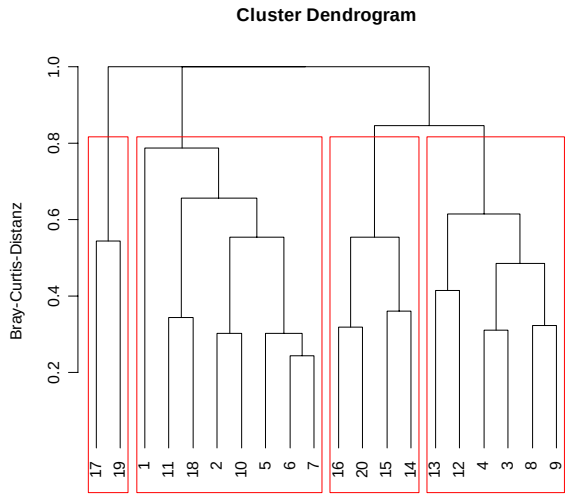
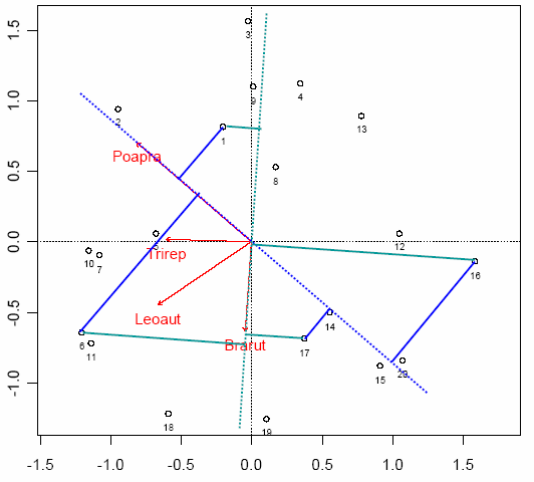
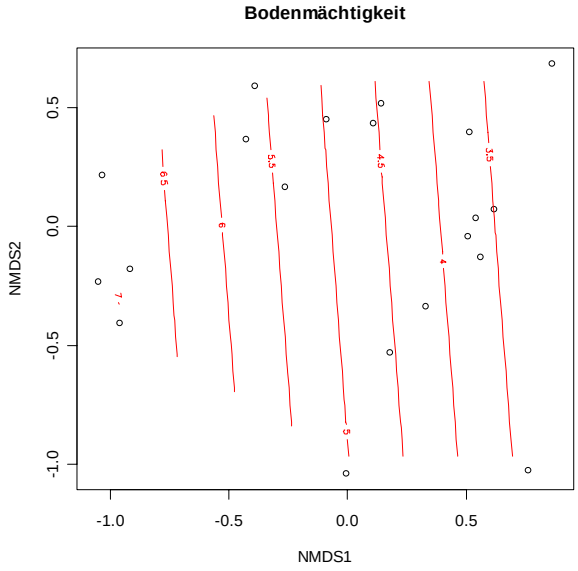
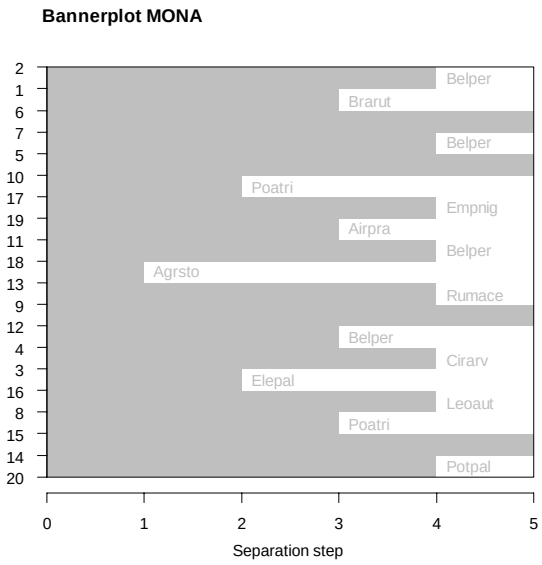
Multivariate Methoden in der Vegetationskunde
Benutzung des Programms R
und Umsetzung der Methoden in R

Arne Saatkamp, Freiburg, Februar 2005
Verändert und erweitert durch
Hiltrud Brose und Michael Rudner, Freiburg, Oktober 2007
Verbesserte Version, April 2008









Fakultät für Biologie, Abteilung Geobotanik Albert-Ludwigs-Universität Freiburg

INHALT

Das Skript wurde in Anlehnung an die hinten genannten Lehrbücher und Skripten entwickelt und stützt sich dazu auch auf die Erläuterungen zu den verwendeten R-Paketen. Insbesondere das Kapitel „Datenaufbereitung für R“ wurde in enger Anlehnung an Crawley 2005 erstellt.

Einführung in R[1]

Zur Geschichte von R: R entstand aus eine freien Version des Statistik-Software Programmes S und S-Plus, dessen Kommandos und Dokumentation auch heute noch einwandfrei für R nutzbar sind. Freie Software heisst dass diese Software frei genutzt werden kann aber auch der Quellcode eingesehen und verändert werden kann außerdem kann diese Software frei weiterverteilt werden, R unterliegt der *GNU General Public Licence*.
Dies macht R nicht nur besonders interessant für Mittellose sondern auch für die Entwicklung wissenschaftlicher statistischer und mathematischer Anwendungen, daher gibt es für R mittlerweile eine kaum überschaubare Anzahl an „*Contributed Packages*" d.h. kleinen integrierbaren Programmpaketen, die in den Bereichen multivariate Statistik; Gen-Sequenz-Analyse und Geostatistik viele der neueren und gängigeren statistischen Methoden abdecken; fehlende Funktionen lassen sich in der „R-Sprache" oder in C programmieren und leicht in das Programm einbinden. Durch die hohe Anzahl der Entwickler und Nutzer gibt es sehr viel frei zugängliche Dokumentation; Hilfe-Mailinglisten und ein enormes Entwicklungstempo. R läuft auf den Betriebssystemen Windows, Linux/UNIX und Macintosh. In diesem Kurs benutzen wir vor allem die Grundversion und die Pakete `vegan`, `cluster` und ggf. `ade4`, diese decken den Großteil der in der neueren ökologischen Literatur häufiger verwandten multivariaten Methoden ab. Ein Tipp zu diesem Skript: alle in Antiqua geschriebenen Teile sind Erläuterungen und Kommentare; alle in `Courier` geschriebenen Teile sind Befehle (oder R-Paket-Bezeichnungen) und können so in die Konsole eingegeben werden. Die R-Konsole ist der Hauptarbeitsbereich; es ist eine Art Fenster in den Befehle eingegeben werden können und die -wenn vollständig- nach Drücken der return/enter Taste vom Programm umgesetzt werden. Auch „copy-paste" funktioniert mit R ! Alternativ lassen sich auch alle Befehle in einem kleinen Text ablegen und durch den Befehl `source()` in dem Programm ausführen. Aber jetzt einfach ein paar Beispiele:

```
read.csv(file="Tabellenobject.csv")->object
# Erläuterung der Funktionsweise eines R-Moduls: 1.Name des Moduls, 2.In Klammern
# stehen Argumente und Operationen, 3. mit einem Pfeil wird das Ergebnis in ein Objekt
# abgelegt

fix(object)
# Öffnet eingeladene Objekte in einem Editor, Veränderungen werden durch schliessen
# gespeichert

3*10/3+1-1^10
#einfache Mathematische Grundfunktionen in R: Multiplikation, Division, Addition etc

# Standardeingabe und Standardausgabe
3*10->ergebnis # Zuweisungspfeil: die Ausgabe wird in ein Objekt geschrieben

ergebnis # „return" oder „enter": der Inhalt wird in der Konsole ausgeschrieben, das kann
bei großen Objekten sehr lang sein !

# es gibt fünf grundsätzliche Datentypen in R: logical, integer, real, complex, und string
(oder character),
wir beschränken uns auf drei Typen:
Zahlen (integer + real beide auch als numeric)
Zeichen(ketten) (character oder string) und
```

[1] http://www.r-project.org

Logische Typen

```
zahlen <- c(-3, 1, 6, 7.5, 8.999)

typen <- c("Boot","Baum","Stein","Fluss","Muschel")

welche <- c(TRUE,FALSE,TRUE,FALSE,TRUE)

# Indexvektoren sind logische Vektoren  mit denen sich Teile auswählen lassen, sie stehen in
# eckigen Klammern

typen[welche]

# dazu muss der Indexvektor natürlich gleich lang sein wie der Vektor aus dem ausgewählt
# wird!

length(welche)
sum(welche)

# sie können hilfreich sein um Objekte auszuwählen:

typen[zahlen>3]
zahlen[zahlen>3]

# damit lassen sich sehr leicht ALLE Zeichen in einem Vektor durch ein anderes ersetzen;
# angenommen wir haben eine Braun-Blanquet-Aufnahme, die folgendermaßen aussieht
# (ohne Arten):

aufnahme<-c(1,.,.,1,+,1,2,.,+,4)

# kann R natürlich mit den Symbolen + und . nichts anfangen (es sind ja keine Zahlen!) um
# aber statistische Analysen durchführen zu können oder um diese Aufnahme in 0|1 umzu-
# wandeln muss dieser Vektor in eine Zahlenfolge umgewandelt werden; dazu kann man
# folgende Kommandos nutzen:

aufnahme[aufnahme==.] <- 0
aufnahme[aufnahme==+] <- 0.5

# Das Ergebnis kann man einfach folgendermaßen anschauen

aufnahme # danach return und der Inhalt entleert sich in die Konsole !
1 0 0 1 0.5 1 2 0 0.5 4

# Dieses sogenannte *code replacement* muß in R also von Hand in Einzelschritten durchge-
# führt werden!

# Über diese grundlegenen Datentypen gibt es auch Objekttypen in großer Anzahl, solche
einfachen wie oben durch zahlen <-c(...,...,...) gezeigt; heissen Vektoren also
eindimensionale Reihen, Matrix nennt man zweidimensionale „Tabellen“, die
Verallgemeinerung in n-Dimensionen nennt man einen „n-dimensionalen Array“

help()        # Zur Anzeige der Hilfe zu einem Modul dessen Name man genau kennt
?function             # entspricht dem Kommando help(funktion)
```

```
help.search()      # Zur Suche nach Stichworten in den Hilfeseiten
ls()               # zeigt alle Objekte an die im Hintergrund gespeichert sind
ls(pat="en")       # zeigt alle Objekte deren Namen ein "en" enthalten
```

Data-Frames sind Tabellen auf deren Spalten sich leichter zugreifen lässt, durch das `$` Zeichen zwischen DataFrameName und Spaltentitel ist diese Spalte einwandfrei bezeichnet und es kann damit gerechnet werden !

