Statistik mit R

Steffen Ehrmann

unter Mitwirkung von
André Schützenmeister
Prof. Dr. Hans-Peter Piepho

Institut für Pflanzenbau und Grünland (340)
Universität Hohenheim

E-Mail:
steffen.ehrmann@uni-hohenheim.de
andre.schuetzenmeister@uni-hohenheim.de
piepho@uni-hohenheim.de

06.05.2010

Inhaltsverzeichnis

Vorwort zum Teil Statistik

Bevor man anfängt, seine Daten nach dem hier im Skript gelernten einzugeben und dann irgendwann nicht mehr weiter zu wissen, sollte man einige Funktionen kennen, die einem auch dann helfen, wenn dieses Skript nicht mehr weiter hilft (siehe '↑ R: Die Dokumentation). Denn die Aufgabe dieses Skriptes besteht hauptsächlich in der Hilfe zur Selbsthilfe. Die in diesem Skript beschrieben Lösungsansätze sind bei weitem nicht alles, was das Programm R zu leisten vermag. In manchen Situationen ist es vielleicht sogar wesentlich einfacher einen eigenen Bearbeitungsansatz zu suchen, der schneller zum Ziel führt, obwohl man ihn erst recherchieren muss. Bei einer großen Datenmenge können optimierte Auswertungsverfahren viel Zeit sparen. Daher ist es sinnvoll sich Gedanken über den Aufbau der Auswertungsverfahren zu machen, bevor man anfängt die Daten einzugeben.

Ich habe mich dazu entschieden das Skript in zwei Teile aufzuteilen:

> Der eine Teil setzt sich aus den Grundlagen zusammen. Hier versuche ich zu erklären, was man wissen muss um R unabhängig vom Statistik-Skript von Herrn Piepho zu bedienen. Für diese Kapitel habe ich mir eigene Überschriften ausgedacht und mit „R: ...“ versehen.
>
> Der andere Teil besteht aus den Beispielaufgaben, die man im vorlesungsbegleitenden Statistik-Skript von Herrn Piepho (B3003) findet. Hier zeige ich in knappen Sätzen an einem Beispiel auf, wie eine bestimmte Aufgabe zu bearbeiten ist. Alle danach folgenden Aufgaben, die man mit dem davor erlernten Wissen bearbeiten kann, stelle ich lediglich als den von R dargestellten Code dar.

Einige Beispiele sind nicht relevant für dieses Skript, diese lass ich einfach aus. Die laufenden Nummern der Beispiele führe ich jedoch virtuell fort, um die Überischt zu wahren. Für diese Kapitel habe ich die selben Überschriften gewählt, wie sie im Statistik-Skript zu finden sind.

Außerdem befinden sich im Anhang alle Beispieldaten und -formeln, die man sich in ein R-Skript kopieren kann (siehe '↑ 1.3 R: Arbeiten mit R-Skripten 1').

Hier möchte ich, um Verwirrung zu vermeiden noch erwähnen, dass im Zusammenhang mit diesem Skript drei verschiedene Skripte eine Rolle spielen. Einmal gibt es das Skript von Herrn Piepho, nachfolgend immer Statistik-Skript oder Biometrie-Skript genannt, dann gibt es dieses Skript, dass ich einfach Skript nenne und als Letztes gibt es in R sogenannte Skripte, in denen man wichtige Informationen zu den verwendeten Daten oder Formeln speichern kann, die ich als R-Skript bezeichne.

Sollten sich gravierende Fehler oder auch nur stördende Rechschreibfehler in diesem Skript befinden, so bitte ich darum mir diese mitzuteilen (steffen.ehrmann@gmx.net).

R: Die Installation

Hier möchte ich eine kleine Einführung zur Installation für diejenigen geben, die vielleicht nicht so bewandert in derlei Dingen sind.

Zur Installation sollte man diese Software erst einmal in Händen halten. Das wird aber in wörtlicher Interpretation nicht möglich sein, da GNU R ein Open-Source-Projekt ist und als so genannte freie Software heruntergeladen werden kann (und auch gar nicht auf CD/DVD erhältlich ist, soweit ich momentan informiert bin). Sollte man diese Software irgendwo kostenpflichtig finden, muss man sich fragen, warum jemand Geld für freie Software ausgeben will. Meistens wird das Geld dann für den Support verlangt.
In diesem Kapitel und generell in diesem Skript möchte ich aber dafür sorgen, dass man diesen Support nicht in Anspruch nehmen muss.

Die Internetseite des R-Projekts befindet sich unter der URL
http://www.r-project.org/. Auf der linken Seite findet man ein Menü in dem alles Mögliche über R erklärt wird, das kann man sich bei Interesse durchlesen.

1. Um zum eigentlichen Download zu kommen, muss man auf den Link 'CRAN' (was „Comprehensive R Archive Network“ bedeutet und die Infrastruktur hinter dem Projekt darstellt) klicken.

2. Dann kommt man auf eine Seite, auf der sämtliche verfügbaren Server nach Nationalität aufgelistet sind, die R und andere Dateien (Packages, etc...) zum Download anbieten. Man wählt einen deutschen Server aus, wobei das eher keine Rolle spielt, da die Software auf allen Servern die selbe sein sollte. An der Sprache ändert sich ebenfalls nichts...

3. Nun kommt man zu den „Frequently used pages“, hier kann man das Betriebssystem auswählen, auf dem man R laufen lassen will.

4. In diesem Beispielfall habe ich Windows ausgewählt, für Linux-Nutzer sieht die folgende Prozedur anders aus, diese will ich aber an dieser Stelle (auch aufgrund der vielen verschiedenen Möglichkeiten) nicht erklären.

5. Um zu den Grunddateien von R zu kommen, wählt man auf der nächsten Seite den Link 'base'.

6. Hier sind nun alle für die Installation wichtigen Dateien aufgelistet. Die Installationsdatei heißt 'R-Versionsnummer-win32.exe'. Klicken auf diesen Link startet den Download der Installationsdatei von R.

7. Nun folgt die Installation. Dazu beachtet man die Installationskontexte, ich denke das muss ich nicht genauer erklären.

8. Nach der Installation kann man R starten. Will man spezielle Anpassungen vornehmen, kann man sich die Reiter in der Menüleiste durchsehen. Das würde ich generel empfehlen, sodass man schon einmal ein Gefühl für R bekommt und weiß womit man es zu tun hat.

R verfügt über keinen wirklichen Editor, der Code muss immer direkt in der Konsole von R eingegeben werden. Oftmals ist dies allerdings der umständliche Weg, da man Code öfter eingeben will oder sich Kleinigkeiten in einem sehr langen String ändern, die man nicht komplett neu schreiben will. Um dies zu vereinfachen existieren viele Editoren wie z.B Tinn-R (http://sourceforge.net/projects/tinn-r/). Die Einarbeitungszeit lohnt sich auf jeden Fall, da durch Tinn-R der Workflow wesentlich vereinfacht und verkürzt wird.

R: Die Dokumentation

Ein wichtiges „Tool“ ist die Dokumentation, die gerade bei open-source-Projekten meist sehr gut entwickelt ist. So auch bei R. Es wird eine Dokumentation zu den mitgelieferten Paketen bereitgestellt. Diese Hilfe kann man mit zwei sehr wichtigen Funktionen in Anspruch nehmen:

> `help.search("Begriff")` / `??Begriff`: Wenn man in etwa weiß, nach was man suchen will, kann man hier einen beliebigen Suchbegriff eingeben. Man bekommt dann in Form eines neuen R-Fensters eine Liste mit möglichen Suchbegriffen die R kennt und erläutern kann. Mit dem doppelten `??` kann man die selbe Funktion durchführen.
>
> `?"Begriff"`: `Begriff` kann durch jeden der von R benutzen Begriffe ersetzt werden, den man mit der Funktion `??""` gefunden hat. Man bekommt einen Hilfetext ausgegeben, der `Begriff` erklärt.

Weiß man also einmal nicht mehr, welche Funktionen wichtig für eine bestimmte Aufgabe sind, oder hat man vergessen welche Aufgabe eine Funktion denn genau hat, kann man mit diesen beiden Funktionen sein Wissen auffrischen.

Man will wissen wie man die angebotene Hilfe sinnvoll nutzen kann, beziehungsweise welche Möglichkeiten der Hilfe denn überhaupt angeboten werden.

```
help.search("help")
```

Ein neues Fenster innerhalb von R öffnet sich. Darin befindet sich folgender Inhalt:

```
Help files with alias or concept or title matching
'help' using regular expression matching:


survival::untangle.specials
                  Help Process the 'specials'
                  Argument of the 'terms' Function.
utils::RSiteSearch
                  Search for Key Words or Phrases in
                  the R-help Mailing List Archives or
                  Documentation
utils::example    Run an Examples Section from the
                  Online Help
utils::help       Documentation
```

```
utils::help.request
                  Send a Post to R-help
utils::help.search
                  Search the Help System
utils::help.start
                  Hypertext Documentation
utils::index.search
                  Search Indices for Help Files


Type '?PKG::FOO' to inspect entry 'PKG::FOO TITLE'.
```

Nun will man sich einige Beispiele angucken, also gibt man in die Konsole von R

```
?example
```

ein, als Antwort wird der Hilfe-Dialog geöffnet. Mit

```
example(example)
```

kann man sich dann die Beispiele der Funktion `example()` zeigen lassen.

R: Begriffe und Notationen

R kann nur via Kommandozeilenparameter bedient werden. Dazu muss man diese Kommandozeilenparameter kennen und verstehen wie sie funktionieren. Das Grundprinzip besteht darin, R eine Anweisung zu geben und dann in der darauf folgenden Zeile eine Antwort zu bekommen. In R wird eine Zeile, in der man Input eingeben kann mit einem > angezeigt. Immer wenn dieses Zeichen am Anfang der Kommandozeile auftaucht, kann man Input eingeben. Der Output enthält diese Zeichen nicht und ist des weiteren durch die blaue Farbe von den roten Anweisungen zu unterscheiden. Ich habe hier im Text die > -Zeichen weggelassen, so dass man das hier dargestellte leicht (und auch Zeilenweise) in die Kommandozeile von R kopieren kann. Der Output ist durch ein Leerzeichen eingeschoben und so vom Input abgehoben.

Bezeichnungen, die vor einer runden Klammer stehen, nennt man Funktionen (`help.search("")`), die Bezeichnungen in der Klammer werden Argumente (`"Begriff"`) genannt.

Argumente müssen einen Namen und einen Wert (`data=x`) haben und können dann in einer beliebigen Reihenfolge innerhalb der Funktion angegeben werden (weitere Beispiele im folgenden Text).

Genauso gut kann man die Werte auch ohne den Namen darstellen, muss dann aber die Reihenfolge beachten. Diese findet man im Hilfekontext zur Funktion.

Bei einigen Argumenten ist es lediglich möglich, zwischen `TRUE` (wahr) und `FALSE` (nicht wahr) zu wählen, diese werden als logische Argumente bezeichnet. Generell ist es auch möglich, `TRUE` mit `T` und `FALSE` mit `F` abzukürzen. Das sollte man sich aber gar nicht erst angewöhnen, da man früher oder später `T` und `F` als Variablen benutzen will.

Bei allen anderen Argumenten muss ein Wert eingegeben werden, welcher von der Aufgabenstellung verlangt wird. Ob hier ein Zahlenwert (numeric) oder ein Buchstabenwert (character) eingesetzt wird, muss man aus dem dazugehörigen Hilfekontext entnehmen.

Wenn man Funktionen und Argumenten keine Werte zuteilt, werden die vom Programm als Standard gesetzten Werte verwendet, sofern diese definiert sind. Bei Argumenten die nur mit `TRUE` und `FALSE` behandelt werden können, habe ich mich bemüht, immer die von R als Standard gesetzte Möglichkeit als erste zu notieren.

Mit dem #-Zeichen ist es möglich im Code Kommentare einzufügen, die nicht als Code gelten. Durch dieses Zeichen zeigt man R an, dass das Folgende irrelevant zur Berechnung ist und lediglich zur Dokumentation des Quellcodes dient (der Text ist dann „auskommentiert“).

R: Die wichtigsten Grundfunktionen / Rechnen mit R

Um die statistischen Auswertungsverfahren anwenden zu können, muss R natürlich Zahlen und Buchstaben darstellen und verarbeiten können. Eine einfache Methode Reihen darzustellen besteht in der `:`-Notation.

```
1:12
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

Genauso kann mit Buchstaben verfahren werden.

```
letters[1:4]
 [1] "a" "b" "c" "d"
```

```
LETTERS[1:4]
 [1] "A" "B" "C" "D"
```

Hier sei zu erwähnen, dass es bei R wichtig ist, immer die Groß-/Kleinschreibung zu beachten. R ist in der Lage, sowohl einfache als auch komplexere Terme auszurechnen und kann dabei auf alle Grundrechenarten und viele weitergehende Rechenarten zurückgreifen. Im Prinzip sind die (rechnerischen) Möglichkeiten nur durch die existierende Auswahl installierter Pakete begrenzt, und weiter Pakete kann man jederzeit nachinstallieren.

```
2+2 # Addition, Subtraktion, Multiplikation, Division
 [1] 4
```

```
abs(-2) # Betrag
 [1] 2
```

```
10^2
 [1] 100
```

```
10**2 # Potenzen
 [1] 100

------------------------------------------------------------------------

exp(-2) # Exponentialfunktion
 [1] 0.1353353

------------------------------------------------------------------------

sqrt(100) # Quadratwurzel
 [1] 10

------------------------------------------------------------------------

round(20/7) # Runden
 [1] 3
```

Hier ist es möglich das Ergebnis auf mehrere Nachkommastellen zu runden, indem man die Anzahl dieser Nachkommastellen im Argument `digits=` angibt.

```
round(20/7, digits=2)
 [1] 2.86
```

et cetera.

Am Anfang jeder Zeile wird eine Nummer ausgegeben, wie man beim Beispiel `rnorm()` sehr gut sieht. Dabei entspricht die Nummer der Stelle des danach folgenden Objekts (erste Zahl der Reihe) in der Objektreihe.

`rnorm()`: erzeugt zufällige normalverteilte Zahlenwerte.

```
rnorm(15)
 [1]  0.001523689 -1.970829104  1.921050688  1.985110394
 [5]  0.822906283  0.564822696 -0.297397728 -2.229342550
 [9] -0.920799017  1.120873290 -2.652977381  1.460718716
[13] -0.139293839 -2.341950626 -0.369230215
```

Zuweisungen sind ein sehr wichtiger Bestandteil von R. Will man zum Beispiel einen Vektor mit einer Variable „verbinden“, so muss man diesen Vektor der Variable zuweisen. Dies geschieht mit der <–-Notation.

```
gewicht <- c(60, 72, 57, 90, 95, 72)

------------------------------------------------------------------------

gewicht
 [1] 60 72 57 90 95 72
```

Daten werden oft in Form von Vektoren eingegeben. Da es sich meist um eindimensionale Daten handelt, verwendet man hier Zeilenvektoren. Dargestellt werden diese durch die Funktion `c()` (combine = Verknüpfen). Buchstabenwerte müssen hier in Hochkommas'angegeben werden (`c("ABC")`), Zahlenwerte können ohne spezielle Ergänzung angegeben werden (`c(123)`). Jeder Wert muss vom folgenden Wert durch ein Komma abgetrennt werden (`c(123, 456, 789, ...)`). Dezimalstellen werden im Englischen mit einem Punkt dargestellt (`c(1.2)`).

```
größe <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
```

```
größe
 [1] 1.75 1.80 1.65 1.90 1.74 1.91
```

Auch Variablen lassen sich miteinander verrechnen. Wichtig ist, dass die verwendeten Vektoren dieselbe Anzahl an Elementen haben und dass diese Elemente sich auch mit den angegeben Rechenmethoden verrechnen lassen.

```
bmi <- gewicht/größe^2
```

```
bmi
 [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

Neben der für einfache Rechenaufgaben üblichen Notation gibt es auch andere Notationen, die das Verarbeiten vieler Daten vereinfachen.

`sum()`: bildet die Summe $\sum$.

`prod()`: bildet das Produkt $\prod$.

`length()`: gibt die Anzahl der Vektorelemente an.

`sort()`: sortiert die Werte der Reihe nach (Standard: Aufsteigend).

```
sum(gewicht)
 [1] 446
```

```
prod(gewicht)
 [1] 151585344000
```

```
length(gewicht)
```

```
 [1] 6
```

```
sort(gewicht)
 [1] 57 60 72 72 90 95
```

Eine weitere wichtige Aufgabe von R ist es, Funktionen darzustellen. Dabei spielt die ~-Notation eine sehr große Rolle. Bei

```
y ~ x
```

stellt `y` also die Funktion von `x` dar.

Da R sehr viele Rechenoperationen durchführen kann, habe ich bis hierher nur die wichtigsten niedergeschrieben. Alle weiteren wichtigen Funktionen und Argumente stelle ich in den folgenden Kapiteln dar.

R: Ein paar nützliche Tipps

Hier habe ich einige nützliche Tipps gesammelt, die das Arbeiten mit R nochmal erheblich erleichtern können.

Setzen des Standardordners für die aktuelle Sitzung:

Will man seinen Arbeitsordner an einem anderen Platz haben, als im Ordner von R, so sollte man diesen Ordner auslagern.

Ist man viel mit R beschäftigt bietet sich das an, da man dann die eigentliche Programmstruktur von R nicht mit unnötigen Daten überschwemmt und irgendwann vielleicht den Überblick verliert.

Um zu sehen, wo sich der Arbeitsordner momentan befinden, tippt man `getwd()` ein (get working directory). Um das Verzeichnis zu ändern tippt man `setwd("dir")` ein. `dir` muss durch das gewünschte Verzeichnis ersetzt werden.

```
getwd()
 [1] "C:/Programme/R/R-2.6.1"

setwd("C:/.../R für Vl Statistik/Arbeitsordner")

getwd()
 [1] "C:/.../R für Vl Statistik/Arbeitsordner"
```

Kopieren im Befehlsfenster von R:

Will man in R etwas vom Code, den man schon eingegeben hat, kopieren, so muss man diesen nicht erst mit „strg-c“ in die Zwischenablage kopieren, sondern kann den Code markieren und sofort mit „strg-x“ an der Stelle einfügen, an der sich der Cursor befindet. R erledigt das Kopieren und Einfügen dann selbsständig.

Installieren neuer Pakete:

Das Installieren neuer Pakete folgt einem recht schlichten Schema. Glücklicherweise muss man dazu nicht die Kommandozeile bemühen, sondern kann alles über das GUI steuern.

1. Pakete → Installiere Paket(e)

2. CRAN mirror auswählen, am besten den, der am nächsten ist
3. Paket auswählen
4. Pakete → Lade Pakete
5. gerade installiertes Paket laden

Pakete können erst dann benutzt werden, wenn sie geladen wurden. Neu installierte Pakete, die nicht von Anfang an installiert waren müssen bei jedem Neueinstieg in das Programm neu geladen werden.

R: „Exportieren“ von Output

In manchen Situationen muss man den Output, den man erhält, weiterverarbeiten. Dazu ist es sinnvoll, diesen Output in einer Textdatei auszugeben. Den Dateityp kann man, neben einigen anderen vorteilhaften Eigenschaften, selbst bestimmen:

`write.table()`: gibt an, dass man eine Datei aus den in der Klammer folgenden Daten schreiben will.

`file=`: gibt den Dateinamen und Dateityp an, unter dem die Datei im aktuellen Arbeitsverzeichnis erstellt wird.

`append=TRUE/FALSE`: (append=anhängen); gibt an, ob die Daten an bereits existierende Daten angehängt werden sollen. Sinnvoll, wenn man eine Datei mit Inhalt aus verschiedenen Daten erhalten will.

`col.names=TRUE/FALSE`: gibt an, ob die Namen der Spalten in der ausgegebenen Datei angegeben sein sollen oder nicht.

`row.names=TRUE/FALSE`: gibt an, ob die Namen der Zeilen in der ausgegebenen Datei angegeben sein sollen oder nicht.

`sep=`: gibt an, mit welchem Zeichen die Daten voneinander getrennt werden sollen ($\hat{=}$ Tabulator).

`dec=`: gibt an, wie die Dezimalstelle aussehen soll.

`quote=TRUE/FALSE`: gibt an, ob die Werte in Hochkommas ausgegeben werden sollen oder nicht.

```
n <- c("Kreuzungsexperiment der japanischen Wunderblume.txt")

------------------------------------------------------------------------

write.table(x, file=n, append=FALSE, col.names=TRUE,
 row.names=FALSE, sep="\t", dec=",", quote=FALSE)
```

Die erste Zeile bindet den Namen der zu erstellenden Datei zur Variable n, was eine Menge Tipparbeit spart, wenn man denselben Dateinamen öfter verwenden will.

Man kann den Output genauso gut via copy+paste aus dem Fenster von R kopieren. Wichtig ist, dass man dann beachtet, dass man die Schriftart „Courier New“ verwendet, denn das ist die Schriftart in der auch R den Code darstellt. Beachtet man das

nicht, werden die Daten aufgrund der Unterschiedlichen Zeichengröße der verschiedenen Schriftarten unschön und eventuell unleserlich dargestellt (in Matrizen zum Beispiel).

R: Anlegen von Datensätzen

Es gibt mehrere Möglichkeiten, Daten in R zu erfassen. Die für mich einfachere ist die der Dataframes, man kann Daten aber auch in Matrizen erfassen. Daten in Matrizen zu erfassen ist zwar schreibaufwendiger, aber bei einigen Methoden notwendig, da diese Methoden lediglich mit Matrizen rechnen können. Ich werde das am Folgenden Beispiel darstellen.

Um ein Datenframe zu erstellen muss man zuerst die relevanten Daten in gleich lange Vektoren schreiben. Diese Vektoren werden dann mit einem Datenframe assoziiert.

`data.frame()`: erzeugt ein Datenframe.

Matrizen stellen zweidimensionale Anordnungen von Zahlen dar und machen dann Sinn, wenn man Daten verschiedener Messungen einer Variablen zuordnen will oder sich für einen bestimmten Test (Chi^2, Fisher, etc) interessiert.

`matrix()`: gibt an, dass eine Matrix aus den in der Klammer folgenden Daten und Argumenten erstellt werden soll.

`data=`: gibt an, welche Werte hier verwendet werden sollen.

`nrow=`: gibt die Anzahl der Reihen an, die in dieser Matrix gegeben sein sollen (row = Reihe).

`ncol=`: gibt die Anzahl der Spalten an, die in der dieser Matrix gegeben sein sollen (col = column = Spalte).

`byrow=FALSE/TRUE`: gibt an, dass die Werte nicht spaltenweise in die Matrix aufgenommen werden sollen (Standard), sondern zeilenweise.

`colnames()`: teilt R mit, welche Namen die Spalten der Matrix, die in der Klammer folgt, haben sollen.

`rownames()`: teilt R mit, welche Namen die Zeilen der Matrix, die in der Klammer folgt, haben sollen.

Will man Messwerte dimensionslos darstellen oder ist die zugehörige Variable im weitesten Sinne eine Zahl, so bietet es sich an Dataframes zu benutzen.

1 Datentypen

1.1 Kategoriale Daten

1.1.1 Nominale Daten

Seite 1, Beispiel 1:

Man erstellt eine leere 3 x 2 Matrix . Die Ausprägungen sind obwohl sie aus Buchstaben bestehen Variablen dieser Matrix und nicht die Namen für die Reihen, wie man vielleicht vermuten könnte.

```
x <- matrix(ncol=2, nrow=3)
```

```
x
      [,1] [,2]
 [1,]   NA   NA
 [2,]   NA   NA
 [3,]   NA   NA
```

Man kann die Matrix mit verschiedensten Werten befüllen. Die erste folgende Zeile gibt an, dass ein Vektor mit den Werten Weiss, Rosa, und Rot mit den Werten in `Spalte x[,1]` assoziiert wird. Die zweite folgende Zeile gibt an, dass ein Vektor mit den Werten 123, 279 und 173 mit den Werten der `Spalte x[,2]` assoziiert wird. Hier sei angemerkt, dass die Anzahl der Werte der einzelnen Vektoren immer ein ganzzahliger Teiler aller anderen Vektoren sein muss.

```
x[,1] <- c("Weiss", "Rosa", "Rot")
x[,2] <- c(123, 279, 173)
```

Hier befüllt man nach den Spalten, man könnte aber mit

```
x[1,] <- c("Weiss", 123)
x[2,] <- c("Rosa", 279)
x[3,] <- c("Rot", 173)
```

genauso gut die Zeilen befüllen. Nun ordnet man den Spalten die passenden Namen zu:

```
colnames(x) <- c("Ausprägung", "Anzahl")
rownames(x) <- c("", "", "")
```

```
x
 Ausprägung Anzahl
 "Weiss"    "123"
 "Rosa"     "279"
 "Rot"      "173"
```

Da in diesem Fall die Zeilen keine Namen haben, benennt man sie hier mit nichts.

Geht man den Weg über die Dataframes, spart man sich einiges an Arbeit. Man erzeugt die Datenvektoren.

```
Ausprägung <- c("Weiss", "Rosa", "Rot")
Anzahl <- c(123, 279, 173)
```

Nun erstellt man den Datenframe, indem man die Vektoren zuordnet.

```
x <- data.frame(Ausprägung, Anzahl)
```

```
x
   Ausprägung Anzahl
 1      Weiss    123
 2       Rosa    279
 3        Rot    173
```

Will man auf die einzelnen Vektoren zugreifen, kann man dies mit dem `$`-Zeichen tun. Dieses Zeichen zeigt an, dass die folgenden Variablen ein Element von den vorhergehenden Variablen sind. In diesem Fall hat x zwei Elemente, die mit der Variable Ausprägung und der Variable Anzahl befüllt sind.

```
x$Ausprägung
 [1] Weiss Rosa  Rot
 Levels: Rosa Rot Weiss
```

Diese Elemente finden man auch bei anderen Objekten (wie Listen), wie man später sehen wird.

Seite 2, Beispiel 1:

```
Befall <- c("ja", "nein")
Anzahl_Pflanzen <- c(152, 58)
d <- data.frame(Befall, Anzahl_Pflanzen)

------------------------------------------------------------------------

d
   Befall Anzahl_Pflanzen
 1     ja             152
 2   nein              58
```

Seite 2, Beispiel 2:

```
Produktionszweig <- c("Milchvieh", "Markfrucht", "Schweinemast",
 "Sonstige")
Anzahl_Betriebe <- c(41, 37, 79, 20)
d <- data.frame(Produktionszweig, Anzahl_Betriebe)

------------------------------------------------------------------------

d
   Produktionszweig Anzahl_Betriebe
 1        Milchvieh              41
 2       Markfrucht              37
 3     Schweinemast              79
 4         Sonstige              20
```

Seite 2, Beispiel 3:

```
x <- matrix(ncol=2, nrow=2)
x[,1] <- c("ja", "nein")
x[,2] <- c(22, 28)
colnames(x) <- c("Missgebildete Klauen", "Anzahl Tiere")

------------------------------------------------------------------------

x
      Missgebildete Klauen Anzahl Tiere
 [1,] "ja"                 "22"
 [2,] "nein"               "28"
```

1.1.2 Ordinale Daten

Seite 2, Beispiel 4:

```
Note <- c("=")
Bedeutung <- c("fehlend oder sehr gering", "sehr gering bis gering",
 "gering", "gering bis mittel", "mittel", "mittel bis stark", "stark",
 "stark bis sehr stark", "sehr stark")
d <- data.frame(Note, Bedeutung)
```

```
d
   Note                Bedeutung
 1    = fehlend oder sehr gering
 2    =   sehr gering bis gering
 3    =                   gering
 4    =        gering bis mittel
 5    =                   mittel
 6    =         mittel bis stark
 7    =                    stark
 8    =     stark bis sehr stark
 9    =               sehr stark
```

Hier habe ich mir die Eigenschaften des Datenframes zunutze gemacht, um mir eine ganze Menge Arbeit zu ersparen. Hierbei handelt es sich nur um eine darstellende Tabelle, die so in der Praxis für Verrechnungen mit R nicht verwendet würde.

Seite 3, Beispiel 1:

```
. <- c("=")
Bonitur <- c("normale Geburt", "leichte Probleme", "starke Probleme")
Anzahl_Tiere <- c(167, 34, 2)
d <- data.frame(., Bonitur, Anzahl_Tiere)
```

```
d
   .          Bonitur Anzahl_Tiere
 1 =   normale Geburt          167
 2 = leichte Probleme           34
 3 =  starke Probleme            2
```

Will man aus den Daten ein Histogramm erstellen, so sollte man dem Vektor Bonitur numerische Daten zuordnen, also die Zahlen, die zur Boniturnote gehören. Warum dies nötig ist, wird im nächsten Abschnitt ersichtlich.

Es gibt eine Vielzahl an Möglichkeiten Histogramme darzustellen. Ich möchte mich hier erst einmal auf das Säulendiagramm beschränken.

`hist()`: erzeugt ein Histogramm.

`breaks =` : gibt die Anzahl der Klassen an, indem die Anfangsstellen der Klassen angegeben werden, am Ende wird die Endstelle angegeben (eignet sich, wenn man ungleichmäßig verteilte Klassen benötigt).

`freq = TRUE/FALSE`: gibt an, ob die Anzahl der Werte (frequency) dargestellt werden soll, oder ob die Dichte der Werte (density) angegeben werden soll.

`main/sub =` : gibt Haupt- und Untertitel an.

`xlab/ylab =` : gibt die Namen für die X- und Y-Achse an.

`col =` : gibt an, in welcher Farbe die Säule dargestellt werden soll.

`density =` : gibt an mit wie vielen Linien pro inch die Balken gefüllt werden sollen.

`angle =` : gibt den Winkel der Linien an (entgegen dem Uhrzeigersinn).

`xlim/ylim =` : gibt den Bereich an, in dem die Werte dargestellt werden sollen.

`labels = FALSE/TRUE`: stellt den Wert der Säule über dieser dar.

`range()`: zeigt Minimum und Maximum eines Vektors an (daraus lässt sich dann leicht die Variationsbreite berechnen).

`nclass.Sturges()`: Berechnet die Zahl der Klassen nach Sturges' Formel.

`nclass.scott()`: Berechnet die Zahl der Klassen nach Terrel Scott.

Seite 4, Beispiel 1:

`rep()`: wiederholt die angegeben Werte; funktioniert nur mit numerischen Werten, daher sollten Boniturnoten immer als Zahlen angegeben werden.

Man erfasst die Werte für die Boniturnoten und für die Häufigkeit (in %) pro Boniturnote jeweils in einen Vektor. (Es ist nicht unbedingt notwendig die Boniturnoten mit einem Namen zu bezeichnen, da er später im Histogramm nicht sichtbar ist, vielmehr wäre es sinnvoll im Histogramm eine Legende anzulegen, in der die Bezeichnungen zu den Noten stehen).

```
Note <- c(1, 2, 3, 4)
Häufigkeit <- c(52, 42, 5, 1)
```

Die Werte der Funktion Noten müssen nun so oft vorliegen, wie in Häufigkeit angegeben. Dies erreicht man mit folgender Zeile:

```
Bonitur <- rep(Note, Häufigkeit)
```

Man verwendet vier Klassen und gibt sie an, indem man die Grenzen dieser Klassen angibt. Dabei ist wichtig, dass die letzte Klasse auch nach oben hin begrenzt sein muss.

```
brk <- c(0, 1, 2, 3, 4)
```

R gibt die Y-Achse des Histogramms in 10er Schritten aus und hört hier bei 50 auf, da der nächste 10er kaum angefangen wird. Da es hier aber schöner aussieht, die Achse bis 60 gehen zu lassen, muss man eine obere Grenze angeben:

```
ylim <- range(0, 60)
```

Nun kann man sich das Histogramm anzeigen lassen.

```
hist(Bonitur, breaks=brk, main="Behandlung 6", xlab="Bonitur",
 ylab="Häufigkeit", ylim=ylim, col="lightgrey")
```

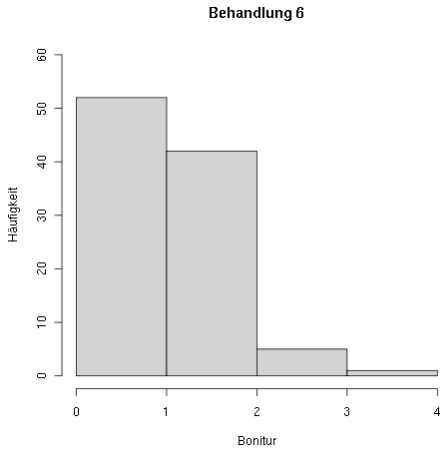


Abbildung 1.1: Histogramm für die Daten der Behandlung 6

```
note <- c(1, 2, 3, 4)
häufigkeit <- c(30, 40, 20, 10)
bonitur <- rep(note, häufigkeit)
ylim <- range(0, 60)
```

```
hist(bonitur, breaks=brk, main="Behandlung 3", ylab="Häufigkeit",
 xlab="Bonitur", ylim=ylim, col="lightgrey")
```

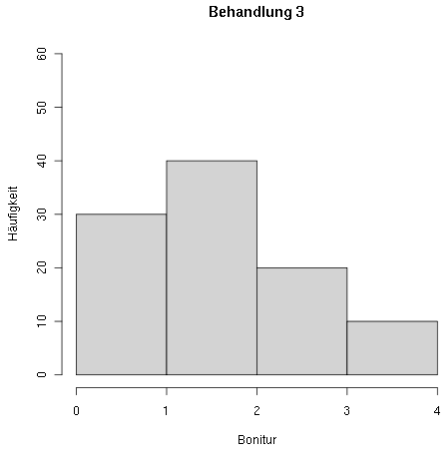


Abbildung 1.2: Histogramm für die Daten der Behandlung 3

Seite 5, Beispiel 1:

Da die Boniturnoten in R nur numerische Werte haben, ließen sie sich leicht auch zur Mittelwertbildung heranziehen, dies ist aber nicht zulässig, wie folgendes Beispiel zeigt:

`mean()`: bildet den Mittelwert (mean = Mittelwert).

```
x <- matrix(ncol=2, nrow=5)
x[,1] <- c("Kein Befall", "Bis 10% Befall", "11 bis 25% Befall",
 "26 bis 50% Befall", "Über 50% Befall")
x[,2] <- c(1, 2, 3, 4, 5)
colnames(x) <- c("Geschätzter Anteil befallener Oberfläche",
 "Boniturwert")
```

```
x
      Geschätzter Anteil befallener Oberfläche Boniturwert
```

```
 [1,] "Kein Befall"                          "1"
 [2,] "Bis 10% Befall"                       "2"
 [3,] "11 bis 25% Befall"                    "3"
 [4,] "26 bis 50% Befall"                    "4"
 [5,] "Über 50% Befall"                      "5"
```

```
y <- matrix(ncol=3, nrow=5)
d1 <- c(0/100, 25/100, 0/100, 20/100)
m1 <- mean(d1)
d2 <- c(1, 3, 1, 3)
m2 <- mean(d2)
y[,1] <- c(1, 2, 3, 4, "Mittelwert")
y[,2] <- c(d1, m1)
y[,3] <- c(d2, m2)
colnames(y) <- c("Pflanze Nr.", "Anteil befallener Oberfläche",
 "Boniturwert")
```

```
y
      Pflanze Nr.  Anteil befallener Oberfläche Boniturwert
 [1,] "1"          "0"                          "1"
 [2,] "2"          "0.25"                       "3"
 [3,] "3"          "0"                          "1"
 [4,] "4"          "0.2"                        "3"
 [5,] "Mittelwert" "0.1125"                     "2"
```

Der Mittelwert des Anteils der befallenen Oberfläche entspricht nach oben stehender Tabelle einer Boniturnote von 3, der Mittelwert der Boniturnoten aber ergibt 2.

1.2 Metrische Daten

Seite 5, Beispiel 2:

Zahlenstrahlen lassen sich einfach durch die plot- und points/lines-Funktion darstellen, dazu aber später mehr.

Seite 6, Beispiel 2:

Bei metrischen Daten ist es, im Gegensatz zu den kategorialen Daten, sinnvoll, die Differenz zwischen zwei Messwerten zu bilden.

```
y1 <- c(50)
y2 <- c(60)
```

```
y2 - y1
 [1] 10
```

Seite 6, Beispiel 3:

```
Entwicklungsstadium <- c("Ei", "Larvenstadium 1",  "Larvenstadium 2",
 "Larvenstadium 3", "Larvenstadium 4",  "Larvenstadium 5", "Puppe",
 "Imago")
Kodierung <- c("E", "L1", "L2", "L3", "L4", "L5", "P", "J")
Bonitur <- c(1:8)
d <- data.frame(Entwicklungsstadium, Kodierung, Bonitur)
```

```
d
   Entwicklungsstadium Kodierung Bonitur
 1                  Ei         E       1
 2     Larvenstadium 1        L1       2
 3     Larvenstadium 2        L2       3
 4     Larvenstadium 3        L3       4
 5     Larvenstadium 4        L4       5
 6     Larvenstadium 5        L5       6
 7              Puppe         P       7
 8              Imago         J       8
```

Wie ich schon unter '↑ 1.1.2 Ordinale Daten' dargestellt habe, reicht eine Notation wie sie im Statistik-Skript zu finden ist nicht aus um daraus ein Histogramm darzustellen. Daher habe ich hier noch die Boniturnote als numerischen Wert hinzugefügt.

Bei R muss man nun besonders darauf achten, ob es sich um kategoriale oder metrische Daten handelt, da beide Arten von Daten unter bestimmten Bedingungen numerisch dargestellt werden.

Hierbei handelt es sich um kategoriale Daten, daher macht eine Aussage über die Differenz zweier Entwicklungsstadien keinen Sinn, wenngleich man diese Differenz ganz leicht berechnen könnte.

Seite 6, Beispiel 4:

Will man Prozentuale Werte darstellen, so bietet es sich an, diesen Wert als Bruch darzustellen. Das bietet sich insofern an, als dass die Tabellen hinten im Statistik-Skript prozentuale Werte auch derartig darstellen.

```
5/100
 [1] 0.05
```

Seite 6, Beispiel 5:

```
Befall <- c("ja", "nein")
Anzahl_Pflanzen <- c(152, 58)
d <- data.frame(Befall, Anzahl_Pflanzen)
s <- sum(Anzahl_Pflanzen)
b <- 152/s
nb <- 58/s

------------------------------------------------------------------------

d
   Befall Anzahl_Pflanzen
 1     ja             152
 2   nein              58

------------------------------------------------------------------------

s
 [1] 210

------------------------------------------------------------------------

b
 [1] 0.7238095 # % befallener Pflanzen

------------------------------------------------------------------------

nb
 [1] 0.2761905 # % nicht befallener Pflanzen

------------------------------------------------------------------------

nb+b
 [1] 1 # Probe ob Gesamtwahrscheinlichkeit = 1
```

Seite 6, Beispiel 6:

```
46/55
 [1] 0.8363636
```

Seite 7, Beispiel 2:

```
60/30
 [1] 2
```

Seite 7, Beispiel 3/4:

Manchmal muss man beachten, dass eine Intervallskala keinen Nullpunkt hat, oder dieser in einer anderen entsprechenden Skala nicht an der selben Stelle liegt.

```
K <- c(273.15) #  273.15K = 0
T1 <- K + 14
T2 <- K + 7

------------------------------------------------------------------------

T1/T2
 [1] 1.024987
```

Am ersten Tag ist es viel mehr um 1.024987 mal wärmer gewesen, als am zweiten Tag. Oft macht es Sinn, die Daten, die intervallskaliert vorliegen, in eine Verhältnisskala umzuwandeln, soweit das möglich ist.

1.3 Zur Wahl des Skalenniveaus

Seite 8, Beispiel 1:

```
Oktavskala <- c(0:9)
Metrische_Skala <- c("Nicht erhoben", "0 bis <0,5% DG",
 "0,5 bis <1% DG", "1 bis <2% DG", "2 bis <4% DG", "4 bis <8% DG",
 "8 bis <16% DG", "16 bis <32% DG", "32 bis <64% DG",
 "64 bis <100% DG")
d <- data.frame(Oktavskala, Metrische_Skala)

------------------------------------------------------------------------

d
    Oktavskala Metrische_Skala
 1           0   Nicht erhoben
 2           1  0 bis <0,5% DG
 3           2  0,5 bis <1% DG
 4           3    1 bis <2% DG
 5           4    2 bis <4% DG
 6           5    4 bis <8% DG
 7           6   8 bis <16% DG
 8           7  16 bis <32% DG
 9           8  32 bis <64% DG
 10          9 64 bis <100% DG
```

Diese Darstellung ist insofern sinnvoll gewählt, als dass man beim Eingeben der Daten gleichzeitig überprüfen kann, ob man dem richtigen Oktavskala-Wert auch den richtigen metrischen Wert zugeteilt hat. Will man die Daten später in einem Histogramm verarbeiten, so kann man den Vektor Oktavskala als Vektor für die Boniturnoten verwenden.

R: Definieren eigener Funktionen

`function()`: erzeugt eine neue Funktion. In der Klammer stehen hier eine oder mehrere Variable, die die Daten widerspiegeln, die mit dieser Funktion bearbeitet werden sollen.

`return()`: gibt die Formel an, mit der die Funktion berechnet werden soll. Hierbei werden die selben Variablen verwendet, die unter function() definiert wurde.

Einige im Statistik-Skript benutzte Formeln gibt es nicht als vorgefertigte Formeln in R. Diese muss und kann man sich glücklicherweise selbst definieren. Werden diese selbst geschriebenen Formeln dann noch in ein R-Skript gespeichert, kann man sich eine riesige Menge an Tipparbeit sparen.

Beim Programmieren ist es allgemein nützlich, sich an einige Konventionen zu halten. Wie man schon am ersten folgenden Beispiel sieht, befindet sich nach der geschweiften Klammer ein Zeilenumbruch und die folgenden Anweisungen sind eingerückt. Bei der Funktion `return()` kann man das selbe beobachten. `return()` stellt hier einen „Block“ von `function{}` dar. Das Einrücken der Funktion `return()`, dient dazu, diesen Block als solchen kenntlich zu machen und eine klare, schnell ersichtliche Struktur in den Code zu bringen (ist aber nicht zwingen notwendig um den Code korrekt in R einzugeben).

Eine geschweifte Klammer stellt eine gewisse Grenze dar. Alles, was in dieser Klammer definiert wird, gilt außerhalb nicht.

Definiert man eigene Funktionen, muss man zuvor überprüfen, ob der Name, den man dieser Funktion geben will, nicht schon vergeben ist. Dies macht man am besten, indem man die Hilfe zu dieser Funktion aufruft. Ist diese Hilfe nicht vorhanden, existiert die Funktion höchstwahrscheinlich noch nicht. Um vorzubeugen, dass man seine eigenen Funktionen von „vor langer Zeit„ nicht überschreibt, sollte man sein(e) R-Skript(e) mit Strg-F durchsuchen.

Da der Code einiger Funktionen das Verständnis eines Menschen, der sich damit nicht beschäftigt, übersteigt, habe ich ein Kapitel erstellt, in dem ich die meiner Meinung nach schwer verständlichen Abschnitte erläutere (↑ 14.4 R: Erläuterung des Codes).

R: Arbeiten mit R-Skripten 1

In den nächsten Kapiteln des Statistik-Skript gibt es eine Anzahl an Daten und Formeln, die man öfter benutzen muss. Man wird sehen, dass man die eine oder andere Formel später selbst definieren muss und Daten zu einem Versuch in verschiedenen Analysen benötigt wird. Sobald man diesen Grad der Komplexität erreicht hat, wünscht man sich, dass man die ganze Schreibarbeit nicht zig mal wiederholen muss.

Glücklicherweise kann R hier Abhilfe schaffen, indem R-Skripte benutzt werden. Ein Skript in R ist also ein Dokument, in dem wichtige Aufschriebe festgehalten werden.

Will man ein R-Skript anlegen, wählt man „Datei → Neues Skript“. In das R-Skript kann man alles relevante schreiben, also zum Beispiel Datenframes, Formeln et cetera. Dabei geht man im selben Stil vor, wie im Befehlsfenster von R.

```
# Maisfeld mit 100'000 Pflanzen, daraus eine Stichprobe von 50
Pflanzen
Mais_50 <- c(175, 172, 179, 167, 163, 154, 163, 164, 157, 177, 186,
 165, 175, 194, 176, 162, 166, 169, 170, 181, 168, 166, 180, 164,
 179, 170, 150, 192, 170, 173, 170, 150, 174, 164, 182, 188, 157,
 165, 172, 168, 179, 179, 164, 162, 178, 162, 182, 171, 182, 183)
```

Hat man erstellt, was man erstellen wollte, speichert man das R-Skript mit einem aussagekräftigen Namen, dabei ist es wichtig, dass man an den Namen ein „.R“ anhängt `(Daten.R)`.

Nun muss man das R-Skript noch in die aktuelle Arbeitsfläche einbinden. Dies geschieht mit der Funktion `source("Daten.R")`. In Kapitel ’↑2.3 R: Arbeiten mit R-Skripten 2’ habe ich das von mir bis dorthin erstellte R-Skript dargestellt. Ich habe alle selbst erstellten Daten nicht mit dem >-Zeichen versehen, um zu verdeutlichen, dass dieser Text sich nicht im Befehlsfenster von R befindet.

2 Beschreibende Statistik für metrische Daten

2.1 Histogramm

Seite 11, Beispiel 1:

```
range(Mais_50)
 [1] 150 194

-------------------------------------------------------------------------

V <- 194-150

-------------------------------------------------------------------------

V
 [1] 44

-------------------------------------------------------------------------

nclass.Sturges(Mais_50)
 [1] 7

-------------------------------------------------------------------------

nclass.scott(Mais_50)
 [1] 5

-------------------------------------------------------------------------

k <- nclass.scott(Mais_50)
b <- V/k

-------------------------------------------------------------------------

b
 [1] 8.8 # wähle also b = 9, besser 10
```

Aus diesen Angaben lassen sich die Klassengrenzen (im Kopf) ableiten. Im Statistik-Skript werden diese auf 145, 155, 165, 175 ,185 und 195 gesetzt.

```
brk <- c(145, 155, 165, 175, 185 ,195)
```

Anschließend wird das Histogramm ausgegeben:

```
hist(Mais_50, freq=FALSE, breaks=brk, main="Histogramm für die
 Maisdaten", xlab="Pflanzenlänge (x)",
 ylab="Relative Häufigkeit (%)", density=10, angle=315)
```

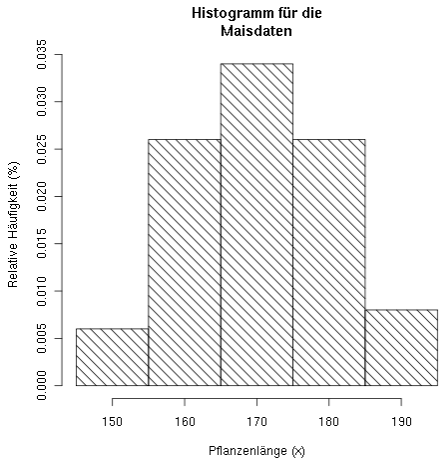


Abbildung 2.1: Säulenhistogramm für die Maisdaten

Stamm-und-Blatt Darstellung

Will man sich einen schnellen und unkomplizierten Überblick über die Verteilung der vorliegenden Daten machen, kann man ein Stamm-und Blatt Diagramm erstellen. Wie auch von Hand ist diese Methode bei R etwas schneller als das ausführliche zeichnen (lassen) eines Histogramms.

`stem()`: erzeugt einen Stem-and-leaf Plot (stem=Stamm).

```
stem(Mais_50)
   The decimal point is 1 digit(s) to the right of the |

   15 | 004
   15 | 77
   16 | 222334444
   16 | 55667889
   17 | 000012234
   17 | 556789999
   18 | 012223
   18 | 68
   19 | 24
```

2.2 Statistische Maßzahlen

2.2.1 Quantile (Perzentile)

`quantile()`: berechnet die Quantile einer Datenmenge.

`probs =` : gibt an welche Quantile dargestellt werden sollen.

Seite 12, Beispiel 1:

```
x <- c(175, 172, 179, 167, 163, 154, 163, 164, 157, 177, 186, 165,
 175, 194, 176, 162)

------------------------------------------------------------------------

quantile(x, probs=0.25)
 25%
 163

------------------------------------------------------------------------

quantile(x, 0.5)
   50%
 169.5

------------------------------------------------------------------------

quantile(x, 0.75)
    75%
 176.25
```

Seite 14, Beispiel 1:

```
quantile(x, 0.5)
   50%
 169.5
```

Seite 14, Beispiel 2:

`seq()`: erzeugt eine regelmäßige Folge an Zahlenwerten.

`from =` : gibt den Anfangswert der Sequenz an.

`to =` : gibt den Endwert der Sequenz an.

`by =` : gibt den Abstand der Werte der Sequenz an.

```
x <- c(663, 642, 644, 647, 649, 656, 656, 657, 659, 663, 665, 671)
```

```
quantile(x, probs=seq(from=0.1, to=0.2, by=0.1))
   10%   20%
 644.3 647.4
```

```
quantile(x, 0.5)
   50%
 656.5
```

Ich habe hier die Standardregel von R zur Berechnung der Quantile angewandt. Da das Statistik-Skript mit SAS erstellt wurde, entsprechen sich die Regeln nicht zu 100%. Es gibt die Möglichkeit, durch das Argument `type=("1-9")` die Regel zu verändern.

Seite 15, Beispiel 1:

```
quantile(x, seq(0,1,0.1))
    0%   10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
 642.0 644.3 647.4 651.1 656.0 656.5 658.2 661.8 663.0 664.8 671.0
```

Generell kann man sich das Arbeiten mit Quantilen in R vereinfachen, wenn man sich immer gleich alle Quantile ausgeben lässt und nicht jedes Quantil einzeln ausrechnet.

2.2.2 Lagemaße

Median

Seite 16, Beispiel 1:

```
x <- c(23, 78, 41, 79, 48)
```

```
sort(x)
 [1] 23 41 48 78 79
```

```
median(x)
 [1] 48
```

Seite 16, Beispiel 2:

```
x <- c(23, 78, 41, 79, 48, 65)
```

```
median(x)
 [1] 56.5
```

Arithmetisches Mittel

Seite 16, Beispiel 3:

```
x <- c(175, 172, 179, 167, 163, 154, 163, 164, 157, 177, 186, 165,
 175, 194, 176, 162)
```

```
mean(x)
 [1] 170.5625
```

Seite 16, Beispiel 4:

```
x <- c(1175, 172, 179, 167, 163, 154, 163, 164, 157, 177, 186, 165,
 175, 194, 176, 162)
```

```
median(x)
 [1] 169.5
```

```
mean(x)
 [1] 233.0625
```

Geometrisches Mittel

(↑Formelübersicht)

`gm()`: berechnet das Geometrische Mittel.

`x =` : Datenvektor der Werte von Interesse.

Seite 18, Beispiel 1:

```
x <- c(1.023, 1.038, 1.024, 1.033)
```

```
gm(x)
 [1] 1.029481
```

Seite 19, Beispiel 1:

```
x <- c(86200, 26900, 265, 1410, 47, 30500, 295, 890, 1340, 170,
 6750, 4000)
```

```
gm(x)
 [1] 1884.463
```

```
mean(x)
 [1] 13230.58
```

```
median(x)
 [1] 1375
```

Um mir hier Tipparbeit zu sparen, habe ich nur eingene Werte ausgewählt. Es ist aber immer noch gut sichtbar, dass bei Datenmengen, die mehrere Zehnerpotenzen beinhalten (schief verteilte Daten), der Median und das geometrische Mittel bessere Lagemaße sind als das arithmetische Mittel, das um eine Zehnerpotenz abweicht.

Harmonisches Mittel

(↑Formelübersicht)

`hm()`: berechnet das Harmonische Mittel.

`x =` : Datenvektor der Werte von Interesse.

Seite 21, Beispiel 1:

```
x <- c(4, 6)
```

```
-------------------------------------------------------------------------

hm(x)
 [1] 4.8
```

Seite 22, Beispiel 1:

```
x <- c(1.3, 1.4, 1.3, 1.2, 1.1, 1.0, 1.5, 1.4, 1.4, 1.3)

-------------------------------------------------------------------------

hm(x)
 [1] 1.272323
```

2.2.3 Streuungsmaße

Seite 23, Beispiel 1:

```
Farm <- c(1:28)
Kein <- c(0.30, 0.34, 0.39, 0.40, 0.40,  0.42, 0.48, 0.54, 0.56, 0.58,
 0.62, 0.68, 0.74, 0.74, 0.78, 0.82, 0.96, 1.02, 1.06, 1.10, 1.44, 1.60,
  1.68, 2.40, 2.40, 2.56, 3.60, 4.50)
NPK <- c(0.80, 1.12, 1.12, 1.60, 2.80, 1.14, 3.20, 1.34, 1.20, 1.22, 1.40,
 2.24, 1.54, 1.52, 1.46, 1.60, 1.60, 1.74, 1.40, 1.44, 4.16, 2.00, 4.80,
 4.48, 9.60, 5.28, 4.80, 5.50)
DAP <- c(1.64, 1.38, 1.70, 2.80, 2.40, 1.56, 1.92, 1.46, 1.66, 1.60, 2.30,
 2.76, 1.66, 2.42, 1.80, 2.50, 2.06, 2.16, 1.74, 1.74, 3.84, 2.40, 2.56,
  3.84, 3.84, 3.24, 5.60, 6.75)
sorghum <- data.frame(Farm, Kein, NPK, DAP)
```

Variationsbreite

(↑Formelübersicht)

`rv()`: berechnet die Variationsbreite.

`x =` : Datenvektor der Werte von Interesse.

Seite 24, Beispiel 1:

```
rv(Kein)
 100%
  4.2
```

Interquartilabstand

`IQR()`: berechnet den Interquartilabstand (Interquatilrange).

Seite 24, Beispiel 2:

```
IQR(Kein)
 [1] 0.955
```

Varianz

`var()`: berechnet die Varianz.

Seite 25, Beispiel 1:

```
var(Kein)
 [1] 1.078945
```

Standardabweichung

Eigentlich ist die Standardabweichung lediglich die Quadratwurzel aus der Varianz. Allerdings gibt es hier eine vorgefertigte Funktion.

`sd()`: berechnet die Standardabweichung.

Seite 26, Beispiel 1:

```
sqrt(var(Kein))
 [1] 1.038723

sd(Kein) # (standard deviation)
 [1] 1.038723
```

Variationskoeffizient

(↑Formelübersicht)

`cv()`: berechnet den Variationskoeffeizienten.

`x =` : Datenvektor der Werte von Interesse.

Seite 27, Beispiel 1:

```
cv(Kein)
 [1] 87.84126
```

Seite 27, Beispiel 2:

```
x <- matrix(nrow=6, ncol=3)
colnames(x) <- c("Kein", "NPK", "DAP")
rownames(x) <- c("Variationsbreite", "Interquartilabstand",
 "Varianz", "Standardabweichung", "Variationskoeffizien(%)",
 "Arithmetisches Mittel")
x1 <- c(rv(Kein), IQR(Kein), var(Kein), sd(Kein), cv(Kein),
 mean(Kein))
x2 <- c(rv(NPK), IQR(NPK), var(NPK), sd(NPK), cv(NPK),
 mean(NPK))
x3 <- c(vr(DAP), IQR(DAP), var(DAP), sd(DAP), cv(DAP),
 mean(DAP))
x[,1] <- round(x1, 2)
x[,2] <- round(x2, 2)
x[,3] <- round(x3, 2)
```

```
x
                          Kein   NPK   DAP
 Variationsbreite         4.20  8.80  5.37
 Interquartilabstand      0.95  2.06  1.08
 Varianz                  1.08  4.00  1.59
 Standardabweichung       1.04  2.00  1.26
 Variationskoeffizien(%) 87.84 77.62 49.47
 Arithmetisches Mittel    1.18  2.58  2.55
```

2.3 Box-Plot

`boxplot()`: erzeugt einen Plot, in dem sich auch mehrere Boxen befinden können.

`formula =` : Eine Formel, wie zum Beispiel `y ~ grp`, wobei `y` ein numerischer Vektor an Daten ist, der anhand der Gruppen `grp` aufgespalten werden soll.

`data =` : gibt an, von welchem Datensatz die Daten entnommen werden sollen.

`subset =` : gibt einen möglichen zusätzlichen Vektor an, anhand dessen die Daten für den Boxplot selektiert werden sollen.

`range =` : gibt an, wieviel mal den Interquartilabstand die Whiskers von der Box entfernt sein sollen. Bei Null wird der maximale Wert eingetragen.

`xlab/ylab/main =` : Wie schon bekannt werden mit diesem Argument die x-Achse, y-Achse und die Überschrift angegeben.

`col =` : gibt die Farbe der Boxen an. Kann auch ein Vektor sein der jede Box einzeln einfärbt.

`width =` : gibt die relative Breite der Boxen zueinander an.

`horizontal = FALSE/TRUE`: gibt an, ob die Boxen horizontal ausgegeben werden sollen.

`add = FALSE/TRUE`: gibt an ob eine neue eingegebene Box zum vorhergehenden Plot hinzugefügt werden soll. (hier ist es dann wichtig, dass man im weiteren Schritt definiert, wo diese neuen Boxen sich befinden sollen, so dass keine Überschneidungen zu sehen sind).

`at =` : gibt an, wo sich die Boxen befinden sollen.

Wenn R ein Boxplot erstellt, gibt es das arithmetische Mittel nicht an. Dieses muss man also selber eintragen, wenn man es als Vergleichswert eintragen will. Folgende Funktionen sind nützlich, um in ein Koordinatensystem nachträglich Vermerke eintragen zu können.

`points()`: erzeugt einen Punkt im Koordinatensystem. Dabei muss man den x- und y-Wert angeben und kann Farbe und Form bestimmen.

In R kann man sehr viel an seinen Schaubildern modifizieren. Wenn man sich tiefer gehend damit befassen will, sollte man sich Kapitel '↑ 5.7.1 Die Funktion par()' ansehen.

Seite 27, Beispiel 1:

```
x <- c(154, 157, 162, 163, 163, 164, 165, 167, 172, 175, 175, 176,
 177, 179, 186, 194)
boxplot(x, boxwex=.25, horizontal=TRUE, xlab="Pflanzenlänge")
points(mean(x), 1)
```

Seite 28, Beispiel 1:

```
boxplot(Mais_50, ylab="Pflanzenlänge")
points(mean(Mais_50))
```

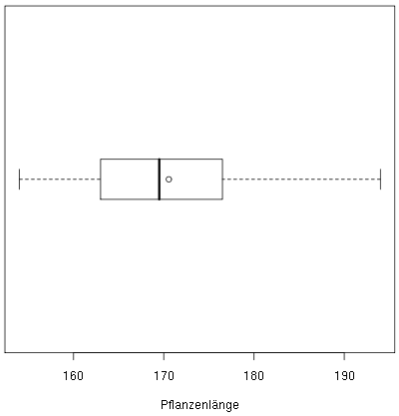


Abbildung 2.2: Boxplot zur Verteilung der Pflanzenlängen

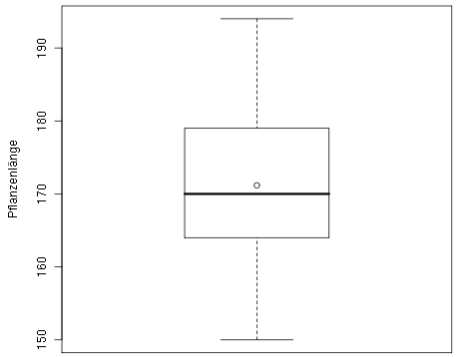


Abbildung 2.3: Boxplot zur Verteilung der Pflanzenlängen der Maisdaten

Seite 29, Beispiel 2:

```
boxplot(Kein, NPK, DAP, boxwex=0.5, names=c("Kein Dünger", "NPK",
 "DAP"), range=0, col=c("white", "grey", "lightgrey"),
 xlab="Düngemethode", ylab="Ertrag (dt/ha)", ylim=c(0, 10),
 main="On-Farm Versuch")
points(1, mean(Kein))
points(2, mean(NPK))
points(3, mean(DAP))
```

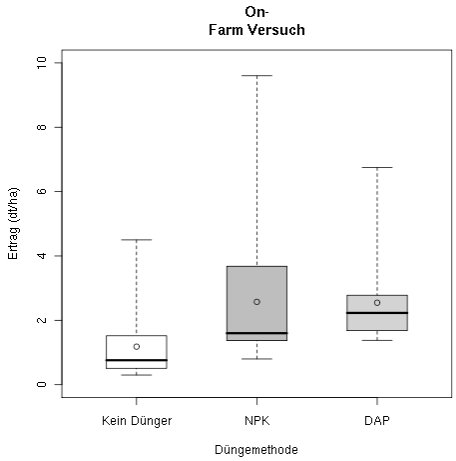


Abbildung 2.4: Boxplot zum On-Farm-Versuch an Sorghum

R: Arbeiten mit R-Skripten 2

Hier jetzt also ein mögliches R-Skript zu den verwendeten Formeln.

```
gm <- function(x){
  return( prod(x)^(1/length(x)) )
}

hm <- function(x){
  return( length(x)/sum(1/x) )
}

rv <- function(x){
  return( quantile(x, 1)-quantile(x, 0) )
}

cv <-function(x){
  return( (sd(x)/mean(x))*100 )
}
```

Diese Formeln finden sich in der ↑Formelübersicht wieder. Ein kleiner Blick zum jetzigen Zeitpunkt schadet sicherlich nicht. Das dort niedergeschriebene kann man nach der Lektüre dieses Skriptes alles verstehen. Hier sei angemerkt, dass ich im Vergleich zur ersten Auflage des Skriptes die Formeln alle in das Formelverzeichnis verschoben habe um den Textfluss besser zu gestalten. Statt der Formel selbst findet sich jetzt immer dieser Verweis, auf den man klicken kann. Man gelangt dann ins Formelverzeichnis.

