00704 0.0
 $ trend   : Time-Series [1:1080] from 1903 to 1993: NA NA NA NA NA ...
 $ random  : Time-Series [1:1080] from 1903 to 1993: NA NA NA NA NA ...
 $ figure  : num [1:12] -0.00036 0.00312 0.00704 0.00228 -0.00213 ...
 $ type    : chr "additive"
 - attr(*, "class")= chr "decomposed.ts"
```

```
plot(manaus_dec)
```

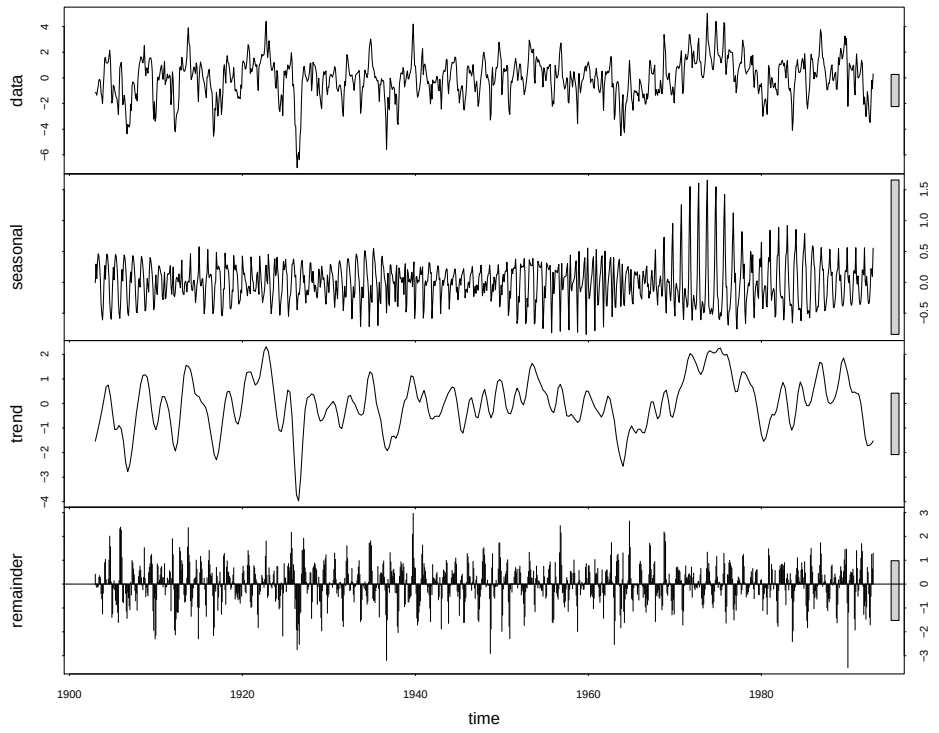
[1] http://www.maths.bath.ac.uk/~sw283/




In der vorliegenden Variante hat die saisonale Komponente einen additiven Character aber es wäre auch eine multiplikative saisonale Komponente möglich (`type="multiplicative"`).

Die Funktion `stl`, seasonal time series decomposition (CLEVELAND *et al.*, 1990) nutzt einen LOESS-Filter:

```
manaus_stl <- stl(manaus, s.window=13)
plot(manaus_stl)
```




### 9.4.3 Periodogrammanalyse

In der Natur treten häufig periodische Phänomene auf, z.B. der Jahreszyklus von Globalstrahlung und Temperatur, der Entwicklungszyklus der Vegetation oder auch die die Beinschlagbewegung eines Wasserflohs (*Daphnia*). An dieser Stelle soll pragmatisch die Anwendung der harmonischen Analyse (Fourieranalyse) zur Approximation periodischer Datenreihen vorgestellt werden.

Prinzipiell läßt sich jede Zeitreihe durch eine Summe aus Sinus- und Kosinustermen mit unterschiedlicher Periode (Fourierreihe) darstellen:

$$x_t = a_0 + \sum_{p=1}^{N/2-1} \left( a_p \cos(2\pi pt/N) + b_p \sin(2\pi pt/N) \right) + a_{N/2} \cos(\pi t), \qquad t = 1 \ldots N \tag{9.3}$$

Hierbei ist $a_0$ der Mittelwert der Zeitreihe, $a_p, b_p$ sind die Koeffizienten der Fourierreihe und $N$ ist die Länge der Zeitreihe oder die Periodendauer. Ähnlich wie bei der linearen Regression, lassen sich die Koeffizienten mit Hilfe eines linearen Gleichungssystems analytisch ermitteln, d.h. es sind lediglich entsprechende Summen zu bilden:

$$a_0 = \bar{x} \tag{9.4}$$

$$a_{N/2} = \sum_{t=1}^{N} (-1)^t x_t / N \tag{9.5}$$

$$a_p = 2\frac{\sum_{t=1}^{N} x_t \cos(2\pi pt/N)}{N} \tag{9.6}$$

$$b_p = 2\frac{\sum_{t=1}^{N} x_t \sin(2\pi pt/N)}{N} \tag{9.7}$$

Darüberhinaus existiert eine besonders leistungsfähige Methode, die „schnelle Fouriertransformation“ (FFT, *fast fourier transform*), mit der die Koeffizienten $a_0, a_p, b_p$ sehr effizient mit Hilfe von komplexen Zahlen ermittelt werden können.

Die Gleichung 9.3 kann auch in eine Form mit nur einem Kosinusterm umgewandelt werden:

$$x_t = a_0 + \sum (R_p \cdot \cos(2\pi pt/N + \Phi_p) \tag{9.8}$$

mit:

$$R_p = \sqrt{a_p^2 + b_p^2} \tag{9.9}$$

$$\Phi_p = \arctan(-b_p/a_p) \tag{9.10}$$

Dies hat den Vorteil, dass die Amplituden $R_i$ und die Phasenverschiebungen $\Phi_i$ der einzelnen Frequenzen ($2\pi/N$, $4\pi/N$, …, $\pi$) direkt ablesbar sind. Trägt man $R_p^2/2$, den Varianzanteil des $p$-ten harmonischen Terms, gegen die Frequenz $\omega_p = 2\pi p/N$ grafisch auf, ergibt sich das Periodogramm (welches unter Umständen noch geglättet werden muss).

Da die harmonische Analyse den untersuchten Prozeß in die Varianzanteile einzelner Frequenzen zerlegt, ist es auch möglich, eine Zeitreihe auf der Basis ausgewählter Frequenzanteile zu synthetisieren.

### 9.4.4 Implementierung in R

Zur Vereinfachung der Analyse ist es sinnvoll, zwei Hilfsfunktionen zu erstellen. Eine Funktion dient zur Ermittlung der Koeffizienten $a_p$ und $b_p$ (Gl. 9.4 bis 9.7), die zweite Funktion zur Synthese der harmonischen Funktion entsprechend Gleichung 9.3. Im folgenden sind mehrere Möglichkeiten angegeben, für die Analyse der klassische Weg und eine Funktion die die in R enthaltene FFT benutzt. Auch für die Synthese existieren verschiedene Wege, z.B. der klassische Weg über die Gleichung 9.3 oder 9.8 oder eine inverse FFT (hier nicht dargestellt). In R bietet es sich außerdem an, auf Schleifen zu verzichten und dafür mit der Matrixmultiplikation zu arbeiten. Für die Praxis ist jedoch jeweils eine der Funktionen ausreichend, z.B. die Analyse über die FFT (`harmonic.fft`) und die Synthese über den klassischen Weg mit Matrixmultiplikation (`synth.harmonic`):

```
## classic method of harmonic analysis
harmonic.classic <- function(x, pmax=length(x)/2) {
 n  <- length(x)
 t  <- 0:(n-1)
 a0 <- mean(x)
 a  <- numeric(pmax)
 b  <- numeric(pmax)
 for (p in 1:pmax) {
   k    <- 2 * pi * p * t / n
   a[p] <- sum(x * cos(k))
   b[p] <- sum(x * sin(k))
 }
 list(a0=a0, a=2*a/n, b=2*b/n)
}
## fast fourier version of harmonic analysis
harmonic.fft <- function(x) {
  n  <- length(x)
  pf <- fft(x)      # Fast Fourier Transform
  a0 <- Re(pf[1])/n # first element = mean
  pf <- pf[-1]      # drop first element
  a  <-  2*Re(pf)/n # Real part of complex
  b  <- -2*Im(pf)/n # Imaginary part
  list(a0=a0, a=a, b=b)
}
### ================================================

## synthesis of a harmonic function
## (classic method)
synth.harmonic.classic <- function(t, fpar, n, ord) {
  a <- fpar$a; b <- fpar$b; a0 <- fpar$a0
  x <- a0
  for (p in ord) {
    k <- 2 * pi * p * t/n
    x <- x + a[p] * cos(k) + b[p] * sin(k)
  }
  x
}
## synthesis of a harmonic function
## version with amplitude (R) and phase (Phi)
synth.harmonic.amplitude <- function(t, fpar, n, ord) {
  a   <- fpar$a; b <- fpar$b; a0 <- fpar$a0
  R   <- sqrt(a * a + b * b)
  Phi <- atan2(-b, a)
  x   <- a0
  for (p in ord) {
    x <- x + R[p] * cos(2 * pi * p * t/n + Phi[p])
  }
```

```
  x
}
## synthesis of a harmonic function
## classic method with matrices
synth.harmonic <- function(x, fpar, n, ord) {
  a <-  fpar$a; b <- fpar$b; a0 <- fpar$a0
  k <- (2 * pi * x/n) %*% t(ord)
  y <-    a0 + cos(k) %*% a[ord] +
               sin(k) %*% b[ord]
  y
}
```

Ein zur Testung geeigneter Datensatz kann mit Hilfe von Sinus- und Kosinusfunktionen erzeugt werden, ein normalverteilter Fehlerterm fügt ein gewisses „Rauschen“ hinzu:

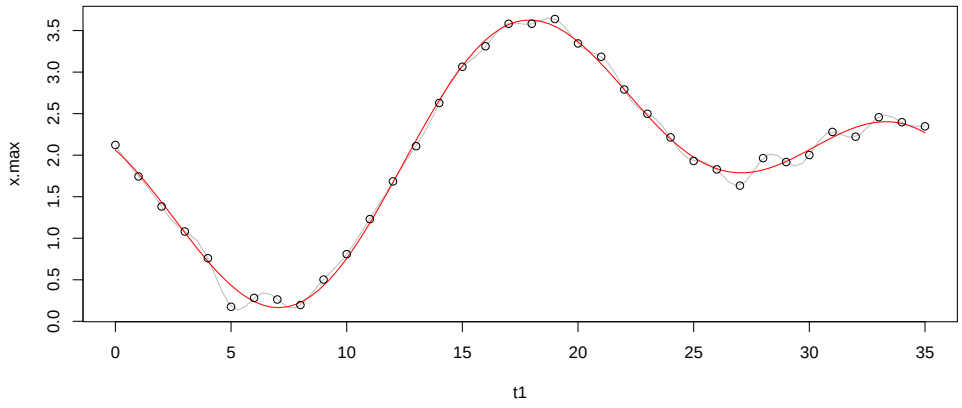
```
n <- 36
t <- 0:(n-1)
x <- 2 + sin(t*4*pi/n+2) + sin(t*2*pi/n + 4) + rnorm(n, sd=0.1)
```

Die Ermittlung der Koeffizienten erfolgt nun mit:

```
fpar <- harmonic.fft(x)
```

Danach zeigt der Aufruf von `fpar` die ermittelten Koeffizienten an. Synthetisierte Zeitreihen mit der maximalen Ordnung, bzw. mit einer hier optimalen Ordnung 2 erhalten wir mit:

```
t1<-seq(min(t), max(t), length=100)
x.max <-  synth.harmonic(t1, fpar, n, ord=1:(n/2))
x.1 <-  synth.harmonic(t1, fpar, n, ord=1:2)
plot(t1, x.max, col="gray", type="l")
lines(t1, x.1, col="red")
points(t,x)
```




Hierbei ist `t1` ein beliebig festgelegter Definitionsbereich, für den die Funktion gezeichnet werden soll, `fpar` enthält die Fourierkoeffizienten und `n` ist die Periodendauer (Anzahl der Werte des Originaldatensatzes). Mit dem Vektor `ord` werden die harmonischen Ordnungen angegeben, die zur Synthese der Funktion

verwendet werden sollen. Es müssen nicht unbedingt alle Ordnungen von 1 bis $n/2$ benutzt werden, sondern es können auch einzelne Ordnungen herausgegriffen werden.

Auf ähnliche Weise wie mit `synth.harmonic`, kann die Berechnung mit Hilfe der Koeffizienten $a_0, a_p, b_p$ und der Gleichung 9.3 auch außerhalb von R (z.B. in einer Tabellenkalkulation) erfolgen.

### 9.4.5 Übung

#### Problem

Als Antrieb ökologischer Modelle werden häufig Jahresgänge von Temperatur und Globalstrahlung benötigt. Eine einfache Möglichkeit besteht darin, diese durch eine harmonische Funktion mit einer niedrigen Ordnung darzustellen. Finden Sie eine harmonische Funktion zur Beschreibung des mittleren Jahresganges der Globalstrahlung in $\mathrm{J\,cm^{-2}d^{-1}}$. Benutzen Sie hierzu die Daten von 1981 bis 1990 der Station Wahnsdorf bei Dresden (Quelle: Täglicher Wetterbericht des Meteorologischen Dienstes der DDR oder Online-Zugriff über das World Radiation Data Center[2].

#### Lösung

Zunächst müssen die Hilfsfunktionen für die harmonische Analyse (s.o.) eingegeben oder eingelesen werden. Anschließend werden die Daten aus einer Textdatei eingelesen, wobei die erste Zeile die Variablennamen enthält (`header=TRUE`). Wir lassen uns die Variablennamen anzeigen, kopieren die Spalte `igl` in die Variable `x` und erzeugen uns, um die nicht ganz einfache Datumsrechnung zu vermeiden, eine neue Zeitvariable `t`.

```
dat <- read.table("http://www.simecol.de/data/igl8190_dd.dat",
  header=TRUE)
names(dat)
```

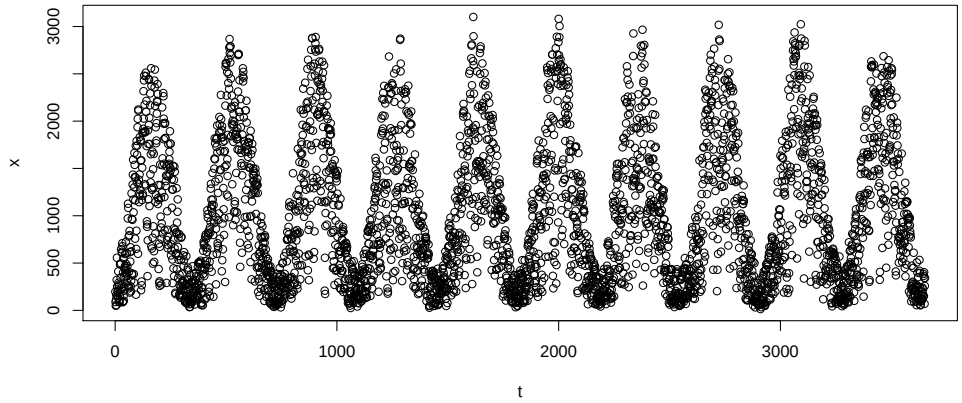
```
[1] "date"         "igl"          "interpoliert"
```

```
x   <- dat$igl
t   <- 1:length(x)
```

Anschließend stellen wir die Daten grafisch als Zeitreihe über mehrere Jahre dar:

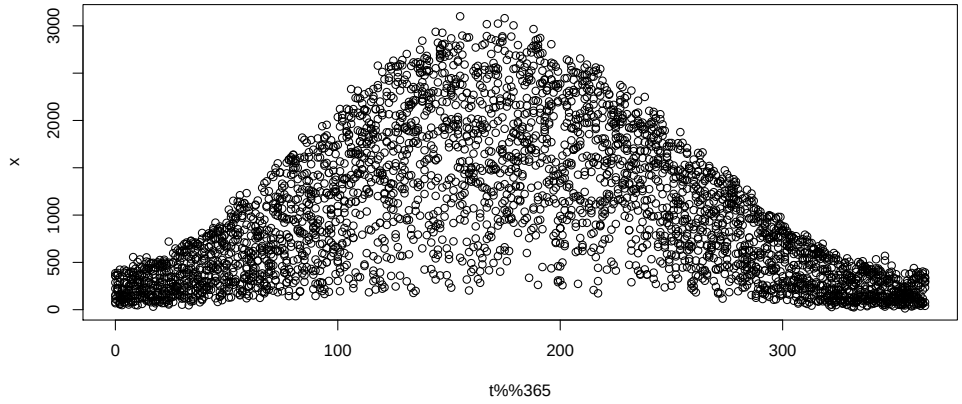
```
plot(t, x)
```

[2] `http://wrdc-mgo.nrel.gov/`



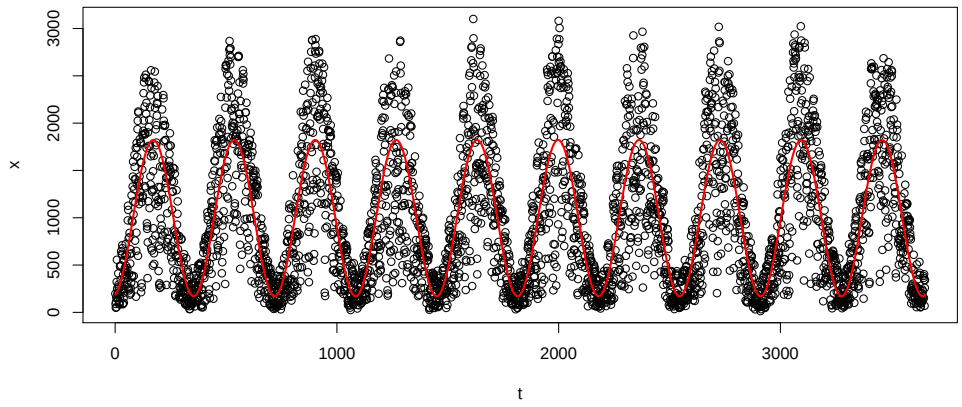

Alternativ können auch alle Jahre übereinander geplottet werden, wobei der Operator `%%` der Modulo-Operator (Rest einer ganzzahligen Division) ist. Das bedeutet in diesem Fall, dass der Tag 366 wieder bei 1 geplottet wird.

```
plot(t %% 365, x)
```




Nach dieser ersten Orientierung folgt nun die eigentliche Analyse (dieses Mal über die FFT) und die grafische Darstellung des Ergebnisses:

```
fpar <- harmonic.fft(x)
plot(t, x)
lines(synth.harmonic.classic(t,fpar,length(x),ord=10), col="red", lwd=2)
```

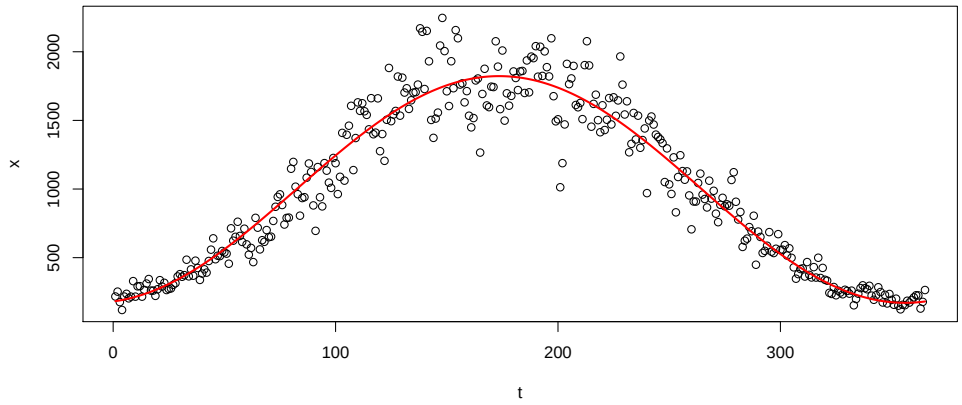



Da der verwendete Datensatz 10 Jahre enthält, stellt `ord=10` einen jährlichen Zyklus dar. Wir experimentieren nun ein wenig mit der Fourierordnung herum und stellen z.B. die Ordnungen `ord=1`, `ord=1:10`, `ord=1:100` dar.

### Lösung 2

Eine alternative Möglichkeit besteht darin, zunächst die 10jährigen Mittel für alle 365 Tage zu berechnen und anschließend direkt eine harmonische Funktion 1. Ordnung zu berechnen. Die Mittelwertberechnung kann extern in einem Tabellenkalkulationsprogramm oder mit der sehr leistungsfähigen R-Funktion `aggregate` erfolgen, die ihre Argumente als Dataframes oder Listen erwartet. Das erste Argument kennzeichnet die auszuwertenden Daten, das zweite Argument die Gruppierung und das dritte Argument die anzuwendende Funktion.

```
meanyear <- aggregate(list(x=x), list(yr=t%%365), mean)
x <- meanyear$x
t <- 1:length(x)
plot(t, x)
fpar <- harmonic.fft(x)
lines(synth.harmonic.classic(t,fpar,length(x),ord=1), col="red", lwd=2)
```




Zur weiteren Verwendung ist es nun notwendig, die ermittelte Funktion in einer geschlossenen Form aufzuschreiben, wobei sich die Koeffizienten ja bekanntlich in der Liste `fpar` befinden. Bei Angabe nur des

Jahreszyklus ($p = 1$) ergibt sich aus Gleichung 9.3:

```
cat(fpar$a0, fpar$a[1], fpar$b[1], "\n")
```

```
996.746 -815.988 126.741
```

```
plot(t,x)
x1 <- 997 - 816 * cos(2*pi*1*t/365) + 127 * sin(2*pi*1*t/365)
lines(t,x1)
```

Als mathematische Funktion geschrieben ergibt das:

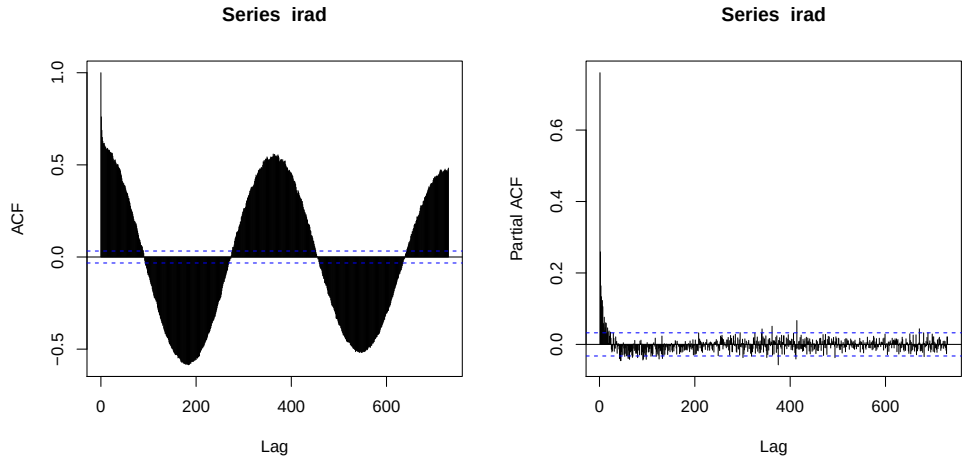
$$x = 997 - 816 \cdot \cos(2\pi \cdot t/365) + 126 \cdot \sin(2\pi \cdot t/365) \tag{9.11}$$

In unserem Beispiel ist es wegen des Jahreszyklus mehr oder weniger offensichtlich, welche Fourierordnungen benötigt werden. In der Regel ist das aber im voraus nicht bekannt und muss über eine Periodogramm- oder Frequenzanalyse ermittelt werden (siehe SCHLITTGEN and STREITBERG, 1989; BOX *et al.*, 1994).

### 9.4.6 Frequenzspektren

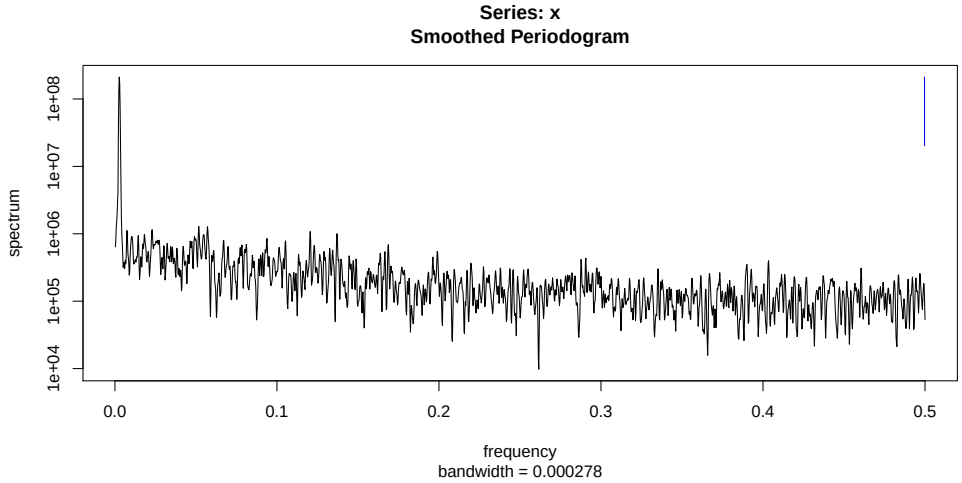
Selbstverständlich existiert in R auch hierzu eine vordefinierte Funktion. Zunächst wandeln wir die Datenreihe in ein Zeitreihenobjekt um und sehen uns die Autokorrelogramme an:

```
dat <- read.table("http://www.simecol.de/data/igl8190_dd.dat",
  header=TRUE)
x   <- dat$igl
irad <- ts(dat$igl)
par(mfrow=c(1,2))
acf(irad, lag.max=2*365)
pacf(irad, lag.max=2*365)
```




Anschließend folgt eine Spektralanalyse, der Parameter `spans` definiert das Fenster für die Glättungsfunktion (hier: modifizierter Daniell smoother):

```
sp <- spectrum(irad, spans=c(2, 2))
```

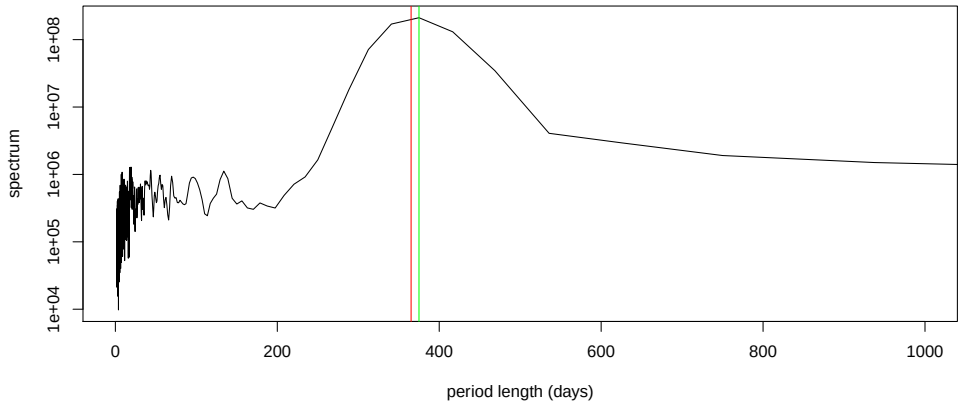



Anstelle über den Frequenzbereich lässt sich das Spektrum auch über die Periodendauer (1/freq) plotten (hier mit logarithmiescher y-Skalierung):

```
with(sp, plot(1/freq, spec, type="l", xlim=c(0,1000),
  ylab="spectrum", log="y", xlab="period length (days)"))
abline(v=365.25, col="red")
(smax <- 1/sp$freq[sp$spec == max(sp$spec)])
```