NA ist das Zeichen das auftaucht wenn zwischen Zahlen ein Wert fehlt oder versucht wird mit Typen zu rechnen, was nicht geht, alle Operationen mit NA ergeben wieder NA, bei vielen Funktionen gibt es daher entweder die Möglichkeit NAs nicht zu berrücksichtigen oder sie durch 0 zu ersetzen. (NA = "not available")
Am Anfang gibt es erfahrungsgemäß viele „Probleme“ mit NAs, das beruht darauf dass beim Import oft Fehler auftreten und Zahlen als Typen importiert werden, mit `fix()` lässt sich überprüfen ob eine Variable `numeric` oder `character` ist und der Typ läßt sich von Hand ändern (auf den Spaltentitel klicken und Typ ändern) oder ein Neuimport mit der Option in `read.csv(...,colClasses="numeric")`; die alle Spalten gezwungen als Zahlen importiert, weitere Details siehe Hilfeseite des `read.csv`-Moduls.

```
hist()             # zur schnellen Übersicht der Verteilung einer numerischen Variable
library(vegan)     # nicht alle Funktionen sind im Basispaket enthalten
decostand()        # für Transformationen
write.table()      #zum Export
source()           # zum Einlesen von Skripten d.h. einer Liste von Kommandos die
                   # direkt auszuführen sind
```

`plot()` # die generelle graphische Darstellungsmöglichkeit funktioniert mit fast allen Analysemodulen irgendwie! Parameter wie Achsenlängen, Beschriftung Zeichengröße usw. können einzeln festgelegt werden. Es ist auch möglich Grafiken nach und nach aufzubauen oder mehrere auf einer Seite anzuordnen.

`par(mfrow=c(3,2))` # par wird verwendet um Grafikparameter zu setzen. Mit mfrow werden Zeilen und Spalten der Grafikausgabe eingestellt.

Export über den GUI (*graphical user interface*: die „Windows-Klick-Oberfläche“) von Bildern als .emf Durch das .emf – Format sind die Graphiken dann in Word und Powerpoint einfacher weiter zu verarbeiten.[2]
Eine sehr gute Einführung in R bietet die freie Dokumentation „An introduction to R“ in den html-Hilfeseiten des Programmes, diese finden sich im GUI unter „HELP“ dann „html Help“ oder im Internet.

[2] Die Güte der Grafik hängt von der Bildschirmausgabe ab. Für Druckvorlagen sollte der Weg über das Postscript-Format beschritten werden!

Datenaufbereitung für R

(in enger Anlehnung an Crawley 2005)

Wenn man mit R arbeiten will, muss man wissen, in welcher Form die Daten organisiert sein müssen, damit auch alles so läuft, wie es soll. Ein weiterer wichtiger Punkt ist, wie man die Daten in R hineinbekommt (siehe nächster Abschnitt).

R arbeitet mit sog. „**dataframes**". Diese bestehen aus Zeilen und Spalten. Die Zeilen enthalten verschiedene Objekte (Beobachtungen/ Messungen/ Aufnahmen etc.), die Spalten enthalten die Werte der Variablen, die die Objekte beschreiben. **Entscheidend ist, dass alle Werte einer Variablen in einer Spalte stehen!** Das klingt banal, manchmal neigt man aber dazu, seine Daten anders darzustellen. Stellen Sie sich vor, Sie hätten in verschiedenen Wiesen, die gar nicht, früh im Jahr oder spät im Jahr gemäht wurden, Daten erhoben. Das könnte man folgendermaßen darstellen:

nicht gemäht	früh gemäht	spät gemäht
15	10	19
17	7	23
13	9	25
14	13	23
19	10	30

In dieser Tabelle stehen nicht alle Werte einer Variablen in einer Spalte! Für R muss die Tabelle folgendermaßen aussehen:

nicht gemäht	15
nicht gemäht	17
nicht gemäht	13
nicht gemäht	14
nicht gemäht	19
früh gemäht	10
früh gemäht	7
früh gemäht	9
früh gemäht	13
früh gemäht	10
spät gemäht	19
spät gemäht	23
spät gemäht	25
spät gemäht	23
spät gemäht	30

Nun steht in jeder Spalte eine Variable, in der ersten das Mahdregime, in der zweiten die erhobenen Daten.

Wenn man Vegetationsaufnahmen untersucht, bedeutet das, dass nicht wie sonst üblich, die Aufnahmen in den Spalten und die Arten in den Zeilen stehen, sondern dass es in R genau umgekehrt ist: die Aufnahmen stehen in den Zeilen (jede Zeile ein Objekt), und die Arten stehen in den Spalten (jede Spalte enthält eine beschreibende Variable)[3].

Eine Möglichkeit, Zeilen und Spalten zu vertauschen, ist die Daten in Excel zu kopieren und in ein neues Excelblatt einzufügen, wobei man Zeilen und Spalten vertauscht. Man kopiert also den entsprechenden Bereich und merkt sich, wie viele Zeilen und Spalten man hat (z.B. 15 Zeilen und 20 Spalten). Dann fügt man diesen Bereich wie folgt in ein neues Excelblatt ein: unter Bearbeiten wählt man die Option „Inhalte einfügen…" und aktiviert dort das Kästchen „transponieren". Schon sind Zeilen und Spalten vertauscht.

[3] Es ist natürlich auch möglich, dass die Arten die Objekte sind, die man untersucht, und sie durch die Aufnahmen beschrieben werden, in denen sie vorkommen. Dann stehen die Arten in den Zeilen und die Aufnahmen in den Spalten.

Die andere Möglichkeit: der Befehl zum transponieren in R heißt ganz einfach `t()`, wenn man also Zeilen und Spalten eines Datensatzes vertauschen möchte, geht das so:

```
t(datensatz) -> t.datensatz
```

Einlesen von Daten aus Excel

Hat man einen Datensatz in Excel vorliegen, in dem nicht in allen Feldern Werte eingetragen sind, kann man das ändern, bevor man die Daten einliest (wie man das nach dem Einlesen ändert, wird später erklärt). In einer Tabelle mit Vegetationsaufnahmen wird für Arten, die in einer Aufnahme nicht vorkommen, kein Eintrag gemacht. R würde dies als „fehlenden Wert" interpretieren (NA = not available). Also ersetzt man alle Leerstellen wie folgt mit null: man markiert den entsprechenden Bereich in Excel, dann wählt man unter Bearbeiten → Suchen… und dort die Karteikarte „Ersetzen". Im Feld „Suchen nach" macht man keinen Eintrag (noch nicht einmal ein Leerzeichen), bei „Ersetzen durch" gibt man 0 ein. Dann geht man auf „Alle ersetzen". War in irgendeinem Feld ein Leerzeichen, muss man das noch einmal separat ersetzen.

Liegen die Daten in Excel vor, muss man das entsprechende Tabellenblatt in einem Format abspeichern, das R lesen kann.
Sie können unter „Datei → Speichern unter…" zwischen „Text (Tabstopp-getrennt)(*.txt)" und „CSV (Trennzeichen-getrennt)(*.csv)" wählen. Wenn Sie sich für erste Variante mit den Tabstopps entscheiden, müssen Sie darauf achten, dass nirgendwo sonst in der Tabelle Leerzeichen auftauchen (z.B. in Variablennamen), da diese sonst von R auch als Trennzeichen interpretiert werden.

Zum Einlesen der Daten dienen die Befehle `read.table`, `read.csv` und `read.csv2` (hier sind jeweils die wichtigsten Optionen angegeben, alle Optionen finden Sie in der Hilfe von R.):

```
read.table(file, header = FALSE, sep = "", dec = ".", row.names, nrows = -
1,skip = 0)
```
[4]

- `file`: der Name der Datei, die eingelesen werden soll. Wenn die Datei sich in Ihrem R-Arbeitsverzeichnis[5] befindet, reicht der Name, sonst muss der ganze Pfad eingegeben werden. Wenn Sie den ganzen Pfad eingeben, muss dieser in Anführungszeichen stehen und Sie müssen je einen doppelten Backslash (\\) verwenden.
- Mit `header = TRUE` oder `header = FALSE` geben Sie an, ob die erste Zeile die Namen der Spalten enthält oder nicht. Wenn Sie nichts angeben, nimmt R an, dass es keine Spaltennamen gibt, es sei denn, die erste Zeile (und wirklich nur die erste) enthält ein Element weniger als alle anderen.
- `sep = ""` gibt an, dass als Trennzeichen ein Leerzeichen verwendet wird. Das Trennzeichen trennt die Werte in einer Zeile voneinander. Hat man ein anderes Trennzeichen, muss man dieses hier angeben, oder den entsprechenden der beiden anderen Befehle verwenden (dann muss man das Trennzeichen nicht jedes Mal im Befehl ändern).