3 Einführung in die schließende Statistik für normalverteilte Daten

3.1 Normalverteilung (3.2)

In R gibt es eine ganze Palette an Verteilungen, eine davon ist die Normalverteilung. (Andere Verteilungen sind die Binomialverteilung und die Poissonverteilung. Daneben bietet R noch sehr viele andere Verteilungen, die aber hier nicht relevant sind.)

Um in R mit Verteilungen arbeiten zu können, muss man jedoch einiges über Verteilungen wissen, das über das Wissen aus dem Statistik-Skript hinaus geht.

Jede Verteilung ist mit einem eigenen „Suffix“ versehen, bei der Normalverteilung ist dies `...norm`. Nun gibt es verschieden Präfixe, die verschiedene Möglichkeiten zur Berechnung der Verteilung darstellen.

> `dnorm()`: gibt die Wahrscheinlichkeitsdichte für ein Einzelereignis der Verteilung an (density). Nützlich um Schaubilder einer Verteilung zeichnen zu lassen.
>
> `pnorm()`: gibt die „Unterschreitungswahrscheinlichkeit“ eines Ereignisses an. Die Fläche unter der Dichtefunktion bis zu einem bestimmten Wert wird berechnet. Nützlich um die Wahrscheinlichkeit aller Ereignisse in einem Intervall zu berechnen.
>
> `qnorm()`: gibt die Qunatile der Normalverteilung (quantile).
>
> `rnorm()`: gibt zufällige normalverteilte Zahlen (random).
>
> `mean =` : definiert einen Mittelwert, um den die Zahlen ausgegeben werden. Standard = 0
>
> `sd =` : definiert die Standardabweichung der Daten. Standard =1
>
> `lower.tail = TRUE/FALSE`: gibt statt Unterschreitungswahrscheinlichkeit die Überschreitungswahrscheinlichkeit wieder.
>
> `plot()`: erzeugt ein Schaubild (funktioniert wie boxplot).

Genau so, wie hier mit der Normalverteilung verfahren wird, kann auch mit der t-Verteilung (und einigen anderen Verteilungen) verfahren werden. Dann wird statt `dnorm` `dt` geschrieben, wobei die Wahrscheinlichkeitsdichte für ein Einzelereignis der t-Verteilung angegeben wird. Weitere Verteilungen sind im Paket 'stats' aufgelistet, man

kann den Hilfekontext aufrufen, indem man `library(help="stats")` in die Kommandozeile eingibt. Dann werden im Index gleich zu oberst die Verteilungen (Distribution) aufgelistet.

Seite 34, Beispiel 1:

```
Mais50 <- sort(Mais_50)

plot(Mais50, dnorm(Mais50, mean(Mais50), sd=sd(Mais50)),
 type="l", ylab="Wahrscheinlichkeitsdichte", xlab="x",
 main="Wahrscheinlichkeitsverteilung der Maisdaten")
```

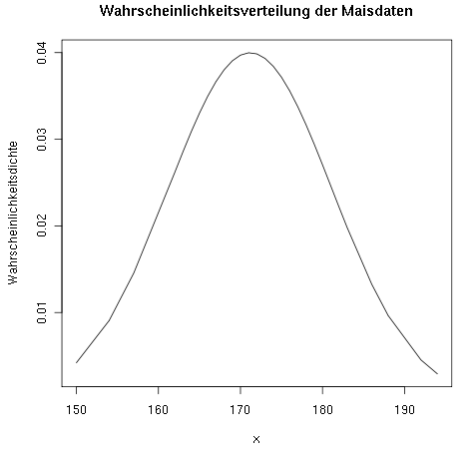


Abbildung 3.1: Wahrscheinlichkeit der Maisdaten

```
pnorm(185, mean(Mais50), sd(Mais50), lower.tail=FALSE)
 [1] 0.08299551
```

Seite 37, Beispiel 1:

```
pnorm(185, mean(Mais50), sd(Mais50))-pnorm(175, mean(Mais50),
 sd(Mais50))
 [1] 0.2679071
```

Seite 38, Beispiel 1:

```
pnorm(145, mean(Mais50), sd(Mais50))
 [1] 0.004344376
```

Seite 38, Beispiel 2:

```
pnorm(175, mean(Mais50), sd(Mais50))-pnorm(145, mean(Mais50),
 sd(Mais50))
 [1] 0.644753
```

3.2 Vertrauensintervall für einen Mittelwert (3.5)

In diesem Teil wage ich einen größeren Schritt nach vorne und definiere eine große Menge an Funktionen selbst. Hier kann man sehen wie R „hinter den Kulissen„ funktioniert. Es bietet sich an, sich nun genauer mit R auseinander zu setzen, denn dann fällt das Verständnis wesentlich leichter. Allerdings ist das Vertrauensintervall ab Kapitel '↑ 3.7 Test zum Vergleich zweier verbundener Stichproben (3.10)' nocheinmal als Bestandteil des t-Tests abgehandelt, dort in wesentlich einfacherer Weise.

Zur Berechnung des Mittelwertes wird eine große Stichprobe benötigt (über 30 Einzelwerte).

(↑Formelübersicht)

`ci()`: berechnet das Vertrauensintervall für einen Mittelwert.

`x =` : Datenvektor der Werte von Interesse.

`alpha =` : Irrtumswahrscheinlichkeit zur Berechnung der Quantile der t-Verteilung; default=.05.

Seite 42, Beispiel 1:

```
ci(Mais_50)
 [1] 168.4146 173.9454
```

Der Wert für alpha ist standardmäßig auf 0.05 eingestellt. Sollte ein anderer Wert von Interesse sein, kann man diesen wie gewohnt festlegen.

```
ci(Mais_50, .1)
 [1] 168.8592 173.5008
```

Dies ist das Vertrauensintervall für die Irrtumswahrscheinlichkeit von 10%.

3.3 Vertrauensintervall für einen Mittelwert bei kleinen Stichproben (3.6)

`qt()`: stellt die (zweiseitigen) kritischen Werte der t-Verteilung dar. Das erstes Argument ist die Sicherheitswahrscheinlichkeit, das zweite Argument sind die Freiheitsgrade.

Bei "unendlich„ großen Stichproben kann man von einem t-Wert von 1.96 ausgehen. Bei kleinen Stichproben (unter 30 Einzelnwerte) muss man aber auf die Werte aus der Tabelle II (zweiseitig) im Statistik-Skript zurückgreifen. Allerdings hat R diese Werte schon einprogrammiert, sie sind über die Funktion `qt()` abrufbar.

(↑Formelübersicht)

`ci.small()`: berechnet das Vertrauensintervall für einen Mittelwert bei kleinen Stichproben.

`x =` : Datenvektor der Werte von Interesse.

`alpha =` : Irrtumswahrscheinlichkeit zur Berechnung der Quantile der t-Verteilung; default=.05.

Seite 44, Beispiel 1:

```
ci.small(Weizen_13)
 [1] 23.63463 42.98075
```

3.4 Stichprobenumfang zur Schätzung eines Mittelwertes (3.7)

Es ist sehr wichtig im Vorraus zu wissen wie man einen Versuch anlegen muss, um aussagekräftige Ergebnisse zu erhalten. Eine gute Erhebung fängt bei einer guten Planung an.

Ist man sich über die maximale Breite des Vertrauensintervalls im Klaren, kann man mit folgender Formel ganz einfach den Stichprobenumfang zur Schätzung eines Mittelwertes berechnen:

(↑Formelübersicht)

`n.mean()`: berechnet den Stichprobenumfang zur Schätzung eines Mittelwertes.

`var =` : Fehlervarianz.

`HB =` : halbe Breite des Vertrauensintervalls.

Durch einsetzen der Varianz und der halben Breite des Vertrauensintervalls erhält man dann den Stichprobenumfang.

Seite 45, Beispiel 1:

```
n.mean(10**2, 1)
 [1] 400
```

3.5 Vertrauensintervall für die Differenz von 2 Mittelwerten (verbundene Stichprobe) (3.8)

(↑Formelübersicht)

`ci.diff2.dep()`: berechnet das Vertrauensintervall für die Differenz von zwei Mittelwerten (verbunden).

`x =` : Datenvektor zur Berechnung des ersten Mittelwertes.

`y =` : Datenvektor zur Berechnung des zweiten Mittelwertes.

`alpha =` : Irrtumswahrscheinlichkeit zur Berechnung der Quantile der t-Verteilung; default=.05.

Seite 45, Beispiel 2:

```
ci.diff2.dep(LM, CCA)
 [1] -0.1856225  0.5570511
```

3.6 Vertrauensintervall für die Differenz von 2 Mittelwerten (unverbundene Stichprobe) (3.9)

Hier tritt zum ersten mal die gepoolte Varianz auf, daher nehme ich sie mit in dieses Skript auf.

(↑Formelübersicht)

`var.pooled()`: berechnet die "gepoolte Varianz".

`x =` : Datenverktor zur Berechnung der ersten Varianz.

`y =` : Datenverktor zur Berechnugn der zweiten Varianz.

(↑Formelübersicht)

`ci.diff2.indep()`: berechnet das Vertrauensintervall für die Differenz von zwei Mittelwerten (unverbunden).

`x =` : Datenvektor zur Berechnung des ersten Mittelwertes.

`y =` : Datenvektor zur Berechnung des zweiten Mittelwertes.

`alpha =` : Irrtumswahrscheinlichkeit zur Berechnung der Quantile der t-Verteilung; default=.05.

Seite 48, Beispiel 1:

Da im Statistik-Skript leider keine Daten für das zweite Feld angegeben sind, muss man bei diesem Beispiel in die Formel gehen und die Einzelkomponenten einsetzen.

```
n1 <- 50
n2 <- 40
t_Tab <- qt(1-0.05/2, n1+n2-2)

------------------------------------------------------------------------

t_Tab
 [1] 1.98729

------------------------------------------------------------------------

v1 <- 10^2
v2 <- 9.2^2
m1 <- 171.2
m2 <- 173.7
s <- sqrt(var.pooled(data1, data2))
x_min <- (m1-m2)-t_tab*s*sqrt(1/n1+1/n2)
x_max <- (m1-m2)+t_tab*s*sqrt(1/n1+1/n2)

------------------------------------------------------------------------

c(x_min, x_max)
 [1] -6.569663  1.569663
```

Würden Daten angegeben sein, würde man die beiden Datensätze in zwei (unabhängige) Vektoren (`Mais_50`, `Mais_40`) schreiben. Diese Vektoren werden dann verrechnet:

```
ci.diff2.indep(Mais_50, Mais_40)
 [1] -6.569663  1.569663
```

3.7 Test zum Vergleich zweier verbundener Stichproben (3.10)

Um t-Tests aller Art durchführen zu können, reichen ein paar wenige Funktionen aus.

`t.test()`: erstellt einen t-Test.

`x =` : gibt einen numerischen Vektor an Datenwerten an, der verrechnet werden soll.

`alternative = c("two.sided", "less", "greater")`: gibt an, ob man einen einseitigen oder einen zweiseitigen t-Test durchführen will. Standard ist `two.sided`, bei less wird die untere Region getestet, bei greater die Obere.

`mu =` : gibt den wahren Mittelwert an. (Nur dann relevant, wenn man einen Datensatz gegen einen bekannten Mittelwert testen will.) Standard ist 0.

`paired = FALSE/TRUE`: gibt an, ob man einen verbundenen oder einen unverbundenen t-Test durchführen will (paired = gepaart).

`var.equal = FALSE/TRUE`: gibt an, ob die Varianz beim unverbundenen t-Test bei beiden Proben die selbe sein soll.

`conf.level =` : gibt die Vertrauenswahrscheinlichkeit manuell an, Standard ist 0.95.

t-Tests lassen sich in R sehr leicht berechnen. Dazu ist es lediglich nötig, die relevanten Daten in Vektoren zu erfassen und die notierten Funktionen anwenden zu können.
Das folgende Beispiel zeigt einen Zwei-Stichproben-Test, dass heißt, es werden zwei Datensätze miteinander verglichen.

Seite 50, Beispiel 1:

```
LM <- [1] 2.2 2.2 1.9 1.2 1.3 0.9 1.0 0.5 1.8 1.1 1.6 1.0 1.6 0.6
CCA <- [1] 3.5 2.0 2.9 0.4 0.6 0.5 0.6 0.3 2.2 0.7 0.9 0.3 1.1 0.3
phalombe <- data.frame(LM, CCA)
```

Die relevanten Daten.

```
t.test(LM, CCA, paired=TRUE)
```

Da die Daten hier verbunden sind, benutzt man das Argument `paired=TRUE`.

```
        Paired t-test
```

Hier kann man noch einmal überprüfen ob man den richtigen Test angewandt hat.

```
data:  LM and CCA
```

Hier sind die benutzten Datenvektoren noch einmal aufgeführt.

```
t = 1.0805, df = 13, p-value = 0.2996
```

Hier fängt es an interessant zu werden. `t=` stellt den "t_Vers-Wert„ dar, `df=` die Freiheitsgrade (df = degrees of freedom) und `p-value` den p-Wert. Nun muss man nicht die t-Werte mit der schon aus dem Statistik-Skript bekannten Tafel vergleichen, sondern lediglich den p-Wert ablesen und mit dem Signifikanzniveau von 5% (0.05) vergleichen. In diesem Beispiel sieht man, dass der p-Wert bei 0.3 liegt, was größer als 0.05 ist. Daher kann man keine Signifikanz nachweisen. ($p =< 0.05 \rightarrow$ Signifikanz; $p > 0.05 \rightarrow$ keine Signifikanz)

```
alternative hypothesis: true difference in means is not equal to 0
```

Hier sieht man die Alternativhypothese. Man kann den Wert ablesen, den man getestet hat und das man zweiseitig getestet hat (not equal to, bei einem einseitigen t-Test würde greater oder less geschrieben stehen).

```
95 percent confidence interval:
 -0.1856225  0.5570511
```

Hier kann man auf unkomplizierte Weise das Vertrauensintervall ablesen und kann sich die komplizierten Formeln aus den Kapiteln 3.5 bis 3.9 sparen.

```
sample estimates:
mean of the differences
              0.1857143
```

Hier kann man den Mittelwert der Differenzen ablesen.

3.8 Test zum Vergleich zweier unverbundener Stichproben (3.11)

Auch hier wird ein Zwei-Stichproben-Test berechnet.

Seite 55, Beispiel 1:

```
A <- c(29.9, 11.4, 25.3, 16.5, 21.1, NaN)
B <- c(26.6, 23.7, 28.5, 14.2, 17.9, 24.3)
Tomaten <- data.frame(A, B)
```

```
Tomaten
      A    B
 1 29.9 26.6
 2 11.4 23.7
 3 25.3 28.5
 4 16.5 14.2
 5 21.1 17.9
 6  NaN 24.3
```

Da der Dünger A nur auf 5 Parzellen geteset wurde, muss man den 6. Wert als `NaN` bezeichnen, das zeigt an, dass hier kein Wert vorhanden ist (NaN = Not a Number).

```
t.test(A, B, var.equal=TRUE)

         Two Sample t-test

 data:  A and B
 t = -0.4437, df = 9, p-value = 0.6677
 alternative hypothesis: true difference in means is not equal to 0
 95 percent confidence interval:
  -10.326908   6.940241
 sample estimates:
 mean of x mean of y
  20.84000  22.53333
```

Im Statistik-Skript werden die Varianzen gepoolt, das heißt es wird eine gemeinsame Varianz berechnet. Daher kann man hier das Argument `var.equal=TRUE` anwenden. Würde man die Varianzen nicht poolen, müsste man von zwei unterschiedlichen Varianzen ausgehen.

3.9 Stichprobenumfang für den unverbundenen t-Test (3.14)

Es ist sehr wichtig im Vorraus zu wissen, wie man einen Versuch anlegen muss, um aussagekräftige Ergebnisse zu erhalten. Eine gute Erhebung fängt bei einer guten Planung an.

`power.t.test()`: erstellt eine Liste mit allen relevanten Daten um Aussagen über die Teststärke zu treffen.

`n =` : gibt den Stichprobenumfang an.

`delta =` : gibt die kleinste Nachzuweisende Differenz an.

`sd =` : gibt die Standardabweichung an.

`sig.level =` : gibt die Irrtumswahrscheinlichkeit an.

`power =` : gibt die Teststärke/Güte des Tests an.

`type = c("two.sample", "one.sample", "paired")`: gibt den Typ des Tests an.

`alternative = c("two.sided", "one.sided")`: gibt an, ob ein zweiseitiger oder ein einseitiger Test durchgeführt wird.

`power.t.test()` rechnet die Teststärke genauer aus, als die Formel aus dem Statistik-Skript (diese ist nur eine Näherungsfunktion), allerdings habe ich zum Verständnis eine Formel geschrieben, die genau nach der Formel im Statistik-Skript rechnet.

(↑Formelübersicht)

`t.test.planning()`: berechnet die Teststärke.

`n =` : Stichprobenumfang (entweder 'n' oder 'power' angeben).

`power =` : Güte oder Teststärke zur Berechnung der Quantile der Standardnormalverteilung.

`delta =` : Kleinste nachzuweisende Diffenrenz.

`var =` : Fehlervarianz.

`alpha =` : Irrtumswahrscheinlichkeit zur Berechnung der Quantile der Standardnormalverteilung, default=.05.

Seite 61, Beispiel 1:

```
power.t.test(delta=5, sd=sqrt(7.733), power=.9, sig.level=.05)

     Two-sample t test power calculation

              n = 7.602981
          delta = 5
             sd = 2.780827
      sig.level = 0.05
          power = 0.9
    alternative = two.sided

 NOTE: n is number in *each* group

-----------------------------------------------------------------------

t.test.planning(power=.9, delta=5, var=7.733)

 n       = 6.500312
 power   = 0.9
 delta   = 5
 var     = 7.733
 alpha   = 0.05
```

3.9.1 Post-hoc Berechnung der Teststärke (3.14.1)

Auch hier kann man power.t.test und t.test.planning anwenden.

Seite 65, Beispiel 1:

```
power.t.test(n=4, delta=5, sd=sqrt(7.733), sig.level=.05)

     Two-sample t test power calculation

              n = 4
          delta = 5
             sd = 2.780827
      sig.level = 0.05
          power = 0.5675938
    alternative = two.sided

 NOTE: n is number in *each* group

-------------------------------------------------------------------------

t.test.planing(n=4, delta=5, var=7.733)

 power   = 0.7199958
 n       = 4
 delta   = 5
 var     = 7.733
 alpha   = 0.05
```

Hier sieht man sehr gut, dass die Formel aus dem Statistik-Script nur approximativ ist, denn `power.t.test` gibt den exakten Wert an.

3.10 Test des Parameters μ (3.17)

Hierbei handelt es sich um einen Ein-Stichproben-Test, bei dieser Art von t-Tests wird ein Datensatz mit einem theoretischen Mittelwert verglichen.

Seite 69, Beispiel 1:

Leider liegen die genauen Daten nicht vor, daher kann ich dieses Beispiel lediglich beschreiben. Würden die Daten vorliegen, würde man sie in den Vektor

```
Einkommen_Studenten <- c(nicht vorliegende Daten)
```

schreiben. Man würde einen ganz normalen t-Test mit dem Datenvektor befüllen und den Vergleichsmittelwert hinzufügen.

```
t.test(Einkommen_Studenten, mu=1300)
```

Das Ergebnis würde wie bei den anderen t-Tests aussehen, allerdings würden eben die entsprechenden Daten angegeben.

3.11 Vertrauensintervall für eine Varianz (3.18)

Bei der folgenden Funktion muss man über die Quantile der Chi-Quadraht-Verteilung gehen.

(↑Formelübersicht)

`ci.var()`: berechnet das Vertrauensintervall für eine Varianz.

`x =` : Datenverktor zur Berechnung der Varianz.

`alpha =` : Irrtumswahrscheinlichkeit zur Berechnung der Quantile der Standardnormalverteilung, default=.05.

Seite 71, Beispiel 1:

Da die eigentlichen Daten hier nicht vorliegen, gehe ich wieder in die Formeln und ziehe die einzelnen Komponenten heraus:

```
n <- 50
alpha <- .05
var <- 100
q1 <- qchisq(1-alpha/2, (n-1))
q2 <- qchisq(alpha/2, (n-1))
x_min <- ((n-1)*var)/q1
x_max <- ((n-1)*var)/q2

------------------------------------------------------------------------

c(x_min, x_max)
 [1]  69.77829 155.28484
```

3.12 Test zum Vergleich zweier unabhängiger Stichprobenvarianzen (3.19)

Die Varianzen zweier Stichproben können genauso wie die Mittelwerte miteinander verglichen werden. Dazu braucht man allerdings eine andere Art von Test, der aber im Endeffekt genauso wie der t-Test funktioniert.

`var.test()`: erstellt einen Test zum Vergleich der Varianzen.

```
var.test(Tomaten$A, Tomaten$B)

        F test to compare two variances

data:  Tomaten$A and Tomaten$B
```

```
F = 1.7792, num df = 4, denom df = 5, p-value = 0.54
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
  0.2408255 16.6611835
sample estimates:
ratio of variances
          1.779191
```

Wie immer gilt auch hier: Will man mehr Informationen über dieses Thema erhalten, kann man mit die Funktion `?var.test` recherchieren.

3.13 Einseitige und zweiseitige Tests (3.20)

Seite 76, Beispiel 1:

Auch hier sind leider keine genaueren Daten vorhanden. Hier kommt das Argument `alternative= c("two.sided", "less", "greater")` zum Tragen. Bei `greater` gilt $\mu > \mu 0$.

3.14 Äquivalenztest am Beispiel zweier unverbundener Stichproben (3.21)

Hier habe ich erneut selbst eine Funktion geschrieben, die den Äquivalenztest sehr einfach und schnell darstellt.

(↑Formelübersicht)

`equi.test()`: berechnet den Äquivalenztest am Beispiel zweier unverbundener Stichproben.

`x =` : Datenvektor zur Berechnung des ersten Mittelwertes.

`y =` : Datenvektor zur Berechnung des zweiten Mittelwertes.

`delta =` : Äquivalenzgrenze zur Berechnung der durch einen Versuch bestimmten Quantile der t-Verteilung.

`alpha =` : Irrtumswahrscheinlichkeit zur Berechnung der Quantile der Standardnormalverteilung, default=.05.

Eigentlich handelt es sich hier um einen t-Test gegen zwei Alternativen, da dies aber aufgrund des zusätzlichen delta nicht möglich ist, kann man hier nicht mit dem t-Test operieren.

Noch einfacher wäre die Berechnung eines 90%-igen Vertrauensintervalls und zu prüfen, ob dies innerhalb von $(-\delta, \delta)$ liegt.

Seite 79, Beispiel 1:

```
x1 <- 11.6
x2 <- 12.8
n1 <- 5
n2 <- 5
s <- 3.1623
t_exp1 <- (x1-x2+delta)/(s*sqrt(1/n1+1/n2))
t_exp2 <- (x1-x2-delta)/(s*sqrt(1/n1+1/n2))
t_tab <- qt(1-alpha, n1+n2-2)

------------------------------------------------------------------------

t_exp1
 [1] 1.899987

------------------------------------------------------------------------

t_exp2
 [1] -3.099978

------------------------------------------------------------------------

t_tab
 [1] 1.859548

------------------------------------------------------------------------

if(t_exp1 > t_tab & t_exp2 < -1*t_tab){
>> sehr langer Code <<
+   }

t_exp1  = 1.899987
t_exp2  = -3.099978
t_tab   = 1.859548

$`test decision:`
[1] "t_exp1 > t_tab & t_exp2 < -1*t_tab -> equivalence"
```

4 Die einfache Varianzanalyse

`anova()`: berechnet die Varianzanalyse (analysis of variance).

`lm()`: passt lineare Modelle an, sodass diese von `anova()` verwendet werden können. Andere Möglichkeiten ergeben sich mit `aov()` und `lme()`.

`summary()`: stellt eine Zusammenfassung dar.

`pairwise.t.test()`: berechnet multiple Mittelwertsvergleiche zwischen allen getesteten Mittelwerten.

4.1 Die Varianzanalyse-Tabelle (4.4)

Seite 91, Beispiel 1:

Um eine Varianzanalyse durchführen zu können, müssen alle Ergebnisse einer Behandlung in einen Vektor geschrieben werden und alle Behandlungen den Ergebnissen entsprechend in einen zweiten Vektor, sodass das Ergebnis an der selben Stelle im einen Vektor steht, wie die dazugehörige Behandlung im anderen Vektor.

```
Ertrag <- c(21, 34, 32, 24, 27, 23, 31, 23, 32, 29, 27, 37, 31,
 27, 32, 25, 19, 34, 26, 34)
Sorte <- c("B", "D", "D", "E", "C", "E", "A", "B", "A", "C",
 "E", "A", "D", "D", "A", "B", "B", "C", "E", "C")
```

Mit der Funktion `lm()` passt man an die Daten im Datenframe ein lineares Modell an und mit `anova()` lässt man sich die Varianzanalysetabelle ausgeben.

```
fit <- lm(Ertrag ~ Sorte)
```

```
anova(fit)
 Analysis of Variance Table

 Response: Ertrag
           Df Sum Sq Mean Sq F value    Pr(>F)
 Sorte      4 348.80   87.20  11.276 0.0001997 ***
 Residuals 15 116.00    7.73
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
 Warning message:
 In model.matrix.default(mt, mf, contrasts) :
   variable 'Sorte' converted to a factor
```

`Df` bezeichnet wieder die Freiheitsgrade, `Sum Sq` die Quadratsumme, `Mean Sq` das Mittelquadrat und `F value` den F_Vers-Wert. Neben dem F_Vers-Wert sieht man einen Wert, der mit `Pr(>F)` bezeichnet ist. Hier kann man sehr schnell, wie beim t-Test das Ergebnis der Varianzanalyse ablesen. In der letzten Zeile ist das Signifikanzniveau ersichtlich. Je kleiner ein p-Wert (Pr) ist, desto signifikanter sind die Unterschiede zu bewerten. `Signif.codes` gibt das Signifikanzniveau an. Im Beispiel ist der Test bei α=0,001 signifikant, weil hinter dem p-Wert drei Sterne stehen.

Demnach kann man hier höchst signifikante Sortenunterschiede feststellen.

Hier möchte ich kurz erklären, was es denn mit diesem `lm()` auf sich hat, da es mir selbst nicht leicht gefallen ist, das zu verstehen. Wie schon gesagt, werden hier die Daten an ein lineares Modell angepasst. „lineares Modell“ heißt in diesem Fall, eine Funktion der Form $\alpha + \beta * x$, wobei β die Steigung und α der y-Achsen-Abschnitt ist. `lm()` versucht nun also anhand der gegebenen Daten herauszufinden, welchen Wert β und α haben. Diese Werte kann man sich dann an eine Variable binden, wie hier zum Beispiel `fit`. Die Tilde (~) gibt an, dass das davor stehende die Zielvariable ist, die in Abhängigkeit des dahinter Stehenden angegeben werden soll (Menge der erklärenden Variablen), hier soll also `Ertrag` (Prädikatorvariable) in Abhängigkeit von `Sorte` (erklärende Variable) angegeben werden ($f(Ertrag) = \alpha + \beta * Sorte$).

Ein kleiner nützlicher Tipp am Rande:

```
summary(varan)
      Ertrag       Sorte
  Min.   :19.00   A:4
  1st Qu.:24.75   B:4
  Median :28.00   C:4
  Mean   :28.40   D:4
  3rd Qu.:32.00   E:4
  Max.   :37.00
```

Mit der Funktion `summary()` lässt sich ein schneller Überblick über Objekte verschaffen. Entsprechend dem Objekt werden hier nützliche Informationen aller Art dargestellt.

```
summary(phalombe)
        LM              CCA
  Min.   :0.50   Min.   :0.300
  1st Qu.:1.00   1st Qu.:0.425
  Median :1.25   Median :0.650
  Mean   :1.35   Mean   :1.164
  3rd Qu.:1.75   3rd Qu.:1.775
  Max.   :2.20   Max.   :3.500
```

4.2 Multiple Mittelwertvergleiche (4.5)

4.2.1 LSD-Test (4.5.1)

`pairwiset.t.test()`: erstellt einen LSD-Test.

Seite 93, Beispiel 1:

```
pairwise.t.test(Ertrag, Sorte, p.adjust="none")

        Pairwise comparisons using t tests with pooled SD

 data:  Ertrag and Sorte

   A       B       C       D
 B 5.1e-05 -       -       -
 C 0.32523 0.00036 -       -
 D 0.32523 0.00036 1.00000 -
 E 0.00101 0.14791 0.00808 0.00808

 P value adjustment method: none
```

Hier werden die p-Werte aller Mittelwerte zu allen anderen Mittelwerten dargestellt. Leider sind die Signifikanzen nicht direkt eingezeichnet, diese lassen sich aber anhand der bekannten Signifikanz-codes leicht ablesen. Man kann ablesen, dass keine Korrekturmethode berechnet wurde. Es gibt eine ganze Reihe Korrekturmethoden, die man mit dem Argument `p.adjust = ""` definieren kann, die aber im Statistik-Skript nicht behandelt wurden.

Will man Mittelwerte von Datensätzen unterschiedlicher Länge vergleichen, sind also die Varianzen nicht bei allen Datensätzen die selben, kann man den `oneway.test()` anwenden. Da im Statistik-Skript diese Art von Test aber nicht genauer erläutert wird, gehe ich an dieser Stelle auch nicht näher darauf ein.

4.2.2 Tukey-Test (4.5.3)

`aov()`: erzeugt ein Varianzanalyse-Modell (*formt die eingegebenen Daten in ein Modell um, dass zu Berechnungen mit Varianzanalysen benötigt wird*).

`TukeyHSD()`: erstellt einen Tukey-Test.

`order = FALSE/TRUE`: gibt an, ob die Daten vor dem Berechnen der größe nach sortiert werden sollen. Dann werden die Differenzen alle positiv.

Seite 98, Beispiel 1:

```
a <- aov(Ertrag ~ Sorte, varan)
```

```
TukeyHSD(a, ordered=TRUE)
   Tukey multiple comparisons of means
     95% family-wise confidence level
     factor levels have been ordered

 Fit: aov(formula = Ertrag ~ Sorte, data = varan)

 $Sorte
              diff        lwr       upr     p adj
 E-B 3.000000e+00 -3.0720458  9.072046 0.5626758
 D-B 9.000000e+00  2.9279542 15.072046 0.0028509
 C-B 9.000000e+00  2.9279542 15.072046 0.0028509
 A-B 1.100000e+01  4.9279542 17.072046 0.0004220
 D-E 6.000000e+00 -0.0720458 12.072046 0.0535237
 C-E 6.000000e+00 -0.0720458 12.072046 0.0535237
 A-E 8.000000e+00  1.9279542 14.072046 0.0076223
 C-D 3.552714e-15 -6.0720458  6.072046 1.0000000
 A-D 2.000000e+00 -4.0720458  8.072046 0.8436387
 A-C 2.000000e+00 -4.0720458  8.072046 0.8436387
```

Bei `diff` werden die Differenzen der Mittelwerte angegeben, wie sie auch im Statistik-Skript zu finden sind. Hier wird dann aber nicht, wie gewohnt, ein Wert für HSD ausgegeben, sondern ein p-Wert, der die Signifikanzen anzeigt (`p adj`). `lwr` und `upr` geben die untere und obere Grenze des Vertrauensintervalls für die Differenz der Mittelwerte an.

Buchstabendarstellung

Um Signifikanzen sinnvoll darzustellen, sind die oben genannten Methoden nicht die besten. Daher existiert das Pakte *multcompView*, dass die Buchstabendarstellung so durchführt, wie man das im Statistik-Skript nachlesen kann (Zur Installation von neuen Paketen in R kann man das Kapitel '↑ R: Ein paar nützliche Tipps lesen).

Als erstes werden die p-Werte aus dem Tukey-Test in einem Vektor erfasst

```
X <- c(0.5626758, 0.0028509, 0.0028509, 0.0004220, 0.0535237,
 0.0535237, 0.0076223, 1.0000000, 0.8436387, 0.8436387)
```

Dann werden die entsprechenden zu vergleichenden Behandlungsparameter in einem zweiten Vektor erfasst.

```
Z <- c("E-B", "D-B", "C-B", "A-B", "D-E", "C-E", "A-E", "C-D",
 "A-D", "A-C")
```

Der zweite Vektor dient als Namenbezeichnung für den ersten Vektor.

```
names(X) <- Z
```

```
X
      E-B       D-B       C-B       A-B       D-E       C-E
0.5626758 0.0028509 0.0028509 0.0004220 0.0535237 0.0535237
      A-E       C-D       A-D       A-C
0.0076223 1.0000000 0.8436387 0.8436387
```

Mit `multcompLetters()` kann man sich dann die Buchstabendarstellung anzeigen lassen.

```
multcompLetters(X)
    E    D    C    A    B
 "ab" "ac" "ac"  "c"  "b"
```

5 Einführung in die schließende Statistik für kategoriale Daten

5.1 Kombinatorik

Seite 100, Beispiel 1:

Den Ausdruck 3! (3 Fakultät) berechnet man wie folgt:

```
prod(3:1)
 [1] 6
```

Dieses Beispiel entspricht einer Variation ohne zurücklegen. Hier ist die Reihenfolge wichtig, da man einen randomisierten Versuchsaufbau haben will. Da man auf einer Parzelle nur eine Behandlung durchführen kann, gilt hier, dass man nicht zurücklegen darf.

Permutationen:

Die Grundfrage die sich bei den Permutationen stellt, ist die, wie viele mögliche Reihenfolgen sich für einen Anordnung von k Objekten ergeben. Die Permutationen einer experimentellen Situation ist von folgenden Faktoren abhängig:

1. Spielt die Reihenfolge, in der gezogen wird eine Rolle?
2. Wird eine Kugel wieder zurückgelegt, nachdem sie gezogen wurde?

(↑Formelübersicht)

`perm.rep()`: Variation (Reihenfolge wichtig) mit Zurücklegen.

`perm.norep()`: Variation (Reihenfolge wichtig) ohne Zurücklegen

`comb.rep()`: Kombination (Reihenfolge unwichtig) mit Zurücklegen

`comb.norep()`: Kombination (Reihenfolge unwichtig) ohne Zurücklegen.

`n =` : Anzahl Kugeln.

`k =` : Anzahl Ziehungen.

```
perm.rep(4, 3)
 [1] 64
```

```
perm.norep(5, 3)
 [1] 60
```

```
comb.rep(9, 6)
 [1] 3003
```

```
comb.norep(49, 6)
 [1] 13983816
```

```
choose(49, 6)
 [1] 13983816
```

`choose()` ist eine schon in R vorhandene Funktion, die genau das macht, was ich mit `comb.norep()` darstelle.

5.2 Einige wichtige Grundregeln der Wahrscheinlichkeitsrechnung

Seite 104, Beispiel 1:

```
b <- 5000/100000
```

```
b
 [1] 0.05
```

```
nb <- 1-b
```

```
nb
 [1] 0.95
```

Seite 104, Beispiel 2:

```
wd <- 100000
ww <- 60000
ik <- 40000
g <- wd + ww + ik

------------------------------------------------------------------------

g
 [1] 2e+05

------------------------------------------------------------------------

p1 <- wd/g

------------------------------------------------------------------------

p1
 [1] 0.5

------------------------------------------------------------------------

p2 <- ww/g

------------------------------------------------------------------------

p2
 [1] 0.3

------------------------------------------------------------------------

p3 <- ik/g

------------------------------------------------------------------------

p3
 [1] 0.2

------------------------------------------------------------------------

p1 + p2 + p3
 [1] 1
```

Seite 105, Beispiel 3:

```
p <- p2 + p3 # Additionssatz

------------------------------------------------------------------------
```

```
p
 [1] 0.5
```

Seite 105, Beispiel 3:

```
p <- 1/6

------------------------------------------------------------------------

p
 [1] 0.1666667

------------------------------------------------------------------------

p*p*p*p # Multiplikationssatz
 [1] 0.000771605
```

5.3 Binomialverteilung

Die Binomialverteilung funktioniert ähnlich wie die Normalverteilung, allerdings mit dem Suffix `...binom`.

`dbinom()`: gibt die Wahrscheinlichkeit für ein Einzelereignis der Verteilung an *(density)*. Nützlich, um Schaubilder einer Verteilung zeichnen zu lassen.

`pbinom()`: gibt die „Unterschreitungswahrscheinlichkeit“ eines Ereignisses an. Alle Werte unter dem Ereignis werden addiert und als Wahrscheinlichkeit ausgegeben. Nützlich um die Wahrscheinlichkeit aller Ereignisse in einem Intervall zu berechnen.

`qbinom()`: gibt die Quantile der Verteilung *(quantile)*.

`rbinom()`: gibt zufällige binomial verteilte Zahlen *(random)*.

`size =` : definiert die Größe der Stichprobe.

`prob =` : definiert die Erfolgswahrscheinlichkeit bei jedem einzelnen Versuch.

`lower.tail = TRUE/FALSE`: gibt statt Unterschreitungswahrscheinlichkeit die Überschreitungswahrscheinlichkeit wieder.

Seite 113, Beispiel 1:

```
pbinom(2, size=5, prob=0.75)
 [1] 0.1035156
```

```
------------------------------------------------------------------------

pbinom(4, 10, 0.75)
 [1] 0.01972771


n1 <- c(1:5)
n2 <- c(1:10)
n3 <- c(1:20)
brk1 <- c(0:5)
brk2 <- c(0:10)
brk3 <- c(0:20)
d1 <- 100*(dbinom(1:5, 5, .75))
d2 <- 100*(dbinom(1:10, 10, .75))
d3 <- 100*(dbinom(1:20, 20, .75))
hist(rep(n1, d1), brk1, ylab="P(X=k|n=5;p=0.75", xlab="k", freq=FALSE)
hist(rep(n2, d2), brk2, ylab="P(X=k|n=10;p=0.75", xlab="k", freq=FALSE)
hist(rep(n3, d3), brk3, ylab="P(X=k|n=20;p=0.75", xlab="k", freq=FALSE)
```

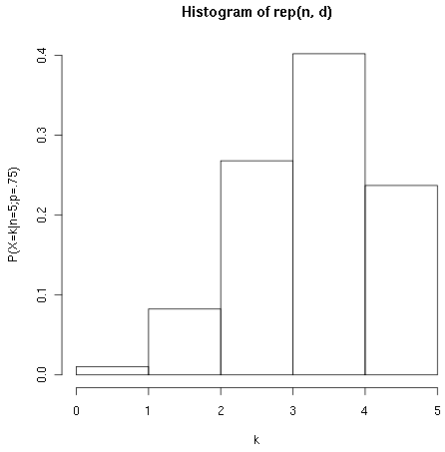


Abbildung 5.1: Binomialverteilung mit Stichprobenumfang 5

Seite 115, Beispiel 1:

```
dbinom(1:5, 5, 0.75)
 [1] 0.01464844 0.08789063 0.26367187 0.39550781 0.23730469
```

5.3.1 Mittelwert und Varianz der Binomialverteilung

Zur Berechnung des Mittelwertes und der Varianz einer Binomialverteilung kann man nicht dieselben Formeln benutzen, wie sie bei der Normalverteilung angewandt werden,

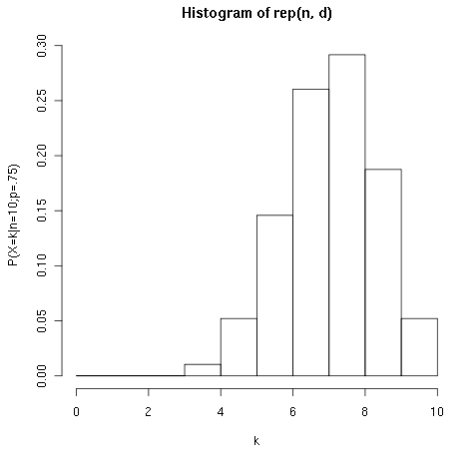


Abbildung 5.2: Binomialverteilung mit Stichprobenumfang 10

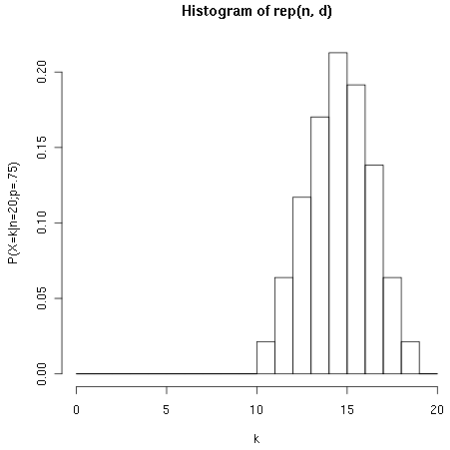


Abbildung 5.3: Binomialverteilung mit Stichprobenumfang 20

da es sich hier um kategoriale Daten handelt, aus denen man keinen Mittelwert bilden kann. Vielmehr muss man wissen, wie viel Prozent des Gesamtumfanges betroffen sind und kann dann mit den Formeln im Statistik-Skript von Hand weiterrechnen, oder vorher definierte Funktionen benutzen. Um die Arbeit auch hier zu vereinfachen, habe ich wieder eine eigene Funktion entwickelt.

(↑Formelübersicht)

`param.binom()`: berechnet einige Parameter zur Binomialverteilung.

`p =` : Prozentualer Wert der Erfolgswahrscheinlichkeit.

`success =` : Zählwert der Erfolgswahrscheinlichkeit.

`total =` : Stichprobenumfang.

Seite 116, Beispiel 1:

```
param.binom(success=5, total=100)

 p               = 0.05
 success         = 5 out of 100
 mean.binom      = 5
 var.binom       = 4.75
 var.p           = 0.000475
 sd.p            = 0.02179449
```

5.3.2 Schätzen des Parameters p der Binomialverteilung

Wie man schon oben gesehen hat, behandelt diese Formel alle relevanten Parameter dieses Kapitels, daher muss man lediglich die relevanten Daten einsetzen und bekommt auf bequeme Art und Weise seine Ergebnisse.

Seite 116, Beispiel 2:

```
param.binom(p=.07, total=100)

 p               = 0.07
 succes          = 7 out of 100
 mean.binom      = 7
 var.binom       = 6.51
 var.p           = 0.000651
 sd.p            = 0.0255147
```

5.3.3 Tests für den Parameter der Binomialverteilung

Da die Ressourcen bei einem Computerprogramm weitaus größer sind als wenn man von Hand rechnet, wendet man hier immer den Exakten Test an, da man dann genauere Ergebnisse herausbekommt.

`binom.test()`: erstellt einen exakten Test für den Parameter p der Binomialverteilung.

`alternative = c("two.sided","less", "greater")`: gibt an, ob man einen einseitigen oder einen zweiseitigen Test durchführen will. Standard ist `two.sided`, bei `less` wird die untere Region getestet, bei `greater` die Obere.

`conf.level =` : gibt das Vertrauensintervall manuell an, Standard ist 0.95.

Seite 118, Beispiel 1 (Beispiel 5.3.3.1):

```
binom.test(143, 567, .25)

         Exact binomial test

 data:  143 and 567
 number of successes = 143, number of trials = 567, p-value = 0.9227
 alternative hypothesis: true probability of success is not equal to 0.25
 95 percent confidence interval:
  0.2169468 0.2900671
 sample estimates:
 probability of success
              0.2522046
```

Seite 119, Beispiel 1:

```
binom.test(7, 20, .25)

         Exact binomial test

 data:  7 and 20
 number of successes = 7, number of trials = 20, p-value = 0.3055
 alternative hypothesis: true probability of success is not equal to 0.25
 95 percent confidence interval:
 0.1539092 0.5921885
 sample estimates:
 probability of success
                  0.35
```

5.3.4 Vertrauensintervall für den Parameter der Binomialverteilung

Auch das Vertrauensintervall für den Parameter p der Binomialverteilung lässt sich mit dem `binom.test()` herausfinden, wie man oben vielleicht schon gesehen hat

Seite 125, Beispiel 1:

```
binom.test(9, 10, 0.9)

         Exact binomial test

 data:  9 and 10
 number of successes = 9, number of trials = 10, p-value = 1
 alternative hypothesis: true probability of success is not equal to 0.9
 95 percent confidence interval:
  0.5549839 0.9974714
 sample estimates:
```

```
 probability of success
                  0.9
```

5.3.5 Vergleich von zwei Binomialwahrscheinlichkeiten – unverbundenen Stichprobe

Um zwei Binomialwahrscheinlichkeiten zu vergleichen, wendet man den `prop.test()` an

`prop.test()`: erstellt einen Test zum Vergleich zweier Binomialwahrscheinlichkeiten *(prop = proportion = Anteil/Prozentsatz)*

`alternative = c("two.sided","less", "greater")`: gibt an, ob man einen einseitigen oder einen zweiseitigen Test durchführen will. Standard ist `two.sided`, bei less wird die untere Region getestet, bei greater die Obere.

`conf.level =` : gibt das Vertrauensintervall manuell an, Standard ist 0.95.

`correct = TRUE/FAlSE`: gibt an, ob eine Yates-Korrektur durchgeführt werden soll.

```
success <- c(289, 274)
total <- c(316, 316)

------------------------------------------------------------------------

prop.test(success, total)

         2-sample test for equality of proportions with continuity
  correction

 data:  success out of total
 X-squared = 3.1887, df = 1, p-value = 0.07415
 alternative hypothesis: two.sided
 95 percent confidence interval:
  -0.004182592  0.099119301
 sample estimates:
    prop 1    prop 2
 0.9145570 0.8670886
```

5.3.6 Vergleich von zwei Binomialwahrscheinlichkeiten – verbundenen Stichprobe

Um zwei Binomialwahrscheinlichkeiten bei abhängigen Stichproben zu vergleichen ist der `mcnemar.test()` von nöten. Um den `mcnemar.test()` anwenden zu können, muss man die vorliegenden Daten in Form einer Matrix erfassen. Diese Matrix stellt die 2x2-Kontingenztafel dar.

`mcnemar.test()`: erstellt den McNemar-Test zum Vergleich zweier verbundener Binomialwahrscheinlichkeiten.

`correct = TRUE/FALSE`: gibt an, ob die Yates-Korrektur angewandt werden soll.

Seite 128, Beispiel 1:

```
x <- matrix(c(180, 73, 3, 10), nrow=2, dimnames = list("Methode A" =
c("+", "-"), "Methode B" = c("+", "-")))
```

```
x
            Methode B
Methode A    +   -
                   + 180  3
                   -  73  10
```

```
mcnemar.test(x)

         McNemar's Chi-squared test with continuity correction

 data:  x
 McNemar's chi-squared = 62.6447, df = 1, p-value = 2.476e-15
```

Seite 129, Beispiel 1:

```
x <- matrix(c(156, 18, 32, 37), nrow=2, dimnames = list("Methode A" =
c("+", "-"), "Methode B" = c("+", "-")))
```

```
x
               Methode B
 Methode A     +    -
                     + 156  32
                     -  18   37
```

```
mcnemar.test(x)

         McNemar's Chi-squared test with continuity correction

 data:  x
 McNemar's chi-squared = 3.38, df = 1, p-value = 0.06599
```

Wendet man die Korrektur an, so bekommt man einen nicht signifikanten Unterschied heraus, wendet man die Korrektur nicht an, ist das Ergebnis signifikant.

```
mcnemar.test(x, correct=FALSE)

         McNemar's Chi-squared test

 data:  x
 McNemar's chi-squared = 3.92, df = 1, p-value = 0.04771
```

5.4 Poissonverteilung

Die Poissonverteilung funktioniert ähnlich wie die Normalverteilung, allerdings mit dem Suffix ...pois.

`dpois()`: gibt die Wahrscheinlichkeit für ein Einzelereignis der Verteilung an *(density)*. Nützlich um Schaubilder einer Verteilung zeichnen zu lassen.

`ppois()`: gibt die „Unterschreitungswahrscheinlichkeit" eines Ereignisses an. Alle Werte unter dem Ereignis werden addiert und als Wahrscheinlichkeit ausgegeben. Nützlich um die Wahrscheinlichkeit aller Ereignisse in einem Intervall zu berechnen.

`qpois()`: gibt die Qunatile der Verteilung *(quantile)*.

`rpois()`: gibt zufällige binomial verteilte Zahlen *(random)*.

`lambda =` : gibt den Erwartungswert/Mittelwert an.

`lower.tail = TRUE/FALSE`: gibt statt Unterschreitungswahrscheinlichkeit die Überschreitungswahrscheinlichkeit wieder.

```
n <- (0:20)
brk <- c(-1:20)
p1 <- 100*dpois(0:20, 2)
p2 <- 100*dpois(0:20, 5)
p3 <- 100*dpois(0:20, 10)
hist(rep(n, p1), brk, freq=FALSE, ylab="P(X=k)", main="Lambda=2")
hist(rep(n, p2), brk, freq=FALSE, ylab="P(X=k)", main="Lambda=5")
hist(rep(n, p3), brk, freq=FALSE, ylab="P(X=k)", main="Lambda=10")
```

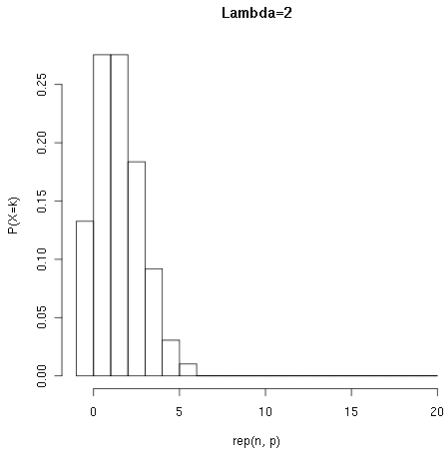


Abbildung 5.4: Poissonverteilung mit Mittelwert 2

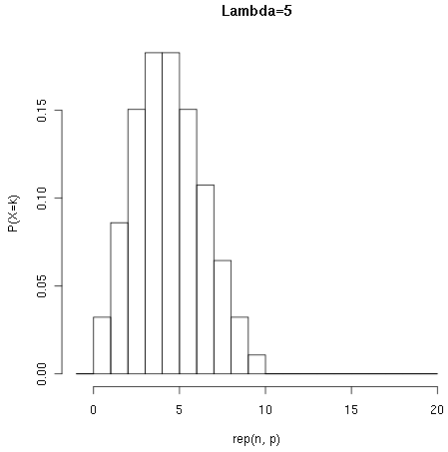


Abbildung 5.5: Poissonverteilung mit Mittelwert 5

5.4.1 Parameterschätzung (5.4.2)

Schätzen des Parameters der Poisson-Verteilung

Der Parameter der Poisson-Verteilung lässt sich auch als ein gewichteter Mittelwert verstehen. Dabei sind zwei Vektoren von Interesse, nämlich die Werte, die beobachtet wurden, und die Gewichtung dieser Werte. Als ersten Vektor definiert man also die Zählwerte von k und als zweiten Vektor definiert man die beobachtete Häufigkeit B.

`weighted.mean()`: berechnet den gewichteten Mittelwert.

Die „Gewichtung“ der Zählwerte findet in Form einer Multiplikation statt, wie sie auch im Statistik-Skript zu sehen ist. Hier ist wichtig zu beachten, dass die Zählwerte an der selben Stelle im ersten Vektor stehen wie die dazugehörige beobachtete Häufigkeit im zweiten Vektor.

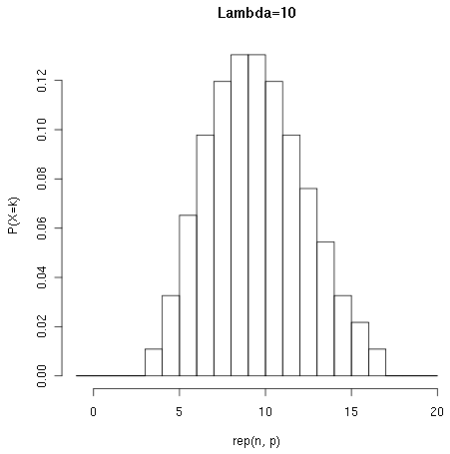


Abbildung 5.6: Poissonverteilung mit Mittelwert 10

Seite 143, Beispiel 1:

```
k <- c(0:4)
B <- c(158, 73, 21, 3, 1)
```

```
weighted.mean(k, B)
 [1] 0.5
```

Erwartete Häufigkeit einer Poisson-Verteilung

(↑Formelübersicht)

`ek()`: berechnet die erwartet Häufigkeit einer Poisson-Verteilung.

`n =` : Anzahl.

`k =` : Anzahl k.

`l =` : lambda.

`prec =` : Werte, deren Erwartungswert < prec sind, werden zu einer Klasse zusammengefasst. Wenn ein E(k=x) < prec wird, dann werden alle Klassen mit k>x zu einer Klasse zusammengefasst. Der zurückgegebene Vektor lässt sich dann mit vec[„1“] oder vec[„0..5“] oder ... bzw. vec[1], vec[5], vec[n] indizieren; default = 1.

`plot = TRUE/FALSE`: logisch, gibt an, ob ein Schaubild ausgegeben werden soll oder nicht; default = `TRUE`.

`script = TRUE/FALSE`: logisch; `TRUE` = Sollen die Klassen so zusammengefasst werden wie im Statistik-Skript d.h. alle E_k > prec werden zur nächsten Klasse dazugenommen? `FALSE` = alle E_k < prec werden zu einer eigenständigen Klassse zusammengefasst; default = `TRUE`.

`dd =` : Dezimalstellen, auf die gerundet werden soll; default = 3.

Seite 144, Beispiel 1:

```
E <- ek(n=256, k=4, l=0.5, plot=FALSE)
B <- c(158, 73, 21, 4)

------------------------------------------------------------------------

data.frame(B, E)
        B       E
 0    158 155.272
 1     73  77.636
 2     21  19.409
 3...   4   3.683
```

subsubsectionVertrauensintervall für den Parameter einer Poisson-Verteilung

(↑Formelübersicht)

`ci.lambda()`: berechnet das Vertrauensintervall für den Parameter einer Poisson-Verteilung.

`n =` : Anzahl Proben.

`lambda =` : Gewichteter Mittelwert lambda der Poisson-Verteilung.

`alpha =` : Irrtumswahrscheinlichkeit zur Berechnung der Quantile der Poisson-Verteilung; default=.05.

Seite 145, Beispiel 1:

```
ci.lambda(256, 0.5)
 [1] 0.4171324 0.5946144
```

Seite 145, Beispiel 2:

```
k <- c(0:5)
B <- c(1, 1, 2, 0, 0, 1)
l <- weighted.mean(k, B)
```

```
------------------------------------------------------------------------

l
 [1] 2

------------------------------------------------------------------------

ci.lambda(5, l)
 [1] 0.9524829 3.6921660
```

Um hier lambda berechnen zu können, muss man die Zählwerte in die empirische Verteilung umwandeln. Dazu erstellt man einfach einen Vektor, der die relevanten Zählwerte enthält und zählt ab, wie oft diese vorkommen. Daraus erstellt man den zweiten Vektor.

Seite 145, Beispiel 3:

```
ci.lambda(98, 3.02)
 [1] 2.685736 3.384648
```

5.4.2 Test für den Vergleich zweier Parameter $\lambda 1$ und $\lambda 2$ (5.4.3)

Da bei einer Poisson-Verteilung keine obere Grenze festzustellen ist, kann man hier leider den `prop.test()` nicht verwenden. Vielmehr muss man sich seine eigenen Funktion erstellen.

(↑Formelübersicht)

`lambda.test()`: berechnet den Test zum Vergleich zweier Parameter $\lambda 1$ und $\lambda 2$.

`l1 =` : Poisson-Parameter der ersten Gruppe.

`l2 =` : Poisson-Parameter der zweiten Gruppe.

`n1 =` : Stichprobenumfang der ersten Gruppe.

`n2 =` : Stichprobenumfang der zweiten Gruppe.

`alpha =` : Irrtumswahrscheinlichkeit zur Berechnung der Quantile der Poisson-Verteilung; default=.05.

Seite 146, Beispiel 1:

```
lambda.test(3.02, 3.75, 98, 95)

 z_exp   = 2.758062
 z_tab   = 1.959964
```

```
$`test decision:`
[1] "z_exp > z_tab -> significance"
```

5.5 Chi-Quadrat-Anpassungstest

Hier wollen wir wissen, ob eine beobachtete Verteilung mit einer theoretischen Verteilung übereinstimmt. Um dies herauszufinden, wendet man den `chisq.test()` an. Dieser Test wird angewendet, um zu überprüfen, ob eine empirische Verteilung mit einer theoretischen Verteilung übereinstimmt, um Signifikanz in einer 2*2-Feldertafel nachzuweisen, oder sogar um Signifikanz in einer r*c-Feldertafel nachzuweisen. Da die bisherige Version von `chisq.test()` die Freiheitsgrade ohne Berücksichtigung der geschätzten Parameter berechnet, habe ich dies nun in die Funktion mit aufgenommen. In diesem Fall binde ich die neue Funktion wieder an `chisq.test()`, da der alte Code fehlerhaft war und nicht mehr benötigt wird.

(↑Formelübersicht)

`chisq.test()`: erstellt einen χ^2-Kontigenztafel-Test und einen Anpassungstest.

`correct = TRUE/FALSE`: gibt an, ob die Yates-Korrektur angewandt werden soll.

`p =` : gibt einen Vektor der Wahrscheinlichkeiten an, mit denen die Daten verglichen werden sollen.

`q =` : Anzahl der zu schätzenden Parameter.

Seite 154, Beispiel 1:

Will man eine empirische Verteilung gegen eine angenommene Verteilung testen, so muss man die empirisch ermittelten Daten in einen Vektor schreiben. Die angenommene Verteilung muss man in Form von Wahrscheinlichkeiten angeben. `chisq.test()` vergleicht nun die empirischen Daten mit den erwarteten Daten.

```
E <- ek(256, 4, 0.5, plot=FALSE)
p <- E/sum(E)
```

```
p
          0           1           2       3...
 0.60653125 0.30326562 0.07581641 0.01438672
```

```
B <- c(158, 73, 21, 4)

------------------------------------------------------------------------

chisq.test(B, q=1, p=p)

         Chi-squared test for given probabilities

 data:  b
 X-squared = 0.4825, df = 2, p-value = 0.7857

 Warning message:
 In chisq.test(b, q = 1, p = p) : Chi-squared approximation may be
  incorrect
```

5.5.1 Berechnung der erwarteten Häufigkeiten für eine Binomialverteilung

Schätzen des Parameters p der Binomialverteilung bei Messwiederholungen

Leider existiert für den Gewichteten Mittelwert bei Binomialverteilungen wieder keine Formeln, daher habe ich wieder selbst eine definiert.

(↑Formelübersicht)

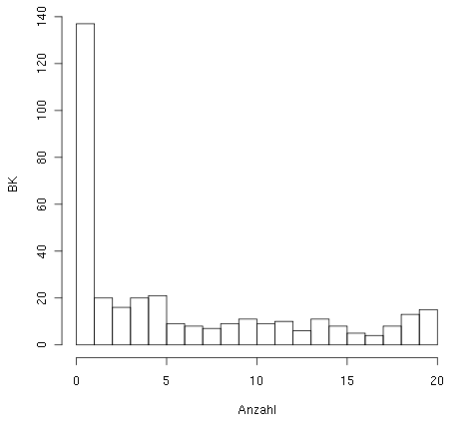
`weighted.mean.binom()`: berechnet den gewichteten Mittelwert einer Binomialverteilung.

`x =` : Vektor der Werte, dessen gewichteter Mittelwert berechnet werden soll.

`w =` : Vektor der Gewichtungen, muss die selbe Länge haben wie x.

Seite 159, Beispiel 1:

```
k <- 0:20
B <- c(104, 33, 20, 16, 20, 21, 9, 8, 7, 9, 11, 9, 10, 6, 11, 8, 5, 4,
 8, 13, 15)

------------------------------------------------------------------------

weighted.mean.binom(k, B)
 [1] 0.2987032

------------------------------------------------------------------------

hist(rep(k, B), breaks=c(20), xlab="Anzahl", ylab="Bk", main=" ")
```



Erwartete Häufigkeit einer Binomial-Verteilung

(↑Formelübersicht)

`rec.binom()`: berechnet die erwartete Häufigkeit einer Binomialverteilung.

`h =` : Summe aller beobachteten Häufigkeiten.

`p =` : Wahrscheinlichkeit der Binomialverteilung.

`n =` : Stichprobenumfang der Probe.

`k =` : Zählwert der beobachteten Häufigkeit.

Hier muss man sich jeden Wert selbst errechnen und kann so auch gleich sehen, ob ein kleiner oder großer Wert herauskommt, den man dann mit den anderen kleinen/großen Werten in eine Klasse nehmen kann.