```
[1] 375
```

```
abline(v=smax, col="green")  # spectral maximum is 375 days
```




Die Linien markieren das erwartete und das empirische Maximum bei 365.25 bzw. 375 Tagen.

### 9.4.7 Weitere Aufgaben

1. Ermitteln Sie die Amplitude und die Phasenverschiebung der Globalstrahlungs-Funktion.
2. Stimmt das Ergebnis mit Ihren Erwartungen überein (Kalender).

3. Stellen Sie die Funktion in einer Form nach Gleichung 9.8 dar und prüfen Sie das Ergebnis grafisch.
4. Vergleichen Sie die Globalstrahlungsfunktion 10. Ordnung von Lösung 1 mit der Funktion 1. Ordnung von Lösung 2.
5. Stellen Sie die Funktion und die Daten mit Ihrem bevorzugten Grafik- oder Tabellenkalkulationsprogramm dar.
6. Versuchen Sie, die Epilimniontemperatur eines Gewässers mit einer harmonischen Funktion zu beschreiben (Datensatz `t_epi7.dat`). Wieviele Fourier-Ordnungen werden benötigt?

## 9.5 ARIMA-Modellierung

Im Fokus der sogenannten ARIMA-Modellierung[3] (BOX *et al.*, 1994) steht die Vorhersage. Sowohl Periodogrammanalyse als auch ARIMA-Modellierung basieren auf der Autokorrelationsfunktion. Es handelt sich lediglich um eine unterschiedliche Betrachtungsweise. Es wird versucht, Zeitreihen mit deutlichen Trends oder Zyklen zunächst in annähernd stationäre Zeitreihen zu überführen. Besonders wichtig hierbei sind gewöhnliche Differenzfilter und saisonale Differenzfilter, die auch mehrfach angewendet werden können (SCHLITTGEN and STREITBERG, 1989). In Umkehrung dieser Differentiation bei der Modellidentifikation wird bei der Vorhersage die Integration angewendet.

Ein schwieriges Problem bei der Entwicklung eines ARIMA-Modells ist die Spezifikation, d. h. die Entscheidung, welche AR und MA-Ordnungen verwendet werden sollen. Beim BOX-JENKINS-Ansatz werden hierzu die Autokorrelationsfunktion (ACF) und die partielle Autokorrelationsfunktion (PACF) herangezogen. Hierbei treten häufig charakteristische Muster auf, die einen ersten Anhaltspunkt über die Ordnung des beobachteten Prozesses geben. Weiterhin wird vorgeschlagen, mehrere alternative Modelle anzupassen und diese an Hand der mittleren quadratischen Fehler sowie der Signifikanz der Parameter zu bewerten („Overfitting“, BOX *et al.*, 1994).

Hierbei ist die Autokorrelationsfunktion besonders zur Bestimmung reiner MA-Prozesse und die partielle Autokorrelationsfunktion von AR-Prozessen geeignet. Nach SCHLITTGEN and STREITBERG (1989) ist die Benutzung von ACF und PACF zur Spezifikation gemischter ARMA-Prozesse ein allerdings „delikates Problem“.

### 9.5.1 Moving-Average-Prozesse

„Ein stochastischer Prozeß $(X_t)$ heißt Moving-Average-Prozeß der Ordnung $q$, kurz MA$(q)$-Prozeß, wenn er sich in der Form:

$$X_t = \varepsilon_t - \beta_1 \varepsilon_{t-1} - \cdots - \beta_q \varepsilon_{t-q}$$

darstellen läßt. Dabei ist $(\varepsilon_t)$ ein White-Noise-Prozeß“ (SCHLITTGEN and STREITBERG, 1989). Das bedeutet, der gegenwärtige Wert $X_t$ ist ausschließlich von „Zufallsschocks“ $\varepsilon_t$ der Vergangenheit abhängig.

[3] AR: Autoregressive, I: Integrated, MA: Moving Average

### 9.5.2 Autoregressive Prozesse

„Ein stochastischer Prozeß ($X_t$) heißt Autoregressiver Prozeß der Ordnung $p$, kurz AR($p$)-Prozeß, wenn er der Beziehung:

$$X_t = \alpha_1 X_{t-1} + \cdots + \alpha_p X_{t-p} + \varepsilon_t$$

genügt. Dabei ist ($\varepsilon_t$) ein White-Noise-Prozeß“ (SCHLITTGEN and STREITBERG, 1989). Der AR-Prozeß entspricht also formal einer multiplen linearen Regression, wobei der aktuelle Wert als eine Funktion der vorangegangenen Werte aufgefaßt wird.

### 9.5.3 ARIMA-Prozesse

Ein Prozeß, der aus sich einem AR(p)- und einem MA(q)-Anteil zusammensetzt, wird als ARMA(p,q)-Prozeß bezeichnet. Wurden vorher noch eine oder mehrere Differenzenbildungen durchgeführt, spricht man von ARIMA(p,d,q)-Prozessen. Wird nicht nur die aktuelle Vorgeschichte betrachtet, sondern auch eine saisonale Verschiebung, ergeben sich ARIMA(p,d,q)(P,D,Q)-Prozesse, die auch SARIMA (seasonal ARIMA) genannt werden.

### 9.5.4 Anpassung von ARIMA-Modellen

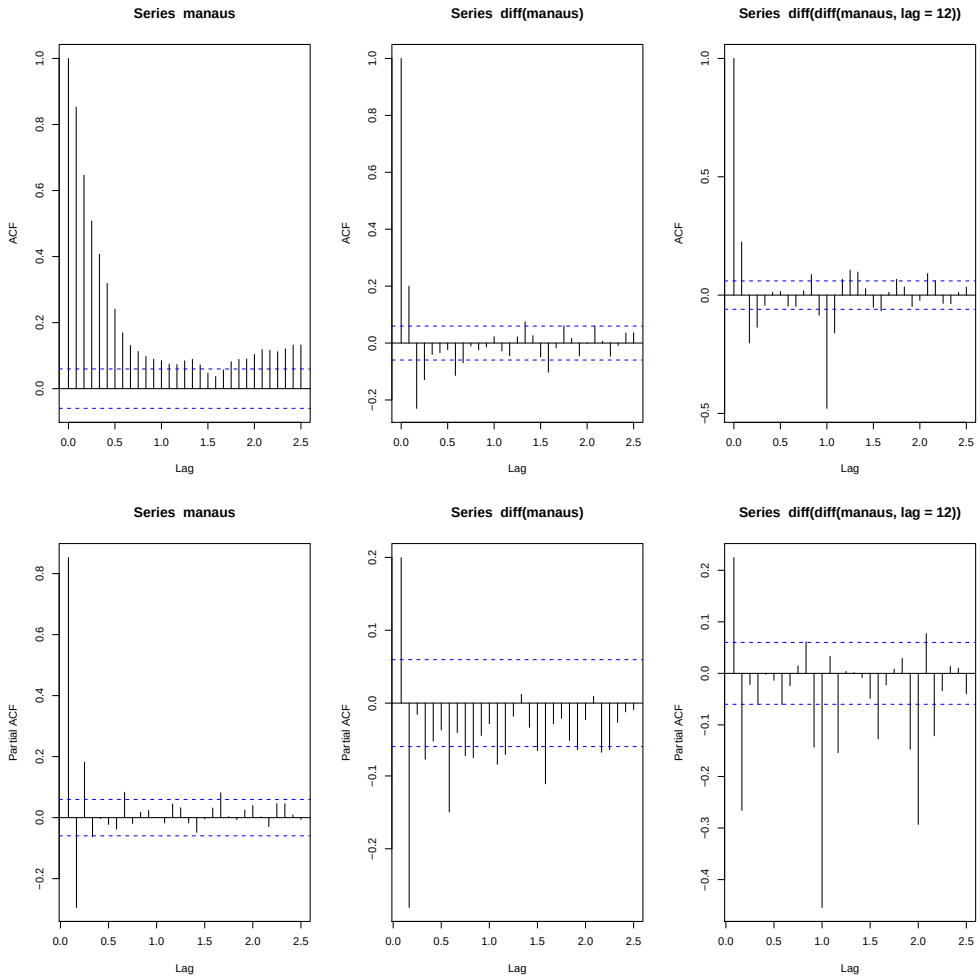
Die Ermittlung der Parameterwerte für ein vorher spezifiziertes Modell bezeichnet man als Identifikation. Hierfür stehen unterschiedliche Verfahren zur Verfügung, die auf dem Maximum-Likelihood-Kriterium beruhen. Das Hauptproblem besteht in der Auswahl (Spezifikation) des optimalen Modells. Hierbei wurden verschiedene Entscheidungskriterien propagiert, z.B. „overfitting“ und anschließdende Modellvereinfachung nach visuellen Kriterien oder eine sehr detaillierte Interpretation der Autokorrelationsdiagramme (z.B. von BOX *et al.*, 1994).

Diese Hinweise sind nach wie vor gültig, anstelle eines rein visuellen Modellvergleichs wird jedoch heute noch mehr als zuvor die AIC-basierte Modellselektion empfohlen. Bei Einsatz entsprechender Rechenleistung gelingt es sogar, die in Frage kommenden Modelle (nach acf, pacf ausgewählt) automatisiert durchzuprobieren. Wir benutzen hierzu wieder den `manaus`-Datensatz:

```
library(boot)
data(manaus)
```

Zunächst einmal schauen wir uns die Korrelogramme für die Zeitreihe und ihre Differenzen an. Bei der „doppelten“ Differenzierung wird zunächst saisonal (`lag=12`) und anschließend mit `lag=1` differenziert;

```
par(mfrow=c(2,3))
acf(manaus)
acf(diff(manaus))
acf(diff(diff(manaus, lag=12)))
pacf(manaus)
pacf(diff(manaus))
pacf(diff(diff(manaus, lag=12)))
```




Ohne auf Einzelheiten eingehen zu wollen erkennen wir dass auf jeden Fall differenziert werden muss und dass ein oder zwei AR-Parameter und wahrscheinlich auch MA-Parameter und saisonale Parameter erforderlich sind.

Das folgende kurze Skript (nach KLEIBER and ZEILEIS, 2008, verändert) fittet alle Modelle mit 0 bis 1 bzw. 2 Parametern für alle Komponenten, die Anzahl der Differenzierungen ist für die Lags 1 und saisonal 12 auf je 1 fixiert. Achtung: Die Festlegung der durchzutestenden Modelle erfordert ein wenig Fingerspitzengefühl.

```
dat <- manaus
pars <- expand.grid(ar=0:2, diff=1, ma=0:2, sar=0:1, sdiff=1, sma=0:1)
aic  <- numeric(nrow(pars))
for (i in seq(along=aic))
  aic[i] <- AIC(arima(dat, unlist(pars[i, 1:3]), unlist(pars[i, 4:6])),
    k=log(length(dat)))
ndx_best  <- which.min(aic)
(pars_best <- pars[ndx_best,])
```

```
   ar diff ma sar sdiff sma
26  1    1  2   0     1   1
```

```
## und jetzt nochmal das 'beste' (d.h. most parsimonious) Modell fitten
(m <- arima(dat, unlist(pars_best[1:3]), unlist(pars_best[4:6])))
```

```
Call:
arima(x = dat, order = unlist(pars_best[1:3]), seasonal = unlist(pars_best[4:6]))

Coefficients:
         ar1      ma1      ma2     sma1
      0.7424  -0.5647  -0.4353  -0.9906
s.e.  0.0231   0.0312   0.0308   0.0338

sigma^2 estimated as 0.5766:  log likelihood = -1248.63,  aic = 2507.27
```

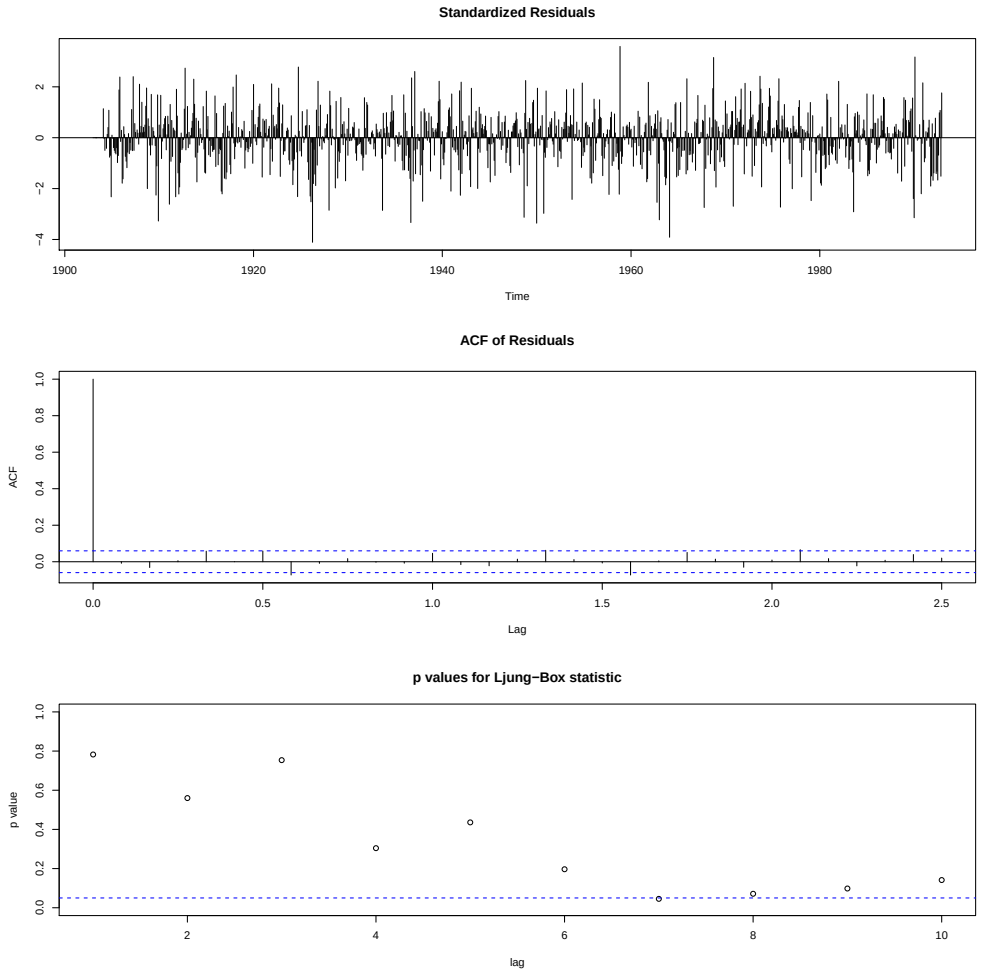
Wir erkennen, dass das beste Modell eine Modell vom Typ S*ARIMA*$(1,1,2)(0,1,1)_{12}$ ist. Mit Hilfe der angegebenen Standardfehler lässt sich auch das Signifikanzniveau ablesen.

Im Paket **forecast** (HYNDMAN and KHANDAKAR, 2008) enthält eine vollautomatische Funktion, die für das Beispiel ein ähnliches aber kein identisches Ergebnis erzielt:

```
library(forecast)
auto.arima(dat)
```

Eine Überprüfung eines gefundenen Modells erhält man mit `tsdiag`:

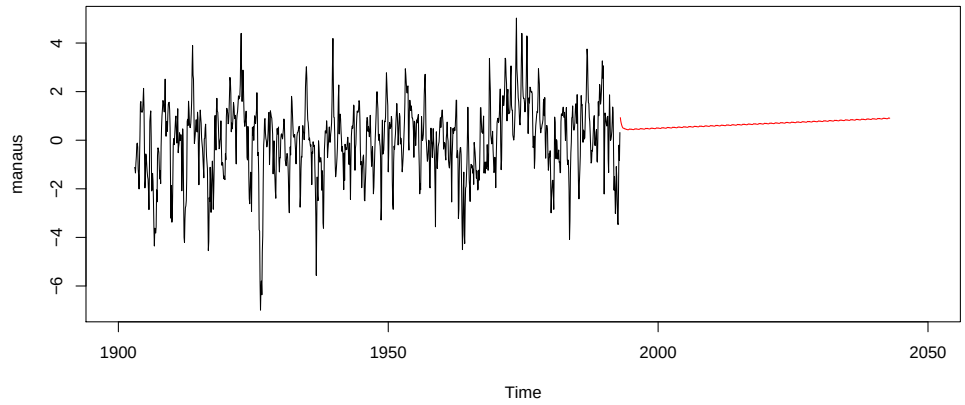
```
tsdiag(m)
```



Man erkennt, dass die Residuen kein offensichtliches Muster zeigen (also stationär erscheinen) und dass die ACF (selbstverständlich mit Ausnahme von $lag = 0$) die Signifikanzschranken nicht wesentlich überschreitet. Auch die p-Werte der Ljung-Box Statistik sind größer als 0.05, d.h. es die Residuen unterscheiden sich nicht signifikant vom Weißen Rauschen.

Der letzte Schritt ist die Anwendung des gefundenen Modells für eine Vorhersage. Hierzu kann die Funktion `predict` verwendet werden, z.B. für einen Zeitraum von 50 Jahren:

```
pr <- predict(m, n.ahead=50*12)
plot(manaus, xlim=c(1900, 2050))
lines(pr$pred, col="red")
```




Im Paket **forecast** findet man dazu noch wesentlich leistungsfähigere Funktionen und „schönere“ Grafiken.

### 9.5.5 Aufgaben

1. Verändern sie am obigen Beispiel die Anzahl der zu fittenden ARIMA-Paramater und bewerten Sie die Ergebnisse.
2. Fitten Sie ARIMA-Modelle für die Zeitreihen TSP und TSD.

## 9.6 Identifizierung von Strukturbrüchen

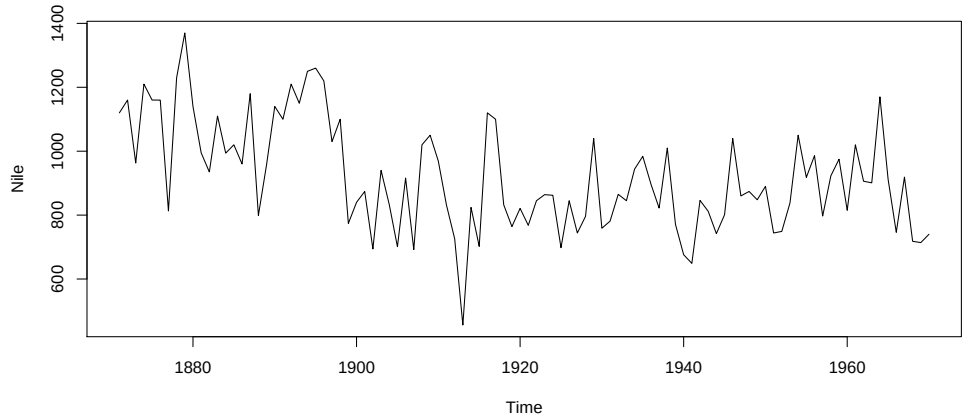
Von einem Strukturbruch in einer Zeitreihe spricht man, wenn ein oder mehrere statistische Parameter nicht über die gesamte Zeitreihe hinweg konstant sind. So kann sich z.B. ein Lageparameter, z.B. der Mittelwert, ein Trend oder auch ein anderer Verteilungsparameter wie Varianz und Covarianz ändern.

### 9.6.1 Testen auf Strukturbrüche

Das Paket **strucchange** (ZEILEIS *et al.*, 2002) implementiert eine Sammlung von Tests zur Identifikation von Strukturänderungen oder Parameterinstabilität von Zeitreihen. Es sind generell zwei Ansätze verfügbar: Fluktuationstests und F-basierte Tests. Hierbei versuchen Fluktuationstests, die strukturelle Stabilität durch kumulative oder gleitende Summen zu fassen (CUSUMs bzw. MOSUMs).

Der Nil-Datensatz (DURBIN and KOOPMAN, 2001, und die dort zitierte Literatur) enthält Messungen des jährlichen Durchflusses des Nils bei Ashwan von 1871–1970:

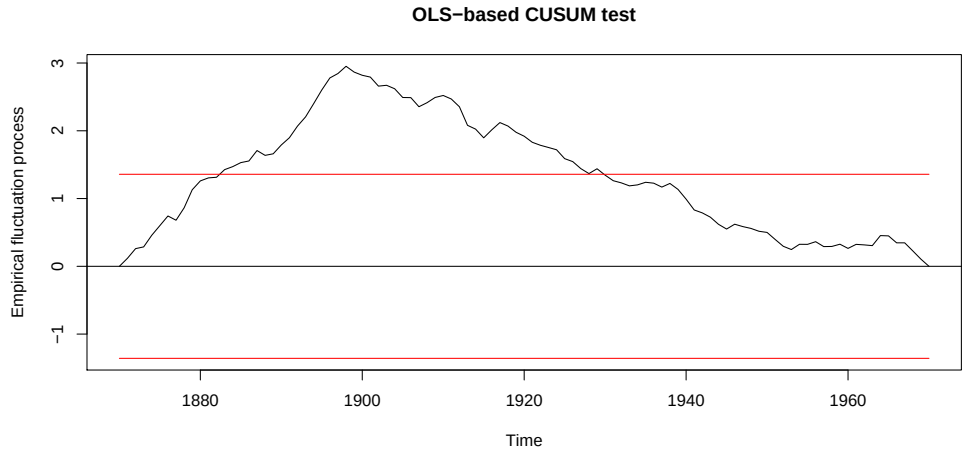
```
library(strucchange)
data("Nile")
plot(Nile)
```




Wir wollen nun testen, ob ein Strukturbruch bezogen auf den Lageparameter vorliegt, d.h. ob es Perioden mit unterschiedlichem Durchfluss existieren. Hierzu verwenden wir einen *ordinary least squares* OLS-CUSUM Test. Die Familie der OLS-CUSUM und MOSUM-Tests ist flexiblel für unterschiedliche Testprobleme einsätzbar, das folgenden Beispiel testet den allereinfachsten Fall durch Vergleich mit einem Nullmodell. Die dazu erforderlichen Funktionen sind `efp` (*empirical fluctuation model*) und `sctest` (*structural change test*):

```
ocus <- efp(Nile ~ 1, type="OLS-CUSUM")
plot(ocus)
sctest(ocus)
```

```
        OLS-based CUSUM test

data:  ocus
S0 = 2.9518, p-value = 5.409e-08
```



Anstelle einer bloßen Prüfung auf Stabilität des Mittelwertes mit einem Nullmodell ( `1`) sind auf der rechten Seite z.B. auch zeitverschobene Sigale möglich (siehe das Beispiel in KLEIBER and ZEILEIS, 2008, S. 171) und es können auch Covariate angegeben werden.

### 9.6.2 Breakpoint-Analyse

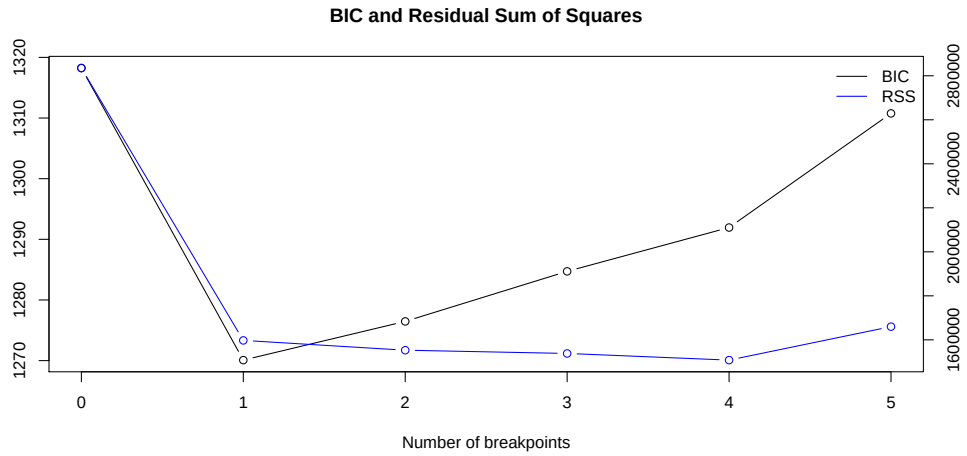
Die Aufgabe der hier vorgestellten *breakpoint-Analyse* (BAI and PERRON, 2003; ZEILEIS *et al.*, 2002, 2003) ist es, festzustellen, ob, wieviele und wo in einer Zeitreihe Strukturbrüche hinsichtlich eines spezifizierten linearen Modells vorliegen, z.B. ob und wo sich ein Mittelwert oder ein Trend ändert. Das Besondere an der vorgestellten Methode ist es, dass über eine BIC-basierte Modellselektion ein Modell mit einer optimalen Anzahl und Lage von Brüchen gefunden werden kann.

Wir demonstrieren auch hier wieder nur einen der einfachsten Fälle, die Konstanz des Lageparameters. Die Funktion `breakpoints` dient dazu, eine Reihe von Strukturbruchmodellen systematisch durchzutesten und zu bewerten. Als Ergebnis erhält man Kandidaten für die Strukturbrüche sowie deren RSS (*residual sums of squares*) und BIC (*Bayes Information Criterion*). Das Ergebnis lässt sich mit `summary` ausgeben und das Verhalten von `RSS` und `BIC` mit einer Plotfunktion visualisieren:

```
bp.nile <- breakpoints(Nile ~ 1)
summary(bp.nile)
```

```
         Optimal (m+1)-segment partition:

Call:
breakpoints.formula(formula = Nile ~ 1)

Breakpoints at observation number:

m = 1      28
m = 2      28       83
m = 3      28    68 83
m = 4      28 45 68 83
m = 5   15 30 45 68 83

Corresponding to breakdates:

m = 1        1898
m = 2        1898           1953
m = 3        1898      1938 1953
m = 4        1898 1915 1938 1953
m = 5   1885 1900 1915 1938 1953

Fit:

m   0       1       2       3       4       5
RSS 2835157 1597457 1552924 1538097 1507888 1659994
BIC    1318    1270    1276    1285    1292    1311
```

```
## the BIC also chooses one breakpoint
plot(bp.nile)
```

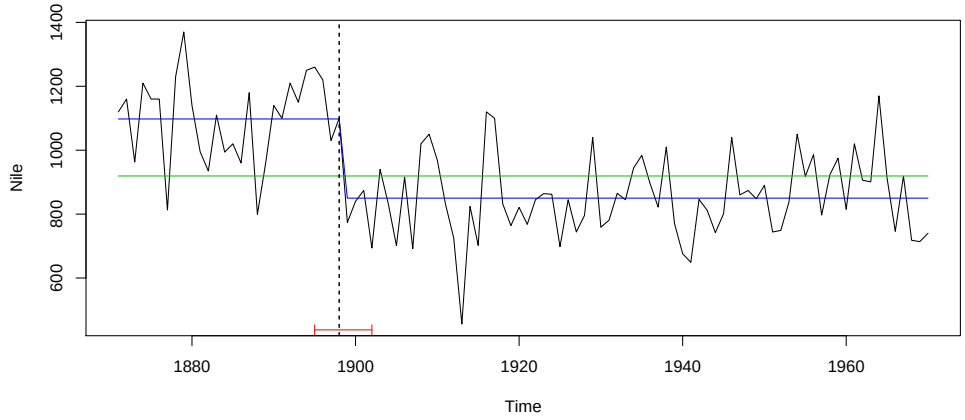



Man erkennt, dass das optimale Modell das Modell mit genau einem Strukturbruch ist. Zur Visualisierung der Breakpoints kann man nun ein stückweise lineares Modell anpassen. Die Funktion `breakfactor` erzeugt hierzu eine sogenannte Dummyvariable mit Codes für die einzelnen Segmente. Weitere generische Funktionen dienen zur Veranschaulichung von Lage `lines(bp.nile)` und Vertrauensintervallen (`confint(bp.nile), lines(ci.nile)`) des Strukturbruchs:

```
## fit null hypothesis model and model with 1 breakpoint
fm0 <- lm(Nile ~ 1)
fm1 <- lm(Nile ~ breakfactor(bp.nile,  breaks = 1))
plot(Nile)
lines(ts(fitted(fm0),  start = 1871),  col = 3)
lines(ts(fitted(fm1),  start = 1871),  col = 4)
lines(bp.nile)
## confidence interval
ci.nile <- confint(bp.nile)
ci.nile
```

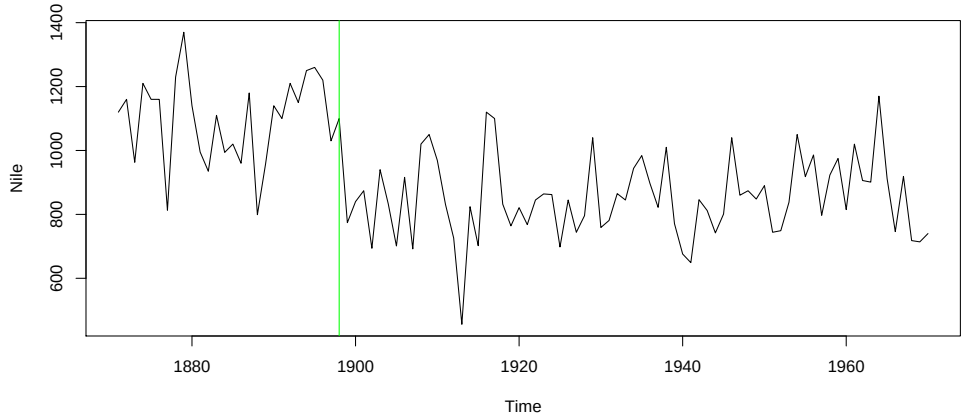
```
        Confidence intervals for breakpoints
        of optimal 2-segment partition:

Call:
confint.breakpointsfull(object = bp.nile)

Breakpoints at observation number:
  2.5 % breakpoints 97.5 %
1    25          28     32

Corresponding to breakdates:
  2.5 % breakpoints 97.5 %
1  1895        1898   1902
```

```
lines(ci.nile)
```




Anstelle der „generischen“ Funktionen kann man die Ergebnisse natürlich auch mit Standardfunktionen auswerten, hier z.B. die Lage des Strukturbruchs:

```
plot(Nile)
dat <- data.frame(time = time(Nile), Q = as.vector(Nile))
abline(v=dat$time[bp.nile$breakpoints],  col="green")
```




Zur Prüfung, ob das Modell mit Strukturbruch signifikant besser ist als das Nullmodell dient entweder eine paarweise ANOVA oder der Vergleich des AIC:

```
anova(fm0, fm1)
```

```
Analysis of Variance Table

Model 1: Nile ~ 1
Model 2: Nile ~ breakfactor(bp.nile, breaks = 1)
  Res.Df     RSS Df Sum of Sq     F    Pr(>F)
1     99 2835157
2     98 1597457  1   1237700 75.93 7.439e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Vergleich per AIC
AIC(fm0,fm1)
```

```
    df      AIC
fm0  2 1313.031
fm1  3 1257.663
```

Man erkennt, dass das Modell mit dem Strukturbruch im Jahr 1898 gegenüber dem Nullmodell signifikant besser ist.

Für die gefitteten Modelle sind natürlich auch weitere Analysen möglich, z.B. ein diagnostischer Vergleich von Autokorrelation und Spektraldichte der Residuen oder eine Prüfung auf deren Normalverteiltheit:

```
acf(residuals(fm0))
acf(residuals(fm1))
spectrum(residuals(fm0), log="no",  spans=c(5, 5))
spectrum(residuals(fm1), log="no",  spans=c(5, 5))
qqnorm(residuals(fm0))
qqnorm(residuals(fm1))
```

### 9.6.3 Aufgabe

Im folgenden Beispiel mit einem Originaldatensatz von IHLE *et al.* (2005) geht es darum, wie sich die Abundanz der Cyanobakteriengattung *Microcystis* im Sediment einer hocheutrophen Talsperre ändert. Die Hypothese bestand, vereinfacht gesagt, darin, dass ein Habitatwechsel zwischen benthischer und pelagischer Lebensweise vorliegt. Es wurde das Strukturbruchverfahren verwendet um Zeitpunkte zu identifizieren, diese wurden auf Koinzidenz mit anderen Größen untersucht.

Die Analyse folgt dem vorangegangenen Beispiel. Ein kleiner Unterschied besteht jedoch darin, dass die Zeitreihe nicht äquidistant ist. Das ist im vorliegenden Fall kein großes Problem, da sich die Analyse auf den Lageparameter beschränkt. Anstelle der Klasse `ts` verwenden wir deshalb die Klasse `zoo` für nicht äquidistante Zeitreihen.

Testen Sie das Beispiel und diskutieren Sie das Ergebnis an Hand der Publikation von IHLE *et al.* (2005).

