1. R-Kurs.Teil 1: Grundlagen

Jörg Wicke (joerg.wicke[at]praehist.uni-halle.de)

1.1. Einführung

1.1.1. Benutzeroberfläche

- Statusmeldung
- Menüführung

1.1.2. Befehlseingabe

- Taschenrechnerfunktionen
- Befehlsstruktur: ***Funktionsname(Objekt, Argument)*** *;* Argumente sind Steuerungsangaben für die Funktion. Runde Klammern umschließen den Gegenstand der Funktion
- Klammersetzung: Verschachtelung möglich, auch zeilenübergreifende Eingabe (Prompt wird dann zu „+“ und erwartet mindestens eine schließende Klammer)
- „Zeilengedächtnis“ im Arbeitsspeicher – Pfeiltasten ⇑ und ⇓
- Kommentarzeile: eine Eingabe mit dem Zeichen # am Zeilenanfang wird nicht ausgeführt
- Semikolon kann als Trennzeichen mehrerer Funktionen in einer Zeile gesetzt werden
- häufige Fehlermeldungen und Ursachen:
  - falscher Variablentyp in der Eingabe - Variable umwandeln s. 1.2.1
  - Ergebnis „NA“ – Eingabe enthält evtl. leere Felder, s. 1.2.4
  - „Objekt x nicht gefunden“ – auf Groß-/Kleinschreibung achten
  - Kommasetzung bei Aufzählungen (meist Komma vergessen)
  - bei verschachtelten Eingaben Klammerebene nicht geschlossen

1.1.3. Skriptdateien, Arbeitsverzeichnis

Für das Speichern von Befehlsabfolgen sollten Skriptdateien angelegt werden. Das sind Textdateien mit Programmcode, die unmittelbar in R ausgeführt werden können. Sie helfen beim Strukturieren und Optimieren des Programmablaufs und sparen natürlich Tippzeit. Vor allem aber dokumentieren sie die verwendeten Methoden und sollten deshalb unbedingt auskommentiert werden. Umfangreichere Skriptdateien, deren Ergebnisse in Publikationen vorgelegt werden, lassen sich etwa als Anhang mit veröffentlichen.

- Dateierweiterung einer Skriptdatei lautet *.R
- Ändern des Arbeitsverzeichnisses im Menü oder mit `setwd(“xverzeichnispfad“)`
- Einlesen von Skriptdateien in den Editor im Menu */Datei/Öffne Skript...* Komfortabler und übersichtlicher sind jedoch Editoren von Drittanbietern (s. 1.7.1)
- Ausführen von Skriptdateien im Menu */Datei/Lese R-Code* ein...

1.1.4. Speichern einer Arbeitssitzung (History und Workspace)

Weit weniger Komfort als Skriptdateien bietet die Möglichkeit, die eingegebenen Befehle in chronologischer Reihe, quasi tagebuchartig, in einer Verlaufsdatei (im Textformat) zu speichern. Diese Dateien können direkt in R eingelesen werden oder als Grundlage für eine Skriptdatei dienen.

- via Menü */Datei/Speichere History…* bzw. */Lade History…*
- via Befehlszeile mit den Eingaben:

```
savehistory(file = "xdatei.Rhistory")
loadhistory(file = "xdatei.Rhistory")
```

Die gesamte Arbeitsumgebung, d. h. alle im Arbeitsspeicher von R vorhandenen Objekte, lassen sich als ein Speicherabbild (ein sog. Workspace) speichern. Das bietet sich u. a. an, wenn die Arbeit noch in der Entwicklungsphase steht und kurzzeitig unterbrochen wird. Ausgewählte strukturierte Daten (Tabellen usw.) können auch anderweitig gespeichert/gelesen werden (s. 1.2.2. und 1.2.3.)

- via Menü */Datei/Sichere Workspace…* bzw. */Lade Workspace…*
- via Befehlszeile:

```
save.image(file=“xdatei.RData“, ascii=TRUE)
```

1.1.5. R-internes Hilfesystem

- Menü */Hilfe* entweder */HTML-Hilfe* oder */Handbücher(PDF)*
- Informationen zu bestimmten Funktionen mit der Funktion `help("x")` oder `?x`
- Volltextsuche in den installierten Paketen mit `help.search("x")` oder `??x`
  Jeder R-Hilfeeintrag gibt für eine Funktion folgende Informationen aus: Beschreibung, Gebrauch, Argumentsetzung (wichtig!), Details, Aufbau des Ergebnisobjektes, Verwandte Funktionen und Beispielcodes zur Funktion (lehrreich!)