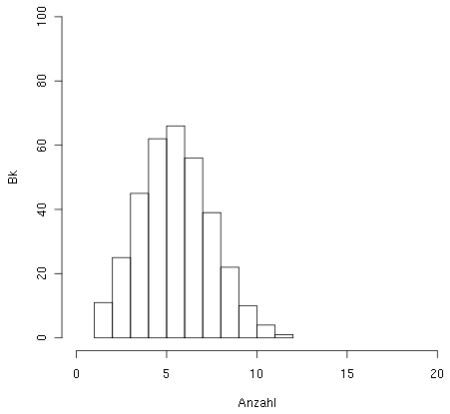
Seite 161, Beispiel 1:

```
rec.binom(347, 0.2987, 20, 0)
 [1] 0.2873467
rec.binom(347, 0.2987, 20, 1)
 [1] 2.447753
rec.binom(347, 0.2987, 20, 2)
 [1] 9.904273
rec.binom(347, 0.2987, 20, 3)
 [1] 25.31076
rec.binom(347, 0.2987, 20, 4)
 [1] 45.81689
rec.binom(347, 0.2987, 20, 5)
 [1] 62.44633
rec.binom(347, 0.2987, 20, 6)
 [1] 66.49337
```

```
rec.binom(347, 0.2987, 20, 7)
 [1] 56.64215
rec.binom(347, 0.2987, 20, 8)
 [1] 39.20346
rec.binom(347, 0.2987, 20, 9)
 [1] 22.26356
rec.binom(347, 0.2987, 20, 10)
 [1] 10.43082
rec.binom(347, 0.2987, 20, 11)
 [1] 4.038846
rec.binom(347, 0.2987, 20, 12)
 [1] 1.290179
rec.binom(347, 0.2987, 20, 13)
 [1] 0.3381645
rec.binom(347, 0.2987, 20, 14)
 [1] 0.07201607
rec.binom(347, 0.2987, 20, 15)
 [1] 0.01226933
rec.binom(347, 0.2987, 20, 16)
 [1] 0.00163306
rec.binom(347, 0.2987, 20, 17)
 [1] 0.0001636608
rec.binom(347, 0.2987, 20, 18)
 [1] 1.161782e-05
rec.binom(347, 0.2987, 20, 19)
 [1] 5.208738e-07
rec.binom(347, 0.2987, 20, 20)
 [1] 1.109261e-08
```

```
k <- 0:20
B <- c(0.2873467, 2.447753, 9.904273, 25.31076, 45.81689, 62.44633,
 66.49337, 56.64215, 39.20346, 22.26356,10.43082, 4.038846, 1.290179,
 0.3381645, 0.07201607, 0.01226933, 0.00163306, 0.0001636608,
 1.161782e-05, 5.208738e-07, 1.109261e-08)
```

```
hist(rep(k, B), breaks=c(12), xlab="Anzahl", ylab="Bk", main=" ",
 ylim=range(0,100), xlim=range(0,20))
```



5.6 Test auf Unabhängigkeit in der 2 * 2 Feldertafel (4-Feldertafel)

5.6.1 Test bei großen Stichproben

Seite 165, Beispiel 1:

```
x <- matrix(c(56, 6759, 272, 11396), 2, byrow=TRUE, dimnames =
 list("Impfung"=c("ja", "nein"), "Gesundheitszustand"=c("erkrankt",
 "gesund")))
```

```
x
           Gesundheitszusatnd
 Impfung erkrankt gesund
              ja               56       6759
        nein             272     11396
```

```
chisq.test(x, correct=FALSE)

         Pearson's Chi-squared test

 data:  x
 X-squared = 56.2341, df = 1, p-value = 6.434e-14
```

Will man nicht, dass die Yates-Korrektur durchgeführt wird, so muss man diese deaktivieren, indem man `correct =` auf `FALSE` setzt.

Hier ist sehr schön zu sehen, dass der Test ein höchst signifikantes Ergebnis liefert, die Merkmale Impfung und Gesundheitszustand also auf jeden Fall abhängig von einander

sein müssen.

5.6.2 Die Yates-Korrektur

Seite 169, Beispiel 1:

```
x <- matrix(c(5, 9, 15, 3), 2, byrow=TRUE, dimnames =
 list("Wasser"=c("Lehmwasser", "Regenwasser"), "Keimverhalten"=c("nicht
 gekeimt", "gekeimt")))
```

```
x
                                Keimverhalten
              Wasser    nicht gekeimt gekeimt
   Lehmwasser                                5               9
 Regenwasser                                15               3
```

```
chisq.test(x)

         Pearson's Chi-squared test with Yates' continuity correction

 data:  x
 X-squared = 5.7228, df = 1, p-value = 0.01675
```

```
chisq.test(x, correct=FALSE)

         Pearson's Chi-squared test

 data:  x
 X-squared = 7.619, df = 1, p-value = 0.005775
```

Wie man sieht, ist die Yates-Korrektur als Standard aktiviert. Deaktiviert man diese Korrektur, verfälscht sich das Ergebnis.

5.6.3 Fishers exakter Test

Seite 169, Beispiel 2:

`fisher.test()`: erstellt den Fisher-exakt-Test.

```
x <- matrix(c(4, 1, 3, 7), 2, byrow=TRUE, dimnames=list("Behandlung"=
 c("X", "Y"), "Ergebnis"=c("Erfolg", "Kein Erfolg")))
```

```
------------------------------------------------------------------------

x
                          Ergebnis
 Behandlung Erfolg Kein Erfolg
                      X           4                     1
                      Y           3                     7

------------------------------------------------------------------------

fisher.test(x)

         Fisher's Exact Test for Count Data

 data:  x
 p-value = 0.1189
 alternative hypothesis: true odds ratio is not equal to 1
 95 percent confidence interval:
    0.5027782 521.2731861
 sample estimates:
 odds ratio
   7.876343
```

Eine kleine Erörterung zur `odds ratio` (Chancen-Verhältnis): Die „odds“ = Chancen werden beschrieben durch:

$$\frac{p}{1-p}$$

Die dazugehörige Wahrscheinlichkeit errechnet sich nach:

$$p = \frac{Erfolg}{Erfolg + kein\ Erfolg}$$

Beispiel:
Spielt man ein Spiel, bei dem fünf Spielkarten vorliegen, aus denen man Eine bestimmte ziehen muss, so spricht man von einer Gewinnchance von 1:4, da die Chance diese eine Karte nicht zu ziehen viermal so hoch ist, als die Chance sie zu ziehen. Die prozentuale Wahrscheinlichkeit von 1/5 beträgt 0,2 oder 20%.

Damit liegen die odds in diesem Beispiel also bei 0,2/(1-0,2) = 0,2/0,8 = 0,25. Müsste man Eine aus Vier zeihen, so lägen die odds bei 0,25/0,75 = 0,33, bei Einer aus Drei bei 0,33/0,66 = 0,5 und bei Einer aus Zwei bei 0,5/0,5 = 1. Hier spiegelt sich das Sprichtwort einer 50:50-Chance wieder.

Das Verhältnis, also die „odds-ratio“, ist der Quotient aus zwei verschiendenen Chancen X und Y:

$$oddsratio = \frac{odds_1}{odds_2}$$

In unserem Beispiel sieht dass dann so aus:

$$p_1 = 4/5, 1 - p_1 = 1/5, odds_1 = \frac{4/5}{1/5} = 4$$

$$p_2 = 3/10, 1 - p_2 = 7/10, odds_2 = \frac{3/10}{7/10} = 3/7$$

$$odds\ ratio = \frac{4}{3/7} = 9,33$$

Hier merkt man sofort, dass die von Hand errechnete odds-ratio eine andere ist, als die des `fischer.test()`. Scheinbar handelt es sich hierbei um einen Bug, der so schnell wohl erstmal nicht behoben werden wird. Der p-Wert dieses Tests ist aber durchaus zuverläsig.

Seite 172, Beispiel 1:

```
x <- matrix(c(5, 9, 15, 3), 2, byrow=TRUE, dimnames =
 list("Wasser"=c("Lehmwasser", "Regenwasser"), "Keimverhalten"=c("nicht
 gekeimt", "gekeimt")))

------------------------------------------------------------------------

fisher.test(x)

        Fisher's Exact Test for Count Data

 data:  x
 p-value = 0.009998
 alternative hypothesis: true odds ratio is not equal to 1
 95 percent confidence interval:
  0.01473996 0.72404093
 sample estimates:
 odds ratio
  0.1207754
```

5.7 Test auf Unabhängigkeit in einer r * c Tafel

Auch bei dem Test auf Unabhängigkeit in einer r*c Tafel verwendet man den `chisq.test()`. Dazu muss man die Daten wieder in Form einer Matrix erfassen und diese Matrix dann durch `chisq.test()` laufen lassen.

Seite 179, Beispiel 1:

```
Bf <- c(36, 67, 49, 31, 60, 49, 58, 87, 80)
m <- matrix(Bf, nrow=3, byrow=TRUE, dimnames=list("Bodentyp"=c("I", "II",
 "III"), "Besitzform"=c("Eigenbesitz", "Pacht", "Gemischt")))

------------------------------------------------------------------------

m
```

```
         Besitzform
 Bodentyp Eigenbesitz Pacht Gemischt
      I            36    67       49
      II           31    60       49
      III          58    87       80
```

```
chisq.test(m)

         Pearson's Chi-squared test

 data:  m
 X-squared = 1.5431, df = 4, p-value = 0.819
```

Bei einem p-Wert von 0.819 kann die Nullhypothese der Unabhängigkeit der beiden Merkmale also nicht verworfen werden.

5.7.1 Exakter Test

Auch hier kann man den `fisher.test()` anwenden.

Vorwort zum Teil Biometrie

Dieses Skript ist die Fortsetzung des Skripts „R für Vl Statistik“, genauso wie Herrn Piephos Skript zur Biometrie I/II (A4002/A4003) die Fortsetzung zu seinem Skript zur Statistik ist.

Auch hier wird die Nummerierung an der Stelle weitergeführt, an der sie in „R für Vl Statistik“ aufgehört hat. Allerdings finden sich hier keine Überschneidungen, das Skript zum Teil Statistik geht nahtlos in das Skript zum Teil Biometrie über.

Auch in diesem Skript findet sich eine Aufteilung in zwei Teile. Alles für R relevante Wissen wird unter Kapiteln mit der Überschrift: *„R: ...“* zusammengefasst, die Kapitel aus dem Biometrie-Skript behalten ihre von Herrn Piepho zugeteilten Kapitelnummern, sodass das Finden der gesuchten Kapitel erleichtert wird. In der Überarbeiteten Version von 'R für Vl Statistik' und in diesem Skript ist kein Beispielverzeichnis mehr vorhanden, da man die Beispiele auch über ihre Kapitel eindeutig finden kann.

Im Anhang werden sich allerdings weiterhin alle Daten und Formeln befinden, die hier verwendet werden.

Generell sollte man das Skript 'R für Vl Statistik' durchlesen, da die dort schon erklärten Anwendungsschritte und das Hintergrundwissen als Grundlage voraus gesetzt werden. Einiges aus dem ersten Skript wende ich heute nicht mehr in der Art an, wie ich das damals getan habe. Je länger man sich mit einer Materie beschäftigt, desto eher ändern sich einige Verhaltensweisen, da man dieses oder jenes als praktischer oder schneller anzuwenden erachtet. Da R eine sehr breite Palette an Möglichkeiten bietet, ist die von mir im Skript 'R für Vl Statistik' oder hier dargestellte Methodik nicht unbedingt die Einzige oder am besten angepasste.

Die bereits erstellten R-Skripte `Daten.R` und `Formeln.R` werden hier wieder verwendet und weiter mit Daten und Formeln ergänzt. Ich habe diese zwei Skripte allerdings zu einem Skript (`Input.R`) zusammengefasst, was meinen Arbeitsablauf wesentlich vereinfacht.

In einigen Kapiteln werde ich versuchen, die Logik hinter R ein bisschen besser zu beleuchten. Sonst werden zwar die wichtigsten Funktionen von R dargestellt, will man allerdings tiefer gehend in die Materie eintauchen, erfordert das ein gewisses Detailwissen, dass man sich wesentlich leichter erarbeiten kann, wenn man die Logik hinter den von R angewandten Prozessen versteht.

Außerdem sind in der überarbeiteten Version des Skriptes 'R für Vl Statistik' und in diesem Skript die Kochbuchkästen nicht mehr vorhanden. Stattdessen habe ich mich dazu entschieden, diese Informationen in einer Stichpunktliste zu erfassen. Der Informationsgehalt bleibt der selbe, lediglich die Darstellungsform ändert sich dadurch. Allerdings muss ich dazu sagen, dass ich in diesem Teil des Skriptes an vielen Stellen zu Fließtext übergegangen bin, da Zusammenhänge sich in dieser Form leichter erklären lassen, als

in stichpunktartiger Form.
Des weiteren sind die von mir definierten Funktionen nicht mehr direkt im Text zu finden, da das meiner Meinung nach den Textfluss gestört hat. Die Funktionen sind allerdings nach wie vor im Anhang zu finden. Um klar zu machen, an welcher Stelle ich eine selbst definierte Funktion benutze, habe ich mit `(Formelübersicht)` am Anfang der entsprechenden Auflistung der Argumente der Funktion einen Link zur Funktion gesetzt. Vom Formelverzeichnis aus, kann man mit Klick auf die Überschrift zur Funktion wieder in das Kapitel zurück gelangen, von dem man gekommen ist, beziehungsweise zu dem die Funktion gehört.

Für eine Übersicht über alle Funktionen, die in R regelmäßig Anwendung finden kann man im Internet sogenannte 'Reference Cards' zu R erhalten. Eine recht umfangreiche 'Refcard' findet man von ↑ Tom Short (PDF, 71 KB).

Weiteres Vorlesungesmaterial und eine Übersicht über das bioinformatische Institut der Universität Hohenheim kann man unter der folgenden URL finden:

http://www.uni-hohenheim.de/bioinformatik/

R: Weiterführende Funktionen

Alle, die gerne mit Konsolen arbeiten, also dem PC lieber schreibend die Anweisungen erteilen, anstelle dauernd zur Maus greifen zu müssen, können in diesem Skript etwas über „Anweisungen für Fortgeschrittene“ lesen.

Unter Linux ist dies sogar die einfachere Möglichkeit sein Ziel zu erreichen, da ein geübter Konsolen-Nutzer alle nötigen Anweisungen im Kopf hat und so viel schneller arbeiten kann, als ein Maus-Nutzer. Wird man also oft mit R oder vergleichbaren konsolenbasierten Programmen arbeiten, kann man sich schon von Anfang an angewöhnen die Konsole zu nutzen, da man so, im Vergleich zur Mausnutzung, sehr viel Zeit sparen kann (genau die Zeit, die man braucht um von der Tastatur zur Maus zu greifen, den Zeiger zur richtigen Stelle zu schieben und zu klicken). Die Eingewöhnungszeit scheint sehr lange und die Bedienung über die Konsole scheint ungewöhnlich, aber genauso wie wir uns einst an die Maus gewöhnen mussten, kann man sich auch an die Konsole gewöhnen.

options()

Ist R gestartet kann man sich mit `options()` alle eingestellten Optionen zum Arbeiten mit R anzeigen lassen. Will man sich eine spezielle Option ansehen oder verändern, muss man deren Namen herausfinden und mit `options("name")` anwählen.
Man möchte also beispielsweise das Komma-Zeichen verändern:

```
options("OutDec")
 $OutDec
 [1] "."
```

Um die Option zu verändern, muss man ihr mit der Form `options(name="xy")` einen neuen Wert zuteilen. Gültige Werte für die verschiedenen Optionen finden sich in der Dokumentation `?options`.

```
options(OutDec=",")
options("OutDec")
 $OutDec
 [1] ","
```

dir()/ls()

Die Funktionen `getwd()` und `setwd()` sind schon bekannt, will man aber nun wissen, was sich sonst noch im Arbeitsverzeichnis befindet, benötigt man die Funktion `dir()`. Hier werden alle Dateien alphabetisch sortiert aufgelistet, die sich im Arbeitsverzeichnis befinden.

Mit `ls()` kann man alle Objekte auflisten, die bereits als Variablen verwendet werden und sowohl vom Typ 'Funktion' sein können, als auch von jedem anderen Typ. Erstellen von R-Skripten kann manuell durchgeführt werden. Dazu erzeugt man im Arbeitsverzeichnis eine Datei mit dem Namen des R-Skriptes (also `Input.R`), kopiert die Daten und Formeln dort hinein und bindet sie mit `source("Input.R")` ein.

head()/tail()

Die Funktion `head()` ist interessant, wenn man es mit großen Elementen wie Listen oder Datenframes zu tun hat. Als Output werden dabei der Kopf des Elementes und die ersten `n` Zeilen angezeigt. `n` kann man in `head()` selbst definieren, Standard ist `n = 6`. `tail()` zeigt respektive das Ende eines Elementes.

png()/savePlot()

Bei der Funktion `png()` handelt es sich um die Ansteuerung eines Devices (mehr Informationen zu Devices mit `?Devices`). `png()` sagt R, dass man ein Bild des Formates *.png erstellen möchte. Schaubilder können in R in den Formaten *.jpeg, *.bmp, *.tiff, *.png, etc gespeichert werden. Welches Format man bevorzugt, muss man selbst entscheiden. Eine kleine Entscheidungshilfe findet man zum Beispiel mit `?png` unter Details. Alles was nach dieser Funktion geschrieben wird und relevant für ein Schaubild ist, wird nun also in das Schaubild aufgenommen. Da R nicht automatisch weiß, wann das Schaubild fertiggestellt ist, muss man dem Programm mit der Funktion `dev.off()` mitteilen, dass man nun alle für das Schaubild relevanten Daten eingegeben hat und den `Device` beenden will. In diesem Moment werden die Informationen, die aufgelistet wurden endgültig in das Schaubild eingezeichnet und die Datei wird gespeichert.
Eine leichtere und weniger schreibaufwändige Methode bietet die Funktion `savePlot()`. Hiermit kann man, nachdem man ein Schaubild in R erstellt hat, dieses ganz einfach speichern, indem man die Funktion mit den entsprechenden Argumenten angibt. Als Argumente ist dabei das Format des zu speichernden Schaubildes unerlässlich, außerdem kann man den Namen selbst definieren.

class()/is.()

Mit der Funktion `class()` kann man überprüfen, welche Klasse ein Objekt hat.
`class()` ist vor allem dann oft relevant, wenn man selbst Funktionen entwirft oder

sich einen Überblick über bestimmte Objekte verschaffen will.

Will man überprüfen, ob ein Objekt eine bestimmte Klasse hat, kann man eine Funktion benutzen, die sich aus `is.` und der entsprechenden Klasse zusammensetzt. Will man beispielsweise wissen, ob etwas eine Matrix ist, gibt man `is.matrix()` ein.

rm()

Diese Funktion kann man nutzen um jedes beliebige, vorher definierte, Objekt zu entfernen. Sie wird immer dann benötigt, wenn die Variable komplett entfernt werden muss. Meistens reicht jedoch überschreiben mit einem neuen Objekt.

read.table()

Da die Daten aus Herrn Piephos Skripten nicht in digitaler Form vorliegen, musste ich diese von Hand in die Konsole abtippen. Dies ist jedoch nicht die übliche Vorgehensweise um Daten in R einzulesen. Meist werden die Daten in Excel-Tabellen gesammelt. Diese sollten für die Verwendung in R dann in eine Textdatei (*.txt) umgewandelt werden. Mit der Funktion `read.table()` kann man die Daten dann in einen `data.frame` umwandeln. Für mich ist es einfacher diese Daten wie gehabt einzulesen, um die Standardroutine zu verdeutlichen, werde ich diese an einigen Stellen „simulieren“.

library() oder require()

Will man sich mit Funktionen beschäftigen, die nicht standardmäßig in R enthalten sind, muss man diese Funktionen erst verfügbar machen. Oft ist es so, dass mehrere Funktionen eines Anwendungsbereiches in einem Paket zusammengefasst sind. Auf der CRAN-Seite sind alle Pakete aufgelistet, hier kann man auch nach Paketen suchen, sollte man etwas spezielles vorhaben. Wie Pakete in R hinzugefügt werden, wurde in Kapitel '↑ R: Ein paar nützliche Tipps' bereits beschrieben, wie man mit dem Standard-GUI von R Pakete installiert. Alternativ kann man dies über Funktionen in der Kommandozeile von R steuern.

Oftmals sind von der letzten gespeicherten Session die Pakete noch installiert, diese müssen aber neu geladen werden. Dies geschieht mit den Funktionen `library()` oder `require()`.

```
library(multcomp)
 Lade nötiges Paket: mvtnorm
 Lade nötiges Paket: survival
 Lade nötiges Paket: splines
```

methods()

Mit der Funktion `methods()` kann man sich alle verfügbaren Methoden für eine generische Funktion oder für eine Klasse anzeigen lassen.

```
methods(glht)
 [1] glht.character*  glht.expression* glht.matrix*     glht.mcp*
 [5] glht.means*

    Non-visible functions are asterisked
```

Die so aufgelisteten Begriffe können zur Suche mit `?Begriff` genutzt werden um weiter Informationen zu erhalten.

Matrixoperationen

Eine Anwendung der hier aufgeführten Funktionen kann im Kapitel '↑ 6.7.6 Die Normalengleichung und ihre Lösung (6.8.1)' gefunden werden.

t()/ginv()/solve()/det()

Mit `t()` lässt sich die Transponierte berechnen. Mit `ginv()` kann man die generalisierte Inverse einer Matrix berechnen (`ginv()` befindet sich im Paket `MASS`, dass vor Anwendung dieser Funktion geladen werden muss).`solve()` berechnet die einfache Inverse, falls diese existiert. Die Determinante kann mit `det()` berechnet werden.

eigen()

Die Funktion `eigen()` dient zur Berechnung der Eigenvektoren beziehungsweise Eigenwerte. Die Eigenwerte werden mit den Eigenvektoren ausgegeben, wünscht man nur die Eigenwerte, muss man das Argument `only.values = ` auf `TRUE` setzen.

Operatoren

In R kann eine Matrizenmultiplikation durchgeführt werden, indem man die `%*%`-Notation verwendet. Addition und Subtraktion sind mit den einfachen Operatoren durchführbar. Außerdem existiert die Funktion `kronecker()`, mit der das ↑ Kronecker-Produkt zweier Matrizen (A und B mit den Dimensionen $m \times n$ und $p \times r$) berechnet werden kann. Dabei wird jedes Element der ersten Matrix (A) mit der zweiten Matrix (B) multipliziert. Heraus kommt eine Matrix C mit den Dimensionen $mp \times nr$

Randsummen/-mittelwerte

Randsummen können in R mit den Funktionen `colSums()` und `rowSums()` berechnet werden. Die Randmittelwerte können mit `colMeans()` und `rowMeans()` berechnet werden.

R: Zur graphischen Darstellung in R

GNU R bietet sehr viele Möglichkeiten und Spielereien, die empirischen Werte graphisch darzustellen. In diesem Kapitel möchte ich einen Überblick über einige dieser Möglichkeiten darstellen.

High-Level-Plotting Funktionen

High-Level-Plotting Funktionen geben die Möglichkeit ganze Schaubilder aus einer Datenmenge zu erstellen. Einige Funktionen erkennen anhand der Art der Daten, welcher Schaubildtyp zu erstellen ist, die meisten Funktionen können die Daten in verschiedenen Formen darstellen. Allen ist jedoch gemeinsam, dass sie das Schaubild mit allen relevanten Metadaten (Beschriftung der Achsen, Titel, Label, etc) befüllen können, je nach dem, wie man das in der Funktion definiert.

R bietet sehr viele Pakete, die sich mit der graphischen Darstellung beschäftigen und dementsprechend groß ist die Anzahl der High-Level-Plotting Funktionen. Eine Suche nach `??plot` liefert dementsprechend viele Ergebnisse. Hier wird man zu jedem in der Literatur erwähnten Schaubildtyp eine Funktion finden, daher werden diese hier nicht aufgelistet.

Low-Level-Plotting Funktionen

Diese Funktionen dienen dazu, weitere Informationen zu einem High-Level Schaubild hinzu zu fügen. Diese Funktionen überschreiben das alte Schaubild nicht, sondern ergänzen es. Es können theoretisch unendlich viele Informationen in Form von Low-Level Funktionen zu einem Schaubild hinzugefügt werden.

`matlines()` ist in der Lage, die Spalten einer Matrix gegen die Spalten einer zweiten Matrix dar zu stellen, kann jedoch auch mit Dataframes benutzt werden. Dabei muss in beiden Elementen die selbe Anzahl an Zeilen vorhanden sein, da sonst ein Konflikt entsteht. Zu aller erst müssen allerdings die Werte geordnet werden. Dazu muss die Prädikatorvariable in einer aufsteigenden Reihenfolge sortiert werden. Diese Funktion ist beispielsweise dann interessant, wenn man Vertrauensintervalle aus einem gegebene Datensatz mit einer auf diesen Versuch bezogenen Variablen vergleichen will.

`abline()` fügt eine gerade Linie zum Schaubild hinzu. Der Funktion müssen also mindestens Der y-Achsenabschnitt und die Steigung mitgeteilt werden. Diese Funktion ist dann interessant, wenn man lineare Zusammenhänge darstellen will.

`lines()` verbindet Koordinaten durch eine Linie und stellt diese dar, sehr interessant ist dies vor allem dann, wenn man mathematische Funktionen zu einem Schaubild hinzufügen will (mehr dazu weiter unten).

`points()` fügt Punkte zum Schaubild hinzu. Welches Symbol als Punkt verwendet werden soll und in welcher Farbe diese Symbole dargestellt werden können wird weiter unten erklärt.

Die Funktionen `polygon()`, `rect()` und `box()` können geometrische Formen zu einem Schaubild hinzufügen. Die Funktion `polygon()` funktioniert ähnlich wie `lines()`, denn sie verbindet ebenfalls Punkte miteinander. `polygon()` braucht jedoch zwei Vektoren (in Form von mathematischen Funktionen), die die Grenzen des Polygons determinieren, während `lines()` nur einen Vektor benötigt und damit eine Linie darstellt. `rect()` stellt eines oder mehrere Rechtecke dar (engl. rectangle). `box()` zeichnet eine Box um das Schaubild.

`mtext()` und `text()` fügen Text zum Schaubild hinzu. Dabei erzeugt `mtext()` einen Text in den Randbereichen außerhalb des eigentlichen Schaubildes (engl. margin) und `text()` einen Text in das Schaubild. Bei beiden Funktionen kann die Position genau bestimmt werden.

`arrows()` fügt einen Pfeil zum Schaubild hinzu, der sich zwischen zwei Koordinatenpunkten befindet.

`legend()` fügt eine Legende zum Schaubild hinzu. Für diese Funktion gibt es sehr viele Argumente, da man eigentlich die komplette Legende selbst definieren kann.

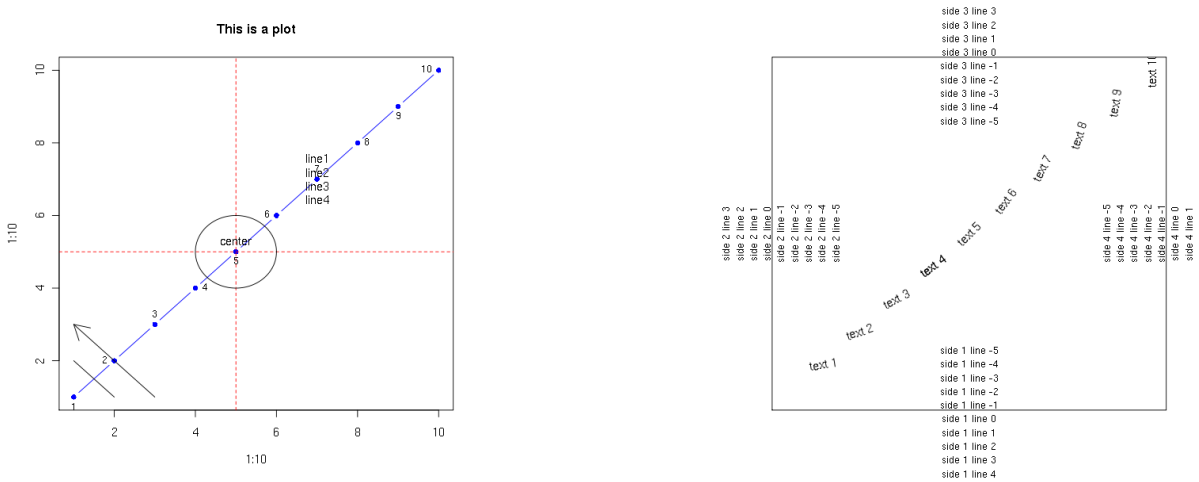
...

Zusammensetzen eines Schaubildes mittels ausschließlich Low-Level-Plotting Funktionen

Ein Schaubild kann komplett aus Low-Level-Plotting Funktionen zusammengesetzt werden. Das macht eigentlich keinen Sinn (da es, wie bereits angesprochen, für alle möglichen Schaubildtypen eine R-Routine gibt), aber um einmal zu zeigen, dass es möglich ist, möchte ich hier das entsprechende Beispiel zeigen (Dieses Beispiel ist komplett aus Dr. Ingo Holz' Vorlesungsunterlagen zur Vorlesung 'R für Ökologen' entnommen).

```
windows()
plot.new()
plot.window(xlim=c(1, 10), ylim=c(1, 10))
box()
axis(side=1)
axis(side=2)
mtext("1:10", side=c(1, 2), line=3)
title(main="This is a plot")
points(1:10, 1:10, col="blue", type="b", pch=16)
abline(h=5, v=5, col="red", lty=2)
text(1:10, 1:10, labels=1:10, cex=.8, pos=c(1:4))
text(5, 5, labels="center", pos=3)
theta <- seq(0, 2*pi, length=360)
x <- cos(theta)+5
```

```
y <- sin(theta)+5
lines(x, y)
text(7, 7, labels="line1\nline2\nline3\nline4")
lines(x=c(2, 1), y=c(1, 2))
arrows(3, 1, 1, 3)
savePlot("Bild016.png", "png")

windows()
plot.new()
plot.window(xlim=c(0, 10), ylim=c(0, 10))
box()
for(i in 1:4){
  for(j in -5:4){
    mtext(paste("side", i, "line", j, sep=" "), side=i, line=j, cex=.8)
  }
}
for(i in 1:10){
  text(i, i, labels=paste("text", i, sep=" "), srt=i*9)
}
savePlot("Bild017.png", "png")
```



(a) Ein Schaubild aus ausschließlich Low-Level-Plotting Funktionen

(b) Beschriftungsmöglichkeiten für Achsen und im Schaubild

Abbildung 5.7: Darstellung der Low-Level-Plotting Funktionen

Darstellen von mathematischen Funktionen

In einigen Fällen will man manuell Funktionen in Schaubilder einfügen um beispielsweise eine Statistik in Zusammenhang mit einer Funktion darstellen zu können. Viele Möglichkeiten sind schon durch R gegeben und haben oft einen Sinn, so wie sie verarbeitet werden. Es schadet meiner Meinung nach aber nicht, wenn man weiß, wie man so etwas manuell durchführen würde.

Mit der Funktion `lines()` ist es möglich, einen Satz an Koordinatenwerten durch eine Linie zu verbinden. Diese Funktion wird sowohl durch die Angabe der x-Werte, als auch der y-Werte gesteuert. Zusätzlich können graphische Parameter (`par()`) angegeben werden, die das Aussehen der darzustellenden Linien definieren.

Die Logik hinter einer mathematischen Funktion in R ist analog zu einer „R-Funktion“. Dabei wird mit der Routine `z <- function{} return{}` operiert. Die Elemente des mathematischen Termes

$$f(x) = a + bx$$

werden dabei wie folgt verabeitet:

$f(x)$: `function{x}`

$a + bx$: `return{a+b*x}`

Es wird also die Funktion von x in Abhängigkeit der in `return{}` definierten Koeffizienten ausgegeben. Die in `function{}` definierten Variablen müssen dabei auch in `return{}` enthalten sein, die in `return{}` enthaltenen Koeffizienten müssen mit einem klaren Zahlenwert, oder einer Funktion, die zu solch einem Wert führt, versehen sein. In der Praxis macht dieser reduzierte Ansatz meist nicht so viel Sinn, da man die Werte der Koeffizienten schon in `return()` einsetzen müsste, da diese sonst fehlen. Ein allgemeinerer Ansatz wäre daher, die Koeffizienten schon als Variablen in `function()` aufzunehmen. So kann man eine allgemeinere Funktion mit vielen verschiedenen Werten für x und auch für die Koeffizienten spezifizieren:

```
function{x, coeff1, coeff2} return{coeff1+coeff2*x}
```

Um dies an einem Beispiel zu verdeutlichen, arbeite ich mit den Vertrauensintervallen für die Vorhersage eines neuen Wertes (Im Biometrie-Skript Seite 22, Beispiel 1). Hier möchte ich das Konfidenzband manuell darstellen. Dazu benötige ich die Funktion der beiden Schaubilder, die dieses Konfidenzband darstellen. Diese ist mir vorteilhafterweise schon im Kochbuchkasten im Biometrieskript gegeben, sodass ich lediglich die Werte von Seite 23 einsetzen muss um die vollständige Funktion zu erhalten.

```
x <- regen
a <- 15.13554
b <- 0.08117
f <- function(x, a, b) return(a+x*b-9.286399*sqrt((1/26)+
 (((x-139.3462)**2)/48335.88)))
g <- function(x, a, b) return(a+x*b+9.286399*sqrt((1/26)+
 (((x-139.3462)**2)/48335.88)))
plot(regen, ernte)
lines(0:250, f(0:250, a, b))
lines(0:250, g(0:250, a, b))
abline(lm.ernte_regen)
savePlot("Bild018.png", "png")
```

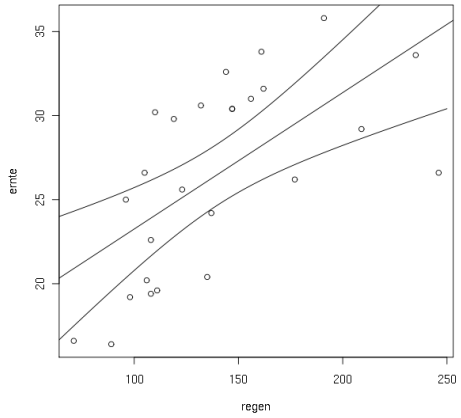


Abbildung 5.8: 95%-Vertrauensintervall (Konfidenzband) für die Vorhersage des Verlaufs der Regressionsgeraden

So kann man an die Variablen `a` und `b` jeden beliebigen Wert binden. Dies spielt dann eine Rolle, wenn man den Wert dieser Variablen einer anderen Funktion, wie beispielsweise `lm()`, entnimmt. Theoretisch kann man die Funktion sogar noch weiter verallgemeinern, indem man auch die Werte für t_{tab}, s, n, $\overline{x}$ und SQ_x der jeweiligen Funktion entnimmt.

Eine weitere Möglichkeit, mathematische Funktionen darzustellen ergibt sich mit der High-Level-Plotting Funktion `curve()`, hierbei wird allerdings das davor dargestellte Schaubild überschrieben (es eignet sich also nur dann, wenn man ein mathematische Funktion darstellen will, zu der man mit Low-Level-Plotting Funktionen weitere Daten hinzufügt.

Hier habe ich das Beispiel aus Kapitel '↑ 6.2.2 t-Tests und Vertrauensintervalle' genommen, welches mit `matlines()` jedoch besser gelöst werden kann.

Interaktive Funktionen

Diese Funktionen erlauben es, während und nach der Erstellung des Schaubildes in dieses eingreifen zu können. Dabei kann man Einzelwerte mit der Maus selektieren um die Werte in Erfahrung zu bringen und/oder weiter zu verarbeiten.

Mit der Funktion `identify()` kann man die Position des Datenpunktes in einem Datensatz ausfindig machen. Dieser wird dann im Schaubild neben den Punkt geschrieben, im Fenster von R wird nach klicken mit der rechten Maustaste ein Vektor aller ausgewählten Punkte angezeigt.

Mit der Funktion `locator()` kann man jeden beliebigen Punkt im Schaubild anklicken und für irgend eine Aktion auswählen. So ist es beispielsweise denkbar eine von Hand gezeichnete Linie (`lines(locator(5))` einzufügen, oder ein Polygon (`polygon(locator(5))` (mit 5 Punkten in beiden Fällen) zu zeichnen.

Die so ermittelten Werte können auch an eine Variable gebunden werden, falls man damit weiterrechnen will (`Punkte <- locator()`).

Visuelle Variablen

Viele Plotting Funktionen, seien sie nun High-Level oder Low-Level, können über eine Vielzahl an Argumenten gesteuert werden. Dabei kann man die Farbe, die Form oder die verwendeten Symbole selbst bestimmen. Die folgenden Unterkapitel liefern einen Überblick über diese Argumente und deren Anwendung und sollen eher als Glossar verstanden werden.

Farben

Es gibt viele Stellen in Funktionen, an denen man eine Farbe spezifizieren kann. Dies geschieht entweder durch das Argument `col =` oder eine andere Funktion, die einen Farbwert als Argument hat (`fg =` , `bg =` , ...).
Eine Palette der möglichen Farbwerte (als Namen) kann man sich mit `colors()` anzeigen lassen und mit `palette()` kann man sich eine Teilmenge davon als vereinfachte und öfter/einfacher zu verwendende Palette definieren.
Mit `gray()`/`rainbow()` ist es möglich eine abgestufte Palette zu erstellen, die entweder in Grautönen oder farbig ist. Dabei ist es möglich, Einfluss auf die Farbwerte zu nehmen, hierzu ist die Dokumentation zu Rate zu ziehen.
Farbwerte können nicht nur als Namen angegeben werden (`red`, `blue`, ...) sondern auch als ↑ Hexadezimal-Wert, wie das folgende Beispiel zeigt:

```
rainbow(10)
 [1] "#FF0000FF" "#FF9900FF" "#CCFF00FF" "#33FF00FF"
 [5] "#00FF66FF" "#00FFFFFF" "#0066FFFF" "#3300FFFF"
 [9] "#CC00FFFF" "#FF0099FF"
```

Dies scheint auf den ersten Blick kompliziert, allerdings funktioniert ein Hexadezimal-Wert genauso wie ein Kombination aus Buchstaben (Wörter) oder Zahlen als Codierung für eine bestimmte Farbe. Die Anordnung/Kombination der Buchstaben und Zahlen in Hexadezimal-Code erlauben allerdings eine systematische Darstellung sehr vieler verschiedener Farben, was mit den normalen Codes (Wörtern) für diese Farben nicht erreicht werden kann, da es dort eben kein derart standardisiertes Vorgehen gibt.

Symbole und Linientypen

Alle Schriftzeichen, Symbole und Linien in Schaubildern können beeinflusst werden.

Mit dem Argument `pch =` kann man definieren, welches Symbol ein Punkt haben soll. Dies kann durch ganze Zahlen geschehen. Verwendet werden können die ganzen Zahlen von `0` bis `25` und von `32` bis `127`. Diese Zahlen stehen für Symbole und
↑ ASCII-Schriftzeichen, respektive. Im Fall der ASCII-Schriftzeichen kann man auch die Buchstaben selbst verwenden, anstelle der Zahlen, die für die entsprechenden Buchstaben codieren.

Mit dem Argument `type =` kann man definieren, wie die Punkte eines Schaubildes verbunden werden sollen. Dabei gilt: `"p"` = Punkte; `"l"` = Linien; `"b"` = Punkte,

verbunden durch Linien; "o" = Punkte, verbunden durch Linien die über die Punkte gehen; "h" = vertikale Linien, die von der x-Achse bis zum y-Wert reichen; "s" = Stufen, wobei die Werte durch die obere Kante der vertikalen Linien repräsentiert werden; "S" = Stufen, wobei die Werte durch die untere Kante der vertikalen Linien repräsentiert werden.

Mit dem Argument lty = kann man die Art der Linien definieren. Dabei gilt: 1 = solide; 2 = gestrichelt; 3 = gepunktet; 4 = abwechselnd gestrichelt/gepunktet; 5 = lange Striche; 6 = abwechselnd lange/kurze Striche.

Mit dem Argument lwd = kann man die Breite einer Linie relativ zur Standardbreite (1) definieren. lwd=5 würde also eine Linie 5 mal so breit wie die Standardlinie ausgeben.

Mit dem Argument font = kann man die Schriftart definieren. Dabei gilt: 1 = normal; 2 = **fett**; 3 = *kursiv*; 4 = ***kursiv und fett***

Mit dem Argument cex = kann man die Schrift- und Punktgröße relativ zur Standardgröße (1) definieren. cex=.75 würde also eine Größe 3/4 so groß wie die Standardgröße ausgeben.

Die Funktion par()

Die Funktion par() dient dazu Informationen, über graphische Parameter auszulesen oder zu verändern. Eine Übersicht über alle Möglichkeiten, die sich mit dieser Funktion bieten, liefern diverse Referenzkarten für R. Die Referenzkarte für R-Grafiken nach ↑ Andreas Plank (PDF, 465 KB) liefert beispielsweise eine genaue Auflistung aller für die Funktion par() verwendbaren Argumente. Viele dieser Argumente finden auch in anderen (High-Level-Plotting) Funktionen Verwendung, es existieren allerdings auch einige Argumente, die ausschließlich in par() Verwendung finden. Einige Parameter sind „read-only“ (_*R.O.*_), können also lediglich betrachtet und weiterverarbeitet, allerdings nicht modifiziert werden. par() stellt eine Art Kontrollzentrale für alle möglichen Parameter dar, die man in und um ein Schaubild verändern kann.

Generell

Mit dem Argument adj = kann man definieren, wie Text ausgerichtet sein soll. Dies ist auch in anderen Funktionen modifizierbar. Ein Wert von 0 erzeugt einen nach links ausgerichteten Text, ein Wert von 1 erzeugt einen nach rechts ausgerichteten Text. Dazwischen kann jeder beliebige Wert verwendet werden. 0.5 (Standard) bedeutet, dass der Text zentriert ist.

Mit den Argumenten srt = und crt = kann man definieren in welchem Winkel Wörter und Buchstaben respektive rotiert werden sollen.

Mit dem logischen Argument new = kann man definieren, ob die nächste High-Level-Plotting Funktion das alte Schaubild überschreiben soll oder nicht (Standard = FALSE)

Die Achsen betreffend

Mit dem Argument `bty =` kann man die Box um ein Schaubild definieren. Dabei gilt: `"o"` = (Standard) eine geschlossene Box um das gesamte Schaubild; `"l"` = nach oben und rechts offen; `"7"` = nach links und unten „offen“; `"c"` = nach rechts offen; `"u"` = nach oben offen; `"]"` = nach links „offen“; `"n"` = keine Box wird gezeichnet.

Mit dem Argument `las =` kann man die Ausrichtung der Achsenbeschriftung definieren. Dabei gilt: `0` = beide parallel zur Achse; `1` = beide horizontal; `2` = beide senkrecht zur Achse; `3` = beide senkrecht.

Mit den Argumenten `xaxp = c(x1, x2, n)` und `yaxp = c(y1, y2, n)` kann man definieren, an welcher Stelle die Begrenzungsmarken liegen und welches Intervall dazwischen definiert ist. Hierzu sieht man sich am besten auch die Dokumentation zur Funktion `axTicks()` an, da die Methode mit `xaxp/yaxp` noch nicht vollständig von S in R implementiert ist.

Mit den Argumenten `xaxs =` und `yaxs =` kann man definieren, wie die Intervalle der Achsen berechnet werden sollen.

Mit den Argumenten `xaxt =` und `yaxt =` kann man den Typ der Achsen definieren. Da dieses Argument ebenfalls noch nicht vollständig implementiert ist, sind nur die Werte `"n"` und `"s"/"l"/"t"` gültig. Dabei steht `"n"` für „keine Achse“ (not) und `s` für Standard.

Mit den logischen Argumenten `xlog =` und `ylog =` kann man definieren, ob eine logarithmische Skala verwendet werden soll (Standard = `FALSE`).

Um das Erscheinungsbild der Achsen weiter zu verändern, gibt es Argumente wie `col.axis =` , `cex.axis =` und `font.axis =` , die nach dem schon beschrieben Schema der Funktionen `col`, `cex` und `font` funktionieren. Diese Form der Argumente gibt es übrigens auch für die Argumente `lab -> col.lab/cex.lab/font.lab`, `main -> col.main/cex.main/font.main` und `sub -> col.sub/cex.sub/font.sub`.

Das Layout betreffend

Das Argument `din = ; _*R.O.*_` zeigt die Dimensionen des Schaubildes.

Mit den Argumenten `mfrow = c(nr, nc)` und `mfcol = c(nr, nc)` kann man definieren wie viele Schaubilder in einem Device (ein extra Fenster, in das ein Schaubild geplottet wird) dargestellt werden sollen und in welcher Reihenfolge diese dargestellt werden soll. `nr` steht für die Anzahl der Reihen und `nc` für die Anzahl der Spalten. Bei `mfrow` werden die Schaubilder den Reihen nach angeordnet, bei `mfcol` den Spalten nach.

Mit dem Argument `mfg = c(i, j)` kann man definieren welches Schaubild als nächstes geplottet wird. Dabei muss durch `mfrow/mfcol =` diese Position verfügbar sein.

Mit den Argumenten `mar = (bottom, left, top, right)` und `mai = (bottom, left, top, right)` kann man definieren wie Breit die Außenbereiche um ein einzelnes Schaubildes (margin) sein sollen. `mar` muss in „Zeilen Text

(row)“ angegeben sein und `mai` in inches.

Mit den Argumenten `oma = (bottom, left, top, right)` und `omi = (bottom, left, top, right)` kann man definieren wie Breit der Außenbereich um alle Schaubilder gemeinsam sein soll. Hat man mit `mfrow` definiert, dass mehrere Schaubilder angezeigt werden sollen, wird mit `oma/omi` die Breite um alle Schaubilder definiert. Existiert nur ein Schaubild, kommt dieses Argument `mar/mai` gleich.

Mit dem Argument `pty = (plotting type)` kann man definieren, wie die Plotting-Region aussehen soll. Dabei gilt: `"s"` = quadratische (square) Plotting-Region; `"m"` = maximale Plotting-Region.

6 Korrelation und Regression

6.1 Die Pearson'sche Produkt-Moment-Korrelation

Seite 1, Beispiel 3:

```
regen <- c(177, 96, 144, 105, 111, 135, 209, 161, 246, 108, 137, 71, 119,
 108, 132, 89, 147, 98, 106, 123, 156, 191, 162, 235, 147, 110)
ernte <- c(26.2, 25, 32.6, 26.6, 19.6, 20.4, 29.2, 33.8, 26.6, 22.6, 24.2,
 16.6, 29.8, 19.4, 30.6, 16.4, 30.4, 19.2, 20.2, 25.6, 31, 35.8, 31.6,
 33.6, 30.4, 30.2)
```

In R muss man nicht auf die Schätzung der Korrelation aus Herrn Piephos Skript zurückgreifen, sondern kann sich die Funktionen `cor()` und `cor.test()` zunutze machen.

```
cor(regen, ernte)
 [1] 0.6291297
```

```
cor.test(regen, ernte)

         Pearson's product-moment correlation

 data:  regen and ernte
 t = 3.9651, df = 24, p-value = 0.0005754
 alternative hypothesis: true correlation is not equal to 0
 95 percent confidence interval:
  0.3196827 0.8173081
 sample estimates:
       cor
 0.6291297
```

Mit `cor.test()` kann man sich nicht nur den Korrelationskoeffizienten anzeigen lassen, sondern auch die Signifikanz, die in Form eines p-Wertes ausgegeben wird, und das (1-α)100%-ige Vertrauensintervall für den Korrelationskoeffizienten.

6.2 Regression

Um die Regression in R zu bearbeiten, ist es wichtig, davor zu wissen, dass man ein lineares Modell (↑Regressionsmodell) an die Daten anpassen muss. R macht dies bereits automatisch mit der Funktion `lm()`.

```
df.ernte_regen <- data.frame(ernte, regen)
lm.ernte_regen <- lm(ernte ~ regen)

-------------------------------------------------------------------------

lm.ernte_regen

 Call:
 lm(formula = ernte ~ regen)

  Coefficients:
 (Intercept)        regen
    15.13554      0.08117

-------------------------------------------------------------------------

png("Bild019.png")
plot(regen, ernte)
abline(lm.ernte_regen)
dev.off()
 null device
           1

-------------------------------------------------------------------------

plot(regen, ernte)
abline(lm.ernte_regen)
savePlot("Bild019", "png")
```

Hier sieht man die Anwendung der Funktion `png()`. In dieser Form wird in das Arbeitsverzeichnis eine Datei mit dem Namen „Bild001.png“ geschrieben, die die zwischen `png()` und `dev.off()` angegebenen Daten enthält. Das zweite Beispiel verdeutlicht die etwas eingängigere und nicht so schreib-aufwändige Methode. Dabei kann man das `png()` am Anfang weglassen.

Die Daten `Regen` sind hier die Prädikator- oder Einflussvariable und die Daten `Ernte` sind die Zielvariable.

Um auf einzelne Komponenten oder deren Attribute zurückgreifen zu können, ist die Funktion `attributes()` wichtig.

```
attributes(lm.ernte_regen)
 $names
  [1] "coefficients"  "residuals"     "effects"
  [4] "rank"          "fitted.values" "assign"
  [7] "qr"            "df.residual"   "xlevels"
 [10] "call"          "terms"         "model"

 $class
 [1] "lm"
```

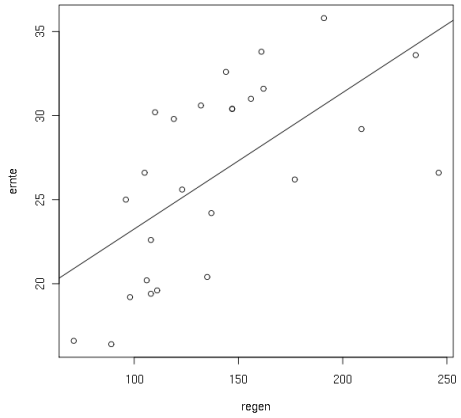


Abbildung 6.1: Lineare Regression für die Regendaten

```
lm.ernte_regen$coef
 (Intercept)       regen
 15.13553937  0.08116919
```

Mit `attributes()` kann man sich in diesem Fall verschiedene Attribute des Models anzeigen lassen. In diesem Beispiel ist die Komponente `"coefficients"(coef)` wichtig. Nun kann man mit der `$`-Notation auf die Elemente der Liste/des Dataframes zugreifen. `(Intercept)` stellt den y-Achsenabschnitt dar und `regen` die Steigung. Man kann diese beiden Variablen mit `lm.ernte_regen$coef[1]` oder `lm.ernte_regen$coef[2]` anwählen und in eine Formel der Form $y = a + bx$ einsetzen.

Seite 14, Beispiel 1:

```
x <- 200
y <- lm.ernte_regen$coef[1]+lm.ernte_regen$coef[2]*x

------------------------------------------------------------------------

y
 [1] 31.36938
```

6.2.1 Streuungszerlegung

Das Bestimmtheitsmaß lässt sich in R einfach mit der Funktion `cor()` berechnen, indem dieser quadriert wird. Damit wird also der Korrelationskoeffizient quadriert.

```
r2 <- cor(regen, ernte)**2
```

```
------------------------------------------------------------------------

r2
 [1] 0.3958042
```

Seite 19, Beispiel 1:

```
anova(lm.ernte_regen)
 Analysis of Variance Table

 Response: ernte
           Df Sum Sq Mean Sq F value    Pr(>F)
 regen      1 318.46  318.46  15.722 0.0005754 ***
 Residuals 24 486.13   20.26
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Mit der schon bekannten Funktion `anova()` erstellt man die Varianzanalyse-Tablle für ein lineares Modell. Hier sind die Werte 'Freiheitsgrade' (`DF`), 'Summe der Quadrate' (`Sum Sq`), 'Mittelquadrate' (`Mean Sq`) und der F-Wert (`F value`) angegeben. Der p-Wert (`Pr(>F)`) zeigt die Signifikanz an, bei diesem Wert handelt es sich um den selben Wert, den man auch mit `cor.test()` ermitteln kann. Bei einem definierten Signifikanzniveau von $\alpha = 0.05$ liegt hier Signifikanz vor, da der p-Wert mit 0.0005 um zwei Zehnerpotenzen unter 0.05 liegt.

6.2.2 t-Tests und Vertrauensintervalle

Will man die für 'R für Vl Statistik' entwickelten Funktion zur Berechnung der verschiedenen im Skript angesprochenen Vertrauensintervalle verwenden, sollte man die Funktion `ci.reg()` ein kleines bisschen modifizieren, sodass man damit sowohl den Wert des Vertrauensintervalls auf der Regressionsgeraden, als auch für einen neuen Wert aus den Daten errechnen kann.

(↑Formelübersicht)

`ci.reg()`: berechnet das Vertrauensintervall für die Vorhersage eines neuen Einzelwertes an der Stelle x_0.

`x =` : Zielvariablen.

`y =` : Variable die x beschreiben soll (Prädikatorvariable).

`x_0 =` : neuer Einzelwert.

`rl = FALSE/TRUE`: gibt an, ob man das Vertrauensintervall auf der Regressionsgeraden (regression line) berechnen will; default=FALSE.

`alpha =` : Irrtumswahrscheinlichkeit zu Berechnung der Quantile der t-Verteilung; default = 0.05.

Nachdem man die Zeilen der neuen Funktion (befindet sich im Anhang; klicken auf Formelübersicht!) in das Dokument `Input.R` eingetragen und damit die alte Funktion `ci.reg()` ersetzt hat, kann man (nach Einbinden mit `source("Formeln.R")`) die Vertrauensintervalle berechnen.

Seite 22, Beispiel 1

```
ci.reg(ernte, regen, 200, rl=TRUE)
 [1] 28.22527 34.51349
ci.reg(ernte, regen, 200)
 [1] 21.56293 41.17582
```

Um die Konfidenzbänder berechnen zu können, muss man die Funktion `predict()` anwenden. Mit `intervall = "pred"/"conf"` kann man festelegen, ob man einen Einzelwert betrachten will, oder die Regressionsgerade respektive.

```
ernte <- ernte[order(regen)]
regen <- regen[order(regen)]
df.ernte_regen <- data.frame(ernte, regen)
pp <- predict(lm.ernte_regen, df.ernte_regen, interval="pred")
pc <- predict(lm.ernte_regen, df.ernte_regen, interval="conf")
```

```
head(df.ernte_regen)
   ernte regen
 1  16.6    71
 2  16.4    89
 3  25.0    96
 4  19.2    98
 5  26.6   105
 6  20.2   106
```

```
head(pp)
        fit      lwr      upr
 1 20.89855 11.00221 30.79490
 2 22.35960 12.65784 32.06135
 3 22.92778 13.28655 32.56901
 4 23.09012 13.46458 32.71566
 5 23.65830 14.08202 33.23459
 6 23.73947 14.16950 33.30944
```

```
plot(regen, ernte)
matlines(df.ernte_regen$regen, pc, lty=1, col="black")
matlines(df.ernte_regen$regen, pp, lty=2, col="black")
legend( "bottomright", lty=1:2, legend=c("Regressionsgerade",
 "Einzelwert"))
savePlot("Bild020.png", "png")
```

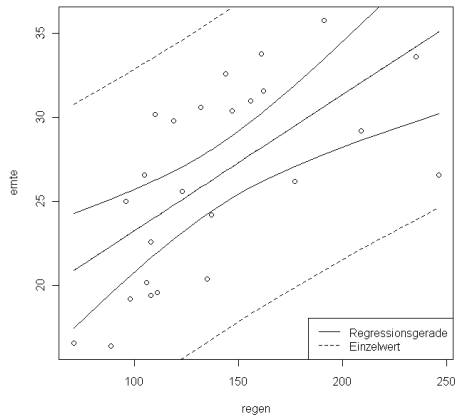


Abbildung 6.2: 95%-Vertrauensintervall (Konfidenzband) für die Vorhersage eines neuen Wertes und den Verlauf der Regressionsgeraden

Das Vertrauensintervall der Steigung lässt sich mit `confint()` berechnen.

```
confint(lm.ernte_regen)
                  2.5 %     97.5 %
 (Intercept) 8.97282667 21.2982521
 Regen       0.03891959  0.1234188
```

Hierbei wird von R auch gleich das Vertrauensintervall für den y-Achsen-Abschnitt der Regressionsgeraden (`Intercept`) mit berechnet, also die Regressionslinie an der Stelle x_0.

6.2.3 Inverse Regression

Für die inverse Regression, also zur Berechnung eines x_0-Wertes und dessen Vertrauensintervall anhand eines gegebenen y_0-Wertes, existiert bisher noch keine Funktion.

(↑Formelübersicht)

`ci.inv.reg()`: berechnet das Vertrauensintervall für die inverse Regression.

`x =` : Einflussvariable.

`y =` : Zielvariable.

`y_0 =` : gegebener y_0-Wert.

`sx = FALSE/TRUE`: Fragt, ob der geschätzte Wert für x_0 angezeigt werden soll oder nicht (show x-value); default = FALSE

`alpha =` : Irrtumswahrscheinlichkeit zu Berechnung der Quantile der t-Verteilung; default = 0.05.

Seite 26, Beispiel 1:

```
leu <- c(.02, .05, .10, .30, .40, .50, .60)
ext <- c(.08, .15, .29, .88, 1.13, 1.42, 1.69)

-------------------------------------------------------------------------

ci.inv.reg(leu, ext, 0.35, sx=TRUE)
 [1] 0.1057516 0.1185059 0.1311259
```

Dieses Intervall besteht aus drei Zahlen, die erste Zahl zeigt den unteren Wert des Vertrauensintervalls, die letzte zeigt den oberen Wert des Vertrauensintervalls und die Zahl in der Mitte zeigt den x-Wert.

```
ci.inv.reg(leu, ext, 0.35)
 [1] 0.1057516 0.1311259
```

6.3 Nichtlineare Regression durch Transformation der Variablen (6.4)

In diesem Kapitel werden im Skript einige Transformationen vorgestellt. Diese in R umzusetzen ist nicht sehr schwierig, da keine neuen Funktionen hinzukommen. Vom Grundprinzip her muss man die gegebene Daten lediglich so transformieren, dass ein lineares Modell an die Daten angepasst werden kann und wie in Kapitel 6.2 damit verfahren werden kann.

Seite 32, Beispiel 1:

```
lai <- c(0.5, 0.6, 1.8, 2.5, 2.8, 5.45, 5.6, 7.2, 8.75, 9.6, 10.4, 12)
ltr <- c(75, 72, 42, 29, 27, 10, 9, 5, 2, 2, 1, 0.9)

-------------------------------------------------------------------------

plot(lai, ltr)
```

```
savePlot("Bild021.png", "png")
```

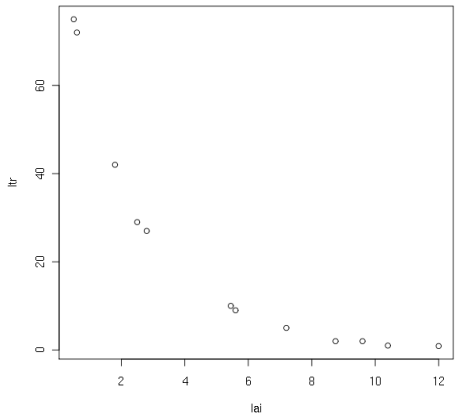


Abbildung 6.3: Verteilung der Datenpunkte

Nach Betrachten der Verteilung der Datenpunkte vermutet man eine Exponentialfunktion. Damit weiß man, dass man die Daten transformieren muss, indem man das Modell logarithmiert, genauer gesagt die y-Werte.

```
log.ltr <- log(ltr)
fit <- lm(log.ltr~lai)
```

```
log.ltr
 [1]  4.3174881  4.2766661  3.7376696  3.3672958  3.2958369
 [6]  2.3025851  2.1972246  1.6094379  0.6931472  0.6931472
[11]  0.0000000 -0.1053605
```

```
fit

 Call:
 lm(formula = log.ltr ~ lai)

 Coefficients:
 (Intercept)          lai
      4.4579      -0.4034
```

```
plot(lai, log.ltr)
abline(fit)
savePlot("Bild022.png", "png")
```

Damit sind der y-Achsenabschnitt (`(Intercept)`) und die Steigung (`lai`) des Modells bekannt. Durch Rücktransformieren des y-Achsenabschnittes des linearen Modells lässt sich der y-Achsenabschnitt des nicht-linearen Modells ermitteln und dessen Schaubild zeichnen.