[4] Die Einstellungen, die hier angegeben sind die Standardeinstellungen zu den Befehlen. Gibt man nicht anderes an, werden diese verwendet.
[5] Welches Ihr Arbeitsverzeichnis ist, erfahren Sie durch den Befehl `getwd()`. Ein neues Arbeitsverzeichnis setzen können Sie mit `setwd()`. Dafür geben Sie in der Klammer den gewünschten Pfad in Anführungszeichen ein.

- `dec = "."` setzt als Dezimaltrennzeichen den Punkt. Hier gilt ebenfalls: wenn man Kommas verwendet, muss man den Punkt in ein Komma ändern, oder man benutzt einen anderen Befehl, bei dem das Komma schon als Dezimaltrennzeichen festgelegt ist.
- `row.names` gibt einen Vektor für die Zeilennamen an. Dies kann entweder ein Vektor mit den Zeilennamen sein (etwa `row.names = c("Alter", "Größe","Gewicht", "Haarfarbe", "Augenfarbe"`) oder eine einzelne Zahl, die die Spalte in der Tabelle angibt, in der die Zeilennamen stehen (meistens ist das wohl die erste). Wenn es Überschriften für die Spalten gibt und die erste Spalte ein Element weniger enthält als die übrigen Spalten, wird die erste Spalte automatisch für die Zeilennamen genommen. Sonst werden die Zeilen durchnummeriert. Bei `row.names = NULL` werden die Zeilen auch durchnummeriert.
- `nrows = -1` gibt die maximale Zahl der Zeilen an, die eingelesen werden sollen. Negative und andere ungültige Zahlen werden ignoriert.
- `skip = 0` gibt die Zahl der Zeilen an, die übersprungen werden sollen, bevor die Daten eingelesen werden.

→ Benutzen Sie `read.table`, wenn Sie ihre Daten in Excel als „Text (Tabstopp-getrennt)(*.txt)" gespeichert haben.

Die anderen beiden Befehle unterscheiden sich von read.table nur durch die Standardeinstellungen. Wenn Sie Ihre Excel-Datei also als mit Semikolons getrennt gespeichert haben, können Sie sie leichter mit `read.csv2` einlesen.

```
read.csv(file, header = TRUE, sep = ",", dec=".")

read.csv2(file, header = TRUE, sep = ";", dec=",")
```

→ Benutzen Sie `read.csv2`, wenn Sie ihre Daten in Excel als „CSV (Trennzeichen-getrennt)(*.csv)" gespeichert haben.

Es empfiehlt sich – vor allem bei sehr großen Datensätzen, aber auch bei kleinen, wenn man damit weiterrechen will – dem Datensatz gleich einen Namen zuzuweisen. Dies geschieht einfach mit einem Zuweisungspfeil:

```
Datensatzname <- read.csv2(file, header = TRUE, sep = ";", dec=",")
```

oder andersherum

```
read.csv2(file, header = TRUE, sep = ";", dec=",") -> datensatzname.
```

Mit `fix(datensatzname)` können Sie sich Ihre Tabelle noch einmal ansehen. Änderungen, die Sie in der erscheinenden Tabelle machen, werden beibehalten! Wenn Sie auf die Spaltennamen klicken, öffnet sich ein Fenster, in dem Sie den Typ der Werte (numeric oder character) ändern können, oder auch einen anderen Variablennamen eingeben können.

Wenn in Ihrem Datensatz noch „NA"- Einträge auftauchen, weil an der entsprechenden Stelle im Datensatz kein Eintrag vorhanden war, können Sie diese mit folgendem Befehl durch 0 ersetzen:

```
datensatzname[is.na(datensatzname)]<-0
```

Auslesen von Daten aus R

Möchte man seine Daten aus R wieder in Excel haben, ist der einfachste Befehl:

```
write.csv2(datensatzname, file)
```

Bei `file` geben Sie das Verzeichnis an (in Anführungszeichen, mit doppeltem Backslash (\\)), in das die Datei geschrieben werden soll. R erzeugt dort eine Datei, die sie ganz einfach mit Excel öffnen können. Sie können diese Datei dann auch wieder als .xls-Datei speichern (Speichern unter….).

Zugriff auf Teildaten eines Datensatzes in R

Wenn Sie Ihre Daten eingelesen haben, können Sie über den Befehl
`attach(datensatzname)`
den Zugriff auf die einzelnen Variablen über den Variablenamen möglich machen.

Mit dem Befehl
`names(datensatzname)`
wird eine Liste mit allen Variablennamen in dem Datensatz ausgegeben.

Sie können aber auch über die Verknüpfung
`Datensatzname$variable`
auf die Variable zugreifen

Manchmal möchte man vielleicht nur Teile eines Datensatzes, den man schon in R eingelesen hat, betrachten. Dafür gibt es in R ein paar elegante Möglichkeiten. Der Zugriff erfolgt über Angabe der gewünschten Teile in eckigen Klammern:

`[Elemente]`
`a[12]` ist das 12. Element des Vektors a.
`a[-12]` ist die Variable a ohne das 12. Element.

`[Zeile, Spalte]`
`b[2,5]` ist das Element in der 2. Zeile der 5. Spalte eines dataframes b.

Wenn Sie aus einem Datensatz eine Spalte, aber alle Zeilen auswählen möchten, geben Sie die Spalte an und lassen den Platz für die Zeile leer: `b[,5]` ist die komplette 5. Spalte des Datensatzes b.

Wenn Sie eine bestimmte Zeile möchten und alle Spalten, lassen Sie den Platz für die Spalte leer: `b[2,]` ist die komplette 2. Zeile des Datensatzes b.

Wenn Sie nun mehrere Spalten oder Zeilen auswählen möchten können dieses über die Angabe der Zeilen bzw. Spalten „von:bis“ (dadurch wir eine Sequenz von ganzen Zahlen erstellt, also ergibt z.B. `1:10` die Zahlenfolge 1,2,3,4,5,6,7,8,9,10). Die Zeilen 3 bis 5 und die Spalten 7 bis 18 eines Datensatzes b wählt man also über `b[3:5,7:18]` aus. Natürlich kann man auch den Platz für die Zeilen bzw. Spalten frei lassen, wenn man alle Zeilen bzw. Spalten auswählen möchte:
`b[3:5,]` sind alle Spalten der Zeilen 3 bis 5,
`b[,7:18]` sind alle Zeilen der Spalten 7 bis 18.

Es ist auch möglich, wenn man eine bestimmte Spalte möchte, den Spaltennamen (sofern es einen gibt) anzugeben: `b[,Länge]` bezeichnet alle Zeilen der Spalte „Länge“ des Datensatzes b.

Manchmal kann es nötig sein, Zeilen oder Spalten aufgrund bestimmter Kriterien auszuwählen. Wenn Sie nun etwa aus dem Datensatz b alle Zeilen hätten, bei denen die Variable „Länge“ Werte von mindestens 10 aufweist, erhalten Sie diese durch `b[,Länge >=10 ]`.

Transformationen in R

Transformationen werden häufig in der Datenanalyse gebraucht um beispielsweise eine nicht-normalverteilte Variable in eine normalverteilte zu überführen. Transformationen lassen sich in zwei Typen unterteilen: Skalartransformationen und Vektortransformationen. Skalartransformationen wandeln jeden Wert einzeln und unabhängig von den übrigen Werten um und sind teilweise rückführbar, vorrausgesetzt man weiß, welche Methode verwendet wurde!

Der erste Schritt betreffs Transformationen ist die Wahl der Codierung der Deckungsklassen (*Code replacement*). Wählt man die Mitte der Klassen als Deckungsanteil in Prozent oder wählt man die Rangstufe der Klasse? Entsprechend spannt man einen Wertebereich von 0-100 oder von 0-9 auf.

***Code replacement* für eine Matrix/ einen Dataframe**

Mit der Funktion read.table werden die Daten als data.frame eingelesen. Zunächst erstellt man eine Matrix, die genauso viele Zeilen (z) und Spalten (s) hat, wie der eingelesene data.frame, die Matrix enthält zunächst nur Nullen:

```
tabelle <- matrix(0,z,s)
```

Dann werden die Werte aus der eingelesenen Originaltabelle („tabelle.original“) umgewandelt, aber nicht in den Originaldaten selber, sondern in der eben neu erstellten Matrix „tabelle“:

```
tabelle[tabelle.original=="5"] <- 88
tabelle [tabelle.original =="4"] <- 63
tabelle [tabelle.original =="3"] <- 38
tabelle [tabelle.original =="2"] <- 15
tabelle [tabelle.original =="1"] <- 3
tabelle [tabelle.original =="+"] <- 2
tabelle [tabelle.original =="r"] <- 1
```

In der Matrix namens "tabelle" stehen nun Prozentwerte. Diese Matrix wird nun in einen "data.frame" umgewandelt:

```
data.frame(tabelle) -> tabelle.df
```

Jetzt fehlen noch die Artnamen, die wir aus dem ursprünglich eingelesenen Dataframe "tabelle.original" nehmen,

```
names(tabelle.df) <- names(tabelle.original)
```

und die Aufnahmenummer (die row.names), die wir ebenfalls aus dem eingelesenen Datensatz nehmen:

```
row.names(tabelle) <- row.names(tabelle.original)
```

Skalartransformationen (Transformationen *s.str.; monotonic transformations*) sind zum Beispiel Mutliplikation, Addition, Division, Subtraktion einer Konstanten, Potenztransformation, Logarithmische Transformationen und die Binärtransformation (diese ist nicht umkehrbar).