```
dat    <- read.table("http://www.simecol.de/data/cells_sediment.txt",
  header = TRUE)
time   <- as.Date(as.character(dat$date), format = "%d.%m.%Y")
## zoo: Z's ordered observations, von strucchange automatisch nachgeladen
Cells <- zoo(dat$cells, time)
par(mfrow=c(2, 1))
bp <- breakpoints(Cells ~ 1, h = 5)
plot(bp)
fm1 <- lm(Cells ~ breakfactor(bp))
plot(Cells, type = "p")
lines(fitted(fm1), col = "red")
abline(v = time[bp$breakpoints], col = "blue")
(ci <- confint(bp))
sctest(Cells ~ 1, type = "OLS-CUSUM")
```

# 10 Multivariate Verfahren

Multivariate Verfahren dienen dazu, gleich mehrere Variablen simultan auszuwerten. Dies kann zum einen sinnvoll sein, um sich einen Überblick über einen größeren Datensatz zu verschaffen oder um erste Hypothesen zu bilden (explorative Analyse), die später noch im Detail untersucht werden sollen. Zum anderen werden multivariate Verfahren dann angewendet, wenn sich der zu untersuchende Effekt auf mehrere Variablen verteilt, z.B. auf das Vorkommen mehrerer Arten, auf mehrere Peaks in einer chemischen Analyse oder mehrere Unterschiede an verschiedenen Stellen eines DNA-Abschnitts.

Im Allgemeinen beginnt man mit einer rechteckigen Datenstruktur, z.B. einer Matrix oder einem Dataframe und man bezeichnet die Zeilen als Beobachtungen, Sites, Fälle oder Objekte und die Spalten als Variablen, Eigenschaften oder Spezies. Die multivariaten Verfahren sollen in dieser Matrix nach Strukturen, genauer nach Ähnlichkeiten, Unterschieden und Korrelationen suchen, nach Möglichkeit die Anzahl der Dimensionen reduzieren und das Ergebnis numerisch oder grafisch darstellen. Das folgende Kapitel gibt zunächst einen knappen Überblick über wichtige Verfahren und zeigt dann Beispiele zu deren Anwendung in R. Eine detaillierte Darstellung der Hintergründe findet sich in LEGENDRE and LEGENDRE (1998) oder auch in LEYER and WESCHE (2007).

## 10.1 Grundkonzepte

Die benutzten Grundkonzepte sind Kovarianz bzw. Korrelation (siehe oben), Abstand und Ähnlichkeit.

Da sehr verschiedene Abstands- und Ähnlichkeitsmaße existieren ist es sinnvoll, zunächst deren grundlegende Eigenschaften axiomatisch zu definieren.

Für ein Abstandsmaß $d$ zwischen den mehrdimensionalen Punkten $x_i$ und $x_j$ gelten die Bedingungen:

1. $d(x_i,x_j) \geq 0$
2. $d(x_i,x_j) = d(x_j,x_i)$
3. $d(x_i,x_i) = 0$

Darüberhinaus heißt ein Abstandsmaß metrisch, wenn:

- $d = 0$ nur bei Gleichheit und
- die Dreiecksungleichung (der Umweg ist länger als der direkte Weg) gilt.

Auf ähnliche Weise lässt sich auch ein Ähnlichkeitsmaß $s$ definieren:

1. $s(x_i,x_j) \leq s_{max}$
2. $s(x_i,x_j) = s(x_j,x_i)$
3. $s(x_i,x_i) = s_{max}$

und es ist metrisch, wenn:

- $s_{max}$ gilt nur bei Gleichheit und
- es gilt die Dreiecksungleichung.

Man kann Ähnlichkeit und Abstand wechselseitig transformieren. Allerdings existieren dazu unterschiedliche Möglichkeiten, z.B.:

$s = 1 - d/d_{max} \quad d = 1 - s/s_{max}$
$s = exp(-d) \quad d = -ln(s - s_{min})$

In vielen Fällen ergibt sich dadurch eine Vereinfachung, dass einige Autoren den Wertebereich für Ähnlichkeitsmaße auf das Intervall $(0,1)$ beschränken und entsprechend die Werte $d = 1 - s$ Unähnlichkeitmaße nennen, während für die allgemeineren Abstandsmaße das Intervall $(0,\infty)$ gilt.

Aus den vielen verschiedenen Abstandsmaßen sind vielleicht die folgenden fünf die für uns wichtigsten:

- Euklidischer Abstand (kürzeste Verbindung zweier Punkte im Raum),
- Manhattan-Abstand (um die Ecke herum, wie bei den Hochhäusern in Manhattan),
- Chiquadrat-Abstand zum Vergleich von Häufigkeiten,
- Mahalanobis-Abstand (berücksichtigt Kovarianz),
- Bray-Curtis Dissimilarity Index (speziell zum Vergleich von Artenlisten).

#### Euklidischer Abstand

Der euklidische Abstand (kürzeste Verbindung nach dem Satz des Pythagoras) erscheint zunächst als einzig natürliches Kriterium:

$$d = \sqrt{\sum (x_{ij} - x_{ik})^2}$$

In der Praxis ist das jedoch nicht immer so, da wir ja oft unterschiedliche Maßeinheiten oder ganz unterschiedliche Prozesse miteinander vergleichen wollen. Wenn wir z.B. die Phosphor- und Stickstoffkonzentration eines Gewässers in $mgl^{-1}$ angeben, dann hat der Phosphor im Regelfall nur verschwindend kleine Werte, so dass der Unterschied der Stickstoffkonzentrationen die Abstandsberechnung dominiert.

Es ist also unbedingt notwendig, die unterschiedlichen Maßeinheiten und Wertebereiche vergleichbar zu machen (Skalierung). Eine Möglichkeit bietet die Standardisierung:

$$x_i' = \frac{x_i - \bar{x}}{s}$$

d.h. die Subtraktion des Mittelwertes und die Division durch die Standardabweichung. Allerdings ist auch das nicht immer angebracht, wenn wir zum Beispiel an häufigere und seltenere Arten denken. Ein Einzelfund soll ja durchaus weniger Gewicht erhalten als eine dominante Art.

Soll durch die unterschiedlichen Werte (z.B. Peakhöhen oder Abundanzen) eine Wichtung ausgedrückt werden, führt man lediglich eine Zentrierung durch:

$$x_i' = x_i - \bar{x}$$

Weitere Möglichkeiten der Daten-Voraufbereitung sind:

- die Transformation der Daten, z.B. durch Potenzen, Wurzeln oder Logarithmen,
- die Bildung von Rängen,
- die Abstandsberechnung oder Ähnlichkeitsberechnung für binäre (ja/nein) oder nominale Datentypen.

#### Manhattan-Abstand

Der Manhattan-Abstand (oder City-Block-Distanz) entspricht bildlich einem Abstand „um die Ecke herum“, d.h. er ist die Summe aller Abstände über alle Dimensionen:

$$d = \sum |x_{ij} - x_{ik}|$$

#### Mahalanobis-Abstand

Der Mahalanobis-Abstand ist ein spezielles Maß, das die Kovarianz (d.h. die wechselseitige Abhängigkeit) der betrachteten Dimensionen berücksichtigt. Nehmen wir an, wir hätten vier Variablen, z.B. Stickstoff-, Phosphor und Phytoplanktonkonzentration in einem See sowie die Seetiefe. Die ersten drei Variablen sind Trophiekriterien und normalerweise untereinander korreliert, da sie alle von der externen Belastung abhängen. Wenn wir nun eine multivariate Analyse durchführen, geht die Trophie gleich dreimal in die Statistik ein, die Seetiefe nur einmal und die möglicherweise vorhandenen kleinen Unterschiede zwischen Stickstoff, Phosphor und Phytoplankton gehen in der Analyse unter.

An dieser Stelle kann der Mahalanobis-Abstand helfen, da er die Korrelationen zwischen den Variablen berücksichtigt und unterdrückt. Allerdings können dadurch unter Umständen unbedeutende Effekte und Fehler ein größeres Gewicht erlangen und die Analyse verfälschen.

Auf der anderen Seite bietet der Mahalanobis-Abstand jedoch den Vorteil, dass wegen der Berücksichtigung der Kovarianzstruktur keine Skalierung (Vergleichbarmachung unterschiedlicher Skalen) erforderlich ist.

#### Chiquadrat-Distanz

Die Chiquadrat-Distanz ($\chi^2$-Distanz) dient zum Vergleich von Häufigkeiten, wie sie z.B. in der Korrespondenzanalyse verwendet werden (siehe CA und CCA). Der Parameter Chiquadrat ($\chi^2$) gibt an, wie sehr eine beobachtete Häufigkeit von einer erwarteten Häufigkeit abweicht, d.h. allgemein:

$$\chi^2 = \sum_{i=1}^{n} \frac{(B_i - E_i)^2}{E_i}$$

wobei $B_i$ die beobachtete und $E_i$ die erwartete Häufigkeit ist. Zieht man die Wurzel, erkennt man, dass $\chi$ letztlich ein gewichteter euklidischer Abstand ist, wobei das Gewicht der Kehrwert der erwarteten Häufigkeit ist. Für die Korrespondenzanalyse bedeutet das, dass jede Häufigkeit auf die Gesamtsumme der Werte in den Spalten bzw. Zeilen der Tabelle bezogen wird. Variablen mit großen Werten (z.B. häufige Arten) werden folglich herabgewichtet, Variablen mit geringeren Häufigkeiten dagegen betont.

In der von LEYER and WESCHE (2007) verwendeten Schreibweise ergibt sich somit:

$$D_{\chi^2_{1,2}} = \sqrt{x_{++}} \sqrt{\sum_{k=1}^{m} \frac{1}{x_{+k}} \left( \frac{x_{1k}}{x_{1+}} - \frac{x_{2k}}{x_{2+}} \right)^2} \tag{10.1}$$

Tabelle 10.1: Beispiel für die Berechnung des Bray-Curtis-Koeffizienten aus Artenhäufigkeiten aus (LEYER and WESCHE, 2007)

| | 1 | 2 | 3 | 4 | 5 | 6 | B | C | w |
|---|---|---|---|---|---|---|---|---|---|
| Objekt 1 | 7 | 3 | 2 | 0 | 4 | 0 | 16 | | |
| Objekt 2 | 4 | 4 | 0 | 0 | 6 | 5 | | 19 | |
| Min | 4 | 3 | 0 | 0 | 4 | 0 | | | 11 |

Hierbei ist $x_{+k}$ die Summe aller Werte für die Art $k$, $x_{1+}$ die Summe aller Werte für die Beobachtung 1 und $x_{++}$ die Summe aller Werte in der Tabelle.

#### Bray-Curtis-Abstand

Der Bray-Curtis-Abstand ist ein nichtmetrisches Abstandsmaß für den Vergleich von Abundanzen. Im Zähler steht die Summe der Abundanzdifferenzen und im Nenner die Summe aller Abundanzsummen der zu vergleichenden Beobachtungen (z.B. Probenahmestellen):

$$d = \frac{\sum |x_{ij} - x_{ik}|}{\sum (x_{ij} + x_{ik})} \tag{10.2}$$

Alternativ ergibt sich der Bray-Curtis-Koeffizient ($S_{bc}$) auch als das doppelte Verhältnis zwischen der in zwei Beobachtungen gemeinsam vorkommenden Arten ($w$) zur Summe der Einzelvorkommen in beiden Beobachtungen ($B$ bzw. $C$):

$$s_{bc} = \frac{2w}{B+C} \tag{10.3}$$

bzw. als Distanzmaß (Bray-Curtis-Unähnlichkeit):

$$d_{bc} = 1 - s_{bc} \tag{10.4}$$

Das folgende Beispiel soll die Anwendung verdeutlichen. Wir gehen von 2 Objekten (z.B. Probenahmestellen) aus an denen insgesamt 6 Arten gefunden wurden (siehe Tab. 10.1).

Zunächst berechnen wir den Bray-Curtis-Koeffizienten (`sbc`) bzw. den Bray-Curtis-Unähnlichkeit (`1-sbc`) gemäß Gleichung 10.3:

```
ob1 <- c(7, 3, 2, 0, 4, 0); ob2 <- c(4, 4, 0, 0, 6, 5)
B <- sum(ob1); C <- sum(ob2); w  <- sum(pmin(ob1, ob2))
sbc <- 2*w/(B+C)
sbc
```

```
[1] 0.6285714
```

```
1 - sbc
```

```
[1] 0.3714286
```

sowie zum Vergleich nach Gleichung 10.4:

```
sum(abs(ob1-ob2))/sum(ob1+ob2)
```

```
[1] 0.3714286
```

und zum Schluss mit der „vorkonfektionierten“ Funktion `vegdist` aus dem **vegan**-Paket:

```
library(vegan)
vegdist(rbind(ob1, ob2))
```

```
          ob1
ob2 0.3714286
```

In allen drei Fällen ergib sich das selbe Ergebnis, wobei die `vegdist`-Funktion normalerweise dazu dient, eine komplette Abstandsmatrix für den paarweisen Vergleich einer Mehrzahl von Objekten zu erstellen.

## 10.2 Ordinationsverfahren

Ordinationsverfahren dienen dazu, Messungen „in eine Reihenfolge zu bringen“, wobei diese Reihenfolge nicht auf eine Strecke beschränkt ist sondern mehrdimensional sein kann. Die Anzahl der Dimensionen ist dabei kleiner oder gleich der ursprünglichen Anzahl an Variablen, wobei im Regelfall versucht wird die in den Daten vorhandene Information auf möglichst wenige Dimensionen zu reduzieren (Dimensionsreduktion).

In der Vergangenheit wurde eine große Zahl unterschiedlicher Verfahren entwickelt, die sich grob in Einmatrixverfahren (auch *unconstrained ordination* genannt) und Zweimatrixverfahren (*constrained ordination*) unterteilen lassen.

Wie schon der Name sagt, betrachten die Einmatrixmethoden zunächst nur eine einzelne Matrix, z.B. eine Artenliste oder eine Matrix mit physikalisch-chemischen Daten. Bei den Zweimatrixmethoden unterscheidet man eine abhängige Matrix (z.B. eine Artenliste) und eine unabhängige oder Erklärungsmatrix (z.B. Umweltfaktoren). Im Folgenden werden wir deshalb oft vereinfacht von „Artenliste“ und „Umweltmatrix“ sprechen, grundsätzlich sind die besprochenen Verfahren aber auch auf andere Datensätze anwendbar, z.B. auf Ergebnisse von Simulationsmodellen oder auf Marketingdaten.

### 10.2.1 PCA: Principal Components Analysis (Hauptkomponentenanalyse)

Die Hauptkomponentenanalyse hat das Ziel, in einen vieldimensionalen Raum (der durch die Variablen gebildet wird) neue Koordinatenachsen so hineinzulegen, dass sich ein möglichst großer Teil der Information in möglichst wenigen Ebenen wiederfindet. Unter „Information“ ist hier die gesamte Varianz der Daten zu verstehen und ein möglichst großer Varianzanteil soll durch möglichst wenige künstliche Koordinatenachsen, die Hauptkomponenten, dargestellt werden (Verringerung der Dimensionalität). Das Verfahren beruht darauf, dass das urprüngliche Koordinatensystem mit Hilfe von Matrixoperationen (Eigenwertzerlegung) gedreht wird.

Die Arbeitsgrundlage des Verfahrens und ein nützliches Nebenergebnis ist die Kovarianz- bzw. Korrelationsmatrix. Nähere Details sind der entsprechenden Literatur zu entnehmen (siehe z.B. VENABLES and RIPLEY, 2002).

Ein besonderer Vorteil der Hauptkomponentenanalyse besteht darin, dass die Variablen und die Objekte in einer gemeinsamen Grafik (einem Biplot) dargestellt werden können. Ein weiterer Vorteil ist, dass die PCA sehr gut nachvollziehbar und damit interpretierbar ist. Darüber hinaus kann die PCA auch als Ausgangspunkt für weitere Verfahren dienen, z.B. zur Dimensionsreduktion für Clusteranalysen, als Startpunkt für eine MDS (Multidimensional Scaling) oder zur Durchführung einer Hauptkomponentenregression.

Die Hauptnachteile sind, dass die PCA auf den euklidischen Abstand angewiesen ist und dass es häufig nicht gelingt, einen ausreichenden Anteil der Information in eine Ebene zu bringen.

### 10.2.2 CA: Korrespondenzanalyse

Die Korrespondenzanalyse wurde gleich mehrfach unter verschiedenem Namen entwickelt und lässt sich auch auf unterschiedlichem Wege berechnen. Die klassische Methode basiert auf der sogenannten Gradientenanalyse (wie lassen sich Arten nach einem Umweltgradienten ordnen) und wird auch als *weighted averaging* bezeichnet (siehe LEYER and WESCHE, 2007, für eine Beschreibung des klassischen Verfahrens).

In modernen Programmen wird die CA jedoch ähnlich wie die PCA immer mit Hilfe von Eigenwertzerlegungen berechnet, wobei im Unterschied zur PCA lediglich die die Chiquadrat-Distanz anstelle des euklidischen Abstandes verwendet wird.

Kurz zusammengefasst ist also die PCA eine Methode zur Analyse metrischer Daten, die CA dagegen eine Methode zur Analyse nominaler Häufigkeitsdaten.

### 10.2.3 PCO: Principal Coordinate Analysis

Sowohl PCA als auch CA sind jeweils auf ein bestimmtes Abstandsmaß beschränkt. Die PCO-Methoden (oder auch PCoA) sind demgegenüber eine Weiterentwicklung, da sie auch andere Abstandsmaße (oder Unähnlichkeitsmaße) als den euklidischen Abstand erlauben. Sie sind deshalb flexibler, allerdings ist die Interpretation aufgrund der sehr vielen unterschiedlichen Varianten manchmal nicht ganz einfach. PCO ist die klassische Form der MDS, das metrische Multidimensional Scaling.

Die R-Funktionen `dist` aus dem `stats`-Package oder `vegdist` aus dem `vegan`-Package erlauben neben dem euklidischen und dem Chiquadrat-Abstand weitere Abstandsmaße, z.B. die Bray-Curtis-Distanz oder die Canberra-Distanz. Darüber hinaus können der PCO auch eigene Abstandsmatrizen (z.B. auf Basis der Renkonen-Zahl) übergeben werden.

### 10.2.4 Nichtmetrisches Multidimensional Scaling (NMDS)

Die PCO ist das Verfahren, das die beste **verzerrungsfreie** geometrische Projektion höherdimensionaler Daten auf eine niedrigere Dimension erzeugt. Manchmal erhält man jedoch eine noch bessere Abbildung der Ähnlichkeitsstruktur, wenn man eine gewisse Verzerrung in Kauf nimmt. Im Gegensatz zur Suche nach Koordinatenachsen mit einem möglichst hohen Varianzanteil sucht man bei den nichtmetrischen Verfahren nach einer Darstellung mit einer maximalen Korrelation der Distanzen in der Abbildungsebene.

Es existieren unterschiedliche iterative Verfahren, die dieses Ziel anstreben, z.B. das Nonlinear Mapping (Sammon) und Kruskal's Nonmetric Multidimensional Scaling (NMDS). In beiden Verfahren wird die Güte der Abbildung durch einen Stress-Wert angegeben, die Berechnungsverfahren sind jedoch verschieden. In den Paketen **vegan** und **MASS** wird der sogenannte „Stress 1" angegeben (Tabelle 10.2):

$$S_1 = \sqrt{\frac{\sum_{i \neq j}(\theta(d_{ij}) - \tilde{d}_{ij})^2}{\sum_{i \neq j} \tilde{d}_{ij}^2}}$$

mit $\tilde{d}_{ij}$ = Ordinationsdistanz und $\theta(d_{ij})$ = beobachte Unähnlichkeit. Eine weitere Bewertungsmöglichkeit liefert der sogenannte Shepard-Plot. Dieser vergleicht die in der NMDS dargestellten Abstände mit den Originalabständen. Der NMDS-Fit ist als Treppenkurve eingezeichnet.

Da die NMDS iterativ arbeitet, erhalten wir je nach Ausgangslage unterschiedliche Ergebnisse. Es ist nicht möglich, *die* beste Darstellung zu finden, sondern man findet nur lokale Minima. Man muss deshalb entweder mehrere Versuche mit unterschiedlicher Startkonfiguration durchführen, z.B. mit Hilfe von Zufallszahlen oder man startet mit einer bekanntermaßen guten Konfiguration, die z.B. mit Hilfe einer PCO ermittelt wurde. Die Funktion `isoMDS` führt standardmäßig eine NMDS mit PCO-Startkonfiguration durch. Für eine mehrfache automatische MDS mit zufälliger Initialisierung kann die Funktion `metaMDS` aus dem `vegan`-Paket verwendet werden.

Tabelle 10.2: Richtwerte für den Stress für die Funktion `isoMDS` sowie zum Vergleich die Richtwerte der SPSS-Prozedur ALSCAL

| | Stress 1 (R) | Stress 2 (SPSS-ALSCAL) |
|---|---|---|
| gering | 0.2 | 0.4 |
| ausreichend | 0.1 | 0.2 |
| gut | 0.05 | 0.1 |
| ausgezeichnet | 0.025 | 0.05 |
| perfekt | 0 | 0 |

### 10.2.5 CCA und RDA: Kanonische Korrespondenzanalyse und Redundanzanalyse

Redundanzanalyse und Kanonische Korrespondenzanalyse und sind mit einer multiplen linearen Regression vergleichbar, bei der aber im Unterschied nicht nur mehrere unabhängige Variablen (Erklärungsvariablen) sondern auch eine komplette Matrix von abhängigen Variablen (z.B. eine komplette Artenmatrix) miteinander in Beziehung gesetzt werden.

Bei der RDA und der CCA handelt es sich also um Zweimatrixverfahren. Diese werden auch als *constrained ordination* bezeichnet, weil hier zwischen dem durch die Umweltmatrix erklärbaren Dimensionen (*constrained axes*) und den übrigen Dimensionen unterschieden wird (*unconstrained axes*).

Von ihrer unterschiedlichen Herkunft abgesehen, sind beide Verfahren vom Kern her relativ ähnlich wobei die RDA mit euklidischen Abständen arbeitet, die CCA jedoch mit der Chiquadratdistanz. Daraus folgt, dass die PCA letztlich ein Spezialfall der RDA und die CA ein Spezialfall der CCA ist, wenn man jeweils die erklärenden „Umwelt“-Matrizen weglässt.

Die CCA ist in der Vegetationsökologie sehr weit verbreitet und wird dort, zusammen mit einer speziellen Variante der DCA (Detrended Correspondence Analysis) als Standardverfahren empfohlen. Ein großer Vorteil der Verfahren ist es, dass Umweltfaktoren und Artenlisten simultan analysiert und dargestellt werden können, ein weiterer dass mit diesen Verfahren außerordentlich lange Erfahrungen in der Interpretation existieren.

Allerdings ist es (zumindest am Anfang) manchmal nicht leicht zu verstehen, dass nur der erklärbare Anteil der Information („Inertia“=Masse genannt) dargestellt wird. Außerdem produzieren RDA und CCA manchmal unerwartete Artefakte (sogenannte Bogen- und Hufeisen-Effekte), die auch bei der dafür entwickelten DCA nur unvollständig (und in einer nach Meinung einiger Autoren wenig nachvollziehbaren Weise) behoben werden. Die DCA ist in R unter dem Namen `decorana` zu finden.

## 10.3 Vector Fitting

„Vector fitting“ ist die nachträgliche Anpassung von Umweltvektoren an eine mit einer Einmatrixmethode (z.B. CA, PCA oder NMDS) erhaltene Ordination.

Da sich hierdurch also ebenfalls der Einfluss von Umweltfaktoren auf eine Artzusammensetzung darstellen lässt, ist die NMDS mit anschließendem Vector-Fitting eine einfachere und auch leichter verständliche Alternative zur CCA.

## 10.4 Randomisierungstests

Ordinationsverfahren liefern zunächst nur eine graphische Darstellung. In vielen Fällen ist es jedoch wünschenswert, auch Informationen über die Signifikanz der dargestellten Verhältnisse zu bekommen. Im Zusammenhang mit multivariaten Verfahren werden zu diesem Zweck meistens Randomisierungsverfahren verwendet. Diese beruhen darauf, dass ein geeignetes Testkriterium berechnet wird und die beobachtete Konfiguration mit einer Vielzahl zufälliger Konfigurationen verglichen wird.

Je nach Aufgabenstellung existieren unterschiedliche Randomisierungstests, z.B.:

- Der Mantel-Test vergleicht zwei Distanz-Matrizen (z.B. Artenliste und Umweltmatrix) und prüft die Hypothese ob zwischen diesen Distanzmatrizen eine Korrelation besteht (z.B. Zusammenhang zwischen Umweltfaktoren und Besiedlung).
- Procrustes-Analysen erlauben es, Ähnlichkeiten und Unterschiede zweier Ordinationen darzustellen (Kongruenzanalyse). Insbesondere testet der Procrustes-Test ob zwei Ordinationen signifikant verschieden sind. Er kann deshalb als Alternative zu Mantel-Tests verwendet werden.
- ANOSIM ist ein Resampling-Analogon zur ANOVA und prüft die Hypothese ob ein Unterschied zwischen zwei oder mehreren Aufnahmeserien (z.B. nach anthropogenem Eingriff) besteht. Der ANOSIM-Test arbeitet mit Rängen. Ein Problem ist, dass der ANOSIM-Test nicht robust gegenüber unterschiedlichen Gruppengrößen und Gruppenheterogenitäten ist, so dass man unter Umständen falsch signifikante Ergebnisse erhalten kann (OKSANEN, 2010).
- Für die Zweimatrixverfahren (z.B. CCA oder RDA) existiert ein Permutationstest für die Signifikanz der Constraints. In R kann dieser einfach über die Funktion `anova` mit einem Ergebnisobjekt einer CCA oder RDA aufgerufen werden.
- Die Multiple Response Permutation Procedure (MRPP) ist von einer Aussagekraft ähnlich ANOSIM, arbeitet im Gegensatz dazu aber mit den Originaldistanzen. Das Verfahren erkennt Unterschiede der Lage (Artenhäufigkeit) und auch der Streuung (Diversität).
- Ein relativ neues und vielversprechendes Verfahren zur multivariate Varianzanalyse von Distanzmatrizen (ANDERSON, 2001) ist in R unter dem Namen ADONIS zu finden. Das Verfahren untersucht also,

inwiefern komplette Artenlisten von bestimmten mit Hilfe einer Modellformel spezifizierten Umweltfaktoren abhängen. Wichtig ist dabei oft, die Permutation auf entsprechende Strata zu beschränken, z.B. wenn die Aufnahmen zu unterschiedlichen Zeiten in der Saison gemacht wurden.

In R sind die genannten Verfahren über die Funktionen `mantel`, `anosim`, `mrpp` und `adonis` im **vegan**-Paket zu finden. Darüber hinaus bietet das speziell aus dem Statistikpaket PRIMER (CLARKE, 1993; CLARKE and WARWICK, 2001) bekannte Verfahren BIOENV (R-Funktion `bioenv`) eine weitere Methode um die „beste Teilmenge von Erklärungsvariablen“ zu identifizieren die eine Artenliste erklären.

Multivariate Resampling-Tests ermöglichen es, Artenzusammensetzungen (oder andere multivariate Datensätze) direkt miteinander zu vergleichen ohne dass vorher irgendwelche Indizes (z.B. Diversitätsindizes) berechnet werden müssen. Allerdings gelten auch hier die üblichen statistischen Voraussetzungen, insbesondere die Repräsentativität und die Unabhängigkeit von Stichproben. Es zeigt sich, dass multivariate Resamplingtests oft relativ leicht signifikante Unterschiede finden. Die Frage ist jedoch immer was es bedeutet, wenn Artenzusammensetzungen unterschiedlich sind. Eine fachspezifische Interpretation und Bewertung hinsichtlich der Relevanz der Ergebnisse sind hier noch wichtiger als sonst.

## 10.5 Klassifikationsverfahren

Die Clusteranalyse dient dazu, Gruppen ähnlicher Objekte zu finden. Im Gegensatz zu den Ordinationsverfahren PCA, PCO, MDS usw. sollen die Gruppen nicht in ihrer Lage zueinander visualisiert werden, sondern deren Abstände in Form eines Baumdiagramms. Während es den Ordinationsverfahren nicht immer gelingt, die vorhandene Datenstruktur auf wenige Dimensionen, z.B. eine Ebene, abzubilden, kommen Clusterverfahren auch mit hochdimensionalen, nicht korrelierten Daten zurecht. Es ist deshalb oft sinnvoll, Ordinationsverfahren und Clusteranalysen zu kombinieren.

Grundsätzlich existieren hierarchische, nichthierarchische, aufsteigend und absteigend klassifizierende Verfahren der Clusteranalyse. Im folgenden sollen zwei Verfahren, die aufsteigende (agglomerative) hierarchische Clusterung und ein nichthierarchisches absteigendes (divisives) Verfahren vorgestellt werden.

#### 10.5.0.1 Hierarchische Clusteranalyse

Die hierarchische Clusteranalyse startet mit den Einzelobjekten und fasst die jeweils nächsten Objekte zu einem Cluster zusammen. Der Algorithmus ist:

1. Suche kleinsten Abstand,
2. Fasse Objekte mit kleinstem Abstand zu neuem Objekt zusammen,
3. Aktualisiere die Abstandsmatrix (Abstände zum neuen Objekt),
4. Gehe zu 1., wenn noch mehr als 2 Objekte existieren.

Hierbei entsteht in Schritt 3 das Problem, wie der Abstand zwischen einem Einzelindividuum und einem bereits gebildeten Cluster bestimmt werden soll. Dem entsprechend existiert eine große Vielzahl an Möglichkeiten und es fällt auf den ersten Blick manchmal schwer, sich zu entscheiden:

| | |
|---|---|
| Single Linkage | nächster Nachbar |
| Complete Linkage | entferntestes Element |
| Average Linkage | mittlerer Abstand |
| Median (gewichtet/ungewichtet) | Abstand der Mediane |
| Centroid (gewichtet/ungewichtet) | Schwerpunkt |
| WARD (gewichtet/ungewichtet) | minimale Varianz |
| flexible Strategien | einstellbares Verhalten |

Während *single linkage* jeweils den kürzesten Abstand der nächsten Elemente eines Clusters benutzt, arbeitet *complete linkage* so, dass die jeweils entferntesten Elemente den Abstand bestimmen. Die WARD-Methode stellt eine Besonderheit dar. Hierbei wird der Abstand so berechnet, dass die Varianz des bei der Zusammenfassung entstehenden Clusters möglichst gering bleibt. Alle anderen Verfahren verwenden mittlere Abstände zwischen den Clustern.

*Single linkage* produziert häufig sehr weit verzweigte Cluster und Ketteneffekte, während die Cluster bei *complete linkage* meist erst sehr spät zusammengefasst werden. In der Regel erhält man den besten Kompromiss mit Hilfe der WARD-Strategie oder einem Mittelwertverfahren. Außerdem ist es keine schlechte Idee, mehrere Verfahren nacheinander durchzuführen und sich die Stabilität der Konfiguration (werden die Individuen immer zum selben Cluster geordnet) anzusehen.

#### 10.5.0.2 Nichthierarchische k-Means-Clusteranalyse

Bei der nichthierarchischen Clusteranalyse wird von vornherein festgelegt, wieviele Gruppen gebildet werden sollen. Die Cluster werden dann entweder aus einer Startkonfiguration oder mit Hilfe einere zufälligen Initialisierung iterativ zusammengefasst. Die Güte der Clusterung wird dann entweder durch Quadratsummen oder einen *stress*-Wert angegeben.

Die Anzahl der Cluster ergibt sich entweder aus sachlichen Überlegungen des Anwendungsfalles oder sie wird auf Basis einer hierarchischen Clusteranalyse festgelegt. K-Means-Clusterung kann angewendet werden wenn sehr viele Individuen klassifiziert werden sollen und ein Baumdiagramm zu unübersichtlich ist. Eine andere sinnvolle Anwendung besteht in der simultanen Darstellung der Ergebnisse einer k-Means-Clusterung und eines Displayverfahrens, z.B. einer MDS.

## 10.6 Beispiele und Implementierung in R

Obwohl die vorgestellten Verfahren eine zum Teil sehr verschiedene Arbeitsweise besitzen, ist es sinnvoll, die Implementation im Zusammenhang vorzustellen. Neben den hier vorgestellten Möglichkeiten die im wesentlichen aus den Paketen **stats**, **MASS** und **vegan** stammen finden sich weitere Verfahren in `cluster`, `knn`, `ade4`, `cclust`, `daisy` und anderen.

Das **vegan**-Paket hat sich in letzter Zeit in einer sehr erfreulichen Richtung weiterentwickelt und enthält jetzt eine ganze Reihe fortgeschrittener Funktionen die direkt auf die Bedürfnisse von Ökologen zugeschnitten sind.

Solche Erweiterungen betreffen z.B. die starke Erweiterung und Automatisierung der `metaMDS`-Funktion, die direkte Verknüpfung von Umweltdaten mit MDS-Grafiken (vector fitting und surface fitting) sowie erweiterte Randomisierungstests. Diese erlauben es eine ANOVA-artige Analyse mit ganzen Artenlisten (Ähnlichkeitsmatrizen) durchzuführen.

Ganz besonders erfreulich ist, dass diese Funktionen inzwischen gut dokumentiert sind. Auf der Homepage[1] des **vegan**-package Authors Jari Oksanen findet man ein sehr gut verständliches und empfehlenswertes Tutorial[2] und weitere Materialien zur multivariaten Statistik.

Interessant ist darüber hinaus auch der Ansatz der „französischen Schule“ der multivariaten Statistik (siehe z.B. DRAY and DUFOUR, 2007) für die auch eine dazu passende graphische Benutzeroberfläche existiert (THIOULOUSE and DRAY, 2007).

### 10.6.1 Beispiel 1: Seendatensatz

Das folgende Beispiel zeigt einige Kriterien mehrerer Seen Brandenburgs und Mecklenburgs aus CASPER (1985). Das Beispiel ist bewusst so klein, so dass man eigentlich keine multivariate Statistik benötigt. Es hat jedoch den Vorteil, dass man die Ergebnisse ohne weiteres im Kopf mit der Realität vergleichen kann.

| Seen | z (m) | t (a) | P ($\mu$g/l) | N (mg/l) | Chl ($\mu$g/l) | PP ($gCm^{-2}a^{-1}$) | ST m |
|---|---|---|---|---|---|---|---|
| S | 23.7 | 40 | 2.5 | 0.2 | 0.7 | 95 | 8.4 |
| NN | 5.9 | 10 | 2 | 0.2 | 1.1 | 140 | 7.4 |
| NS | 7.1 | 10 | 2.5 | 0.1 | 0.9 | 145 | 6.5 |
| BL | 25.2 | 17 | 50 | 0.1 | 6.1 | 210 | 3.8 |
| SL | 7.8 | 2 | 30 | 0.1 | 4.7 | 200 | 3.7 |
| DA | 5 | 4 | 100 | 0.5 | 14.9 | 250 | 1.9 |
| HS | 6.3 | 4 | 1150 | 0.75 | 17.5 | 420 | 1.6 |

#### Hauptkomponenten

Wir führen zunächst eine Hauptkomponentenanalyse durch und versuchen die Ergebnisse zu interpretieren, anschließend führen wir eine PCO durch. Zum Schluss wenden wir die MDS an, und zwar einmal mit einer PCO als Startkonfiguration (Default bei `isoMDS`) und einmal mit einer zufälligen Startkonfiguration.

Der Datensatz enthält in der ersten Spalte einen Bezeichner des Objektes (Seename). Dieser wird mit `row.names` als Zeilennamen der Matrix verwendet und anschließend aus dem eigentlichen Datensatz entfernt. Die Funktion `scale` führt eine Standardisierung durch und `prcomp` macht die eigentliche Analyse:

```
library(vegan)
library(MASS)
dat <- read.table("http://www.simecol.de/data/seen_bb.txt",header=TRUE)
# erste Spalte enthaelt Zeilennamen
row.names(dat) <- dat[,1]
# diese erste Spalte hinauswerfen
dat <- dat[,-1]
# Hauptkomponenten (mit standardisierten Daten)
pca <- prcomp(scale(dat))
# Varianzanteile (in Prozent)
summary(pca)
```

[1] http://cc.oulu.fi/~jarioksa/
[2] http://cc.oulu.fi/~jarioksa/opetus/metodi/vegantutor.pdf

```
Importance of components:
                          PC1    PC2     PC3    PC4     PC5     PC6       PC7
Standard deviation     2.2007 1.1411 0.76855 0.5006 0.11502 0.01502 4.169e-17
Proportion of Variance 0.6919 0.1860 0.08438 0.0358 0.00189 0.00003 0.000e+00
Cumulative Proportion  0.6919 0.8779 0.96228 0.9981 0.99997 1.00000 1.000e+00
```

```
biplot(pca, choices=c(1,2))
```

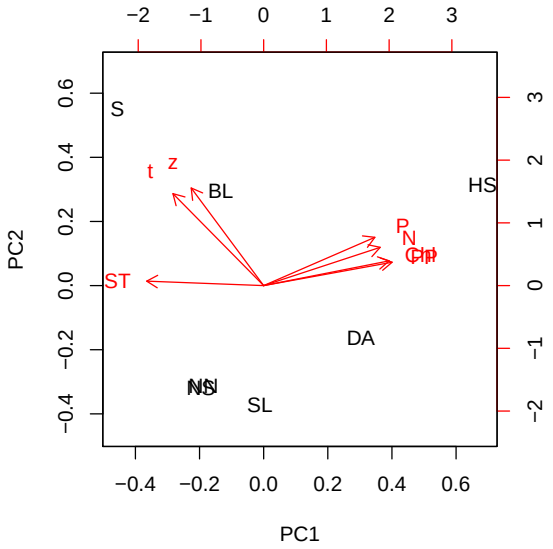



Abbildung 10.1: PCA des Seen-Datensatzes.

### Interpretationshilfe

Als erstes schaut man sich die von `summary(pca)` ausgegebenen Varianzanteile an. Diese entscheiden, ob die Dimensionsreduktion erfolgreich war und wie viele Hauptkomponenten man sich ansehen muss wobei die Hauptkkomponenten nach ihren Varianzanteilen sortiert vorliegen. Im vorliegenden Beispiel enfallen 88% der Varianz bereits auf die ersten beiden Hauptkomponenten. Das bedeutet, dass die Darstellung von PC 1 und PC 2 bereits 88% der Information enthält.

Die Grafische Darstellung mit `biplot` stellt Objekte (Seen) und Variablen simultan dar. Je nach verwendetem Algorithmus kann die Grafik an der x- oder y-Achse gespiegelt dargestellt sein, das heißt es kommt auf die relativen Beziehungen der Objekte und Variablen an, rechts oder links, oben oder unten sind nicht entscheidend.

Man interpretiert diese zunächst hinsichtlich der Objekte, z.B.:

- Nehmitzsee Nord ist ähnlich dem Nehmitzsee Süd, sie bilden ein Cluster,
- Stechlinsee und Dagowsee sind grundverschieden.

Bei der Interpretation der Variablen ist zu beachten, dass nur die langen Pfeile in der dargestellten Ebene liegen. Das bedeutet, nur lange Pfeile sind interpretierbar, kurze Pfeile zeigen in eine nicht dargestellte Dimension:

- Stickstoff, Phosphor, Chlorophyll und Primärproduktion sind korreliert,
- eine hohe Chlorophyllkonzentration bedeutet niedrige Sichttiefe,
- Phosphor und mittlere Tiefe sind nicht korreliert (zumindest nicht im vorliegenden Datensatz).

Auch eine kombinierte Interpretation ist möglich, wobei man die jeweiligen Objekte immer rechtwinklig auf die Pfeile projizieren muss:

- Der Stechlinsee hat die größte Tiefe und die längste Verweilzeit.
- Der Haussee hat die höchste Nährstoff- und Chlorophyllkonzentration sowie die größte planktische Primärproduktion.

#### Principal Coordinate Analysis

Die PCO kann ähnlich durchgeführt werden, die vorhandenen Möglichkeiten zur Verwendung beliebiger Abstandsmaße bieten aber beim vorliegenden Datensatz gegenüber der PCA keinen Vorteil da der verwendete euklidische Abstand für physikalisch-chemische Daten durchaus angemessen ist.

Der Unterschied zum vorangegangenen Beispiel besteht jedoch darin, dass nicht die Originaldaten sondern eine Abstandsmatrix übergeben wird. Diese erhält man mit den Funktionen `dist` (hat als Standard den euklidischen Abstand) bzw. der erweiterten Funktion `vegdist` (liefert als Standard den Bray-Curtis Dissimilarity, kann u.a. aber auch euklidische Abstände liefern):

```
pco <- cmdscale(vegdist(dat, method="euclidean"), k=2, eig=TRUE)
plot(pco$points, type="n")
text(pco$points, row.names(dat))
```

Der Parameter `k` bestimmt die Anzahl der zu extrahierenden Dimensionen, der Parameter `eig` ermöglicht die Rückgabe der für die grafische Darstellung wichtigen Eigenwerte.

#### NMDS

Mit Hilfe der Funktion `isoMDS` erhält man eine Darstellung, bei der die Konfiguration der PCO als Start verwendet wird (bitte `trace` auf `TRUE` setzen um Zwischenergebnisse anzuzeigen):

```
mds <- isoMDS(vegdist(dat, method="euclidean"), trace=FALSE)
mds$stress
```

```
[1] 1.670176e-14
```

```
plot(mds$points, type="n")
text(mds$points, row.names(dat))
```

Man kann eine MDS auch mit einem zufälligen Start durchführen (`initMDS`) oder auch die Ergebnisse mit Hilfe von `postMDS` so skalieren damit sie leichter interpretierbar werden.

```
dist <- vegdist(dat, method="euclidean")
mds <- isoMDS(dist, initMDS(dist), maxit=200, trace=FALSE)
mds <- postMDS(mds, dist)
mds$stress
```

```
[1] 0.007561136
```

```
plot(mds$points, type="n")
text(mds$points, row.names(dat))
```

Wird eine zufällige Startkonfiguration benutzt, dann ist es normalerweise angeraten, mehrere Versuche durchzuführen und das jeweils beste Ergebnis separat abzuspeichern. Die Funktion `metaMDS` aus dem Paket `vegan` führt eine solche MDS mit mehrfachem zufälligen Start automatisch durch und gibt das beste Ergebnis dieser Versuche aus. Da `metaMDS` allerdings eher auf ökologische Daten zugeschnitten ist, findet sich das dazu passende Beispiel etwas weiter unten im Text.

Einen Shepard-Plot erhält man mit:

```
stressplot(mds, dist(dat))
mds$stress
```

```
[1] 0.007561136
```

und zum Vergleich für einen schlechten Fit sollen einmal 10 Dimensionen mit je 10 Zufallszahlen $U(0,1)$ dargestellt werden:

```
set.seed(123)
rdat <- matrix(runif(100), 10)
mds  <- isoMDS(d<-vegdist(rdat, method="euclid"), trace=FALSE)
stressplot(mds, d, pch=16)
mds$stress
```

```
[1] 10.22633
```

Der Shepard-Plot Vergleicht trägt die Ordinationsdistanzen gegen die ursprünglichen Distanzen auf. Der Fit ist als Treppenkurve eingezeichnet. Je näher die Punkte an der Treppenkurve liegen umso besser ist der Fit. Die zusätzlich angegebenen Bestimmtheitsmaße ($R^2$) sind fast immer sehr hoch und deshalb mit Vorsicht zu werten, Details siehe OKSANEN (2010).

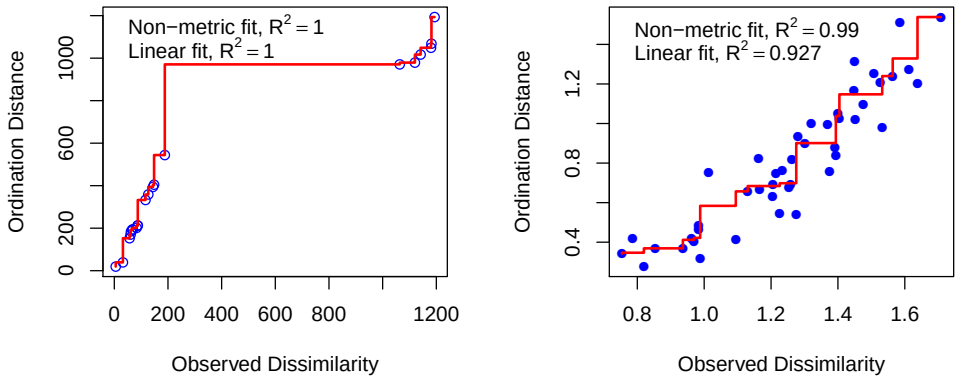



Abbildung 10.2: Shepard-Plot für den Seendatensatz (links) und zum Vergleich für gleichverteilte Zufallszahlen.

### Hierarchische Clusteranalyse

Die hierarchische Clusteranalyse ist sehr einfach anzuwenden. Da die Abstandsberechnung während der Clusterung wiederholt durchgeführt wird, muss die Funktion zur Abstandsberechnung (`dist` oder `vegist`) direkt mit übergeben werden.

Für den Seendatensatz ist wegen der unterschiedlichen Maßeinheiten eine Standardisierung erforderlich, als Abstandsmaß wählen wir den euklidischen Abstand (da es sich um metrische Daten handelt), als Agglomerationsalgorithmus ist die Methode nach Ward gut geeignet. Zum Vergleich verwenden wir drei weitere Agglomerationsmethoden:

```
par(mfrow=c(2,2))
plot(hclust(dist(scale(dat), method="euclid"), method="complete"),
  main="Complete")
plot(hclust(dist(scale(dat), method="euclid"), method="single"),
  main="Single")
plot(hclust(dist(scale(dat), method="euclid"), method="average"),
  main="Average")
plot(hclust(dist(scale(dat), method="euclid"), method="ward.D"),
  main="Ward")
```

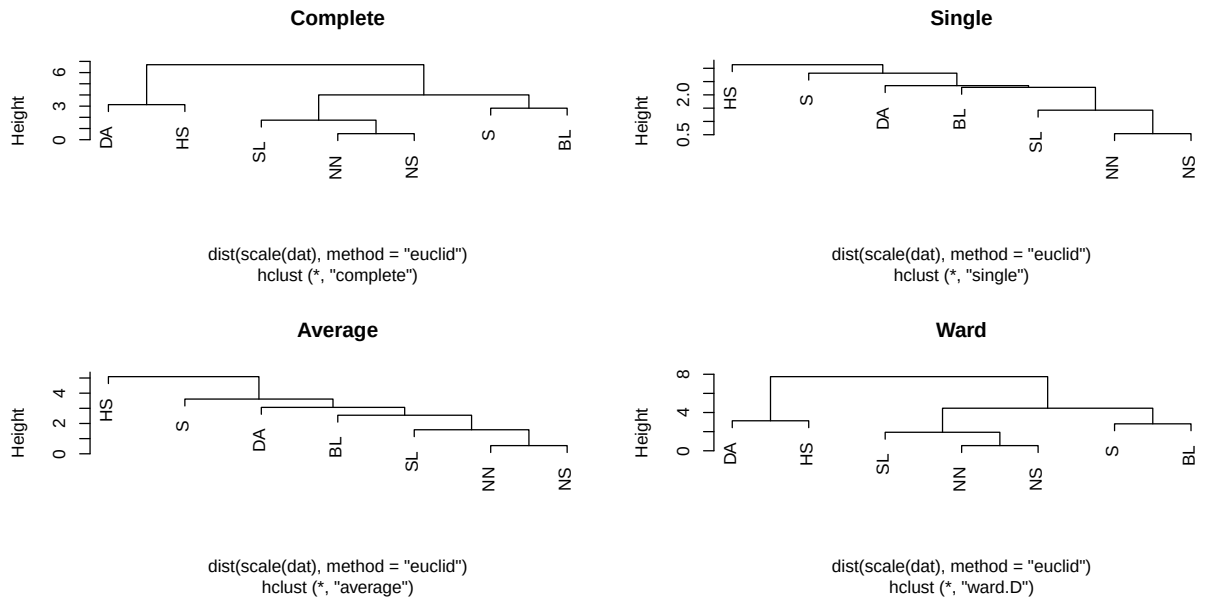



Abbildung 10.3: Clusteranalyse für den Seendatensatz.

### Clusteranalyse mit Mahalanobis-Abstand

Wenn der Mahalanobis-Abstand verwendet werden soll, dann reicht es nicht aus, diesen am Anfang zu berechnen und einfach die Abstandsmatrix an `hclust` zu übergeben, da das Abstandsverfahren auch während der Clusterung benötigt wird. Allerdings ist weder in `dist` noch in `vegdist` ein Mahalanobisabstand enthalten. Eine vollkommen „legale“ Alternative besteht darin, die Original-Koordinaten so zu transformieren, dass der euklidische Abstand der transformierten Objekte dem Mahalanobis-Abstand der untransformierten

Objekte entspricht. Hierbei hilft eine Eigenwertzerlegung, die am einfachsten über eine PCA (ohne Standardisierung) durchgeführt werden kann. Anschließend werden alle Hauptkomponenten einheitlich skaliert, was dazu führt, dass auch die sonst unwichtigen höheren Komponenten ein großes Gewicht erhalten. Für das Seenbeispiel bedeutet das, dass die eigentlich unwichtigen Unterschiede (und die Messfehler) eine sehr hohe Bedeutung erhalten und sich ein ggf. unerwartetes Bild ergibt: (Abb. 10.4, links):

```
pc <- prcomp(dat)
pcdata <- as.data.frame(scale(pc$x))
cl <- hclust(dist(pcdata), method="ward.D")
plot(cl, main="PC 1...7")
```

Zur Demonstration, dass das unerwartete Verhalten auf die „hochgewichteten“ letzten Hauptkomponenten zurückzuführen ist, lassen wir einmal die letzten beiden Komponenten willkürlich weg und erhalten das erwartete Bild (Abb. 10.4, rechts):

```
plot(hclust(dist(pcdata[,1:4]), method="ward.D"), main="PC 1...4")
```

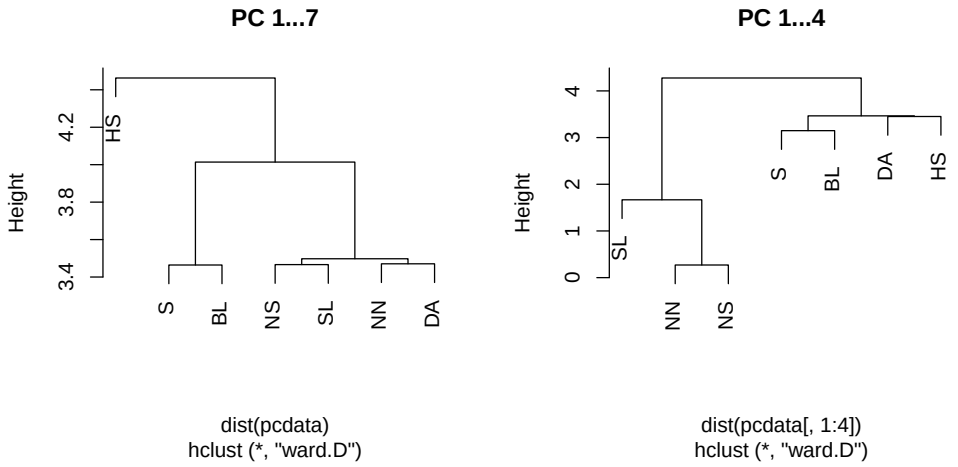



Abbildung 10.4: Clusteranalyse mit Hauptkomponenten mit allen (links) bzw. nur mit den ersten vier Hauptkomponenten (rechts).

Der Mahalanobis-Abstand ist also eher für Probleme geeignet, bei denen andere Verfahren keine genügende Auftrennung erzielen, was z.B. bei der Analyse von Peaks aus chemischen Analysen der Fall sein könnte. Das Fazit ist, dass die Wahl eines geeigneten Abstandsmaßes sehr sorgfältig vorgenommen werden muss, da je nach Abstandsmaß völlig unterschiedliche Ergebnisse möglich sind.

#### Kombination aus NMDS und k-Means-Clusterung

Die NMDS kann sinnvoll mit einer Clusteranalyse kombiniert werden, z.B. indem man die gebildeten Gruppen in einem Diagramm farbig markiert. Besonders einfach ist eine Kombination aus NMDS mit der nichthierarchischen k-Means-Clusterung. Bei k-Means wird die Anzahl der zusammenzufassenden Cluster apriori vorgegeben werden, z.B. `centers=3` entsprechend dem Ergebnis der hierarchischen Clusterung (Abb. 10.5):

```
dat <- read.table("http://www.simecol.de/data/seen_bb.txt",header=TRUE)
row.names(dat) <- dat[,1]
```

```
dat <- dat[,-1]
mds <- isoMDS(vegdist(dat, method="bray"))
```

```
initial  value 1.486552
iter   5 value 0.589242
iter  10 value 0.012067
iter  10 value 0.000134
iter  10 value 0.000000
final  value 0.000000
converged
```

```
km <- kmeans(dat, centers = 3)
plot(mds$points, type="n")
text(mds$points, row.names(dat), col=km$cluster)
```

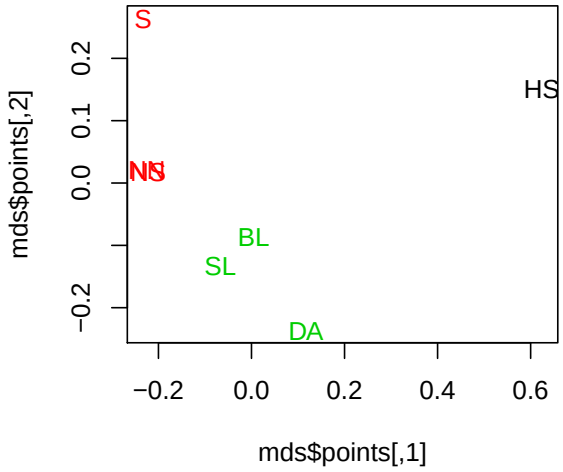



Abbildung 10.5: Kombination aus NMDS und k-Means Clusteranalyse.

### 10.6.2 Beispiel 2: Ein Datensatz aus dem vegan-Paket

Der folgende Abschnitt ist eine Kurzzusammenfassung der in der Onlinehilfe und im **vegan**-Tutorial (OKSANEN, 2010) enthaltenen Beispiele. Der Datensatz besteht aus zwei Matrizen, der Artenliste (Community Matrix) `varespec` und der Umweltmatrix `varechem`. Er ist als Testdatensatz direkt im **vegan**-Paket enthalten entstammt einer Publikation von VÄRE *et al.* (1995):