```
a <- exp(fit$coef[1])
f <- function(x) return(a*exp(x*fit$coef[2]))
```

```
a
 [1] 86.3054
```

```
plot(lai, ltr)
lines(0:13, f(0:13))
savePlot("Bild023.png", "png")
```

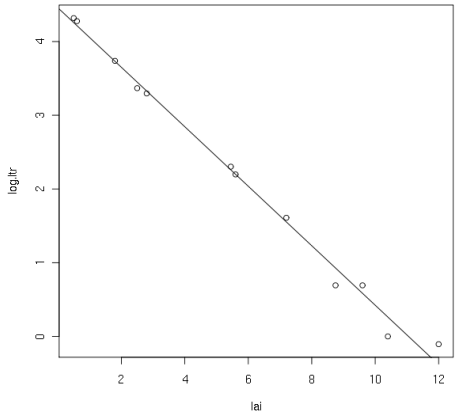


Abbildung 6.4: Verteilung der Datenpunkte nach logarithmieren der y-Werte, mit Regressionsgerade

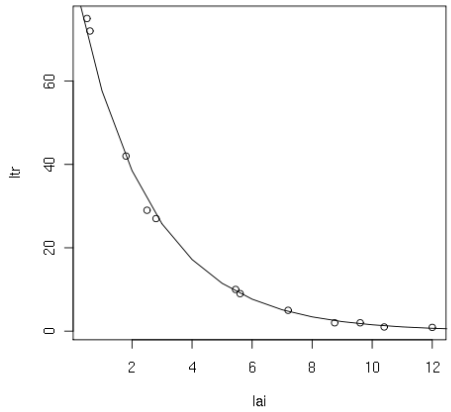
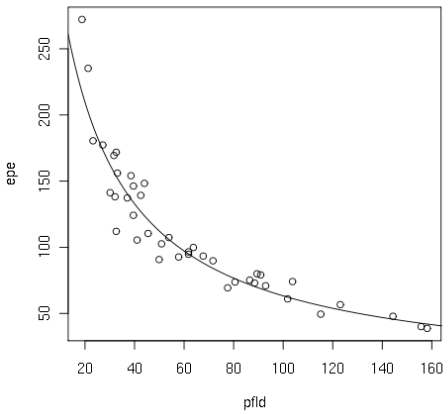


Abbildung 6.5: Schaubild der Originaldaten

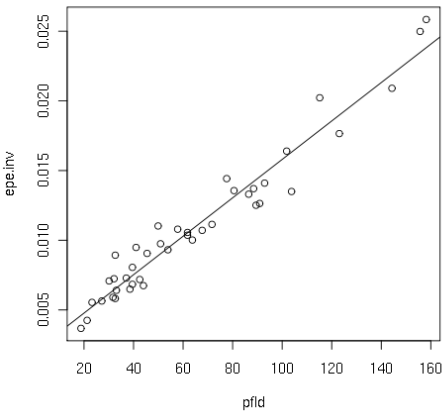
Seite 34, Beispiel 1:

```
epe <- c(272.15, 235.23, 180.47, 177.31, 141.28, 169.39, 138.17, 171.81,
 112.02, 156.09, 137.29, 154.10, 124.17, 146.28, 105.47, 139.24, 148.31,
 110.44, 90.72, 102.62, 107.36, 92.66, 96.52, 94.71, 99.86, 93.37, 89.78,
 69.34, 73.74, 75.17, 72.98, 79.94, 79.13, 70.93, 60.99, 74.09, 49.45,
 56.65, 47.84, 40.03, 38.70)
pfld <- c(18.78, 21.25, 23.23, 27.18, 30.15, 31.67, 32.12, 32.62, 32.62,
 33.07, 37.07, 38.55, 39.54, 39.54, 41.02, 42.50, 43.98, 45.47, 49.92,
 50.90, 53.87, 57.82, 61.78, 61.78, 63.75, 67.71, 71.66, 77.59, 80.56,
 86.49, 88.46, 89.45, 90.93, 92.91, 101.81, 103.78, 115.15, 123.06,
 144.31, 155.68, 158.15)
```

```
a <- fit$coef[1]
b <- fit$coef[2]
f <- function(x) return(1/(a+b*x))
epe.inv <- 1/epe
fit <- lm(epe.inv~pfld)

------------------------------------------------------------------------

fit

 Call:
 lm(formula = epe.inv ~ pfld)

 Coefficients:
 (Intercept)         pfld
   0.0020134    0.0001379

------------------------------------------------------------------------

f(0)
 [1] 496.6804

------------------------------------------------------------------------

1/ci.reg(epe.inv, pfld, 0, rg=TRUE)
 [1] 816.6084 356.8680

------------------------------------------------------------------------

plot(pfld, epe)
lines(0:200, f(0:200))
savePlot("Bild024.png", "png") # Schaubild (a)
plot(pfld, epe.inv)
abline(fit)
savePlot("Bild025.png", "png") # Schaubild (b)
```



(a) Schaubild von y gegen x



(b) Schaubild von 1/y gegen x

Abbildung 6.6: Einzelpflanzenertrag (y) gegen Pflanzdichte (x)

Seite 37, Beispiel 1:

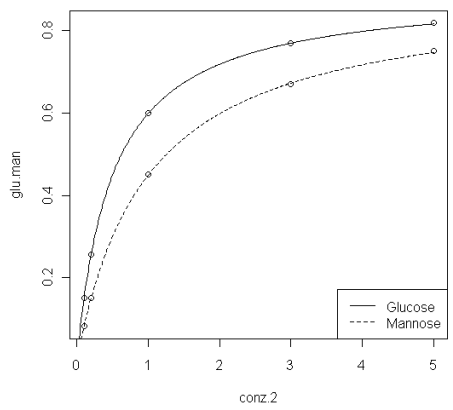
```
conz <- c(.1, .2, 1, 3, 5)
glu <- c(.15, .256, .6, .77, .818)
man <- c(.082, .15, .45, .67 , .75)
```

```
conz.inv <- 1/conz
glu.inv <- 1/glu
man.inv <- 1/man
fit1 <- lm(glu.inv~conz.inv)
fit2 <- lm(man.inv~conz.inv)
a1 <- fit1$coef[1]
b1 <- fit1$coef[2]
a2 <- fit2$coef[1]
b2 <- fit2$coef[2]
Vmax1 <- 1/a1
Km1 <- b1*Vmax1
Vmax2 <- 1/a2
Km2 <- b2*Vmax2
f1 <- function(x) return((x*Vmax1)/(Km1+x))
f2 <- function(x) return((x*Vmax2)/(Km2+x))
glu.man <- c(glu, man)
conz.2 <- c(conz, conz)
```

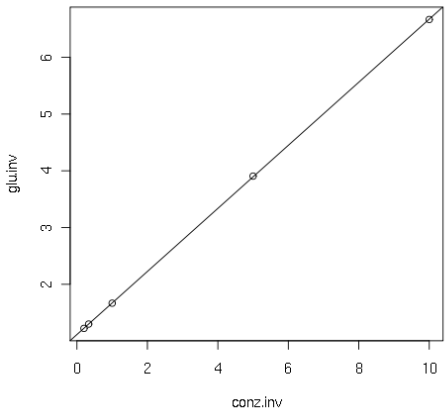
```
fit1

 Call:
```

```
 lm(formula = glu.inv ~ conz.inv)

 Coefficients:
 (Intercept)     conz.inv
      1.1141       0.5559

--------------------------------------------------------------------------------

fit2

 Call:
 lm(formula = man.inv ~ conz.inv)

 Coefficients:
 (Intercept)     conz.inv
       1.118        1.108

--------------------------------------------------------------------------------

Vmax1
 [1] 0.8975625

--------------------------------------------------------------------------------

Km1
 [1] 0.4989146

--------------------------------------------------------------------------------

Vmax2
 [1] 0.8946033

--------------------------------------------------------------------------------

Km2
 [1] 0.991322

--------------------------------------------------------------------------------

plot(conz.2, glu.man)
lines(seq(0, 5, .01), f1(seq(0, 5, .01)), lty=1)
lines(seq(0, 5, .01), f2(seq(0, 5, .01)), lty=2)
legend( "bottomright", lty=1:2, legend=c("Glucose", "Mannose") )
savePlot("Bild026.png", "png")
plot(conz.inv, glu.inv)
abline(fit1)
savePlot("Bild027.png", "png")
```

Zur Berechnung des Vertrauensintervalls der K_m-Werte benötigen wir die Funktion des linearen Zusammenhanges. Damit können wir über die inverse Regression und eine Rücktransformation die Vertrauensintervalle anzeigen lassen.



(a) Schaubild von x gegen y für Glucose und Mannose



(b) Schaubild von 1/y gegen 1/x für Glucose

Abbildung 6.7: Reaktionsgeschwindigkeit (y) gegen Substratkonzentration (x)

```
fg <- function(x) return(a1+b1*1/x)
fm <- function(x) return(a2+b2*1/x)

------------------------------------------------------------------------

1/ci.inv.reg(conz.inv, glu.inv, fg(Km1), sx=TRUE)
 [1] 0.5128202 0.4857639 0.4989146

------------------------------------------------------------------------

1/ci.inv.reg(conz.inv, man.inv, fm(Km2), sx=TRUE)
 [1] 1.0153805 0.9684043 0.9913220
```

Aufgrund der Berechnung der Inversen ist das Intervall hier „falschrum“ beziehungsweise invers angegeben. Der erste Wert ist der inverse Wert der Untergrenze des Vertrauensintervalls, der letzte Wert ist der inverse Wert der Obergrenze des Vertrauensintervalls und der mittlere Wert ist der ermittelte x-Wert.

Seite 40, Beispiel 1:

```
dose <- c(0.375, 0.75, 1.5, 3, 6, 12, 24)
ld <- c(0, 1, 8, 11, 16, 18, 20)
m <- 20
mort <- ld/m


log.x <- log(dose)
```

```
elogit.y <- function(x) return(log((x+1/(2*m))/(1-x+1/(2*m))))
fit <- lm(elogit.y(mort)~log.x)
a <- fit$coef[1]
b <- fit$coef[2]
y <- function(x) return(exp(a+b*x)/(1+exp(a+b*x)))
```

```
log.x
 [1] -0.9808293 -0.2876821  0.4054651  1.0986123  1.7917595
 [6]  2.4849066  3.1780538
```

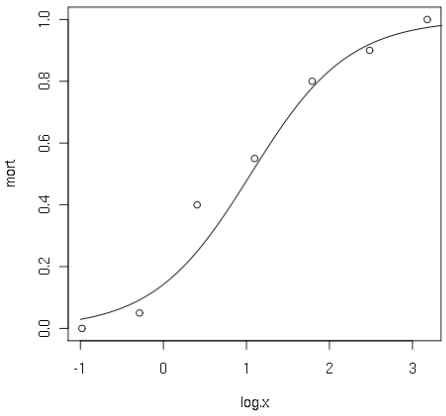
```
elogit.y(mort)
 [1] -3.7135721 -2.5649494 -0.3856625  0.1910552  1.2992830
 [6]  2.0014800  3.7135721
```

```
fit

 Call:
 lm(formula = elogit.y(mort) ~ log.x)

 Coefficients:
 (Intercept)        log.x
      -1.796        1.705
```

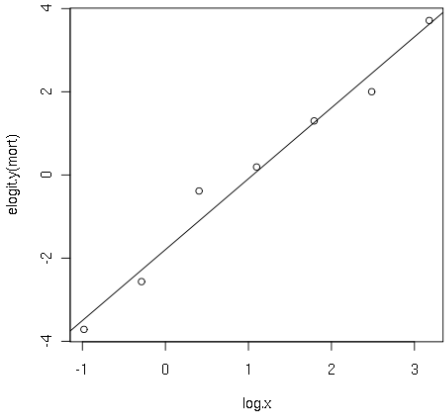
```
plot(log.x, mort)
lines(seq(-1, 4, .01), y(seq(-1, 4, .01)))
savePlot("Bild028.png", "png")
plot(log.x, elogit.y(mort))
abline(fit)
savePlot("Bild029.png", "png")
```

```
elogit.y(.5)
 [1] 0
```

```
exp(ci.inv.reg(log.x, elogit.y(mort), 0, sx=TRUE))
 [1] 1.446855 2.867033 5.665618
```



(a) y gegen log(x)



(b) elogit(y) gegen log(x)

Abbildung 6.8: gestorbene Insektenlarven (y) gegen logarithmierte Dosis (x)

6.4 Korrelation bei nichtlinearen Zusammenhängen (6.5)

Wie aus dem Skript zur Vl. Statistik schon bekannt ist, kann man mit der Funktion `sort()` einen beliebigen Vektor zum Beispiel aufsteigend sortieren lassen. Hier braucht man allerdings nicht nur eine Sortierung der Daten, sondern auch eine Auflistung der Ränge. Hierfür kann man sich die Funktion `rank()` zu Nutze machen. Man kann verschiedene Methoden definieren, die Rangbindungen zu behandeln; für die Ränge, die für dieses Beispiel von Nöten sind reicht aber die Standardeinstellung.

Seite 46, Beispiel 1:

```
age <- c(15, 15, 15, 18, 28, 29, 37, 37, 44, 50, 50, 60, 61, 64, 65, 65,
 72, 75, 75, 82, 85, 91, 91, 97, 98, 125, 142, 142, 147, 147, 150, 159,
 165, 183, 192, 195, 218, 218, 219, 224, 225, 227, 262, 262, 267, 246,
 258, 276, 185, 300, 301, 305, 312, 317, 338, 347, 354, 357, 375, 394,
 516, 535, 554, 591, 648, 660, 705, 723, 756, 768, 860)
lens <- c(21.66, 22.75, 22.30, 31.25, 44.79, 44.55, 50.25, 46.88, 52.03,
 63.47, 61.13, 81.00, 73.09, 79.09, 79.51, 65.31, 71.90, 86.10, 94.10,
 92.50, 105.00, 101.70, 102.90, 110.00, 104.30, 134.90, 130.68, 140.58,
 155.30, 152.20, 144.50, 142.15, 139.81, 153.22, 145.72, 161.10, 174.18,
 173.30, 173.54, 178.86, 177.68, 173.73, 159.98, 161.29, 187.07, 176.13,
 183.40, 186.26, 189.66, 186.09, 186.70, 186.80, 195.10, 216.41, 203.23,
 188.38, 189.70, 195.31, 202.63, 224.82, 203.30, 209.70, 233.90, 234.70,
 244.30, 231.00, 242.40, 230.77, 242.57, 232.12, 246.70)
```

```
r.x <- rank(age)
r.y <- rank(lens)
```

Aus Platzgründen lasse ich es an dieser Stelle aus, die Tabelle wie im Skript darzustellen. Diese kann man mit `df <- data.frame(age, lens, r.x, r.y)` einfach selbst ansehen.

Die Spearman'sche Rangkorrelation ist in der Funktion `cor()` bereits enthalten. Dazu muss man die Methode passend auswählen, was mit `method = "spearman"` geschieht. Genauso bei `cor.test()`.

```
cor(age, lens, method="spearman")
 [1] 0.984061
```

```
cor(age, lens)
 [1] 0.8696296
```

```
cor.test(age, lens, method="spearman")

         Spearman's rank correlation rho

 data:  age and lens
 S = 950.6033, p-value < 2.2e-16
 alternative hypothesis: true rho is not equal to 0
 sample estimates:
      rho
 0.984061

 Warning message:
 In cor.test.default(age, lens, method = "spearman") :
   Cannot compute exact p-values with ties
```

Trotz der Warnung, dass diese Routine den exakten p-Wert bei „ties", also Rangbindung, nicht berechnen kann (da der p-Wert bei großen Werten von ρ nur approximativ gültig ist), ist doch ersichtlich, dass ρ mit 0.984061 sehr groß und der p-Wert mit `p-value < 2.2e-16` sehr klein ist. Daher ist dieser Zusammenhang höchst signifikant. Es wird die Alternativhypothese angenommen, nach der ρ nicht gleich 0 ist, also eine Korrelation existiert.

Seite 49, Beispiel 1:

```
p1 <- c(4, 1, 6, 5, 3, 2, 7)
p2 <- c(4, 2, 5, 6, 1, 3, 7)
```

```
cor.test(p1, p2, method="spearman")

         Spearman's rank correlation rho

 data:  p1 and p2
 S = 8, p-value = 0.02381
 alternative hypothesis: true rho is not equal to 0
 sample estimates:
       rho
 0.8571429
```

Bei der Funktion `cor.test()` kann man mit dem Argument `exact = TRUE/FALSE` determinieren, ob der exakte p-Wert berechnet werden soll oder nicht. Bei Stichproben mit n $<$ 10 wird immer der exakte p-Wert berechnet.

6.5 Test auf Linearität (6.6)

Um zu verstehen, wie das lineare Modell hier zu determinieren ist (also die Funktion `lm()` aufzurufen ist), möchte ich an dieser Stelle auf die Funktion `factor()` eingehen. Diese Funktion wird verwendet, um eine quantitative Variable in einen qualitative Variable umzuwandeln.

```
n
  [1]   0   0   0  35  35  35  70  70  70 105 105 105 140 140
 [15] 140

------------------------------------------------------------------------

factor(n)
  [1] 0   0   0   35  35  35  70  70  70  105 105 105 140 140
 [15] 140
 Levels: 0 35 70 105 140
```

In diesem Fall werden also die gemeinsamen Stufen des Datenvektors ermittelt (eine genaue Erklärung dazu in Kapitel '↑6.7 Lineare Modelle in Matrizenschreibweise (6.8)'). Alternativ kann man die Funktion `gl()` verwenden, die einen Vektor erzeugt, der faktorisiert ist und die Stufen enthält, die gewünscht sind. Dabei ist es egal welchen Wert die Stufen haben, da die Werte lediglich zur Abgrenzung der Stufen voneinander dienen.

```
n <- gl(5, 3, labels=1:5)

------------------------------------------------------------------------

n
  [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
 Levels: 1 2 3 4 5
```

```
-------------------------------------------------------------------------

n <- gl(5, 3, labels=c(0, 35, 70, 105, 140))

-------------------------------------------------------------------------

n
  [1] 0   0   0   35  35  35  70  70  70  105 105 105 140 140 140
 Levels: 0 35 70 105 140
```

Seite 50, Beispiel 1:

```
n <- c(0, 0, 0, 35, 35, 35, 70, 70, 70, 105, 105, 105, 140, 140, 140)
wme <- c(9.9, 7.8, 10.7, 20.3, 22.6, 23.9, 27.5, 30.3, 29.2, 31.4, 27.2,
 33.4, 28.1, 25.7, 31.9)

-------------------------------------------------------------------------

fit1 <- lm(wme ~ n)

-------------------------------------------------------------------------

anova(fit1)
 Analysis of Variance Table

 Response: wme
           Df Sum Sq Mean Sq F value    Pr(>F)
 n          1 651.47  651.47  26.735 0.0001801 ***
 Residuals 13 316.78   24.37
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

-------------------------------------------------------------------------

plot(n, wme)
abline(fit1)
savePlot("Bild030.png", "png")
```

Mit dem dargestellten Test würde man lediglich testen, ob das (scheinbar) lineare Modell eine von Null verschiedene Steigung hat oder nicht (also ob `n` (die Prädikatorvariable) einen Effekt auf `wme` (Die Zielvariable) hat). Da hier Signifikanz angezeigt wird (`n` einen Effekt auf `wme` hat), ist die Steigung also verschieden zu Null.

Da in diesem Fall aber durchaus bezweifelt werden kann, dass es sich um Linearität handelt, muss man untersuchen, ob eine signifikante Abweichung von der Linearität gegeben ist. Dazu führt man den Lack-of-fit-Term δ_i ein, der die Abweichung der Mittelwerte der einzelnen Stufen von der Regression angibt. Im Modell von R stellt sich das folgendermaßen dar

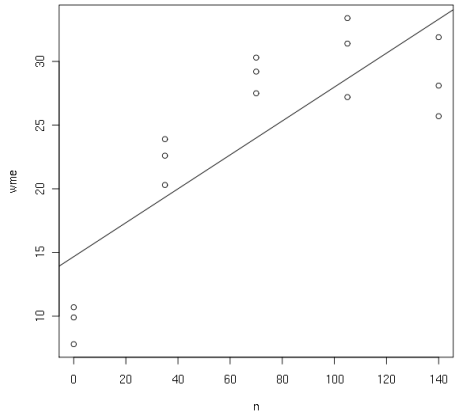


Abbildung 6.9: Schaubild der Rübenerträge gegen die N-Menge mit fraglichem linearen Zusammenhang

```
fit2 <- lm(wme ~ n + factor(n))
```

Der Term `factor(n)` stellt den „Lack-of-fit“ dar. Hier fehlt also etwas zum linearen Modell. Geprüft wird, ob dieser fehlende Effekt groß genug ist um signifikant zu sein, denn dann ist das Modell, das den Lack-of-fit enthält, nicht mit einem linearen Zusammenhang beschreibbar.

Mit `anova()` werden die beiden Modelle nun verglichen. Dies kann man als geschachteltes Modell ansehen, mehr dazu in Kapitel ’↑ 6.8 Vergleich von geschachtelten Modellen mittels F-Test (6.9)’.

```
anova(fit1, fit2)
 Analysis of Variance Table

 Model 1: wme ~ n
 Model 2: wme ~ n + factor(n)
   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
 1     13 316.78
 2     10  54.69  3    262.09 15.976 0.000384 ***
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

`Model 2` ist also signifikant von `Model 1` unterschiedlich. Nun kann man also sagen, dass die Mittelwerte der einzelnen Düngestufen nicht mit einem linearen Zusammenhang beschrieben werden können, da der Lack-of-fit Term signifikant ist.

6.6 Residuen, Modellvoraussetzungen und Ausreißer (6.7)

6.6.1 Residuen-Plots (6.7.2)

Seite 56, Beispiel 1:

Residuen in R werden von `lm()` berechnet (mehr dazu in Kapitel '↑ 6.7 Lineare Modelle in Matrizenschreibweise (6.8)'). Diese können mit `residuals()` ausgegeben werden. Dabei handelt es sich jedoch lediglich um die Roh-Residuen. Will man studentisierte/-standardisierte Residuen, kann man sich der Funktion `studres()` bedienden. Dazu muss jedoch, mit `library(MASS)`, erst das Paket `MASS` geladen werden.

```
library(MASS)
```

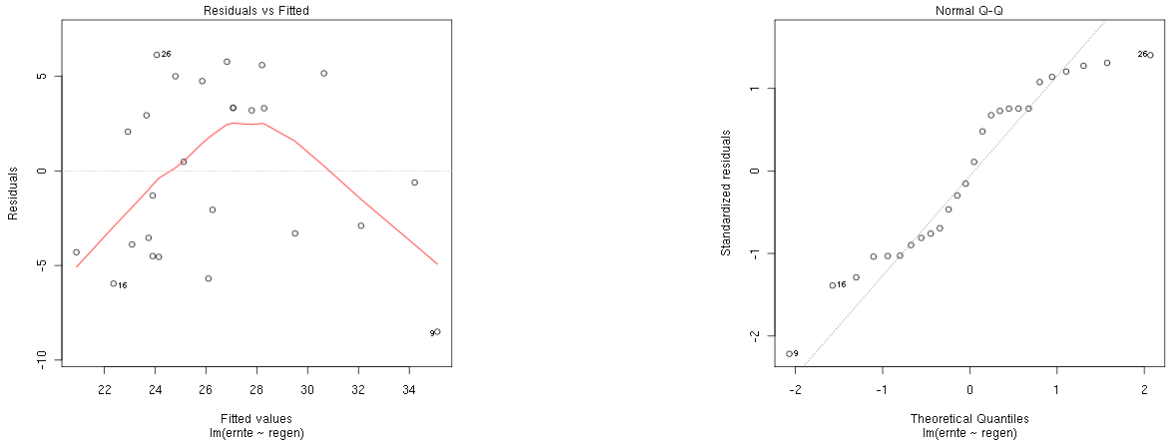
```
residuals(lm.ernte_regen)
          1           2           3           4           5
 -3.3024860  2.0722184  5.7760972  2.9416957 -4.5453195
          6           7           8           9          10
 -5.6933801 -2.8999001  5.5962210 -8.5031602 -1.3018119
         11          12          13          14          15
 -2.0557184 -4.2985519  5.0053270 -4.5018119  4.7501275
         16          17          18          19          20
 -5.9595973  3.3325897 -3.8901200 -3.5394735  0.4806502
         21          22          23          24          25
  3.2020669  5.1611453  3.3150518 -0.6102991  3.3325897
         26
  6.1358497
```

```
studres(lm.ernte_regen)
          1           2           3           4           5
 -0.7531201  0.4715102  1.3299320  0.6673455 -1.0407591
          6           7           8           9          10
 -1.3094137 -0.6866502  1.2921731 -2.4339101 -0.2924152
         11          12          13          14          15
 -0.4581077 -1.0282245  1.1467106 -1.0324545  1.0807364
         16          17          18          19          20
 -1.4177289  0.7486692 -0.8943976 -0.8058402  0.1069535
         21          22          23          24          25
  0.7204128  1.2165552  0.7483967 -0.1511367  0.7486692
         26
  1.4339563
```

Nun muss man sich nicht die Arbeit machen und jedes einzelne Schaubild zur Residuenanalyse selbst erstellen, sondern kann einfach die Funktion `plot()` verwenden um sich so verschiedene Plots zur Analyse des linearen Modells anzeigen zu lassen.

```
plot(lm.ernte_regen)
```

An dieser Stelle des Skriptes sind lediglich die ersten beiden Schaubilder dieser Analyse von Bedeutung. Hier werden für das erste Schaubild nicht die studentisierten Residuen, sondern die Rohresiduen dargestellt. Diese zeigen allerdings eine ähnliche Verteilung wie die studentisierten Residuen.



(a) Residuen gegen geschätzte Werte

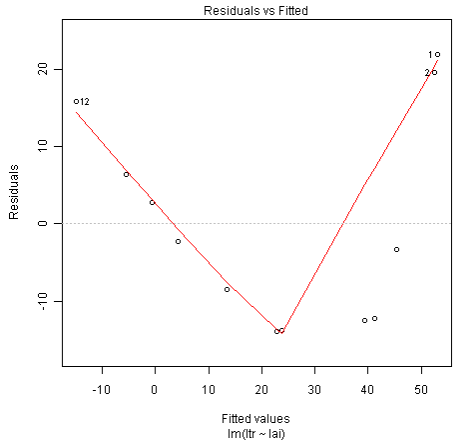
(b) Standardisierte Residuen gegen Normalscores (q-q-plot)

Abbildung 6.10: Residuenanalyse für die Regendaten

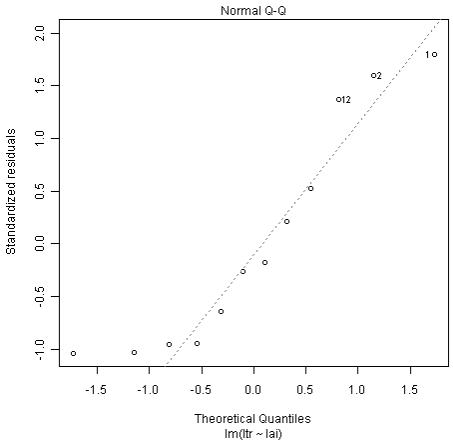
Seite 59, Beispiel 1:

```
lai <- c(0.5, 0.6, 1.8, 2.5, 2.8, 5.45, 5.6, 7.2, 8.75, 9.6, 10.4, 12)
ltr <- c(75, 72, 42, 29, 27, 10, 9, 5, 2, 2, 1, 0.9)

------------------------------------------------------------------------

fit <- lm(ltr ~ lai)

------------------------------------------------------------------------

plot(fit)
```

In diesem Beispiel sind leider nur die Daten vorhanden, die man auch in Kapitel 6.3 finden kann. Damit kann man zwar grob zeigen, welchen Effekt Daten haben, die von der Normalverteilung abweichen, mit mehr Daten wird dies allerdings wesentlich deutlicher, wie im Biometrie-Skript zu sehen ist.



(a) Residuen gegen geschätzte Werte



(b) Standardisierte Residuen gegen Normalscores (q-q-plot)

Abbildung 6.11: Residuenanalyse für die Reisdaten, die von der Normalverteilung abweichen

Seite 62, Beispiel 1:

Für dieses Beispiel muss man die Daten selbst simulieren. Dies kann man jedoch sehr leicht durchführen, man muss lediglich die Vektoren erstellen, die die Daten enthalten, beziehungsweise die beiden Achsen im Schaubild aufspannen.

```
x <- c(1:100)
y <- 10+3*x+rnorm(100, 0, x)
```

```
plot(x, y)
savePlot("Bildbla.png", "png")
```

```
lm.varhet <- lm(y ~x)
```

```
plot(lm.varhet)
```

Die Funktion `rnorm()` erzeugt zufällige normalverteilte Zahlen, dabei muss man sowohl die Anzahl der Werte angeben, als auch den Mittelwert und die Standardabweichung der Werte. Da hier nach Werten gefragt wird, deren Residuen eine Varianz von x_i^2 haben, und die Varianz die quadrierte Standardabweichung ist, kann man für den Wert der Standardabweichung `x` einsetzen.

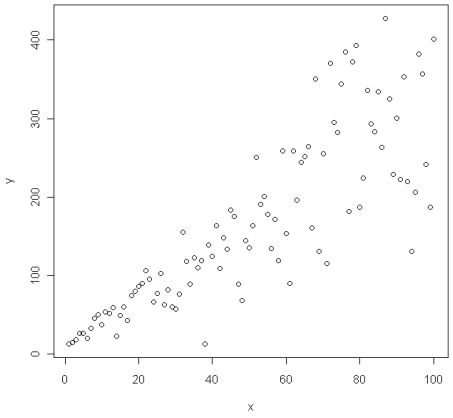
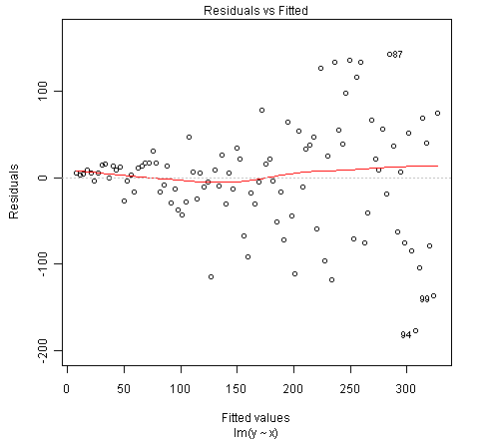
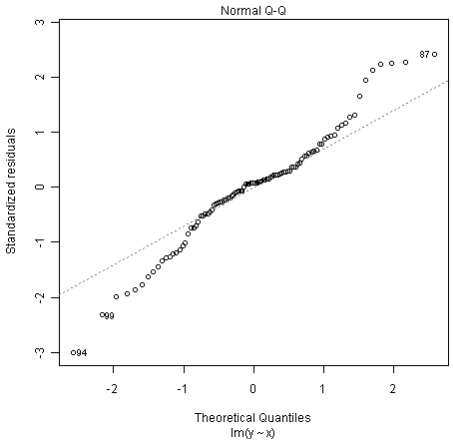


Abbildung 6.12: Schaubild von y gegen x, simulierten Daten



(a) Residuen gegen geschätzte Werte



(b) Standardisierte Residuen gegen Normalscores (q-q-plot)

Abbildung 6.13: Residuenanalyse für die Daten mit Varianzheterogenität

Seite 64, Beispiel 1:

```
x <- seq(1, 10, 0.1)
y <- x + exp(rnorm(91, 0, 1))
```

```
plot(x, y)
savePlot("Bildbla2.png", "png")
```

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

```
plot(fit)
```

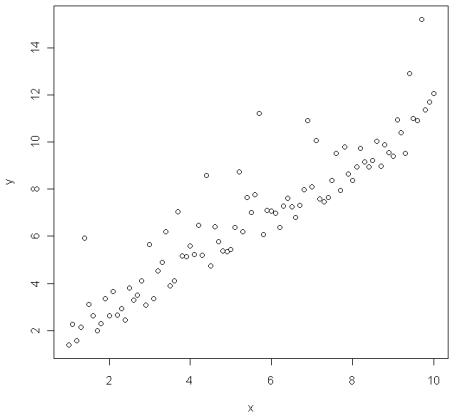
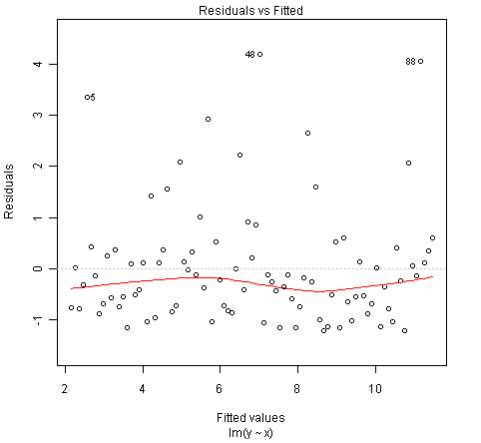
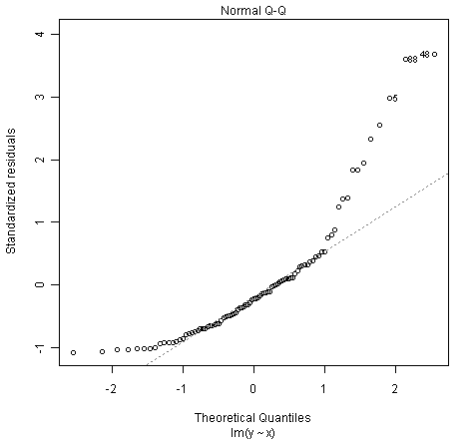


Abbildung 6.14: Schaubild von y gegen x, simulierten Daten



(a) Residuen gegen geschätzte Werte



(b) Standardisierte Residuen gegen Normalscores (q-q-plot)

Abbildung 6.15: Residuenanalyse für die Daten mit Abweichung von der Normalverteilung

6.7 Lineare Modelle in Matrizenschreibweise (6.8)

6.7.1 Die Wilkinson-Rogers-Notation

Im Biometrieskript wird an vielen Stellen erwähnt, wie eine Aufgabe mit SAS zu bewältigen ist. Für R wird dies in diesem Kapitel versucht, für SAS bietet Herr Piepho eine weiterführende Vorlesung an.

Das lineare Modell ist als

$$y = \boldsymbol{X}\boldsymbol{\beta} + e$$

darstellbar, hierbei ist $\boldsymbol{X}$ die Designmatrix, $\boldsymbol{\beta}$ der Parametervektor, der mit X multipliziert wird (und sich bei der Regressionsanalyse aus y-Achsen-Abschnitt (α) und Steigung (β) zusammensetzt; bei einer Varianzanalyse sind diese Koeffizienten andere, feste Effekte) und e der Vektor aller Fehlerterme. α und β werden mit der Kleinst-Quadrat-Schätzung geschätzt. Der Fehlerterm wird ebenfalls aus den Daten errechnet. Damit muss lediglich die Design-Matrix X spezifiziert werden.

Zum Grundlegenden Verständnis dazu sollte man sich mit der Wilkinson-Rogers Notation (Wilkinson & Rogers, 1973) (Tabelle 6.1) vertraut machen. Diese Notation vereinfacht das Erfassen von Modellen in vielen Computerprogrammen, so auch in R.

Tabelle 6.1: Wilkinson-Rogers-Notation für lineare Modelle in R

Operator	Syntax	Bedeutung im Modell
~	y ~x	y ist abhängig von x
+	x_1 + x_2	x_1 und x_2 einschließen
-	x_1 - x_2	x_1 einschließen aber x_2 ausschließen
:	x_1 : x_2	reiner Interaktionseffekt zwischen x_1 und x_2
*	x_1 * x_2	Haupteffekte und Interaktionseffekt zwischen x_1 und x_2 (Tensorprodukt)
/	x_1 / x_2	hierarchisches Modell mit Faktor x_1 und Subfaktor x_2
^	x^n	einschließen aller Interaktionen bis zur Stufe n
I()	I(x)	Interpretiere(x); wichtig für Transformationen im Modell wie x^2; (I(x^2))

Hier sei erwähnt, dass die `I()`-Notation etwas ungewöhnlich und auf den ersten Blick vielleicht unverständlich erscheint. Diese Notation ist nötig, da zum Beispiel potenzierte Terme nicht einfach so in ein Modell aufgenommen werden können. Dies muss man der Funktion hier mitteilen. In Kapitel '↑6.10 Polynomregression (6.11)' wird an einem Beispiel die Funktionsweise genauer erläutert.

6.7.2 einfache lineare Modelle

Ein einfaches lineares Modell lautet:

$$y_i = \alpha + \beta x_i + e_i$$

mit α und β als Koeffizienten und e_i als Fehler der i-ten Behandlung. Dies kann man auch in Matrizen-Schreibweise formulieren. Das ist insofern interessant und wichtig, da alle Statistik-Pakete mit Matrizen arbeiten. Will man die Details hinter einem Prozess verstehen, macht es daher immer Sinn, die zugrundeliegende Matrix zu betrachten. Für dieses Modell lautet sie:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_i \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} + \begin{pmatrix} e_1 \\ \vdots \\ e_i \\ \vdots \\ e_n \end{pmatrix}$$

oder

$$y = \boldsymbol{X}\boldsymbol{\beta} + e$$

(s.o.).

Bei den Regendaten handelt es sich um ein derartiges einfaches lineares Modell. Die Ernte eines Feldes ist laut diesem Modell linear vom Niederschlag abhängig. Das Modell lautet:

$$y = 0.08117x + 15.13554 + e$$

In R wurde dies folgendermaßen eingegeben:

```
lm.ernte_regen <- lm(ernte ~ regen)
```

```
lm.ernte_regen

 Call:
 lm(formula = ernte ~ regen)

 Coefficients:
 (Intercept)        regen
    15.13554      0.08117

abline(lm.ernte_regen)
```

`ernte` (die Zielvariable) wurde in Abhängigkeit von `regen` (der Prädikatorvariable) angegeben. Will man den y-Achsenabschnitt nicht berechnen, kann man das Modell mit `lm(ernte ~ regen - 1)` spezifizieren.

```
lm.ernte_regen <- lm(ernte ~ regen - 1)
```

```
lm.ernte_regen
```

```
Call:
lm(formula = ernte ~ regen - 1)

Coefficients:
 regen
0.1803
```

Die Funktion `lm()` schätzt nun, über die Methode der kleinsten Quadrate, die Werte für α und β und nennt diese `(Intercept)` und `regen`, respektive. Die Steigung wird mit dem Namen der Variable bezeichnet, da in einem Modell mehrere Prädikatorvariablen vorkommen können, die auf diese Weise leichter zu identifizieren sind.

Die Werte für `regen` und `ernte` werden in Vektoren angegeben, weswegen es einfach ist, diese zu und in Matrizen zu verarbeiten.

`lm()` bedient sich also genau der Matrixschreibweise $y = \boldsymbol{X}\boldsymbol{\beta}+e$, wobei y in Form der Zielvariable angegeben wird, die Designmatrix $\boldsymbol{X}$ aus den Prädikatorvariablen nach der Wilkinson-Rogers Notation abgeleitet wird, die Koeffizienten α und β über die Methode der kleinsten Quadrate geschätzt werden und im Parameterverktor $\boldsymbol{\beta}$ zusammengefasst werden und schließlich die Residuen e berechnet werden. Die Designmatrix $\boldsymbol{X}$ kann man mit der Funktion `model.matrix()` berechnen. Hier muss als Input das lm-Objekt angegeben werden.

6.7.3 Lineare Modelle mit mehreren Prädikatorvariablen

In Kapitel '↑ 6.5 Test auf Linearität (6.6)' handelt es sich um ein Modell, bei dem zu prüfen ist, ob es tatsächlich einer linearen Verteilung folgt. Hier wurde folgender Code in R verwendet:

```
fit2 <- lm(wme ~ n + factor(n))
```

Das Modellgleichung lautet:

$$y_{ij} = \alpha + \beta x_i + \delta_i + e_{ij}$$

Der Effekt durch δ_i ist der Lack-of-fit Effekt. Um diesen weiteren Fehler schätzen zu können, braucht `lm()` eine zweite Prädikatorvariable. `lm()` wurde also mitgeteilt, dass es eine zweite Prädikatorvariable gibt und dass diese `factor(n)` sein soll. Dabei werden die Faktoren von `n` (also die Düngerstufen) herangezogen.
Mit

```
anova(fit2)
 Analysis of Variance Table

 Response: wme
           Df Sum Sq Mean Sq F value    Pr(>F)
 n          1 651.47  651.47 119.127 7.088e-07 ***
```

```
factor(n)  3 262.09   87.36  15.976  0.000384 ***
Residuals 10  54.69    5.47
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

könnte man testen, ob `factor(n)` (also der Lack-of-fit Term) einen Effekt auf `wme` hat. Hier sieht man, wie auch schon bei Beispiel 1 auf Seite 50 im Skript, dass `factor(n)` einen signifikanten Effekt auf `wme` hat, also der Lack-of-fit signifikant ist.

Will man ein lineares Modell anpassen, dass mehrere Prädikatorvariablen hat, kann man sich eine ganze Menge Tipparbeit sparen, wenn man das Modell folgendermaßen definiert:

```
lmratten <- lm(end ~ ., df.ratten)

lmratten

 Call:
 lm(formula = end ~ ., data = df.ratten)

 Coefficients:
 (Intercept)       anfang       futter
     34.6257       0.4387       0.2173
```

Dieses Beispiel ist im Biometrieskript auf Seite 94 zu finden. Dabei soll ermittelt werden, von welchen Faktoren das 'Endgewicht' einer Ratte abhängig ist. Im Dataframe sind die Faktoren 'Futteraufnahme' und 'Anfangsgewicht' aufgelistet. Mit der `.`-Notation teilt man `lm()` mit, dass alle noch nicht aufgelisteten Faktoren additiv dem Modell zugeordnet werden sollen, dabei ist zu beachten, dass diese Faktoren auch tatsächlich erklärende Variablen sind. Außerdem muss definiert werden, welchem Datensatz diese Faktoren entnommen werden sollen, hier `df.ratten`.

6.7.4 Lineare Modelle mit qualitativen Prädikatorvariablen

Eine Variable ist dann qualitativ, wenn ihr Wert nicht nur als Zahl, sondern als Faktor gesehen werden kann. Der Faktor kann aus Zahlen oder aus Buchstaben bestehen, da lediglich von Bedeutung ist, dass eine Gruppe (Stufe) von der anderen abgetrennt werden kann, indem sich die Bezeichnungen für die Gruppen (Stufen) unterscheiden.

Wie das Beispiel auf Seite 69 des Biometrieskriptes zeigt, kann `anova()` auch prüfen, ob eine qualitative Variable einen Einfluss auf den Mittelwert einer numerischen Variable hat. Die qualitativen Variablen können direkt mit `anova()` getestet werden, oder als numerische Stufen erfasst werden und dann mit `factor()` in das lineare Modell aufgenommen werden.

```
ertrag <- c(31, 32, 37, 32, 21, 23, 25, 19, 27, 29, 34, 34, 34, 32, 31,
 27, 24, 23, 27, 26)
sorte.a <- gl(5, 4, labels=LETTERS[1:5])
```

```
sorte.b <- c(1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5)
df.ertrag_sorte.a <- data.frame(ertrag, sorte.a)
df.ertrag_sorte.b <- data.frame(ertrag, sorte.b)
lm.ertrag_sorte.a <- lm(ertrag ~ sorte.a, df.ertrag_sorte.a)
lm.ertrag_sorte.b <- lm(ertrag ~ factor(sorte.b), df.ertrag_sorte.b)
```

```
anova(lm.ertrag_sorte.a)
 Analysis of Variance Table

 Response: ertrag
           Df Sum Sq Mean Sq F value    Pr(>F)
 sorte.a    4 348.80   87.20  11.276 0.0001997 ***
 Residuals 15 116.00    7.73
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
anova(lm.ertrag_sorte.b)
 Analysis of Variance Table

 Response: ertrag
                Df Sum Sq Mean Sq F value    Pr(>F)
 factor(sorte.b)  4 348.80   87.20  11.276 0.0001997 ***
 Residuals       15 116.00    7.73
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
anova(lm.ertrag_sorte.a, lm.ertrag_sorte.b)
 Analysis of Variance Table

 Model 1: ertrag ~ sorte.a
 Model 2: ertrag ~ factor(sorte.b)
   Res.Df RSS Df Sum of Sq F Pr(>F)
 1     15 116
 2     15 116  0         0
```

`gl()` erstellt einen Vektor, der als Variablen Faktoren mit den spezifizierten Namen enthält. Diese sollen die Namen für die Arten repräsentieren. Da hier lediglich wichtig ist, dass die Faktoren der gleichen Gruppe angehören, kann man diese auch mit `1, 2, 3, 4, 5` bezeichnen, wie im zweiten Beispiel verdeutlicht. Der letzte Test zeigt, dass die Modelle identisch sind. In der Praxis hat sich bewährt, dass man die Werte für die qualitativen Variablen schon als solche ins Dataframe aufnimmt, da sich dann die Namen der Faktoren im Output des Modells nicht mehr ändern und man damit unter wesentlich einfacheren Bedingungen weiter Analyseschritte durchführen kann.

6.7.5 Lineare Modelle mit Interaktionen

In Kapitel '↑10 Zweifaktorielle Varianzanalyse - Wechselwirkungen' werden lineare Modelle behandelt, in denen eine Interaktion zwischen zwei qualitativen Faktoren vorliegt. Um herauszufinden, ob Interaktionen in einem Modell einen signifikanten Effekt haben, muss ein Modell, welches die Effekte enthält, gegen ein dazu um die Interaktionen reduziertes Modell getestet werden. Um die Interaktionen im linearen Modell darzustellen, verwendet man die $*$-Notation. Mehr dazu in Kapitel 10.

6.7.6 Die Normalengleichung und ihre Lösung (6.8.1)

In diesem Unterkapitel wird in kurzen Sätzen darauf eingegangen, wie die Matrixoperationen in R angewendet werden können.

Seite 75, Beispiel 1:

```
df.ernte_regen <- data.frame(ernte, regen)
lm.ernte_regen <- lm(ernte ~ regen, df.ernte_regen)
X <- model.matrix(lm(ernte~regen, df.ernte_regen))
XX <- t(X) %*% X
Xy <- t(X) %*% df.ernte_regen$ernte
XX.inv <- solve(XX)
b <- XX.inv %*% Xy
```

```
X
    (Intercept) regen
 1            1   177
 2            1    96
 3            1   144
 4            1   105
 5            1   111
 6            1   135
 7            1   209
 8            1   161
 9            1   246
 10           1   108
 11           1   137
 12           1    71
 13           1   119
 14           1   108
 15           1   132
 16           1    89
 17           1   147
 18           1    98
 19           1   106
 20           1   123
 21           1   156
```

```
 22           1   191
 23           1   162
 24           1   235
 25           1   147
 26           1   110
 attr(,"assign")
 [1] 0 1
```

```
XX
             (Intercept)  regen
 (Intercept)          26   3623
 regen              3623 553187
```

```
Xy
                [,1]
 (Intercept)   687.6
 regen       99737.8
```

```
XX.inv
              (Intercept)         regen
 (Intercept)  0.440178622 -2.882872e-03
 regen       -0.002882872  2.068856e-05
```

```
b
                     [,1]
 (Intercept) 15.13553937
 regen        0.08116919
```

```
lm.ernte_regen

 Call:
 lm(formula = ernte ~ regen, data = df.ernte_regen)

 Coefficients:
 (Intercept)        regen
    15.13554      0.08117
```

Mit der Funktion `model.matrix()` erstellt man eine Design-Matrix. Mit `t(Matrix)` erstellt man die Transponierte einer Matrix und mit `solve(Matrix)` bildet man die Inverse einer Matrix. Mit der `%*%` führt man Multiplikationen bei Matrizen durch. In `b` finden wir also die Koeffizienten für die Geradengleichung für die gesuchte Regression.

Wie weiter oben schon beschrieben, handelt es sich bei dieser Methode lediglich um eine Demonstation der Rechenoperationen bei Matrizen, da `lm()` diese Regression mit wesentlich weniger Aufwand erstellt.

6.7.7 Vertrauensintervall und Tests für Linearkombinationen der Parameter (6.8.2)

Seite 86, Beispiel 1:

```
k <- as.matrix(c(1, 200))
kt <- t(k)
L.dach <- kt%*%b
rang.x <- rankMatrix(X)[1]
n <- length(df.ernte_regen$regen)
t.tab <- qt(1-.05/2, n - rang.x)
SQ.Fehler <- anova(lm.ernte_regen)$"Sum Sq"[2]
FG.Fehler <- n - rang.x
s2 <- SQ.Fehler/FG.Fehler
z <- sqrt(kt%*%XX.inv%*%k*s2)
L <- c(L.dach-t.tab*z, L.dach+t.tab*z)

------------------------------------------------------------------------

k
      [,1]
 [1,]    1
 [2,]  200

------------------------------------------------------------------------

kt
      [,1] [,2]
 [1,]    1  200

------------------------------------------------------------------------

L.dach
          [,1]
 [1,] 31.36938

------------------------------------------------------------------------

rank.x
 [1] 2

------------------------------------------------------------------------

n
 [1] 26
```

```
------------------------------------------------------------------------

t.tab
 [1] 2.063899

------------------------------------------------------------------------

SQ.Fehler
 [1] 486.1267

------------------------------------------------------------------------

FG.Fehler
 [1] 24

------------------------------------------------------------------------

s2
 [1] 20.25528

------------------------------------------------------------------------

L
 [1] 28.22527 34.51349
```

Mit der Funktion `rankMatrix()` wurde der Rang der Matrix ermittelt. Bei dieser Funktion kann man einige Einstellungen vornehmen, die aber an der Stelle nicht wichtig sind. Mit der Funktion `qt()` wird das 97,5%-Quantil der t-Verteilung bei 24 Freiheitsgraden ermittelt. Alle weiteren Rechenoperationen sollten zu diesem Punkt der Skriptes bekannt sein.

Die oben gezeigte Methode ist recht umständlich und mit recht viel Schreibarbeit verbunden, daher macht es Sinn, sich die Funktion `predict()` anzusehen, die vorhersagen zu Objekten trifft, die von der Klasse des Objektes und den Argumenten, die in der Funktion definiert werden, abhängen.

```
fit <- lm( ernte ~ regen )

------------------------------------------------------------------------

predict(fit, newdata=data.frame(regen=200), se.fit=TRUE,
 interval="confidence")
 $fit
        fit      lwr      upr
 1 31.36938 28.22527 34.51349

 $se.fit
 [1] 1.523383
```

```
$df
[1] 24

$residual.scale
[1] 4.500586
```

Die Funktion `predict()` ruft hier die eigentliche Funktion `predict.lm()` auf, deren Hilfe-Kontext man entnehmen kann, wie die Vorhersagen zu spezifizieren sind. Mit dem Argument `se.fit =` wird angezeigt, dass der Standardfehler angezeigt werden soll, mit dem Argument `intervall =` wird angegeben, welches Intervall berechnet werden soll. Das Argument `newdata =` ist ein Dataframe, in dem spezifiziert wird, welche Variabel zur Vorhersage betrachtet werden soll. So ist also sehr einfach das Vertrauensintervall ermittelt, dass ein Regen-Wert von 200 mm an Ertrag liefert.

Seite 86, Beispiel 2:

```
confint(fit)
                 2.5 %     97.5 %
 (Intercept) 8.97282667 21.2982521
 regen       0.03891959  0.1234188
```

Auch hier kann man die Funktion `confint()` einsetzen.

6.8 Vergleich von geschachtelten Modellen mittels F-Test (6.9)

Beim Testen auf Linearität wurde schon angesprochen, dass es sich dabei um ein geschachteltes Modell handelt. Ein geschachteltes Modell zeichnet sich dadurch aus, dass ein reduziertes Modell von einem vollen Modell abgeleitet wird. In diesem reduzierten Modell wird angenommen, dass eine der Prädikatorvariablen keine Effekt auf die Zielvariable hat.

Seite 92, Beispiel 1:

Bei den Regendaten haben wir nur eine Prädikatorvariable, will man also ein reduziertes Modell davon haben, muss man das in R folgendermaßen spezifizieren:

```
lm.ernte_regen <- lm(ernte ~ regen, df.ernte_regen)
lm.ernte_regen.0 <- lm(ernte ~ 1)

-------------------------------------------------------------------------

anova(lm.ernte_regen)
 Analysis of Variance Table
```

```
 Response: ernte
           Df Sum Sq Mean Sq F value    Pr(>F)
 regen      1 318.46  318.46  15.722 0.0005754 ***
 Residuals 24 486.13   20.26
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
anova(lm.ernte_regen.0)
 Analysis of Variance Table

 Response: ernte
           Df Sum Sq Mean Sq F value Pr(>F)
 Residuals 25 804.58   32.18
```

Hier ist die Zielvariable `ernte` also lediglich vom Mittelwert und dem Fehler abhängig. Man hat einen Freiheitsgrad mehr, da ein Parameter weniger geschätzt werden musste, allerdings sind die Fehler wesentlich größer, da dieses Modell die Effekte eben nicht so gut erklären kann wie das volle Modell. Ob jedoch eine Signifikanz zwischen den beiden Modellen gegeben ist, muss erst geprüft werden.

Seite 93, Beispiel 1:

```
lm.ertrag_sorte.a <- lm(ertrag ~ sorte.a, df.ertrag_sorte.a)
lm.ertrag_sorte.0 <- lm(ertrag ~ 1, df.ertrag_sorte.a)
```

```
lm.ertrag_sorte.a

 Call:
 lm(formula = ertrag ~ sorte.a, data = df.ertrag_sorte.a)

 Coefficients:
 (Intercept)     sorte.aB     sorte.aC     sorte.aD
          33          -11           -2           -2
    sorte.aE
          -8
```

```
lm.ertrag_sorte.0

 Call:
 lm(formula = ertrag ~ 1, data = df.ertrag_sorte.a)

 Coefficients:
 (Intercept)
```

```
        28.4

-------------------------------------------------------------------------

anova(lm.ertrag_sorte.0, lm.ertrag_sorte.a)
 Analysis of Variance Table

 Model 1: ertrag ~ 1
 Model 2: ertrag ~ sorte.a
   Res.Df   RSS Df Sum of Sq      F    Pr(>F)
 1     19 464.8
 2     15 116.0  4     348.8 11.276 0.0001997 ***
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Auch hier sind höchst signifikante Unterschiede zu erkennen, das reduzierte Modell gilt also nicht, die Sorte hat einen Einfluss auf den Ertrag.

6.9 Multiple lineare Regression (6.10)

Um die Daten hier einzulesen, möchte ich nun einmal auf die übliche Art und Weise zurückgreifen, wie dies in R getan wird.

Für die Rattendaten habe ich die Datei 'Rattendaten.txt' erstellt. Hierein habe ich die Daten so geschrieben, wie es Excel ausgeben würde, würde man die darin enthaltenen Informationen in eine *.txt-Datei speichern würde. Um die Daten in R einzulesen verwendet man die Funktion `read.table()`.

```
df.ratten <- read.table("/media/studium/Studienbegleitende Unterlagen/R
 für Vl Biometrie I_II/Dokument/Rattendaten.txt", header=TRUE, sep="\t",
 dec=".")

-------------------------------------------------------------------------

head(df.ratten)
   anfang futter   end
 1   55.8    289 114.8
 2   45.8    316 109.7
 3   48.1    304 111.3
 4   43.3    299 126.0
 5   50.1    353 144.7
 6   40.1    298 121.0

-------------------------------------------------------------------------

summary(df.ratten)
     anfang          futter           end
  Min.   :32.40   Min.   :224.0   Min.   :103.0
  1st Qu.:40.95   1st Qu.:289.5   1st Qu.:110.5
  Median :45.80   Median :303.0   Median :120.2
```

```
  Mean   :45.73   Mean   :304.7   Mean   :120.9
  3rd Qu.:49.10   3rd Qu.:317.0   3rd Qu.:127.5
  Max.   :65.40   Max.   :355.0   Max.   :144.7
```

```
df.ratten <- stack(df.ratten)
```

```
summary(df.ratten)
      values          ind
  Min.   : 32.4   anfang:35
  1st Qu.: 50.1   end   :35
  Median :120.2   futter:35
  Mean   :157.1
  3rd Qu.:289.0
  Max.   :355.0
```

```
df.ratten <- unstack(df.ratten)
```

```
head(df.ratten)
   anfang   end futter
 1   55.8 114.8    289
 2   45.8 109.7    316
 3   48.1 111.3    304
 4   43.3 126.0    299
 5   50.1 144.7    353
 6   40.1 121.0    298
```

Mit dem Argument `header =` definiert man, ob R die Spaltenbezeichnung mit importieren soll, oder lediglich die Daten. Mit dem Argument `sep =` kann man definieren, welchen Operator R zum trennen der einzelnen Daten erkennen soll (engl. seperator), `"\t"` steht dabei für Tabulator (Speichert man eine *.xls-Datei als *.txt-Datei, fügt Excel zur Abgrenzung der einzelnen Zellen einen Tabulator ein. Mit `sep =` teilt man R nun also mit, dass die Datenwerte durch einen Tabulator getrennt sind). Mit dem Argument `dec =` sollte man das Trennzichen wählen, dass man mit `options(OutDec="")` definiert hat. Hier ist zu empfehlen, dass man sich generell an die Standards der englischen Sprache hält, da man sonst „Zeichenfehler“ macht, die einem nicht auffallen, da man sich keine Gedanken darüber gemacht hat.

Mit der Funktion `stack()` kann man die Werte in Abhängigkeit ihrer Bezeichnung auflisten. Das ist dann interessant, wenn man mit kategorialen Daten arbeitet, also zum Beispiel eine Artenliste verarbeiten will (in diesem Beispiel jedoch eher nicht). `unstack()` macht dies wieder rückgängig.

Seite 94, Beispiel 1:

```
lm.ratten <- lm(df.ratten$end ~ df.ratten$anfang + df.ratten$futter)
lm.ratten.1 <- lm(df.ratten$end ~ df.ratten$futter)
lm.ratten.2 <- lm(df.ratten$end ~ df.ratten$anfang)

------------------------------------------------------------------------

lm.ratten

 Call:
 lm(formula = df.ratten$end ~ df.ratten$anfang + df.ratten$futter)

 Coefficients:
      (Intercept)  df.ratten$anfang  df.ratten$futter
          34.6257            0.4387            0.2173

------------------------------------------------------------------------

anova(lm.ratten, lm.ratten.1)
 Analysis of Variance Table

 Model 1: df.ratten$end ~ df.ratten$anfang + df.ratten$futter
 Model 2: df.ratten$end ~ df.ratten$futter
   Res.Df    RSS Df Sum of Sq      F  Pr(>F)
 1     32 3079.9
 2     33 3371.0 -1    -291.0 3.0239 0.09166 .
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

------------------------------------------------------------------------

anova(lm.ratten, lm.ratten.2)
 Analysis of Variance Table

 Model 1: df.ratten$end ~ df.ratten$anfang + df.ratten$futter
 Model 2: df.ratten$end ~ df.ratten$anfang
   Res.Df     RSS Df Sum of Sq      F   Pr
 1     32  3079.9
 2     33  4140.1 -1   -1060.1 11.014 0.002263 **
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

------------------------------------------------------------------------

lm.ratten.1

 Call:
 lm(formula = df.ratten$end ~ df.ratten$futter)

 Coefficients:
      (Intercept)  df.ratten$futter
```

```
          46.5489            0.2440
```

Hieraus kann man erkennen, dass das um `df.ratten$anfang` reduzierte Modell sich nicht signifikant vom vollen Modell unterscheidet, während das um `df.ratten$futter` reduzierte Modell sich signifikant vom vollen Modell unterscheidet. Wird `df.ratten$futter` dem Modell entnommen, kommt ein anderes Ergebnis heraus. Daraus schließt man, dass `df.ratten$futter` einen Effekt hat, während sich das Modell nach Entnahme von `df.ratten$anfang` nicht signifikant vom vollen Modell unterscheidet (dieses reduzierte Modell also den Zusammenhang genauso erklärt, wie das Volle), dieser Faktor also keinen Effekt hat.

6.9.1 Das multiple Bestimmtheitsmaß (6.10.2)

Das multiple Bestimmtheitsmaß ist in der Ausgabe der Funktion `summary()` zum linearen Modell als `Multiple R-squared` enthalten.

```
summary(lm.ratten)

 Call:
 lm(formula = df.ratten$end ~ df.ratten$anfang + df.ratten$futter)

 Residuals:
      Min       1Q   Median       3Q      Max
 -19.1577  -7.6131   0.1422   5.3334  23.3541

 Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
 (Intercept)      34.62570   20.63443   1.678  0.10308
 df.ratten$anfang  0.43874    0.25230   1.739  0.09166 .
 df.ratten$futter  0.21731    0.06548   3.319  0.00226 **
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

 Residual standard error: 9.811 on 32 degrees of freedom
 Multiple R-squared: 0.3564,     Adjusted R-squared: 0.3162
 F-statistic: 8.862 on 2 and 32 DF,  p-value: 0.0008658
```

6.9.2 Multikollinearität (6.10.3)

```
df.trees <- read.table("G:/R für Vl Biometrie I_II/Dokument/trees.txt",
 header=TRUE, sep="\t", dec=".")
lm.trees <- lm(Volume ~ Diam + Diam2, df.trees)
lm.trees2 <- lm(Volume ~ Diam2 + Diam, df.trees)
lm.trees3 <- lm(Volume ~ Diam, df.trees)
lm.trees4 <- lm(Volume ~ Diam2, df.trees)

-------------------------------------------------------------------------
```

```
anova(lm.trees2) # Test für Diam bereinigt um Diam2
 Analysis of Variance Table

 Response: Volume
           Df Sum Sq Mean Sq  F value Pr(>F)
 Diam2      1 7532.4  7532.4 403.8669 <2e-16 ***
 Diam       1   51.5    51.5   2.7586 0.1079
 Residuals 28  522.2    18.7
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(lm.trees) # Test für Diam2 bereinigt um Diam
 Analysis of Variance Table

 Response: Volume
           Df Sum Sq Mean Sq  F value Pr(>F)
 Diam       1 7581.8  7581.8 406.5139 <2e-16 ***
 Diam2      1    2.1     2.1   0.1116 0.7408
 Residuals 28  522.2    18.7
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(lm.trees3) # Test für Diam alleine
 Analysis of Variance Table

 Response: Volume
           Df Sum Sq Mean Sq F value    Pr(>F)
 Diam       1 7581.8  7581.8  419.36 < 2.2e-16 ***
 Residuals 29  524.3    18.1
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(lm.trees4) # Test für Diam2 alleine
 Analysis of Variance Table

 Response: Volume
           Df Sum Sq Mean Sq F value    Pr(>F)
 Diam2      1 7532.4  7532.4  380.78 < 2.2e-16 ***
 Residuals 29  573.7    19.8
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Diam <- df.trees$Diam
```

```
Diam2 <- df.trees$Diam2
```

```
cor(Diam, Diam2)
 [1] 0.9951005
```

```
summary(lm.trees)

 Call:
 lm(formula = Volume ~ Diam + Diam2, data = df.trees)

 Residuals:
     Min      1Q  Median      3Q     Max
 -8.1398 -3.0906  0.1918  3.6109  9.7435

 Coefficients:
             Estimate Std. Error t value Pr(>|t|)
 (Intercept)  -37.016      3.425 -10.808 1.68e-11 ***
 Diam           4.221      2.541   1.661    0.108
 Diam2          0.707      2.116   0.334    0.741
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

 Residual standard error: 4.319 on 28 degrees of freedom
 Multiple R-squared: 0.9356,     Adjusted R-squared: 0.931
 F-statistic: 203.3 on 2 and 28 DF,  p-value: < 2.2e-16
```

```
summary(lm.trees3)

 Call:
 lm(formula = Volume ~ Diam, data = df.trees)

 Residuals:
     Min      1Q  Median      3Q     Max
 -8.0654 -3.1067  0.1520  3.4948  9.5868

 Coefficients:
             Estimate Std. Error t value Pr(>|t|)
 (Intercept) -36.9435     3.3651  -10.98 7.62e-12 ***
 Diam          5.0659     0.2474   20.48  < 2e-16 ***
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

 Residual standard error: 4.252 on 29 degrees of freedom
 Multiple R-squared: 0.9353,     Adjusted R-squared: 0.9331
 F-statistic: 419.4 on 1 and 29 DF,  p-value: < 2.2e-16
```

```
-------------------------------------------------------------------------

summary(lm.trees4)

 Call:
 lm(formula = Volume ~ Diam2, data = df.trees)

 Residuals:
      Min       1Q   Median       3Q      Max
 -8.48516 -2.92001 -0.03166  2.84336 10.82193

 Coefficients:
             Estimate Std. Error t value Pr(>|t|)
 (Intercept) -36.8253     3.5250  -10.45 2.43e-11 ***
 Diam2         4.2042     0.2155   19.51  < 2e-16 ***
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

 Residual standard error: 4.448 on 29 degrees of freedom
 Multiple R-squared: 0.9292,     Adjusted R-squared: 0.9268
 F-statistic: 380.8 on 1 and 29 DF,  p-value: < 2.2e-16
```

6.9.3 Variablenselektion (6.10.4)

Mit der Funktion `summary()` kann man sich die Zusammenfassung eines Modells ansehen. Hier ist nicht nur das multiple Bestimmtheitsmaß zu finden, sondern auch das adjustierte Bestimmtheitsmaß `Adjusted R-squared` und einige andere Statistik. Wie man zum mittleren Abweichungsquadrat oder Mallows C_p kommt, wird im weiteren Verlauf des Kapitels erklärt.