Vektortransformationen (Relativierungen; *relativizations*) betrachten die anderen Werte einer Variable oder gar die ganze Matrix; dazu gehören Zentrierung (Mittelwert = 0); Relativierun-

gen *s.str* (alle Werte zwischen 0 und 1) und Standardisierung (Mittelwert = 0 und Standardabweichung = 1).

Die **Skalartransformationen in R** sind meist einfach durch die entsprechenden mathematischen Funktionen zu erreichen (vorrausgesetzt alle Elemente der Matrix sind Zahlen „`numeric`"!)

```
zahlen*10->zahlen.verzehnfacht
zahlen+1->zahlen.pluseins
log(zahlen)->zahlen.log # natürlicher Logarithmus beachte aber Werte unter Null !
log10();logb(,base=...) # dekadischer Logarithmus, Logarithmen mit beliebiger Basis
sqrt(zahlen)->zahlen.wurzel
zahlen^2->zahlen.quadrat
```

```
library(vegan) # die Binärtransformation lässt sich gut mit einem vegan-Modul erreichen:
decostand(zahlen,"pa")->zahlen.pa # für Binärtransformation: > 0 → 1; ≤ 0 → 0
```

```
zahlen.arcsin <- 2/pi*asin(sqrt(Deckung/100)) # Arcus-Sinus Transformation
```

Hierbei ist zu beachten, dass die Deckungswerte des Eingangsdatensatzes auf dem Wertebereich [0;1] liegen sollten und dass Winkelwerte in R grundsätzlich im Bogenmaß angegeben werden.

Vektortransformationen in R sind durch das Modul `decostand()` aus dem Paket `vegan` und durch das Modul `scale()` einfach durchzuführen, bei decostand lässt sich mit der Option MARGIN=1 (Zeilen) oder MARGIN=2 (Spalten) entscheiden ob sich die Vektortransformation auf die Spaltensumme/-mittelwert/-maximum etc. bezieht oder auf die Zeile; der default ist mit * angegeben:

Zentrierung:
`scale(zahlen, center=TRUE)->zahlen.zentriert` # *Spalten; wenn Zeilen dann t(zahlen) statt zahlen benutzen !

Standardisierung:
`decostand(zahlen,"standardize")->zahlen.standardisiert` # *Spalten

Relativierung (Standardisierung auf Werte zwischen 0 und 1):
`decostand(zahlen,"range")->zahlen.relativiert` # *Spalten

Chi-Quadrat Transformation:
`decostand(zahlen,"chi.square")->zahlen.chi2transformiert` # * Zeilen

Teilen durch das Maximum:
`decostand(zahlen,"max")->zahlen.maxgeteilt` # *Spalten

Teilen durch die Summe:
`decostand(zahlen,"total")->zahlen.summengeteilt` # *Zeilen

Binär mit Median:
`zahlen.bmed <- as.numeric(zahlen>median(zahlen))` # pro Art

Doppelte Relativierung
`zahlen.dr <- wisconsin(zahlen)` # Normieren auf das Maximum der Spalten und
Teilen durch die Zeilensummen

Distanzmaße in R

Mit dem Befehl `vegdist()` aus dem Package vegan (Paket laden: `library(vegan)`) lassen sich u.a. folgende Distanzmaße berechnen[6]:

Euklidische Distanz („`euclidian`“),
Manhattan-Distanz („`manhattan`“),
Bray-Curtis-Distanz („`bray`“)
(für weitere Distanzmaße siehe Dokumentation des Moduls mit `help(vegdist).`

```
vegdist(aufnahmen, method="bray", binary= FALSE) -> distanz.aufnahmen
```

Ebenfalls lässt sich die **Jaccard-Distanz** berechnen, dabei ist aber zu beachten, dass R hier eine qantitative (d.h. es wird mit Häufigkeiten gerechnet) Version berechnet. Durch den Zusatz `binary = T` wird die Distanz für presence/ absence-Daten errechnet:

```
vegdist(aufnahmen, method="jaccard", binary=T) -> distanz.jac
```

Wenn man nun den Jaccard-Index als Ähnlichkeitsmaß berechnen will, muss man die Jaccard-Distanz noch umrechnen: `1 - distanz.jac -> index.jac`

Die **Sørensen-Distanz** entspricht der Bray-Curtis-Distanz über presence/absence transformierte Daten:

```
vegdist(aufnahmen, method="bray", binary=T) -> distanz.soer
```

Durch die Einstellung `binary=T` transformiert R die Daten in presence/absence-Daten, dann berechnet `vegdist()` daraus die Bray-Curtis-Distanz; diese ist gleich der Sörensen-Distanz wenn es nur Nullen und Einsen gibt! Wenn man nun den Sørensen-Index als Ähnlichkeitsmaß berechnen will, muss man die Sørensen-Distanz noch umrechnen: `1 - distanz.soer -> index.soer`

Darüberhinaus gibt es auch das Modul `dist()` mit den Distanzmaßen:
`euclidean'; `manhattan'; sind denen aus `vegdist()` gleich, darüberhinaus gibt es noch:
`maximum': Maximale Distanz zwischen zwei Komponenten von x und y (Supremum Norm)
`canberra': Summme von (|x_i - y_i| / |x_i + y_i|). Terme der Summe in denen Nenner oder Zähler Null sind werden als fehlende Werte behandelt und einfach weggelassen
`binary': (aka asymmetric binary): Die Vektoren werden als binäre Bits betrachtet, also Null heißt "AUS" und Nicht-Null heisst "AN"; Die Distanz ist dann der Anteil der Bits beider Vektoren mit nur einem "AN" and denen wo mindestens einer der beiden Vektoren ein „AN“ hat.

PS: Es gibt in den Modulen zur Klassifikation `agnes()` und `diana()` aus dem Packet `cluster` die Möglichkeit für die jeweilige Klassifikation zwischen zwei Methoden (Euklidische Distanz und Manhattan) zu wählen, diese Module akzeptieren sowohl direkt Aufnahmematrizen (Tabellen) als auch Distanzmatrizen, mehr dazu im nächsten Kapitel !

[6] Die Distanz wird zwischen den Zeilen eines Datensatzes berechnet, d.h. also bei Vegetationstabellen, dass die Arten in Spalten und die Aufnahmen in Zeilen stehen müssen. Ist dies noch nicht der Fall, der Datensatz aber schon in R eingelesen, kann man die Tabelle mit dem Befehl `t()` transponieren, also etwa `t(vegetationstabelle) -> aufnahmen`.
Dabei kann man diesen Befehl auch in die Berechnung des Distanzmaßes einbauen:
`vegdist(t(vegetationstabelle), method="bray", binary= FALSE) -> distanz.aufnahmen`.

Klassifikation in R

Allgemein kann die Klassifikation meist in 3 Schritte eingeteilt werden:
1. Berechnung von Distanzen zwischen den Objekten, (siehe vorheriges Kapitel).
2. Konstruierung der Beziehungen zwischen Objekten (bei hierarchischen Verfahren Dendrogramme/ Bäume) nach einer „clustering strategy“
3. Darstellung der Strukturen, dies geht meist einfach durch `plot()` unter Angabe des Objektes aus dem zweiten Schritt

Agglomerative Verfahren

Das klassische Modul zur Herstellung hierarchisch-agglomerativer „Bäume“ aus Distanzmatrizen ist das modul `hclust(distanzmatrix, method = "complete")` aus dem Paket `vegan.` Hierbei gibt es verschiedene wählbare Verfahren (unter der Angabe `method=„...“`).

Folgende Verfahren sind wählbar:
`"ward"` – Ward’s Methode
`"single"` – single linkage (= nearest neighbour; immer zum nächsten Nachbar)
`"complete"` – complete linkage (= farthest neighbour; zum entferntesten Nachbar)
`"average"` – average linkage (zum Mittelpunkt des Nachbar-Clusters)
`"mcquitty"` – Verfahren nach McQuitty
`"median"` – Median oder Gower-Verfahren
`"centroid"` – centroid linkage (zum gewichteten Gruppenmittel)
Der „default“ d.h. die Methode die das Modul benutzt wenn keine angegeben wird ist das complete-linkage

Das Modul `agnes()` aus dem Packet `cluster` bietet ähnliche agglomerative Verfahren; also: `"average"` (average linkage), `"single"` (single linkage), `"complete"` (complete linkage), `"ward"` (Ward's Methode) und `"weighted"` (weighted average linkage). Außerdem ist das sog. „flexible clustering“ möglich, bei dem mit dem Befehl `par.method` die Koeffizienten der kombinatorischen Grundgleichung („Lance-Williams formula“) selbst gesetzt werden können.
Der *default* ist das sog. average linkage.

Darstellung und Weiterverarbeitung von Klassifikationsergebnissen

Mit `plot()` lässt sich ein Objekt, das von `hclust()` erzeugt wurde, als Dendrogramm darstellen. Dabei kann die Beschriftung der einzelnen Äste des Dendrogramms selbst gewählt werden:

- `labels =` Ein Vektor mit der Beschriftung für die einzelnen Äste des Baumes. Als *default* werden die Zeilennamen oder –nummern genommen. Mit `labels = NULL` werden keine Beschriftungen angezeigt.