1.1.6. Internethilfen

- www.rseek.org
- www.rchaeology.tk

1.2. Datenmanagement

1.2.1. Datenstrukturen

- Die Erzeugung von Objekten in R erfolgt mit einem Zuweisungspfeil (Tasten „Minus“ und „Größer“) „->“: `x<-3.1415;` `0->y` Objektausgabe mit: `x; y`
- einzelne Werte werden mit „c“zu längeren Vektoren verkettet: `vec<-c(1,3,4:6,10.2,11)`
  Ausgabe von Elementen eines Objekts durch Kombination mit eckigen Indexklammern:
  `vec[2]` `vec[c(3,5,7)]` `vec[-1]` eckige Klammer = Indexklammer
- Vektoren-Arten entsprechen Datenarten in R. Drei Arten wichtig für Daten: *numeric* für metrische und intervallskalierte Merkmale, *factor* für nominale und *ordered factor* für ordinale Merkmale. Zwei weitere Arten: *character* für Texteingabe und *logic* für Auswahlen (s. u. 1.2.4. u. 1.2.5.)
  - numerische Vektoren mit Punkt (!) als Dezimalzeichen
  - einfache Reihen sind mit Doppelpunkt „:“ implizierbar: `1:5`
  - Zeichenvektor: `satz<-c("Das","sind","vier","Worte")`
  - Variablentyp wird beim Einlesen von R automatisch ermittelt bzw. „geraten“– Achtung bei gemischten Vektoren wird niedrigstes Informationsniveau gewählt (z. B. *character*)
  - die Eingabereihenfolge wird im Index gespeichert
  - Nominalvariable mit vier (einmalig auftretenden) Ausprägungen; das Argument *level* steuert die Reihenfolge der Ausprägungen
    ```
    zeit<- factor(c("Winter","Sommer","Herbst","Frühling"))
    zeit2<- factor(c("Winter","Sommer","Herbst","Frühling"), levels
    =c("Winter","Sommer","Herbst","Frühling"))
    vec<-(c("Jan","Feb","Mrz","Apr","Mai","Jun","Jul","Aug",
    "Sep","Okt","Nov","Dez")
    mon <- factor(vec, levels=vec)
    ```
  - Bei Ordinalvariable ist die bei *levels* angegebene Reihenfolge ausschlaggebend, andernfalls würde alphanumerisch sortiert; nun zusätzlich Parameter *ordered=TRUE*
    ```
    zeit<- factor(zeit, levels=c("Frühling", "Sommer","Herbst",
    "Winter"),ordered=TRUE)
    ```
    Ausprägungen stehen nun in der gewünschten Ordnung und (!) sind ordinalskaliert
  - eine metrische Variable ist ein numerischer Vektor:
    `temp<- c(-5,-6,0,5.5,10,18.9,23,22,20,16,6,-2.3)`
- Eine Matrix ist ein „Zahlenfeld“ bzw. eine Tabelle mit der man rechnen kann
  `mx <- matrix(c(0,4.2,5:8), nrow=3, ncol=4)`
  Ausgabe mit 2D-Indizierung `mx[1,2];` `mx[ ,2]`
- Ein *Dataframe* ist die R-Variante einer Datentabelle mit verschiedenen Datentypen in den einzelnen Spalten und der wichtigste Objekttyp für die Speicherung von Daten:
  `jahr<-data.frame(mon, zeit= zeit[c(1,1,1,2,2,2,3,3,3,4,4,4)],temp)`
  - Auswahl/Abfrage aus *Dataframes* über Indexklammern und mit $-Zeichen verknüpftem Variablennamen *jahr[zeile,spalte]* oder *jahr$mon* oder *jahr$mon[zeile]; jahr["mon"]* oder *jahr[,c("temp","mon")]* usw.
  - zu einem Dataframe zusammengefasste Vektoren müssen die gleiche Länge haben
  - Verknüpfung der Zeilen erfolgt über Indices, d.h. im Normalfall wie Eingabereihenfolge
  - R versucht, die Variablentypen zu erkennen und umzuwandeln
- Eine Liste ist eine Art Datenbündel und kann alle möglichen Objekttypen beinhalten:
  `bundel<-list(kalender=jahr, zahlen=vec)`

Auswahl/Abfrage ähnlich Dataframe: *bundel$kalender$mon[1]* oder *bundel[[1]][1,1]*

- Bei Erfüllung der Strukturvoraussetzungen ist Umwandlung z.B. Matrix in Dataframe möglich:
  `mx_datfr <- as.data.frame(mx)`
  Vergleiche bei Nichterfüllung `(as.matrix(jahr) -> jahr_mx)`
- Information über Aufbau und Typ eines R-Objektes ergibt: `str(mx_datfr)`

1.2.2. Datenimport

- eine Messreihe als Vektor per Hand eingeben `x<-scan(dec=",")`
- eine Datentabelle als Dataframe aus dem Zwischenspeicher (!) einlesen:
  ```
  x <- read.delim2("clipboard", dec=",", header=TRUE, row.names="idx")
  x2 <- read.table("clipboard", dec=",", header=TRUE, row.names="idx")
  ```
  „idx“ ist der Name (!) der Spalte mit den zukünftigen Zeilennamen
- eine Datei im .txt-Format als Dataframe einlesen:
  `x<-read.table(file="dateiname.txt",header=TRUE,dec=",",sep="\t")`
  Zeilenüberschriften (*header*) sind nur für Wertespalten nötig, nicht für die Spalte mit dem Zeilennamen; d. h. die erste Zeile mit den Spaltennamen besitzt einen Eintrag weniger als die übrigen Zeilen
- Wird die Datei nicht gefunden ist eventuell das Arbeitsverzeichnis falsch gesetzt und mit *setwd (“C:\Verzeichnisname\“)* oder */Datei/Verzeichnis wechseln...* zu ändern
- Interaktives Einlesen als Dataframe:
  `x<-read.table(file.choose(),header=TRUE,dec=",",sep="\t", row.names=1)`
- nach erfolgreichem Import prüft man, ob korrekt transkribiert (Sonderzeichen, Umlaute, Dezimal-zeichen) wurde und ob Spaltennamen nicht verrutscht sind: `str(x)`
- Import des bewährten dBASE-Formats ist mit Paket *foreign* möglich (installieren/laden, dann):
  `read.dbf(file.choose())`

1.2.3. Export einer Datentabelle (Dataframe)

- in den Zwischenspeicher: `write.table(x,file="clipboard",sep="\t",dec=",")`
- in eine Textdatei: `write.table(x,file="datei.txt",sep="\t",dec=",")`

1.2.4. Datenaufbereitung

- Nicht beobachtbare Werte kennzeichnet R mit dem Eintrag *NA* (=nicht aufgenommen). Bei manueller Eingabe wird NA (ohne ““) erkannt. Beim Import werden leere Felder in NA umgewandelt.
- Die Anzahl von NA´s prüft man mit: `sum(is.na(x))`
- viele Funktionen können nicht mit NA´s rechnen, bieten aber die Möglichkeit, diese Werte vor der Berechnung auszuschließen, indem der Parameter *na.rm=TRUE* gesetzt wird.
- Quantitative in qualitative Daten umwandeln:
  ```
  jahr$temp_qual[jahr$temp<=10]="kalt"   # die neue Spalte temp_qual
  jahr$temp_qual[jahr$temp>10]="warm"   # wird nach Bedingung aufgefüllt
  jahr$temp_qual<-factor(jahr$temp_qual)
  # R erkennt die Werte nun als Nominalvariable
  ```