```
library(vegan)
data(varechem)
data(varespec)
```

#### NMDS

Eine MDS erhält man vollautomatisch mittels:

```
mds <- metaMDS(varespec)
plot(mds, type = "t")
```

wobei die Automatik aus folgenden Schritten besteht:

1. Wenn „nötig“ wird wurzeltransformiert und eine Wisconsin-Doppelstandardisierung durchgeführt (zuerst Artweise: Abundanzen durch maximale Abundanz der Art, anschließend Probenahmestellen: Abundanz durch jeweilige Gesamtabundanz),
2. Es wird die Bray-Curtis Dissimilarity verwendet,
3. Es werden mehrere zufällige Starts durchgeführt und das Ergebnis mit einem Procrustes-Test mit der jeweils vorherigen besten Lösung verglichen,
4. Zur besseren Interpretation wird eine Rotation durchgeführt, so dass die größte Varianz der site-scores auf 1. Achse liegt,
5. Es wird eine Skalierung durchgeführt, so dass 1 Einheit 50% der Community-Ähnlichkeit von der Replikat-„Ähnlichkeit“ entspricht.
6. Die Spezies-Scores werden der Endkonfiguration als gewichtete Mittel der Umweltvariablen hinzugefügt. Es ergibt sich somit ein Biplot.

Obwohl die meisten Schritte absolut sinnvoll sind rate ich dringend dazu, die Transformation und die Standardisierung in die eigene Verantwortung zu übernehmen um reproduzierbare Ergebnisse zu erhalten, z.B. ohne Standardisierung und Transformation:

```
mds <- metaMDS(varespec, distance = "bray", autotransform = FALSE)
```

oder mit Wurzel und Wisconsin-Transformation:

```
mds <- metaMDS(wisconsin(sqrt(varespec)), distance = "bray",
  autotransform = FALSE, trace=FALSE)
mds
```

```
Call:
metaMDS(comm = wisconsin(sqrt(varespec)), distance = "bray",      autotransform =

global Multidimensional Scaling using monoMDS

Data:     wisconsin(sqrt(varespec))
Distance: bray

Dimensions: 2
Stress:     0.1825658
Stress type 1, weak ties
Two convergent solutions found after 20 tries
Scaling: centring, PC rotation, halfchange scaling
Species: expanded scores based on 'wisconsin(sqrt(varespec))'
```

Mit Hilfe von trace kann man die Anzeige von Zwischenergebnissen einschalten (sehr zu empfehlen) oder ausschalten (damit das Skript nicht zu lang wird).

Ob eine Transformation sinnvoll ist oder nicht, dazu kann man ja einmal die Automatik „befragen“.

Für die grafische Darstellung existiert neben der Spezialfunktion `ordiplot` auch eine generische Plotfunktion:

```
plot(mds)
plot(mds, type="t")
plot(mds, display="sites")
plot(mds, display="species")
```

Für eine detaillierte Beschreibung wird auf die online-Hilfe und insbesondere das **vegan**-Tutorial verwiesen (OKSANEN, 2010). Dort finden wir auch näheres zur 3D-Grafik, z.B.:

```
mds <- metaMDS(varespec, distance = "bray", autotransform = FALSE, k = 3)
ordirgl(mds, type = "t", col = "yellow")
orgltext(mds, text = names(varespec), display = "species",col = "cyan")
axes3d()
```

Hierbei ermöglicht der Parameter `k=3` eine dreidimensionale NMDS.

### Vector Fitting

Die Wirkung von Umweltvariablen lässt sich mit Hilfe des sogenannten *vector fitting* untersuchen (Abb. 10.6), hierbei ist ein Permutationstest inklusive (z.B. mit 1000 Randomisierungen):

```
mds <- metaMDS(varespec, trace = FALSE)
ef <- envfit(mds, varechem, permu = 1000)
ef
```

```
***VECTORS

             NMDS1    NMDS2     r2   Pr(>r)
N        -0.05726 -0.99836 0.2537 0.046953 *
P         0.61966  0.78487 0.1938 0.100899
K         0.76639  0.64238 0.1809 0.096903 .
Ca        0.68513  0.72842 0.4119 0.002997 **
Mg        0.63247  0.77459 0.4270 0.002997 **
S         0.19130  0.98153 0.1752 0.125874
Al       -0.87164  0.49014 0.5269 0.000999 ***
Fe       -0.93606  0.35184 0.4450 0.001998 **
Mn        0.79873 -0.60169 0.5231 0.000999 ***
Zn        0.61752  0.78656 0.1879 0.097902 .
Mo       -0.90309  0.42946 0.0609 0.519481
Baresoil  0.92494 -0.38012 0.2508 0.056943 .
Humdepth  0.93286 -0.36024 0.5200 0.000999 ***
pH       -0.64803  0.76161 0.2308 0.073926 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Permutation: free
Number of permutations: 1000
```

```
plot(mds, type = "t")
plot(ef, p.max = 0.1)
```

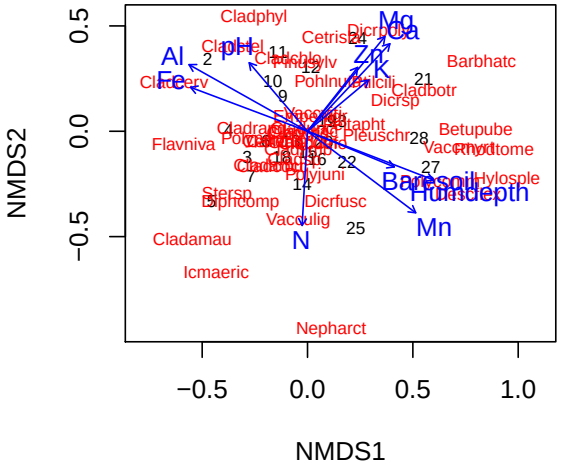



Abbildung 10.6: NMDS des varespec-Datensatzes mit gefitteten Umweltvektoren.

Das Ergebnis ist ein Triplot. Im obigen Fall wurden zunächst naiv alle möglichen Umweltvariablen gefittet, beim Plot aber nur die Variablen mit einem p-Wert < 0.1 eingezeichnet. Besser ist es, die interessierenden Variablen a-priori zu spezifizieren und mit Hilfe einer Modellformel anzugeben:

```
ef <- envfit(mds ~ Al + Ca, varechem, permu = 1000)
ef
plot(mds, type = "t")
plot(ef)
```

Ein Problem des Vector-Fitting ist dass hierbei eine lineare Beziehung zwischen den Umweltvektoren und der Ordination angenommen wird. Das ist nicht immer gegeben, zumal die NMDS ja eine nichtmetrische Verzerrung vornimmt.

Die Funktion `ordisurf` (*surface fitting*) benutzt einen GAM-Ansatz (generalized additive model) um eine eventuelle Nichtlinearität darzustellen. Im folgenden Beispiel sind Vector Fitting und Surface Fitting für die beiden Varialen Al und Ca dargestellt (Abb. 10.7):

```
ef <- envfit(mds ~ Al + Ca, varechem)
plot(mds, display = "sites", type = "t")
plot(ef)
tmp <- with(varechem, ordisurf(mds, Al, add = TRUE))
with(varechem, ordisurf(mds, Ca, add = TRUE,
col = "green4"))
```

```
Family: gaussian
Link function: identity
```

```
Formula:
y ~ s(x1, x2, k = 10, bs = "tp", fx = FALSE)

Estimated degrees of freedom:
4.72  total = 5.72

REML score: 156.6552
```

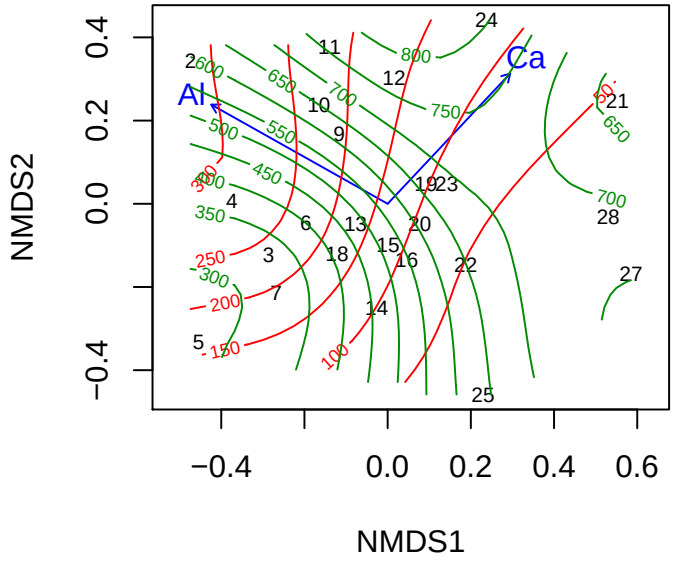



Abbildung 10.7: NMDS des varespec-Datensatzes mit gefitteten Umweltvektoren und GAM-Isolinien für Al (rot) und Ca (grün).

Mit `vis.gam(tmp)` ist auch eine 3D-Darstellung möglich.

### Mantel-Test

Mit Hilfe des Mantel-Tests kann man untersuchen ob eine Beziehung zwischen den Umweltfaktoren und der Artenliste besteht:

```
veg.dist <- vegdist(varespec)
env.dist <- vegdist(scale(varechem), "euclid")
mantel(veg.dist, env.dist)
```

```
Mantel statistic based on Pearson's product-moment correlation

Call:
mantel(xdis = veg.dist, ydis = env.dist)

Mantel statistic r: 0.3047
```

```
      Significance: 0.002

Upper quantiles of permutations (null model):
  90%   95% 97.5%   99%
0.110 0.138 0.160 0.194
Permutation: free
Number of permutations: 999
```

Man beachte, dass für die Artenliste die Bray-Curtis dissimilarity (default) und für die Umweltfaktoren der euklidische Abstand (nach Skalierung) verwendet wurde. Anstelle von standardisierten Umweltvariablen ist es auch möglich, eine Distanzmatrix der wichtigsten Hauptkomponenten zu verwenden.

Zur Visualisierung kann man die Distanzen direkt gegeneinander plotten:

```
plot(veg.dist, env.dist)
```

wobei die Beziehung mehr oder weniger linear oder monoton aussehen sollte.

#### BIOENV

Bioenv ist eine Methode zur Auswahl der besten Teilmenge von Umweltvariablen, so dass deren euklidische Distanz die maximale Rangkorrelation mit der Arten-Disssimilaritätsmatrix aufweist:

```
sol <- bioenv(wisconsin(varespec) ~ log(N) +
P + K + Ca + pH + Al, varechem)
summary(sol)
```

```
                    size correlation
P                      1      0.2513
P Al                   2      0.4004
P Ca Al                3      0.4005
P Ca pH Al             4      0.3619
log(N) P Ca pH Al      5      0.3216
log(N) P K Ca pH Al    6      0.2822
```

Im vorliegenden Fall ist die Teilmenge also P und Al bzw. P, Ca und Al.

#### CCA

Die Kanonische Korrespondenzanalyse ist eine Zweimatrixmethode, die zum einen den Chiquadrat-Abstand benutzt und zum anderen zwei Arten von Achsen produziert. Zum einen sind es die sogenannten *constrained*-Achsen die nur die Anteile der Gesamtinformation (als Total Inertia bezeichnet) darstellen, die durch die Umweltvariablen erklärbar sind und zum anderen die übrigen *unconstrained*-Achsen. Man beachte also, dass im Unterschied zur NMDS in der Grafik normalerweise nur der Informationsanteil der Artenliste dargestellt wird der durch die Umweltmatrix erklärbar ist.

```
vare.cca <- cca(varespec, varechem)
vare.cca
```

```
Call: cca(X = varespec, Y = varechem)

              Inertia Proportion Rank
```

```
Total          2.0832     1.0000
Constrained    1.4415     0.6920   14
Unconstrained  0.6417     0.3080    9
Inertia is scaled Chi-square

Eigenvalues for constrained axes:
  CCA1   CCA2   CCA3   CCA4   CCA5   CCA6   CCA7   CCA8   CCA9  CCA10  CCA11
0.4389 0.2918 0.1628 0.1421 0.1180 0.0890 0.0703 0.0584 0.0311 0.0133 0.0084
 CCA12  CCA13  CCA14
0.0065 0.0062 0.0047

Eigenvalues for unconstrained axes:
    CA1     CA2     CA3     CA4     CA5     CA6     CA7     CA8     CA9
0.19776 0.14193 0.10117 0.07079 0.05330 0.03330 0.01887 0.01510 0.00949
```

```
plot(vare.cca)
```

Die Anzahl der *constrained axes* entspricht der Anzahl der Umweltvariablen. Das heißt dass eine große Zahl von Umweltvariablen dazu führt, dass immer mehr Freiheitsgrade vorhanden sind und sich die CCA letztlich der CA annähert. Viele constraints bedeuten praktisch „keine constraints“. Es ist deshalb sinnvoll, die möglichen Erklärungsvariablen a-priori zu spezifizieren:

```
vare.cca <- cca(varespec ~ P + Ca + Al, varechem)
vare.cca
```

```
Call: cca(formula = varespec ~ P + Ca + Al, data = varechem)

              Inertia Proportion Rank
Total          2.0832     1.0000
Constrained    0.5243     0.2517    3
Unconstrained  1.5589     0.7483   20
Inertia is scaled Chi-square

Eigenvalues for constrained axes:
  CCA1   CCA2   CCA3
0.3453 0.1489 0.0301

Eigenvalues for unconstrained axes:
   CA1    CA2    CA3    CA4    CA5    CA6    CA7    CA8
0.3694 0.2230 0.2049 0.1793 0.1342 0.1008 0.0853 0.0772
(Showing 8 of 20 unconstrained eigenvalues)
```

```
plot(vare.cca)
```

Eine RDA wird nach dem gleichen Muster durchgeführt. Da hier auch für die Artenmatrix der euklidische Abstand verwendet wird ist sie heute nur noch in Ausnahmefällen angeraten und eher für Probleme mit zwei Matrizen mit metrischen Daten anzuwenden.

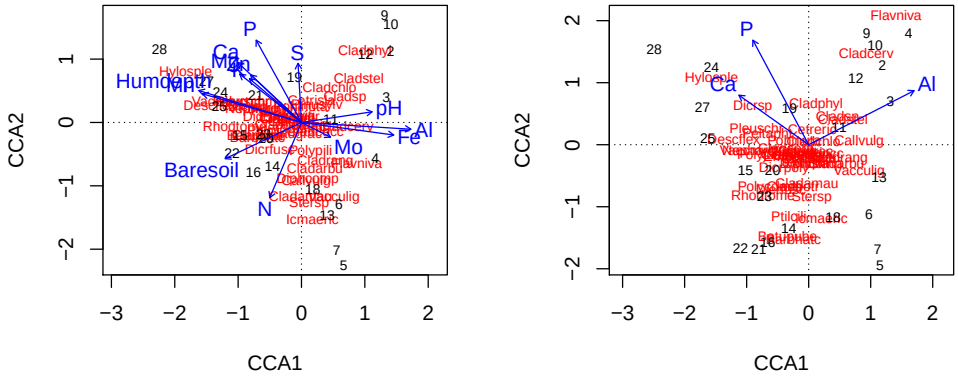



Abbildung 10.8: CCA des varespec-Datensatzes mit der kompletten Umweltmatrix als constraint (links) bzw. nur P, Ca, Al (rechts).

Tabelle 10.3: Benthische Besiedlung zweier Bäche in der Nähe von Dresden (site: ID der Probenahmestelle, Habitat: p=pool, r riffle, Bach: g= Gauernitzbach, t= Tännichtgrundbach, Hochwasser: v= vorher, n= nachher, Quelle: Carola Winkelmann, unveröffentlichte, für das Tutorial aggregierte Originaldaten, näheres zum Experiment, siehe WINKELMANN *et al.* 2007)

| site | Habitat | Bach | Hochwasser | Mollusca | Diptera | Baetis | Plecoptera | Coleoptera | Turbellaria | Heptageniidae | Ephemeroptera | Gammarus | Trichoptera | Acari | Nematoda | Oligochaeta |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GP9 | p | g | n | 3 | 165 | 91 | 14 | 6 | 3 | 9 | 136 | 256 | 45 | 6 | 0 | 11 |
| GR9 | r | g | n | 31 | 438 | 728 | 31 | 728 | 11 | 31 | 0 | 65 | 367 | 3 | 0 | 503 |
| TP9 | p | t | n | 0 | 26 | 3 | 20 | 9 | 0 | 3 | 20 | 119 | 40 | 0 | 0 | 23 |
| TR9 | r | t | n | 0 | 11 | 6 | 37 | 11 | 0 | 0 | 3 | 68 | 26 | 0 | 0 | 23 |
| GP8 | p | g | v | 23 | 913 | 31 | 14 | 3 | 9 | 6 | 26 | 901 | 37 | 3 | 20 | 0 |
| GR8 | r | g | v | 225 | 1066 | 310 | 199 | 461 | 48 | 91 | 23 | 688 | 600 | 26 | 0 | 284 |
| TP8 | p | t | v | 17 | 2204 | 54 | 117 | 11 | 0 | 11 | 20 | 2525 | 77 | 3 | 3 | 68 |
| TR8 | r | t | v | 3 | 520 | 74 | 762 | 125 | 20 | 65 | 0 | 668 | 173 | 3 | 0 | 267 |
| GP7 | p | g | v | 26 | 247 | 68 | 6 | 3 | 3 | 3 | 20 | 813 | 9 | 6 | 0 | 0 |
| GR7 | r | g | v | 117 | 509 | 290 | 63 | 191 | 6 | 26 | 11 | 682 | 117 | 60 | 9 | 131 |
| TP7 | p | t | v | 26 | 3477 | 17 | 28 | 0 | 0 | 3 | 37 | 2693 | 17 | 0 | 0 | 0 |
| TR7 | r | t | v | 48 | 429 | 57 | 412 | 97 | 9 | 63 | 9 | 808 | 102 | 26 | 6 | 57 |
| GP10 | p | g | n | 3 | 159 | 14 | 31 | 3 | 0 | 48 | 91 | 100 | 23 | 0 | 0 | 17 |
| GR10 | r | g | n | 0 | 68 | 191 | 26 | 51 | 3 | 253 | 0 | 80 | 233 | 3 | 0 | 11 |
| TP10 | p | t | n | 0 | 51 | 0 | 6 | 9 | 0 | 0 | 6 | 71 | 0 | 0 | 0 | 31 |
| TR10 | r | t | n | 0 | 28 | 6 | 40 | 14 | 0 | 0 | 0 | 40 | 54 | 0 | 0 | 0 |

## 10.7 Aufgaben

### 10.7.1 Beispiel „Bach 1“

Im Rahmen einer Makrozoobenthosuntersuchung wurden Arthäufigkeiten in zwei Bächen (Gauernitzbach, Tännichtgrundbach) und unterschiedlichen Habitaten (pool und riffle) ermittelt (WINKELMANN *et al.*, 2007, 2008; SCHMIDT *et al.*, 2009). Ein Teil der Proben wurde vor, ein Teil nach einem starken Hochwasser genommen. Zur Vereinfachung für das Beispiel im Tutorial wurden die Daten in taxonomische Gruppen zusammengefasst (Tabelle 10.3).

Es soll untersucht werden, inwiefern sich Bäche und Habitate unterscheiden und ob das Hochwasser die Besiedlung verändert hat, sowie gegebenenfalls welche taxonomischen Gruppen besonders durch das Hochwasser beeinträchtigt wurden. Protokollieren und werten Sie die Ergebnisse!

### 10.7.2 Lösung

Zunächst werden die benötigten Pakete geladen und der Datensatz eingelesen. Da der Datensatz nicht nur die Artenliste enthält sondern auch „Umwelfaktoren“ (Bach, Hochwasser, Habitat) ist eine gewisse Vorauf-bereitung nötig. Sinnvoll ist es außerdem, die Site-ID als Zeilennamen den Dataframes zuzuweisen.

```
library(vegan)
library(MASS)
dat <- read.table("gauernitz.txt", head=TRUE)
row.names(dat) <- dat$site
env <- dat[c("Habitat", "Bach", "Hochwasser")]
bio <- dat
bio$site <- bio$Habitat <- bio$Bach <- bio$Hochwasser <- NULL
```

In einer ersten Analyse soll eine MDS durchgeführt werden. Wir benutzen zunächst die „Kernroutine“ `isoMDS` direkt und verwenden den Bray-Curtis-Abstand:

```
dist <-vegdist(bio, method="bray")
mds <- isoMDS(dist)
plot(mds$points, type="n")
text(mds$points, as.character(dat$site), col=as.numeric(dat$Hochwasser))
```

Die `metaMDS`-Funktion hat gegenüber dieser Vorgehensweise eine Reihe von Vorteilen (s.o.), wir schalten jedoch die Transformationsautomatik ab. Zusätzlich kann in die resultierende Abbildung ein Vektor-Fit der Umweltvariablen hineinprojiziert werden, nebenbei wird auch gleich noch ein Resamplingtest durchgeführt:

```
## Meta-MDS, vector fitting etc...
mds <- metaMDS(bio, autotransform=FALSE)
efit <- envfit(mds ~ Hochwasser + Bach + Habitat, env, permu = 1000)
efit
plot(mds, type="t")
plot(efit, add=TRUE)
```

Wie erwähnt, kann eine MDS unter Umständen nicht alle wichtigen Informationen in eine Ebene bringen. Wie gut dies geklappt hat, sagt uns der Stress-Wert und der Shepard-Plot (Stressplot). Man beachte dabei dass im Unterschied zu anderen Programmen der Stresswert in Prozent ausgegeben wird:

```
mds$stress
stressplot(mds)
```

Mit Hilfe von Klassifikationsverfahren erhält man eine direkte Darstellung der Abstände zwischen den Objekten, z.B. mit Hilfe einer hierarchischen Clusteranalyse:

```
cP<-hclust(vegdist(bio), method="ward.D")
plot(cP)
```

Es ist auch möglich, Clusteranalyse und MDS miteinander zu kombinieren, entweder mit der Funktion `ordicluster`:

```
plot(mds, display="sites", type="t")
ordicluster(mds, cP, col="blue")
```

oder indem man zunächst eine „günstige Anzahl“ Cluster ermittelt und die Objekte in der MDS entsprechend einfärbt:

```
plot(cP)
rect.hclust(cP, 4)
grp <- cutree(cP, 4)
plot(mds, type="n")
text(mds$points, row.names(bio), col=grp)
```

#### Kanonische Korrespondenzanalyse

Da die Tabelle `env` im vorliegenden Beispiel nur potentiell relevante Umweltfaktoren enthält spricht nichts dagegen, eine CCA mit der kompletten Umweltmatrix durchzuführen:

```
#cc <- cca(bio ~ Habitat + Bach + Hochwasser, env)
## abgekürzte Schreibweise:
cc <- cca(bio ~ ., env)
cc
plot(cc)
ordihull(cc, env$Habitat, col="blue")
#ordispider(cc, env$Habitat, col="blue")
#ordiellipse(cc, env$Habitat, col="blue") # erfordert package ellipse
```

wobei die funktionen `ordihull`, `ordispider` und `ordiellipse` dazu dienen können, zusammengehörige Objekte zu markieren.

Resampling-Tests dienen dazu, die Signifikanz der untersuchten Umweltfaktoren zu testen oder die Zufälligkeit oder Nichtzufälligkeit der Ordinationsachsen zu prüfen. Einen Resamplingtest haben wir im Zusammenhang mit dem Vector Fitting für die MDS oben bereits „nebenbei“ durchgeführt, ein noch einfacherer Resamplingtest (ANOSIM) arbeitet direkt mit der Distanzmatrix:

```
anosim(dist, dat$Hochwasser)
```

Besser als ANOSIM sind allerdings Tests auf Basis einer bereits durchgeführten Ordination. Im Folgenden testen wir zunächst die Signifikanz einer gesamten Ordination (d.h. alle Umweltfaktoren), dann die Terme (einzelne Umweltfaktoren) und anschließend die Ordinationsachsen:

```
anova(cc)
anova(cc, by="terms")
anova(cc, by="axis")
```

Zur Identifikation kann auch eine AIC-basierte schrittweise Modellselektion verwendet werden:

```
step(cc)
```

Eine weitere Möglichkeit bietet die partielle CCA. Nehmen wir einmal an, der Unterschied der Habitate ist uns von vornherein bekannt und sein Einfluss soll deshalb von vornherein als Kovariate betrachtet, also „herausgerechnet“ werden:

```
pcc <- cca(bio ~ Hochwasser + Bach + Condition(Habitat), env)
pcc
plot(pcc)
```

#### Procrustes-Analyse

Die Procrustes-Analyse dient ganz allgemein zum Vergleich von zwei Ordinationen. Im folgenden vergleichen wir einmal die obige PCC (Habitat als Kovariate) mit der normalen CCA (Habitat war ein Constraint):

```
proc <- procrustes(cc, pcc)
pproc <- plot(proc)
text(pproc$points, row.names(pproc$points))
```

Dabei zeigen die Pfeile an, in welche Richtung sich die Ordination verändert hat.

Es ist auch möglich, die Ordination für eine in zwei Teilmengen aufgeteilte Situation vor und nach dem Hochwasser zu vergleichen und dazu einen Signifikanztest mit der gesamten Ordination zu machen (`protest`):

```
biov <- subset(bio, env$Hochwasser == "v")
bion <- subset(bio, env$Hochwasser == "n")
ccv <- cca(biov)
ccn <- cca(bion)
proc <- procrustes(ccn, ccv)
pproc <- plot(proc)
text(pproc$points, row.names(pproc$points))
protest(ccv, ccn)
```

**Wichtig:** Die hier gezeigte technische Durchführung zeigt zunächst nur, wie es gemacht werden kann. Viel wichtiger ist es aber darüber nachzudenken **was und wofür** etwas gemacht wurde, d.h. einerseits die Methoden gut zu dokumentieren und vor allem die Ergebnisse tiefgründig zu diskutieren und zu interpretieren.

# 11 Eindimensionale Interpolationsverfahren

## 11.1 Problemstellung

Interpolationsverfahren sind ein in der modernen Datenalanyse unverzichtbare Hilfsmittel. Sie dienen dazu, unbekannte Zwischenwerte aus Messungen einer veränderlichen Variablen in Abhängigkeit von der Zeit (oder dem Raum oder einer anderen Messgröße) zu schätzen. Anwendungsfälle hierfür sind z.B. die Ergänzung einzelner fehlender Werte einer Datenreihe, die Abtastung von Zwischenwerten mit einem feineren Abtastintervall oder die Herstellung einer streng äquidistanten Datenreihe (z.B. tägliche Messung um 9:00) aus einer nicht ganz äquidistanten Reihe (tägliche Messungen zu einem Zeitpunkt zwischen 7:00 und 11:00 Uhr).

In vielen Fällen ist die lineare Interpolation ausreichend, in einigen Fällen werden jedoch Verfahren mit einem glatteren Kurvenverlauf (Kurvenlineal) bevorzugt. Neben der Entstehung von „Brüchen“ besteht ein weiterer Nachteil Nachteil der linearen Interpolation darin, dass auf systematische Weise Extremwerte („Spitzen“, d.h. Minima und Maxima) gekappt werden und sich die Varianz einer Datenreihe dadurch verringert.

Im Folgenden werden drei Verfahren kurz vorgestellt:

- lineare Interpolation,
- Polynome höheren Grades
- stückweise Polynome, insbesondere:
  - kubische Splines,
  - eindimensionale Akima-Interpolation.

Neben eindimensionalen Interpolationsverfahren (eine Unabhängige Variable und eine Abhängige Variable) existieren mehrdimensionale Verfahren (mehrere Unabhängige), z.B. für eine flächenhafte Interpolation wie sie in geografischen Informationssystemen verwendet wird (siehe Kapitel 12). Verfahren, bei denen eine gefundene Kurve nicht notwendigerweise durch alle Datenpunkte führt, nennt man Glättungsverfahren (auch Gleitmittel oder Filter genannt).

Grundsätzlich kann kein Interpolationsverfahren den wirklichen Verlauf zwischen den Messungen rekonstruieren. Interpolationsverfahren setzen deshalb immer eine dem Zweck entsprechend ausreichende Datendichte voraus. Darüber hinaus sollte man immer überlegen ob fehlende Daten nicht per Ausgleichsrechnung (Regression) aus anderen Messungen abgeleitet werden können.

## 11.2 Methodik

Wir beschränken uns im Folgenden zunächst auf eindimensionale Interpolationsverfahren und nennen die abhängige Variable $y$ und die unabhängige Variable $x$, wobei die Datenreihe nach $x$ in aufsteigender Folge sortiert ist.