`rect.hclust(dendrogramm*)` zeichnet -wählbar- Kästchen um Gruppen. Dabei sind folgende Angaben möglich:

`k = {Anzahl der gewünschten cluster},`
`h = {Höhe an der der Baum abgeschnitten werden soll}.`
`which = {Vektor, der die gewünschten Gruppen bezeichnet}`, wenn man von fünf Gruppen etwa die erste und letzte möchte, dann gibt man `which = c(1,5);`
weiteres siehe in `help(rect.hclust).`

* Der Name des Dendrogramms, also der, den man auch bei `plot()` verwendet hat.

`cutree(dendrogramm*)` schneidet das Dendrogramm in Teile, dabei lässt sich der Baum in entweder eine angegebene Zahl von Gruppen durch die Option k = wählen oder an einer bestimmten Höhe mit h = schneiden, das Ergebnis ist ein Vektor mit der Zugehörigkeit der Objekte zu einer Gruppe; dieser Vektor hat die gleiche Reihenfolge wie die Objekte die für die Distanzmaßberechnungen oder Klassifikation benutzt wurden.

Divisive Verfahren:

Divisive Cluster-Verfahren sind im Paket `cluster` enthalten. Insbesondere die Module `mona()` und `diana()` sind hier zu erwähnen. Diese akzeptieren sowohl Vegetationstabellen als auch Distanzmatrizen. Bei direkter Auswertung von Vegetationstabellen kann zwischen Euklidischer und Manhattan-Distanz gewählt werden. Bei der Eingabe einer Distanzmatrix ist man dagegen in der Wahl des Maßes frei. Die Ergebnisobjekte können für die Weiterverarbeitung (v.a. bzgl. Grafik) mit der Funktion `as.hclust()` in ein hclust-artiges Objekt umgewandelt werden.
k-Means-Clustering kann sowohl mit dem Modul `kmeans()` aus dem Paket `stats` als auch mit dem Modul `pam()` aus dem Paket `cluster` durchgeführt werden. Das Ergebnisobjekt aus `pam()` kann zur Darstellung eines sog. Silhouette-Plots genutzt werden, das die Güte der Zuordnung der einzelnen Elemente zu Clustern beschreibt.

Weiterhin enthält das Paket `cluster` auch das Modul `fanny()` zur Fuzzy Klassifizierung.

TWINSPAN ist in R bisher nicht implementiert; aber in dem frei erhältlichen und nutzbaren Programmpaket JUICE enthalten. Außerdem existiert eine frei verfügbare Windows-Version.

---

* Der Name des Dendrogramms, also der, den man auch bei `plot()` verwendet hat.

Ordination in R

Es gibt mehrere Pakete die Ordinationsverfahren enthalten. Am wichtigsten sind `mva`, `vegan` und `ade4`. Es lohnt sich einmal mit `help.search("")` nach den gängigen Verfahren (PCA, CA, CCA, nMDS, PCoA, RDA, DCA, polar ordination) zu suchen und sich die entsprechenden Hilfeseiten mit `help()` anzuschauen.
Bray-Curtis oder polare Ordination ist beispielsweise in R noch(?) nicht programmiert.

PCA – Hauptkomponentenanalyse – principal component analysis – in R

Zum Durchführen der PCA in R gibt es verschiedene Module (`prcomp, princomp, dudi.pca, rda ...`); wir verwenden `rda()` aus `vegan`, da der Ergebnistyp gleich aussieht, wie der von den Funktionen, die wir für CA und DCA verwenden werden.

Eine erste Hauptkomponentenanalyse in R mit dem Beispieldatensatz an Umweltvariablen aus dem Paket vegan:

```
library(vegan)
data(dune.env)
rda(dune.env) # Die Funktion rda() ebenso wie princomp() kann kategoriale Variablen
```

nicht verarbeiten. Daher müssen die kategorialen Variablen in Dummy-Variablen umkodiert werden.

Umkodierung kategorialer Variablen in Dummy Variablen

`dune.env$Use=="Pasture"` # dies zeigt `TRUE` an, wo der Wert `"pasture"` in dem character-Vektor `$Use` steht

`as.numeric(dune.env$Use=="Pasture")->pasture` # die innere Funktion schreibt einen `TRUE/FALSE` Vektor der von der äußeren Funktion (`as.numeric()`) in einen numerischenVektor umgesetzt und anschließend dem Objekt `pasture` zugewiesen wird.
`pasture`

#und so gehen wir mit jedem Zustand der kategorialen Variablen vor; sie müssen **alle** durch dummy-Variablen ersetzt werden ! (sonst kann man keine PCA darüber schicken!)

```
as.numeric(dune.env$Use=="Pasture")->pasture
as.numeric(dune.env$Use=="Haypasture")->haypasture
as.numeric(dune.env$Use=="Hayfield")->hayfield
as.numeric(dune.env$Management=="BF")->BF
as.numeric(dune.env$Management=="SF")->SF
as.numeric(dune.env$Management=="HF")->HF
as.numeric(dune.env$Management=="NM")->NM
as.numeric(dune.env$Manure)->Manure
as.numeric(dune.env$Moisture)->Moisture
as.numeric(dune.env$A1)->A1
```

#mit dem folgenden Befehl wird die Tabelle wieder zusammengebunden, und zwar spaltenweise (`cbind()` heißt: column-bind!). Die Funktion data.frame() erzeugt einen Dataframe – abhängig von den Eingabeanforderungen der Funktionen in der weiteren Analyse kann es notwendig sein, die Matrix in einen Dataframe zu überführen:

```
cbind(Moisture,Manure,A1,pasture,hayfield,BF,SF,HF)->dune.nenv
```

oder

```
data.frame(row.names=row.names(dune.env),Moisture,Manure,A1,pasture,hayfiel
d,BF,SF,HF)->dune.nenv
```
[7]
#dann die eigentliche Analyse; PCA wird mit den Rohdaten gemacht nicht an Distanzmaßmatrizen! (Gegebenenfalls sollte man darüber nachdenken, die Daten vorher zu standardisieren. Das geht entweder mit `decostand()` oder mit `rda(...,scale=TRUE))`

```
rda(dune.nenv)-> dune.nenv.pca # Das geht jetzt.
dune.nenv.pca
```

Zur grafischen Darstellung können wir auf den `plot()`-Befehl zugreifen. Dieser zeigt jedoch im Biplot keine Pfeile für die Arten, was bei der PCA angebracht wäre. Über den Parameter `type="text"` oder `type="points"` kann ein Plot mit Symbolen oder mit Labeln erzeugt werden.

Zur Erleichterung kann eine Funktion `plot.ord()` eingelesen werden, die das Setzen weiterer Grafikparameter erleichtert und standardmäßig Pfeile für die Arten zeichnet (Autor: Cord Peppler-Lisbach, Oldenburg).

Dazu kopieren sie die Datei „f.plot.ord.R" in Ihr Arbeitsverzeichnis und binden diese anschließend über den Befehl `source(„f.plot.ord.R")` in das R-Programm ein (hält ebenso wie das Einbinden einer Bibliothek/eines Pakets für den aktuellen Lauf von R).

```
plot.ord(dune.nenv.pca)
```
[8]

#für die PCA über die Arten muss es mehr Aufnahmen als Arten geben ! Daher wählen wir einfach Arten aus, die eine Abundanzsumme von mehr als 25 haben. Die Sinnhaftigkeit dieser Anforderung, die in der Funktion `princomp()` implizit ist, wird von manchen Vegetationskundlern angezweifelt. Die Funktion `rda()` hat dementsprechend diese Limitierung nicht. Gleichwohl ist zu überlegen, den Datensatz einzuschränken und z.B. alle Arten, die in weniger als 5% der Aufnahmen vorkommen, auszuschließen, da diese zwar stark zum Rauschen im Datensatz beitragen, aber wenig Erklärungswert haben.

---

[7] Mit `cbind()` wird eine Matrix erzeugt, alle Spalten sind vom selben Typ (z.B. `numeric`). Dagegen erzeugt `data.frame()` einen Dataframe, bei der die Spalten verschiedenen Typs sein können. Eine weitere Folge ist der Zugriff auf Zeilen-.und Spaltennamen. In einer Matrix (Ergebnis von `cbind()`) erfolgt der Zugriff über `dimnames[[1]]` oder `dimnames[[2]]`. In einer Tabelle vom Typ Dataframe über `row.names()` und `names()`.