- Kreuztabellen: `jahr_tab <- table(jahr$zeit, jahr$temp_qual)`
  Man beachte, dass selbst eine Tabelle in R ist als Objekt zugewiesen werden kann.
- Sortieren: *order(jahr$temp)* liefert Indices, im Beispiel auf- oder absteigende Zeilennamen
  ```
  (jahrord <- order(jahr$temp))
  # die Zeilen der Tabelle 'jahr' in der Abfolge der Temperatur
  (jahrwarm <- jahr$temp>10)
  # Zeilen der Tabelle 'jahr' mit Temperatur> 10 C°
  ```

1.2.5. Abfragen, Auswahl und Filtern

Unterobjekte (z. B. Zeilen) werden über Indizes im Überobjekt (z. B. Dataframe) verortet. Man kann die Unterobjekte entweder direkt über ihre Indices ansprechen (s. 1.2.1) oder über gewünschte Eigenschaften ihrer Inhalte filtern. Die Indices sind in der Regel „fest“, d.h. sie bleiben auch nach Filtern, umsortieren u. ä. Änderungen bestehen. Sie sind mit Primärschlüsseln in Datenbanken vergleichbar.

- Eine Abfrage liefert für das abgefragte Objekt entweder den Wahrheitswert TRUE an jeder Position des Vektors, der der Frage entspricht (Index und Wahrheitswert)
  `jahr$mon==“Jan“`

- oder nur die Indices:
  which(jahr$mon==“Jan“); grep("J",jahr$mon)
  Funktion *grep("Nadel", Heuhauf)* sucht nach der Buchstabenfolge „*Nadel*“ im Objekt *Heuhauf*
- auch die Indexwerte der gewünschten Teilmenge können zum Filtern verwendet werden:
```
mon_J <- grep("J", jahr$mon)  # wir holen die Indices und...
jahr[mon_J,]                  # rufen die zugehörigen Inhalte
```
- Filtern eines Dataframe ist u.a. mit *subset(Objekt,Bedingung)* möglich:
  subset(jahr, temp>10); subset(jahr, zeit=="Herbst" & temp>10)

1.3. Kontrollstrukturen und Ablaufoptimierung

1.3.1. Bedingungen

- an den Wert einer Variablen geknüpfter Programmablauf wird etwa mit *if* kontrolliert :
```
if (jahr$zeit[6]=="Sommer") {paste(jahr$mon[6],“im Sommer“}
else {paste(jahr$mon[6], “nicht im Sommer“}
```
  (Kein Zeilenumbruch vor else!)
- weitere Kontrollstrukturen siehe z. B. *?"if"*

1.3.2. Wiederholungen

- soll Zeile für Zeile, Spalte für Spalte oder Feld für Feld abgearbeitet werden, kann mit Wiederholungsoperatoren gearbeitet werden. Alle bearbeiteten oder abgefragten Variablen müssen vorher definiert werden:
```
summe <- 0
for (k=1 in length(jahr$temp)  {summe <- summe + jahr$temp[k]}
```

1.3.3. Funktionen

Häufig verwendete Algorithmen kann man als eigene Funktionen programmieren! Sie können dann wie gewohnt unter *Funktionsname(Objekt, Argumente)* aufgerufen werden. Auch hier steht der Inhalt in geschweiften Klammern.

```
meine_funktion<-function(a,b)
{# meine_funktion=Name; a u. b sind erwartete Argumente
 wert.1 <- a+b
 wert.2 <- a/b
 return(list(wert.1, wert.2))
# die Ausgabe, d.h. das Ergebnisobjekt wird zurückgegeben
}
```

- innerhalb der Funktion generierte Variablen gelten nur hier und sind außerhalb nicht verfügbar,
- die Rückgabe der Ergebnisobjekte wirkt umständlich, hilft aber die Daten zu strukturieren,
- Einrücken hält den Quellcode und die Klammersetzung übersichtlich,
- Dokumentation durch Kommentarzeilen (mit Zeichen #) mit hilft bei späterer Weiterverwendung.

1.4. Einfache Statistik

Einfache Übersichtswerte bietet *summary(x)*, wobei die Funktion je nach Datentyp die passenden zusammenfassenden Informationen liefert. Man vergleiche

summary(jahr); summary(jahr$temp); summary(jahr$mon)

1.4.1. Lage- und Streumaße, auch für mehrere Zeilen oder Spalten

```
mean(jahr$temp)        # arithmetisches Mittel
median(jahr$temp)      # unschwer zu erkennen, oder?
sd(jahr$temp)          # Standardabweichung
```

Bei mehreren Spalten in einem Dataframe kann man mit der Funktion *apply* vektorweise arbeiten. Das Argument *MARGIN* besagt, ob über Zeilen (1) oder Spalten (2) gearbeitet wird, und *FUN* gibt die anzuwendende Funktion an (mean, median, sd, min, max, sum …)

apply(mbk.daten[,5:39],MARGIN=2,FUN=sum)

1.4.2. Häufigkeiten

- für kategoriale Daten: `table(jahr$temp_qual)`
- für metrische Daten: `sum(jahr$temp>10 & jahr$temp<20)`
- komfortabler (s. 1.5.2.) - inklusive Berechnung sinnvoller Klassengrenzen – ist: `hist(x)`

1.4.3. Hypothesentests

Wahrscheinlich ist jeder Test, den es momentan gibt, in irgendeinem einem R-Paket vorhanden.

- Chi-Quadrat-Test (ist Vektor gleichverteilt?): `chisq.test(c(20,10,25))`
- Wilcoxon-Rangsummen-Test: `wilcox.test(x=c(1:8),y=(3:10))`

1.5. Einfache Graphiken

Beim Erstellen von Graphiken wird eine „Graphics Device“ (Grafik-Fenster) geöffnet oder verändert. Ist dieses Fenster aktiv, ändert sich die Menüleiste im Hauptfenster. Äußerst praktisch ist die Möglichkeit den Verlauf eines solchen Fensters zu speichern. Dafür muss das Grafikfenster aktiv sein und dann über das Menu */History/Aufzeichnen* Blättern ist mit den Tasten *Bild⇑* und *Bild⇓* möglich.