```
x <- c(0, 1, 1.3, 3.5, 4, 4.7, 6.5, 8, 8.7, 8.9)
y <- c(0.2, 0, 0.5, 0.2, 1.3, 1, 1, 1, 1, 3)
xx <- seq(0, 10, 0.1)
```

### 11.2.1 Lineare Interpolation

Bei der linearen Interpolation werden aufeinanderfolgende Datenpunkte durch Geraden verbunden. Der Algorithmus ist wie folgt:

1. Suche die beiden benachbarten Datenpunkte $x_i$ und $x_{i+1}$, zwischen denen der gesuchte Punkt $x$ liegt.
2. Der gesuchte Wert $\hat{y}$ berechnet sich aus dem gegebenen Wert $x$ über die Geradengleichung:

$$\hat{y} = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \cdot x - x_i$$

Hierbei besteht die Hauptschwierigkeit hierbei in Schritt 1, d.h. einem automatisierten und möglichst effizienten Suchalgorithmus. Lineare Interpolationsverfahren lassen sich bereits mit einfachen Datenbanksystemen automatisieren, in R verwendet man dazu die Funktionen `approx` und `approxfun`:

```
plot(x,y, ylim=c(-3,3), las=1)
lines(approx(x,y, xx), col="red")
lines(approx(x,y, seq(0,10,0.02), method="constant", f=0.5),
  col="blue", lty="dashed")
```

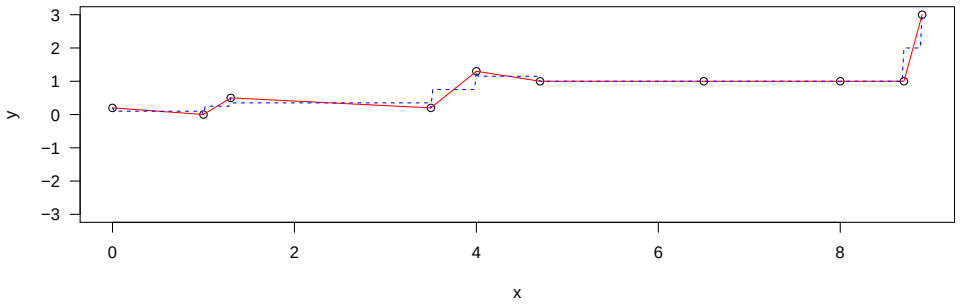



Abbildung 11.1: Lineare Interpolation (durchgezogene Linie) und „konstante Interpolation“ (gestrichelt)

Anstelle einer Interpolation kann es manchen Fällen auch ausreichen, den nächstliegenden Wert zu verwenden oder den Mittelwert zu bilden („konstante Interpolation“, Abb. 11.1, gestrichelt).

### 11.2.2 Polynom-Interpolation

Bei der Polynom-Interpolation passt man eine Funktion der folgenden Form an die Datenreihe an:

$$y < -a_0 \cdot x + a_1 \cdot x + a_2 \cdot x + \cdots + a_p \cdot x$$

Hierbei sind $a_0 \ldots a_p$ die Koeffizienten des Polynoms und $p$ ist der Grad des Polynoms (auch Polynom-Ordnung genannt). Die Anpassung geschieht dabei über lineare Gleichungssysteme (z.B. Lagrange-Polynom):

```
plot(x,y, ylim=c(-3,3), las=1)
m <- lm(y~poly(x,9))
lines(xx,predict(m, newdata=list(x=xx)), col="red")
```

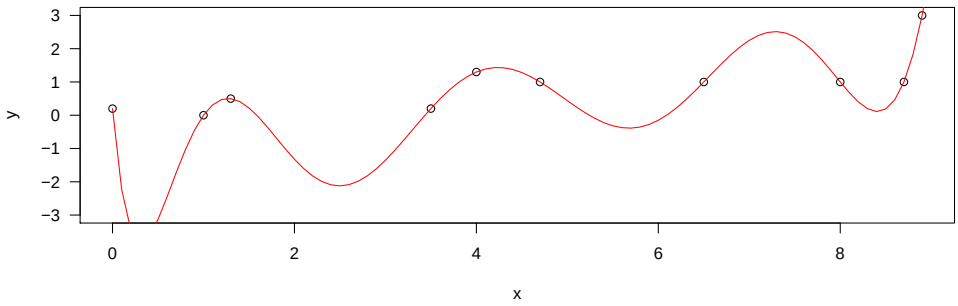



Abbildung 11.2: Polynom 9. Grades

Eine problematische Eigenschaft von Polynomen ist, dass diese mit zunehmendem Grad immer instabiler werden, d.h. sie schwingen stark zwischen den Interpolationspunkten. Passt man Polynome mit einem einfachen Regressionsverfahren (z.B. in einer Tabellenkalkulation) an, treten außerdem häufig numerische Probleme (Rechenungenauigkeiten auf). Dies gilt insbesondere bei Polynomordnungen $> 5$. Aus diesem Grunde werden Lagrange-Polynome höherer Ordnung kaum noch eingesetzt sondern man verwendet stückweise Polynome niedrigerer Ordnung (Splines).

### 11.2.3 Splines

Als Spline (engl.) bezeichnete man ursprünglich biegsame Holzleisten, die man zum Konstruieren glatter Kurven verwendete (z.B. im Schiffs- oder Fahrzeugbau). In der Mathematik versteht man unter Splines stückweise Polynome meist 2. oder 3. Grades (z.B. kubische Splines). Hierbei spielt die Art der „Glattheit“, d.h. Stetigkeit und Differenzierbarkeit an den Stützstellen eine große Rolle. Das Verfahren der Spline-Interpolation wurde bereits im Jahr 1946 veröffentlicht (SCHOENBERG, 1946) und durch AHLBERG *et al.* (1967) populär gemacht.

Die Anpassung von Splines erfolgt mit Hilfe linearer Gleichungssysteme für den gesamten Kurvenzug, d.h. nach Entfernung oder Hinzunahme eines Punktes ist eine Neuberechnung erforderlich. Mit Hilfe geeigneter Lösungsverfahren erfolgt die Anpassung auch für relativ große Datensätze sehr effizient (auf einem Standard-Büro-PC weniger als 1s für 100 000 Datenpaare).

Unter bestimmten Bedingungen neigen auch Splines zum „Schwingen“, insbesondere wenn die $x$-Werte einen sehr unregelmäßigen Abstand aufweisen (Abb. 11.3):

```
plot(x,y, ylim=c(-3,3), las=1)
lines(spline(x,y, n=200), col="red")
```

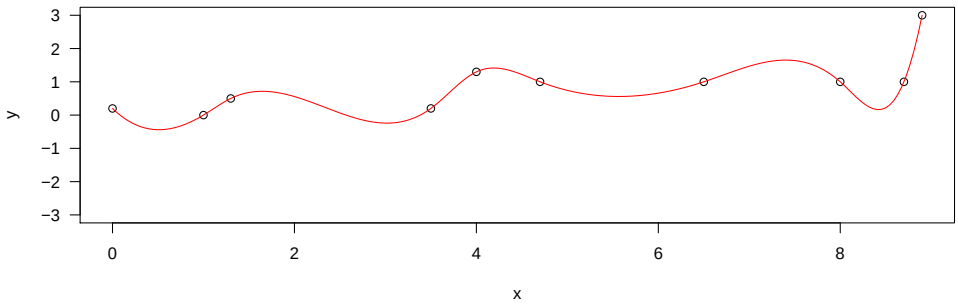



Abbildung 11.3: Kubische Spline-Interpolation.

### 11.2.4 Akima1970-Interpolation

Die AKIMA-Interpolationsmethode wurde mit dem Ziel entwickelt, eine noch näher den Daten folgende Interpolationskurve zu konstruieren. Die AKIMA-Funktionen entsprechen somit „Freihandlinien“, während man Splines mit einem traditionellen Kurvenlineal vergleichen kann, welches auch nur begrenzt enge Biegeradien zulässt.

Von AKIMA wurden zwei Varianten seines Verfahrens veröffentlicht, die *originale* Methode (AKIMA, 1970) und die *verbesserte* Methode (AKIMA, 1991). Beide Methoden basieren wie die Splines auf stückweisen Polynomen, nur wurden die mathematischen Anforderungen hinsichtlich der Glattheit (Differenzierbarkeit und Stetigkeit) gelockert. Die Originalmethode arbeitet mit Polynomen 3. Ordnung, bei der verbesserten Methode sind auch höhere Polynomgrade möglich (Abb. 11.4).

```
library(akima)
plot(x,y, ylim=c(-3,3), las=1)
lines(aspline(x,y, n=200), col="red")
lines(aspline(x,y, n=200, method="improved"), col="blue", lty="dashed")
lines(aspline(x,y, n=200, method="improved", degree=6), col="black",
  lwd=2, lty="dotted")
```

## 11.3 Glättungsverfahren

Von den Interpolationsverfahren zu unterscheiden sind die Glättungsverfahren. Diese werden verwendet, um eine Ausgleichslinie durch einen verrauschten Datensatz zu erhalten, z.B. zur bloßen Visualisierung des „Trends“ in einem Diagramm, zur Abtastung des mittleren Verlaufes oder auch zur Schätzung von Vertrauensintervallen.

### 11.3.1 Überblick

In den letzten Jahrzehnten wurden hierzu eine ganze Reihe unterschiedlicher Verfahren entwickelt (insbesondere aus er Klasse der *kernel density estimators* (Kerndichteschätzer), von denen hier nur eine pragmatische Auswahl kurz gezeigt werden kann.

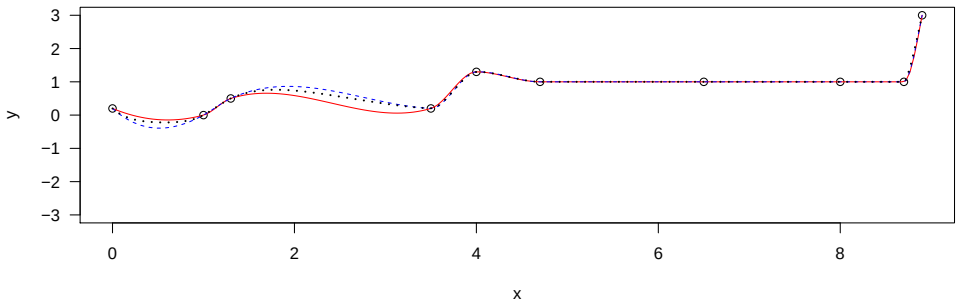



Abbildung 11.4: AKIMA-Interpolation, Originalmethode (durchgezogene Linie) und verbesserte Methode mit Polynomgrad 3 (gestrichelt) bzw. 6 (gepunktet).

Viele Verfahren sind darüber hinaus nicht auf Zeitreihen beschränkt, sondern sie lassen sich auch für mehrdimensionale Probleme anwenden und bieten Möglichkeiten zur Schätzung von Vertrauensintervallen (z.B. die GAM's *generalized additive models*, siehe R packages `gam` und `mgcv`). Weitere Verfahren sind im Gebiet der Geostatistik zu finden.

### 11.3.2 Beispiele

Wie im vorangegangenen Fall erzeugen wir uns auch hier wieder einen Testdatensatz:

```
x <- 1:100
y <- rnorm(1:100) + 3 * sin(x/10)
```

und wenden nacheinander verschiedene Glättungsfunktionen an:

```
plot(x, y, las=1)
## Lowess-Smoother
panel.smooth(x, y, col.smooth="black")
## Loess-Smoother = verbesserter Lowess
lines(loess.smooth(x, y), col="green")
## Dieser Smoother lässt sich gut tunen
lines(loess.smooth(x, y, family="gaussian", evaluation=100), col="blue")
lines(loess.smooth(x, y, span=0.1, evaluation=100), col="red")
## Supersmoother, macht tuning meist automatisch
lines(supsmu(x,y), lwd=2, col="brown")
```

Die Abbildung 11.5 zeigt ein paar Beispiele für unterschiedliche Glättungsfunktionen. Hierbei nutzt die Funktion `panel.smooth` den originalen Lowess-Glätter. Eine verbesserte Variante ist der Loess-Glätter (VENABLES and RIPLEY, 1999), an dem zusätzlich einige der auch bei Lowess vorhandenen Einstellungsmöglichkeiten gezeigt sind. Hierbei ist `loess.smooth` eine benutzerfreundliche Schnittstelle zur Basisfunktion `loess`. Der Parameter `span` bestimmt den Glättungsgrad und `evaluation` bestimmt die Anzahl der auszugebenden Punkte. Sollen statt einer betimmten Anzahl die y-Werte an bestimmten Stellen von x abgetastet werden, muss die Basisfunktion `loess` verwendet werden. Diese erlaubt über ein sogenanntes Formelinterface auch den mehrdimensionalen Fall, d.h. bis zu 4 unabhängige Variablen.

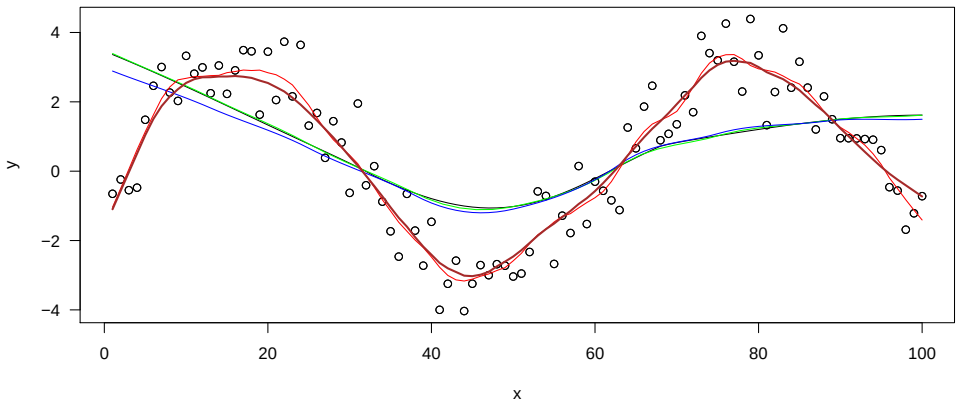



Abbildung 11.5: Beispiel für Glättungsfunktionen (schwarz: Lowess-Smoother, grün, blau, rot: Loess, braun, dick: Supersmoother

Des Super-Smoother (`supsmu`)ist eine weitere nützliche Glättungsfunktion. Ihr Vorteil besteht unter anderem darin, dass die Glattheit standarmäßig automatisch durch ein Cross-Validierungs-Verfahren bestimmt wird. Damit dieses auch gut funktioniert ist im allgemeinen eine Anzahl von mindestens 40 Punkten erforderlich.

# 12 Graphische Darstellung räumlicher oder zeitlich-räumlicher Daten

In der Ökologie und in der ökologischen Modellierung müssen häufig Messergebnisse oder Simulationsdaten mit zweidimensionalem Flächenbezug dargestellt werden, z.B. bei einer vegetationskundlichen Kartierung eines Untersuchungsgebietes oder bei einem Horizontalsurvey (z.B. Echolotung) eines Sees oder einer Talsperre. Ein ähnliches Problem tritt auch auf, wenn Profildaten mit eindimensionalem Raumbezug in ihrer zeitlichen Änderung dargestellt werden müssen, z.B. der zeitliche Verlauf der thermischen Schichtung eines Untersuchungsgewässers. In beiden Fällen liegt somit letztlich eine dreidimensionale Datenstruktur vor. Hierbei werden die beiden räumlichen Koordinaten bzw. eine räumliche und eine zeitliche Koordinate als unabhängige Variablen ($x$ und $y$) betrachtet. Die eigentlichen Messgrößen (z.B. Abundanz des Pflanzenbewuchses oder Phosphorkonzentration im Sediment) sind dementsprechend die abhängigen Größen ($z$), wobei diese entweder einzeln (univariate Verfahren) oder kombiniert (multivariate Verfahren) statistisch ausgewertet werden können. Dementsprechend werden manchmal auch die Koordinaten $x$ und $y$ als Matrix $\mathbf{X}$ und die abhängigen Variablen als Matrix $\mathbf{Y}$ bezeichnet.

## 12.1 Grundlagen

Zur Darstellung von dreidimensionalen Daten werden häufig farbige Kartendarstellungen, Kontourlinien (Isoplethen) oder auch dreidimensionale Projektionsdarstellungen (3D-Gebirge) genutzt. Diesen Darstellungsarten ist gemeinsam, dass zur Konstruktion ein regelmäßiges zweidimensionales Gitter benötigt wird. Liegt eine solche Datenstruktur bereits vor, können die Standard-R-Funktionen `image`, `contour` oder `persp` meistens direkt angewendet werden.

Beim Vorliegen unregelmäßig verteilter Messungen kann man entweder spezielle Darstellungsarten verwenden (z.B. aus dem Paket `scatterplot3d`) oder es muss in einem ersten Schritt ein regelmäßiges Gitter erzeugt werden. Erst in einem zweiten Schritt folgt dann die eigentliche Darstellung.

Zur Erzeugung des regelmäßigen Gitters existieren sehr unterschiedliche, zum Teil sehr ausgefeilte Verfahren, die ihrerseits oftmals in unterschiedlichen Implementationsvarianten verfügbar sind. So finden sich z.B. in mindestens sechs R-Packages Funktionen für das Kriging (`spatial`, `sgeostat`, `geoR`, `geoRglm`, `fields`, `RandomFields`) und in anderen Paketen noch weitere geostatistische Verfahren.

Bei räumlich-zeitlichen Daten (Profildaten) kann man zuweilen durchaus eindimensionale Interpolationsverfahren anwenden, z.B. kann man für einen Jahresgang von Temperaturprofilen in einem See zunächst die Vertikalprofile äquidistant interpolieren (z.B. in 1 m-Schrittweite) und anschließend für jede Tiefenstufe eine zeitliche Interpolation (z.B. bei 7 bis 14 tägigen Daten auf Tagesschrittweite) durchführen.

In der Regel wird jedoch eine zweidimensionale Interpolation benutzt. Im einfachsten, von manchen Grafikprogrammen benutzten, Fall wird für die darzustellende abhängige Variable $z_{i,j}$ für jeden Gitterpunkt ein ungewichteter oder mit Hilfe einer einfachen Abstandsfunktion gewichteter Mittelwert aus den $k$ (z.B. 4) nächsten Werten von $z$ gebildet.

Eine Alternative ist es, eine zweidimensionale Regressionsfunktion, z.B. Trendflächen oder polynomiale Flächen, Splines oder Akima-Funktionen anzupassen. Hierbei werden die Messwerte entweder als „fehlerfrei“ (die Fläche berührt alle Datenpunkte) oder als fehlerbehaftet betrachtet, was zu einer Glättung führt.

In vielen Fällen ist es empfehlenswert, ein universelles geostatistisches Verfahren anzuwenden, das „Kriging“. Hierbei wird unter anderem untersucht, inwiefern nahe beieinanderliegende Messwerte voneinander abhängig, also korreliert sind. Man beschreibt diesen Zusammenhang durch ein Korrelogramm, das die Korrelation der $z$-Werte in Abhängigkeit vom räumlichen Abstand zeigt oder einem Variogramm, d.h. einer Funktion der Varianz der abhängigen Variablen $z$ von der räumlichen Entfernung.

Das Gebiet der Geostatistik ist außerordentlich vielseitig und umfangreich und kann im Folgenden nur anhand von relativ einfachen Beispielen vorgestellt werden. Für eine nähere Beschäftigung mit dem Thema sollte in jedem Fall weiterführende Literatur (z.B. ARMSTRONG, 1998; WACKERNAGEL, 1998) herangezogen werden.

## 12.2 Eindimensionale Interpolation

Bei dieser zugegebenermaßen sehr pragmatischen Methode zur Konstruktion eines rechteckigen Gitters werden die Daten zunächst in einer Dimension (z.B. in $x$-Richtung) in einen äquidistanten Abstand gebracht und anschließend in der anderen Dimension ($y$-Richtung). Ein solches Verfahren bietet sich dann an, wenn

- die Daten in einer Richtung viel dichter vorliegen als in der anderen Richtung oder
- wenn $x$ und $y$ in verschiedenen Dimensionen vorliegen oder
- wenn (wie im untenstehenden Beispiel der Wassertiefe) die Daten bereits in einer Dimension äquidistant vorliegen und nur in einer Dimension (z.B. der Zeit) interpoliert werden muss.

### 12.2.1 Implementation in R

Zur linearen Interpolation kann die R-Funktion `approx` verwendet werden, für eine Spline-Interpolation steht die Funktion `spline` zur Verfügung. Beide Funktionen erwarten einen Vektor mit unabhängigen $x$-Werten und einen Vektor mit einer abhängigen Größe $y$. Hierbei ist zu beachten, dass es sich in unserem Fall bei der abhängigen Größe ja eigentlich um die 3. Dimension, also die $z$-Daten handelt.

Manchmal soll jedoch keine eigentliche Interpolation, sondern eine Glättung durchgeführt werden, z.B. mit den Funktionen `ksmooth`, `loess.smooth`, `smooth.spline` oder `supsmu` aus dem `modreg`-Package.

### 12.2.2 Beispiel: Vertikalprofile der Temperatur in einem Gewässer

Die Datei `t_depth.dat` enthält ausgewählte Temperaturprofile eines Jahres aus der Talsperre Bautzen. In vertikaler Richtung liegen die Profile bereits äquidistant vor und müssen nur noch in zeitlicher Richtung interpoliert werden.

Zunächst werden die Daten eingelesen:

```
dat <- read.table("data/t_depth.dat", head=TRUE)
```

Die Datenstruktur enthält in der ersten Spalte `day` die Nummer des Messtages im Jahr und in den übrigen Spalten jeweils die Temperaturen einer Wassertiefe von 0 m bis 10 m.

Zur Vereinfachung kopieren wir zunächst die Tageszahl in eine separate Variable `day` und löschen die Tagesspalte aus dem Dataframe, der jetzt nur noch die Temperaturprofile enthält. Zusätzlich definieren wir eine später benötigte Variable `depth` mit den Tiefenstufen:

```
day <- dat$day
dat$day <- NULL
depth <- 0:10 # Datensatz enthaelt Tiefen von 0:10 in 1m-Schritten
```

Anschließend können wir die Temperaturdaten prinzipiell schon als Isoplethen mit Hilfe von

```
contour(as.matrix(dat))
```

oder mit

```
image(as.matrix(dat))
```

darstellen. Allerdings berücksichtigt diese Darstellung (neben der fehlerhaften Skalierung) nicht, dass die Probenahmetermine nicht äquidistant sind. Die Interpolation aller Tiefenstufen kann innerhalb einer `for()`-Schleife mit Hilfe der Funktion `approx` durchgeführt werden. Von jedem Durchlauf werden die Ergebnisse mit `cbind` gesammelt und ergeben die Datenstruktur `newdat`:

```
newdat <- NULL
newday <- seq(0,365,7)
for (i in 1:ncol(dat)) {
 newtemp <- approx(day, dat[,i], newday)$y
 newdat  <- cbind(newdat, newtemp)
}
```

Eine Darstellung mit:

```
image(as.matrix(newdat))
```

zeigt die interpolierten Tiefenprofile. Für eine präsentationsreife Darstellung müssen nun nur noch die Wertebereiche für die Achsen, die Schrittweite der Farbintervalle, eine sinnvolle Farbskala und die Beschriftung der Achsen angegeben werden. Damit die maximale Tiefe unten und die Wasseroberfläche oben erscheint, drehen wir `ylim` entsprechend um. Die Grafikfunktion `filled.contour` ist eine Variante von `contour`, bei der die Flächen zwischen den Kontourlinien farbig gefüllt sind und eine Legende dargestellt wird:

```
filled.contour(x=newday,                          # x-Werttebereich
               y=depth,                           # y-Wertebereich
               as.matrix(newdat),                 # die Datenmatrix
               ylim=c(max(depth),0),              # max. unten, 0m oben
               levels=seq(0,24,2),                # 0 ... +2 ... 26 °C
               col=rev(rainbow(n=13,end=0.7)), # Farbskala
               xlab="Julian Day",
               ylab="Depth (m)",
               key.title=title("T (°C)")
              )
```

Es kann sein, dass mancher die mit Hilfe von `approx` und `filled.contour` erreichte Glättung weniger gut findet. Wir wissen schließlich nicht, wie der Verlauf zwischen den Stützstellen in Wirklichkeit verläuft. Eine Alternative bietet hier die folgende Vorgehensweise: Zum einen benuzten wir keine lineare Interpolation, sondern gehen von einem während einer bestimmten Zeit konstanten Temperaturwert aus und schalten einfach immer in der Mitte zwischen zwei Terminen auf den neuen Wert um. Hierzu interpolieren wir mit `approx` und der Methode `"constant"` in einer kleinen Schrittweite (1 Tag), wobei immer genau in der Mitte zwischen den Stützstellen umgeschaltet werden soll (`f=.05`). Zum anderen benutzen wir eine leicht veränderte Routine, `slContour` aus einem eigenen Package `limnotools`[1], die wahlweise `smooth=FALSE` auf die Interpolation verzichtet, diese aber (falls mittels `drawlines=TRUE` gewünscht) als Linien einzeichnet:

```
library(limnotools)
newdat <- NULL
newday <- seq(0,365,1) # Interpolation mit 1tägiger Schrittweite
for (i in 1:ncol(dat)) {
 newtemp <- approx(day, dat[,i], newday, method="constant", f=0.5)$y
 newdat  <- cbind(newdat, newtemp)
}
slContour(x=newday,
              y=depth,
              as.matrix(newdat),
              ylim=c(max(depth),0),           # max. unten, 0m oben
              levels=seq(0,24,2),             # 0 ... +2 ... 26 °C
              col=rev(rainbow(n=13,end=0.7)), # Farbskala
              xlab="Julian Day",
              ylab="Depth (m)",
              key.title=title("T (°C)"),
              smooth=FALSE,
              drawlines=FALSE
             )
```

## 12.3 Trend-Oberflächen

Diese Methode beruht darauf, eine polynomiale Funktion der Ordnung $p$ an die Daten anzupassen. Zur Anpassung kann die Funktion `surf.ls` aus dem `spatial`-Package verwendet werden (VENABLES and RIPLEY, 2002), eine weitere Funktion `trmat` berechnet die Funktionswerte über ein regelmäßiges Gitter.

Als Beispiel benutzen wir einen synthetischen Datensatz, der annähernd die Verhältnisse eines „Geo-Datensatzes“ widerspiegeln soll. Nennen wir ihn einfach *Berg-und-Tal*-Datensatz:

```
set.seed(153)
n <- 200
k <- 0.5
xyz <- data.frame(
  x = runif(n, min=0, max=20),
```

[1] Das Paket `limnotools` kann von Github `https://github.com/rsachse/limnotools` heruntergeladen werden, die untenstehende Funktion findet sich jedoch auch im Anhang C

```
  y = runif(n, min=0, max=20)
)
xyz$z <-   cos(xyz$x * k) + sin(-xyz$y * k) + rnorm(n, sd=0.1)
```

Hierbei definieren $x$ und $y$ zufällig verteilte Messpunkte, die Sinus- und Cosinusfunktionen fügen Berge und Täler hinzu und mit Hilfe von `rnorm` addiert einen zufälligen „Messfehler“. Die Konstante `k` kann verwendet werden um eine unterschiedliche „Rauhigkeit“ (Anzahl der Berge und Täler) zu erzeugen.

Nun fitten wir eine Trendoberfläche (hier: Polynomfläche 3. Ordnung) und stellen diese dar. Da die Spalten hier $x$, $y$ und $z$ heißen, können wir die gesamte Matrix in einem Argument an `surf.ls` (Fitten von Trendflächen mit Summe der kleinsten Quadrate) übergeben. die Funktion `trmat` erzeugt aus dem Ergebnis ein rechteckiges Gitter.