[8] Falls wir `princomp()`verwenden, lassen sich die Ergebnisse einfach darstellen:

```
princomp(dune.nenv)->dune.pca # das geht jetzt!?
#zur graphischen Darstellung nutzen wir den biplot()
biplot(dune.pca) # dies macht den klassischen Biplot!
summary(dune.pca) # resümiert kurz die Analyse !
loadings(dune.pca) # dies zeigt auf welche Variablen wichtig sind bzw. mit welchen Achsen korrelieren !
princomp(dune[,(dune.s>25)])->dune.sp.pca
biplot(dune.sp.pca)
summary(dune.sp.pca)
biplot(dune.sp.pca$scores[,2:3],dune.sp.pca$loadings[,2:3]) # dies plottet 2. gegen 3. Achse; das gleiche erreicht man mit der Option choices=c(2,3):
biplot(dune.sp.pca,choices=c(2,3))
biplot(dune.sp.pca,choices=c(2,3),scale=0)
biplot(dune.sp.pca,choices=c(2,3),scale=0,xlabs=dune.env$Moisture)
```

```
data(dune)
apply(dune,MARGIN=2,sum)->dune.s
dune.s
dune.s>25[9]

dune[,(dune.s>25)]
dune.sp.pca <- rda(decostand(dune[,(dune.s>25)],“standardize“))
plot.ord(dune.sp.pca)
plot.ord(dune.sp.pca, choices=c(2,3)) # dies plottet die 2. gegen die 3. Achse
```

PCA mit Schokoladen-Datensatz

```
# Einlesen des Datensatzes (wenn Arbeitsverzeichnis gesetzt)
schoko <- read.table("schokolade.txt")

# Funktion rda aus vegan
# Werte standardisiert
schoko.pca <- rda(decostand(schoko,"standardize"))

summary(schoko.pca) # Ergebnis
plot.ord(schoko.pca) # Biplot
```

Zugriff auf Werte

```
# Aufruf der Achsenwerte für Aufnahmen und Arten
scores(schoko.pca)
# Zugriff auf versch. Ordinationsachsen über choices)
scores(schoko.pca, choices=c(2,3))
scores(schoko.pca, choices=1:4)
# Auswahl von Aufnahmen oder Arten über display
scores(schoko.pca, choices=1:4, display = „sites“)
scores(schoko.pca, choices=1:4, display = „species“)
# Zugriff auf Eigenwerte über $CA$eig
schoko.pca$CA$eig
```

[9] alternativ kann es auch sinnvoll sein, Arten auszuschließen, die nur in einigen wenigen Aufnahmen vorkommen – ungeachtet ihrer Häufigkeit:

```
apply(decostand(dune, "pa"),MARGIN=2,sum)-> dune.anz
# reduzierter Datensatz (nur Arten die in mehr als 4 Aufnahmen vorkommen):
dune.red <- dune[,dune.anz>4]
```

Detaillierter Plot

```
plot(schoko.pca, type="n")# Leerer Biplot

# Einzeichnen der Aufnahmen/Objekte
points(scores(schoko.pca,display=„sites“,cex=0.5))

# Kennzeichnen der Punkte nach Eigenschaften
points(scores(schoko.pca,display="sites"),cex=schoko$Voll*0.5,col="blue")
points(scores(schoko.pca,display="sites"),cex=schoko$Weiß*1.5,col="green")
points(scores(schoko.pca,display="sites"),cex=schoko$Dunkel,col="red")
# Eintragen der drei Faktoren
text(scores(schoko.pca,display="spec")[2:4,],
     names(schoko)[2:4],col=c("green","blue","red"))
# Einzeichnen von Pfeilen (Auswahl)
arrows(x0=0,y0=0,x1=scores(schoko.pca,display="spec")[2:4,1],
     y1=scores(schoko.pca,display="spec")[2:4,2])
```

Overlays

Overlay mit Feuchtestufen anstatt Aufnahmenummern; Overlays (Ersetzen der Aufnahmepunkte durch Werte der Umweltvariablen oder Zeigerwerte für Arten) sind DIE Interpretationsmethode für alle Ordinationen wenn wir noch nichts über die (ökologische) Bedeutung der Achsen wissen!

Zugriff auf das Symbol

1. Leeren Plot zeichnen
   `plot(schoko.pca, type="n")`
2. Einzeichnen der Objekte
   `points(scores(schoko.pca,display="sites",...)`

2.1 Wahl des Symbols
   `points(...,pch=...)`

2.2 Wahl der Größe des Symbols
   `points(...,cex=...)`

2.3 Ggf. nur Auswahl von Punkten
   `points(scores(...)[Auswahl],...)`

2.4 Beschriftung der Punkte
`text(scores...,  as.character(daten$variable))`

Loadings sagen etwas über den „Informationsbeitrag“ einer Variable zum Zustandekommen einer Achse aus, deshalb sollten sie bei einer Interpretation mitangegeben werden; hohe Werte bedeuten einen großen Beitrag, das Vorzeichen sagt etwas über die Richtung der Korrelation aus!

`scores(dune.sp.pca,display=“species“)->dune.sp.pca.loadings` #So können die Loadings in einem Ergebnisobjekt von `rda()` abgerufen werden.[10]

Scores sind einfach die Koordinaten der Aufnahmen/Samples:

`scores(dune.sp.pca,display=“sites)->dune.sp.pca.scores` #So kann auf die Scores in einem Ergebnisobjekt von `rda()` zugegriffen werden. [11]

Übrigens:

`biplot(dune.pca.scores[,1:2], dune.pca.loadings[,1:2])` # das sagt was man eigentlich bei einem biplot der ersten beiden Achsen benutzt aber

Nur bei Analysen mit der Funktion `princomp()` gilt:

---

[10] `loadings(dune.sp.pca)->dune.pca.loadings` #So können die Loadings in einem Ergebnisobjekt von `princomp()` abgerufen werden.

[11] `dune.sp.pca$scores->dune.pca.scores` #So kann auf die Scores in einem Ergebnisobjekt von `princomp()` zugegriffen werden.

```
biplot(dune.sp.pca) # macht genau das gleiche! Und ist halt etwas kürzer ;-)
```

Folgendes stellt nur die Variablen als Pfeile ohne die Objekte (*observations*) dar:

```
plot(dune.sp.pca.loadings[,1:2],pch=12)
arrows(x0=0,y0=0,x1=dune.pca.loadings[,1],y1=dune.pca.loadings[,2])
```

Ein weiteres Beispiel aus dem Kurs:

```
apply(varespec,MARGIN=2,sum)->boreal.s
boreal.s
boreal.s>25
varespec[,(boreal.s>25)]
rda(varespec[,(boreal.s>25)])->boreal.sp.pca[12]
plot.ord(boreal.sp.pca)
summary(boreal.sp.pca)
```

Vektor einzeichnen

Die Funktion `envfit()` aus `vegan` erlaubt das Anpassen von Umweltvariablen an einen Plot aus indirekter Ordination. Dazu können alle Umweltvariablen in der Datei verwendet werden oder einige ausgesucht und über eine Funktionsgleichung eingetragen werden. Das Zeichn ~ bedeutet hier das Ordinationsergebnis „`dune.pca`“ in Abhängigkeit von den Umweltvariablen „`A1` und `Moisture`“. Das Anpassen eines Vektors an das Ordinationsergebnis beinhaltet die Annahme eines linearen Zusammenhangs zwischen der Umweltvariablen und den Objekten im Ordinationsraum.

1. Plot zeichnen

```
plot.ord(dune.pca)
fit <- envfit(dune.pca,dune.env)
fit <- envfit(dune.pca~A1+Moisture,dune.env)
plot(fit)
```

Isolinien einzeichnen

Mit der Funktion `ordisurf`() können Umweltvariablen als Oberfläche an das Ordinationsdiagramm angepasst und mit Isolinien dargestellt werden. Intern werden dafür GAMs verwendet. Schränkt man die Anzahl der Freiheitsgrade ein, so nähert man sich einem linearen Modell an.

1. Plot zeichnen	`plot.ord(dune.pca)`2. Berechnen & Zeichnen
der Isolinien	`ordisurf(dune.pca,dune.env$A1)`
2.1 Wahl des Regressionsmodells	`ordisurf(..., family =...)`
2.2 Wahl der Freiheitsgrade	`ordisurf(...,knots=10)`
2.3 Anzahl der Isolinien	`ordisurf(...,nlevels=5)`

[12] `princomp(varespec[,(boreal.s>25)])->boreal.sp.pca; biplot(boreal.sp.pca)`

CA – Korrespondenzanalyse – *correspondence analysis* – in R

Auch hier gibt es mehrere Module (`cca(), duci.coa(), decorana(),...`) wir benutzen `cca()` aus dem Paket `vegan` mit dem Beispieldatensatz "`dune`"aus dem Paket `vegan`, und das Modul `decorana()` aus dem Paket `vegan`.[13]

```
library(vegan)
data(dune)                          # erst den Datensatz laden !

dune.ca <- cca(dune)                # CA rechnen
# Werte extrahieren
scores(dune.ca,display="species")->dune.ca.loadings
scores(dune.ca,display="sites")->dune.ca.scores

#Biplot zeichnen (bei CA mit Punkten statt Pfeilen für die Arten: var.axes=0)
biplot(dune.ca.scores[,1:2],dune.ca.loadings[,1:2],var.axes=0)

# mit der Option cex=0.5 werden die Kürzel auch kleiner und lesbarer!
biplot(dune.ca.scores[,1:2],dune.ca.loadings[,1:2],var.axes=0, cex=0.5)
```

In diesen beiden Plots sind die Artwerte gewichtete Mittel der Aufnahmewerte dargestellt. Man kann auch die Aufnahmewerte als gewichtete Mittel der Artwerte darstellen lassen.
Also zuerst noch mal die Artwerte als gewichtete Mittel der Aufnahmewerte:
`plot(dune.ca)`
Nun die Aufnahmewerte als gewichtete Mittel der Artwerte:
`plot(dune.ca,scaling=1)`

Die wichtigen Parameter an der sich Bedeutung und Güte der Achsen ablesen und bewerten lassen sind die *eigenvalues* und die *total intertia*, diese ist wiederum einfach die Summe der *eigenvalues*; wenn man den *eigenvalue* einer Achse durch die *total inertia* teilt und mal 100 nimmt hat man einen Prozentwert der etwas über den Informationsgehalt dieser Achse im Bezug zum gesamten Datensatz aussagt, also:

```
dune.ca$CA$eig # Zugriff auf Eigenwerte über $CA$eig
sum(dune.ca$CA$eig)
(dune.ca$CA$eig/sum(dune.ca$CA$eig))*100->dune.ca.info
```