Die intelligente Programmierung der Allround-Funktion *plot()* bietet in vielen Fällen an das Datenformat angepasste, brauchbare Diagramme. Mit ein paar Argumenten, die auch in vielen anderen Graphikarten verwendet werden können, lässt sich die Darstellung anpassen:

```
plot(jahr$temp,main="Temperatur im Jahreslauf",xlab="Monate",ylab="°C")
```

1.5.1. Punktwolken

```
plot(x,y, pch=3, cex=0.5) # pch=Punktsymbol-Art; cex=Größe der Punkte
```

- Beschriftung ist mit Funktion *text(x-koordinate,y-Koordinate, labels=Beschriftung)* möglich.

1.5.2. Histogramme

```
hist(jahr$temp); hist(jahr$temp, breaks=c(-20,0,20,40))
```

- Klassenanzahl wird automatisch berechnet oder die Werte bei *breaks*= angegeben
- Das Ergebnisobjekt enthält interessante Angaben, wie z.B. die Häufigkeiten:
  `hist(jahr$temp)$counts`
  - den genauen Aufbau des Objekts zeigt das bekannte `str(hist(jahr$temp))`

1.5.3. Boxplots

Da hier verschieden skalierte Daten miteinander dargestellt werden, kann die Datenabfrage über eine „Formel“ (*formula*) erfolgen. In Formeln steht die Tilde „~“ als Trennzeichen der beiden Formelhälften.

```
boxplot(temp~zeit,data=jahr) # Ratiovariable ~ Nominalvariable
```

1.5.4. Speichern von Graphiken

In R werden verschiedene Wege zur Speicherung selbst erstellter Abbildungen angeboten. Auch die automatische, d.h. im Quelltext erzwungene, Speicherung ist unter Anwendung vieler optionaler Argumente zum Dateityp, zur Auflösung, Größe usw. möglich. Gemessen am Aufwand, der Nutzerfreundlichkeit und den Möglichkeiten zur Nachbearbeitung in Graphikprogrammen ist für Anfänger in R folgende Methode noch effektiver:

- Graphikfenster anklicken, dann Menu */Datei/Speichern als...* z.B. als .pdf-Datei (Vektorgrafik)
- zur code-generierten Speicherung von Graphiken siehe etwa die Hilfeseite `?bmp`

1.6. Pakete

Pakete enthalten mehrere, thematisch meist zusammenhängende, Funktionen, zugehörige Hilfedateien und Beispieldatensätze. Für jedes Paket gibt es ein Befehlsverzeichnis in .pdf-Format, das sog. Manual. Zu einigen werden auch ausführliche Dokumentationen wie etwa Bücher angeboten (z. B. Greenacre 2007). Die Dateien sind unter Laufwerk/R-Verzeichnis/library/Paketname zu finden. Pakete müssen separat in R zuerst (einmalig) installiert und dann, vor der Verwendung, geladen werden. Nach der Installation steht die interne Hilfe zur Verfügung (s. 1.6.3.), aber erst nach dem Laden lassen sich die Befehle eines Paketes verwenden.

1.6.1. Suchen

Ein R-Paket für eine bestimmte Methode kann man auf zweierlei Arten suchen. Auf der Hilfe-Seite www.rseek.org kann man einen Begriff oder Methodennamen eingeben, wie etwa *point pattern analysis* und erhält zahlreiche Vorschläge. Alternativ bietet CRAN im sog. Aufgaben-Überblick ('Task Views') nach Disziplinen und/oder Methodenfeldern geordnete Verzeichnisse von Funktionen und Paketen an (http://cran.r-project.org/web/views/). Wählt man etwa *Multivariate*, erhält man einen Überblick (in diesem Task View), welche Pakete welche Methoden beherrschen. Dieser Überblick ist z.B. bei 'Multivariate' nicht ganz vollständig. So fehlt beispielsweise das Paket *anacor* (für CA und CCA).
Eine vollständige Liste aller Erweiterungs-Pakete (n=3496, Stand 26.1.2012) steht bereit unter http://cran.r-project.org/web/packages/available_packages_by_name.html

1.6.2. Installieren

Ein Paket muss in R installiert werden, damit R dessen Funktionen kennt. Wenn der PC online ist, kann dies direkt mit einem Befehl erfolgen

```
install.packages("ca", dependencies="Depends")
```

Ansonsten muss die (Windows-)Zip-Datei heruntergeladen werden. Z. B. Paket *ca* unter http://cran.r-project.org/web/packages/ca/index.html und dort bei *Downloads* das *Windows binary*. Man beachte, dass manche Pakete andere Pakete benötigen – angezeigt als zweiter Listenpunkt (*Depends*). Das separate installieren kann durch das Argument *dependencies="Depends"* automatisiert werden.
Einzeln separat heruntergeladene Pakete kann man im Menü des R-GUI installieren */Pakete/Installiere Paket(e) aus lokalen Zip-Dateien...* Die installierten Pakete zeigt

```
installed.packages()
```

1.6.3. Laden

Um mit einem Paket in R zu arbeiten, muss das Paket geladen werden, nachdem es installiert wurde. Das kann über das Menü */Pakete/Lade Paket...* erfolgen. Alternativ gibt es zwei Befehle dafür:

```
require(ca)       # OHNE Anführungsstriche
library(ca)
```

Welche Pakete geladen sind zeigt: `search()`

1.6.4. Zitation

R-Pakete sind wissenschaftliche Arbeiten, die zu zitieren sind. Wie ein Paket zitiert werden soll, zeigt:

```
citation("ca")    # Bildschirmausgabe
```

- Ausgabe in eine Datei:

```
sink("Paket_ca_Zitat.txt")    # Umleitung in die Datei gewählten Namens
citation("ca")                # Es erfolgt keine Bildschirmausgabe
sink()            # Beenden der Umleitung; Zitat steht in .txt-Datei
```

1.7. Editoren und Frontends

1.7.1. Editoren

- Textpad, über Zusatzdatei wird R-spezifische Syntax eingefärbt

1.7.2. Frontends

Neben reinen Editoren, die im luxuriösesten Fall einen Knopf zur Skript-Übergabe an die R-Konsole verfügen, gibt es mittlerweile komfortable Lösungen, die via Menüführung vorgefertigte Datenmanagement- und Analysemittel anbieten und, wenn gewünscht, fast nur mit der Maus bedient werden können. Sie sind mit kommerziellen Programmen wie SPSS vergleichbar.