Seite 111, Beispiel 1:

```
df.zement <- read.table("/media/studium/Studienbegleitende Unterlagen/R
 für Vl Biometrie I_II/Dokument/Zement.txt", sep="\t", dec=".",
 header=TRUE)
lm.zement <- lm(df.zement$y ~ df.zement$x1 + df.zement$x2 + df.zement$x3
 + df.zement$x4)
lm.zement.3 <- lm(df.zement$y ~ df.zement$x1 + df.zement$x2
 + df.zement$x4)

-------------------------------------------------------------------------

head(df.zement)
   x1 x2 x3 x4     y
 1  7 26  6 60  78.5
 2  1 29 15 52  74.3
 3 11 56  8 20 104.3
 4 11 31  8 47  87.6
 5  7 52  6 33  95.9
```

```
 6 11 55  9 22 109.7
```

```
summary(lm.zement)

 Call:
 lm(formula = df.zement$y ~ df.zement$x1 + df.zement$x2 + df.zement$x3 +
     df.zement$x4)

 Residuals:
     Min      1Q  Median      3Q     Max
 -3.1719 -1.7596  0.1948  1.3392  4.3634

 Coefficients:
              Estimate Std. Error t value Pr(>|t|)
 (Intercept)   63.6712    73.0416   0.872   0.4088
 df.zement$x1   1.5424     0.7763   1.987   0.0822 .
 df.zement$x2   0.4974     0.7545   0.659   0.5282
 df.zement$x3   0.0892     0.7867   0.113   0.9125
 df.zement$x4  -0.1574     0.7391  -0.213   0.8367
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

 Residual standard error: 2.55 on 8 degrees of freedom
 Multiple R-squared: 0.9809,     Adjusted R-squared: 0.9714
 F-statistic:   103 on 4 and 8 DF,  p-value: 6.487e-07
```

```
summary(lm.zement.3)

 Call:
 lm(formula = df.zement$y ~ df.zement$x1 + df.zement$x2 + df.zement$x4)

 Residuals:
     Min      1Q  Median      3Q     Max
 -3.0991 -1.8398  0.2286  1.2547  4.3395

 Coefficients:
              Estimate Std. Error t value Pr(>|t|)
 (Intercept)   71.7610    14.7370   4.869 0.000884 ***
 df.zement$x1   1.4556     0.1219  11.939 8.04e-07 ***
 df.zement$x2   0.4151     0.1934   2.146 0.060405 .
 df.zement$x4  -0.2383     0.1806  -1.320 0.219490
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

 Residual standard error: 2.406 on 9 degrees of freedom
 Multiple R-squared: 0.9809,     Adjusted R-squared: 0.9746
 F-statistic: 154.2 on 3 and 9 DF,  p-value: 4.702e-08
```

All subset regression:

Für derart komplexe Fragestellungen wie hier, reichen die Standardpakete von R nicht mehr aus um Funktionen bereit zu stellen. Für die nächsten Kapitel ist das Paket `wle: Weighted Likelihood Estimation` wichtig.

Will man alle Regressionsmodelle durchgerechnet sehen, kann man sich dem Paket `leaps: regression subset selection` und der Funktion `leaps()` bedienen. Da diese Funktion aber keinen sehr schönen Output liefert, gehe ich an dieser Stelle nicht näher darauf ein.

Best subset regression:

Mit der Funktion `mle.cp()` kann man Regressionsmodelle anhand des Mallows C_p selektieren.

```
lm.zement <- lm(df.zement$y ~ ., df.zement)

--------------------------------------------------------------------------

lm.zement

 Call:
 lm(formula = df.zement$y ~ ., data = df.zement)

 Coefficients:
 (Intercept)           x1           x2           x3           x4
     63.6712       1.5424       0.4974       0.0892      -0.1574

--------------------------------------------------------------------------

mle.cp(lm.zement)

 Call:
 mle.cp(formula = lm.zement)


 Mallows Cp:
      (Intercept) x1 x2 x3 x4    cp
 [1,]           1  1  1  0  0 2.564
 [2,]           1  1  1  0  1 3.013
 [3,]           1  1  1  1  0 3.045
 [4,]           1  1  0  1  1 3.435
 [5,]           0  1  1  1  1 3.760
 [6,]           1  1  1  1  1 5.000

 Printed the first  6  best models
```

Hierbei bedeutet eine `1` in der Matrix, dass der Faktor in das (virtuelle) Modell mit aufgenommen wurde um Mallows C_p zu berechnen.

Backward elimination:

Um die Backward-, Forward- und Stepwise-selection durchzuführen benötigt man die Funktion `mle.stepwise()`. Hiermit werden die F-Werte der F-Statistik dargestellt, mit der die Modelle gegeneinander getestet werden. Da diese Werte nicht immer die beste Wahl sind, sollte man die hier dargestellten Selektionen mit `mle.cp()` überprüfen. Mit den Argumenten `f.in = 4` und `f.out = 4` kann man die F-Werte festlegen, bei denen ein Faktor in das Modell aufgenommen oder herausgenommen werden soll. Standardmäßig sind diese auf 4 eingestellt.

```
mle.stepwise(lm.zement, type="Backward")

 Call:
 mle.stepwise(formula = lm.zement, type = "Backward")


  Backward  selection procedure

 F.out:  4

 Last  2  iterations:
      (Intercept) x1 x2 x3 x4
 [1,]           1  1  1  0  1 0.01285
 [2,]           1  1  1  0  0 1.74200
```

Hierbei bedeutet eine `0`, dass der Faktor entfernt wurde um die Signifikanz zum vollen Modell zu prüfen. In der zweiten Zeile der Matrix wurde neben x_3 der Faktor x_4 entfernt. Mit dem so reduzierten Modell findet man den kleinsten C_p-Wert, wie `mle.cp(lm.zement)` zeigt.

Forward selection:

```
mle.stepwise(lm.zement, type="Forward")

 Call:
 mle.stepwise(formula = lm.zement, type = "Forward")


  Forward  selection procedure

 F.in:  4

 Last  3  iterations:
      (Intercept) x1 x2 x3 x4
 [1,]           1  0  0  0  1  22.660
 [2,]           1  1  0  0  1 103.300
 [3,]           1  1  1  0  1   4.606
```

Wie man sieht, wurde hier lediglich der Typ verändert. Der Output sieht ähnlich aus, allerdings ist diesmal wichtig, welche Faktoren hinzugefügt wurden. Wie auch mit der SAS-Routine, wird x_4 als erster Faktor aufgenommen, da x_4, im Vergleich zu allen Variablen, den größten Effekt zu haben scheint. Ist x_4 in das Modell aufgenommen, folgen x_2 und x_3. Diese haben laut `mle.stepwise()` beide einen signifikanten Effekt auf y. x_3 hat keinen Zusätzlichen positiven Effekt mehr auf das Modell, wird also nicht aufgenommen. An diesem Beispiel kann man also auch in R demonstrieren, dass ein Blick auf Mallows C_p sinnvoll ist, um zu überprüfen, ob tatsächlich das beste Modell ausgewählt wurde. Dass der Grund für den hohen Erklärwert von x_4 in der Multikollinearität liegt, wird hieraus nicht ersichtlich.

Stepwise regression:

```
mle.stepwise(lm.zement, type="Stepwise")

 Call:
 mle.stepwise(formula = lm.zement, type = "Stepwise")


  Stepwise  selection procedure

 F.in:  4
 F.out:  4

 Last  4  iterations:
      (Intercept) x1 x2 x3 x4
 [1,]           1  0  0  0  1  22.660
 [2,]           1  1  0  0  1 103.300
 [3,]           1  1  1  0  1   4.606
 [4,]           1  1  1  0  0   1.742
```

Diese Methode bessert den durch die Forward-selection begangenen Fehler wieder aus, indem der Faktor x_4 wieder entfernt wird, da der F-Wert erst dann unter 4 fällt, wenn der Faktor wieder aus dem Modell entfernt wird.

Diese Funktion scheint noch in der Entwicklung zu sein, da man sich leicht vorstellen kann, wie sie weiter verbessert werden könnte. Hier könnte eine ganze Menge mehr Statistik dargestellt werden, die sowieso berechnet werden muss, um zu diesem Ergebnis zu kommen. In SAS scheint dieses Verfahren wesentlich besser integriert zu sein, wie das Biometrieskript zeigt.

Seite 117, Beispiel 1:

```
df.margarine <- read.table("/media/studium/Studienbegleitende Unterlagen/R
 für Vl Biometrie I_II/Dokument/Margarine.txt", sep="\t", dec=".",
 header=TRUE)
lm.margarine <- lm(Absatz ~., df.margarine)
```

```
slm.m <- step(lm.margarine)
```

```
lm.margarine

 Call:
 lm(formula = Absatz ~ ., data = df.margarine)

 Coefficients:
 (Intercept)        Preis      Werbung      Besuche
    785.0628     -47.5382       0.5584       9.6673
```

```
mle.cp(lm.margarine)

 Call:
 mle.cp(formula = lm.margarine)


 Mallows Cp:
 (Intercept)       Preis     Werbung     Besuche          cp
           1           1           1           1           4

 Printed the first  1  best models
```

```
slm.m
 Start:  AIC=378.54
 Absatz ~ Preis + Werbung + Besuche

           Df Sum of Sq     RSS     AIC
 <none>                  826851     379
 - Preis    1    210272 1037123     385
 - Besuche  1    820195 1647046     402
 - Werbung  1   2956597 3783448     433
```

```
summary(slm.m)

 Call:
 lm(formula = Absatz ~ Preis + Werbung + Besuche, data = df.margarine)

 Residuals:
     Min      1Q  Median      3Q     Max
 -270.00 -140.26  -27.58  134.30  232.47

 Coefficients:
              Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept) 785.06282  228.23049   3.440  0.00160 **
Preis       -47.53817   16.41000  -2.897  0.00664 **
Werbung       0.55844    0.05141  10.863 1.96e-12 ***
Besuche       9.66732    1.68968   5.721 2.19e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 158.3 on 33 degrees of freedom
Multiple R-squared: 0.8445,     Adjusted R-squared: 0.8303
F-statistic: 59.72 on 3 and 33 DF,  p-value: 2.005e-13
```

Hier wird mit `mle.cp()` nur ein Modell ausgegeben. Wie schon erwähnt, muss diese Funktion noch durch andere Funktionen ergänzt werden. Will man die Kleinst-Quadrat-Schätzung, muss man diese aus `lm()` ablesen, indem man dort ein reduziertes Modell schätzen lässt.

Generell kann man sagen, dass Mallows C_p nicht das einzige Kriterium für die Modellwahl ist. In R scheint vielmehr das Informationskriterium `AIC` (Akaike Information Criterion)von Bedeutung zu sein, da die darauf beruhende Methode sehr professionell integriert ist und auch über die Standardpakete verfügbar ist. An erster Stelle ist der `AIC`-Wert aufgelistet, dann folgt der lineare Zusammenhang. Darunter ist aufgelistet, wie sich einige Werte verändern, wenn man Faktoren wegnimmt oder hinzufügt. Hier wurden lediglich Faktoren weggenommen, daher ist überall ein – davor. Es ist klar zu erkennen, dass mit jedem wegfallenden Faktor der SSQ-Wert (SQ_{Fehler}) ansteigt. Außerdem wird `AIC` immer größer. Dies ist genauso wie bei Mallows C_p mit steigendem Wert ein Zeichen dafür, dass das Modell die Daten nicht angemessen beschreibt. Mit der Funktion `summary()` kann man sich auch hier eine Varianzanalysetabelle und alle wichtigen statistischen Werte anzeigen lassen.

6.10 Polynomregression (6.11)

In diesem Kapitel zeigt sich, ob man verstanden hat, wie die Funktion `lm()` funktioniert, da hier das erste mal eine etwas abstraktere Anwendung statt findet. Da auch ein Polynom durch die Kleinstquadratschätzung geschätzt werden kann, kann man hier wieder die Funktion `lm()` anwenden. Nun muss man die Faktoren aber dementsprechend definieren, dass das Modell auch tatsächlich ein Polynom repräsentiert.

Ein Polynom hat die verallgemeinerte Form

$$\eta(x) = \beta_0 x^0 + \beta_1 x^1 + \beta_2 x^2 + ... + \beta_n x^n$$

, was mit $x^0 = 1$ und $\beta_0 = \alpha$ auch als

$$\eta(x) = \alpha + \beta_1 x^1 + \beta_2 x^2 + ... + \beta_n x^n$$

dargestellt werden kann. Ein Polynom zweiten Grades ist ein Polynom, bei dem maximal ein Faktor zweiter Potenz vorhanden ist ($\eta(x) = \alpha + \beta_1 x^1 + \beta_2 x^2$). Bei dem Faktor x^2

handelt es sich im Modell lediglich um eine weitere Prädikatorvariable, die durch β_2 gewichtet wird. Damit ist also klar, welches Modell in `lm()` spezifiziert werden muss, das selbe wie bei linearen Zusammenhängen.

Seite 119, Beispiel 1:

```
kalk <- c(0, 2, 4, 6, 8)
weizen <- c(44.4, 54.6, 63.8, 65.7, 68.9)

------------------------------------------------------------------------

kalk1 <- kalk
kalk2 <- kalk**2
kalk3 <- kalk**3
df.kalk <- data.frame(weizen, kalk, kalk**2, kalk**3)

------------------------------------------------------------------------

df.kalk
   weizen kalk1 kalk2 kalk3
 1   44.4     0     0     0
 2   54.6     2     4     8
 3   63.8     4    16    64
 4   65.7     6    36   216
 5   68.9     8    64   512

------------------------------------------------------------------------

lm.kalk.12 <- lm(weizen ~ 1, df.kalk)
lm.kalk.2 <- lm(weizen ~ kalk1, df.kalk)
lm.kalk <- lm(weizen ~ kalk + I(kalk**2), df.kalk)

------------------------------------------------------------------------

anova(lm.kalk.12, lm.kalk.2, lm.kalk)
 Analysis of Variance Table

 Model 1: weizen ~ 1
 Model 2: weizen ~ kalk1
 Model 3: weizen ~ kalk + I(kalk^2)
   Res.Df    RSS Df Sum of Sq       F   Pr(>F)
 1      4 397.31
 2      3  36.11  1    361.20 195.214 0.005084 **
 3      2   3.70  1     32.41  17.514 0.052629 .
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

------------------------------------------------------------------------

x <- seq(0, 8, 0.1)
```

```
-------------------------------------------------------------------------

plot(kalk, weizen)
lines(x, lm.kalk$coef[1] + lm.kalk$coef[2]*x
 + lm.kalk$coef[[3]]*x**2)
savePlot("Bild033.png", "png")
```

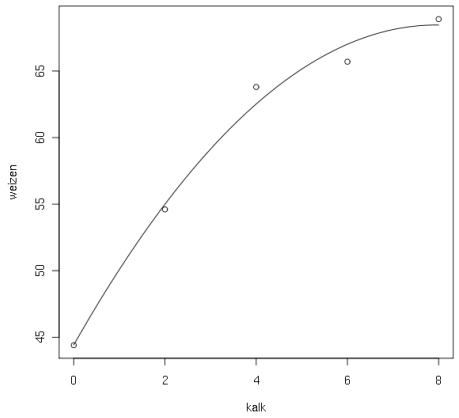


Abbildung 6.16: Schaubild der Weizenerträge in Abhängigkeit der Kalkgabe mit angepasstem Polynom

Ein Polynom höheren Grades kann mit der `I()`-Notation leicht erstellt werden, indem das Modell um die Terme höherer Potenz erweitert wird, die gewünscht sind.

6.11 „Eigentliche“ nichtlineare Regression (6.12)

Nichtlineare Modell werden in R mit der Funktion `nls()` angepasst, beziehungsweise wird mit dieser Funktion die Kleinstquadratschätzung für alle möglichen nichtlinearen Zusammenhänge geschätzt. Hierzu muss man der Funktion mitteilen, um welches Modell es sich handelt und welche Werte als Startwerte verwendet werden sollen.

Seite 125, Beispiel 1:

```
a <- 70
b <- 44.4
c <- 0.3545
init <- c(alpha=a, beta=b, gamma=c)
f <- formula(weizen ~ alpha - (alpha - beta)*exp(-exp(gamma)*kalk))
nls.kalk <- nls(f, start=init)

-------------------------------------------------------------------------
```

```
nls.kalk
 Nonlinear regression model
   model:  weizen ~ alpha - (alpha - beta) * exp(-exp(gamma) * kalk)
    data:  parent.frame()
  alpha   beta  gamma
 72.433 44.181 -1.354
  residual sum-of-squares: 3.569

 Number of iterations to convergence: 5
 Achieved convergence tolerance: 3.777e-06
```

```
plot(kalk, weizen)
alpha <- 72.433
beta <- 44.181
gamma <- -1.354
x <- kalk
curve(alpha - (alpha-beta)*exp(-exp(gamma)*x), add=TRUE)
savePlot("Bild034.png", "png")
```

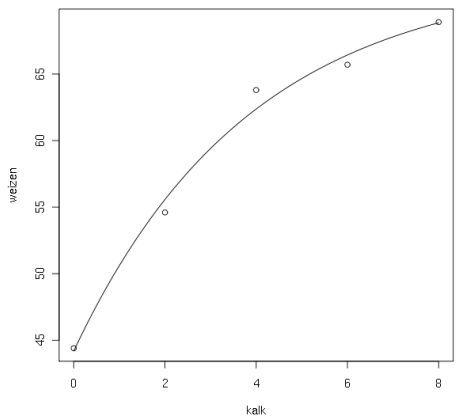


Abbildung 6.17: Schaubild der Weizenerträge in Abhängigkeit der Kalkgabe mit angepasster Mitscherlich-Funktion

`nls()` versucht nun anhand der Startwerte die eigentlichen Koeffizienten zu finden. Das Argument `start =` ist ein unerlässliches Argument, da der Algorithmus sonst nicht arbeiten kann. In dem Vektor, der bei `start =` angegeben wird, müssen die Namen der Variablen enthalten sein, die im Modell als Koeffizienten verwendet werden, außerdem muss eben der Startwert der entsprechenden Variablen, also der konkrete Wert des Parameters enthalten sein. Statt `alpha` hätte man in `init` auch `a` definieren können, dann müsste die Formel in `nls()` auch dieses `a` enthalten (`weizen ~ a - (a - beta)*exp(-(exp(gamma)*kalk)`.

6.11.1 Schätzen der Parameter (6.12.2)

In `nls()` kann man das Argument `algorithm =` definieren. Hier wird festgelegt, welches Verfahren verwendet wird, um die Koeffizienten zu schätzen. Standardmäßig ist hier Gauss-Newton eingestellt.

6.11.2 Startwerte (6.12.3)

Prinzipiell kann man die Startwerte selbst schätzen, allerdings gibt es in R einige sinnvolle Funktionen, die einem diese Aufgabe abnehmen. Diese Funktionen werden als `selfStart`-Modelle bezeichnet. Sie haben gemeinsam, dass sich ihr Namen aus `SS...` und der Bezeichnung für die Art der Regression zusammensetzt.

```
::SSasymp               Asymptotic Regression Model
::SSasympOff            Asymptotic Regression Model with an Offset
::SSasympOrig           Asymptotic Regression Model through the Origin
::SSbiexp               Biexponential model
::SSfol                 First-order Compartment Model
::SSfpl                 Four-parameter Logistic Model
::SSgompertz            Gompertz Growth Model
::SSlogis               Logistic Model
::SSmicmen              Michaelis-Menten Model
::SSweibull             Weibull growth curve model
```

Die durch diese Funktionen geschätzten Werte müssen an eine Zweite Funktion, `getInitial()` weitergegeben werden, welche die Startwerte für das entsprechende Modell ausgibt.

Seite 138, Beispiel 1:

Für dieses Beispiel, das eine exponentielles Modell darstellt, wird ein asymptotisches Regressionsmodell angenommen, die Startwerte müssen also mit `SSasymp` berechnet werden. Das Modell folgt der Funktion `Asym+(R0-Asym)*exp(-exp(lrc)*input)`.

```
gI <- getInitial(weizen ~ SSasymp(kalk, alpha, beta, gamma), df.kalk)
```

```
gI
 $alpha
 [1] 72.43267

 $beta
 [1] 44.18076

 $gamma
 [1] -1.354431
```

```
gI$alpha
 [1] 72.43267
```

```
gI$beta
 [1] 44.18076
```

```
gI$gamma
 [1] -1.354431
```

Der Funktion `SSasymp` sollte mitgeteilt werden, welchen Namen die Koeffizienten haben, da man sie so leicht ansteuern kann. Dies geschieht, indem man die der Reihe nach erscheinenden Koeffizienten mit den entsprechenden Namen versieht. `alpha`, `beta` und `gamma` könnte man auch weglassen, dann könnte man sie aber nicht über die `$`-Notation ansprechen.

```
alpha <- gI$alpha
beta <- gI$beta
gamma <- gI$gamma
init <- c(alpha=alpha, beta=beta, gamma=gamma)
nls.kalk <- nls(f, start=init)
```

```
nls.kalk
 Nonlinear regression model
   model:  weizen ~ alpha - (alpha - beta) * exp(-exp(gamma) * kalk)
    data:  parent.frame()
  alpha   beta  gamma
 72.433 44.181 -1.354
  residual sum-of-squares: 3.569

 Number of iterations to convergence: 1
 Achieved convergence tolerance: 5.214e-07
```

Setzt man die so ermittelten Werte in die Funktion ein, erhält man das gefragte Modell.

```
weizen = alpha - (alpha - beta)*exp(-exp(gamma)*kalk)
```

mit `alpha = 72.4326`, `beta = 44.1808` und `gamma = -1.3544` erhält man

$$\eta = 72.4326 - 28.2518e^{-0.2581x}$$

.

Seite 140, Beispiel 1:

Auch hier kann man wieder die Werte „aus dem Schaubild“ ablesen oder sich einer Funktion bedienen. Im logistischen Modell ist diese `SSlogis`. Die Koeffizienten sind folgende: `Asym` = die Asymptote, der sich das Modell annähert; `xmid` = der x-Wert an dem `Asym/2` für y gilt; `scal` = ein Faktor. Das Modell folgt der Funktion `Asym/(1+exp((xmid-input)/scal))`

```
df.schnitt <- data.frame(schnitt, zeit)
init <- c(alpha=70, beta=14.5, gamma=0.07)

-------------------------------------------------------------------------

getInitial(schnitt ~ SSlogis(zeit, Asym, xmid, scal), df.schnitt)
     Asym     xmid     scal
 72.46235 38.86747 14.84583

-------------------------------------------------------------------------

init.2 <- c(Asym=72.46235, xmid=38.86747, scal=14.84583)
```

Wie man sieht, hat man mit `SSlogis` nun 3 Startwerte bekommen, die erstmal nichts mit den „von Hand“ geschätzten Startwerten zu tun haben. Die liegt jedoch daran, dass der Funktion `SSlogis` die Formel
`schnitt ~ Asym/(1+exp((xmid-input)/scal))` zugrunde liegt. Diese Formel beschreibt das selbe Modell wie die Formel
`schnitt ~ alpha/(1+beta*exp(-gamma*zeit))`, also die im Biometrieskript zu findende Formel.

$$\eta = \frac{Asym}{1+e^{\frac{xmid-\theta}{scal}}} = \frac{Asym}{1+e^{\frac{xmid}{scal}-\frac{\theta}{scal}}} = \frac{Asym}{1+e^{\frac{xmid}{scal}}e^{-\frac{\theta}{scal}}}$$

, mit $\alpha = Asym$, $\beta = e^{\frac{xmid}{scal}}$ und $\gamma = \frac{1}{scal}$ erhält man also

$$\eta = \frac{\alpha}{1+\beta e^{-\gamma\theta}}$$

.

Nun kann man das nichtlineare Modell berechnen lassen, indem man die durch `SSlogis` gewonnen Werte derart umformt und in das Modell aus der Literatur einsetzt, oder indem man diese Werte nicht umformt und in das Modell von R einsetzt. Beides mal bekommt man Werte für die Koeffizienten, die das jeweilige Modell bestmöglich beschreiben.

```
init.3 <- c(alpha=72.46235, beta=38.86747/14.84583, gamma=1/14.84583)
f <- formula(schnitt ~ alpha/(1+beta*exp(-gamma*zeit)))
g <- formula(schnitt ~ Asym/(1+exp((xmid-zeit)/scal)))

-------------------------------------------------------------------------
```

```
nls(f, start=init)
 Nonlinear regression model
   model:  schnitt ~ alpha/(1 + beta * exp(-gamma * zeit))
    data:  parent.frame()
    alpha     beta    gamma
 72.46222 13.70934  0.06736
  residual sum-of-squares: 8.057

 Number of iterations to convergence: 4
 Achieved convergence tolerance: 5.118e-06

-------------------------------------------------------------------------

nls(g, start=init.2)
 Nonlinear regression model
   model:  schnitt ~ Asym/(1 + exp((xmid - zeit)/scal))
    data:  parent.frame()
  Asym  xmid  scal
 72.46 38.87 14.85
  residual sum-of-squares: 8.057

 Number of iterations to convergence: 1
 Achieved convergence tolerance: 1.756e-06

-------------------------------------------------------------------------

nls(f, start=init.3)
 Nonlinear regression model
   model:  schnitt ~ alpha/(1 + beta * exp(-gamma * zeit))
    data:  parent.frame()
    alpha     beta    gamma
 72.46223 13.70934  0.06736
  residual sum-of-squares: 8.057

 Number of iterations to convergence: 9
 Achieved convergence tolerance: 2.109e-06
```

Hier kann man erkennen, welche Auswirkungen es hat, benutzt man eigene Werte oder die von `SS...` geschätzten Werte in Kombination mit der dazugehörenden Funktion. Die schon durch die Funktion `SS...` geschätzten Werte verursachen viel weniger Aufwand in den Iterationsschritten und führen zu einer schnelleren Konvergenz.

6.11.3 Schließende Statistik (6.12.4)

Seite 144, Beispiel 1:

```
lm.kalk.0 <- lm(weizen ~ 1)

-------------------------------------------------------------------------
```

```
anova(nls.kalk, lm.kalk.0)
 Analysis of Variance Table

 Model 1: weizen ~ alpha - (alpha - beta) * exp(-gamma * kalk)
 Model 2: weizen ~ 1
   Res.Df Res.Sum Sq Df  Sum Sq F value   Pr(>F)
 1      2       3.57
 2      4     397.31 -2 -393.74  110.33 0.008982 **
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

6.11.4 Ein weiteres Beispiel (6.12.6)

Seite 148, Beispiel 1:

Dieses Beispiel wird hier nicht vorgerechnet, allerdings befinden sich die Daten dazu im Anhang bei allen anderen Daten.

6.12 Lineare Kontraste (6.13)

Das Paket `multcomp` stellt auch für die Berechnung der linearen Kontraste eine Funktion zur Verfügung. Dabei handelt es sich um `glht()`.

`glht()`: erstellt unter anderem multiple Mittelwertvergleiche.

`model =` : gibt das Modell an, dass die Koeffizienten zum Vergleich liefert.

`linfct =` : gibt an, welche Hypothese getestet werden soll.

`alternative = c("two.sided", "less", "greater")`:

`mcp()`: erzeugt den Input für `linfct =` indem verschiedene Formen der Eingabe der Kontraste in ein für `linfct =` verständliches Format umgewandelt werden.

Seite 153, Beispiel 1:

```
kon <- c(75, 67, 70, 75, 65, 71, 67, 67, 76, 68)
glu2 <- c(57, 58, 60, 59, 62, 60, 60, 57, 59, 61)
fru2 <- c(58, 61, 56, 58, 57, 56, 61, 60, 57, 58)
glufru <- c(58, 59, 58, 61, 57, 56, 58, 57, 57, 59)
sac2 <- c(62, 66, 65, 63, 64, 62, 65, 65, 62, 67)



df.zucker <- data.frame(Menge=c(kon, glu2, fru2, glufru, sac2),
```

```
 Art=c(rep("Kontrolle", 10), rep("Glucose", 10),  rep("Fructose", 10),
  rep("Glucose_Fructose", 10), rep("Saccharose", 10)))
lm.zucker <- lm(Menge ~ 0 + Art, df.zucker)
contr <- rbind(c(-.25, -.25, -.25, 1, -.25), c(-.5, -.5, 1, 0, 0))
rownames(contr) <- c("test1", "test2")
colnames(contr) <- c("Fructose", "Glucose", "Glucose_Fructose",
 "Kontrolle", "Saccharose")
glht.zucker <- glht(lm.zucker, linfct=mcp(Art = contr))
```

```
lm.zucker

 Call:
 lm(formula = Menge ~ Art - 1, data = df.zucker)

 Coefficients:
        ArtFructose           ArtGlucose  ArtGlucose_Fructose
               58.2                 59.3                 58.0
       ArtKontrolle        ArtSaccharose
               70.1                 64.1
```

```
contr
      Fructose Glucose Glucose_Fructose Kontrolle Saccharose
test1    -0.25   -0.25            -0.25         1      -0.25
test2    -0.50   -0.50             1.00         0       0.00
```

```
summary(glht.zucker)

          Simultaneous Tests for General Linear Hypotheses

 Multiple Comparisons of Means: User-defined Contrasts


 Fit: lm(formula = Menge ~ 0 + Art, data = df.zucker)

 Linear Hypotheses:
           Estimate Std. Error t value Pr(>|t|)
 test1 == 0  10.2000     0.8258  12.352   <1e-10 ***
 test2 == 0  -0.7500     0.9046  -0.829    0.651
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
 (Adjusted p values reported -- single-step method)
```

Hier sei zu erwähnen, dass das lineare Modell so spezifiziert wurde, dass das Gesamtmittelwert nicht berechnet wird. Nur wenn man den Gesamtmittel nicht mit ins Modell aufnimmt, können alle Behandlungsmittelwerte berechnet werden.

Um die linearen Kontraste testen zu können, gibt es mehrere Möglichkeiten den Input für `glht()` zu spezifizieren. Eine Methode besteht darin, eine Matrix zu erstellen, in der angegeben ist, welche Faktoren wie getestet werden sollen. Im ersten Test wollten wir wissen, wie sich die Kontrolle im Vergleich zu allen anderen Behandlungen verhält. Die Matrix muss die selbe Anzahl Spalten haben, wie es Koeffizienten im Modell gibt und die Nummer der Spalte in der Matrix codiert für den Koeffizienten, der an der entsprechenden Stelle im Modell auftaucht. Um den Test durchführen zu können, muss nach Addition der Koeffizienten (in diesem Fall Behandlungsmittelwerte) 0 herauskommen. Bei vier Behandlungen auf der einen Seite und einer auf der anderen Seite, bekommen die vier Behandlungen also jeweils eine `-0.25` und der einzelne Wert eine `1`, damit testen wir also, ob sich das Mittel aus den 4 Zuckerbehandlungen von der Kontrolle unterscheidet.

`linfct =` stellt die zu testende Hypothese bereit. Da es verschiedene Möglichkeiten gibt die Hypothese zu spezifizieren, wird eine zweite Funktion (`mcp()`) vorgeschaltet, die diese verschiedenen Spezifikationen vereinheitlicht und in einem für `linfct =` verständlichen Format ausgibt. In `mcp()` wird angegeben, welche Faktoren verglichen werden sollen, indem die Variable angegeben wird, die die Faktoren enthält. Nach einem '=' folgt der Teil der Spezifikation, der variabel gestaltet werden kann, in diesem Fall wurden die Vergleiche über eine Matrix spezifiziert. In der Matrix können beliebig viele Zeilen enthalten sein, jede Zeile stellt einen Vergleich dar.

Eine andere Möglichkeit ergibt sich, indem schon vorgefertigte Objekte verwendet werden. Eines dieser Objekte ist `"Tukey"`, hierbei handelt es sich um eine Matrix, in der alle paarweisen, linearen Kontraste enthalten sind, es werden also alle paarweisen Mittelwertvergleiche durchgeführt, womit es sich, bei beibehalten aller anderen Voreinstellungen um einen Tukey-HSD-Test handelt.

```
glht.zucker <- glht(lm.zucker, linfct=mcp(Art = "Tukey"))
```

```
summary(glht.zucker)
```

```
         Simultaneous Tests for General Linear Hypotheses

 Multiple Comparisons of Means: Tukey Contrasts


 Fit: lm(formula = Menge ~ 0 + Art, data = df.zucker)

 Linear Hypotheses:
                                  Estimate Std. Error t value Pr(>|t|)
 Glucose - Fructose == 0             1.100      1.045   1.053 0.829102
 Glucose_Fructose - Fructose == 0   -0.200      1.045  -0.191 0.999688
 Kontrolle - Fructose == 0          11.900      1.045  11.392  < 1e-04 ***
 Saccharose - Fructose == 0          5.900      1.045   5.648  < 1e-04 ***
 Glucose_Fructose - Glucose == 0    -1.300      1.045  -1.245 0.725610
 Kontrolle - Glucose == 0           10.800      1.045  10.339  < 1e-04 ***
 Saccharose - Glucose == 0           4.800      1.045   4.595 0.000318 ***
```

```
Kontrolle - Glucose_Fructose == 0    12.100      1.045  11.584  < 1e-04 ***
Saccharose - Glucose_Fructose == 0    6.100      1.045   5.840  < 1e-04 ***
Saccharose - Kontrolle == 0          -6.000      1.045  -5.744  < 1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

Weitere Möglichkeiten lassen sich der Hilfe zu `glht` entnehmen (`?glht`)

6.13 Einige Grundlagen der Matrizenrechnung

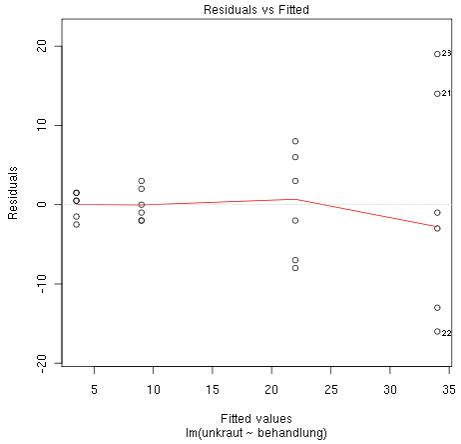
Die in diesem Kapitel angesprochenen Rechenoperationen werden bereits in Kapitel '↑ 5.7.1 R: Weiterführende Funktionen' angesprochen.

7 Transformation zur Erzielung der Voraussetzungen

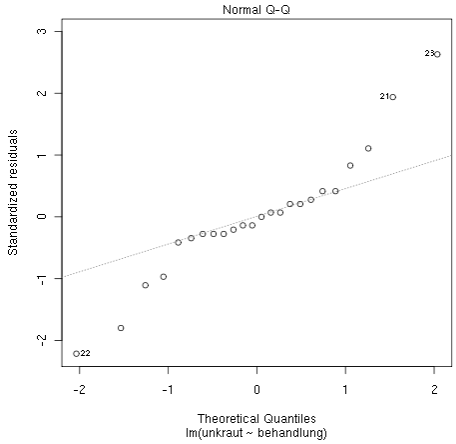
7.1 Beispiel einer einfachen Varianzanalyse

Seite 163, Beispiel 1:

```
a <- c(4, 5, 2, 5, 4, 1)
b <- c(8, 11, 9, 12, 7, 7)
c <- c(25, 28, 20, 15, 14, 30)
d <- c(33, 21, 48, 18, 53, 31)

------------------------------------------------------------------------

unkraut <- c(a, b, c, d)
behandlung <- gl(4, 6, labels=letters[1:4])
df.herb <- data.frame(unkraut, behandlung)
lm.herb <- lm(unkraut ~ behandlung, df.herb)

------------------------------------------------------------------------

anova(lm.herb)
 Analysis of Variance Table

 Response: unkraut
            Df Sum Sq Mean Sq F value    Pr(>F)
 behandlung  3 3361.1  1120.4  17.876 7.022e-06 ***
 Residuals  20 1253.5    62.7
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

------------------------------------------------------------------------

par(mfrow=c(2, 2))
plot(lm.herb)
unkraut.log <- log(unkraut)
lm.herb.log <- lm(unkraut.log ~ behandlung, df.herb)
plot(lm.herb.log)
```

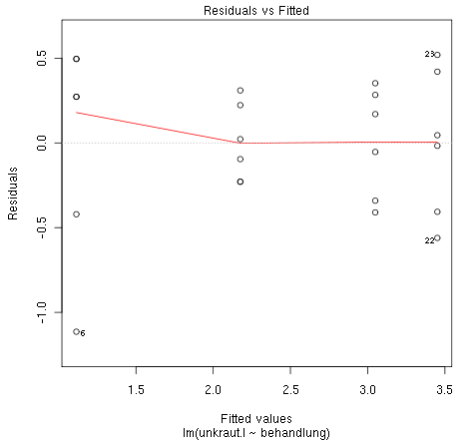


(a) Residuen gegen geschätzten Wert (Mittelwert)

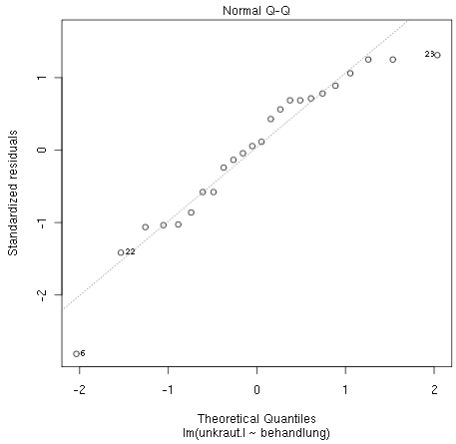


(b) Residuen gegen Normalscores (Q-Q-Plot)

Abbildung 7.1: Residuenanalyse zu den untransformierten Herbiziddaten



(a) Residuen gegen geschätzten Wert (Mittelwert)



(b) Residuen gegen Normalscores (Q-Q-Plot)

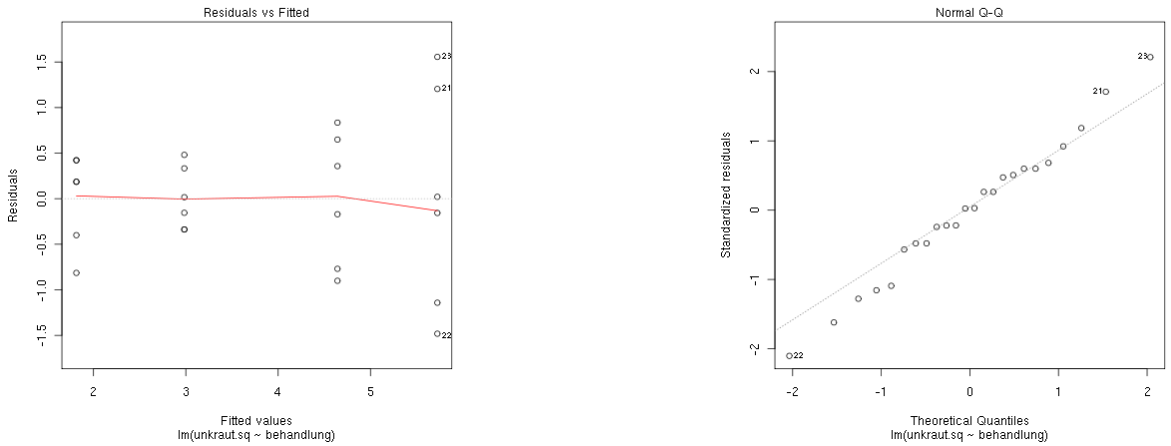
Abbildung 7.2: Residuenanalyse zu den log-transformierten Herbiziddaten

```
unkraut.sq <- sqrt(unkraut)
lm.herb.sq <- lm(unkraut.sq ~ behandlung, df.herb)
```

```
anova(lm.herb.log)
 Analysis of Variance Table
```

```
Response: unkraut.log
            Df  Sum Sq Mean Sq F value   Pr(>F)
behandlung   3 19.3285  6.4428  34.216 4.54e-08 ***
Residuals   20  3.7660  0.1883
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
plot(lm.herb.sq)
```



(a) Residuen gegen geschätzten Wert (Mittelwert)

(b) Residuen gegen Normalscores (Q-Q-Plot)

Abbildung 7.3: Residuenanalyse zu den wurzel-transformierten Herbiziddaten

Im Skript 'R für Vl Statistik' wurde schon erklärt, wie ein Mittelwertvergleich mit Hilfe einer Buchstabendarstellung aussehen kann. Dort wurde die Funktion `TukeyHSD()` eingeführt, die einen HSD-Test durchführt. Hier soll allerdings die Funktion `glht()` eingeführt werden, mit der unter anderem ein LSD-Test durchgeführt werden kann.

```
herb.glht <- glht(lm.herb, linfct=mcp(behandlung="Tukey"))
herb.glht <- summary(herb.glht, test=adjusted("none"))
pvals <- herb.glht$test$pvalues
names(pvals) <- gsub(" ", "", names(pvals))
```

```
multcompLetters(pvals)
   b    c    d    a
 "a"  "b"  "c"  "a"
```

Hier wurde nun also ein LSD-Test durchgeführt, der daran erkennbar ist, dass keine Adjustierung des multiplen Testrisikos durchgeführt wurde, das Argument `test =` steht

auf `adjusted("none")`. An dieser Stelle sei erwähnt, wie man zusätzliche Informationen zur Funktionsweise der Funktion `summary()` in Bezug auf die Klasse `glht()` bekommt.

```
methods(summary)
  [1] summary.aareg*          summary.aov             summary.aovlist
  [4] summary.aspell*         summary.cch*            summary.connection
  [7] summary.coxph*          summary.coxph.penal*    summary.data.frame
 [10] summary.Date            summary.default         summary.ecdf*
 [13] summary.factor          summary.glht*           summary.glm
 [16] summary.infl            summary.lm              summary.loess*
 [19] summary.manova          summary.matrix          summary.mlm
 [22] summary.nls*            summary.packageStatus*  summary.POSIXct
 [25] summary.POSIXlt         summary.ppr*            summary.prcomp*
 [28] summary.princomp*       summary.ratetable*      summary.stepfun
 [31] summary.stl*            summary.survfit*        summary.survreg*
 [34] summary.table           summary.tukeysmooth*

    Non-visible functions are asterisked

?summary.glht
```

Mit der Funktion `methods()` kann man sich, wie schon in Kapitel '↑ 5.7.1 R: Weiterführende Funktionen' erwähnt, anzeigen lassen, welche erweiterten Möglichkeiten es für eine Funktion gibt (, die eventuell in der ursprünglichen Dokumentation nicht erwähnt werden). Da wir Objekte der Klasse `glht()` betrachten wollen, liegt es nahe, die Funktion `summary.glht()` näher zu betrachten. Dort findet man dann auch eine Auflistung aller Möglichkeiten einer Adjustierung des multiplen Testrisikos.

7.2 Studentisierte Residuen im allgemeinen linearen Modell

Auf die Berechnung von Residuen und von studentisierten Residuen wurde bereits in Kapitel '↑ 6.6 Residuen, Modellvoraussetzungen und Ausreißer (6.7)' eingegangen.

7.3 Einige gängige (varianzstabilisierende) Transformationen

In Kapitel '↑ 6.3 Nichtlineare Regression durch Transformation der Variablen (6.4)' wurden einige einfache Transformationen vorgestellt, die Daten so transformieren sollen, dass sie als linearer Zusammenhang dargestellt werden können.

Im Endeffekt steckt hinter der Transformation zur Stabilisierung der Varianz nicht viel mehr, man transfomiert die Daten und betrachtet dann die Residuen in der Residuenanalyse.

8 Versuchsanlagen

8.1 Randomisation (8.2)

Hier wird dargestellt, wie die Randomisation in R durchgeführt werden kann. Dabei spielt die Funktion `order()` die wichtigste Rolle.

```
Sorte <- gl(5, 4, labels=letters[1:5])
Zufallszahlen <- rnorm(20)
```

```
Zufallszahlen
  [1]  1.0744410  1.8956548 -0.6029973 -0.3908678 -0.4162220 -0.3756574
  [7] -0.3666309 -0.2956775  1.4418204 -0.6975383 -0.3881675  0.6525365
 [13]  1.1247724 -0.7721108 -0.5080862  0.5236206  1.0177542 -0.2511646
 [19] -1.4299934  1.7091210
```

```
df <- data.frame(Zufallszahlen, Sorte)
```

```
df
    Zufallszahlen Sorte
 1     0.48934096     a
 2    -0.77891030     a
 3     1.74355935     a
 4    -0.07838729     a
 5    -0.97555379     b
 6     0.07065982     b
 7    -1.51859953     b
 8     0.86377903     b
 9     0.50156839     c
 10   -0.35478133     c
 11   -0.48842889     c
 12    0.93629395     c
 13   -1.06240839     d
 14   -0.98382087     d
 15    0.42424788     d
 16   -0.45131348     d
 17    0.92508480     e
 18   -0.19862081     e
 19    1.19485102     e
```

```
 20    0.49554471     e
```

```
df <- data.frame(Sorte, Zufallszahlen)
df <- df[order(df$Zufallszahlen) ,]
```

```
df
    Sorte Zufallszahlen
 7      b   -1.51859953
 13     d   -1.06240839
 14     d   -0.98382087
 5      b   -0.97555379
 2      a   -0.77891030
 11     c   -0.48842889
 16     d   -0.45131348
 10     c   -0.35478133
 18     e   -0.19862081
 4      a   -0.07838729
 6      b    0.07065982
 15     d    0.42424788
 1      a    0.48934096
 20     e    0.49554471
 9      c    0.50156839
 8      b    0.86377903
 17     e    0.92508480
 12     c    0.93629395
 19     e    1.19485102
 3      a    1.74355935
```

```
df <- cbind(df, Parzelle=c(1:20))
```

```
df
    Zufallszahlen Sorte Parzelle
 7    -1.51859953     b        1
 13   -1.06240839     d        2
 14   -0.98382087     d        3
 5    -0.97555379     b        4
 2    -0.77891030     a        5
 11   -0.48842889     c        6
 16   -0.45131348     d        7
 10   -0.35478133     c        8
 18   -0.19862081     e        9
 4    -0.07838729     a       10
 6     0.07065982     b       11
 15    0.42424788     d       12
```

```
1     0.48934096     a       13
20    0.49554471     e       14
9     0.50156839     c       15
8     0.86377903     b       16
17    0.92508480     e       17
12    0.93629395     c       18
19    1.19485102     e       19
3     1.74355935     a       20
```

Aus dieser „Tabelle“ kann man nun ablesen, wie der zuvor erstellte Plan ausgefüllt werden kann.

Zu aller erst erstellt man ein Dataframe und spezifiziert die Daten. Dabei ist zu beachten, dass die Daten im Biometriskript gleichverteilt sind und hier normalverteilte Zahlen verwendet wurden. Da man lediglich eine zufällige Reihenfolge erstellen will, nach der man sortieren kann, spielt dies jedoch keine Rolle (man könnte genauso gut zufällig Buchstaben ausgeben lassen, die man dann nach dem Alphabet sortiert). Im zweiten Schritt wird dieser Dataframe sortiert und zwar nach den Zufallszahlen.

Die eckige Klammer dient der Referenzierung eines Objektes. Referenzieren heißt so viel wie verweisen (engl. (to) refer) und bedeutet in diesem Fall, dass `df` mit der Funktion `order(df$Zufallszahlen)` referenziert werden soll. Da `order(df$Zufallszahlen)` aufgrund der Zufallszahlen zu einer anderen Anordnung der Sorten führt, wird durch diese Funktion `df` so umsortiert, wie in der Funktion angegeben. Hinter der Funktion `order(df$Zufallszahlen)` ist ein `,` zu sehen. Dies kommt daher, da `df` ein zweidimensionales Objekt ist und damit angegeben werden muss, was mit der zweiten Dimension, also den Spalten passiert. Gibt man garnichts an, lässt also `,` weg, bekommt man eine Fehlermeldung. Mit `,` gibt man an, dass mit den Reihen nichts passieren soll.

Die Funktion `cbind()` fügt an ein Objekt wie eine Matrix oder einen Dataframe einen Vektor hinzu, der genauso lang sein muss, wie die Matrix oder das Dataframe Zeilen hat.

Bei Dataframes mit mehreren Spalten muss nun die Spalte an der ersten Stelle von `order()` stehen, nach der als erstes sortiert werden soll; die Spalte muss an der zweiten Stelle stehen, nach der als zweites sortiert werden soll, et cetera.

8.2 Randomisierte vollständige Blockanlage (8.4)

Hier kommt das oben erwähnte nun zur Anwendung.

```
Block <- gl(4, 6)
Behandlung <- rep(1:6, 4)
Zufallszahlen <- rnorm(length(Behandlung))
df <- data.frame(Block, Zufallszahlen, Behandlung)

------------------------------------------------------------------------
```

```
df
    Block Zufallszahlen Behandlung
 1      1    0.22727141          1
 2      1    0.52314488          2
 3      1   -0.74806318          3
 4      1   -0.18760189          4
 5      1   -0.46797107          5
 6      1    0.49590279          6
 7      2   -0.07968570          1
 8      2   -1.04677247          2
 9      2   -2.65703922          3
 10     2    0.17389554          4
 11     2    1.33152686          5
 12     2    0.14751514          6
 13     3    1.57451621          1
 14     3    0.18375638          2
 15     3    0.28610596          3
 16     3   -0.43880756          4
 17     3   -1.00218833          5
 18     3    0.61009149          6
 19     4    0.10458321          1
 20     4    0.92280537          2
 21     4   -0.43525220          3
 22     4    1.98307320          4
 23     4   -0.35945913          5
 24     4    0.06585868          6
```

```
df <- df[order(df$Block, df$Zufallszahlen) ,]
df <- cbind(df, Spalte=c(1:6))
```

```
df
    Block Zufallszahlen Behandlung Spalte
 3      1   -0.74806318          3      1
 5      1   -0.46797107          5      2
 4      1   -0.18760189          4      3
 1      1    0.22727141          1      4
 6      1    0.49590279          6      5
 2      1    0.52314488          2      6
 9      2   -2.65703922          3      1
 8      2   -1.04677247          2      2
 7      2   -0.07968570          1      3
 12     2    0.14751514          6      4
 10     2    0.17389554          4      5
 11     2    1.33152686          5      6
 17     3   -1.00218833          5      1
 16     3   -0.43880756          4      2
 14     3    0.18375638          2      3
 15     3    0.28610596          3      4
```

```
 18     3    0.61009149          6      5
 13     3    1.57451621          1      6
 21     4   -0.43525220          3      1
 23     4   -0.35945913          5      2
 24     4    0.06585868          6      3
 19     4    0.10458321          1      4
 20     4    0.92280537          2      5
 22     4    1.98307320          4      6
```

Hier werden die Daten nun anhand zweier Spalten des Dataframes sortiert. Dabei findet die Sortierung so statt, dass zuerst nach `df$Block` sortiert wird und dann nach `df$Zufallszahlen`. Damit ist sichergestellt, dass die Behandlungen über die Blöcke randomisiert sind und nicht über das gesamte Feld.

Außerdem gibt es zur Randomisation einer Blockanlage das Paket blockrand mit den Funktionen `blockrand()` und `plot.blockrand()`. Muss man viele Ransomisationen durchführen, bietet es sich an, dieses Paket genauer unter die Lupe zu nehmen.

8.3 Lateinisches Quadrat (8.5)

```
# 1. Erstellen des Grundplans
m <- scan(what="character")
1: a b c d b a d c c d a b d c b a
17:
 Read 16 items
m <- matrix(m,4,4)

------------------------------------------------------------------------

m
      [,1] [,2] [,3] [,4]
 [1,] "a"  "b"  "c"  "d"
 [2,] "b"  "a"  "d"  "c"
 [3,] "c"  "d"  "a"  "b"
 [4,] "d"  "c"  "b"  "a"

------------------------------------------------------------------------

# 2. Randomisieren der Zeilen
m <- m[order(rnorm(4)),]

------------------------------------------------------------------------

# 3. Randomisieren der Spalten
m <- m[,order(rnorm(4))]

------------------------------------------------------------------------

m
```

```
     [,1] [,2] [,3] [,4]
[1,] "d"  "a"  "b"  "c"
[2,] "a"  "d"  "c"  "b"
[3,] "c"  "b"  "a"  "d"
[4,] "b"  "c"  "d"  "a"
```

Hier wurde mit der Funktion `scan()` die Reihenfolge der Felder des lateinischen Quadrates eingelesen, die Felder werden Horizontal von links nach recht eingelesen, am Zeilenende wird in die nächste Zeile gesprungen. Beim randomisieren der Zeilen und Spalten werden die Werte der Matrix zufällig in den Zeilen und Spalten gegeneinander verschoben.

8.4 Auswertung einer Blockanlage (8.7)

8.4.1 Varianzanalyse einer Blockanlage (8.7.1)

Seite 198, Beispiel 1:

```
Saatstärke <- c(25, 25, 25, 25, 50, 50, 50, 50, 75, 75, 75, 75, 100, 100,
 100, 100, 125, 125, 125, 125, 150, 150, 150, 150)
Block <- c(1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1,
 2, 3, 4)
IR8 <- c(5113, 5398, 5307, 4678, 5346, 5952, 4719, 4264, 5272, 5713, 5483,
 4749, 5164, 4831, 4986, 4410, 4804, 4848, 4432, 4748, 5254, 4542, 4919,
 4098)
```

```
df.reis <- data.frame(Saatstärke, Block, IR8)
```

```
df.reis
    Saatstärke Block  IR8
 1          25     1 5113
 2          25     2 5398
 3          25     3 5307
 4          25     4 4678
 5          50     1 5346
 6          50     2 5952
 7          50     3 4719
 8          50     4 4264
 9          75     1 5272
 10         75     2 5713
 11         75     3 5483
 12         75     4 4749
 13        100     1 5164
 14        100     2 4831
```

```
 15        100     3 4986
 16        100     4 4410
 17        125     1 4804
 18        125     2 4848
 19        125     3 4432
 20        125     4 4748
 21        150     1 5254
 22        150     2 4542
 23        150     3 4919
 24        150     4 4098
```

```
lm.reis <- lm(IR8 ~ factor(Block) + factor(Saatstärke))
```

```
anova(lm.reis)
 Analysis of Variance Table

 Response: IR8
                    Df  Sum Sq Mean Sq F value   Pr(>F)
 factor(Block)       3 1944361  648120  5.8622 0.007416 **
 factor(Saatstärke)  5 1198331  239666  2.1678 0.112809
 Residuals          15 1658376  110558
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Hier unterscheidet sich die Darstellung wieder von der im Biometrieskript. Zur Erfassung in R sind die Daten nicht in der typischen Tabelle erfasst, sondern in einem Dataframe. In der Verarbeitung mit `lm()` werden die Variablen `Saatstärke` und `Block` als Faktoren angegeben, da hier der qualitative Wert von Interesse ist, also ob die verschiedenen Stufen der Variablen einen Effekt haben. Dass jedoch auch die quantitativen Werte dieser Variablen einen Effekt haben können, zeigt Kapitel ’↑ 8.6 Regression in einer Blockanlage (8.9)’.

An dieser Stelle soll auch erwähnt werden, dass die Zahl der Stufen der qualitativen Vektoren als die r- und t-Werte aus dem Biometrieskript angesehen werden können, die wichtig für die Berechnung der Freiheitsgrade sind. Sollte sich also, nach Auswerten mit `anova()`, ein Fehler in den Freiheitsgraden eingeschlichen haben, kann das daran liegen, dass die Stufen der Variablen nicht so sind, wie sie sein sollten.

Seite 199, Beispiel 1:

```
IR8.2 <- c(NA, 5398, 5307, 4678, 5346, 5952, 4719, 4264, 5272, 5713, 5483,
 4749, 5164, 4831, 4986, 4410, 4804, 4848, NA, 4748, 5254, 4542, 4919,
 4098)
```

```
df.reis <- data.frame(Saatstärke, Block, IR8.2)
```

```
df.reis
    Saatstärke Block IR8.2
 1          25     1    NA
 2          25     2  5398
 3          25     3  5307
 4          25     4  4678
 5          50     1  5346
 6          50     2  5952
 7          50     3  4719
 8          50     4  4264
 9          75     1  5272
 10         75     2  5713
 11         75     3  5483
 12         75     4  4749
 13        100     1  5164
 14        100     2  4831
 15        100     3  4986
 16        100     4  4410
 17        125     1  4804
 18        125     2  4848
 19        125     3    NA
 20        125     4  4748
 21        150     1  5254
 22        150     2  4542
 23        150     3  4919
 24        150     4  4098
```

```
lm.reis <- lm(IR8.2 ~ factor(Block) + factor(Saatstärke))
```

```
anova(lm.reis)
 Analysis of Variance Table

 Response: IR8.2
                   Df  Sum Sq Mean Sq F value   Pr(>F)
 factor(Block)      3 1991588  663863  5.8927 0.009106 **
 factor(Saatstärke)  5 1036665  207333  1.8404 0.174002
 Residuals          13 1464571  112659
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Sollten sich fehlende Werte in den Daten befinden, sollten diese durch `NA` ersetzt werden, falls nicht schon passiert. Dann haben die fehlenden Werte keine Auswirkung auf die

Auswertbarkeit mit `lm()`.

8.4.2 Mittelwertvergleiche einer Blockanlage (8.7.2)

Auch hier kann man sich wieder die Funktion `multcompLetters()` zunutze machen.

```
aov <- aov(IR8 ~factor(Block) + factor(Saatstärke))
t <- TukeyHSD(aov)
x <- t$`factor(Saatstärke)`[,4]

-------------------------------------------------------------------------

multcompLetters(x)
 $Letters
  50  75 100 125 150  25
 "a" "a" "a" "a" "a" "a"

 $LetterMatrix
        a
 50  TRUE
 75  TRUE
 100 TRUE
 125 TRUE
 150 TRUE
 25  TRUE
```

Man sieht also, dass keine signifikanten Unterschiede in den Saatstärke-Mittelwerten vorliegen. Der Tukey-Test liefert auch die Ergebnisse für die Mittelwerte der Blöcke, darauf wird in Kapitel '↑ 10 Zweifaktorielle Varianzanalyse - Wechselwirkungen' tiefer eingegangen.