```
library("spatial")
poly <- surf.ls(3, xyz)
grid.poly <- trmat(poly, min(xyz$x), max(xyz$x),
                         min(xyz$y), max(xyz$y), 50)
image(grid.poly)
contour(grid.poly, add=TRUE)
points(xyz$x, xyz$y, pch="+")
```

Wir erhalten eine polynomiale Trendfläche, die wir allerdings aus Mangel an Vergleichsmöglichkeiten noch nicht richtig beurteilen können (Abb. 12.1, links).

## 12.4 Lokale Trendflächen

Anstatt eine polynomiale Fläche global an die Daten anzupassen, können auch Funktionen verwendet werden, die die Daten lokal approximieren. Hierzu existieren mehrere Verfahren, z.B. „Loess-Polynome“ (`loess`) oder „Akima-Polynome“ (`interp` aus dem `akima`-Package):

```
library(akima)
grid.akima <- interp(xyz$x, xyz$y, xyz$z)
image(grid.akima)
contour(grid.akima, add=TRUE)
points(xyz$x, xyz$y, pch="+")
```

Im Vergleich zur vorherigen Darstellung erhalten wir eine wesentlich detailliertere Grafik (Abb. 12.1, rechts).

## 12.5 Thin Plate Splines

Das Package `fields` enthält eine Routine zur Anpassung von Thin Plate Splines (TPS), die eine Verallgemeinerung von kubischen Splines darstellen[2]. Sie werden durch zwei Parameter kontrolliert, $m$ („order of derivative penalty“) und $\lambda$ dem Glättungsparameter. Im eindimensionalen Fall und für $m = 2$ erhält man gewöhnliche kubische Splines. Der Parameter $\lambda$ bestimmt den Glättungsgrad, bei $\lambda = 0$ erhält man eine Interpolation ohne Glättung und bei $\lambda \to \infty$ erhält man ein polynomiale Fläche der Ordnung $m-1$.

[2]Die hier gegebene Darstellung orientiert sich eng an der Fields-Dokumentation, die online erhältlich ist, siehe `http://www.image.ucar.edu/GSP/Software/Fields/MAN.shtml`

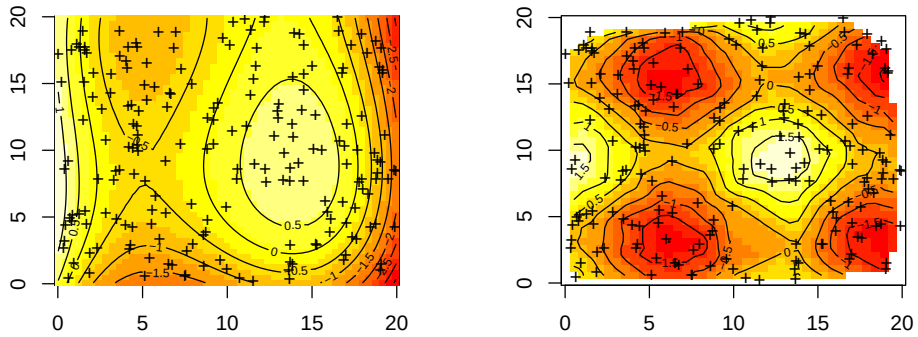


Abbildung 12.1: Globale (Polynom 3. Ordnung, links) und lokale (AKIMA, rechts) Trendflächen für den Berg- und Tal-Datensatz.

Der Parameter $\lambda$ kann für die gegebenen Daten mittels GCV (Generalized Cross Validation) automatisch bestimmt werden.

Die Funktion `Tps` erwartet als erstes Argument `x` die unabhängigen Koordinaten (nach unserer Nomenklatur $x$ und $y$) und als zweites Argument `Y` die abhängige(n) Variable(n), d.h. nach unserer Bezeichnung $z$. Angewendet auf den Berg-und-Tal-Datensatz erhalten wir demnach:

```
library(fields)
tps <- with(xyz, Tps(x=cbind(x, y), Y=z))
```

Man beachte zum einen die `width`-Funktion zur Vereinfachung der Schreibweise (Vermeidung von `xyz$x` usw.) und zum anderen die abweichende Bezeichnung der Koordinaten (`x` und `Y`)[3].

Mit Hilfe von:

```
print(tps)
plot(tps)
```

kann man nähere Informationen zur Diagnostik erhalten, z.B. den Glättungsparameter $\lambda$ =0.00023 (Erläuterung, siehe Online-Hilfe). Bei der Berechnung kann man entweder einen festen Parameter

```
lambda
```

angeben oder, wie im vorliegenden Fall, ihn durch GCV (generalized cross validation) automatisch schätzen lassen. Ein $\lambda = 0$ bedeutet keine Glättung (also Interpolation), ein größerer Wert starke Glättung.

Mit Hilfe der `predictSurface`-Funktion interpolieren wir ein regelmäßiges Gitter und können das Ergebnis mit `contour` oder `image` darstellen:

```
grid.tps <- predictSurface(tps)
image(grid.tps)
contour(grid.tps, add=TRUE)
with(xyz, points(x, y, pch="+"))
```

[3] Im **fields**-Paket sind die unabhängigen Variablen nicht auf zwei Dimensionen beschränkt sondern können auch höherdimensional definiert werden, z.B. für Wolken in der Atmosphäre oder 3D-Verteilungen im Wasserkörper

Anschließend können wir z.B. den Einfluss von $\lambda$ testen und verwenden hierfür beispielsweise Werte von 1 oder 10.

```
par(mfrow=c(2,2))
for (L in c(100, 0.2, 0.001, 0)) {
  grid.tps <- predictSurface(tps, lambda=L)
  image(grid.tps, main= paste("lambda=", L))
  contour(grid.tps, add=TRUE)
}
```

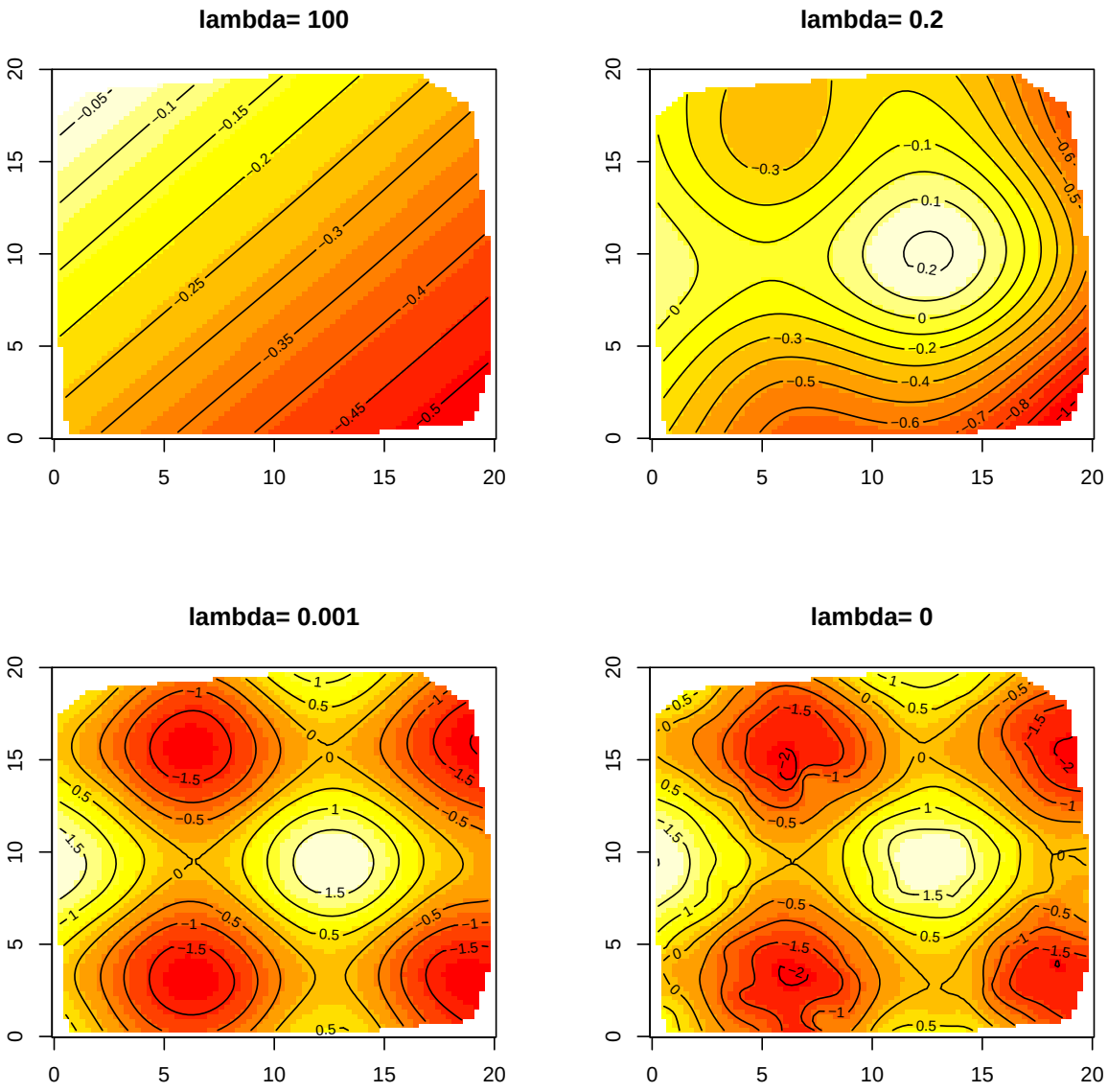



Abbildung 12.2: Thin Plate Splines mit unterschiedlicher Glättung

## 12.6 Kriging

Die wohl bekannteste Klasse geostatistischer Verfahren, das „Kriging“, benutzt Informationen über die Kovarianz der vorliegenden räumlichen Daten. Hierfür wird zunächst eine Kovarianzfunktion geschätzt, die anschließend für eine gewichtete Mittelung benachbarter Datenpunkte verwendet wird.

Kriging-Verfahren sind unter anderem im `spatial`-Package als auch im `fields`-Package enthalten. Im Folgenden verwenden wir die Version aus `fields`. Die Anwendung erfolgt ähnlich, wie bei `Tps`. Für das Skript wurden die Warnungen ausgeschaltet, in der Praxis sollte man letztere jedoch eingeschaltet lassen und darüber nachdenken:

```
kr <- with(xyz, Krig(x=cbind(x, y), Y=z, give.warnings = FALSE))
```

Mit Hilfe von:

```
print(kr)
plot(kr)
```

erhalten wir wieder nähere Informationen zur Diagnostik (Erläuterung, siehe Fields-Manual) und können das Ergebnis mit `contour` oder `image` darstellen:

```
grid.kr <- predictSurface(kr)
image(grid.kr)
contour(grid.kr, add=TRUE)
with(xyz, points(x, y, pch="+"))
```

Zur Bewertung des Ergebnisses ist es wichtig, sich den Standardfehler der geschätzten Fläche anzusehen. Dies funktioniert sowohl für Thin Plate Splines als auch für das Kriging mit Hilfe der Funktion `predictSurface.se`.

## 12.7 Weitere graphische Möglichkeiten

### 12.7.1 Farben

Anstelle der Standardpalette von `image` können selbstverständlich auch eigene Farbpaletten verwendet werden. Sinnvoll sind z.B. die Paletten `rainbow` oder `topo.colors`. Darüber hinaus bietet das Paket **RColorBrewer** eine Reihe von Paletten die auf besonders gute Erkennbarkeit der Farbabstufungen optimiert sind.

Einige Beispiele:

```
image(grid.kr, col= topo.colors(20))
image(grid.kr, col= rev(rainbow(n=60, end=0.7)))
library(RColorBrewer)
myPalette <- colorRampPalette(c(brewer.pal(11, "RdYlBu"), "darkblue"))
image(grid.kr, col= myPalette(60))
```

Im letzten Beispiel erzeugt die Funktion `brewer.pal` zunächst eine Palette mit 11 Farben an die noch ein zusätzliches Dunkelblau angehängt wird, so dass die Palette zunächst 12 Farben enthält. Eine weitere Funktion `colorRampPalette` erzeugt daraus eine Funktion, die zwischen diesen Farben interpolieren kann und es ermöglicht, z.B. 60 abgestufte Farben zu erhalten.

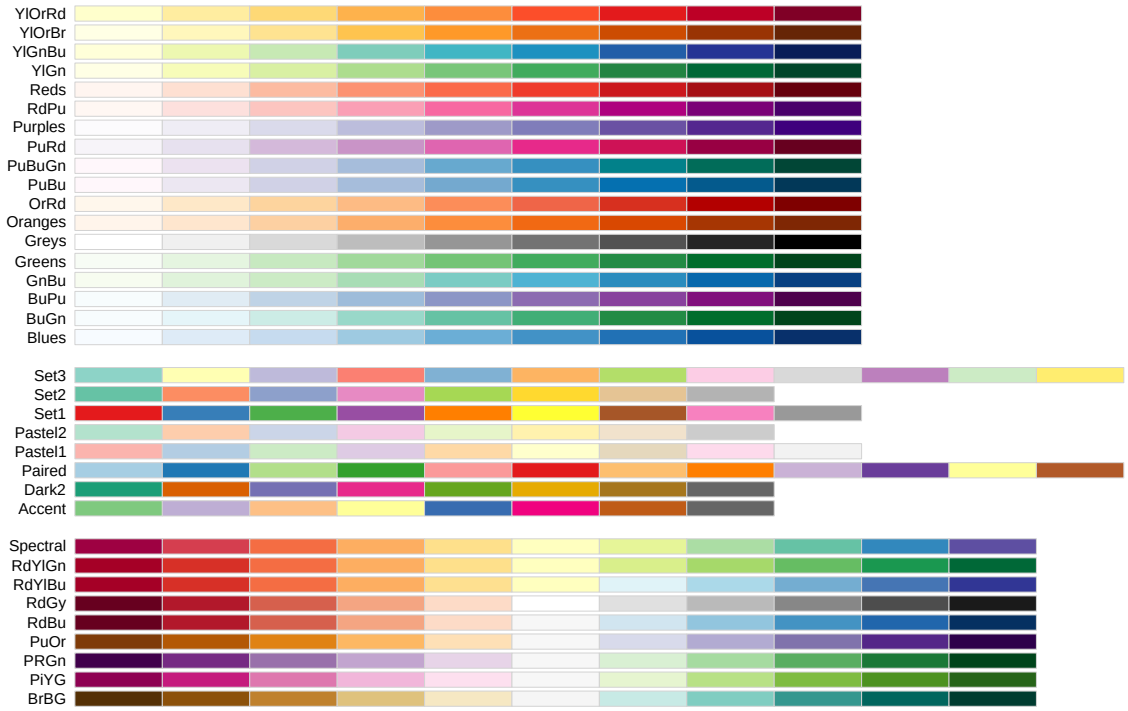



Abbildung 12.3: Paletten aus dem RColorBrewer-Paket.

### 12.7.2 3D-Grafiken

In R existieren zwei grundsätzlich unterschiedliche Methoden zur Erzeugung von dreidimensionalen Grafiken:

- statische Grafiken mit der Funktion `persp` aus dem standarmäßigen **graphics**-Paket bzw. dem Paket **scatterplot3d**, die z.B. für Publikationen verwendet werden können,
- dynamische Grafiken mit dem Paket **rgl**, die am Bildschirm interaktiv betrachtet und gedreht werden können.

#### 3D Grafiken mit persp

Die Funktion `persp` dient zur perspektivischen Darstellung von Flächen. Sie gibt als Rückgabewert die 3D-Geometrieinformation aus. Die Funktion `trans3d` kann mit Hilfe dieser Geometriematrix dreidimensionale Koordinaten so in zwei Dimensionen transformieren, dass diese mit den üblichen Grafikfunktionen, z.B. `points` oder `lines` in das 3D-Koordinatensystem projiziert werden können:

```
grid.data <- grid.kr
res <- with(grid.data, persp(x, y, z, main="persp"))
```

```
with(xyz, points(trans3d(x, y, z, pmat=res), pch=16, col="red"))
```

### 3D Grafiken mit scatterplot3d

Die Funktion `scatterplot3d` verfolgt einen ähnlichen Ansatz wie `persp`. Hier werden allerdings Datenpunkte und keine Flächen gezeichnet.

Im folgenden Beispiel sollen die interpolierten Gitterpunkte einfach als Datenpunkte dargestellt werden. Hierzu muss zunächst die spezielle Gitter-Datenstruktur der Interpolationsfläche (`x` und `y` als Vektoren, `z` als Matrix in eine „normale“ Datenstruktur mit x, y und z-Koordinaten umgewandelt werden. Dies funktioniert sehr einfach, indem man mit `expand.grid` in eine Tabelle aller Koordinatenkombinationen umwandelt und die Matrix `x` einfach in einen Vektor umformatiert (spaltenweises hintereinanderhängen):

```
dd <- expand.grid(x = grid.data$x, y=grid.data$y)
dd$z <- as.vector(grid.data$z)
```

Mit `scatterplot3d` zeichnen wir zunächst die Rohdaten. Als Ergebnis werden wie bei `persp` ebenfalls Geometrieinformationen ausgegeben, allerdings in einer ganz speziellen Form. Der Rückgabewert (im Beispiel `res`) ist eine Liste, die Funktionen zum Plotten weiterer Daten enthält. Im Beispiel verwenden wir die Funktion `points3d` zur Darstellung der Gitterpunkte. Zu beachten ist außerdem die Farbcodierung der Punkte, die abhängig von der z-Koordinate mit Hilfe einer linearen Interpolation vorgenommen wird:

```
library(scatterplot3d)
zlen <- 20
cols <- terrain.colors(zlen)
zlim <- range(na.omit(c(xyz$z, grid.data$z)))
col <- cols[floor(zlen * (grid.data$z - zlim[1]) /(zlim[2] - zlim[1]))+1]
s3d <- with(xyz,
scatterplot3d(x, y, z, highlight.3d=TRUE,
      col.axis="blue", col.grid="lightblue",
      main="scatterplot3d", pch=20)
)
s3d$points3d(dd, pch=16, cex=0.2, col=col)
```

### Dynamische 3D-Grafiken mit rgl

Bei **rgl** handelt es sich um ein Paket für dreidimensionale interaktive Graphiken auf Basis der Open GL-Schnittstelle des Betriebssystems. Die benutzten Funktionen sind im wesentlichen selbsterklärend, Details finden sich in der online-Hilfe:

```
library(rgl)
grid.data <- grid.kr
zlim <- range(na.omit(c(xyz$z, grid.data$z)))
zlen <- 20
cols <- terrain.colors(zlen)
col <- cols[floor(zlen * (grid.data$z - zlim[1]) /(zlim[2] - zlim[1]))+1]
open3d()
with(xyz, {
```

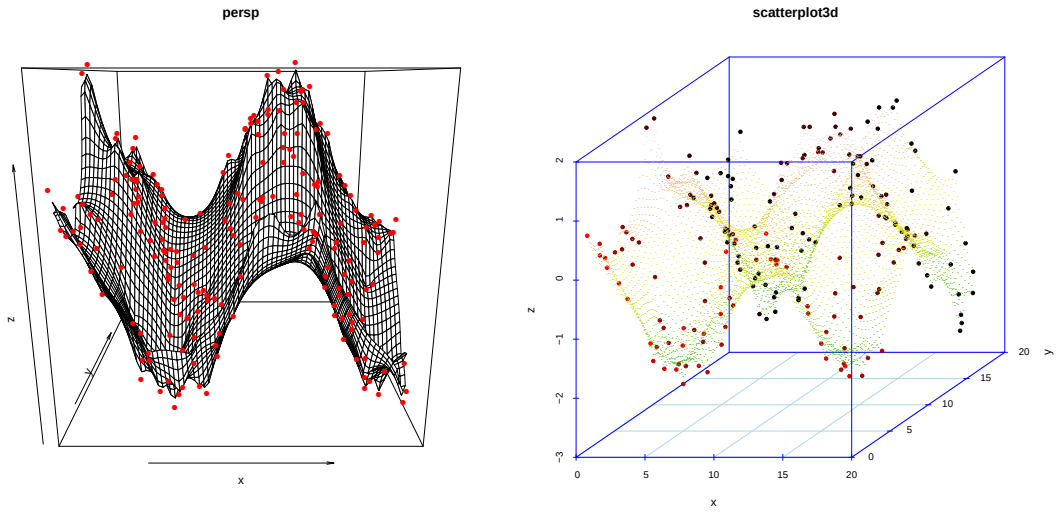



Abbildung 12.4: 3D-Grafiken mit `persp` (links) und `scatterplot3d`(rechts).

```
  plot3d(x, y, z)
  spheres3d(x, y, z, radius = 0.05)
})
aspect3d(x=1, y=1, z=0.5)
with(grid.data, surface3d(x, y, z, color=col, back="lines"))
```

## 12.8 Aufgaben

1. Vergleichen Sie die Ergebnisse der unterschiedlichen Verfahren und bewerten Sie die Anwendbarkeit. Verändern Sie insbesondere die Kontrollparameter der verschiedenen Verfahren, z.B. den Polynomgrad von `surf.ls`, `linear = FALSE` bei `interp` oder `lambda` bei Thin Plate Splines und Kriging.
2. Modifizieren Sie auch die Datensätze und vergleichen Sie insbesondere die verschiedenen Trendflächen mit Hilfe der **rgl**-Grafiken.
3. Wenden Sie die Thin Plate Spline und das Kriging auf den „Temperatur-Datensatz“ an und bewerten Sie die Ergebnisse.

## 12.9 Anwendungshinweise

Es zeigt sich, dass Thin Plate Splines und Kriging nur eingeschränkt auf den Temperaturdatensatz anwendbar sind, zumindest nicht mit den Standardeinstellungen. Die Ursache hierfür ist, dass bei den Profildaten sowohl die Skale (0 bis 10 bzw. 0 bis 365) als auch die Kovarianzfunktion in $x$ und in $y$-Richtung völlig verschieden sind. Das ist nicht verwunderlich, da die in den beiden Richtungen wirkenden Prozesse völlig verschieden sind. Zum einen besteht ein zeitlicher, zum anderen ein räumlicher Zusammenhang. Es handelt sich um einen anisotropen Datensatz.

Wir kommen zumindest *etwas* weiter, wenn wir zumindest die unterschiedliche Skalierung in $x$ und $y$-Richtung beheben. Hierzu kennt die `Krig`-Funktion den Parameter `scale.type`, der unterschiedliche Modi besitzt. Für das Temperaturprofilproblem ist der Modus `range` oder der Modus `unit.sd` sinnvoll.

```
dat <- read.table("data/t_depth.dat", head=TRUE)
tmp <- expand.grid(x=dat$day, y=0:10)
dat$day <- NULL
tmp$z <- as.vector(as.matrix(dat))
kr <- Krig(x=cbind(tmp$x, tmp$y), Y=tmp$z, scale.type="range")
grid.data <- predictSurface(kr)
image(grid.data, ylim=c(10,0))
contour(grid.data, ylim=c(10,0), add=T)
```

Weitere Möglichkeiten wären:

- die sorgfältige Analyse des Kovarianzdiagramms (bzw. Variogramms) und die Benutzung einer angemesseneren Kovarianzfunktion,
- die Verwendung eines Algorithmus für anisotrope Probleme, oder
- die Angabe, ob es im Nahbereich „Klumpen“ gibt (Nugget-Effekt).

Für nähere Details zur Geostatistik ist eine Konsultation der Originalliteratur oder entsprechender Bücher dringend angeraten, z.B. WACKERNAGEL (1998) oder ARMSTRONG (1998), zur Umsetzung in R auch RIBEIRO, JR. and DIGGLE (2001) oder WOOD (2001). Für einen Überblick über Pakete für *spatial statistics* existiert ein CRAN Task View `http://cran.r-project.org/src/contrib/Views/Spatial.html`.

## 12.10 Zweidimensionale Häufigkeitsdiagramme

Manchmal tritt das Problem auf, dass zwar $x$- und $y$-Koordinaten vorliegen, dass aber scheinbar keine abhängige Variable existiert. Vielmehr bestehen die Beobachtungen aus „Spuren“, die ein Prozess in der Ebene hinterlassen hat, z.B. Wanderungsbewegungen von Fischen in einem See, von Zugvögeln in Europa oder auch abstrakt von Kurven in einem Diagramm, z.B. bei Monte-Carlo-Simulationen.

Die Datenaufbereitung besteht darin, dass zunächst ein Gitter definiert wird und danach die Ereignishäufigkeit in jeder Gitterzelle gezählt werden muss. Ein solches Problem lässt sich natürlich durchaus für jede Spalte oder Zeile eines Gitters einzeln mit Hilfe eines Histogramms (`hist`) oder mit den R-Funktionen `cut` (Zuordnung zu Klassen) und `table` (Zählen von Häufigkeiten) lösen. Eine einfachere Möglichkeit bietet jedoch die Funktion `hist2d` aus dem `gplots`-Package.

#### Beispiel

Zunächst erzeugen wir ein paar „Spuren“ auf dem Bildschirm (Abb. 12.5, links). Wir benutzen hierzu einfach eine Sinuskurve, die sowohl in der Amplitude ($a$) als auch in der Periodenlänge ($b$) etwas zittert (wegen der normalverteilten Fehlerterme). Damit die Kurven in einem einheitlichen Maßsstab dargestellt werden können, erstellt die erste Zeile eine leere Grafik.

Damit die Kurven etwas besser zu sehen sind, färben wir sie mit den 8 Standardfarben ein (immer von schwarz über rot ... bis grau und von vorn). Dies besorgt der Modulo-Operator (`%%`) der den Rest der Division ausgibt. Die `rbind`-Funktion in der Schleife kombiniert alle Spuren zu einer langen Datenstruktur.

```
# eine leere Grafik
plot(NA, xlim=c(0,10), ylim=c(-3,3), xlab="x", ylab="y")
# Erzeugung von "Simulationsdaten"
xy <- NULL
x <- seq(0,10,length=100)
for (i in 1:40) {
 a <- rnorm(x)*0.5 + 1
 b <- rnorm(x)*0.1 + 1
 y <- a*sin(b*x)
 lines(x, y, col=i %% 8)
 xy <- rbind(xy,cbind(x,y))
}
```

Wenn wir das Ergebnis mit kleinen Punkten plotten, erhalten wir bereits einen Eindruck von der Dichte, sehen aber nicht, wenn Punkte übereinander liegen (Abb. 12.5, Mitte):

```
plot(xy, pch=".")
```

Die Klasseneinteilung und graphische Darstellung erledigt die Funktion `hist2d` aus dem **gplots**-Paket:

```
library(gplots)
gr <- hist2d(xy, nbins=c(100, 40))
```

Auch eine Darstellung mit Hilfe von `image` oder einer anderen geeigneten Funktion ist möglich, wenn man das Listenelement `counts` an ein zusätzliches Element `z` zuweist. Unter Umständen kann es dabei nützlich sein, die Häufigkeiten zu transformieren, z.B. als Wurzel oder Logarithmus (Abb. 12.5, rechts):

```
gr$z <- log(gr$counts)
image(gr, col=topo.colors(20))
```

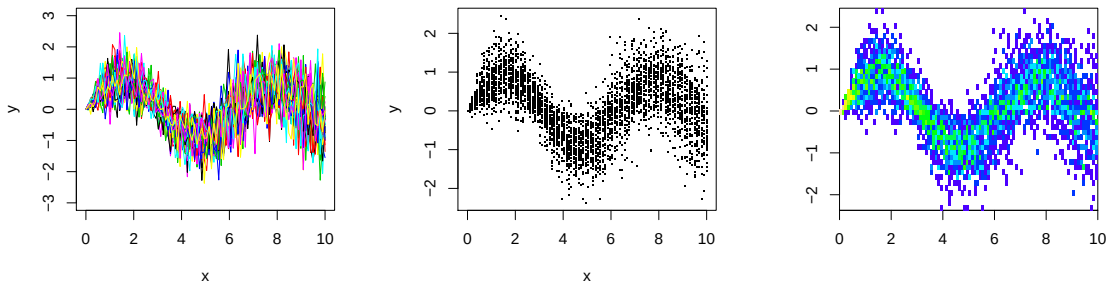



Abbildung 12.5: Spuren-Datensatz (links), Punktdiagramm (Mitte) und 2 dimensionales Häufigkeitsdiagramm der logarithmierten Häufigkeiten (rechts).

Eine weitere interessante Möglichkeit eröffnet das Paket **hexbin**[4] mit dem eine zweidimensionale Klasseneinteilung auf hexagonalen Gittern möglich ist (Abb. 12.6):

[4]Das Paket gehört zum BioConductor-Projekt. Vor der Installation muss deshalb das Repository BioC Software mit ausgewählt werden.

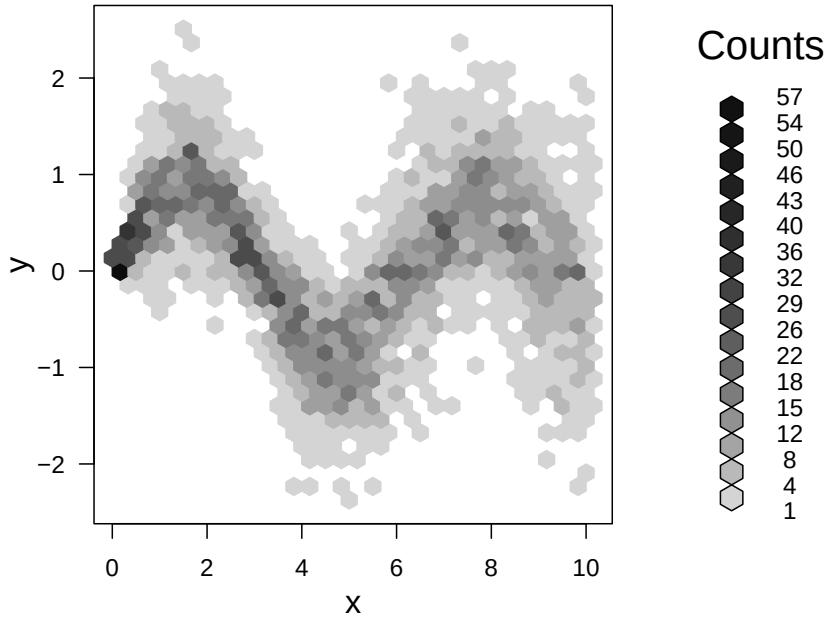



Abbildung 12.6: Hexagonales Häufigkeitsdiagramm für den Spuren-Datensatz

```
library(hexbin)
bin <- hexbin(xy)
plot(bin)
```

## 12.11 Alternative Darstellungen, die Interpolation vermeiden

Hat man nur vergleichsweise wenige Punkte, so dass eine Isoplethenfläche eher eine Anmaßung wäre (eine bunte „Seifenhaut“, aufgespannt zwischen weit entfernten Stützstellen), dann ist es vielleicht angemessen, die dritte Dimension in einem x-y-Diagramm codiert darzustellen, z.B. durch unterschiedliche Symbole, die Größe der Symbole oder die Farbe der Symbole. Alternativ kann man auch kleine Rechtecke (Säulen) direkt auf die Zeichnung setzen.

Mit Hilfe von Zufallszahlen und ausgefüllten Kreisen funktioniert das sehr einfach und ergibt einen so genannten „bubble plot“ (Abb.12.7). Da die Plot-Symbole 21 bis 25 unterschiedliche Farben für die Umrandung und die Füllung erlauben, kann man sogar noch weitere Dimensionen codieren.