- R-Commander. Als R-Paket *Rcmdr* installierbar
- Ein flammneues sehr handliches Frontend ist *R-Studio*, erhältlich unter http://rstudio.org/
- Rkward: http://sourceforge.net/apps/mediawiki/rkward/
- TINN
- R Productivity Environment (REvolution)

1.9. Allgemeine Literatur zu R

Alternativ finden Sie viele kostenlose Anleitungen im Netz. Eine Sammlung deutschsprachiger Anleitungen finden Sie in unserem Blog www.rchaeology.tk (Mitblogger sind stets Willkommen!) im

Menu *R-Einführungen.* Oder googlen Sie unter *„Alles was ein R braucht“* (mit Anführungsstrichen). Eine umfassende und dabei zielführende englischsprachige Online-Einführung ist z.B. „Quick-R“. In eckigen Klammern stehen unsere höchst subjektiven Einschätzungen zu den Werken.

1.9.1. Allgemeine einführende Literatur zu R

D. Adler, R in a Nutshell (Sebastopol 2010). [sehr kompakt]
M. Crawley, The R Book (Chichester 2007). [sehr umfangreich, gut für Anfänger und Fortgeschrittene, allerdings werden Methoden unterschiedlich detailliert erklärt]

D. Wollschläger, Grundlagen der Datenanalyse mit R. Eine anwendungsorientierte Einführung (Berlin 2010). [ein Mittelweg]
P. Teetor, The R-Cookbook (Sebastopol 2011). [Sammlung von Code-Schnippseln]

1.9.2. Programmieren, Daten, Grafiken etc.

U. Ligges, Programmieren mit R (Berlin 2008, 3. Aufl.) [beste dt.-sprachige Programmiereinführung].
H. Mittal, R Graphs Cookbook (Birmingham 2011). [aktuelle, einfach aufgebaute Grafikanleitung]
Ph. Spector, Data Manipulation with R (New York 2008). [DER Wegweiser zur Datenaufbereitung]

1.9.3. Grundlegende Statistik mit R

Chr. Duller, Einführung in die nichtparametrische Statistik mit SAS und R. Ein anwendungsorientiertes Lehr- und Arbeitsbuch (Heidelberg 2008). [wichtige Tests für nicht-normalverteilte Daten]
J. Groß, Grundlegende Statistik mit R. Eine anwendungsorientierte Einführung in die Verwendung der Statistik Software R (Wiesbaden 2010). [Einführung in einfache Statistik mit R]
L. Sachs/J. Hedderich, Angewandte Statistik. Methodensammlung mit R (Berlin 2006, 12. Aufl.). [Das R Pendant zum guten alten SACHS von anno 1974, nicht lesbar aber unverzichtbar]

2. R-Kurs. Teil 2: Aktuelle Methoden mit R

Georg Roth/Köln (groth[at]uni-koeln.de)

2.1. Korrespondenzanalyse mit Paketen ca und vegan

2.1.1. Daten und Pakete für die Korrespondenzanalyse

- Der MBK-Datensatz wird in Form eines vorbereiteten R-Speicherabbildes eingelesen:
  ```
  load(file.choose())
  ```

- Die Pakete installieren und laden:
  ```
  library(vegan)    # Paket 'vegan' laden (LADEN != INSTALLIEREN)
  library(ca)       # Paket 'ca' laden
  ```

2.1.2. Korrespondenzanalyse mit Paket 'ca'

Ein detailliertes Skript zur CA mit Paket *ca* steht auf www.rchaeology.tk bereit.

- Durchführen der CA:
  ```
  ca(abu) -> cabu; cabu          # Ergebnisobjekt erzeugen und anzeigen
  str(cabu)                      # Aufbau des Objekts
  summary(cabu) -> cabusum       # Zusammenfassung mit Kennwerten

  # siehe rchaeology.tk:  1) Eigenwerte,  2) Sparten(Zeilen und
  # Spalten)-Kennwerte: Masse, Qualität, Inertia, Prinzipalkoordinaten,
  # Korrelation (Übereinstimmung Sparteninertia mit Achse), Kontribution
  # (Beitrag Sparteninertia zu Achseninertia)
  ```

- Ein Ausdruck mit Zeilen (Prinzipalkoordinaten) im Raum der Spalten und mit mehreren Informationen aus der Zusammenfassung; siehe auch die Hilfe bei `?plot.ca`
  ```
  plot(cabu, map="rowprincipal", what = c("all", "all"), mass =
  c(TRUE, FALSE), contrib = c("relative", "relative"), col = c(4, 2),
  pch = c(16, 1, 17, 24), labels = c(0, 2))
  # je dicker die Punkte desto wichtiger für das CA-Ergebnis
  ```

- Das Eigenwert-Diagramm (Screeplot) zeigt wie die Inertia (Streuung) auf die Achsen verteilt ist.

```
barplot(cabu$sv^2, ylim=c(0,1), names.arg=paste(1:length(cabu$sv),
". ", sep=""), cex.names=.5, names.srt=90, las=1,xlab="Achse",
ylab="Inertia als Eigenwert", border=1, col=gray(.8) )
# Die quadrierten Singulärwerte sind die Eigenwerte
```

- Das Exportieren der Punktkoordinaten wird im Rchaeology-CA-Artikel erläutert.