8.5 Auswertung eines Lateinischen Quadrats (8.8)

8.5.1 Varianzanalyse eines Lateinischen Quadrates (8.8.1)

Seite 210, Beispiel 1:

```
Tag <- factor(rep(c(1, 2, 3, 4, 5), 5))
Tageszeit <- gl(5, 5, labels=c("08:30", "10:00", "11:30", "14:00",
 "15:30"))
Betrieb <- factor(c("A", "B", "C", "D", "E", "D", "C", "E", "B", "A", "C",
 "A", "D", "E", "B", "B", "E", "A", "C", "D", "E", "D", "B", "A", "C"))
Bakterienzahl.log <- c(1.9, 1.2, 0.7, 2.2, 2.3, 2.3, 2.0, 0.6, 2.6, 2.3,
 2.1, 1.5, 1.7, 1.1, 3.0, 2.9, 1.1, 1.2, 1.8, 2.6, 1.8, 2.1, 2.0, 2.4,
 2.5)

-------------------------------------------------------------------------
```

```
df.milch <- data.frame(Tag, Tageszeit, Betrieb, Bakterienzahl.log)
```

```
df.milch
    Tag Tageszeit Betrieb Bakterienzahl.log
 1    1     08:30       A               1.9
 2    2     08:30       B               1.2
 3    3     08:30       C               0.7
 4    4     08:30       D               2.2
 5    5     08:30       E               2.3
 6    1     10:00       D               2.3
 7    2     10:00       C               2.0
 8    3     10:00       E               0.6
 9    4     10:00       B               2.6
 10   5     10:00       A               2.3
 11   1     11:30       C               2.1
 12   2     11:30       A               1.5
 13   3     11:30       D               1.7
 14   4     11:30       E               1.1
 15   5     11:30       B               3.0
 16   1     14:00       B               2.9
 17   2     14:00       E               1.1
 18   3     14:00       A               1.2
 19   4     14:00       C               1.8
 20   5     14:00       D               2.6
 21   1     15:30       E               1.8
 22   2     15:30       D               2.1
 23   3     15:30       B               2.0
 24   4     15:30       A               2.4
 25   5     15:30       C               2.5
```

```
lm.milch <- lm(Bakterienzahl.log ~ Tageszeit + Tag + Betrieb, df.milch)
```

```
anova(lm.milch)
 Analysis of Variance Table

 Response: Bakterienzahl.log
           Df Sum Sq Mean Sq F value    Pr(>F)
 Tageszeit  4 0.6416  0.1604  1.3434 0.3100851
 Tag        4 5.2536  1.3134 11.0000 0.0005532 ***
 Betrieb    4 2.7456  0.6864  5.7487 0.0080354 **
 Residuals 12 1.4328  0.1194
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Bei einem Lateinischen Quadrat wird lediglich ein weiterer Faktor in das Modell hinzugenommen.

Seite 211, Beispiel 1:

```
Bakterienzahl.2.log <- c(1.9, 1.2, NA, 2.2, 2.3, 2.3, 2.0, 0.6, 2.6, 2.3,
 2.1, 1.5, 1.7, 1.1, 3.0, 2.9, 1.1, 1.2, 1.8, NA, 1.8, 2.1, 2.0, 2.4, 2.5)

------------------------------------------------------------------------

df.milch <- data.frame(Tageszeit, Tag, Betrieb, Bakterienzahl.2.log)
lm.milch <- lm(Bakterienzahl.2.log ~ Tageszeit + Tag + Betrieb,
 df.milch)

------------------------------------------------------------------------

anova(lm.milch)
 Analysis of Variance Table

 Response: Bakterienzahl.2.log
           Df Sum Sq Mean Sq F value   Pr(>F)
 Tageszeit  4 0.4128  0.1032  0.7986 0.552797
 Tag        4 3.6582  0.9145  7.0769 0.005691 **
 Betrieb    4 2.7516  0.6879  5.3230 0.014672 *
 Residuals 10 1.2923  0.1292
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

8.5.2 Mittelwertvergleiche in einem Lateinischen Quadrat (8.8.2)

```
aov <- aov(Bakterienzahl.log ~ Tageszeit + Tag + Betrieb)
t <- TukeyHSD(aov)
x <- t$Betrieb[,4]

------------------------------------------------------------------------

multcompLetters(x)
    B    C    D    E    A
  "a" "ab"  "a"  "b" "ab"
```

8.6 Regression in einer Blockanlage (8.9)

Seite 217, Beispiel 1:

Die Variable Saatstärke der Reisdaten kann, wie in Kapitel ’↑ 8.4.1 Varianzanalyse einer Blockanlage (8.7.1)’, als qualitative Variable aufgefasst werden, da es sich um eine nu-

merische Variable handelt, kann diese aber auch quantitativ interpretiert werden. Dann ist eine Regression möglich.

```
lm.reis <- lm(IR8 ~ factor(Block) + Saatstärke + factor(Saatstärke))
```

```
anova(lm.reis)
 Analysis of Variance Table

 Response: IR8
                    Df  Sum Sq Mean Sq F value   Pr(>F)
 factor(Block)       3 1944361  648120  5.8622 0.007416 **
 Saatstärke          1  760035  760035  6.8745 0.019240 *
 factor(Saatstärke)  4  438296  109574  0.9911 0.442293
 Residuals          15 1658376  110558
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Hier wird unter anderem überprüft, ob der Effekt auf den Ebenen der Variable `Saatstärke` einen signifikanten Einfluss hat, also zu einem von der Linearität abweichenden Schaubild führt. Dies wurde schon in Kapitel ’↑ 6.5 Test auf Linearität (6.6)’ geschildert, allerdings ohne zusätzliche Variable (`Block`).

Da der Faktor der Variable `Saatstärke` (Lack-of-fit) also keinen Einfluss hat, kann man das Modell so umgestalten, dass der Faktor nicht mehr enthalten ist, also das reduzierte lineare Modell verwendet wird.

```
lm.reis <- lm(IR8 ~ factor(Block) + Saatstärke)
```

```
anova(lm.reis)
 Analysis of Variance Table

 Response: IR8
               Df  Sum Sq Mean Sq F value   Pr(>F)
 factor(Block)  3 1944361  648120  5.8733 0.005161 **
 Saatstärke     1  760035  760035  6.8874 0.016692 *
 Residuals     19 2096672  110351
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
lm.reis

 Call:
 lm(formula = IR8 ~ factor(Block) + Saatstärke)

 Coefficients:
```

```
    (Intercept)  factor(Block)2  factor(Block)3  factor(Block)4
       5523.533          55.167        -184.500        -667.667
     Saatstärke
         -4.168
```

```
(55.167-184.500-667.667+0)/4
 [1] -199.25
```

```
5523.533-199.25
 [1] 5324.283
```

`lm()` führt die Restriktion anders durch, als dies im Biometrieskript getan wurde, hier wird der erste Effekt gleich Null gesetzt und nicht, wie bei SAS, der Letzte. Dadurch ergeben sich andere Werte. Wie die anschließende Rechnung zeigt, kommt im Endergebnis jedoch das Selbe heraus.

Seite 220, Beispiel 1:

```
Stickstoff <- c(0, 0, 0, 0, 0, 25, 25, 25, 25, 25, 50, 50, 50, 50, 50,
 100, 100, 100, 100, 100, 150, 150, 150, 150, 150, 200, 200, 200, 200,
 200)
Block <- c(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1,
 2, 3, 4, 5, 1, 2, 3, 4, 5)
Zuckerrohr <- c(89.49, 54.56, 74.33, 78.20, 61.51, 108.78, 102.01, 105.04,
 105.23, 106.52, 136.28, 129.51, 132.54, 132.73, 134.02, 157.63, 167.39,
 155.39, 146.85, 155.81, 185.96, 176.66, 178.53, 195.34, 185.56, 195.09,
 190.43, 183.52, 180.99, 205.69)
```

```
lm.zucker <- lm(Zuckerrohr ~ factor(Block) + Stickstoff +
 I(Stickstoff**2))
```

```
anova(lm.zucker)
 Analysis of Variance Table

 Response: Zuckerrohr
                Df Sum Sq Mean Sq  F value    Pr(>F)
 factor(Block)   4    275      69   0.8484    0.5092
 Stickstoff      1  49465   49465 609.6003 < 2.2e-16 ***
 I(Stickstoff^2) 1   3912    3912  48.2155  4.45e-07 ***
 Residuals      23   1866      81
 ---
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

`Stickstoff` ist die Prädikatorvariable, muss also für dieses Polynom quadriert werden.

9 Zweistufige Stichproben

9.1 Modell und Auswertung

Bei zweistufigen Stichproben erfolgt die Modellbildung genauso, wie bei anderen Stichproben.

Seite 225, Beispiel 1:

```
Calcium <- c(3.28, 3.09, 3.03, 3.03, 3.52, 3.48, 3.38, 3.38, 2.88, 2.80,
 2.81, 2.76, 3.34, 3.38, 3.23, 3.26)
Blatt <- c(1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4)
Messung <- c(1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)

------------------------------------------------------------------------

df.calcium <- data.frame(Blatt=factor(Blatt), Messung=factor(Messung), Calcium)
lm.calcium <- lm(Calcium ~ Blatt, df.calcium)

------------------------------------------------------------------------

lm.calcium

 Call:
 lm(formula = Calcium ~ Blatt, data = df.calcium)

 Coefficients:
 (Intercept)       Blatt2       Blatt3       Blatt4
      3.1075       0.3325      -0.2950       0.1950

------------------------------------------------------------------------

anova(lm.calcium)
 Analysis of Variance Table

 Response: Calcium
           Df  Sum Sq  Mean Sq F value   Pr(>F)
 Blatt      3 0.88837 0.296123  44.853 8.52e-07 ***
 Residuals 12 0.07923 0.006602
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In dieses Modell wurde lediglich der Effekt der Blätter aufgenommen, da der Effekt der Messung nicht relevant sein wird, Messungen finden unter standardisierten Bedingungen

statt um diesen zusätzlichen Effekt auszuschließen.

```
confint(lm.calcium)
                       2.5 %     97.5 %
 (Intercept)      3.01898224  3.1960178
 factor(Blatt)2   0.20731698  0.4576830
 factor(Blatt)3 -0.42018302 -0.1698170
 factor(Blatt)4   0.06981698  0.3201830
```

Mit `confint()` können nun die Vertrauensintervalle des Modells getestet werden. Hier werden allerdings die Vertrauensintervalle für den y-Achsenabschnitt `(Intercept)` und die Steigung der Faktoren berechnet. Daran sind wir hier nicht interessiert.

Da R mit dieser Methode keine Funktion zur Berechnung der Vertrauensintervalle liefert, habe ich diese Funktion wieder selbst geschrieben.

(↑Formelübersicht)

`ci.two.stage()`: Vertrauensintervall für zweistufige Stichproben

`x =` : Datenvektor.

`y =` : erster Vektor mit Faktoren, enthält die Beobachtungseinheiten.

`z =` : zweiter Vektor mit Faktoren, enthält die Messwiederholungen.

`alpha =` : Irrtumswahrscheinlichkeit zur Berechnung der Quantile der Standardnormalverteilung, default=.05.

```
ci.two.stage(Calcium, factor(Blatt), factor(Messung))
 [1] 2.732676 3.598574
```

Da der Blatteffekt in diesem Zusammenhang zufällig ist, kann man dieses Modell als ↑gemischtes Modell oder als zufälliges Modell (da keine festen Effekte vorhanden sind) betrachten. Damit ergibt sich die Möglichkeit das Modell mit der Funktion `lme()` aus zu werten. Hierzu muss man das Paket `nlme` laden.

```
lme.calcium <- lme(Calcium ~ 1, data=df.calcium, random= ~ 1|Blatt)
```

```
lme.calcium
 Linear mixed-effects model fit by REML
   Data: df.calcium
   Log-restricted-likelihood: 9.277319
   Fixed: Calcium ~ 1
 (Intercept)
    3.165625
```

```
 Random effects:
  Formula: ~1 | Blatt
         (Intercept)  Residual
 StdDev:   0.2690357 0.0812532

 Number of Observations: 16
 Number of Groups: 4
```

```
anova(lme.calcium)
             numDF denDF  F-value p-value
 (Intercept)     1    12 541.4606  <.0001
```

```
intervals(lme.calcium)
 Approximate 95% confidence intervals

  Fixed effects:
                lower     est.    upper
 (Intercept) 2.869213 3.165625 3.462037
 attr(,"label")
 [1] "Fixed effects:"

  Random Effects:
   Level: Blatt
                      lower      est.     upper
 sd((Intercept)) 0.1186757 0.2690357 0.6098991

  Within-group standard error:
     lower      est.     upper
 0.0544615 0.0812532 0.1212248
```

Bei der Spezifizierung gemischter Modelle muss man beachten, dass sowohl ein Teil mit festen Effekten, als auch ein Teil mit zufälligen Effekten im Modell enthalten ist. Diese müssen getrennt angegeben werden, da verschiedene Methoden zur Berechnung verwendet werden. Bei der Funktion `lm()` gibt es das Argument `formula =` , in dem man die Modellformel spezifiziert. Da bei gemischten Modellen die zufälligen Effekte zu den festen Effekten unterschiedlich berechnet werden, existiert hier zusätzlich das Argument `random =` , in dem man die zufälligen Effekte angibt. Auch hier bedient man sich der ~-Notation um den Zusammenhang darzustellen. Die Prädikatorvariable (in diesem Beispiel ist das `Calcium`) kann hier jedoch weg gelassen werden. Weitere Informationen zur Spezifizierung dieser Terme kann man der Hilfe entnehmen (`?lme -> random =`)

Weiterhin gibt es die Funktion `lmer()`, mit der sich ebenfalls gemischte Modelle berechnen lassen. Im Vergleich zu `lme()` ergeben sich jedoch einige Nachteile, allerdings existieren auch einige Vorteile. Die Funktion befindet sich im Paket `lme4`. Einer

der entscheidenden Nachteile ist der, dass keine sehr gut ausgearbeitete/verbreitete Dokumentation dazu existiert und damit wohl nicht so viel Erfahrung darüber existiert. Die Arbeitsgruppe um Herrn Piepho ist jedoch schon seit einiger Zeit mit gemischten Modellen beschäftigt, so kann man hier sicherlich nach einführenden/weiterführenden Informationen fragen.

Mit der Funktion `intervals()` lassen sich die Vertrauensintervalle zu den Parametern einiger Objekten berechnen. Bei `Fixed effects:` sind die Parameter aufgelistet, die den festen Effekten zuzuordnen sind. Hier ist das der Gesamtmittelwert μ. Wie man sieht, weicht dieses Vertrauensintervall leicht von dem im Biometrie-Skript berechneten ab. Dies kommt daher, dass lme() für die Berechnung des Intervalls offenbar die Quantile der t-Verteilung mit t(r-1)=12 Freiheitsgraden verwendet, was den Rest-Freiheitsgraden entspricht, während im Biometrie-Skript die t-Verteilung mit t-1=3 Freiheitsgraden verwendet wird. Letzeres Verfahren ist exakt, während das von lme() berechnete Intervall nur näherungsweise gültig ist. Das Paket hat leider keine Option zur Berechnung adäquater Fehlerfreiheitsgrade. Die Berechnung von Freiheitsgraden in gemischten Modellen ist ein komplexes Problem. Die beste bekannte Methode ist die Kenward-Roger Methode, die aber in lme() nicht verfügbar ist (wohl aber in der SAS Prozedur MIXED). Der Gesamtmittelwert ist jedoch genau der selbe.

9.2 Optimale Allokation

Für eine optimale Allokation muss nun lediglich der Output der Varianzanalyse in die Formel

$$r = \sqrt{\frac{k_1\sigma^2}{k_2\sigma_b^2}}$$

eingesetzt werden.

Um den Output verwenden zu können, muss man auf die Struktur des anova-Objektes zugreifen. Man verwendet die Funktion `str()` um sich einen Überblick über das Objekt zu verschaffen.

```
aov <- anova(lm.calcium)
```

```
str(aov)
 Classes 'anova' and 'data.frame':       2 obs. of  5 variables:
  $ Df     : int  3 12
  $ Sum Sq : num  0.8884 0.0792
  $ Mean Sq: num  0.2961 0.0066
  $ F value: num  44.9 NA
  $ Pr(>F) : num  8.52e-07 NA
  - attr(*, "heading")= chr  "Analysis of Variance Table\n" "Response:
  Calcium"
```

```
mq <- aov$"Mean Sq"
```

```
mq
 [1] 0.296122917 0.006602083
```

```
mq_beh <- mq[1]
mq_feh <- mq[2]
sigmadach2 <- mq_feh
sigmadach2b <- (mq_beh-mq_feh)/r
```

```
sigmadach2b
 [1] 0.07238021
```

```
sigmadach2
 [1] 0.006602083
```

Die hier durchgeführte 'optimale Allokation' basiert auf den Calcium-Daten und nicht auf dem Beispiel in Kapitel 9.2. Die Berechnung von `sigmadach2` und `sigmadach2b` erfolgt nach dem Kochbuchkasten des Kapitels 9.1.

Bisher wurden Elemente der Struktur verschiedener Objekte nicht in `""` gesetzt, dies ist auch nicht notwendig, wenn das Element kein Leerzeichen enthält. Befindet sich ein Leerzeichen im Elementname, ist dies jedoch notwendig.

Mit der Funktion `levels()` kann man auf die Stufen eines Faktors zugreifen und entweder abrufen welche Levels vorhanden sind oder diese sogar verändern.

10 Zweifaktorielle Varianzanalyse - Wechselwirkungen

10.1 Vorgehen bei balancierten Daten (10.3/10.4)

Seite 246, Beispiel 1:

```
Dünger <- c(rep(1, 9), rep(2, 9))
Schutz <- rep(c(1, 2, 3), 6)
Ertrag <- c(21.3, 22.3, 23.8, 20.9, 21.6, 23.7, 20.4, 21.0, 22.6, 12.7,
 12.0, 14.5, 14.9, 14.2, 16.7, 12.9, 12.1, 14.5)
```

Mit `1, 2` in `Dünger` = 'Düngung D1' und 'Düngung D2' respektive
und `1, 2, 3` in `Schutz` = 'Pflanzenschutz P1', 'Pflanzenschutz P2' und 'Pflanzenschutz P3' respektive.

```
df.wein <- data.frame(Dünger=factor(Dünger), Schutz=factor(Schutz),
 Ertrag)
lm.wein <- lm(Ertrag ~ Dünger*Schutz, df.wein) # mit Wechselwirkungen?
```

```
anova(lm.wein)
 Analysis of Variance Table

 Response: Ertrag
                              Df  Sum Sq Mean Sq  F value    Pr(>F)
 factor(Dünger)                1 296.867 296.867 312.1268 5.904e-10 ***
 factor(Schutz)                2  17.781   8.891   9.3475   0.00357 **
 factor(Dünger):factor(Schutz) 2   1.688   0.844   0.8873   0.43715
 Residuals                    12  11.413   0.951
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
lm.wein <- lm(Ertrag ~ Dünger + Schutz, df.wein) # ohne Wechselwirkungen!
```

```
glht.wein1 <- glht(lm.wein, linfct=mcp(Dünger="Tukey"))
```

```
 Warnmeldung:
 In mcp2matrix(model, linfct = linfct) :
   covariate interactions found -- default contrast might be inappropriate

-------------------------------------------------------------------------

sum.glht1 <- summary(glht.wein1, test=adjusted("none") )

-------------------------------------------------------------------------

cld(sum.glht1)
   2   1
 "a" "b"

-------------------------------------------------------------------------

glht.wein2 <- glht(lm.wein, linfct=mcp(Schutz="Tukey"))
 Warnmeldung:
 In mcp2matrix(model, linfct = linfct) :
   covariate interactions found -- default contrast might be inappropriate

-------------------------------------------------------------------------

sum.glht2 <- summary(glht.wein, test=adjusted("none") )

-------------------------------------------------------------------------

cld(sum.glht2)
   2   3   1
 "a" "b" "a"
```

An diesem Beispiel sieht man, dass die Wahl der Bezeichnungen durchaus wichtig ist. Bei großen Datensätzen ist es daher sinnvoll vielleicht doch etwas längere oder komplexere Bezeichnungen für Variablen einzuführen, da die zusätzliche Arbeit dadurch aufgewogen wird, dass das Ergebnis besser interpretierbar/verstehbar ist.

Nach der Wilkinson-Rogers Notation ist x*y das selbe wie x+y+x:y, man kann sich so einiges an Schreibarbeit sparen.

Hier wurden anstelle der Bezeichnungen 'D1' oder 'P1' einfach Zahlen für die verschiedenen Stufen der Faktoren verwendet um Tipparbeit zu sparen. Betrachtet man das Ergebnis ohne das im Hinterkopf zu haben, fragt man sich eventuell, welche Ausführung denn jetzt welche Faktoren und welche Stufen darstellt. In diesem Beispiel kann man das jedoch noch leicht nachvollziehen, da lediglich zwei Faktoren betrachtet werden, von denen der Eine drei und der Andere zwei Stufen hat.

Je mehr Faktoren, beziehungsweise Stufen für die Faktoren, desto mehr Vergleiche muss man aber durchführen. Da kann es also schnell zu Verwirrungen kommen, welchen man eben durch ordentliche Bezeichnungen vorbeugen kann.

Die $\star$-Notation beinhaltet die Addition der Faktoren und die Interaktion der Faktoren (sogenanntes Tensorprodukt). So kann man sich Tipparbeit sparen.

Seite 248, Beispiel 1:

```
düng <- c(rep("Kontrolle", 12), rep("Stroh", 12), rep("Phosphat + Stroh",
 12), rep("Kalk + Phosphat + Stroh", 12))
chem <- rep(c("Kontrolle", "N + O", "CO2", "H2CO3"), 12)
Weizen <- c(21.4, 20.9, 19.6, 17.6, 21.2, 20.3, 18.8, 16.6, 20.1, 19.8,
 16.4, 17.5, 12.0, 13.6, 13.0, 13.3, 14.2, 13.3, 13.7, 14.0, 12.1, 11.6,
 12.0, 13.9, 13.5, 14.0, 12.9, 12.4, 11.9, 15.6, 12.9, 13.7, 13.4, 13.8,
 13.1, 13.0, 12.8, 14.1, 14.2, 12.0, 13.8, 13.2, 13.6, 14.6, 13.7, 15.3,
 13.3, 14.0)

------------------------------------------------------------------------

df.weizen <- data.frame(düng, chem, Weizen)
lm.weizen <- lm(Weizen ~ factor(düng)*factor(chem))

------------------------------------------------------------------------

anova(lm.weizen)
 Analysis of Variance Table

 Response: Weizen
                           Df  Sum Sq Mean Sq  F value    Pr(>F)
 factor(düng)               3 306.242 102.081 124.2993 < 2.2e-16 ***
 factor(chem)               3   9.171   3.057   3.7222  0.021077 *
 factor(düng):factor(chem)  9  25.462   2.829   3.4449  0.004536 **
 Residuals                 32  26.280   0.821
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Der Effekt der Wechselwirkungen ist hier eindeutig signifikant, es müssen also die Mittelwertvergleiche getrennt für die Stufen der anderen Faktoren durchgeführt werden.

```
### 1. Vergleich der Düngerbehandlungen getrennt für jede Stufe der
 chemischen Behandlungen ###

df.Unbehandelt <- df.weizen[which(df.weizen$chem == "Kontrolle"),]

------------------------------------------------------------------------

df.Unbehandelt
                       düng      chem Weizen
 1                 Kontrolle Kontrolle   21.4
 5                 Kontrolle Kontrolle   21.2
 9                 Kontrolle Kontrolle   20.1
 13                    Stroh Kontrolle   12.0
 17                    Stroh Kontrolle   14.2
 21                    Stroh Kontrolle   12.1
 25         Phosphat + Stroh Kontrolle   13.5
 29         Phosphat + Stroh Kontrolle   11.9
 33         Phosphat + Stroh Kontrolle   13.4
 37 Kalk + Phosphat + Stroh Kontrolle   12.8
```

```
 41 Kalk + Phosphat + Stroh Kontrolle   13.8
 45 Kalk + Phosphat + Stroh Kontrolle   13.7
```

```
lm.Unbehandelt <- lm(Weizen~düng, df.Unbehandelt)
glht.Unbehandelt <- glht(lm.Unbehandelt, linfct=mcp(düng="Tukey"))
LSD.Unbehandelt <- summary(glht.Unbehandelt, test=adjusted("none"))
```

```
cld(LSD.Unbehandelt)
          Kontrolle      Phosphat+Stroh               Stroh
                "a"                 "b"                 "b"
          Kalk+Phosphat+Stroh
                          "b"
### Wiederholen für die anderen chemischen Behandlungen durch Einsetzen
 der Bezeichnung der chemischen Behandlung in
 'df.Bezeichnung <- df.weizen[which(df.weizen$chem == "Bezeichnung"),]'
 und abarbeiten des Prozederes ###
```

```
### 2. Vergleich der chemischen Behandlungen getrennt für jede Stufe
 der Düngerbehandlungen ###

df.Kontrolle <- df.weizen[which(df.weizen$düng == "Kontrolle"),]
```

```
df.Kontrolle1
         düng      chem Weizen
 1  Kontrolle Kontrolle   21.4
 2  Kontrolle     N + O   20.9
 3  Kontrolle       CO2   19.6
 4  Kontrolle     H2CO3   17.6
 5  Kontrolle Kontrolle   21.2
 6  Kontrolle     N + O   20.3
 7  Kontrolle       CO2   18.8
 8  Kontrolle     H2CO3   16.6
 9  Kontrolle Kontrolle   20.1
 10 Kontrolle     N + O   19.8
 11 Kontrolle       CO2   16.4
 12 Kontrolle     H2CO3   17.5
```

```
lm.Kontrolle <- lm( Weizen~chem, df.Kontrolle )
glht.Kontrolle <- glht( lm.Kontrolle, linfct=mcp(chem="Tukey") )
LSD.Kontrolle <- summary(glht.Kontrolle,test=adjusted("none"))
```

```
cld(LSD.Kontrolle)
    H2CO3 Kontrolle       N+O       CO2
      "a"       "b"       "b"       "a"

### Wiederholen für die anderen Düngerbehandlungen durch Einsetzen der
 Behandlungsbezeichnungen in
 'df.Bezeichnung <- df.weizen[which(df.weizen$düng == "Bezeichnung"),]'
 und abarbeiten des Prozederes ###
```

Hier wurden die Daten zunächst nach der Behandlung selektiert, für die man einen Mittelwertvergleich anstellen will. An diese Daten wurde dann ein lineares Modell angepasst, aus dem man dann mittels der Funktion `glht()` die multiplen Mittelwertvergleiche erstellt werden. Mit der Funktion `cld()` kann man aus diesem Datenoutput eine schöne und übersichtliche Buchstabendarstellung darstellen lassen.

Hier wurden lediglich die jeweils ersten Mittelwertvergleiche der Behandlungen dargestellt. So sind die im 1. Vergleich zu sehenden Buchstaben die Buchstaben, die in Herrn Piephos Skript in Tabelle 10.5 in der ersten Spalte der Mittelwerte zu sehen sind, die Buchstaben des 2. Vergleichs sind in Tabelle 10.6 in der ersten Zeile zu sehen.

10.2 Vorgehen bei unbalancierten Daten (10.5/10.6)

Bei unbalancierten Daten ist das Vorgehen ähnlich dem bei balancierten Daten. Jedoch muss beachtet werden, dass der Modellaufbau der richtigen Sequenz folgt.

Seite 253, Beispiel 1:

```
Ertrag.2 <- c(21.3, 22.3, 23.8, 20.9, 21.6, NA, 20.4, 21.0, 22.6, 12.7,
 12.0, 14.5, 14.9, 14.2, 16.7, NA, 12.1, 14.5)

------------------------------------------------------------------------

### Test für 'Schutz' ###
lm.wein <- lm(Ertrag.2 ~ factor(Dünger)*factor(Schutz), df.wein)

------------------------------------------------------------------------

anova(lm.wein)
 Analysis of Variance Table

 Response: Ertrag.2
                              Df  Sum Sq Mean Sq  F value    Pr(>F)
 factor(Dünger)                1 242.581 242.581 226.5697 3.381e-08 ***
 factor(Schutz)                2  13.604   6.802   6.3529   0.01657 *
 factor(Dünger):factor(Schutz) 2   2.168   1.084   1.0126   0.39769
 Residuals                    10  10.707   1.071
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
-------------------------------------------------------------------------

### Test für 'Dünger' ###
lm.wein.2 <- lm(Ertrag.2 ~ factor(Schutz)*factor(Dünger), df.wein)

-------------------------------------------------------------------------

anova(lm.wein.2)
 Analysis of Variance Table

 Response: Ertrag.2
                              Df  Sum Sq Mean Sq  F value    Pr(>F)
 factor(Schutz)                2   4.339   2.170   2.0265    0.1825
 factor(Dünger)                1 251.845 251.845 235.2226 2.824e-08 ***
 factor(Schutz):factor(Dünger) 2   2.168   1.084   1.0126    0.3977
 Residuals                    10  10.707   1.071
 ---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Der Test von `lm.wein` ist aufgrund der nicht-Signifikanz der Interaktion gleichzeitig auch der Test für `Schutz` (um `Dünger` bereinigt). Hier wird Signifikanz für `Schutz` angezeigt. Um `Dünger` zu testen muss man diesen um `Schutz` bereinigen. Dies wird beim Test von `lm.wein.2` gemacht, indem die Reihenfolge der beiden Faktoren umgekehrt wird. Mit dieser Notation ändert man nicht nur die Reihenfolge der Interaktionen, sondern auch die der Hauptwirkungen.

```
### Mittelwertvergleich für Pflanzenschutzbehandlung (Schutz) ###

fit.wein <- lm( Ertrag.2 ~ Schutz + Dünger - 1, df.wein)
Coeff <- coef(fit.wein)

-------------------------------------------------------------------------

Coeff
   Schutz1   Schutz2   Schutz3   Dünger2
 21.254359 21.217949 23.241538 -8.035897

-------------------------------------------------------------------------

Coeff[1:3] + Coeff[4]/2
  Schutz1  Schutz2  Schutz3
 17.23641 17.20000 19.22359

-------------------------------------------------------------------------

fit.wein$coefficients[1:3] <- Coeff[1:3] + Coeff[4]/2
glht.wein <- glht(fit.wein, linfct=mcp(Schutz="Tukey"))
sum.glht <- summary(glht.wein, test=adjusted("none"))

-------------------------------------------------------------------------
```

```
cld(sum.glht)
   2   3   1
 "a" "b" "a"
```

```
### Mittelwertvergleiche für Dünger ###

fit.wein <- lm( Ertrag.2 ~ Dünger + Schutz - 1, df.wein)
Coeff <- coef(fit.wein)
```

```
Coeff
     Dünger1     Dünger2     Schutz2     Schutz3
 21.25435897 13.21846154 -0.03641026  1.98717949
```

```
Coeff[1:2] + (Coeff[3]+Coeff[4])/3
  Dünger1  Dünger2
 21.90462 13.86872
```

```
fit.wein$coefficients[1:2] <- Coeff[1:2] + (Coeff[3] + Coeff[4])/3
glht.wein <- glht(fit.wein, linfct=mcp(Dünger="Tukey"))
sum.glht <- summary(glht.wein, test=adjusted("none"))
```

```
cld(sum.glht)
   2   1
 "a" "b"
```

Hier wird der Mittelwertvergleich mit den adjustierten Mittelwerten durchgeführt. Da ein Varianzanalysemodell meist überparametrisiert ist, muss eine Restriktion durchgeführt werden, bei der eine der Variablen heraus fällt (also auf `0` gesetzt wird).

Mit `- 1` in der Bildung des linearen Modells zeigt man an, dass der Gesamteffekt nicht mit auf genommen werden soll, sondern lediglich die Einzeleffekte dargestellt werden sollen.

Um nun die adjustierten Randmittelwerte zu erhalten, muss man den Mittelwert der einfachen Mittelwerte innerhalb der Stichprobe berechnen und nicht den Mittelwert über die gesamte Stichprobe. In R erreicht man dies, indem man den Mittelwert der Koeffizienten der nicht betrachteten Faktoren (inklusive der durch die Restriktion weg gefallenen Faktoren) bildet und zu den Koeffizienten der betrachteten Faktoren addiert. Mit den so erhaltenen Randmittelwerten kann man dann die Buchstabendarstellung mit Hilfe der Funktion `glht()` und `cld()` darstellen.

11 Elementare nichtparametrische Verfahren

11.1 Test für zwei unverbundene Stichproben

11.1.1 Median-Test (Fisher-Test des Medians)

```
Dobutamin <- c(18.80, 41.70, 41.24, 49.98, 63.90, 29.86, 25.92, 36.92,
 54.90, 37.04, 28.94, 33.80, 18.08)
Fenoterol <- c(34.16, 67.82, 60.68, 39.04, 37.72, 46.78, 50.80, 128.60,
 31.18, 35.04, 56.40, 22.62, 26.09)
```

```
x <- c(Dobutamin, Fenoterol)
```

```
median(x)
 [1] 37.38
```

```
x <- matrix(c(8, 5, 5, 8), 2)
```

```
x
      [,1] [,2]
 [1,]    8    5
 [2,]    5    8
```

```
fisher.test(x)

         Fisher's Exact Test for Count Data

 data:  x
 p-value = 0.4338
 alternative hypothesis: true odds ratio is not equal to 1
 95 percent confidence interval:
   0.4133567 16.5168885
```

```
sample estimates:
odds ratio
  2.465765
```

siehe auch '↑ 5.6 Test auf Unabhängigkeit in der 2 * 2 Feldertafel (4-Feldertafel)'

11.1.2 Mann-Whitney-Test

Der Mann-Whitney-Test ist in R lediglich ein Spezialfall des Kruskal-Wallis-Test. Wie beim t-Test (`t.test`) kann man über das Argument `paired =` definieren, ob eine verbundene oder eine unverbundene Stichprobe getestet werden soll.

`wilcox.test()`: Führt den Wilcoxon-Test durch.

`x =` : Erster (numerischer) Vektor.

`y = NULL`: Zweiter (numerischer) Vektor. Steht standardmäßig auf `NULL`, was bedeutet, dass der Vektor nicht beachtet wird.

`alternative = c("two.sided", "less", "greater")`: spezifiziert die Alternativhypothese H_1.

`paired = FALSE/TRUE`: gibt an, ob eine verbundene oder eine unverbundene Stichprobe getestet werden soll.

`exact = NULL`: gibt an, ob der exakte Test berechnet werden soll; logisch.

`conf.int = FALSE/TRUE`: gibt an, ob das Vertrauensintervall ausgegeben werden soll.

`conf.level = 0.95`: gibt das Niveau des Vertrauensintervalls an.

Seite 266, Beispiel 1:

```
wilcox.test(x=Dobutamin, y=Fenoterol)

        Wilcoxon rank sum test

data:  Dobutamin and Fenoterol
W = 60, p-value = 0.2226
alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(x=Dobutamin, y=Fenoterol, exact=FALSE)

        Wilcoxon rank sum test with continuity correction
```

```
 data:  Dobutamin and Fenoterol
 W = 60, p-value = 0.2184
 alternative hypothesis: true location shift is not equal to 0
```

Da hier von Interesse ist, ob die Mediane zwei verschiedener unabhängiger Gruppen gleich sind, muss man beide Gruppen angeben. Das Argument `paired =` ist standardmäßig auf `FALSE`, also testet man tatsächlich eine unverbundene Stichprobe und führt damit den Mann-Whitney-Test durch. Setzt man `paired =` auf `TRUE` führt man den Vorzeichen-Rangtest nach Wilcoxon durch.

11.2 Kruskal-Wallis-Test (H-Test) für mehr als zwei unverbundene Stichproben

`kruskal.test()`: Führt den Kruskal-Wallis-Rangsummentest durch.

`x =` : (numerischer) Vektor, oder Liste (numerischer) Daten.

`g =` : Vektor, der die Daten von `x` anhand von Faktoren in Gruppen gliedert.

```
Propanolol <- c(13.00, 10.00, 12.86, 11.38, 9.94, 13.24, 7.32, 1.10,
 12.56, 14.36)
Placebo <- c(20.94, 25.56, 19.64, 9.26, 16.42, 16.22, 16.90, 26.46, 15.26,
 10.90, 18.16, 21.50, 26.58)

------------------------------------------------------------------------

x <- c(Propanolol, Placebo, Dobutamin, Fenoterol)
g <- factor(rep(1:4, c(10, 13, 13, 13)), labels=c("Propanolol",
 "Placebo", "Dobutamin", "Fenoterol"))

------------------------------------------------------------------------

kruskal.test(x, g)

         Kruskal-Wallis rank sum test

 data:  x and g
 Kruskal-Wallis chi-squared = 34.9844, df = 3, p-value = 1.228e-07
```

Der `kruskal.test()` erhält die Daten in Form eines (numerischen) Vektors und weiß alleine aus diesen Informationen nicht, aus welchen Daten die Rangsummen gebildet werden sollen, die verglichen werden sollen. Daher gibt es das Argument `g =` , in dem man die Gruppen spezifiziert, für welche die Rangsummen gebildet werden sollen. Dabei handelt es sich um einen (alphanumerischen) Vektor (also einen, der Buchstaben enthalten kann).

Das Ergebnis dieses Tests zeigt Signifikanz, also sind die Mediane der vier Gruppen nicht gleich.

11.3 Vorzeichen-Rangtest nach Wilcoxon für zwei verbundene Stichproben

```
vor <- c(1.4580, 1.6550, 3.0160, 0.8110, 1.6120, 0.1580, 0.1660, 0.2870,
 0.7460, 1.1900, 2.4150, 157.4910, 7.1520, 1.5650, 9.2190, 3.0970, 0.9720,
 0.8110, 1.8430, 2.5290, 7.3290, 0.9550, 4.2000, 1.9720, 1.7070, 4.0680,
 3.8100, 12.7812, 0.3831, 3.3303)
nach <- c(1.4160, 0.8020, 1.8210, 0.7880, 1.5040, 0.1450, 0.1120, 0.3180,
 0.4930, 1.8090, 1.0610, 220.3630, 4.5660, 1.3240, 5.3350, 2.1090, 0.4000,
 1.0690, 1.8090, 1.9260, 4.7940, 0.7850, 1.7240, 0.6820, 1.2520, 1.1470,
 2.4112, 8.1911, 0.1406, 1.2813)

------------------------------------------------------------------------

wilcox.test(vor, nach, paired=TRUE, conf.int=TRUE)

         Wilcoxon signed rank test

 data:  vor and nach
 V = 404, p-value = 0.0001886
 alternative hypothesis: true location shift is not equal to 0
 95 percent confidence interval:
  0.286 1.354
 sample estimates:
 (pseudo)median
          0.724
```

Hier wurden die beiden Vektoren angegeben. Zusätzlich wurde angegeben, dass es sich um eine verbundene Stichprobe handelt. Die Funktion `wilcox.test()` kann, je nach Input, sowohl den Vorzeichen-Test, als auch den Vorzeichen-Rang-Test durchführen. Im zweiten Beispiel wurde der Vorzeichen-Rang-Test durchgeführt, da zwei Stichproben angegeben waren. Beim ersten Beispiel waren allerdings auch zwei Stichproben angegeben, hier wurde als zweite Stichprobe der standardmäßig eingestellte Wert 0 von `y` verwendet.

```
t.test(vor, nach, paired=TRUE)

         Paired t-test

 data:  vor and nach
 t = -0.511, df = 29, p-value = 0.6132
 alternative hypothesis: true difference in means is not equal to 0
 95 percent confidence interval:
  -5.477793  3.287820
 sample estimates:
 mean of the differences
```

```
-1.094987
```

Der t-Test liefert ein nicht-signifikantes Ergebnis. Dieses ist jedoch nicht gültig, da es sich um einen extrem schief verteilten Datensatz handelt, also die Voraussetzung der Normalverteilung nicht gegeben sind.

11.4 Friedman-Test für mehr als zwei verbundene Stichproben

`friedman.test()`: führt den Friedman-Rangsummentest durch.

`y =` : gibt den Vektor der Daten an.

`groups =` : gibt den Vektor der Gruppen (Behandlugnen, ...) an.

`blocks =` : gibt den Vektor der Blöcke an.

`formula =` : gibt eine Formel an, die dem Test mitteilt, wie die Daten (Werte, Gruppen, Blöcke) verarbeitet werden sollen.

```
none <- c(10, 14, 17, 8, 9, 31)
ddt<- c(4, 2, 0, 3, 2, 11)
malathion <- c(3, 6, 8, 0, 3, 16)

------------------------------------------------------------------------

m <- matrix(c(none, ddt, malathion), 6, dimnames=list(1:6,
 c("Kontrolle", "DDT", "Malathion")))

------------------------------------------------------------------------

friedman.test(m)

         Friedman rank sum test

 data:  m
 Friedman chi-squared = 9.3333, df = 2, p-value = 0.009404

------------------------------------------------------------------------

Block <- rep(c(1, 2, 3, 4, 5, 6), 3)
Behandlung <- c(rep("Kontrolle", 6), rep("DDT", 6), rep("Malathion", 6))
x <- c(none, ddt, malathion)

------------------------------------------------------------------------

df.insekt <- data.frame(Anzahl=c(none, ddt, malathion), Block,
 Behandlung)
```

```
friedman.test(df.insekt$Anzahl, df.insekt$Behandlung, df.insekt$Block)

         Friedman rank sum test

 data:  df.insekt$Anzahl, df.insekt$Behandlung and df.insekt$Block
 Friedman chi-squared = 9.3333, df = 2, p-value = 0.009404

------------------------------------------------------------------------

friedman.test(Anzahl ~ Behandlung | Block, df.insekt)

         Friedman rank sum test

 data:  Anzahl and Behandlung and Block
 Friedman chi-squared = 9.3333, df = 2, p-value = 0.009404
```

Mir gefällt die dritte Methode am besten, da hier gleich klar wird, welchen Zusammenhang man beschreibt und mir persönlich das Arbeiten mit Dataframes im Vergleich zum Arbeiten mit Matrizen sympathischer ist.

12 Kovarianzanalyse

Bei der Kovarianzanalyse geht man im Endeffekt genauso wie bei der Varianzanalyse vor. Zu beachten ist jedoch, dass die Einflussfaktoren in der richtigen Reihenfolge angepasst werden.

12.1 Varianzanalyse

Seite 289, Beispiel 1:

```
Anfangsgewicht <- c(61, 59, 76, 50, 61, 54, 57, 45, 41, 40, 74, 75, 64,
 48, 62, 42, 52, 43, 50, 40, 80, 61, 62, 47, 59, 42, 47, 42, 40, 40, 62,
 55, 62, 43, 57, 51, 41, 40, 45, 39)
Zunahme <- c(1.4, 1.79, 1.72, 1.47, 1.26, 1.28, 1.34, 1.55, 1.57, 1.26,
 1.61, 1.31, 1.12, 1.35, 1.29, 1.24, 1.29, 1.43, 1.29, 1.26, 1.67, 1.41,
 1.73, 1.23, 1.49, 1.22, 1.39, 1.39, 1.56, 1.36, 1.40, 1.47, 1.37, 1.15,
 1.22, 1.48, 1.31, 1.27, 1.22, 1.36)
Futter <- c(rep("Futter 1", 10), rep("Futter 2", 10), rep("Futter 3", 10),
 rep("Futter 4", 10))

------------------------------------------------------------------------

df.schwein <- data.frame(Anfangsgewicht=Anfangsgewicht, Futter=
factor(Futter), Zunahme=Zunahme)
ancova.schwein <- lm(Zunahme ~ Anfangsgewicht*Futter, df.schwein)

------------------------------------------------------------------------

anova(ancova.schwein)
 Analysis of Variance Table

 Response: Zunahme
                              Df  Sum Sq Mean Sq F value  Pr(>F)
 Anfangsgewicht                1 0.14972 0.14972  7.0566 0.01221 *
 factor(Futter)                3 0.16878 0.05626  2.6516 0.06544 .
 Anfangsgewicht:factor(Futter) 3 0.02531 0.00844  0.3977 0.75558
 Residuals                    32 0.67896 0.02122
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

In dieser Varianzanalysetabelle sind alle relevanten Daten enthalten, um zu einer Interpretation zu kommen. Die Wechselwirkung `Anfangsgewicht:factor(Futter)`

stellt hier den Lack-of-fit-Term δ_i dar, mit dem getestet werden soll, ob die Interaktionen einen signifikanten Einfluss auf den Zusammenhang haben. Da hier deutlich keine Signifikanz vorliegt, kann man also davon ausgehen, dass keine Interaktionen zwischen den Behandlungen existieren und damit die Schaubilder parallel sind.

`Anfangsgewicht` wurde vor `factor(Futter)` angepasst, da es sich bei `Anfangsgewicht` um die Kovariable handelt und man den Effekt von `factor(Futter)` um die Kovariable bereinigt haben will.

12.2 Mittelwertvergleiche

Die Interaktionen zeigen in der Varianzanalyse-Tabelle keine Signifikanz, daher passen wir ein reduziertes Modell an:

```
ancova.schwein <- lm(Zunahme ~ Anfangsgewicht + Futter - 1, df.schwein)
```

```
anova(ancova.schwein)
 Analysis of Variance Table

 Response: Zunahme
               Df Sum Sq Mean Sq  F value    Pr(>F)
 Anfangsgewicht  1 75.124  75.124 3733.418 < 2.2e-16 ***
 Futter          4  2.284   0.571   28.376 1.491e-10 ***
 Residuals      35  0.704   0.020
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
summary(ancova.schwein)

 Call:
 lm(formula = Zunahme ~ Anfangsgewicht + Futter - 1, data = df.schwein)

 Residuals:
       Min         1Q     Median         3Q        Max
 -0.247385 -0.103743 -0.003023  0.097445  0.301270

 Coefficients:
               Estimate Std. Error t value Pr(>|t|)
 Anfangsgewicht 0.005376   0.002031   2.647   0.0121 *
 FutterFutter 1 1.171537   0.119258   9.824 1.35e-11 ***
 FutterFutter 2 1.023311   0.120388   8.500 4.97e-10 ***
 FutterFutter 3 1.165440   0.114756  10.156 5.65e-12 ***
 FutterFutter 4 1.058880   0.110100   9.617 2.33e-11 ***
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
 Residual standard error: 0.1419 on 35 degrees of freedom
 Multiple R-squared: 0.991,      Adjusted R-squared: 0.9897
 F-statistic: 769.4 on 5 and 35 DF,  p-value: < 2.2e-16
```

Bisher war es so, dass wir die Mittelwerte leicht aus der Varianzanalyse-Tabelle ablesen und modifizieren konnten. Da wir hier allerdings nicht den y-Achsenabschnitt zur Analyse heranziehen wollen, sondern den Mittelwert aller Gewichtszunahmen, können wir hier nicht den von `lm()` geschätzten Wert (`Estimate`) verwenden. Stattdessen müssen wir diesen Mittelwert gesondert berechnen.

```
F1 <- df.schwein[which(df.schwein$Futter == "Futter 1"),]
F2 <- df.schwein[which(df.schwein$Futter == "Futter 2"),]
F3 <- df.schwein[which(df.schwein$Futter == "Futter 3"),]
F4 <- df.schwein[which(df.schwein$Futter == "Futter 4"),]
```

```
F1
    Anfangsgewicht   Futter Zunahme
 1              61 Futter 1    1.40
 2              59 Futter 1    1.79
 3              76 Futter 1    1.72
 4              50 Futter 1    1.47
 5              61 Futter 1    1.26
 6              54 Futter 1    1.28
 7              57 Futter 1    1.34
 8              45 Futter 1    1.55
 9              41 Futter 1    1.57
 10             40 Futter 1    1.26
```

```
mean(F1[,1]); mean(F1[,3])
 [1] 54.4
 [1] 1.464
```

```
mean(F2[,1]); mean(F2[,3])
 [1] 55
 [1] 1.319
```

```
mean(F3[,1]); mean(F3[,3])
 [1] 52
 [1] 1.445
```

```
mean(F4[,1]); mean(F4[,3])
```

```
 [1] 49.5
 [1] 1.325
```

```
mean(df.schwein[,1])
 [1] 52.725
```

Da die Daten einen linearen Zusammenhang aufweisen, könnte man die Mittelwerte für die Futterbehandlungen auch über die Mittelwerte der Anfangsgewichte ausrechnen, indem man den von `lm()` geschätzten Wert für die Futterbehandlung als y-Achsenabschnitt betrachtet, den Mittelwert der Anfangsgewichte als x-Wert und den von `lm()` geschätzten Wert für `Anfangsgewicht` als Steigung betrachtet.

```
1.171537+0.005376*54.4
 [1] 1.463991
```

Aus dem von `lm()` geschätzten Wert für `Anfangsgewicht` und den oben berechneten Mittelwerten lassen sich nun die korrigierten Mittelwerte berechnen:

```
m1 <- mean(F1[,3]) - ancova.schwein$coef[[1]]*(mean(F1[,1]) -
 mean(df.schwein[,1]))
m2 <- mean(F2[,3]) - ancova.schwein$coef[[1]]*(mean(F2[,1]) -
 mean(df.schwein[,1]))
m3 <- mean(F3[,3]) - ancova.schwein$coef[[1]]*(mean(F3[,1]) -
 mean(df.schwein[,1]))
m4 <- mean(F4[,3]) - ancova.schwein$coef[[1]]*(mean(F4[,1]) -
 mean(df.schwein[,1]))
coef <- c(m1, m2, m3, m4)
```

```
coef
 [1] 1.454995 1.306769 1.448898 1.342338
```

```
sqx <- sum((df.schwein[,1] - mean(df.schwein[,1]))**2)
```

Alle anderen Werte zur Berechnung der t_{Vers}-Werte lassen sich aus dem bisherigen Output ablesen, der $MQ^{(2)}_{Fehler}$-Wert ist in der Varianzanalyse-Tabelle des reduzierten Modells zu finden. Der t_{Tab}-Wert lässt sich mit der Funktion `qt()` ermitteln.

```
qt(0.975, 35)
 [1] 2.030108
```

Die lsmeans kann man in R bisher nicht berechnen, allerdings kann man sich einen paarweisen Vergleich der Mittelwerte ausgeben lassen. Dies folgt der schon bekannten Funktion `glht()`. Dazu muss man zu aller erst die Koeffizienten des reduzierten Modells durch die korrigierten Mittelwerte ersetzten und anschließend ein glht-Objekt erstellen und mit `summary()` betrachten.

```
ancova.schwein$coefficients
 Anfangsgewicht FutterFutter 1 FutterFutter 2 FutterFutter 3
    0.005376161    1.454994930    1.306769234    1.448897717
 FutterFutter 4
    1.342338120
```

```
ancova.schwein$coefficients[2:5] <- coef
```

```
ancova.schwein

 Call:
 lm(formula = Zunahme ~ Anfangsgewicht + Futter - 1, data = df.schwein)

 Coefficients:
 Anfangsgewicht  FutterFutter 1  FutterFutter 2  FutterFutter 3
       0.005376        1.454995        1.306769        1.448898
 FutterFutter 4
       1.342338
```

```
glht.schwein <- glht(ancova.schwein, linfct=mcp(Futter="Tukey") )
```

```
summary(glht.schwein, test=adjusted("none"))


          Simultaneous Tests for General Linear Hypotheses

 Multiple Comparisons of Means: Tukey Contrasts


 Fit: lm(formula = Zunahme ~ Anfangsgewicht + Futter - 1, data =
  Futter.df)

 Linear Hypotheses:
                           Estimate Std. Error t value Pr(>|t|)
 Futter 2 - Futter 1 == 0 -0.148226   0.063450  -2.336   0.0253 *
 Futter 3 - Futter 1 == 0 -0.006097   0.063625  -0.096   0.9242
 Futter 4 - Futter 1 == 0 -0.112657   0.064214  -1.754   0.0881 .
 Futter 3 - Futter 2 == 0  0.142128   0.063730   2.230   0.0322 *
```

```
Futter 4 - Futter 2 == 0  0.035569   0.064414   0.552   0.5843
Futter 4 - Futter 3 == 0 -0.106560   0.063641  -1.674   0.1030
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- none method)
```

Hier sei erwähnt, dass als Strukturelement des Modells nicht nur `coef` verwendet werden kann, da die Koeffizienten sonst nicht in der Art im Modell stehen, dass weiterführende Funktionen darauf zugreifen können.

12.3 Ein soziologisches Beispiel

Auch für dieses Beispiel finden sich die Daten im Anhang. Das Prinzip der Methode folgt dem bereits aufgeführten Wissen.

13 Messwiederholungen

```
weight <- c(57, 60, 52, 49, 56, 46, 51, 63, 49, 57, 59, 54, 56, 59, 57,
 52, 52, 61, 59, 53, 59, 51, 51, 56, 58, 46, 53, 86, 93, 77, 67, 81, 70,
 71, 91, 67, 82, 85, 71, 75, 85, 72, 73, 70, 86, 80, 79, 88, 75, 75, 78,
 69, 61, 72, 114, 123, 111, 100, 104, 102, 94, 112, 90, 110, 121, 90,
 108, 116, 97, 97, 105, 109, 101, 100, 100, 101, 92, 95, 93, 78, 89, 139,
 146, 144, 129, 121, 131, 110, 130, 112, 139, 156, 110, 151, 148, 120,
 116, 138, 120, 111, 106, 111, 123, 100, 103, 114, 90, 104, 172, 177,
 185, 164, 151, 153, 141, 154, 140, 169, 191, 138, 189, 177, 144, 140,
 171, 129, 126, 133, 122, 140, 119, 108, 138, 107, 122)

-------------------------------------------------------------------------

df.ratten <- data.frame(weight=weight, ratte=rep(paste("R",1:27,sep=""),
5), week=c(rep("W0", 27), rep("W1", 27), rep("W2", 27), rep("W3", 27),
 rep("W4", 27)), chemikalie=c(rep( c(rep("Kontrolle", 10), rep("Tyroxin",
  7), rep("Thiouracil", 10)), 5)))
lm.ratten0 <- lm(weight ~ chemikalie,
 df.ratten[which(df.ratten$week=="W0"),])
lm.ratten1 <- lm(weight ~ chemikalie - 1,
 df.ratten[which(df.ratten$week=="W1"),])
lm.ratten2 <- lm(weight ~ chemikalie - 1,
 df.ratten[which(df.ratten$week=="W2"),])
lm.ratten3 <- lm(weight ~ chemikalie - 1,
 df.ratten[which(df.ratten$week=="W3"),])
lm.ratten4 <- lm(weight ~ chemikalie - 1,
 df.ratten[which(df.ratten$week=="W4"),])

-------------------------------------------------------------------------

cld0 <- cld(summary(glht(lm.ratten0, linfct=mcp(chemikalie="Tukey")),
 test=adjusted(type="none")))
cld1 <- cld(summary(glht(lm.ratten1, linfct=mcp(chemikalie="Tukey")),
 test=adjusted(type="none")))
cld2 <- cld(summary(glht(lm.ratten2, linfct=mcp(chemikalie="Tukey")),
 test=adjusted(type="none")))
cld3 <- cld(summary(glht(lm.ratten3, linfct=mcp(chemikalie="Tukey")),
 test=adjusted(type="none")))
cld4 <- cld(summary(glht(lm.ratten4, linfct=mcp(chemikalie="Tukey")),
 test=adjusted(type="none")))

-------------------------------------------------------------------------

chem.mat <- cbind(
```

```
paste(round(unique(predict(lm.ratten0)),1), cld0$mcletters$Letters),
paste(round(unique(predict(lm.ratten1)),1), cld1$mcletters$Letters),
paste(round(unique(predict(lm.ratten2)),1), cld2$mcletters$Letters),
paste(round(unique(predict(lm.ratten3)),1), cld3$mcletters$Letters),
paste(round(unique(predict(lm.ratten4)),1), cld4$mcletters$Letters) )

-------------------------------------------------------------------------

colnames(chem.mat) <- paste("Woche ", 0:4, sep="_")
rownames(chem.mat) <- unique(as.character(df.ratten$chemikalie))
chem.mat <- as.data.frame(chem.mat)

-------------------------------------------------------------------------

chem.mat
            Woche _0 Woche _1 Woche _2 Woche _3 Woche _4
 Kontrolle      54 a   78.5 a    106 a  130.1 a  160.6 a
 Tyroxin      55.6 a   75.9 a 104.9 ab  134.1 b  164.3 b
 Thiouracil   54.7 a   76.3 a   95.8 b  108.2 b  124.4 b
```

Hier wird ebenfalls mit den Funktionen `glht()` und `cld()` gearbeitet. Das Prozedere unterscheidet sich nicht zu dem davor schon kennengelernten. Einige Anmerkungen gibt es jedoch.

Das Argument, von dem die Werte für die Bildung der Regression genommen werden, sind nicht mehr der komplette Datensatz, sondern nur ein Teil davon. Die Funktion `which()` ermittelt die Elemente eines Objektes der Klasse 'logical', an denen ein `TRUE` für die gefragte Eigenschaft steht. In diesem Fall wird die Spalte `week` des Datensatzes `df.ratten` anhand der Wochen indiziert (die Funktion `which()` sucht nach den Elementen, bei denen die in der Funktion definierten Wert auf `TRUE` steht), so dass der Mittelwertvergleich für jede Woche einzeln durchgeführt werden kann. Würde diese Indizierung nicht vorgenommen, würde bei der Auswahl der Gliederungsvariable für die Buchstabendarstellung (in `glht()`) ein Vergleich über alle Wochen gemacht.

Für die Vergleiche der anderen Variablen, wie der Differenz der letzten und ersten Woche, der Steigung oder dem adjustierten Mittel, müsste der Datenframe `df.ratten` um diese Werte ergänzt werden, ebenfalls über `which()` selektiert werden und dann wie gezeigt weiter behandelt werden.

14 Einführung in multivariate Verfahren

In diesem kompletten Kapitel wird das Paket 'vegan' benötigt. Dabei handelt es sich um ein Paket das von Ökologen entwickelt wurde, um synökologische Zusammenhänge statistisch erfassen zu können. In diesem Paket finden sich sowohl Dissimilaritätsanalysen als auch Clusteranalysen und ↑Ordinationsverfahren (Hauptkomponentenanalyse).

14.1 Zwei einführende Beispiele

```
margarine.quan <- c(4.500, 5.167, 5.069, 3.800, 3.444, 3.500, 5.250,
 5.857, 5.083, 5.273, 4.500, 4.000, 4.225, 3.824, 5.400, 5.056, 3.500,
 3.417, 4.429, 4.083, 3.600, 4.000, 4.375, 3.833, 4.765, 3.800, 3.778,
 3.875, 4.583, 4.929, 4.667, 3.909, 4.200, 3.875, 3.833, 3.438, 2.400,
 3.765, 4.000, 3.917, 3.857, 4.000, 4.091, 3.900, 3.250, 2.167, 4.235,
 5.000, 3.944, 4.625, 4.333, 4.071, 4.000, 4.091, 3.700, 3.750, 3.750,
 4.471, 5.000, 5.389, 5.250, 4.417, 5.071, 4.250, 4.091, 3.900, 4.000,
 3.273, 3.765, 5.000, 5.056, 5.500, 4.667, 2.929, 3.818, 4.545, 3.600,
 2.000, 1.857, 1.923, 4.000, 5.615, 6.000, 3.250, 2.091, 1.545, 1.600,
 1.500, 4.625, 3.750, 3.529, 4.000, 4.222, 4.750, 4.500, 4.571, 3.750,
 3.909, 3.500, 4.125, 3.417, 3.529, 4.600, 5.278, 5.375, 3.583, 3.786,
 4.167, 3.818, 3.700)
margarine.qual <- c(1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0,
 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0)
margarine.names <- c("Sanella", "Homa", "SB", "Delicado", "Hollbutt",
 "Weihbutt", "DuDarfst", "Becel", "Botteram", "Flora", "Rama")
```

14.2 Distanzen und Ähnlichkeit

`vegdist()`: berechnet Dissimilaritätsmaße.

`x =` : Matrix der Daten.

`method =` : gibt den Typ des Dissimilaritätsmaße an. Die Dokumentation gibt eine ausführliche Übersicht über die Dissimilaritätsmaße an (inklusive mathematischer Formel).

`diag = FALSE/TRUE`: berechnet die Diagonale.

`upper = FALSE/TRUE`: gibt an, ob die Werte über der Diagonalen ausgegeben werden sollen.

Da es sich bei dem Paket um ein ökologisch motiviertes Paket handelt, werden sich hier sicher nicht die Dissimilaritätsmaße finden, die wichtig für andere Disziplinen der Wissenschaft sind. Daher existiert die Funktion `designdist()`, in der man alle möglichen Dissimilaritätsmaße definieren kann.

Die Funktion `decostand()` kann einige Standardisierungen durchführen, die für Ökologen interessant sind. Andere Standardisierungen sind schon standardmäßig in R enthalten.

14.2.1 Euklidische Distanz

```
x <- matrix(margarine.quan, 11)
```

```
vegdist(x, method="euclidean", upper=TRUE)**2
            1         2         3         4         5         6         7
 1             3.779798  3.805458 15.197500 21.441917 25.484375  4.881973
 2   3.779798            6.299324 23.929248 30.498747 38.583673 10.839495
 3   3.805458  6.299324           14.175908 25.003747 28.964333  3.993785
 4  15.197500 23.929248 14.175908            6.495567 11.881875 11.691623
 5  21.441917 30.498747 25.003747  6.495567            3.605542 16.410046
 6  25.484375 38.583673 28.964333 11.881875  3.605542           15.886848
 7   4.881973 10.839495  3.993785 11.691623 16.410046 15.886848
 8   6.024971  8.073059  3.455255 18.361621 26.956786 32.336096  6.422444
 9   2.267705  5.317423  1.098369 15.928905 25.333702 29.999080  5.155550
 10  2.908833  6.162373  2.356723 16.520483 25.905942 28.194958  3.825300
 11  2.112500  3.384248  1.735908 17.030000 26.767767 32.271875  6.932023
            8         9        10        11
 1   6.024971  2.267705  2.908833  2.112500
 2   8.073059  5.317423  6.162373  3.384248
 3   3.455255  1.098369  2.356723  1.735908
 4  18.361621 15.928905 16.520483 17.030000
 5  26.956786 25.333702 25.905942 26.767767
 6  32.336096 29.999080 28.194958 32.271875
 7   6.422444  5.155550  3.825300  6.932023
 8             3.394606  6.376058  6.021621
 9   3.394606            1.564432  1.117505
 10  6.376058  1.564432            2.152283
 11  6.021621  1.117505  2.152283
```

Hier wurden die Werte, nach Umformen in ein dist-Objekt, quadriert. Bei der quadrierten Euklidischen Distanz handelt es sich um ein Dissimilaritäsmaß, dass in diesem Paket nicht als Standard definiert ist. Mit der Funktion `desingdist()` kann man jedoch beliebig viele neue Dissimilaritätsmaße definieren.