```
set.seed(123)
x <- runif(10) * 10
y <- runif(10) * 10
z <- runif(10) *  5
```

```
f <- rep(1:2, 5)
nf <- layout(matrix(c(1, 2), 1, 2, byrow=TRUE), c(3, 1), 1)
par(mar=c(4.1,4.1,2,1))
plot(x, y, cex=z, pch=21, bg=2 + f, col="red", xlim=c(0, 10), ylim=c(0, 10))
par(mar=c(4.1,0,2,1))
lvl  <- seq(0.5, 5, 0.5)
nlvl <- length(lvl)
plot(rep(1, nlvl), lvl, cex=lvl, axes=FALSE, xlab="", ylab="",
  main="z", pch=21, xlim=c(0, 3), ylim=c(0, max(lvl)+1), bg="grey")
text(rep(2, nlvl), lvl, lvl, pos = 4)
```

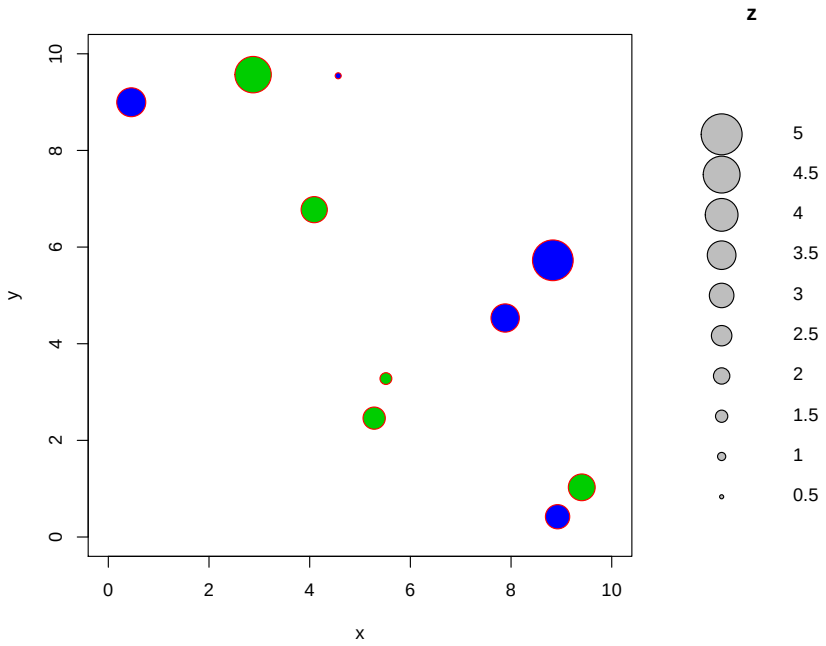



Abbildung 12.7: Bubble-Plot zur Darstellung dreidimensionaler Daten.

Wenn die Kreise zu groß oder zu klein werden, muss man die $z$-Daten natürlich mit einem entsprechenden Faktor multiplizieren oder eine geeignete Transformation durchführen. Das Beispiel demonstriert zusätzlich die Konstruktion einen nutzerdefinierten Legende. Hierzu wird mit Hilfe der Funktion `layout` die Grafikfläche in eine breite Hauptgrafikfläche und einen schmaleren Legendenbereich (rechts) aufgeteilt. Man beachte, dass die Legende letztlich auch ein „normaler“ Plot ist, bei dem lediglich die Achsenbeschriftung unterdrückt wurde.

## 12.12 Weitere Aufgaben

1. Schreiben Sie eine Funktion für eine Isoplethendarstellung von Temperaturprofilen, die folgende Elemente unter einem gemeinsamen Aufruf vereint:
   a) Aufbereitung des Datensatzes in eine für die weitere Analyse geeigneten Form,
   b) Anwendung eines geeigneten Interpolationsverfahrens,
   c) Graphische Darstellung inklusive Legende.

2. Schreiben Sie eine Funktion zur dreidimensionalen Darstellung von Datenpunkten mit dem **rgl**-Paket, die die folgenden Elemente enthält:
   a) Aufbereitung des Datensatzes in eine für die weitere Analyse geeigneten Form,
   b) Anwendung eines geeigneten Interpolationsverfahrens,
   c) Erstellung einer geeigneten Farbcodierung.
3. Schreiben Sie sich eine möglichst allgemeingültige Funktion für Bubble-Plots.

Hinweis: versuchen Sie, die Routinen möglichst allgemeingültig zu konstruieren. Benutzen sie sinnvolle defaults, reichen Sie optionale Argumente mit Hilfe des **...**-Argumentes an die verwendeten Basisfunktionen durch.

# Literaturverzeichnis


ADLER, J., 2010: R in a Nutshell. A Desktop Quick Reference. O' Reiley.

AHLBERG, J. H., E. N. NILSON and J. WALSH, 1967: The Theory of Splines and Their Applications. Academic Press, New York.

AKIMA, H., 1970: A new method of interpolation and smooth curve fitting based on local procedures. Journal of Assc. for Comp. Mach. **17**: 589–602.

AKIMA, H., 1991: A method of univariate interpolation that has the accuracy of a third-degree polynomial. ACM Trans. Math. Softw. **17**: 341–366.

AMERICAN PUBLIC HEALTH ASSOCIATION, 1992: Standard Methods for the Examination of Water and Wastewater. American Public Health Association, Inc., 18th edition.

ANDERSON, M., 2001: A new method for non-parametric multivariate analysis of variance. Austral Ecology **26**: 32???46.

ARMSTRONG, M., 1998: Basic Linear Geostatistics. Springer, Berlin.

BAI, J. and P. PERRON, 2003: Computation and analysis of multiple structural change models. J. Appl. Econ. **18:**: 1–22.

BATES, D. M. and D. G. WATTS, 1988: Nonlinear Regression and its Applications. Wiley and Sons, New York.

BIGLER, C. and J. WUNDER, 2003: Statistische Datenanalyse mit R: Eine Einf??hrung f??r Umweltwissenschaftler. ETH Z??rich, version 1.0 edition.

BOX, G. E., G. M. JENKINS and G. C. REINSEL, 1994: Time Series Analysis: Forecasting and Control. Prentice Hall Inc., Englewood Cliffs, NJ, third edition.

CANTY, A. and B. RIPLEY, 2009: **boot**: Bootstrap R (S-Plus) Functions. R package version 1.2-41.

CASPER, S. J., 1985: Lake Stechlin: A temperate oligotrophic lake, volume 58. Springer.

CLARKE, K. R., 1993: Non-parametric multivariate analysis of changes in community structure. Australian Journal of Ecology **18**: 117–143.

CLARKE, K. R. and R. M. WARWICK, 2001: Change in Marine Communities: An Approach to Statistical Analysis and Interpretation. PRIMER-E, Plymouth, second edition.

CLEVELAND, R. B., W. S. CLEVELAND, J. MCRAE and I. TERPENNING, 1990: STL: A seasonal-trend decomposition procedure based on loess. Journal of Official Statistics **6**: 3–73.

CLEVELAND, W. S., 1981: LOWESS: A program for smoothing scatterplots by robust locally weighted regression. The American Statistician **35**: 54.

CRAWLEY, M. J., 2002: Statistical Computing. An Introduction to Data Analysis Using S-PLUS. Wiley, Chichester.

CRAWLEY, M. J., 2012: The R book. John Wiley & Sons.

DALGAARD, P., 2002: Introductory Statistics with R. Springer.

DALGAARD, P., 2008: Introductory Statistics with R. Springer, second edition.

DOBSON, A. J., 1983: An Introduction to Statistical Modelling. Chapman and Hall, London.

DRAY, S. and A.-B. DUFOUR, 2007: The **ade4** package: Implementing the duality diagram for ecologists. Journal of Statistical Software **22**. URL `http://www.jstatsoft.org/v22/i04/`.

DURBIN, J. and S. J. KOOPMAN, 2001: Time Series Analysis by State Space Methods. Oxford University Press. URL `http://www.ssfpack.com/DKbook.html`.

ENGELN-MÜLLGES, G. and F. REUTTER, 1996: Numerik Algorithmen. VDI Verlag, Düsseldorf, 8th edition.

EVERITT, B., 1987: Introduction to Optimization Methods and their Application in Statistics. Chapmann and Hall, London.

FARAWAY, J. J., 2002: Practical Regression and ANOVA Using R. University of Michigan. URL `http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf`.

FISHER, R. A., 1936: The use of multiple measurements in taxonomic problems. Annals of Eugenics **7**: 179–188.

FOFONOFF, R. C., N. P. ANDMILLARD, 1983: Algorithms for computation of fundamental properties of seawater. UNESCO technical papers in marine science **44**: 1–53.

GRAFEN, A. and R. HAILS, 2002: Modern Statistics for the Life Science. Oxford University Press, Oxford. `http://www.oup.com/grafenhails/`.

GRIMM, H. and R.-D. RECKNAGEL, 1985: Grundkurs Biostatistik. Gustav Fischer Verlag, Jena.

HÅKANSON, L. and R. H. PETERS, 1995: Predictive Limnology: methods for predictive modelling. SPB Academic Publishing, Amsterdam.

HALL, D. J., 1964: An experimental approach to the dynamics of a natural population of daphnia galeata mendotae. Ecology **45**: 94–112.

HARRELL, F. E., 2001: Regression Modeling Strategies, with Applications to Linear Models, Survival Analysis and Logistic Regression. Springer. URL `http://biostat.mc.vanderbilt.edu/twiki/bin/view/Main/RmS`, iSBN 0-387-95232-2.

HOLM, S., 1979: A simple sequentially rejective multiple test procedure. Scand. J. Stat. **6**: 67–70.

HYNDMAN, R. J. and Y. KHANDAKAR, 2008: Automatic time series forecasting: The forecast package for r. Journal of Statistical Software **27**: 1–22. URL `http://www.jstatsoft.org/v27/i03`.

IHAKA, R. and R. GENTLEMAN, 1996: R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics **5**: 299–314.

IHLE, T., S. J?HNICHEN and J. BENNDORF, 2005: Wax and wane of *Microcystis* and microcystins in lake sediments: a case study in Quitzdorf Reservoir (Germany). J. Phycol. **41**: 479–488.

JÄHNICHEN, S., T. IHLE and T. PETZOLDT, 2008: Variability of microcystin cell quota: a small model explains dynamics and equilibria. Limnologica **38**: 339–349. URL http://www.sciencedirect.com/science/article/B7GX1-4SVD18N-1/1/360c98dfd30c856eac5278b1791bd987.

JÄHNICHEN, S., T. PETZOLDT and J. BENNDORF, 2001: Evidence for control of microcystin dynamics in Bautzen Reservoir (Germany) by cyanobacterial population growth rates and dissolved inorganic carbon. Archiv für Hydrobiologie **150**: 177–196.

KLEIBER, C. and A. ZEILEIS, 2008: Applied Econometrics with R. Springer.

KÖHLER, W., G. SCHACHTEL and P. VOLESKE, 2002: Biostatistik. Eine Einführung in die Biometrie für Biologen und Agrarwissenschaftler. Springer-Verlag, Berlin, Heidelberg, second edition.

KÖHLER, W., G. SCHACHTEL and P. VOLESKE, 2007: Biostatistik. Eine Einführung in die Biometrie für Biologen und Agrarwissenschaftler. Springer-Verlag, Berlin, Heidelberg, fourth edition.

KRAMBECK, H.-J., 1995: Application and abuse of statistical methods in mathematical modelling in limnology. Ecol. Model. **78**: 7–15.

LEGENDRE, P. and L. LEGENDRE, 1998: Numerical Ecology. Elsevier, second edition.

LEYER, I. and K. WESCHE, 2007: Multivariate Statistik in der ??kologie eine Einführung. Springer.

MCLEOD, A., 2009: Kendall: Kendall rank correlation and Mann-Kendall trend test. URL http://CRAN.R-project.org/package=Kendall, r package version 2.1.

OKSANEN, J., 2010: Multivariate Analysis of Ecological Communities in R: **vegan** Tutorial. URL http://cc.oulu.fi/~jarioksa/opetus/metodi/vegantutor.pdf.

PRESS, W. H., S. A. TEUKOLSKY, W. T. VETTERLING and B. P. FLANNERY, 1992: Numerical Recipes in C. Cambridge University Press, second edition. http://www.nr.com/.

QIAN, S. S., 2009: Environmental and Ecological Statistics with R. Chapman and Hall, Boca Raton. URL http://www.duke.edu/~song/eeswithr.htm.

R CORE TEAM, 2014: R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL www.r-project.org, ISBN 3-900051-07-0.

RIBEIRO, JR., P. J. and P. J. DIGGLE, 2001: **geoR**: A package for geostatistical analysis. R News **1**: 14–18. URL http://CRAN.R-project.org/doc/Rnews/.

RUDOLF, M. and W. KUHLISCH, 2008: Biostatistik. Eine Einf??hrung f??r Biowissenschaftler. Pearson Studium.

SACHS, L., 1992: Angewandte Statistik. Springer-Verlag, Berlin, 7th edition.

SCHLITTGEN, R. and B. H. J. STREITBERG, 1989: Zeitreihenanalyse. R. Oldenbourg Verlag, Wien. 3. Auflage.

SCHMIDT, S., M. K??NIG-RINKE, K. KORNEK, C. WINKELMANN, M. WETZEL, J. KOOP and J. BENNDORF, 2009: Finding appropriate reference sites in large-scale aquatic field experiments. Aquatic Ecology **43**: 169–179. URL http://dx.doi.org/10.1007/s10452-007-9155-6.

SCHOENBERG, I., 1946: Contributions to the problem of approximation of equidistant data by analytic functions. Quart. Appl. Math. **4**: 45–99, 112–141.

SHUMWAY, R. H. and D. S. STOFFER, 2006: Time Series Analysis and Its Applications: With R Examples. Springer.

STERNBERG, H. O., 1987: (1987) aggravation of floods in the amazon river as a consequence of deforestation? Geografiska Annaler **69A**: 201–219.

STERNBERG, H. O., 1995: Waters and wetlands of brazilian amazonia: An uncertain future. In: NISHIZAWA, T. and J. UITTO, eds., *The Fragile Tropics of Latin America: Sustainable Management of Changing Environments*, 113–179. United Nations University Press.

THIOULOUSE, J. and S. DRAY, 2007: Interactive multivariate data analysis in R with the **ade4** and **ade4TkGUI** packages. Journal of Statistical Software **22**. URL http://www.jstatsoft.org/v22/i06/.

VÄRE, H., R. OHTONEN and J. OKSANEN, 1995: Effects of reindeer grazing on understorey vegetation in dry pinus sylvestris forests. Journal of Vegetation Science **6**: 523???530.

VENABLES, B., 1998: Exegeses on linear models. Paper presented to the S-PLUS User???s Conference Washington, DC. URL http://www.stats.ox.ac.uk/pub/MASS3/Exegeses.pdf.

VENABLES, W. N. and B. D. RIPLEY, 1999: Modern Applied Statistics with S-Plus. Springer, New-York, third edition.

VENABLES, W. N. and B. D. RIPLEY, 2002: Modern Applied Statistics with S. Springer, New-York, fourth edition.

VENABLES, W. N., D. M. SMITH and THE R DEVELOPMENT CORE TEAM, 2001: An Introduction to R: A Programming Environment for Data Analysis and Graphics, Version 1.4.0. www.r-project.org.

VOLLENWEIDER, R. A. and J. KEREKES, 1980: OECD cooperative programme for monitoring of inland waters. (Eutrophication control). Synthesis report, Organisation for Economic Co-operation and Development, Paris.

WACKERNAGEL, H., 1998: Multivariate Geostatistics. Springer, Berlin, second edition.

WINKELMANN, C., T. PETZOLDT, J. H. E. KOOP, C. MATTHAEI and J. BENNDORF, 2008: Benthivorous fish reduce stream invertebrate drift in a large-scale field experiment. Aquatic Ecology **42**: 483–493.

WINKELMANN, C., S. WORISCHKA, J. H. KOOP and J. BENNDORF, 2007: Predation effects of benthivorous fish on grazing and shredding macroinvertebrates in a detritus-based stream food web. Limnologica - Ecology and Management of Inland Waters **37**: 121–128. URL http://www.sciencedirect.com/science/article/B7GX1-4MS9JN7-3/2/af2a1d09a552b3cd8369f2e153a3f485.

WOOD, S. N., 2001: mgcv: GAMs and generalized ridge regression for R. R News **1**: 20–25. URL http://CRAN.R-project.org/doc/Rnews/.

WOOD, S. N., 2006: Generalized Additive Models: An Introduction with R. Chapman & Hall.

ZAR, J. H., 1996: Biostatistical Analysis. Prentice-Hall, Upper Saddle River, NJ, third edition.

ZEILEIS, A., C. KLEIBER, W. KRÄMER and K. HORNIK, 2003: Testing and dating of structural changes in practice. Computational Statistics and Data Analysis **44**: 109–123.

ZEILEIS, A., F. LEISCH, K. HORNIK and C. KLEIBER, 2002: `strucchange`: An R package for testing for structural change in linear regression models. Journal of Statistical Software **7**: 1–38. URL `http://www.jstatsoft.org/v07/i02/`.

ZUUR, A., E. IENO, N. WALKER, A. SAVELIEV and G. SMITH, 2009: Mixed Effects Models and Extensions in Ecology with R. Springer Verlag.

# A Auswahlhilfe für statistische Verfahren

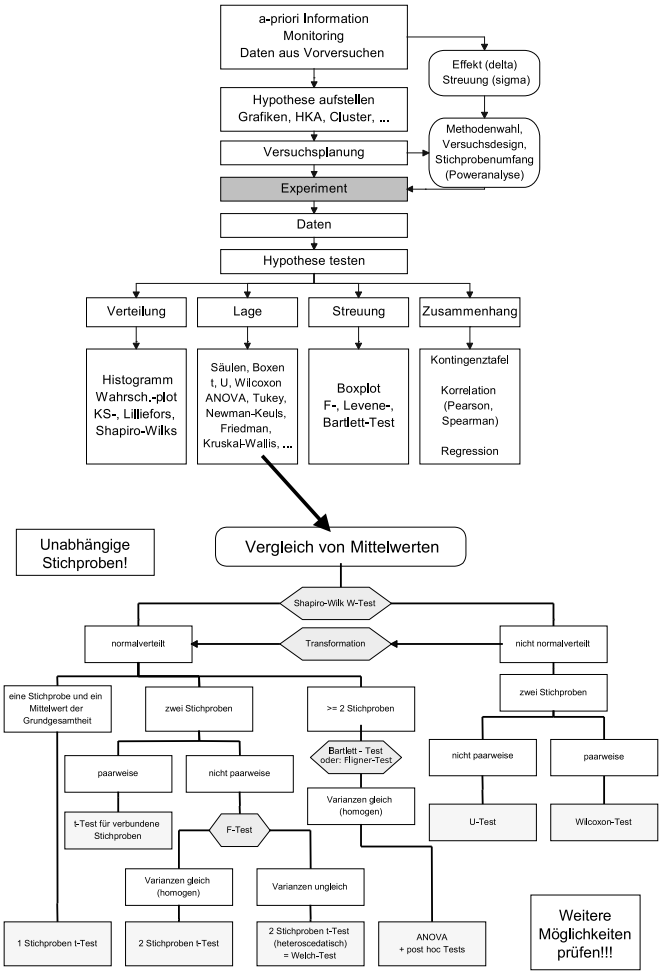



Abbildung A.1: Hilfsschema zur Auswahl eines geeigneten statistischen Verfahrens. Achtung, das Schema ist weniger als das kleine Einmaleins.

# B Mathematische und Chemische Symbole in R-Grafiken

In R-Grafiken können nahezu beliebige mathematische Formeln verwendet werden, und zwar in druckreifer Form. Es existiert hierzu eine spezielle „Expression-Syntax“, die auch für eine symbolische Verarbeitung (z.B. Bilden der ersten Ableitung) genutzt werden kann. einen Überblick über die Möglichkeiten erhält man mit:

```
demo(plotmath)
```

Einige wichtige Syntaxelemente:

- Hochstellung `x^{2}`, Tiefstellung `x[i]`,
- griechische Buchstaben, z.B. $\mu - Symbol$: `mu`
- Grad-Zeichen: `degree`, Abstand: `~`
- Für Ladungsangaben in chemischen Formeln, z.B. bei $PO_4^{3-}$ ist ein kleiner Trick erforderlich. Da „3-“ mathematisch nicht erlaubt ist, erhält man hier eine Fehlermeldung. Als Lösung fügt man eine nicht angezeigte „0“ mit `phantom(0)` hinzu, so dass die Formel „3-0“ heißt und mathematisch o.k. ist.

Die im folgenden dargestellten Beispiele können beliebig abgewandelt werden. Es lohnt sich, häufig benutzte Maßeinheiten und chemische Formeln vorzudefinieren, so dass sie jederzeit abgerufen werden können.

```
unit <- list(
  mgl  = expression(paste("(",mg~l^{-1},")")),
  mugl = expression(paste("(",mu,g~l^{-1},")")),
  temp = expression(paste("T (",degree,"C)")),
  oxy  = expression(paste(O[2]," (",mg~l^{-1},")")),
  phyto= expression(paste("Phytoplankton (",mg~l^{-1},")")),
  phos = expression(paste(PO[4]^{3-phantom(0)},"(",mu,g~l^{-1},")"))
)


x  <-1:10
y  <- 2 * x + 3 + rnorm(x)
reg<- lm(y~x)
r2 <- summary(reg)$r.squared
r2 <- round(r2,2)
a  <- reg$coeff[1]
b  <- reg$coeff[2]

plot(x, y, xlab=unit$phos, ylab=unit$phyto)
abline(reg, col="red")
```

Ein weiteres Problem tritt dann auf, wenn man versucht, variable Elemente in Formeln einzufügen. Hierzu muss eine Art Suchen und Ersetzen angewendet werden, d.h. mit `substitute` werden Platzhalter in der `expression` durch die jeweiligen Werte ausgetauscht.

```
text(2, 18, pos=4,
  substitute(paste(r^{2}," = ", xxxxx), list(xxxxx=r2)))
text(2, 20, pos=4,
  substitute(paste(y == aaa + bbb ~ x, ), list(aaa=a, bbb=b)))
```

Der optionale Parameter `pos=4` bedeutet hierbei, dass der Text rechts neben die angegebene Koordinate geschrieben werden soll.

Wem das alles zu umständlich ist, kann die Beschriftung natürlich auch nachträglich in einem Grafikprogramm hinzufügen, z.B. mit Inkscape[1].

[1] `http://www.inkscape.org`

# C Zusätzliche Quelltexte

Die Funktion `slcontour` ist eine leicht modifizierte Version der R-Funktion `contour`. Die Funktion ist u.a. in einem aktuell nicht weiterentwickelten Paket `limnotools` enthalten, das über den GitHub-Link `https://github.com/tpetzoldt/limnotools` heruntergeladen werden kann.

Alternativ kann man sich einfach den folgenden Code kopieren.

```
## modified version of filled.contour from package graphics
slContour <- function (x = seq(0, 1, len = nrow(z)),
          y = seq(0, 1, len = ncol(z)),
          z,
          xlim = range(x, finite=TRUE),
          ylim = range(y, finite=TRUE),
          zlim = range(z, finite=TRUE),
          levels = pretty(zlim, nlevels), nlevels = 20,
          color.palette = cm.colors,
          col = color.palette(length(levels) - 1),
          plot.title, plot.axes, key.title, key.axes,
          asp = NA, xaxs="i", yaxs="i", las = 1, axes = TRUE,
          smooth = TRUE,
          drawlines = TRUE, drawlabels = TRUE, ...) {
    if (missing(z)) {
        if (!missing(x)) {
            if (is.list(x)) {
                z <- x$z
                y <- x$y
                x <- x$x
            }
            else {
                z <- x
                x <- seq(0, 1, len = nrow(z))
            }
        }
        else stop("no `z' matrix specified")
    }
    else if (is.list(x)) {
        y <- x$y
        x <- x$x
    }
    if (any(diff(x) <= 0) || any(diff(y) <= 0))
        stop("increasing x and y values expected")

    mar.orig <- (par.orig <- par(c("mar","las","mfrow")))$mar
    on.exit(par(par.orig))

    w <- (3 + mar.orig[2]) * par('csi') * 2.54
    layout(matrix(c(2, 1), nc=2), widths=c(1, lcm(w)))
    par(las = las)

    ## Plot the `plot key' (scale):
    mar <- mar.orig
    mar[4] <- mar[2]
    mar[2] <- 1
    par(mar = mar)
    plot.new()
    plot.window(xlim=c(0,1), ylim=range(levels), xaxs="i", yaxs="i")
```

```
    rect(0, levels[-length(levels)], 1, levels[-1], col = col)
    if (missing(key.axes)) {
        if (axes)
            axis(4)
    }
    else key.axes
    box()
    if (!missing(key.title))
    key.title

    ## Plot contour-image::
    mar <- mar.orig
    mar[4] <- 1
    par(mar=mar)
    plot.new()
    plot.window(xlim, ylim, "", xaxs = xaxs, yaxs = yaxs, asp = asp)

    if (!is.matrix(z) || nrow(z) <= 1 || ncol(z) <= 1)
        stop("no proper `z' matrix specified")
    if (smooth == TRUE) {
      if (!is.double(z))
        storage.mode(z) <- "double"
      .Internal(filledcontour(as.double(x),
                              as.double(y),
                              z,
                              as.double(levels),
                              col = col))
    } else {
      image(x, y, z, col = col, breaks = c(levels, max(z, na.rm = TRUE)),
            add = TRUE, axes = FALSE)
    }
    if (drawlines){
      contour(x, y, z, nlevels=length(levels),
              xlim = range(x, finite = TRUE),
              drawlabels = drawlabels, add=TRUE)
    }
    if (missing(plot.axes)) {
        if (axes) {
            title(main="", xlab="", ylab="")
            axis(1)
            axis(2)
        }
    }
        else plot.axes
    box()
    if (missing(plot.title))
        title(...)
    else
    plot.title
    invisible()
}
```