2.1.3. Korrespondenzanalyse mit Paket 'vegan'

Alternativ lässt sich eine CA auch im Paket *vegan* rechnen. Dieses Paket muss als die Königin der Multivariatstatistik-Pakete gelten. Eine Einführung zu vielen interessanten Methoden gibt der Autor Jari Oksanen in einem Online-pdf (http://cc.oulu.fi/~jarioksa/opetus/metodi/vegantutor.pdf).

- Die CA mit *vegan* ist ebenfalls nur ein Befehl:

```
cca(abu)->cavegabu; cavegabu
# Erzeugen und Anzeigen des Ergebnisobjekts
plot(cavegabu, scaling=1, display="sites")
# Biplot nur Zeilen in Prinzipal-Skalierung
```

- Aber *vegan* bietet mehr, etwa das Hinein-Interpolieren einer Kovariablen in den Biplot. So lassen sich Bezüge zwischen CA-Ergebnis und Kovariablen intuitiv deuten. Eine CCA mit dieser Variablen als Kovariable ist wegen der Testmöglichkeit aber vorzuziehen (! s. u.).

```
ordisurf(cavegabu, mbk.daten[,41], add = TRUE, col = 4)
# Isolinien zu den X-Koordinaten im Biplot
ordisurf(cavegabu, mbk.daten[,42], add = TRUE, col = 3)
# Isolinien zu den Y-Koordinaten im Biplot
```

- Das Vegan-CA-Objekt setzt aber andere Schwerpunkte; es verzichtet etwa auf Kennwerte. Für die archäologische Interpretation der reinen CA ist daher *ca* einfacher nutzbar.

```
summary(cavegabu)
```

2.2. Kanonische Korrespondenzanalyse (CCA) mit Paket vegan

2.2.1. Kanonische Korrespondenzanalyse (CCA)

Die CCA ordnet wie die CA die Zeilen nach Ähnlichkeit an, benutzt aber nur den Teil der multivariaten Information, der sich mit einer zusätzlichen Variablen (sog. Kovariable) verbinden lässt. Vereinfacht gesagt werden der CCA-Raum und die Lage der Punkte anhand der Kovariablenwerte prognostiziert – ähnlich einer Regression mit den Typ-Abundanzen als Abhängige. Da dieses Ergebnis nur einen Teil der Gesamt-Inertia abbildet, lässt sich testen, ob eine Kovariable einen signifikanten Einfluss auf die Abundanzen ausübt. Anders gesagt kann man so archäologische Hypothesen zur Erklärung der (C)CA statistisch testen, weshalb das Verfahren unbedingt als Standard-Methode gelehrt werden muss.

- CA und CCA werden in *vegan* mit demselben Befehl durchgeführt:

```
cca(abu~., data=mbk.daten[,41:42]) -> ccabu; ccabu
# durch die Tilde „~." werden alle Variablen der Datentabelle
# „data=mbk.daten" als Kovariablen benutzt; hier sind es die X-
# und die Y-Koordinate

summary(ccabu)
# Die eingeschränkte ('constrained') Inertia ist der mit der
# Kovariablen verbundene Informationsanteil der Abundanzen
```

2.2.2. Test der CCA

Ob die Kovariablen einen signifikanten (nicht-zufälligen) Anteil der multivariaten Information erklären, kann man mit Permutations-Testen bestimmen. Bei mehreren Kovariablen wird zuerst die Gesamtlösung getestet, dann jede Kovariable einzeln und dann jede Achse einzeln:

```
anova(ccabu, alpha=0.05, beta=0.01, step=1000, perm.max=999)
# testet Gesamtmodell auf 5 % Alpha- und 1 % Betafehler
```

```
anova(ccabu, alpha=0.05, beta=0.01, step=1000, perm.max=999, by= "terms")
# testet jede Kovariable einzeln: Y-Koordinate ist nicht signifikant!

anova(ccabu, alpha=0.05, beta=0.01, step=1000, perm.max=999, by= "axis")
# nur die erste Achse erklärt einen signifikanten Anteil
```

2.2.3. Triplot des erfolgreichen CCA-Modell

Der CCA-Ausdruck mit Zeilen- und Spalten-Punkten sowie Pfeilen oder Punkten für metrische bzw. nominale Kovariablen heißt Triplot, weil drei Arten von Punkten gemeinsam abgebildet werden. Die Kovariablen-Symbole verdeutlichen die Beziehungen der Kovariablen mit den CCA-Achsen.

- Nachdem man das CCA-Modell mit Testen überprüft hat, rechnet man ein neues nur mit dem/n signifikanten Kovariablen:

```
cca(abu~mbk.daten[,41]) -> ccabux; ccabux
# X-Koord. erklärt 7 % der Inertia

plot(ccabux, scaling=1, type="none", xlim=c(-3.5,1.5), ylim=c(-3, 2) )
# type="none" lässt die Punkte weg und Grenzen werden vorgegeben.
# Der Ausdruck lässt sich so detaillierter steuern...

points(ccabux, display="sites", scaling=1, pch=21, bg=rainbow(11)
[mbk.daten$mbk_stufe])
# Zeilen in Prinzipal-Skalierung mit Regenbogenfarben nach MBK-Stufen


points(ccabux, display="spec",scaling=1, pch=22, bg=8, cex=1.3)
text(ccabux, dis="spec",scaling=1, pos=3, cex=.8, font=3)
# Typen in Standard-Skalierung als graue beschriftete Rechtecke

points(ccabux, dis="bp", col=2, lwd=2)
# Kovariable X-Koord. als roter Pfeil
text(ccabux, dis="bp", "Kovariable", font=2, pos=2) # Beschriftung


legend("topleft", pch=rep(21,11), pt.bg=rainbow(11), levels(
mbk.daten$mbk_stufe), cex=.7)
# Legende für die MBK-Stufen-Farben
title(main="CCA von MBK-Gefäßtypen \n auf geographische X-Koordinate")
```

2.2.4. Partielle CCA (pCCA)

Bei einem CCA-Modell kann man auch die mit einer Kovariablen verbundene Information entfernen und eine CCA-Lösung erzeugen, die diese Information für die Lösung nicht (!) berücksichtigt, eine sog. partielle CCA (pCCA). Das Entfernen von Kovariablen-Einfluß heißt Partialisierung.