`designdist()`: dient zum definieren eigener Dissimilaritätsmaße.

`x =` : Die Daten, die mit dem neuen Dissimilaritäsmaß berechnet werden sollen.

`method =` : gibt die Gleichung an, nach der das Dissimilaritäsmaß berechnet werden soll.

`terms =` : gibt an, wie die Terme für `method =` gefunden werden sollen.

`abcd = FALSE/TRUE`: gibt an, ob eine 2x2 Kontigenztafel herangezogen werden soll, um die Gleichung zu erstellen.

Zur Vereinfachung kann die Gleichung in `method =` mit fünf verschiedenen Variablen definiert werden. Die meisten Dissimilaritätsmaße lassen sich ebenfalls auf diese fünf Variablen reduzieren. Die Variablen können je nach Methode in `terms =` unterschiedlich berechnet werden.

Tabelle 14.1: Terme der Variablen in Form von R-Code

Variable		`terms = "minimum"`	`"quadratic"`
J	gemeinsamer Term	`sum(pmin(x, y))`	`sum(x*y)`
A	Einzelterm für Element 1	`sum(x)`	`sum(x**2)`
B	Einzelterm für Element 2	`sum(y)`	`sum(y**2)`
N	Anzahl Zeilen (sehr selten)	diverse	diverse
P	Anzahl Spalten (selten)	diverse	diverse

Bei binären Daten, also bei `terms = "binary"`, kommt mit den Methoden der beiden anderen Möglichkeiten das selbe heraus, da es sich nur noch um die Zahlen 0 und 1 handelt, die bei diesen unterschiedlichen mathematischen Operationen das selbe Verhalten zeigen.

Will man nun die quadrierte Euklidische Distanz berechnen, muss man die diesem Index zugrunde liegende mathematische Formel in Form der Variablen erfassen. Bei der mathematischen Formel

$$ED^2 = \sum_{k=1}^{p}(x_{ik} - x_{jk})^2$$

handelt es sich um eine binomische Formel, also um eine quadratische Formel. Daher verweden wir das Argument `terms = "quadratic"` und müssen die binomische Formel so umstellen, dass wir es in den für R verständlichen Variablen darstellen können. Dies erhalten wir, wenn wir die binomische Formel auflösen

$$ED^2 = \sum_{k=1}^{p}(x_{ik} - x_{jk})^2 = \sum_{k=1}^{p} {x_{ik}}^2 + \sum_{k=1}^{p} {x_{jk}}^2 - 2 * \sum_{k=1}^{p}(x_{ik} * x_{jk})$$

Mit x_i als erstes zu vergleichendes Element `1` und x_j als zweites zu vergleichendes Element `2` und k als Nummer des Merkmals haben wir mit x_{ik} den Vektor `x` und mit x_{jk} den Vektor `y` jeweils eines Elements über alle seine Merkmale. In R-Code geschrieben ist das genau `sum(x**2)+sum(y**2)-2*sum(x*y)`, also `A+B-2*J`.

```
designdist(x, method="(A+B-2*J)", terms="quadratic")
           1         2         3         4         5         6         7
 2   3.779798
 3   3.805458  6.299324
 4  15.197500 23.929248 14.175908
 5  21.441917 30.498747 25.003747  6.495567
 6  25.484375 38.583673 28.964333 11.881875  3.605542
 7   4.881973 10.839495  3.993785 11.691623 16.410046 15.886848
 8   6.024971  8.073059  3.455255 18.361621 26.956786 32.336096  6.422444
 9   2.267705  5.317423  1.098369 15.928905 25.333702 29.999080  5.155550
 10  2.908833  6.162373  2.356723 16.520483 25.905942 28.194958  3.825300
 11  2.112500  3.384248  1.735908 17.030000 26.767767 32.271875  6.932023
           8         9        10
 2
 3
 4
 5
 6
 7
 8
 9   3.394606
 10  6.376058  1.564432
 11  6.021621  1.117505  2.152283
```

Eine weitere und für die Euklidische Distanz einfacher anzuwendende Methode existiert mit der Funktion `dist()`.

```
dist(x)**2
           1         2         3         4         5         6         7
 2   3.779798
 3   3.805458  6.299324
 4  15.197500 23.929248 14.175908
 5  21.441917 30.498747 25.003747  6.495567
 6  25.484375 38.583673 28.964333 11.881875  3.605542
 7   4.881973 10.839495  3.993785 11.691623 16.410046 15.886848
 8   6.024971  8.073059  3.455255 18.361621 26.956786 32.336096  6.422444
 9   2.267705  5.317423  1.098369 15.928905 25.333702 29.999080  5.155550
 10  2.908833  6.162373  2.356723 16.520483 25.905942 28.194958  3.825300
 11  2.112500  3.384248  1.735908 17.030000 26.767767 32.271875  6.932023
           8         9        10
 2
 3
 4
 5
 6
 7
 8
 9   3.394606
 10  6.376058  1.564432
 11  6.021621  1.117505  2.152283
```

Bei dieser Funktion kann ebenfalls die Methode (`method =`) verändert werden.

14.2.2 Binäre Daten

Bei den binären Daten bietet die Funktion `designdist()` ebenfalls die Möglichkeit die Koeffizienten über eine Kontingenztafel zu definieren. Dazu muss man das Argument `abcd =` auf `TRUE` setzen. Dann kann man auf die Variablen `a, b, c` und `d` zugreifen. Dies vereinfacht die Erfassung neuer Koeffizienten ungemein, da die Koeffizienten binärer Daten in der Literatur meist in diesem „Format“ dargestellt sind.

Tabelle 14.2: Variablen der Kontigenztafel-Notation in Relation zu den „normalen“ Variablen

`abcd`		`method =`
a	1 für beide Elemente	`J`
b	1 für Element 1, 0 für Element 2	`A-J`
c	0 für Element 1, 1 für Element 2	`B-J`
d	0 für beide Elemente	`P-A-B+J`

Der *Simple-Matching* Koeffizient ist definiert als

$$S = \frac{a+d}{a+b+c+d}$$

Dies kann quasi direkt in das `method =` -Argument von `designdist()` eingefügt werden.

```
designdist(x, method="(a+d)/(a+b+c+d)", terms="binary", abcd=TRUE)
      1   2   3   4   5   6   7   8   9  10
 2  0.5
 3  0.5 0.6
 4  0.7 0.2 0.4
 5  0.2 0.5 0.5 0.5
 6  0.4 0.5 0.3 0.5 0.6
 7  0.3 0.8 0.6 0.2 0.5 0.3
 8  0.6 0.3 0.5 0.7 0.4 0.4 0.5
 9  0.4 0.7 0.5 0.5 0.6 0.6 0.5 0.4
 10 0.8 0.3 0.7 0.7 0.2 0.4 0.3 0.8 0.4
 11 0.9 0.6 0.6 0.6 0.3 0.3 0.4 0.5 0.5 0.7
```

14.2.3 gemischte Daten

Um Ähnlichkeitsmaße bei gemischten Daten berechnen zu können, muss man auf die Funktion `daisy()` im Paket 'cluster' zurückgreifen.

14.3 Clusteranalyse

`hclust()`: erstellt die Clusteranalyse eines Datensatzes, der mit einem Dissimlaritätsmaß berechnet wurde.

`d =` : Daten mit der Klasse `'dist'` als Output von beispielsweise `vegdist()`.

`method =` : gibt die Methode an, nach der die Cluster berechnet werden.

```
v <- vegdist(x, method="euclidean")
par(mfrow=c(1, 3))

--------------------------------------------------------------------------

plot(hclust(v, method="average"))
plot(hclust(v, method="complete"))
plot(hclust(v, method="single"))
savePlot("Bild041.png", "png")
```

mit der Funktion `plot()` werden die Dendrogramme geplottet, während mit dem Argument `method =` hier die unterschiedlichen Gruppenähnlichkeitsmaße definiert werden. Wie man sieht, kommt je nach Gruppenähnlichkeitsmaß ein anderes Dendrogramm heraus. Hier sei die Funktion `cutree()` zu erwähnen. Damit kann man das Dendrogramm

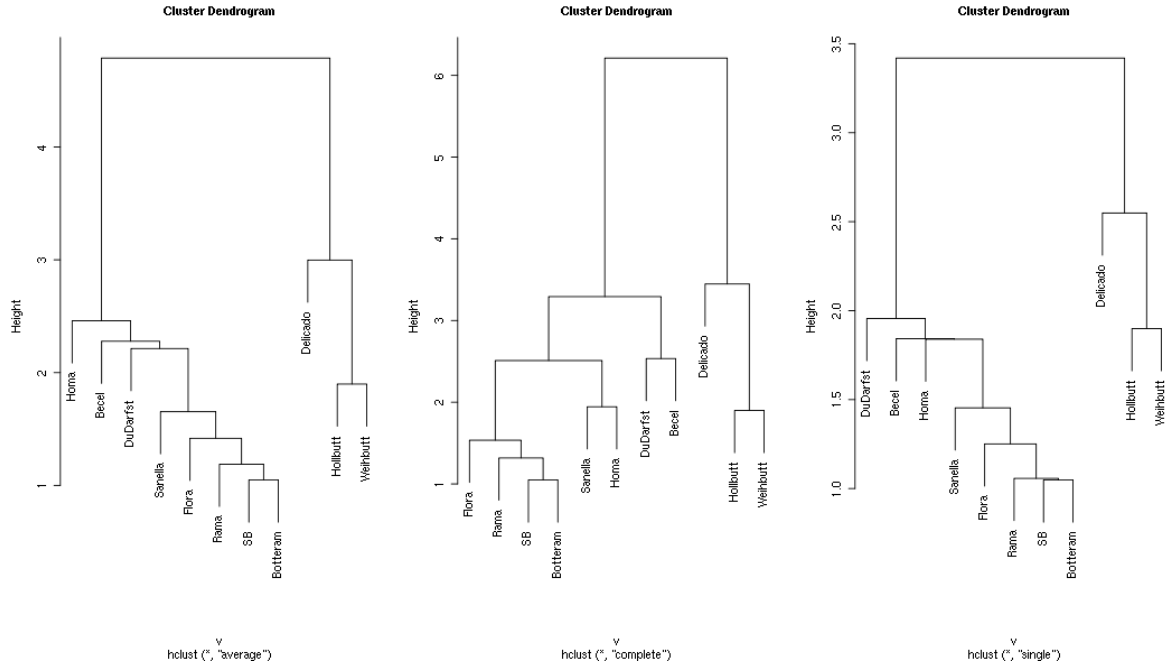


Abbildung 14.1: Dendrogram für die Margarine-Daten (unstandardisierte Daten, Euklidische Distanz

an beliebigen Stellen schneiden, was dazu führt, dass die zu Untersuchenden Einheiten anhand des Dendrograms und des „Schnitts“ in Gruppen eingeteilt werden.

```
tree <- hclust(v, method="average")

-------------------------------------------------------------------------

cutree(tree, k=2)
  Sanella     Homa       SB Delicado Hollbutt Weihbutt DuDarfst    Becel
        1        1        1        2        2        2        1        1
 Botteram    Flora     Rama
        1        1        1
```

Mit der Funktion `mantel()` kann man die Mantel-Statistik zwischen zwei Distanzmatrizen oder der ursprünglichen Distanzmatrix und der für die Clusteranalyse angepassten Distanzmatrix darstellen.

```
mantel(x, v)

 Mantel statistic based on Pearson's product-moment correlation

 Call:
 mantel(xdis = x, ydis = v)

 Mantel statistic r: -0.01897
       Significance: 0.487

 Empirical upper confidence limits of r:
   90%   95% 97.5%   99%
 0.202 0.268 0.353 0.429

 Based on 999 permutations

 Warning message:
 In as.dist.default(xdis) : non-square matrix
```

Hier kann mit dem Argument `method =` die Methode angepasst werden und mit `permutations =` kann man die Anzahl der Permutationen festlegen, mit denen die Signifikanz der Statistik berechnet wird.

14.4 Hauptkomponentenanalyse

Zur Hauptkomponentenanalyse in R kann man die Funktionen `rda()` und `cca()` des Paketes `vegan` heranziehen. Allerdings gibt es noch einige andere Pakete, die diese Aufgabe übernehmen können.

`rda()` führt die ↑Hauptkomponentenanalyse (redundancy analysis) durch, `cca()` führt die ↑kanonische Korrespondenzanalyse (canonical correspondence analysis) durch.

```
schwermetall <- read.table("Schwermetall.txt", header=TRUE, sep="\t",
```

```
 dec=",")
rownames(schwermetall) <- schwermetall[,1]
schwermetall <- schwermetall[,-c(1)]
schwermetall.rda <- rda(schwermetall)
```

```
schwermetall.rda
 Call: rda(X = schwermetall)

                Inertia Rank
 Total         7.416e+09
 Unconstrained 7.416e+09   11
 Inertia is variance

 Eigenvalues for unconstrained axes:
       PC1       PC2       PC3       PC4       PC5       PC6
 6.990e+09 4.253e+08 1.018e+06 3.669e+04 5.971e+02 9.850e+01
       PC7       PC8       PC9      PC10      PC11
 4.070e+01 5.110e+00 4.137e-01 1.566e-01 6.182e-02
```

```
biplot(schwermetall.rda)
 Warning messages:
 1: In arrows(0, 0, g$species[, 1] * arrlen, g$species[, 2] * arrlen,  :
   zero-length arrow is of indeterminate angle and so skipped
 2: In arrows(0, 0, g$species[, 1] * arrlen, g$species[, 2] * arrlen,  :
   zero-length arrow is of indeterminate angle and so skipped
 3: In arrows(0, 0, g$species[, 1] * arrlen, g$species[, 2] * arrlen,  :
   zero-length arrow is of indeterminate angle and so skipped
 4: In arrows(0, 0, g$species[, 1] * arrlen, g$species[, 2] * arrlen,  :
   zero-length arrow is of indeterminate angle and so skipped
 5: In arrows(0, 0, g$species[, 1] * arrlen, g$species[, 2] * arrlen,  :
   zero-length arrow is of indeterminate angle and so skipped
 6: In arrows(0, 0, g$species[, 1] * arrlen, g$species[, 2] * arrlen,  :
   zero-length arrow is of indeterminate angle and so skipped
 7: In arrows(0, 0, g$species[, 1] * arrlen, g$species[, 2] * arrlen,  :
   zero-length arrow is of indeterminate angle and so skipped
 8: In arrows(0, 0, g$species[, 1] * arrlen, g$species[, 2] * arrlen,  :
   zero-length arrow is of indeterminate angle and so skipped
savePlot("Bild042.png", "png")
```

Hier wurden die Daten wieder aus einer Textdatei eingelesen. Diese enthält als Trennzeichen zwischen den Spalten einen Tabulator und als Dezimalzeichen ein Komma. Da die Bezeichnung der Orte nicht mit in die Analyse einfliesen soll, müssen diese entfernt werden. Davor wurde diese Spalte allerdings zur Benennung der Zeilen verwendet.

Mit der Funktion `rda()` wurde dann die Statistik zur Hauptkomponentenanalyse berechnet. Mit der Funktion `biplot()` kann man die Daten in Form eines Biplots ausgeben lassen. Hier sind einige Warnungen zu sehen, die anzeigen, dass einge Pfeile nicht angezeigt werden. Im Plot ist dann zu sehen, dass lediglich Calcium und Eisen

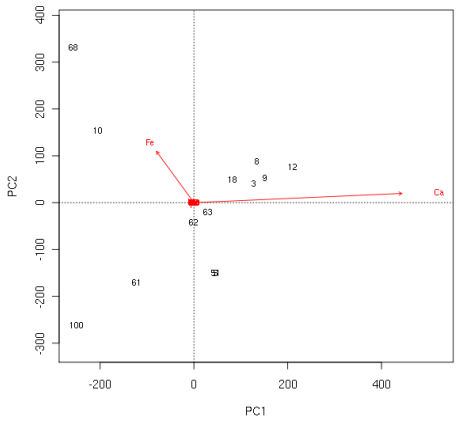


Abbildung 14.2: Biplot der Hauptkomponentenanalyse der unskalierten Schwermetalldaten

abgebildet wurden. Hier ist die Dominanz dieser Werte zu sehen.

```
schwermetall2.rda <- rda(schwermetall, scale=TRUE)
```

```
schwermetall2.rda
 Call: rda(X = schwermetall, scale = TRUE)

               Inertia Rank
 Total              13
 Unconstrained      13   11
 Inertia is correlations

 Eigenvalues for unconstrained axes:
       PC1       PC2       PC3       PC4       PC5       PC6
 6.4072644 4.0192813 0.9976492 0.7149448 0.5389100 0.1834359
       PC7       PC8       PC9      PC10      PC11
 0.0822243 0.0418972 0.0106685 0.0034061 0.0003184
```

```
sum.schwermetall2 <- summary(schwermetall2.rda)
```

```
sum.schwermetall2

 Call:
 rda(X = schwermetall, scale = TRUE)

 Partitioning of correlations:
```

```
              Inertia Proportion
Total              13          1
Unconstrained      13          1

Eigenvalues, and their contribution to the correlations

Importance of components:
                         PC1   PC2    PC3   PC4    PC5    PC6
Eigenvalue             6.407 4.019 0.9976 0.715 0.5389 0.1834
Proportion Explained  0.493 0.309 0.0767 0.055 0.0415 0.0141
Cumulative Proportion 0.493 0.802 0.8788 0.934 0.9752 0.9893
                           PC7     PC8     PC9    PC10     PC11
Eigenvalue             0.08222 0.04190 0.01067 0.00341 0.000318
Proportion Explained  0.00632 0.00322 0.00082 0.00026 0.000020
Cumulative Proportion 0.99567 0.99889 0.99971 0.99998 1.000000

Scaling 2 for species and site scores
* Species are scaled proportional to eigenvalues
* Sites are unscaled: weighted dispersion equal on all dimensions
* General scaling constant of scores:  3.534119


Species scores

       PC1      PC2       PC3      PC4      PC5       PC6
P  -0.1165  0.76241 -0.084741  0.25587  0.51936  0.124482
Fe -0.7879 -0.51013  0.168195  0.06591 -0.06314  0.144487
Mo -0.7301  0.06911 -0.007332  0.62926 -0.06706 -0.023386
OC -0.3422  0.86459  0.079787 -0.27495 -0.01665  0.038180
Ni -0.9208 -0.12727 -0.179310  0.03984 -0.23942  0.038852
CO -0.7853 -0.46731  0.199297 -0.19514  0.20408  0.037053
Cr -0.8400 -0.47194 -0.108488  0.02632 -0.07261  0.058010
Cu -0.8357  0.41853  0.234454 -0.15783  0.03661 -0.007863
PB -0.6440  0.71440 -0.080125  0.07472 -0.12620 -0.037615
Zn -0.4247 -0.01246 -0.857255 -0.19339  0.06867  0.025605
Mn -0.7408 -0.55669  0.133500 -0.14637  0.22380  0.039438
Ca  0.8816 -0.23339 -0.043921  0.05152 -0.09853  0.327257
Cd -0.2861  0.87203  0.130205 -0.11481 -0.23077  0.144327


Site scores (weighted sums of species scores)

         PC1     PC2      PC3      PC4      PC5     PC6
61  -0.95398  2.3642 -0.13813 -0.29089 -0.86985  1.1642
62  -0.01957  0.6125 -0.61708  2.15353  2.14693  0.7919
63   0.50305 -0.2514  1.44383  1.81784 -0.50325 -1.5776
68  -2.22466 -1.7226  0.95361 -0.76600  1.11596  0.5040
3    0.78437 -0.4784  0.07716 -0.33650 -0.26865 -0.1731
5    0.54105 -0.1777 -1.34934 -0.62696  0.46046 -0.7966
51   0.54105 -0.1777 -1.34934 -0.62696  0.46046 -0.7966
8    0.52573 -0.3425 -0.83709 -0.46363  0.07508  0.9105
9    0.87719 -0.5496  0.01677 -0.57148 -0.42792  0.1343
```

```
 10  -1.60702 -0.5251 -1.22668  1.02699 -1.97313 -0.5007
 12   0.98408 -0.1618  1.28841 -0.08382 -0.58814  1.4388
 18   0.56226 -0.1572  0.67812 -0.06169 -0.37615  0.6111
 100 -0.51353  1.5674  1.05977 -1.17043  0.74821 -1.7101
```

```
xlab <- sum.schwermetall2$cont$importance[2, 1]*100
ylab <- sum.schwermetall2$cont$importance[2, 2]*100
```

```
biplot(schwermetall2.rda, xlab=paste("PC1", "[", xlab, "% ]"),
 ylab=paste("PC2", "[", ylab, "% ]"))
savePlot("Bild043.png", "png")
```

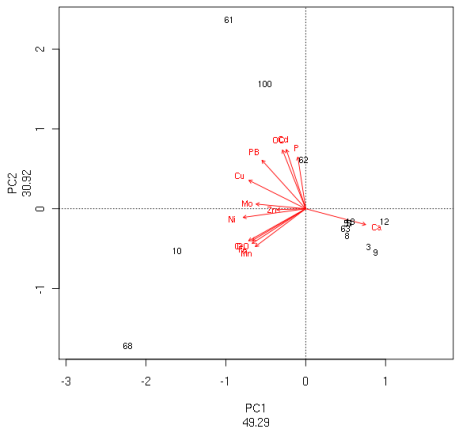


Abbildung 14.3: Biplot der Hauptkomponentenanalyse der skalierten Schwermetalldaten

Nun wurden die Schwermetalldaten skaliert (standardisiert beziehungsweise linear transformiert) und damit auf ein gleiches Niveau gebracht. Der Biplot zeigt die Verteilung, die auch im Biometrieskript zu sehen ist, jedoch ist die x-Achse umgekehrt bezeichnet.

Die Argumente `xlab =` und `ylab =` der Funktion `biplot()` sind schon von `par()` her bekannt und dienen der Beschriftung der Achsen. Dies wurde hier deshalb durchgeführt, um zu zeigen, zu wieviel Prozent die Achse die Varianz erklärt. Zu guter Letzt kann man sich die Zusammenfassung des rda-Objektes ansehen, dort sind alle wichtigen Informationen aufgelistet.

Es ist erwähnt, dass eine Skalierung für `Species` durchgeführt wurde, was dazu führt, dass das Ergebnis nicht von einer Variable (Dimension) dominiert wird. Scores sind die Werte für die n-te Hauptkomponente für 'Sites' bzw. der Anteil an der n-ten Hauptkomponente für 'Species'.

Außerdem existiert in R die Funktion `princomp()`, die ebenfalls eine Hauptkomponentenanalyse berechnet.

R: Erläuterung des Codes

Zuweisungen:

```
CI <- function(data, alpha=.05){
  m <- mean(data)
  t_tab <- qnorm(1-alpha/2)
  n <- length(data)
  x_min <- m-t_tab*sd(data)/sqrt(n)
  x_max <- m+t_tab*sd(data)/sqrt(n)
  return( c(x_min, x_max) )
}
```

Auf die selbe Weise, wie man im Befehlsfenster von R Code eingibt, kann man auch Code in ein R-Skript schreiben. In einer geschweiften Klammer steht Code, der nur in dieser Klammer gilt. Damit ist gemeint, dass die Variablen, die innerhalb der geschweiften Klammer definiert werden, außerhalb nicht als solche erkannt werden.

is.null, if-else, cat:

```
t.test.planning <- function(n=NULL, power=NULL, delta, var,
 alpha=.05){
  if(!is.null(n) & !is.null(power)){
    stop("both, 'n' and 'power' are given, but only one of the two can be
     given")
  }
  if(is.null(n)){
    if(is.null(power)){
      stop("either 'n' or 'power' must be given")
    }
    else{
    q1 <- qnorm(1-alpha/2)
    q2 <- qnorm(power)
    n <- 2*(var/delta**2)*(q1+q2)**2
    cat( "\nn \t=", n )
    cat( "\npower \t=", power )
    cat( "\ndelta \t=", delta )
    cat( "\nvar \t=", var )
    cat( "\nalpha \t=", alpha )
    cat( "\n\n" )
    }
  }
  if(is.null(power)){
    if(is.null(n)){
      stop("either 'n' or 'power' must be given")
```

```
    }
    else{
    q <- qnorm(1-alpha/2)
    power <- sqrt((n*delta**2)/(2*var))- q
    p <- pnorm(power)
    cat( "\npower \t=", p )
    cat( "\nn \t=", n )
    cat( "\ndelta \t=", delta )
    cat( "\nvar \t=", var )
    cat( "\nalpha \t=", alpha )
    cat( "\n\n" )
    }
  }
}
```

Will man eine Variable in eine Formel einbauen, die nur dann verrechnet wird, wenn sie angegeben ist, bietet es sich an, diese in der Variablen-Definition mit `xy = NULL` darzustellen.

Dann kann man die Funktionen `if()` und `else()` verwenden um den Wert einer Variablen auf bestimmte Weise zu überprüfen. Da in diesem Beispiel entweder `n` oder `power` berechnet werden müssen, aber nicht beide, habe ich mich hier für diese Methode entschieden. Ein `!` Im Code bedeutet üblicherweise, dass hier vom Gegenteil (Negation) des Geschriebenen ausgegangen wird. So bedeutet `!=` zum Beispiel „ungleich“.

Auf `if()` folgt immer eine Bedingung und dann was gemacht werden soll, wenn diese Bedingung wahr `(TRUE)` ist. Kann auf diese „Frage“ nicht `TRUE` als Antwort gegeben werden, sondern `FALSE`, kommt zum Tragen, was unter `else` steht.

`cat()` gibt den Inhalt in der dahinter folgenden Klammer aus, dabei kann man festlegen wie und wo der Output ausgegeben werden soll.

missing, stop, is.integer, as.nummeric:

```
weighted.mean.binom <- function(x, w){
  if(missing(w)){
    w <- rep.int(1, length(x))
  }
  else{
    if(length(w) != length(x)){
      stop("'x' and 'w' must have the same length")
    }
  }
  if(is.integer(w)){
    w <- as.numeric(w)
  }
  sum(x * w)/((length(x)-1)*sum(w))
}
```

`missing()` überprüft, ob der Wert einer Variablen vorhanden ist, oder nicht.

`stop()` führt dazu, dass die Berechnung unterbrochen wird und die in der Klammer folgende Nachricht als Grund angezeigt wird. Das macht besonders dann Sinn, wenn

man mit Gleichungen arbeitet, die aus mathematischer Sicht falsch sein, oder in der Praxis nicht vorkommenden Werte ergeben können. `is.integer()` und `is.numeric()` überprüfen die Klasse eines Objektes. `is.numeric()` fragt zum Beispiel, ob die Variable numerisch ist. `as.numeric()` wandelt die Variable dann in eine numerische um, falls diese nicht numerisch ist.

Danksagung

Ich möchte mich bei all den Menschen Bedanken, die mir während der letzten zwei Jahre geholfen haben, dieses Skript zu seiner Fertigstellung zu führen.

An erster Stelle ist das natürlich Herr Piepho, der der Initiator für alles war, die vertraglichen Rahmenbedingungen unkompliziert und zuvorkommend geschaffen hat, mich mit guten Hinweisen und guter Literatur versorgt hat, meine sonstigen Fragen immer gut und verständlich beantworten konnte und vor allem daran geglaubt hat, dass ich diese Arbeit tatsächlich zu Ende führe. Außerdem habe ich es sehr geschätzt, dass ich alle meine Ideen einfließen lassen konnte, alles selbst gestalten durfte und so sehr selbstständig arbeiten konnte.

An zweiter Stelle steht Herr Schützenmeister, der mit seiner Hilfe zu R-bezogenen Fragen immer dann parat war, wenn es etwas komplizierter wurde, weil die R-Dokumentation nicht das lieferte was ich brauchte um den Zusammenhang zu verstehen oder ich diesen Zusammenhang trotzdem nicht verstanden habe. Damit steckt sein zeitlicher und geistiger Aufwand in sehr vielen Methoden, Funktionen und Beispielen. Ohne diese Hilfe hätte ich entweder wesentlich länger gebraucht oder vieles einfach nicht geschafft.

Ich möchte mich auch bei meinen Eltern, Freunden und allen anderen Mitmenschen bedanken, die mir ihre Meinung und Erfahrungen zu diesem Themenkomplex mitgeteilt haben oder mir sonstige Tipps gegeben haben.

Selbstverständlich gibt es viele Menschen, die einen Anteil an diesem Skript haben und sei es auch nur deswegen, weil ich ihnen davon erzählt habe. Jeder Erfahrungsaustausch hat sicherlich seine Spuren in dieser Arbeit hinterlassen.

Als persönlichen Eindruck, kann ich sagen, dass sich diese Erfahrung auf jeden Fall gelohnt hat und mir einige Perspektiven für die Zukunft aufgezeigt hat, für die ich sehr dankbar bin! Außerdem hat mir der Einblick in die Statistik als Auswertungsinstrumentarium der Wissenschaft eine neue mögliche Betrachtungsweise der Dinge dargelegt über die es sich nachzudenken lohnt (insbesondere die Logik hinter Modellen...). Ich kann nur jedem empfehlen, sich ebenfalls auf eine solche Erfahrung einzulassen, sollte sich die Möglichkeit dazu ergeben.

Datenübersicht

Hier kann man alle Daten finden, die ich für 'R für Statistik' und für dieses Skript aus Herrn Piephos Skripten abgeschrieben habe. Dabei handelt es sich um eine direkte Kopie meines R-Skriptes `Daten.R`. Durch Kopieren und Einfügen kann man dieses R-Skript übernehmen. Einige der Daten müssen allerdings in eine andere Datei geschrieben werden, beim entsprechenden Beispiel ist aber niedergeschrieben wie dabei zu verfahren ist.

```
# Maisfeld mit 100'000 Pflanzen, daraus eine Stichprobe von 50 Pflanzen
Mais_50 <- c(175, 172, 179, 167, 163, 154, 163, 164, 157, 177, 186, 165, 175, 194, 176, 162, 166, 169, 170, 181, 168, 166,
180, 164,  179, 170, 150, 192, 170, 173, 170, 150, 174, 164, 182, 188, 157, 165, 172, 168, 179, 179, 164, 162, 178, 162,
182, 171, 182, 183)
Mais50 <- sort(Mais_50)


# On-Farm Versuch, bei dem auf insgesamt 28 Höfen drei Düngemethoden an Sorghum getestet wurden.
# 1. Kein Dünger, 2. NPK (Stickstoff-Phosphor-Kalium), 3. DAP (Di-Ammonium-Phosphat)
# Erträge in dt/ha
Farm <- c(1:28)
Kein <- c(0.30, 0.34, 0.39, 0.40, 0.40,  0.42, 0.48, 0.54, 0.56, 0.58, 0.62, 0.68, 0.74, 0.74, 0.78, 0.82, 0.96, 1.02, 1.06,
1.10, 1.44, 1.60, 1.68, 2.40, 2.40, 2.56, 3.60, 4.50)
NPK <- c(0.80, 1.12, 1.12, 1.60, 2.80, 1.14, 3.20, 1.34, 1.20, 1.22, 1.40, 2.24, 1.54, 1.52, 1.46, 1.60, 1.60, 1.74, 1.40,
1.44, 4.16, 2.00, 4.80, 4.48, 9.60, 5.28, 4.80, 5.50)
DAP <- c(1.64, 1.38, 1.70, 2.80, 2.40, 1.56, 1.92, 1.46, 1.66, 1.60, 2.30, 2.76, 1.66, 2.42, 1.80, 2.50, 2.06, 2.16, 1.74,
1.74, 3.84, 2.40, 2.56, 3.84, 3.84, 3.24, 5.60, 6.75)
sorghum <- data.frame(Farm, Kein, NPK, DAP)


# Ertrag einer Weizensorte auf 13 Parzellen
Weizen_13 <- c(44, 40, 18, 20, 45, 26, 55, 55, 20, 46, 15, 8, 41)


# Phalombe Projekt in Südost-Malawi
# 1. traditionelle Sorte (LM), 2. verbesserte Sorte (CCA)
# Erträge in dt/ha
LM <- c(2.2, 2.2, 1.9, 1.2, 1.3, 0.9, 1.0, 0.5, 1.8, 1.1, 1.6, 1.0, 1.6, 0.6)
CCA <- c(3.5, 2.0, 2.9, 0.4, 0.6, 0.5, 0.6, 0.3, 2.2, 0.7, 0.9, 0.3, 1.1, 0.3)
phalombe <- data.frame(LM, CCA)


# Tomatenerträge bei zwei verschiedenen Düngerbehandlungen
# Dünger A wird auf 5 Parzellen getestet, Dünger B auf 6. Die Verteilung der Düngemethoden erfolgte zufällig
A <- c(29.9, 11.4, 25.3, 16.5, 21.1, NaN)
B <- c(26.6, 23.7, 28.5, 14.2, 17.9, 24.3)
Tomaten <- data.frame(A, B)


# Erträge 5 verschiedener Sorten(A, B, C, D, E)
# 4 Wiederholungen
# Erträge in dt/ha
# randomisiert
ertrag <- c(31, 32, 37, 32, 21, 23, 25, 19, 27, 29, 34, 34, 34, 32, 31, 27, 24, 23, 27, 26)
sorte.a <- gl(5, 4, labels=LETTERS[1:5])
sorte.b <- c(1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5)
df.ertrag_sorte.a <- data.frame(ertrag, sorte.a)
df.ertrag_sorte.b <- data.frame(ertrag, sorte.b)
lm.ertrag_sorte.a <- lm(ertrag ~ sorte.a, df.ertrag_sorte.a)
lm.ertrag_sorte.b <- lm(ertrag ~ factor(sorte.b), df.ertrag_sorte.b)

# Anzahl von Betrieben klassifiziert nach Besitzform und Bodentyp
# Audubon County (Iowa)
Bf <- c(36, 67, 49, 31, 60, 49, 58, 87, 80)
m <- matrix(Bf, nrow=3, byrow=TRUE, dimnames=list("Bodentyp"=c("I", "II", "III"), "Besitzform"=c("Eigenbesitz", "Pacht",
 "Gemischt")))


# Regendaten
# Regenmenge und Ertrag wurden für 26 Jahre erhoben
regen <- c(177, 96, 144, 105, 111, 135, 209, 161, 246, 108, 137, 71, 119, 108, 132, 89, 147, 98, 106, 123, 156, 191, 162,
235, 147, 110)
ernte <- c(26.2, 25, 32.6, 26.6, 19.6, 20.4, 29.2, 33.8, 26.6, 22.6, 24.2, 16.6, 29.8, 19.4, 30.6, 16.4, 30.4, 19.2, 20.2,
25.6, 31, 35.8, 31.6, 33.6, 30.4, 30.2)
```

```
# Leucin
# zur inversen Regression
# bekannte Leucinkonzentrationen (leu), die zu einer bestimmten colorimetrischen Extiktion (ext) führen. Zur Kalibrierung
eines Gerätes
leu <- c(.02, .05, .10, .30, .40, .50, .60)
ext <- c(.08, .15, .29, .88, 1.13, 1.42, 1.69)
df.ext_leu <- data.frame(ext, leu)
lm.ext_leu <- lm(ext~leu)


# Reissorte IR8
# Licht-Transmissions-Rate (ltr) in Abhängigkeit des Blattflächenindex (lai)
ltr <- c(75, 72, 42, 29, 27, 10, 9, 5, 2, 2, 1, 0.9)
lai <- c(0.5, 0.6, 1.8, 2.5, 2.8, 5.45, 5.6, 7.2, 8.75, 9.6, 10.4, 12)
df.ltr_lai <- data.frame(ltr, lai)
lm.ltr_lai <- lm(ltr~lai)


# Zwiebeln
# Einzelpflanzenertrag (epe) in Abhängigkeit von Pflanzdichte (pfld)
epe <- c(272.15, 235.23, 180.47, 177.31, 141.28, 169.39, 138.17, 171.81, 112.02, 156.09, 137.29, 154.10, 124.17, 146.28,
105.47, 139.24, 148.31, 110.44, 90.72, 102.62, 107.36, 92.66, 96.52, 94.71, 99.86, 93.37, 89.78, 69.34, 73.74, 75.17, 72.98,
79.94, 79.13, 70.93, 60.99, 74.09, 49.45, 56.65, 47.84, 40.03, 38.70)
pfld <- c(18.78, 21.25, 23.23, 27.18, 30.15, 31.67, 32.12, 32.62, 32.62, 33.07, 37.07, 38.55, 39.54, 39.54, 41.02, 42.50,
43.98, 45.47, 49.92, 50.90, 53.87, 57.82, 61.78, 61.78, 63.75, 67.71, 71.66, 77.59, 80.56, 86.49, 88.46, 89.45, 90.93,
92.91, 101.81, 103.78, 115.15, 123.06, 144.31, 155.68, 158.15)


# Enzymtest
# Enzym soll Glucose (glu) und Mannose (man) abbauen, die Reaktionsgeschwindigkeit (10^3 min^-1) wird ermittelt.
conz <- c(.1, .2, 1, 3, 5)
glu <- c(.15, .256, .6, .77, .818)
man <- c(.082, .15, .45, .67 ,.75)


# LD_50 eines Insektizids
# Dosis (dose) des Insektizids und dabei sterbender Larven (ld) bei einer Probengröße von m Larven
dose <- c(0.375, 0.75, 1.5, 3, 6, 12, 24)
ld <- c(0, 1, 8, 11, 16, 18, 20)
m <- 20
mort <- ld/m


# Hasendaten
# Alter (age) in Tagen und Gewicht (lens) der Augenlinse wurden erfasst.
age <- c(15, 15, 15, 18, 28, 29, 37, 37, 44, 50, 50, 60, 61, 64, 65, 65, 72, 75, 75, 82, 85, 91, 91, 97, 98, 125, 142, 142,
147, 147, 150, 159, 165, 183, 192, 195, 218, 218, 219, 224, 225, 227, 262, 262, 267, 246, 258, 276, 185, 300, 301, 305, 312,
317, 338, 347, 354, 357, 375, 394, 516, 535, 554, 591, 648, 660, 705, 723, 756, 768, 860)
lens <- c(21.66, 22.75, 22.30, 31.25, 44.79, 44.55, 50.25, 46.88, 52.03, 63.47, 61.13, 81.00, 73.09, 79.09, 79.51, 65.31,
71.90, 86.10, 94.10, 92.50, 105.00, 101.70, 102.90, 110.00, 104.30, 134.90, 130.68, 140.58, 155.30, 152.20, 144.50, 142.15,
139.81, 153.22, 145.72, 161.10, 174.18, 173.30, 173.54, 178.86, 177.68, 173.73, 159.98, 161.29, 187.07, 176.13, 183.40,
186.26, 189.66, 186.09, 186.70, 186.80, 195.10, 216.41, 203.23, 188.38, 189.70, 195.31, 202.63, 224.82, 203.30, 209.70,
233.90, 234.70, 244.30, 231.00, 242.40, 230.77, 242.57, 232.12, 246.70)


# Gesundheitszustand von Laborratten
# Beurteilung der Rangfolge ihrer Mangelernährung von 2 Personen
p1 <- c(4, 1, 6, 5, 3, 2, 7)
p2 <- c(4, 2, 5, 6, 1, 3, 7)


# Stickstoffmenge (n) gegen Wurzelmassenertrag (wme)
n <- c(0, 0, 0, 35, 35, 35, 70, 70, 70, 105, 105, 105, 140, 140, 140)
wme <- c(9.9, 7.8, 10.7, 20.3, 22.6, 23.9, 27.5, 30.3, 29.2, 31.4, 27.2, 33.4, 28.1, 25.7, 31.9)


# Das Endgewicht(end) [g] von 35 Ratten in Abhängigkeit vom Anfangsgewicht(anfang) [g] und vom
# Futterverzehr(futter) [g]
anfang futter end
55.8 289 114.8
45.8 316 109.7
48.1 234 111.3
43.3 299 126.0
50.1 353 144.7
40.1 298 121.0
47.1 303 125.2
51.0 312 113.7
53.7 333 138.5
41.2 280 105.8
40.2 287 117.7
46.4 338 140.0
45.9 298 117.1
38.0 302 103.0
56.0 355 137.3
32.4 307 109.7
37.5 342 136.3
45.9 310 121.2
40.7 280 104.5
36.4 283 104.0
46.9 305 120.2
```

```
42.2 296 120.5
43.4 290 118.9
45.0 224 126.4
43.8 353 125.4
47.8 282 109.4
50.4 288 121.2
37.9 266 109.2
46.0 318 141.1
42.8 335 131.3
50.7 304 128.5
59.6 296 138.8
43.8 292 109.0
65.4 320 113.7
39.3 305 116.2


# Zementdaten; x1 = Tri-Caclzium-Aluminat, x2 = Tri-Calzium-Silikat, x3 =
# Tetra-Calzium-Ferralit, x4 = Di-Calzium-Silikat,  y = Wärmeentwickung [Kalorien] g-1 Zement
x1 x2 x3 x4 y
7 26 6 60 78.5
1 29 15 52 74.3
11 56 8 20 104.3
11 31 8 47 87.6
7 52 6 33 95.9
11 55 9 22 109.7
3 71 17 6 102.7
1 31 22 44 72.5
2 54 18 22 93.1
21 47 4 26 115.9
1 40 23 34 83.8
11 66 9 12 113.3
10 68 8 12 109.4


# Margarinedaten; Absatz = Verkaufte Menge [Kartons Gebiet-1], Preis = [DM Karton-1], Werbung =
# [DM Gebiet-1],  Besuche = Zahl der Besuche eines Vertreters
Absatz Preis Werbung Besuche
2585 12.5 2000 109
1819 10 550 107
1647 9.95 1000 99
1496 11.5 800 70
921 12 0 81
2278 10 1500 102
1810 8 800 110
1987 9 1200 92
1612 9.5 1100 87
1913 12.5 1300 79
2118 8.5 1550 75
1438 12 550 106
1834 9.5 1980 66
1869 9 1600 80
1574 7 500 90
2597 11 2000 120
2026 10 1680 95
2016 9.5 1700 92
1566 10 1400 65
2169 13 1800 90
1996 11 1600 76
2591 8 2000 89
2604 8.5 1800 108
1277 10 460 78
1789 9 800 88
1824 11 1460 87
1813 12 1300 103
1513 11.5 600 89
1172 13 750 68
1987 9 900 106
2056 10.5 1250 96
1513 9 850 78
1756 12.5 950 86
2007 13 1500 125
2079 11 1850 109
1664 9.9 1200 60
1699 12.5 1600 79


# Kalkgabe als Einflussfaktor auf Weizenertrag
kalk <- c(0, 2, 4, 6, 8)
weizen <- c(44.4, 54.6, 63.8, 65.7, 68.9)


# Daten zu Kapitel 6.12.6
N Eb_510
1.100 7.6667
1.218 10.6667
1.230 13.6667
1.330 12.6667
1.410 9.6700
1.430 12.6667
1.440 9.6700
```