---

[13] Bei Verwendung der Funktion `dudi.coa` aus dem Paket `ade4`:
`library(ade4); dudi.coa(dune)->dune.ca`
hier wird der/die NutzerIn gezwungen nach einem Plot der Eigenvalues der Achsen die Anzahl der später weitergenutzten Achsen gefragt; die ersten 3 oder 5 auswählen; es gibt allerdings Vorschläge welche Achsen interpretiert werden (z.B. die Abweichung v.d. broken-stick Verteilung als Regel)
`dune.ca` # dies gibt eine Schnellübersicht wo welche Variablen zu finden sind:
einfach durch anhängen an `dune.ca` von

```
$cw    30     numeric column weights
$lw    20     numeric row weights
$eig   19     numeric eigen values
$tab       20   30   modified array
$li        20   3    row coordinates
$l1        20   3    row normed scores
$co        30   3    column coordinates
$c1        30   3    column normed scores
```

#kann auf die entsprechenden Variablen zugegriffen werden !
Also machen wir einen klassischen Biplot wie bereits in der PCA allerdings ohne die Pfeile (durch die Option `var.axes=0`): `biplot(dune.ca$c1[,1:2],dune.ca$l1[,1:2],var.axes=0)`
`# Zugriff auf Eigenwerte: dune.ca$eig`

`dune.ca.info` # dieser Wert wird oft als *eigenvalue as percentage of total inertia* bezeichnet

Overlays
Wir versuchen mit einem Overlay der Feuchtestufen der Aufnahmen die Achsen zu interpretieren...

```
biplot(dune.ca.scores[,1:2],dune.ca.loadings[,1:2],var.axes=0,cex=0.5,xlabs
=dune.env$Moisture)
```

Eleganter lässt sich dies mit `envfit()` aus dem Paket vegan lösen(s.o.); dies ermöglicht es nicht nur Pfeile durch das Korrelationsdiagramm zu legen sondern es rechnet auch einen Korrelationskoeffizienten aus und testet ihn ! Dieses Module funktioniert sowohl mit der Ausgabe von cca() als auch mit der Ausgabe von `decorana()`. Bei der letzten Funktion müssen jedoch die Parameter ira und iresc so gesetzt werden, dass keine DCA sondern eine CA gerechnet wird `(decorana(,ira=0,iresc=0))`. Mit der Ausgabe von `dudi.coa()` funktioniert das Modul `envfit()` nicht.

```
decorana(dune,ira=0,iresc=0)->dune.2ca
dune.fit <- envfit(dune.2ca, dune.nenv, 1000)
plot(dune.2ca,cex=0.7)
plot(dune.fit,arrow.mul=2)
```

Noch gewitzter geht das mit `ordisurf()`. Dazu braucht man aber kontinuierlich gemessene Variablen, hier haben wir nur die Dicke des A1 Horizonts...; `ordisurf()` legt die Isolinien über den Ordinationsplot:

```
library(MASS)
library(mva)
ordisurf(dune.2ca, choices=c(1, 2), dune.env$A1)
```

DCA – Entzerrte Korrespondenzanalyse – *detrended correspondance analysis* – in R

Hier gibt es das Modul `decorana()`, wir benutzen

```
decorana(dune)->dune.dca
plot(dune.dca)
```

Zum Durchführen der gängigen Methode weitere Optionen dabei siehe `help(decorana)`, darin gibt es eine Reihe interessanter Informationen zu dieser Methode! mit der Option `mk=...` lässt sich die Anzahl der Segmente variieren.

nMDS: nichtmetrisches mehrdimensionales Skalieren (*nonmetrical multidimensional scaling*) – in R

Nun betrachten wir eine neue Ordinationsmethode, die wiederum Vor- und Nachteile gegenüber den bereits besprochenen hat, dies ist nMDS nichtmetrisches mehrdimensionales Skalieren.
Darüberhinaus besprechen wir eine Ordinationsmethode, die es erlaubt, einen Umweltdatensatz hinzuzunehmen und die dargestellte „Information" auf die Information zu begrenzen, die mit diesen Umweltvariablen „erklärbar" ist, die CCA die Kanonische Korrespondenzanalyse.
Die wichtigen neuen Module aus R hierzu sind `metaMDS()` und `cca()`.

Hier gibt es die Module `metaMDS(),isoMDS(), postMDS()`, wir benutzen zunächst `metaMDS()` da es ohne genauere Vorangaben diese Methode durchführt.

```
library(vegan)
data(dune)# wir nehmen den altbekannten Dünendatensatz,
library(MASS) #dieses Paket wird intern benötigt; es stellt durch isoMDS() die
Grundfunktionen für nMDS bereit

dune.nmds <- metaMDS(dune,k=2,dist="bray",expand=FALSE) # das kann dauern !
dune.nmds

# nMDS ordiniert nur eine „Richtung" des Datensatzes gewöhnlich die Aufnahmen auf der
Basis eines Distanzmaßes (berechnet durch Vorkommen/Abundanzen der Arten) daher
stellen wir erstmal nur Aufnahmen dar:

plot(dune.nmds,main="nMDS",type="n") # den Rahmen fürs Plotten erstellen
points(dune.nmds, display = "sites", choices = c(1,2),cex=0.5,pch=17)
# Aufnahmen als Dreiecke darstellen

text(dune.nmds, display = "sites", choices = c(1,2))
# Aufnahmen beschriften

# durch die bekannten Overlays lassen sich die Achsen interpretieren:
data(dune.env)
plot(dune.nmds,main="nMDS",type="n")
points(dune.nmds, display = "sites", choices =
c(1,2),cex=as.numeric(dune.env$Moisture),pch=17)

# Da die Positionen der Arten nicht direkt mitbestimmt werden kann man dies durch
Overlays machen:
plot(dune.nmds,main="nMDS",type="n")
points(dune.nmds, display = "sites", choices = c(1,2),pch=17,cex=0.5)
points(dune.nmds, display = "sites", choices =
c(1,2),cex=as.numeric(dune$Leoaut),col="red")
points(dune.nmds, display = "sites", choices =
c(1,2),cex=as.numeric(dune$Agrsto),col="green")
points(dune.nmds, display = "sites", choices =
c(1,2),cex=as.numeric(dune$Airpra),col="orange")
points(dune.nmds, display = "sites", choices =
c(1,2),cex=as.numeric(dune$Elepal),col="blue")
```

In Modul `metaMDS()` ist eine Methode implementiert die für die Arten eine Position ausrechnet ausgehend von dem Ergebnis der nMDS also den Aufnahmekoordinaten (*scores*); diese ist ähnelt der CA, es sind die sogenannten *weighted average(WA) scores* damit lassen sich jetzt auch die Arten platzieren:

```
text(dune.nmds, display = "species", choices = c(1,2))
```

Jetzt wird deutlich, dass einige Arten im Zentrum ihrer Vorkommen andere aber aus dem plot herauskatapultiert werden ! Grundsätzlich ist dieses Verfahren ähnlich dem in der CA also auch so zu interpretieren; um die Arten innerhalb ihrer Vorkommen darzustellen müssen wir die nMDS mit der Option `expand=FALSE` wiederholen!

```
dune.nmds <- metaMDS(dune,k=2,dist="bray",expand=FALSE)
```

dann wieder die Overlays mit den Artabundanzen:

```
plot(dune.nmds,main="nMDS",type="n")
points(dune.nmds, display = "sites", choices = c(1,2),pch=17,cex=0.5)
points(dune.nmds, display = "sites", choices =
c(1,2),cex=as.numeric(dune$Leoaut),col="red")
points(dune.nmds, display = "sites", choices =
c(1,2),cex=as.numeric(dune$Agrsto),col="green")
points(dune.nmds, display = "sites", choices =
c(1,2),cex=as.numeric(dune$Airpra),col="orange")
points(dune.nmds, display = "sites", choices =
c(1,2),cex=as.numeric(dune$Elepal),col="blue")
```

aber bitte diesmal die Arten in der richtigen Farbe:
`dimnames(dune)[[2]]->species` # damit können wir die Spaltennamen der Vegetationstabelle (hier die Arten!) in einen Vektor schreiben:[14]

```
species
species->colors #wir kopieren den Vektor nochmal
#dann bauen wir uns einen Farbcode für die Arten...
colors[species!="Airpra"]<-"black"
colors[species=="Airpra"]<-"orange"
colors[species=="Elepal"]<-"blue"
colors[species=="Agrsto"]<-"green"
colors[species=="Leoaut"]<-"red"

text(dune.nmds, display = "species", choices = c(1,2), col=colors)
```

[14] Vgl. auch Fußnote 7 auf Seite 16.