- Durch die Angabe *~1* wird eine „normale“ CA gerechnet. Die zu entfernende Kovariable wird als Argument *+Condition([Kovariable])* gesetzt. Hier wird die mit dem Nominal-Merkmal *mbk.daten$stufe* verknüpfte Information, also die „traditionell typologische Zeit“, entfernt.:

```
cca(abu~1 + Condition(mbk.daten$mbk_stufe))-> ccabustu          # pCCA;
# die Zeit – verkörpert in den typologischen MBK-Stufen – ist entfernt!

ccabustu # Etwa ein Drittel der Information (Inertia) ist noch vorhanden!

plot(ccabustu, scaling=1, type="none")# leerer Ausdruck

points(ccabustu, scaling=1, pch=21, bg=rainbow(11)[mbk.daten$mbk_stufe])
# Die Zeilen in Prinzipal-Skalierung mit Regenbogenfarben nach MBK-Stufen
```

- In einem Ausdruck kann man mit *identify([X],[Y], [Namen])* unter Angabe der Koordinaten und der „Punktnamen“ einzelne Punkte beschriften. Bei der pCCA interessiert jetzt, welche Zeilenpunkte auffällig sind:

```
scores(ccabustu, scaling=1, display="sites")[,1]->xx
# Extraktion X-Koordinaten

scores(ccabustu, scaling=1, display="sites")[,2]->yy
# Extraktion Y-Koordinaten

identify(xx,yy, rownames(mbk.daten), cex=.7)   # interaktive Beschriftung
```

2.2.5. Export eines CCA-Modells

In *vegan* werden keine Qualitätskennwerte ausgegeben, deshalb reichen hier Extraktion und Export der Koordinaten. Exportiert wird die erfolgreiche CCA aus 2.2.3. Die Kombination aus Zeilen in Prinzipal- und Spalten in Standardskalierung heißt in der Ökologie 'scaling=1'

```
scores(ccabux, scaling=1, display="sites") -> zeilkoord
# Zeilenkoordinaten in Prinzipal-Skalierung
scores(ccabux, scaling=1, display="species") -> spaltkoord
# Spaltenkoordinaten in Standard-Skalierung
```

- Der Kovariablenpfeil wird im Triplot-Ausdruck noch optimal skaliert. Dieser Wert ist aber nur über die Ergebnisse des Ausdrucks abfragbar – eine kleine Inkonsistenz, für die ich extra nachfragen musste ;-)

```
plot(ccabux,scaling=1,plot=FALSE)->ccabuxplo   #im Ausdruck genutzte Werte
attributes(scores(ccabuxplo,"biplot"))[[3]]->multi   #Pfeillängenmultiplik.
scores(ccabuxplo,"biplot")*multi->pfeilkoord   # Der Pfeil wie im Ausdruck

list(zeilen=cbind(zx=zeilkoord[,1], zy=zeilkoord[,2]),
spalten=cbind(sx=spaltkoord[,1], sy=spaltkoord[,2]),
pfeil=(cbind(px=pfeilkoord[,1], py=pfeilkoord[,2])) ) ->  koordlist
# Listenobjekt mit den Matrizen der einzelnen Koordinaten im 2D-Triplot
```

- Dieses Objekt lässt sich nun mit *sink([Dateiname. # Erweiterung])* auf ein Medium schreiben, indem die Ausgabe umgeleitet wird. Man achte auf den Pfad des Arbeitsverzeichnisses!

```
sink("CCA_auf_Xutm.txt")  # Umleitung in .txt-Datei herstellen
koordlist           # Der Objektaufruf wird in die Datei umgeleitet
sink(); getwd()
# Umleitung wird beendet und Arbeitsverzeichnispfad abgefragt
```

2.3. Punktfeldstatistik (PPP) mit Paket spatstat

Die Punktfeldstatistik ('point pattern analysis' = PPP) oder stochastische Geometrie ist ein Methodenfeld für die Verteilungseigenschaften-Analyse von 2D/3D- Punktverteilungen. Man untersucht etwa, ob und bei welchen Abständen die Punkte eher zufällig, regelhaft oder zusammengedrängt auftreten. Benötigt werden die Koordinaten, der Untersuchungsrahmen – das sog. Fenster – und eventuell Punkteigenschaften. Wenn die Punkte Eigenschaften besitzen spricht man von markierten Punktfeldern. Die Marken können metrische oder nominale Variablen sein.
In der Archäologie können etwa Fundplatz-, Fund- oder Befundverteilungen auf ihre Verteilungseigenschaften analysiert werden. Bearbeitete oder gerade erforschte Beispiele sind neolithische Bergwerksschächte, Megalithkartierungen oder Feuchtboden-Pfahlfelder. Noch gänzlich ungenutzte Zweige sind die Zeit-Raum-Analyse von Punktfeldern – und das in der Archäologie – oder Rasterkarten basierte Teste auf die Ursache von Punktverteilungsregeln!
Die Ausgangsannahmen für eine PPP-Stichprobe lauten:

1) die untersuchten Punkte stellen einen Ausschnitt aus einem – zumindest theoretisch – unendlich großen Punktverteilungsphänomen dar und es gibt keine (!) Auswirkungen der Fenstergrenze auf die Verteilung. Nur manche archäologische Verteilungen lassen sich so als Idealphänomen vorstellen. Im Zweifelsfall schneidet man einen zentralen Teil der Verteilung als Fenster heraus.
2) Die räumliche Verteilung ist stationär, d. h. die Chance eine bestimmte Konstellation zu beobachten ist unabhängig vom Betrachtungsfenster und die Punktdichte ist (im Prinzip) konstant.
3) Es gibt keinen Richtungstrend.

Aber sowohl für 2) wie für 3) gibt es Methoden um sie als Sonderfälle zu behandeln. Einen umfassenden Kurs zur PPP mit R stellt Adrian Baddeley bereit: www.csiro.au/resources/pf16h.