1.591 9.0000
1.591 7.0000
1.637 9.6700
1.648 9.0000
1.830 11.0000
1.982 3.2500
2.082 6.2500
2.120 7.0000
2.122 5.2500
2.150 10.0000
2.183 3.2500
2.217 3.2500
2.232 4.2500
2.336 7.0000
2.420 5.0000
2.458 11.0000
2.478 3.2500
2.531 2.0000
2.534 3.2500
2.564 2.2500
2.590 4.0000
2.625 2.2500
2.652 5.0000
2.690 2.0000
2.722 1.2500
2.817 4.2500
2.834 3.2500
2.863 1.2500
2.942 3.2500
2.984 1.2500
2.989 1.5000
3.042 3.7500
3.094 4.2500
3.121 0.5000
3.124 1.2500
3.146 1.2500
3.173 2.7500
3.199 2.2500
3.237 2.2500
3.264 1.2500
3.311 1.5000
3.342 0.5000
3.346 1.5000
3.363 2.5000
3.366 3.2500
3.430 2.2500
3.443 2.7500
3.463 0.2500
3.469 1.7500
3.495 1.5000
3.503 1.2500
3.504 2.2500
3.538 3.2500
3.600 0.2500
3.612 1.5000
3.618 5.5000
3.632 4.5000
3.641 5.5000
3.653 0.5000
3.653 0.2500
3.671 0.2500
3.675 0.5000
3.708 1.7500
3.729 0.2500
3.793 2.7500
3.821 0.2500
3.912 0.7500
3.919 1.5000
3.977 0.5000
4.002 3.5000
4.009 1.2500
4.014 0.2500
4.024 0.5000
4.033 0.5000
4.041 0.7500
4.049 1.2500
4.052 1.2500
4.053 0.7500
4.063 1.5000
4.087 1.5000
4.097 3.5000
4.130 0.2500
4.160 0.7500
4.205 0.7500
4.213 0.5000
4.264 1.7500
4.273 1.5000
4.275 0.7500
4.279 1.5000
4.331 1.2500

```
4.371 2.5000
4.377 0.2500
4.394 0.2500
4.401 1.2500
4.407 1.2500
4.429 0.7500
4.483 0.2500
4.512 2.5000
4.537 0.7500
4.593 1.0000
4.600 1.7500
4.625 0.2500
4.650 0.2500
4.670 2.7500
4.680 0.7500
4.684 0.7500
4.695 1.7500
4.699 0.2500
4.710 1.2500
4.722 0.7500
4.750 1.5000
4.753 0.0000
4.759 0.2500
4.760 1.7500
4.764 0.2500
4.768 1.7500
4.772 0.7500
4.773 0.7500
4.792 0.2500
4.793 1.0000
4.797 2.2500
4.826 1.7500
4.836 1.0000
4.875 1.2500
4.876 2.2500
4.877 1.0000
4.880 2.2500
4.880 3.0000
4.893 2.0000
4.894 0.0000
4.915 1.0000
4.925 0.7500
4.932 2.0000
4.935 0.0000
4.950 2.7500
4.982 2.0000
4.987 1.0000
4.988 0.0000
4.994 1.7500
5.002 0.2500
5.029 0.2500
5.058 0.7500
5.071 1.2500
5.082 1.2500
5.097 1.7500
5.112 1.0000
5.119 0.7500
5.137 0.0000
5.189 1.2500
5.293 1.0000
5.303 0.2500


# Grünlanddaten
zeit <- c(9, 14, 21, 28, 42, 57, 63, 70, 79)
schnitt <- c(8.93, 10.80, 18.59, 22.33, 39.35, 56.11, 61.73, 64.62, 67.08)


# Anzahl Unkräuter je Parzelle in einem vollständig randomisierten Verusch zum Vergleich von drei Herbiziden (A, B, C) und
# einer Kontrolle (D)
a <- c(4, 5, 2, 5, 4, 1)
b <- c(8, 11, 9, 12, 7, 7)
c <- c(25, 28, 20, 15, 14, 30)
d <- c(33, 21, 48, 18, 53, 31)


# Blockversuch, bei dem Ertrag von Reissorte IR8 bei 6 verschiedenen Aussaatstärken gemessen wurde
Saatstärke <- c(25, 25, 25, 25, 50, 50, 50, 50, 75, 75, 75, 75, 100, 100, 100, 100, 125, 125, 125, 125, 150, 150, 150, 150)
Block <- c(1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)
IR8 <- c(5113, 5398, 5307, 4678, 5346, 5952, 4719, 4264, 5272, 5713, 5483, 4749, 5164, 4831,
4986, 4410, 4804, 4848, 4432, 4748, 5254, 4542, 4919, 4098)
IR8.2 <- c(NA, 5398, 5307, 4678, 5346, 5952, 4719, 4264, 5272, 5713, 5483, 4749, 5164, 4831, 4986, 4410, 4804, 4848, NA,
4748, 5254, 4542, 4919, 4098)


# Lateinisches Quadrat, Bakteriengehalt der Milch von fünf Betrieben (A, B, C, D, E) mit Tageszeit und Tagen
Tag <- factor(rep(c(1, 2, 3, 4, 5), 5))
Tageszeit <- gl(5, 5, labels=c("08:30", "10:00", "11:30", "14:00", "15:30"))
Betrieb <- factor(c("A", "B", "C", "D", "E", "D", "C", "E", "B", "A", "C", "A", "D", "E", "B", "B", "E", "A", "C", "D", "E",
 "D", "B", "A", "C"))
```

```
Bakterienzahl.log <- c(1.9, 1.2, 0.7, 2.2, 2.3, 2.3, 2.0, 0.6, 2.6, 2.3, 2.1, 1.5, 1.7, 1.1, 3.0, 2.9, 1.1, 1.2, 1.8, 2.6,
 1.8, 2.1, 2.0, 2.4, 2.5)
Bakterienzahl.2.log <- c(1.9, 1.2, NA, 2.2, 2.3, 2.3, 2.0, 0.6, 2.6, 2.3, 2.1, 1.5, 1.7, 1.1, 3.0, 2.9, 1.1, 1.2, 1.8, NA,
 1.8, 2.1, 2.0, 2.4, 2.5)


# Zuckerrohrdaten, randomisierte  vollständige Blockanlage mit fünf Blöcken und sechs N-Stufen
Stickstoff <- c(0, 0, 0, 0, 0, 25, 25, 25, 25, 25, 50, 50, 50, 50, 50, 100, 100, 100, 100, 100, 150, 150, 150, 150, 150,
 200, 200, 200, 200, 200)
Block <- c(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
Zuckerrohr <- c(89.49, 54.56, 74.33, 78.20, 61.51, 108.78, 102.01, 105.04, 105.23, 106.52, 136.28, 129.51, 132.54, 132.73,
134.02, 157.63, 167.39, 155.39, 146.85, 155.81, 185.96, 176.66, 178.53, 195.34, 185.56, 195.09, 190.43, 183.52, 180.99,
 205.69)


# Calciumdaten; An vier Blättern einer Rübenpflanze wurden die Calciumwerte in % der Trockenmasse bestimmt, 4 Meßungen
Calcium <- c(3.28, 3.09, 3.03, 3.03, 3.52, 3.48, 3.38, 3.38, 2.88, 2.80, 2.81, 2.76, 3.34, 3.38, 3.23, 3.26)


# Gewächshausversuch; Düngeform und Pflanzenschutzbehandlung als  Einflussfaktoren auf den Ertrag einer Weinsorte
Dünger <- c(rep(1, 9), rep(2, 9))
Schutz <- rep(c(1, 2, 3), 6)
Ertrag <- c(21.3, 22.3, 23.8, 20.9, 21.6, 23.7, 20.4, 21.0, 22.6, 12.7, 12.0, 14.5, 14.9, 14.2, 16.7, 12.9, 12.1, 14.5)


# Gewächshausversuch; verschiedene Dünger und verschiedene chemische Behandlungen als Faktoren für das Wachstum von
# Weizenpflanzen
düng <- c(rep(1, 12), rep(2, 12), rep(3, 12), rep(4, 12))
chem <- rep(c(1, 2, 3, 4), 12)
Weizen <- c(21.4, 20.9, 19.6, 17.6, 21.2, 20.3, 18.8, 16.6, 20.1, 19.8, 16.4, 17.5, 12.0, 13.6, 13.0, 13.3, 14.2, 13.3,
 13.7, 14.0, 12.1, 11.6, 12.0, 13.9, 13.5, 14.0, 12.9, 12.4, 11.9, 15.6, 12.9, 13.7, 13.4, 13.8, 13.1, 13.0, 12.8, 14.1,
 14.2, 12.0, 13.8, 13.2, 13.6, 14.6, 13.7, 15.3, 13.3, 14.0)


# Daten zu Kapitel 11
Dobutamin <- c(18.80, 41.70, 41.24, 49.98, 63.90, 29.86, 25.92, 36.92, 54.90, 37.04, 28.94, 33.80, 18.08)
Fenoterol <- c(34.16, 67.82, 60.68, 39.04, 37.72, 46.78, 50.80, 128.60, 31.18, 35.04, 56.40, 22.62, 26.09)
 Propanolol <- c(13.00, 10.00, 12.86, 11.38, 9.94, 13.24, 7.32, 1.10, 12.56, 14.36)
Placebo <- c(20.94, 25.56, 19.64, 9.26, 16.42, 16.22, 16.90, 26.46, 15.26, 10.90, 18.16, 21.50, 26.58)


# Schweinedaten
Anfangsgewicht <- c(61, 59, 76, 50, 61, 54, 57, 45, 41, 40, 74, 75, 64, 48, 62, 42, 52, 43, 50, 40, 80, 61, 62, 47, 59, 42,
 47, 42, 40, 40, 62, 55, 62, 43, 57, 51, 41, 40, 45, 39)
Zunahme <- c(1.4, 1.79, 1.72, 1.47, 1.26, 1.28, 1.34, 1.55, 1.57, 1.26, 1.61, 1.31, 1.12, 1.35, 1.29, 1.24, 1.29, 1.43,
 1.29, 1.26, 1.67, 1.41, 1.73, 1.23, 1.49, 1.22, 1.39, 1.39, 1.56, 1.36, 1.40, 1.47, 1.37, 1.15, 1.22, 1.48, 1.31, 1.27,
  1.22, 1.36)
Futter <- c(rep("Futter 1", 10), rep("Futter 2", 10), rep("Futter 3", 10), rep("Futter 4", 10))


# Ein soziologisches Beispiel; Kapitel 12.4
# Variable  Description
# bwt  Birth weight in ounces (NA unknown)
# gestation Length of pregnancy in days (NA unknown)
# parity 0= first born, 1=not first born, 9=unknown
# age   Mother's age in years
# height   Mother's height in inches (99 unkmown))
# weight   Mother's prepregnancy weight in pounds (NA unknown)
# smoke  Smoking status of mother 0=not now, 1=yes now, 9=unknown

bwt gestation parity age height weight smoke
120 284 0 27 62 100 0
113 282 0 33 64 135 0
128 279 0 28 64 115 1
123 NA 0 36 69 190 0
108 282 0 23 67 125 1
136 286 0 25 62 93 0
138 244 0 33 62 178 0
132 245 0 23 65 140 0
120 289 0 25 62 125 0
143 299 0 30 66 136 1
140 351 0 27 68 120 0
144 282 0 32 64 124 1
141 279 0 23 63 128 1
110 281 0 36 61 99 1
114 273 0 30 63 154 0
115 285 0 38 63 130 0
 92 255 0 25 65 125 1
115 261 0 33 60 125 1
144 261 0 33 68 170 0
119 288 0 43 66 142 1
105 270 0 22 56 93 0
115 274 0 27 67 175 1
137 287 0 25 66 145 0
122 276 0 30 68 182 0
131 294 0 23 65 122 0
103 261 0 27 65 112 1
146 280 0 26 58 106 0
114 266 0 20 65 175 1
125 292 0 32 65 125 0
```

```
114 274 0 28 66 132 1
122 270 0 26 61 105 0
 93 278 0 34 61 146 0
130 268 0 30 66 123 0
119 275 0 23 60 105 0
113 281 0 24 65 120 0
134 283 0 22 67 130 0
107 279 0 24 63 115 0
134 288 0 23 63 92 1
122 267 0 27 65 101 1
128 282 0 31 65 NA 0
129 293 0 30 61 160 0
110 278 0 23 63 177 0
138 302 0 26 99 NA 1
111 270 0 27 61 119 0
 87 248 0 37 65 130 1
143 274 0 27 63 110 1
155 294 0 32 66 150 0
110 272 0 25 60 90 0
122 275 0 26 66 147 0
145 291 0 26 63 119 1
115 258 0 26 62 130 0
108 283 0 31 65 148 1
102 282 0 28 61 110 0
143 286 0 31 64 126 0
146 267 0 30 67 132 0
124 275 0 22 60 130 0
124 278 0 26 70 145 1
145 257 0 33 65 140 0
106 273 0 28 60 116 0
 75 232 0 33 61 110 0
107 273 0 24 61 96 0
124 288 0 22 67 118 0
122 280 0 23 65 125 1
101 245 0 23 63 130 1
128 283 0 28 63 125 1
104 282 0 36 65 115 1
 97 246 0 37 63 150 0
137 274 0 26 69 137 1
103 273 0 31 63 170 1
142 276 0 38 63 170 0
130 289 0 27 66 130 0
156 292 0 26 63 118 0
133 284 0 25 66 125 1
120 274 0 24 62 120 0
 91 270 0 24 60 149 1
127 274 0 21 62 110 0
153 286 0 26 63 107 1
121 276 0 39 63 130 0
120 277 0 27 63 126 0
 99 272 0 27 62 103 1
149 293 0 35 65 116 0
129 280 0 23 64 104 0
139 292 0 25 68 135 0
114 274 0 33 67 148 1
138 287 0 30 66 145 0
129 274 0 29 71 NA 1
138 294 0 32 65 117 0
131 296 0 37 63 143 0
125 305 0 22 70 196 1
114 NA 0 24 67 113 1
128 281 0 33 59 117 0
134 268 0 28 62 112 0
114 271 0 27 60 104 0
 92 NA 0 31 67 130 0
 85 278 0 23 61 103 1
135 282 0 22 64 100 0
 87 255 0 28 61 100 1
125 302 0 37 62 162 0
128 NA 0 35 62 110 0
105 254 0 29 64 137 0
120 279 0 27 60 121 1
119 274 0 33 64 120 0
116 286 0 24 61 NA 0
107 280 0 36 65 117 1
119 273 0 24 61 108 1
133 279 0 37 66 140 0
155 287 0 33 66 143 0
126 273 0 22 65 150 0
129 303 0 27 64 125 0
137 274 0 29 65 154 0
103 269 0 26 65 NA 1
125 302 0 28 65 125 0
 91 255 0 19 67 136 1
134 293 0 21 65 NA 0
 95 279 0 22 66 145 1
118 276 0 29 64 114 0
141 278 0 33 66 109 1
131 283 0 25 67 215 0
121 264 0 32 66 145 0
```

```
100 243 0 39 65 170 1
131 288 0 24 61 103 0
118 284 0 26 66 133 0
152 288 0 35 67 130 0
121 284 0 34 69 155 0
117 276 0 31 69 150 0
115 283 0 25 61 150 1
112 277 0 23 65 110 0
 94 267 0 30 62 120 1
109 272 0 35 66 154 0
132 225 0 28 67 148 0
117 278 0 25 62 103 0
101 266 0 20 67 110 1
112 294 0 25 64 125 1
128 283 0 24 60 100 0
128 279 0 25 66 147 1
117 258 0 31 64 120 0
134 278 0 24 69 135 0
127 284 0 28 65 145 0
 93 269 0 21 65 104 1
122 275 0 27 65 165 0
100 265 0 39 62 107 1
147 293 0 32 65 123 0
120 299 0 25 65 110 0
144 277 0 30 63 127 0
105 268 0 32 61 115 1
136 276 0 23 66 155 0
102 262 0 24 63 125 0
160 300 0 29 71 175 1
113 275 0 24 68 140 1
126 282 0 38 66 250 0
126 271 0 29 68 148 0
115 278 0 29 61 128 0
127 336 0 29 99 NA 0
119 284 0 20 66 132 0
129 NA 0 23 99 NA 1
123 318 0 21 64 152 0
118 282 0 22 68 135 1
133 287 0 24 60 104 1
105 281 0 39 61 NA 0
134 290 0 22 60 121 0
144 288 0 21 67 111 0
111 273 0 43 62 138 0
125 262 0 36 66 190 0
135 296 0 30 63 123 0
134 289 0 22 63 125 0
116 289 0 22 65 160 1
129 291 0 29 69 123 0
113 301 0 26 67 105 1
131 295 0 23 65 123 1
126 293 0 29 59 110 9
121 272 0 22 62 109 0
121 271 0 25 68 118 1
138 287 0 24 65 115 0
136 278 0 23 61 105 0
120 279 0 30 66 131 0
122 278 0 31 72 155 1
134 267 0 30 66 170 0
101 280 0 25 65 123 1
112 288 0 32 62 125 0
132 290 0 25 64 120 0
136 285 0 23 62 175 0
113 277 0 23 65 192 1
 96 271 0 23 64 116 0
124 277 0 29 63 220 0
113 306 0 21 62 150 0
131 286 0 34 99 NA 1
137 258 0 25 63 117 0
133 268 0 24 61 93 0
107 244 0 20 58 97 0
 96 265 0 28 59 135 1
142 278 0 35 66 136 1
136 275 0 22 63 110 0
 75 239 0 26 63 124 1
125 302 0 32 61 NA 1
104 295 0 26 65 155 1
130 274 0 30 63 150 0
 90 290 0 22 63 168 0
118 276 0 22 66 147 1
123 320 0 22 66 117 0
137 291 0 34 61 110 0
101 268 0 19 63 140 0
142 275 0 25 64 132 0
 98 282 0 20 63 97 1
124 283 0 23 63 112 0
151 310 0 21 65 NA 0
109 281 0 23 61 105 0
150 285 0 22 61 110 1
119 282 0 26 68 150 1
131 280 0 38 65 125 0
```

```
101 272 0 29 63 150 1
113 246 0 19 62 138 1
127 270 0 25 62 150 0
 97 260 0 23 61 99 1
117 282 0 28 64 115 0
150 290 0 21 65 125 0
 85 234 0 33 67 130 0
128 288 0 27 70 145 0
105 233 0 34 61 130 0
 90 269 0 26 67 125 9
115 274 0 22 65 130 1
107 290 0 28 62 135 0
121 275 0 24 63 121 1
119 286 0 20 64 180 0
117 275 0 20 64 145 1
134 264 0 26 68 136 0
117 288 0 35 65 142 0
115 268 0 28 66 128 0
110 254 0 23 63 120 1
130 282 0 21 62 106 1
140 274 0 23 63 106 1
111 284 0 22 99 NA 1
 93 249 0 33 66 117 0
154 292 0 42 65 116 1
125 290 0 19 64 127 0
 93 318 0 31 66 135 0
122 277 0 33 63 135 1
129 267 0 22 63 160 0
126 276 0 23 63 120 0
 85 274 0 24 68 155 0
173 293 0 30 63 110 0
144 329 0 22 65 190 1
114 278 0 25 65 140 1
111 NA 0 27 63 105 1
154 287 0 27 65 125 1
150 274 0 25 67 117 1
111 278 0 21 62 125 0
126 277 0 32 66 128 0
122 261 0 28 65 124 0
141 282 0 24 68 169 0
142 274 0 24 63 125 0
 99 262 0 38 59 110 1
113 286 0 23 63 105 0
149 282 0 21 61 110 0
117 328 0 29 65 125 1
130 274 0 26 64 185 9
106 275 0 31 65 142 9
128 290 0 22 64 118 0
125 286 0 21 64 139 0
114 290 0 30 66 160 0
130 285 0 23 63 128 1
116 148 0 28 66 135 0
 81 256 0 30 64 148 1
124 287 0 27 62 105 1
125 292 0 22 65 122 0
110 262 0 25 66 140 0
125 279 0 23 63 104 1
138 294 0 40 64 125 0
142 284 0 39 66 132 0
115 278 0 23 60 102 1
102 280 0 38 67 140 0
140 294 0 25 61 103 0
133 276 1 22 63 119 0
127 290 0 35 66 165 0
104 274 1 20 62 115 1
119 275 0 42 67 156 1
152 301 0 29 65 150 0
123 284 1 20 65 120 1
143 273 0 19 66 135 0
131 308 0 40 65 160 0
141 319 1 20 67 140 1
129 277 0 30 66 142 1
113 282 1 36 59 140 0
119 292 0 33 62 118 1
109 295 1 23 63 103 1
104 280 1 27 68 146 1
131 282 1 21 66 126 0
110 293 1 28 64 135 1
148 279 0 27 71 189 0
137 283 1 20 65 157 0
117 283 0 27 63 108 0
115 302 1 22 67 135 0
 98 280 0 35 64 122 1
136 303 1 20 68 148 1
121 276 1 23 71 152 1
132 285 1 25 63 140 0
 91 264 0 36 60 100 1
119 294 0 34 59 105 0
 85 273 0 26 60 105 1
106 271 1 26 61 110 1
```

```
132 284 0 29 64 122 0
 80 266 1 25 62 125 0
109 286 0 24 64 125 1
111 306 0 27 61 102 0
143 292 1 21 65 125 0
136 290 0 26 66 135 0
110 285 1 19 64 130 0
 98 257 0 29 66 130 1
108 305 1 24 65 112 0
101 295 0 18 62 145 1
 71 281 0 32 60 117 1
124 292 0 29 68 176 1
 93 256 0 34 66 NA 1
106 276 0 30 66 130 0
101 278 0 25 62 112 1
100 277 0 31 62 100 1
104 269 0 35 63 110 1
117 270 0 24 67 135 1
117 267 0 29 65 120 1
149 279 0 25 67 135 0
135 284 0 25 66 123 0
110 283 1 21 66 129 0
121 276 0 31 67 130 0
142 285 1 24 66 136 0
104 260 0 33 64 145 0
138 296 0 34 66 120 0
112 278 1 21 63 120 0
117 293 0 39 60 120 1
109 282 0 25 62 106 1
131 266 1 28 67 135 0
120 273 0 29 64 130 1
116 270 0 29 63 132 0
140 290 0 23 65 110 0
103 273 1 22 64 110 1
120 279 1 23 67 135 0
139 260 1 32 64 127 0
123 254 0 26 62 130 1
104 280 1 23 64 107 1
131 283 0 31 99 NA 0
111 270 0 22 59 103 0
122 277 0 32 63 157 1
116 271 1 30 67 144 1
129 277 0 27 68 130 1
133 292 0 30 65 112 1
110 277 0 25 61 130 0
105 276 0 22 67 130 0
 93 246 0 37 65 130 0
122 281 0 42 63 103 1
133 293 0 23 64 110 1
130 296 1 22 66 117 1
104 307 0 24 59 122 0
106 278 0 31 65 110 1
120 281 0 33 63 113 0
121 284 0 27 63 NA 1
118 276 1 18 63 128 0
140 290 1 19 67 132 1
114 268 0 22 64 104 0
116 280 0 40 62 159 0
129 284 0 24 64 115 0
120 286 0 22 62 115 1
127 281 0 24 63 112 1
107 278 1 27 99 135 0
 71 234 0 32 64 110 1
 88 274 0 30 66 130 0
107 300 0 19 99 NA 1
122 286 0 23 64 145 0
106 302 1 19 66 147 0
135 285 0 30 66 130 0
107 290 0 26 63 112 0
129 294 0 32 62 170 1
126 274 0 39 62 122 0
116 293 1 26 64 125 0
124 294 0 26 62 122 0
123 281 0 23 68 136 0
145 315 0 39 67 143 1
102 278 0 27 67 135 1
129 293 0 30 65 130 1
 98 276 1 22 61 121 0
110 272 0 28 60 108 0
135 282 0 24 67 128 1
101 278 1 20 62 105 0
 96 266 0 26 65 125 0
104 276 1 18 60 109 1
100 249 0 24 67 100 0
154 292 0 40 66 145 0
127 293 0 31 67 137 0
126 288 0 31 62 150 0
126 282 1 23 66 115 1
127 279 0 26 67 155 1
 98 275 0 25 65 112 1
```

```
127 288 1 21 66 130 0
129 299 0 22 68 145 0
131 292 1 22 64 124 1
132 289 1 19 66 145 0
127 280 0 27 62 118 0
 99 313 1 34 59 100 1
115 290 0 30 64 140 1
145 290 1 24 67 125 0
102 249 1 23 67 134 1
136 299 0 29 64 115 0
121 286 1 99 99 NA 0
121 282 0 22 66 133 0
120 286 0 25 62 105 0
118 261 0 26 60 104 0
127 304 1 26 62 105 0
132 281 1 24 63 117 0
102 258 1 22 65 135 0
143 279 0 39 65 129 1
118 277 0 25 62 120 0
102 286 1 22 64 140 0
163 280 0 35 69 139 0
132 294 0 32 64 116 0
116 276 0 33 61 180 0
138 288 1 19 66 124 0
139 279 0 20 64 143 0
132 298 1 23 61 137 0
 87 282 0 27 63 104 1
131 297 0 30 67 132 0
130 282 0 26 67 147 1
123 290 0 28 66 107 1
115 276 1 18 63 110 0
116 272 0 27 64 130 1
119 286 1 20 67 130 0
125 279 1 19 67 135 0
144 282 0 33 66 155 1
123 269 0 26 67 132 0
120 276 0 23 66 114 0
140 251 0 28 63 210 0
120 271 1 17 64 142 1
116 272 0 99 63 138 1
120 289 1 31 59 102 0
146 280 0 23 61 145 0
112 283 1 21 62 102 1
115 269 0 30 62 115 9
132 278 0 20 64 150 1
146 263 0 39 53 110 1
122 275 0 30 68 140 0
128 292 0 32 66 130 0
119 277 0 24 63 120 1
135 278 0 27 66 148 0
116 315 0 26 99 NA 0
129 235 0 24 66 135 0
116 293 1 28 62 108 0
100 275 0 27 64 111 1
118 280 0 27 99 NA 1
138 257 0 38 67 138 0
123 282 0 22 65 130 0
113 288 1 21 61 120 0
129 280 1 24 65 140 1
122 280 0 24 67 127 1
132 281 1 21 67 140 0
120 269 1 40 63 130 0
114 283 1 20 65 115 0
130 280 0 29 66 135 0
117 286 0 32 66 127 1
142 285 0 33 63 124 0
144 273 0 27 62 118 1
127 262 1 32 64 125 0
115 270 0 25 67 165 1
 85 258 0 41 67 137 0
 99 274 0 28 66 118 1
123 323 1 17 64 140 0
112 281 1 23 61 150 0
 68 223 0 32 66 149 1
102 283 1 19 65 127 1
109 273 0 37 65 138 1
102 267 1 25 60 93 1
 99 275 0 23 61 125 1
 78 256 1 29 65 123 0
128 284 1 19 66 111 1
107 303 1 25 67 133 0
136 295 0 23 64 147 0
101 278 0 27 61 99 1
100 275 1 25 64 125 0
109 272 0 41 66 154 1
117 281 1 21 70 141 1
 88 252 1 21 60 115 1
 95 270 0 35 65 135 1
119 280 1 25 61 NA 1
123 272 0 28 99 NA 0
```

```
127 291 1 24 66 135 1
107 293 0 20 65 155 1
124 291 0 26 66 NA 0
126 262 0 37 66 135 1
 98 278 0 27 63 110 1
 96 241 0 23 64 130 1
104 282 0 24 63 144 0
133 273 1 33 63 135 0
 93 267 0 25 63 135 1
101 280 1 24 65 123 1
118 277 0 21 64 155 0
130 289 0 21 61 130 1
125 288 0 22 63 128 1
140 291 1 19 65 122 0
115 290 1 19 65 118 0
130 293 0 26 63 123 0
114 277 1 31 64 125 0
105 278 0 21 64 120 0
101 289 1 31 60 125 0
132 286 0 26 67 122 1
112 252 0 37 64 162 0
 69 232 0 31 59 103 1
114 264 0 26 63 110 1
123 267 0 29 63 111 1
129 284 1 20 66 130 1
114 283 1 15 64 117 1
115 290 0 31 62 95 0
 98 272 1 35 64 129 0
128 283 0 27 67 126 0
119 279 1 20 99 NA 1
119 271 0 28 64 175 1
154 288 0 25 65 147 0
127 247 1 21 63 140 0
131 263 0 29 64 180 1
129 288 0 28 59 102 0
114 286 1 22 64 116 1
110 280 0 29 62 110 1
103 268 0 31 64 150 1
117 287 0 20 65 115 1
138 282 0 25 64 120 0
126 280 0 24 66 147 1
124 271 0 23 66 145 0
111 284 0 34 62 110 0
132 282 0 28 67 200 1
103 240 0 26 65 140 0
158 285 0 28 62 130 0
146 277 0 32 99 NA 0
101 286 1 21 64 117 1
132 290 0 26 66 125 0
114 293 1 20 66 180 1
 71 277 0 40 69 135 0
116 282 0 19 64 120 0
108 271 0 19 60 109 1
123 298 0 25 64 113 1
129 289 0 37 63 132 0
134 282 0 24 62 110 0
113 298 0 30 60 124 1
123 277 1 20 65 160 0
147 277 0 30 68 160 0
121 270 1 20 62 103 0
125 284 1 19 67 130 0
115 277 1 25 66 128 0
101 289 0 27 59 96 0
 93 271 0 30 65 127 1
109 275 0 33 66 120 0
115 276 1 23 60 106 0
130 293 1 23 65 122 1
123 278 0 21 61 89 0
111 300 0 20 64 108 1
 97 279 1 24 64 138 1
122 292 1 25 65 125 0
124 300 0 28 63 95 0
129 276 0 26 66 145 0
124 290 0 26 59 140 0
107 280 0 20 60 107 1
142 273 1 22 62 125 0
129 287 1 29 66 135 0
174 281 0 37 67 155 0
105 264 0 30 65 105 1
103 291 1 26 63 102 0
124 285 1 27 63 114 0
105 265 0 43 65 124 0
133 275 0 36 65 137 1
161 302 1 22 70 170 1
105 260 0 23 64 197 0
108 281 0 41 66 171 0
153 297 0 27 66 145 0
133 280 1 25 61 130 0
115 269 0 41 63 165 1
127 254 0 27 67 146 1
```

```
128 271 0 41 65 135 1
117 265 0 40 68 134 1
123 274 0 23 66 135 0
119 288 1 22 64 132 1
141 284 1 17 64 105 0
 91 260 0 26 62 110 1
116 291 0 29 65 133 1
116 255 0 24 65 132 0
121 273 0 32 64 112 0
111 274 0 36 67 159 0
102 257 0 25 66 135 0
118 283 0 24 65 150 0
126 294 1 22 65 125 1
 98 286 0 31 62 105 1
131 288 1 28 65 125 0
115 278 0 21 60 113 0
103 281 1 22 59 98 1
147 301 0 26 65 130 0
123 308 1 19 65 135 0
125 283 0 22 65 119 0
117 270 0 30 67 130 1
 99 268 0 29 71 150 0
115 283 0 31 66 127 1
116 265 0 36 63 120 0
118 297 0 35 68 140 1
170 303 1 21 64 129 0
104 270 0 25 61 110 0
108 269 1 20 62 114 0
144 289 1 17 69 130 1
 99 250 1 26 66 115 0
 97 263 1 25 63 107 0
142 284 0 37 68 155 9
 85 270 1 19 63 118 1
130 285 1 24 66 126 1
117 275 0 22 62 115 1
109 302 0 24 64 110 0
147 285 0 24 64 137 0
105 281 1 23 64 115 0
135 278 1 27 68 139 1
115 273 1 23 67 215 1
123 280 0 23 65 140 1
105 274 1 26 61 100 0
154 271 0 36 69 160 1
110 276 0 25 63 107 1
119 285 1 26 62 108 0
103 292 1 28 62 132 0
117 272 0 25 64 116 0
120 289 0 23 69 165 0
145 278 0 24 62 109 0
104 271 0 20 62 98 1
123 268 1 18 62 110 1
124 272 0 27 62 110 0
129 275 0 26 64 115 1
 91 248 0 33 63 202 0
109 295 0 32 61 135 0
108 268 0 22 58 112 1
 79 268 0 36 61 108 0
133 301 0 23 62 108 0
114 309 1 27 62 118 0
128 273 0 34 61 125 0
129 280 1 24 65 126 0
 97 234 1 26 65 112 0
103 276 1 21 62 130 1
176 293 1 19 68 180 0
143 294 0 44 65 145 0
127 292 1 21 68 130 1
107 256 0 28 59 90 1
113 268 0 31 62 100 0
106 279 1 21 62 118 1
152 285 0 24 61 120 1
150 275 0 29 65 145 0
136 278 0 35 64 118 1
151 298 0 37 64 135 9
124 279 0 35 66 129 0
123 284 1 18 64 112 1
119 288 0 37 62 128 0
122 291 0 40 64 155 0
112 250 0 34 67 124 0
 93 270 0 25 64 125 1
109 271 0 27 61 NA 1
136 274 1 20 63 165 0
121 NA 0 31 68 132 0
150 292 0 26 64 124 0
 94 264 1 26 64 135 0
120 280 0 29 99 NA 1
146 306 0 38 63 112 0
129 274 0 19 65 101 1
125 292 0 27 65 117 1
124 273 0 21 63 115 0
141 282 0 27 63 115 0
```

```
 96 266 0 33 67 135 1
138 297 0 30 66 133 1
127 282 0 28 67 134 0
114 251 0 26 64 119 1
103 297 0 31 64 125 0
127 288 1 20 65 115 1
141 292 0 29 62 110 9
113 274 0 23 63 108 1
 99 249 1 31 57 98 1
 97 279 0 33 61 105 1
116 275 1 20 68 145 0
126 297 0 26 66 120 1
158 296 0 28 66 140 9
119 277 0 28 66 130 1
123 283 0 27 62 110 0
129 287 0 24 60 107 0
117 256 0 37 65 132 1
100 275 0 26 60 115 0
131 274 0 28 64 118 1
146 279 0 27 64 124 0
 84 267 0 29 60 95 0
115 302 0 28 64 116 0
115 281 0 25 60 94 0
118 284 0 28 70 145 1
 91 292 1 19 61 125 0
112 255 0 39 60 115 0
115 316 1 29 64 110 0
110 269 0 38 61 102 0
117 277 0 34 66 140 0
109 268 1 29 65 120 1
 99 267 0 22 62 94 0
131 274 0 27 62 160 1
136 291 1 25 61 105 0
130 298 0 20 62 120 0
134 296 0 35 60 117 1
128 271 0 29 65 126 1
150 286 0 38 67 175 0
 86 284 0 39 65 174 1
115 278 0 26 63 112 1
141 281 0 28 99 NA 1
 78 237 1 23 63 144 0
100 295 1 21 68 125 1
116 270 0 25 68 169 0
110 271 1 26 66 135 0
109 283 0 34 64 120 0
113 259 0 38 64 128 0
136 297 1 23 66 135 0
114 NA 0 23 63 116 1
121 273 1 34 61 125 0
117 288 1 28 63 140 0
166 299 0 26 68 140 0
 87 229 0 27 62 138 0
120 294 1 23 66 128 1
 95 286 0 26 66 118 1
132 273 0 28 62 113 0
 90 286 0 32 63 105 1
131 308 0 30 58 150 1
103 279 1 22 65 145 1
144 287 1 33 71 153 1
137 299 0 24 62 115 0
124 270 0 20 64 122 0
136 281 1 27 64 127 0
117 298 1 22 64 160 0
121 269 0 23 62 130 0
116 280 0 34 68 198 0
139 275 0 33 62 118 0
110 280 0 39 67 125 0
 86 242 0 20 64 110 1
133 287 0 20 65 165 0
 81 254 0 23 62 157 0
133 281 0 33 63 120 0
132 284 1 20 66 140 0
132 287 0 29 64 148 0
137 274 0 27 64 126 0
 84 279 0 34 63 190 0
136 279 0 30 69 130 1
 92 270 0 34 62 100 1
114 298 1 28 67 114 0
129 274 0 33 69 136 1
167 288 1 19 63 117 0
 71 NA 0 19 64 120 0
124 282 1 22 65 118 0
105 269 0 27 62 100 1
155 283 1 19 70 137 0
125 279 1 21 66 126 0
125 266 0 21 62 120 1
125 283 1 22 59 96 0
115 315 1 22 62 110 0
174 288 0 25 61 182 0
127 290 0 35 66 122 0
```

```
113 262 0 24 60 105 0
115 273 0 22 66 130 1
139 277 0 35 63 140 0
127 275 0 26 62 125 0
111 289 0 26 99 NA 1
112 272 0 26 60 98 0
143 285 0 30 64 135 1
116 286 1 22 58 105 1
155 279 0 33 61 125 0
121 290 0 31 64 127 0
110 282 1 21 66 125 1
 87 277 0 31 62 120 1
132 330 0 34 64 130 1
105 261 0 32 99 NA 1
129 277 0 24 68 142 0
123 280 0 20 62 105 1
 91 279 1 27 62 118 0
147 286 0 30 68 147 0
144 289 0 20 62 106 0
128 292 0 30 64 127 0
137 318 1 19 64 110 0
104 289 0 24 60 104 1
120 271 0 32 63 130 0
112 277 1 23 64 118 0
138 286 0 26 63 111 0
 96 280 0 27 63 105 1
134 285 0 35 62 134 0
126 285 0 24 64 140 0
112 300 0 29 62 121 0
138 313 1 27 65 111 0
110 275 0 25 63 120 0
 83 253 0 29 63 110 1
112 288 1 20 62 110 0
148 286 0 38 68 160 0
119 300 1 34 63 124 0
 86 246 0 25 64 113 1
110 269 0 38 63 145 1
126 282 0 23 61 120 0
125 272 0 30 60 96 0
136 252 0 27 63 130 0
127 283 1 29 64 119 0
 84 272 0 25 64 150 1
131 278 0 22 66 124 0
123 286 1 21 67 130 1
 96 282 1 30 68 127 1
110 286 0 26 62 100 0
123 282 0 29 68 164 0
152 286 1 19 67 135 0
127 288 0 28 65 155 0
117 269 1 21 64 149 1
125 277 0 29 66 139 1
139 273 0 29 68 130 0
114 280 0 31 66 134 1
 96 280 1 34 62 127 1
124 289 0 29 63 110 0
107 272 0 30 64 140 1
113 277 0 38 64 108 0
 98 292 1 20 65 124 1
119 285 1 28 65 127 0
107 268 0 37 58 112 1
117 255 0 26 61 120 0
117 305 0 24 64 155 0
144 276 0 23 67 129 1
136 268 0 30 63 132 1
121 278 0 28 69 132 0
165 282 0 29 66 145 0
120 279 0 38 64 124 0
125 280 0 30 65 130 1
137 285 0 29 65 110 0
100 288 1 28 61 108 1
134 284 0 28 62 112 0
 88 262 0 20 65 118 1
108 291 0 39 65 135 0
123 271 0 41 64 162 0
141 277 0 38 66 162 0
130 270 1 19 66 130 0
139 299 1 20 67 112 0
130 283 0 32 65 118 0
113 289 1 26 59 91 0
 77 238 1 23 63 103 1
 62 228 0 24 61 107 0
 93 245 0 33 61 100 1
109 275 1 37 63 112 1
145 283 0 27 65 125 1
 92 224 0 19 63 134 1
120 281 0 26 61 115 0
135 284 0 39 67 141 0
113 287 0 36 63 118 0
126 251 1 28 64 123 0
143 270 1 27 70 148 0
```

```
128 282 1 25 64 125 0
 98 262 0 22 67 120 0
110 306 1 32 61 122 0
162 284 0 27 64 126 0
116 292 1 20 65 118 0
128 284 0 23 62 110 0
111 275 1 18 61 108 1
137 280 0 34 60 107 0
134 278 0 28 99 126 1
100 264 0 29 64 120 1
160 271 0 32 67 215 0
112 267 1 22 62 138 0
134 297 0 27 67 170 1
145 308 0 35 64 110 1
116 295 0 32 65 120 0
126 278 0 26 64 150 1
111 285 0 29 65 130 0
126 282 0 33 62 117 0
109 291 0 39 64 107 0
136 291 0 41 66 191 0
119 286 0 22 63 185 1
103 267 1 21 66 150 1
124 284 1 17 62 112 0
155 286 0 31 66 127 0
122 282 1 21 66 110 0
113 285 0 26 66 140 0
122 273 0 26 66 210 0
126 293 1 27 62 111 0
116 277 0 41 64 124 1
102 294 0 21 65 130 1
110 181 0 27 64 133 0
133 285 1 30 64 160 0
125 283 0 29 65 125 0
164 286 1 32 66 143 0
133 297 0 36 61 125 0
135 300 0 25 64 NA 0
124 293 1 19 65 150 0
122 306 1 22 62 100 0
121 271 1 34 63 129 1
100 272 0 30 64 150 1
129 NA 1 19 61 110 0
 90 266 1 26 67 135 0
128 272 1 18 67 109 0
116 280 1 22 59 NA 1
 86 276 1 23 65 125 1
123 282 0 30 63 118 0
 87 275 0 28 63 110 1
128 291 1 27 63 132 0
120 288 0 28 63 125 0
125 301 1 35 68 181 0
118 265 0 27 61 123 0
116 284 1 24 66 117 0
131 262 0 22 67 135 0
151 286 1 22 66 130 0
 88 273 0 20 66 110 1
137 284 0 30 67 110 0
127 289 0 23 67 140 0
 96 278 1 18 60 120 1
129 281 0 31 67 155 0
128 288 1 26 65 114 0
 85 255 0 24 68 159 0
111 281 1 27 64 112 0
124 275 0 28 61 116 0
112 292 1 28 62 110 1
115 281 0 28 61 128 1
 72 271 0 39 61 136 0
122 281 1 24 65 137 1
116 291 0 26 66 153 0
127 272 0 20 64 130 1
 90 266 0 23 61 99 1
 99 273 1 27 59 115 0
144 307 1 26 66 125 0
138 280 1 30 65 175 0
 58 245 0 34 64 156 1
109 265 1 24 63 107 1
110 277 1 19 62 160 0
129 278 0 27 63 128 0
150 284 0 40 67 130 0
128 279 0 27 66 135 0
142 284 1 31 66 137 1
115 268 1 31 64 125 0
108 274 0 28 66 175 9
108 283 0 35 62 108 0
139 281 0 27 63 137 0
115 275 0 25 61 155 1
136 288 0 23 62 217 0
163 289 1 25 64 126 1
131 285 0 26 64 130 0
 77 238 0 38 67 135 1
124 283 1 33 67 156 1
```

```
104 270 1 26 62 115 0
102 267 1 24 61 109 1
 94 268 0 30 62 105 1
158 295 1 37 70 137 0
112 275 1 21 68 143 1
119 286 0 26 64 123 1
 97 279 0 29 68 178 1
 99 252 0 21 64 120 0
115 264 1 23 67 134 1
139 284 0 37 61 121 0
144 304 1 27 58 102 1
 99 270 0 22 63 115 1
105 280 1 22 63 116 0
 89 275 0 34 66 170 0
129 270 0 43 67 160 0
119 270 1 20 64 109 0
114 291 0 35 60 112 0
106 289 0 28 67 120 1
122 292 1 34 65 133 0
136 261 0 24 65 110 0
121 286 1 22 69 130 1
112 282 0 26 65 122 0
112 266 0 26 64 122 0
123 314 0 22 61 121 1
139 286 0 33 65 125 1
125 290 0 36 59 105 0
105 295 1 20 64 112 1
130 276 0 41 68 130 0
146 294 0 22 66 145 1
133 290 0 21 64 145 0
147 296 1 19 67 124 0
109 269 0 23 63 113 0
122 286 0 23 64 120 1
135 260 0 43 65 135 0
107 NA 0 19 60 118 0
117 272 0 32 66 118 0
138 284 0 30 66 133 1
120 283 0 28 64 122 1
119 273 0 35 65 125 1
118 278 1 19 62 126 0
105 330 0 23 64 112 1
113 306 1 21 65 137 0
136 NA 0 36 66 135 0
148 291 1 21 63 115 0
140 281 1 22 69 135 0
134 287 1 33 67 131 0
120 280 0 31 61 111 0
123 296 1 26 64 110 1
102 275 0 43 64 160 0
 55 204 0 35 65 140 0
103 276 1 19 63 149 1
123 283 0 21 65 110 0
105 270 1 27 65 134 1
138 289 0 33 65 155 0
128 281 0 28 63 150 0
139 285 0 30 65 129 1
104 288 1 27 61 122 1
159 296 1 27 64 112 0
118 276 0 29 62 130 1
 99 285 0 25 69 128 1
144 281 0 20 63 120 0
121 270 0 25 62 108 1
117 265 1 24 66 98 0
119 293 1 23 65 127 0
105 281 1 19 61 130 0
125 283 0 37 63 145 1
119 259 0 37 62 130 0
101 273 0 39 60 113 0
105 277 1 25 64 156 0
110 281 0 27 60 110 0
100 270 1 21 65 132 1
 98 284 0 29 68 140 0
127 276 0 37 64 159 0
117 324 0 22 62 164 1
122 278 0 37 68 114 0
122 273 1 23 64 130 1
118 281 1 36 66 140 1
137 303 1 23 66 127 1
120 275 0 32 63 115 1
143 285 0 27 68 185 0
108 270 0 29 67 124 1
131 284 1 19 61 114 1
110 277 0 36 61 116 0
105 276 0 20 62 112 1
133 274 0 30 63 NA 0
125 255 0 23 63 133 0
 78 258 1 24 66 115 1
114 289 0 36 60 115 0
111 278 0 29 65 145 1
103 250 0 40 59 140 0
```

```
114 276 0 26 62 127 0
 75 247 0 36 64 120 1
169 296 0 33 67 185 0
 94 271 0 36 61 130 1
150 287 0 36 62 135 0
144 248 0 30 70 145 0
144 291 0 28 67 130 0
143 313 0 20 68 150 0
145 304 1 25 63 109 1
121 285 0 34 64 110 0
105 256 0 31 66 142 0
134 286 0 25 64 125 0
129 294 1 21 65 132 0
114 276 0 24 63 110 0
 97 265 0 30 61 110 0
160 292 0 28 64 120 0
 65 237 0 31 67 130 0
145 288 0 28 64 116 0
 95 273 0 23 60 90 0
139 293 1 21 69 130 0
123 288 0 27 63 125 0
109 283 0 23 65 112 1
110 268 0 34 64 127 0
122 296 1 24 65 132 0
115 307 0 34 65 128 1
117 323 0 26 62 NA 0
108 279 1 19 64 115 0
120 287 0 23 67 116 1
131 269 0 36 68 145 0
136 283 1 24 63 119 0
125 290 0 32 63 135 0
 96 285 1 20 66 117 1
102 282 1 29 65 125 1
102 288 1 18 65 117 0
112 277 1 22 67 120 0
135 272 0 30 65 130 0
 91 266 0 23 60 120 1
129 276 0 31 63 125 0
155 290 0 26 66 129 1
109 274 0 33 69 144 1
 80 262 1 31 61 100 1
125 273 0 30 64 145 0
 94 284 0 24 63 104 1
148 281 0 27 63 110 1
 73 277 0 29 65 145 0
123 267 1 19 66 132 1
 65 232 0 24 66 125 1
118 279 1 21 64 108 0
102 283 0 39 60 119 0
120 280 0 24 61 118 0
108 270 1 21 65 130 1
122 280 1 45 62 128 0
103 268 0 32 62 97 1
105 312 0 41 61 115 1
126 273 1 25 68 135 0
145 316 0 22 67 142 0
139 293 0 34 66 131 0
124 290 0 26 65 165 0
121 282 0 30 65 122 0
126 299 1 21 60 114 0
119 286 1 33 67 137 0
114 277 1 19 63 107 0
118 272 0 23 64 113 0
127 295 0 36 65 145 0
117 290 1 22 67 110 0
137 277 0 41 65 126 0
133 292 0 29 65 135 0
100 264 0 28 60 111 1
107 273 1 26 65 135 0
115 276 1 20 62 105 1
 91 292 1 26 61 113 1
112 287 0 27 64 110 1
125 289 1 31 61 120 0
157 291 0 33 65 121 0
108 256 1 26 67 130 0
130 279 0 31 62 122 0
135 289 0 25 64 127 0
123 277 0 24 66 122 0
100 281 0 24 61 115 0
124 277 1 23 64 104 0
174 284 0 39 65 163 0
129 278 0 26 67 146 0
119 275 0 27 59 113 1
126 272 1 35 61 120 1
128 267 0 37 61 142 0
116 282 1 19 64 124 0
100 285 0 18 68 127 1
 96 285 0 37 66 135 1
131 279 1 20 68 122 1
110 292 0 35 62 127 0
```

```
108 278 0 28 63 125 1
129 275 0 24 65 135 0
141 285 0 23 67 150 0
110 276 0 31 70 155 0
118 273 0 21 63 120 0
111 267 1 24 60 115 0
160 297 0 20 68 136 0
120 280 0 30 60 115 0
121 281 0 29 63 108 0
113 282 0 30 64 118 1
117 270 0 23 58 115 0
158 267 0 35 64 125 0
128 277 0 39 61 120 0
158 289 0 30 66 140 0
133 289 0 22 65 123 1
163 298 0 37 61 98 0
128 282 1 19 66 118 0
126 271 1 21 60 105 0
127 283 0 42 62 154 1
134 287 0 40 63 118 0
140 274 0 41 63 122 0
102 285 0 29 63 117 1
100 252 0 24 61 150 0
120 295 0 29 59 100 1
 98 279 1 18 65 115 1
130 246 0 19 62 118 0
104 280 0 41 63 118 1
122 285 0 31 62 102 1
137 276 1 25 64 127 0
114 285 1 20 61 104 0
 63 236 1 24 58 99 0
 98 318 0 23 63 107 0
 99 268 0 32 63 124 1
 89 238 1 26 64 136 0
117 283 0 22 65 142 1
143 281 0 29 67 132 0
106 279 0 29 63 125 1
 99 246 0 35 62 106 0
156 300 0 27 65 120 1
 72 266 1 25 66 200 1
 75 266 0 37 61 113 1
 97 285 0 35 61 112 1
106 264 0 41 64 114 0
 91 225 0 18 68 117 1
117 269 1 28 61 99 0
117 284 0 25 66 177 1
112 291 0 23 66 145 0
112 270 0 29 61 124 0
141 293 0 28 61 125 0
131 259 0 19 63 134 0
130 290 0 19 65 123 1
132 270 0 26 67 140 0
114 265 0 23 67 130 1
160 291 0 34 64 110 1
106 283 0 24 63 119 0
 84 260 1 20 64 104 1
112 268 1 25 59 103 0
139 311 0 37 66 135 0
104 267 0 30 63 180 0
130 294 0 32 63 110 1
 71 254 0 19 61 145 1
 82 270 0 21 65 150 1
119 280 1 21 64 128 0
123 353 0 26 63 115 0
115 278 0 27 59 95 0
124 289 1 21 67 145 1
138 292 0 25 65 130 1
 88 276 0 25 63 103 1
146 305 0 23 99 NA 0
128 241 1 17 64 126 0
 82 274 0 31 64 101 1
100 274 0 24 63 113 0
114 271 0 32 61 130 0
 97 269 0 20 65 137 1
126 298 0 24 61 112 0
122 275 1 20 65 127 0
152 295 0 39 62 140 0
116 274 0 21 62 110 1
132 302 0 36 63 145 1
 84 260 1 37 66 140 0
119 277 1 18 61 89 1
104 275 0 24 99 NA 0
106 312 0 24 62 135 1
124 NA 1 39 65 228 0
139 291 0 24 65 160 0
103 273 0 36 65 158 1
112 299 0 24 67 145 1
 96 276 0 33 64 127 1
102 281 1 19 67 135 1
120 300 0 34 63 150 1
```

```
102 338 0 19 64 170 0
 97 255 1 22 63 107 1
113 285 0 22 70 145 0
130 297 0 32 58 130 0
 97 260 1 25 63 115 1
116 273 0 31 61 120 0
114 266 0 29 64 113 0
127 242 0 17 61 135 1
 87 247 1 18 66 125 1
141 281 0 29 54 156 1
144 283 1 25 66 140 0
116 273 0 33 66 130 1
 75 265 0 21 65 103 1
138 286 1 28 68 120 0
 99 271 0 39 69 151 0
118 293 0 21 63 103 0
152 267 0 28 99 119 1
 97 266 0 24 62 109 0
146 319 0 28 66 145 0
 81 285 0 19 63 150 1
110 321 0 28 66 180 0
135 284 1 19 60 95 0
114 290 1 21 65 120 1
124 288 1 21 64 116 1
115 262 1 23 64 136 1
143 281 0 28 65 135 1
113 287 1 29 70 145 1
109 244 1 21 63 102 1
103 278 0 30 60 87 1
118 276 0 34 64 116 0
127 290 0 27 65 121 0
132 270 0 27 65 126 0
113 275 1 27 60 100 0
128 265 0 24 67 120 0
130 291 0 30 65 150 1
125 281 1 21 65 110 0
117 297 0 38 65 129 0


# Rattendaten
weight <- c(57, 60, 52, 49, 56, 46, 51, 63, 49, 57, 59, 54, 56, 59, 57, 52, 52, 61, 59, 53, 59, 51, 51, 56, 58, 46, 53,
 86, 93, 77, 67, 81, 70, 71, 91, 67, 82, 85, 71, 75, 85, 72, 73, 70, 86, 80, 79, 88, 75, 75, 78, 69, 61, 72, 114, 123,
 111, 100, 104, 102, 94, 112, 90, 110, 121, 90, 108, 116, 97, 97, 105, 109, 101, 100, 100, 101, 92, 95, 93, 78, 89, 139,
 146, 144, 129, 121, 131, 110, 130, 112, 139, 156, 110, 151, 148, 120, 116, 138, 120, 111, 106, 111, 123, 100, 103, 114,
 90, 104, 172, 177, 185, 164, 151, 153, 141, 154, 140, 169, 191, 138, 189, 177, 144, 140, 171, 129, 126, 133, 122, 140,
 119, 108, 138, 107, 122)


# Margarinedaten
margarine.quan <- c(4.500, 5.167, 5.069, 3.800, 3.444, 3.500, 5.250, 5.857, 5.083, 5.273, 4.500, 4.000, 4.225, 3.824,
 5.400, 5.056, 3.500, 3.417, 4.429, 4.083, 3.600, 4.000, 4.375, 3.833, 4.765, 3.800, 3.778, 3.875, 4.583, 4.929, 4.667,
 3.909, 4.200, 3.875, 3.833, 3.438, 2.400, 3.765, 4.000, 3.917, 3.857, 4.000, 4.091, 3.900, 3.250, 2.167, 4.235, 5.000,
 3.944, 4.625, 4.333, 4.071, 4.000, 4.091, 3.700, 3.750, 3.750,
 4.471, 5.000, 5.389, 5.250, 4.417, 5.071, 4.250, 4.091, 3.900, 4.000, 3.273, 3.765, 5.000, 5.056, 5.500, 4.667, 2.929,
 3.818, 4.545, 3.600, 2.000, 1.857, 1.923, 4.000, 5.615, 6.000, 3.250, 2.091, 1.545, 1.600, 1.500, 4.625, 3.750, 3.529,
 4.000, 4.222, 4.750, 4.500, 4.571, 3.750, 3.909, 3.500, 4.125, 3.417, 3.529, 4.600, 5.278, 5.375, 3.583, 3.786, 4.167,
 3.818, 3.700)
margarine.qual <- c(1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1,
 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0)
margarine.names <- c("Sanella", "Homa", "SB", "Delicado", "Hollbutt", "Weihbutt", "DuDarfst", "Becel", "Botteram",
 "Flora", "Rama")

# Schwermetalldaten
Orte P Fe Mo OC Ni CO Cr Cu PB Zn Mn Ca Cd
61 3843 15606 2,10 27,98 74 13,0 55 64 22,0 94 250 81278 1,40
62 5089 20977 2,30 4,41 49 12,0 59 27 12,0 86 386 147172 0,50
63 1319 20738 2,20 0,18 41 12,0 45 20 8,1 39 362 163445 0,40
68 996 92871 2,20 2,77 97 98,0 147 59 9,4 90 1910 22461 0,30
3 998 19400 0,39 2,02 41 13,0 47 15 6,0 73 345 216000 0,33
5 1793 2430 0,46 3,38 47 13,0 55 14 7,3 105 431 170000 0,40
51 1793 2430 0,46 3,38 47 13,0 55 14 7,3 105 431 170000 0,40
8 1637 24800 0,42 2,83 46 15,0 56 16 7,9 96 407 221000 0,40
9 900 18700 0,05 0,92 42 10,0 52 15 5,1 73 301 229000 0,34
10 912 65100 2,90 2,40 109 35,0 132 41 16,0 111 777 45700 0,60
12 1415 16300 0,29 2,48 36 9,7 46 17 5,7 44 292 260000 0,62
18 1621 24700 0,59 2,70 46 14,0 52 18 6,8 61 395 193000 0,63
100 3525 16003 0,65 20,00 52 24,0 49 60 17,0 68 415 12188 0,98
```

Formelübersicht

Hier kann man alle Formeln finden, die für 'R für Statistik' und für 'R für Biometrie' „neu definiert“ oder verändert wurden (Hier sei dazugesagt, dass einige der Funktionen in ähnlicher Form irgendwo im Netz auftauchen können. Man sollte sich diese dann genau angucken bevor man sie benutzt um etwaige Fehler zu vermeiden).

In der ersten Version des Statistik-Skriptes standen hier nur die Funktionen und man konnte sich direkt ein R-Skript erstellen, in dem diesen Funktionen enthalten waren. In dieser Version sind allerdings zusätzlich die Links zu finden, mit denen man zu dem Text gelangt, der die Formel und deren Argumente beschreibt. Die # stellen jedoch sicher, dass der Text nach wie vor einfach kopiert werden kann und durch die Links keine Konflikte im R-Skript entstehen.

```
#↑Geometrisches Mittel

gm <- function(x){
  return( prod(x)^(1/length(x)) )
}

#↑Harmonisches Mittel

hm <- function(x){
  return( length(x)/sum(1/x) )
}

#↑Variationsbreite

rv <- function(x){
  return( quantile(x, 1)-quantile(x, 0) )
}

#↑Variationskoeffizient

cv <- function(x){
  return( (sd(x)/mean(x))*100 )
}

#↑Vertrauensintervall für einen Mittelwert (3.5)

ci <- function(x, alpha=.05){
  m <- mean(x)
  t_tab <- qnorm(1-alpha/2)
  n <- length(x)
  x_min <- m-t_tab*sd(x)/sqrt(n)
  x_max <- m+t_tab*sd(x)/sqrt(n)
  return( c(x_min, x_max) )
}

#↑Vertrauensintervall für einen Mittelwert bei kleinen Stichproben (3.6)

ci.small <- function(x, alpha=.05){
  m <- mean(x)
  n <- length(x)
  t_tab <- qt(1-alpha/2, n-1)
  x_min <- m-t_tab*sd(x)/sqrt(n)
  x_max <- m+t_tab*sd(x)/sqrt(n)
  return( c(x_min, x_max) )
}

#↑Stichprobenumfang zur Schätzung eines Mittelwertes (3.7)
```

```
n.mean <- function(var, HB) {
  return( (4*var)/(HB)**2 )
}
```

#↑Vertrauensintervall für die Differenz von 2 Mittelwerten (verbundene Stichprobe) (3.8)

```
ci.diff2.dep <- function(x, y, alpha=.05){
  d <- mean(x)-mean(y)
  D <- x - y
  n <- length(D)
  t_tab <- qt(1-alpha/2, n-1)
  x_min <- d-t_tab*sd(D)/sqrt(n)
  x_max <- d+t_tab*sd(D)/sqrt(n)
  return( c(x_min, x_max) )
}
```

#↑Vertrauensintervall für die Differenz von 2 Mittelwerten (unverbundene Stichprobe) (3.9)

```
var.pooled <- function(x, y){
  n1 <- length(x)
  n2 <- length(y)
  v1 <- var(x)
  v2 <- var(y)
  x <- ((n1-1)*v1+(n2-1)*v2)/(n1+n2-2)
  return( x )
}
```

#↑Vertrauensintervall für die Differenz von 2 Mittelwerten (unverbundene Stichprobe) (3.9)

```
ci.diff2.indep <- function(x, y, alpha=.05) {
  n1 <- length(x)
  n2 <- length(y)
  v1 <- var(x)
  v2 <- var(y)
  m1 <- mean(x)
  m2 <- mean(y)
  t_tab <- qt(1-alpha/2, n1+n2-2)
  s <- sqrt(var.pooled(x, y))
  x_min <- (m1-m2)-t_tab*s*sqrt(1/n1+1/n2)
  x_max <- (m1-m2)+t_tab*s*sqrt(1/n1+1/n2)
  return( c(x_min, x_max) )
}
```

#↑Stichprobenumfang für den unverbundenen t-Test (3.14)

```
t.test.planning <- function(n=NULL, power=NULL, delta, var, alpha=.05){
  if(!is.null(n) & !is.null(power)){
    stop("both, 'n' and 'power' are given, but only one of the two can be given")
  }
  if(is.null(n)){
    if(is.null(power)){
      stop("either 'n' or 'power' must be given")
    }
    else{
    q1 <- qnorm(1-alpha/2)
    q2 <- qnorm(power)
    n <- 2*(var/delta**2)*(q1+q2)**2
    cat( "\nn \t=", n )
    cat( "\npower \t=", power )
    cat( "\ndelta \t=", delta )
    cat( "\nvar \t=", var )
    cat( "\nalpha \t=", alpha )
    cat( "\n\n" )
    }
  }
  if(is.null(power)){
    if(is.null(n)){
      stop("either 'n' or 'power' must be given")
    }
    else{
    q <- qnorm(1-alpha/2)
    power <- sqrt((n*delta**2)/(2*var))- q
    p <- pnorm(power)
    cat( "\npower \t=", p )
    cat( "\nn \t=", n )
    cat( "\ndelta \t=", delta )
    cat( "\nvar \t=", var )
    cat( "\nalpha \t=", alpha )
    cat( "\n\n" )
    }
  }
}
```

#↑Vertrauensintervall für eine Varianz (3.18)

```
ci.var <- function(x, alpha=.05){
  var <- var(x)
  n <- length(x)
  q1 <- qchisq(1-alpha/2, (n-1))
  q2 <- qchisq(alpha/2, (n-1))
  x_min <- ((n-1)*var)/q1
  x_max <- ((n-1)*var)/q2
  return( c(x_min, x_max) )
}
```

#↑Äquivalenztest am Beispiel zweier unverbundener Stichproben (3.21)

```
equi.test <- function(x, y, delta, alpha) {
  x1 <- mean(x)
  x2 <- mean(y)
  n1 <- length(x)
  n2 <- length(y)
  s <- var.pooled(x, y)
  t_exp1 <- (x1-x2+delta)/(s*sqrt(1/n1+1/n2))
  t_exp2 <- (x1-x2-delta)/(s*sqrt(1/n1+1/n2))
  t_tab <- qt(1-alpha, n1+n2-2)
  if(t_exp1 > t_tab & t_exp2 < -1*t_tab){
    cat( "\nt_exp1 \t=", t_exp1 )
    cat( "\nt_exp2 \t=", t_exp2 )
    cat( "\nt_tab \t=", t_tab )
    cat( "\n\n" )
    print( list("test decision:"="t_exp1 > t_tab & t_exp2 <
-1*t_tab -> equivalence") )
  }  else {
    if(t_exp1 < t_tab & t_exp2 < -1*t_tab){
      cat( "\nt_exp1 \t=", t_exp1 )
      cat( "\nt_exp2 \t=", t_exp2 )
      cat( "\nt_tab \t=", t_tab )
      cat( "\n\n" )
      print( list("test decision:"="t_exp1 < t_tab & t_exp2 <
-1*t_tab -> no equivalence") )
    }
    if(t_exp1 > t_tab & t_exp2 > -1*t_tab){
      cat( "\nt_exp1 \t=", t_exp1 )
      cat( "\nt_exp2 \t=", t_exp2 )
      cat( "\nt_tab \t=", t_tab )
      cat( "\n\n" )
      print( list("test decision:"="t_exp1 > t_tab & t_exp2 >
-1*t_tab -> no equivalence") )
    }
    if(t_exp1 < t_tab & t_exp2 > -1*t_tab){
      cat( "\nt_exp1 \t=", t_exp1 )
      cat( "\nt_exp2 \t=", t_exp2 )
      cat( "\nt_tab \t=", t_tab )
      cat( "\n\n" )
      print( list("test decision:"="t_exp1 < t_tab & t_exp2 >
-1*t_tab -> no equivalence") )
    }
  }
}
```

#↑Permutationen:

```
perm.rep <- function(n, k){
  return( n**k )
}


perm.norep <- function(n, k){
  return( prod(n:1)/prod((n-k):1) )
}


comb.rep <- function(n, k){
  return( prod((n+k-1):1)/(prod(k:1)*prod((n-1):1)) )
}


comb.norep <- function(n, k){
  return( prod(n:1)/(prod(k:1)*prod((n-k):1)) )
}
```

#↑Mittelwert und Varianz der Binomialverteilung

```
param.binom <- function(p=NULL, success=NULL, total=NULL){
  if (is.null(total)){
    stop("'total' must be given to calculate 'p' or 'succes'")
  }
  if (is.null(p)){
    n <- total
    p <- success/total
    q <- 1-p
    mean.binom <- n*p
    var.binom <- n*p*q
```

```
    var.p <- (p*(1-p))/n
    sd.p <- sqrt(var.p)
    cat( "\np \t\t=", p)
    cat( "\nsuccess \t\t=", success ,"out of", total)
    cat( "\nmean.binom \t=", mean.binom)
    cat( "\nvar.binom \t=", var.binom)
    cat( "\nvar.p \t\t=", var.p)
    cat( "\nsd.p \t\t=", sd.p)
    cat( "\n\n")
  }
  if (is.null(succes)){
    n <- total
    succes <- p*n
    q <- 1-p
    mean.binom <- n*p
    var.binom <- n*p*q
    var.p <- (p*(1-p))/n
    sd.p <- sqrt(var.p)
    cat( "\np \t\t=", p)
    cat( "\nsuccess \t\t=", success ,"out of", total)
    cat( "\nmean.binom \t=", mean.binom)
    cat( "\nvar.binom \t=", var.binom)
    cat( "\nvar.p \t\t=", var.p)
    cat( "\nsd.p \t\t=", sd.p)
    cat( "\n\n")
  }
}
```

#↑Erwartete Häufigkeit einer Poisson-Verteilung

```
ek <- function( n, k, l, prec=1, plot=TRUE, script=TRUE, dd=3){
  vec <- n * exp( (-1)*l )
  for( i in 1:k )
    vec <- c( vec, vec[i] * l/i )                 # E_k für 0...k=n ausrechnen
  if( plot )
    plot( 0:k, vec, pch=19, main="Erwartungswerte der Poisson-Verteilung", xlab="k", ylab="Erwartungswert" )
  ks <- 0:k                                      # Bezeichnungen für den späteren Vektor
  lt.ind <- which( vec < prec ) # welche E_ks sind < prec
  left <- lt.ind[which(lt.ind < which(vec == max(vec)))] # Indices der E_ks, die zu einer Klasse zusammengefasst werden
 müssen (vorn)
  right <- lt.ind[which(lt.ind > which(vec == max(vec)))]  # für Klasse hinten
  if( length(left) > 0 ){                           # links sind E_k zusammen zu fassen
    if( length(right) > 0 ){                        # rechts - " -
      if( script ){
        ks <- c( paste(0,"..",(ks[max(left)]+1), sep="") , ks[(max(left)+2):(min(right)-2)], paste(vec[(min(right)-1)],
"...", sep="") )
        res <- c(sum(vec[c(left,(max(left)+1))]), vec[(max(left)+2):(min(right)-2)], n - sum(vec[c(left,(max(left)+1))] -
 sum(vec[(max(left)+2):(min(right)-2)])) )
      }
      else{
        ks <- c( paste(0,"..",ks[max(left)], sep="") , ks[(max(left)+1):(min(right)-1)], paste(vec[min(right)],
 "...", sep="" ))
        res <- c(sum(vec[left]), vec[(max(left)+1):(min(right)-1)], n - sum(vec[left] -
sum(vec[(max(left)+1):(min(right)-1)])) )
      }
    }
    else{                                           # rechts keine (nur links)
      if( script ){
        ks <- c( paste(0, "..", ks[(max(left)+1)], sep=""), ks[(max(left)+2):length(ks)])
        res <- c( sum(vec[c(left, (max(left)+1))]), vec[(max(left)+2):length(vec)])
      }
      else{
        ks <- c( paste(0, "..", ks[max(left)], sep=""), ks[(max(left)+1):length(ks)])
        res <- c( sum(vec[left]), vec[(max(left)+1):length(vec)])
      }
    }
  }
  else{                                             # links keine
    if( length(right) > 0 ){                        # aber rechts
      if( script ){
        ks <- c(ks[1:(min(right)-2)], paste( ks[(min(right)-1)], "...", sep="") )
        res <- c( vec[1:(min(right)-2)], n - sum(vec[1:(min(right)-2)]) )
      }
      else{
        ks <- c(ks[1:(min(right)-1)], paste( ks[min(right)], "...", sep="") )
        res <- c( vec[1:(min(right)-1)], n - sum(vec[1:(min(right)-1)]) )
      }
    }
    else{                                           # auch rechts keine
      ks <- ks                                      # es änder sich nichts
      res <- vec
    }
  }

  names(res) <- ks
  return( round(res, dd) )
}
```

#↑Seite 144, Beispiel 1:

```
ci.lambda <- function(n, lambda, alpha=.05){
  q <- qnorm(1-alpha/2)
  x_min <- (1/n)*(sqrt(n*lambda)-0.5*q)**2
  x_max <- (1/n)*(sqrt(n*lambda+1)+0.5*q)**2
  return( c(x_min, x_max) )
}
```

#↑Test für den Vergleich zweier Parameter $\lambda 1$ und $\lambda 2$ (5.4.3)

```
lambda.test <- function(l1, l2, n1, n2, alpha=.05){
  z_exp <- abs( l1 - l2 )/sqrt( ((n1*l1+n2*l2)/(n1+n2))*(1/n1 + 1/n2)  )
  z_tab <- qnorm(1-alpha/2)
  if(z_exp > z_tab){
    cat( "\nz_exp \t=", z_exp)
    cat( "\nz_tab \t=", z_tab)
    cat( "\n\n")
    print( list("test decision:"="z_exp > z_tab -> significance") )
  }
  else {
    cat( "\nz_exp \t=", z_exp)
    cat( "\nz_tab \t=", z_tab)
    cat( "\n\n")
    print( list("test decision:"="z_exp =< z_tab -> no significance") )
  }
}
```

#↑Chi-Quadrat-Anpassungstest

```
chisq.test <- function (x, y = NULL, q, correct = TRUE, p = rep(1/length(x), length(x)),
    rescale.p = FALSE, simulate.p.value = FALSE, B = 2000)
{
    DNAME <- deparse(substitute(x))
    if (is.data.frame(x))
        x <- as.matrix(x)
    if (is.matrix(x)) {
        if (min(dim(x)) == 1)
            x <- as.vector(x)
    }
    if (!is.matrix(x) && !is.null(y)) {
        if (length(x) != length(y))
            stop("'x' and 'y' must have the same length")
        DNAME <- c(DNAME, deparse(substitute(y)))
        OK <- complete.cases(x, y)
        x <- factor(x[OK])
        y <- factor(y[OK])
        if ((nlevels(x) < 2) || (nlevels(y) < 2))
            stop("'x' and 'y' must have at least 2 levels")
        x <- table(x, y)
        names(dimnames(x)) <- DNAME
        DNAME <- paste(DNAME, collapse = " and ")
    }
    if (any(x < 0) || any(is.na(x)))
        stop("all entries of 'x' must be nonnegative and finite")
    if ((n <- sum(x)) == 0)
        stop("at least one entry of 'x' must be positive")
    if (simulate.p.value) {
        setMETH <- function() METHOD <<- paste(METHOD, "with simulated p-value\n\t (based on",
            B, "replicates)")
        almost.1 <- 1 - 64 * .Machine$double.eps
    }
    if (is.matrix(x)) {
        METHOD <- "Pearson's Chi-squared test"
        nr <- nrow(x)
        nc <- ncol(x)
        sr <- rowSums(x)
        sc <- colSums(x)
        E <- outer(sr, sc, "*")/n
        dimnames(E) <- dimnames(x)
        if (simulate.p.value && all(sr > 0) && all(sc > 0)) {
            setMETH()
            tmp <- .C(R_chisqsim, as.integer(nr), as.integer(nc),
                as.integer(sr), as.integer(sc), as.integer(n),
                as.integer(B), as.double(E), integer(nr * nc),
                double(n + 1), integer(nc), results = double(B))
            STATISTIC <- sum(sort((x - E)^2/E, decreasing = TRUE))
            PARAMETER <- NA
            PVAL <- (1 + sum(tmp$results >= almost.1 * STATISTIC))/(B +
                1)
        }
        else {
            if (simulate.p.value)
                warning("cannot compute simulated p-value with zero marginals")
            if (correct && nrow(x) == 2 && ncol(x) == 2) {
                YATES <- 0.5
                METHOD <- paste(METHOD, "with Yates' continuity correction")
            }
            else YATES <- 0
            STATISTIC <- sum((abs(x - E) - YATES)^2/E)
```

```
            PARAMETER <- (nr - 1) * (nc - 1)
            PVAL <- pchisq(STATISTIC, PARAMETER, lower.tail = FALSE)
        }
    }
    else {
        if (length(x) == 1)
            stop("'x' must at least have 2 elements")
        if (length(x) != length(p))
            stop("'x' and 'p' must have the same number of elements")
        if (any(p < 0))
            stop("probabilities must be non-negative.")
        if (abs(sum(p) - 1) > sqrt(.Machine$double.eps)) {
            if (rescale.p)
                p <- p/sum(p)
            else stop("probabilities must sum to 1.")
        }
        METHOD <- "Chi-squared test for given probabilities"
        E <- n * p
        names(E) <- names(x)
        STATISTIC <- sum((x - E)^2/E)
        if (simulate.p.value) {
            setMETH()
            nx <- length(x)
            sm <- matrix(sample(1:nx, B * n, TRUE, prob = p),
                nrow = n)
            ss <- apply(sm, 2, function(x, E, k) {
                sum((table(factor(x, levels = 1:k)) - E)^2/E)
            }, E = E, k = nx)
            PARAMETER <- NA
            PVAL <- (1 + sum(ss >= almost.1 * STATISTIC))/(B +
                1)
        }
        else {
            PARAMETER <- length(x) - 1 - q  # statt PARAMETER <- length(x) - 1
            PVAL <- pchisq(STATISTIC, PARAMETER, lower.tail = FALSE)
        }
    }
    names(STATISTIC) <- "X-squared"
    names(PARAMETER) <- "df"
    if (any(E < 5) && is.finite(PARAMETER))
        warning("Chi-squared approximation may be incorrect")
    structure(list(statistic = STATISTIC, parameter = PARAMETER,
        p.value = PVAL, method = METHOD, data.name = DNAME, observed = x,
        expected = E, residuals = (x - E)/sqrt(E)), class = "htest")
}
```

#↑Schätzen des Parameters p der Binomialverteilung bei Messwiederholungen

```
weighted.mean.binom <- function(x, w){
  if(missing(w)){
    w <- rep.int(1, length(x))
  }
  else{
    if(length(w) != length(x)){
      stop("'x' and 'w' must have the same length")
    }
  }
  if(is.integer(w)){
    w <- as.numeric(w)
  }
  sum(x * w)/((length(x)-1)*sum(w))
}

rec.binom <- function(h, p, n, k ){
  if( k==0 )
    return( h*((1-p)^n) )
  else
    return( rec.binom(h, p, n, k-1) * p/(1-p) * (n-k+1)/k )
}
```

#↑Erwartete Häufigkeit einer Binomial-Verteilung

```
rec.binom <- function(h, p, n, k ){
if( k==0 )
return( h*((1-p)^n) )
else
return( rec.binom(h, p, n, k-1) * p/(1-p) * (n-k+1)/k )
}
```

#↑t-Tests und Vertrauensintervalle

```
ci.reg <- function(x, y, x_0, rl=FALSE, alpha=0.05){
  n <- length(x)
  fit <- lm(x ~ y)
  a <- fit$coef[1]
  b <- fit$coef[2]
  p <- a+b*x_0
  sqx <- sum((x-mean(x))**2)
```

```
  sqy <- sum((y-mean(y))**2)
  sqerror <- sqx-((b)**2)*sqy
  s <- sqrt(sqerror/(n-2))
  my <- mean(y)
  t_tab <- qt(1-alpha/2, n-2)
  if( rl ){
    q <- sqrt((1/n)+(((x_0-my)**2)/sqy))
  }
  else{
    q <- sqrt(1+(1/n)+(((x_0-my)**2)/sqy))
  }
  x_min <- p-t_tab*s*q
  x_max <- p+t_tab*s*q
  return( c(x_min, x_max) )
}
```

#↑Inverse Regression

```
ci.inv.reg <- function(x, y, y_0, sx=FALSE, alpha=0.05){
  n <- length(x)
  fit <- lm(y ~ x)
  a <- fit$coef[1]
  b <- fit$coef[2]
  t_tab <- qt(1-alpha/2, n-2)
  sqx <- sum((x-mean(x))**2)
  sqy <- sum((y-mean(y))**2)
  sqerror <- sqy-((b)**2)*sqx
  s <- sqrt(sqerror/(n-2))
  my <- mean(y)
  mx <- mean(x)
  coef1 <- b**2-((t_tab**2*s**2)/sqx)
  coef2 <- -2*b*(y_0-my)
  coef3 <- (y_0-my)**2-t_tab**2*s**2*(1+1/n)
  x1 <- (-coef2-sqrt(coef2**2-4*coef1*coef3))/(2*coef1)
  x2 <- (-coef2+sqrt(coef2**2-4*coef1*coef3))/(2*coef1)
  x_min <- mx+x1
  x_max <- mx+x2
  x <- (y_0-a)/b
  if( sx ){
    return( c(x_min, x, x_max) )
  }
  else{
    return( c(x_min, x_max) )
  }
}
```

#↑Zweistufige Stichproben

```
ci.two.stage <- function(x, y, z, alpha=.05){
df <- data.frame(x, y)
lm <- lm(x ~ y, df)
aov <- anova(lm)
m <- mean(x)
n1 <- length(levels(y))
n2 <- length(levels(z))
mq <- aov$"Mean Sq"[1]
t_tab <- qt(1-alpha/2, n1-1)
v <- mq/(n1*n2)
x_min <- m-t_tab*sqrt(v)
x_max <- m+t_tab*sqrt(v)
return( c(x_min, x_max) )
}
```

Register

Hier sind alle Funktionen, Methoden und sonstigen Stichwörter aufgelistet, die zum schnellen Einstieg an die Stelle dienen sollen, an der sie erwähnt werden. In der PDF-Version dieses Skriptes ist jedes einzelne Stichwort verlinkt, für eine mögliche Druckversion befinden sich die Seitenzahlen hinter den Stichwörtern. Fett gedruckte Wörter stellen Funktionen von R dar. Kursiv gedruckte Wörter sind Argumente, die teilweise in mehreren Funktionen zur Anwendung kommen. Fett und Kursiv gedruckte Worte sind Pakete, die in diesem Skript Erwähnung finden. Alle Begriffe, die im erklärenden Text verwendet werden und alle Begriffe, die dem Output von R entstammen, sind in normalen Lettern gedruckt.

Viele der hier auftauchenden Worte finden sich nicht nur an der hier erwähnten Stelle wieder, sondern auch an einigen andere Stellen. Hier finden sich also lediglich die wichtigsten Vermerke zu den erwähnten Begriffen. Oftmals ist ein Wort auf mehrere Textstellen bezogen, dann kann man von einer Anwendung in einem anderen Kontext ausgehen, die eine Betrachtung wert ist.