Beispielhaft können wir uns eine Art auch mit einem fortgeschritteneren Overlay-Verfahren anschauen:

```
par(mfrow=c(2,1))
plot(dune.nmds,main="Leontodon autumnalis in einer nMDS",type="n")
points(dune.nmds, display = "sites", choices =
c(1,2),cex=as.numeric(dune$Leoaut),col="red")
```

Hier das neue Overlay im Vergleich, es zeichnet Isolinien um eine aus den Artabundanzen konstruierte Oberfläche:

```
ordisurf(dune.nmds, dune$Leoaut, xlab="Dim1", ylab="Dim2", main="L.
autumnalis als Oberfläche")
```

#Weitere Overlays

```
par(mfrow=c(2,1))
plot(dune.nmds,main="Feuchtestufen",type="n")

points(dune.nmds, display = "sites", choices = c(1,2),
cex=as.numeric(dune.env$Moisture),col="blue")

ordisurf(dune.nmds, as.numeric(dune.env$Moisture), xlab="Dim1",
ylab="Dim2", main="Feuchtestufen als Oberfläche",col="blue")

par(mfrow=c(1,1))
ordisurf(dune.nmds, as.numeric(dune.env$A1), xlab="Dim1", ylab="Dim2",
main="Ah Horizont Tiefe",col="blue")
```

Bedienung des Moduls `metaMDS()`:

```
metaMDS(comm, distance = "bray", k = 2, trymax = 20, autotransform =TRUE,
noshare = 0.1, expand = TRUE, trace = 1, plot = FALSE, previous.best, ...)
```

`comm` für die Aufnahme/Artenmatrix
distance hier kann man die Distanzmaße aus `vegdist()` wählen
`k =` Anzahl der Dimensionen/Achsen
`trymax =` max. Anzahl der Neustarts von einer neuen zufälligen Konfiguration

Die wichtigeren Angaben die über die nMDS gemacht werden sollten sind die verwendete Distanzmethode (`distance`); die Anzahl der Dimensionen (`k`); Achsenexpandierungen (`expand`) und Datentransformationen (`autotransform`) sollten mitberichtet werden; die Methode wie Arten in den Ordinationsplot dargestellt werden; der „Stress“ der sich in der Variable `$stress` im Ergebnisobjekte findet ist wohl eher ein Maß der Heterogenität im Datensatz so wie auch die Anzahl der Zufallsläufe (`$tries`).

CCA: kanonische Korrespondenzanalyse (*canonical correspondence analysis* oder *constrained correspondence analysis*) – in R

Zunächst müssen wir die Daten und das Paket `vegan` laden:

```
library(vegan)
data(dune)
data(dune.env)
```

#Wir brauchen wieder eine Umweltvariablentabelle mit der man rechnen kann daher müssen wir einzelne Variablen durch dummy-Variablen ersetzen und mit `cbind()` eine neue Tabelle erstellen:

```
as.numeric(dune.env$Use=="Pasture")->pasture
as.numeric(dune.env$Use=="Haypasture")->haypasture
as.numeric(dune.env$Use=="Hayfield")->hayfield
as.numeric(dune.env$Management=="BF")->BF
as.numeric(dune.env$Management=="SF")->SF
as.numeric(dune.env$Management=="HF")->HF
as.numeric(dune.env$Management=="NM")->NM
as.numeric(dune.env$Manure)->Manure
as.numeric(dune.env$Moisture)->Moisture
as.numeric(dune.env$A1)->A1
cbind(Moisture,Manure,A1,pasture,hayfield,BF,SF,HF)->dune.nenv
```

Nun zur eigentlichen CCA:

```
dune.cca <- cca(dune ~ dune.nenv)
plot(dune.cca) #dies ist nur ein erster "Rohplot"
```

Methodisch ist es sehr ungünstig einfach alle (Umwelt-) Variablen die wir haben der CCA zu übergeben, viel besser ist man beschränkt sich auf

A) die Variablen die uns interessieren
B) wenige unkorrelierte Variablen die einen großen Teil der Information (etwa in einer PCA) der Umweltvariablen abbilden oder stark mit den floristischen Unterschieden korrelieren (z.B. eine Variable mit hohem r^2 aus einer Regression der Variablen mit z.B. `envfit()` über die *scores* der 1./2. Achse einer CA oder DCA)

Uns waren Feuchtigkeit und Bodenmächtigkeit als mögliche Einflussfaktoren aufgefallen. Was passiert, wenn wir einfach die Sichtweise mittels CCA auf diese beiden Faktoren „beschränken" ?:

```
dune.2cca<-cca ( dune ~dune.nenv[,c(1,3)])
```

oder

```
dune.2cca <- cca(dune~A1+Moisture,dune.nenv)
```

[Falls dune.nenv kein Dataframe sondern eine Matrix ist, muss es erst umgewandelt werden

```
      dune.nenv <- data.frame(dune.nenv)]
plot(dune.2cca)
```

Und was für Arten hängen eigentlich mit der Nutzung und dem Management in unserem Datensatz zusammen?

```
dune.3cca<-cca ( dune ~dune.nenv[,4:8])
plot(dune.3cca)
```

`summary(dune.2cca)` # Gibt nun Hinweise über die durch den Umweltvariablen erklärte Information

```
Total         2.1153
Constrained   0.7968
```

```
Unconstrained 1.3184
anova.cca(dune.3cca) # Hiermit kann auch getestet werden ob diese Variablen
```
signifikant die floristische Information erklären!

Ansatz zum schrittweisen Test des Effekts der Hinzunahme von Variablen.

```
data(varespec)    # Daten laden
data(varechem)
# CCA mit ausgewählten Variablen
vare.cca <- cca(varespec ~ Al + P + K, varechem)
# Test der Güte des Modells
anova(vare.cca)
# Test der Hinzunahme der Variable N zum vorherigen Modell:
      anova(cca(varespec ~ N + Condition(Al + P + K), varechem), step=40)
```

Ein anderer Weg, um herauszufinden, welche Kombination von Umweltvariablen aus statistischen Gründen für die Einbindung in eine CCA geeignet ist, wird mit der Funktion `bioenv()` angeboten.

```
bioenv(dune,dune.nenv)
```
oder
```
bioenv(dune~dune.env$A1+Moisture+Manure)
```

Sie sollten berücksichtigen, dass statistische und nicht ökologische Kriterien zu dieser Variablenauswahl führen. Es mag sein, dass eine andere Auswahl weniger gute Regressionskoeffizienten aufweist, aber dafür besser ökologisch erklärt werden kann.

Eine Funktion, die die Güte der Anpassung der Objekte und Eigenschaften in Ordinationen angibt, ist `goodness()`.

```
goodness(cca(dune),statistic="expl")[,2]
goodness(cca(dune),statistic="expl")[,2]>0.5
auswahl <- goodness(cca(dune),statistic="expl")[,2]>0.5
source("f.plot.ord.R")
plot.ord(cca(dune),sel=auswahl)
```

Einen entsprechenden Ansatz (*constrained ordination*) gibt es auch für die Hauptkomponentenanalyse. Die Ordination heißt dann RDA (Redundanzanalyse). Es gelten die Annahmen für die PCA und die genannten Regeln für die CCA
RDA ist eine alternative Methode zur CCA, die ein anderes implizites Distanzmaß verwendet als die CCA daher oft für „kürzere“ Gradienten verwendet wurde:

```
data(dune)
data(dune.env)
dune.rda <- rda(dune ~ dune.nenv)
# auch hier ist zu überlegen, welche Umweltvariablen verwendet werden sollen.
plot(dune.rda)
```

Partielle Ordination

Bei der partiellen Ordination wird der Einfluss einer oder mehrerer Umweltvariablen in einem ersten Schritt aus dem Datensatz entfernt. Die verbleibende Variabilität wird in einer *constrained ordination* auf den Einfluss weiterer Umweltvariablen zurückgeführt.

```
library(vegan)
data(varespec)
data(varechem)

# Formulieren von Interaktionstermen
vare.cca <- cca(varespec ~ Al + P*(K + Baresoil), data=varechem)
vare.cca
plot(vare.cca)

# CCA mit CA als Umweltfaktor
cca(varespec ~ Ca, varechem)
# Herausnehmen des Einflusses des pH-Werts
cca(varespec ~ Ca + Condition(pH), varechem)
```

Formulierung: in der Formel des CCA-Aufrufs wird die Variable, deren Anteil an der Variabilität herausgenommen werden soll, mit `+Condition(Variable)` angegeben.

Dieser Ansatz wird auch bei der Varianzaufteilung verwendet.

Literaturhinweise

Lehrbücher

Backhaus, K. 2003. *Multivariate Analysemethoden*. Springer, Berlin ; Heidelberg.

Crawley, M.J. 2005. *Statistics, an introduction using R.* Wiley, Chichester.

Glavac, V. 1996. *Vegetationsökologie*. ¬G.¬ Fischer, Jena ; Stuttgart ; Lübeck ; Ulm.

Jongman, R. H. G. 1987. *Data analysis in community and landscape ecology*. Pudoc, Wageningen.

Kent, M. & Coker, P. 1992. *Vegetation description and analysis*. CRC Press, Boca Raton.

Legendre, P. & Legendre, L. 1998. *Numerical ecology*. Elsevier, Amsterdam ; Oxford.

Leps, J. & Smilauer, P. 2003. *Multivariate analysis of ecological data using CANOCO*. Cambridge University Press, Cambridge.

Leyer, I. & Wesche, K. 2007. *Multivariate Statistik in der Ökologie*. Springer, Berlin, Heidelberg.

McCune, B., Grace, J. B. & Urban, D. L. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, Or.

Internetadressen

The ordination web page (Michael Palmer)	http://www.okstate.edu/artsci/botany/ordinate/
Vegan: R functions for vegetation ecologist	http://cc.oulu.fi/~jarioksa/softhelp/vegan.html
The Comprehensive R Archive Network	http://cran.r-project.org/
Skript von Jari Oksanen	http://cc.oulu.fi/~jarioksa/opetus/metodi/
Splus and R for Ecologists	http://labdsv.nr.usu.edu/splus_R/splus.html
Einführung in R	http://www.biostat.uib.no/courses/autumn2003/bio302/lectures/
Interessante Links zum Thema	http://www.homepages.ucl.ac.uk/~ucfagls/ncourse/links.htm
Skript von Andreas Plank	http://www.geo.fu-berlin.de/geol/fachrichtungen/pal/mitarbeiter/plank/formeln_in_R.pdf
Statistikskript von Dormann & Kühn	http://www.ufz.de/data/Dormann2004Statsskript16256348.pdf