2.3.1. Daten und Paket

Der hier verwendete Beispieldatensatz ist ein markiertes Punktfeld mit ab-digitalisierten Punkten für die hallstattzeitlichen Fürstensitze (Stand vor 2009) und die Ha-D-Wagengräber (ohne drei räumliche Ausreißer). Der Datensatz verwendet NICHT den aktuellen Datenstand des „Fürstensitz“-SFB.

- Die Vorbereitung eines R-Punktfeldobjektes kann hier aus Zeitgründen nicht vorgestellt werden. Statt dessen wird ein vorbereitetes R-Speicherabbild geladen. Baddeley's Spatstat-Kurs-pdf gibt Anweisungen zur Erzeugung eines R-Punktfeldobjektes.

```
load(file.choose())  # interaktives Einlesen des R-Speicher-Abbildes
setwd()              # u. U. neuer Arbeitsordner ?
ls()                 # Was steht im Speicherbild ?
hazpp                # Objekt OHNE spatstat als Liste ausgegeben
```

- Paket installieren und laden:

```
install.packages("spatstat", dependencies="Depends")
# Paket 'spatstat' und weitere benötigte Pakete installieren.
library(spatstat) # Paket 'spatstat' laden (LADEN != INSTALLIEREN)
search()          # Liste geladener (!) Pakete
```

2.3.2. PPP-Struktur

- Jetzt wird das R-Punktfeldobjekt anders (!) ausgegeben und kann auch ausgedruckt werden.

```
hazpp          # JETZT wird das Objekt als Punktfeld ausgegeben
summary(hazpp)       # Eigenschaften
plot(hazpp)          # unübersichtlicher Ausdruck

plot(hazpp$window, main="Markiertes HaZ-PF")   # zuerst das Fenster alleine

points(hazpp[marks(hazpp)=="fs"], pch=21, bg=2, cex=1.4)   # „Fürstensitze“
points(hazpp[marks(hazpp)=="wg"], pch=21, col="white", bg=4, cex=1.1) #Wggr.

legend("bottomright", pch=c(21,21), pt.cex=c(1.4,1.1), pt.bg=c(2,4),
c("Fürstensitz", "Wagengrab"))           # Legende rechts unten

        rect(130000,5735000, 180000,5740000, col=8)
        # Maßstab; Argumente: x-min, y-min, x-max, y-max, Füllfarbe
        text(155000,5740000, "50 km", pos=3, cex=.8, font=3)
        # Beschriftung des Maßstabs
```

2.3.3. PPP-Struktur

- Erste Explorationen – die Dichte mit nicht (!) bandbreitenoptimiertem Schätzer (vgl. u. 2.4.4.)

```
plot(density(hazpp))             # Dichte aller Punkte
plot(density (split(hazpp)))     # Dichte nach Punktarten getrennt
plot(density(hazpp[marks(hazpp)=="wg"]))     # Wagengräberdichte
```

- Verteilungsanalyse aller Punkte mit Ripley's K als Besag's L (zur Deutung z.B. Stoyan/Stoyan 1992). Ist der Wert größer als 0 und als das Zufallsintervall liegt eine signifikante Zusammendrängung bei Abständen des Wertes r vor. Ist er kleiner, so sind regelhafte Abstände beim Abstand r vorhanden.

```
ripl <- envelope(hazpp, fun=Lest, nsim=99, nrank=1, r = seq(0,120000,
by=1000), correction = "translate")      # erzeugt das Graphen-Objekt

plot(ripl, (.-r)~r)     # Ripley's K(r) (als L(r) ist Linie waagrecht).
```

2.3.4. Analyse inhomogenes Punktfeld

Das Ausscheren des Graphen oberhalb des Zufallsintervalls weist auf Inhomogenität (unterschiedlich dicht besetzte Bereiche). Dafür wurden Varianten der Standard-Funktionen entwickelt (Suffix -inhom).

```
riplin <- envelope(hazpp, fun=Linhom, nsim=99, nrank=1, r = seq(0,120000,
by=1000), correction = "translate")      # Diesmal mit L für Punktfeld mit
inhomogener Dichte

plot(riplin, (.-r~r), legendpos="topright", xlab="Kreisradius r", main="")
      # Zeichne alle Linien minus r-Wert (!) gegen r-Wert; Legende r. o.

title(main="Ripley's L für die HaZ-Punkte \n als inhomogenes Punktfeld")
```

- Welche Abstände signifikant sind, erfährt man durch eine Auswahl:

```
as.data.frame(riplin) -> ripdat
# Maximalwertsuche fällt leichter, wenn zu 'data.frame' konvertiert

ripdat[,2:5]-ripdat$r -> ripdat[,2:5] # Werte jetzt wie Grafik;
(ripdat$obs > ripdat$hi -> riplog)    # logischer Auswahlvektor
ripdat$r[riplog]                      # Anzeige mit Indizierungsvek
(which.max( ripdat$obs) -> maxi)      # maximales L
ripdat$r[ maxi]     # Bei r = 27 km, d.h. Gruppen mit ~ 55 km Dm.
abline( v= ripdat$r[ maxi], lty=3, col=4, lwd=2)  # Linie bei Max.
```

2.3.5. Analyse inhomogenes Punktfeld

Sind die Wagengräber um die Fürstensitze herum zusammengedrängt? Dafür benutzt man eine punktfeldstatistische „Kreuzfunktion“, die die Lage der J-Punkte um die I-Punkte analysiert. Für die benutzte Funktion braucht man (gute) KDE der beiden Punktarten und dafür das Paket 'ks':

```
hpi( as.matrix(coords (split(hazpp)$fs)) )-> hf; hf  # Bandbreite für FS
(hpi( as.matrix(coords (split(hazpp)$wg))  )-> hw)   # Bandbreite für WG

dimf <-  density.ppp(split(hazpp)$fs, sigma=hf, diggle=TRUE, eps=1000)
         # Dichte-Rasterkarte der FS

dimw <- density.ppp(split(hazpp)$wg, sigma=hw, diggle=TRUE, eps=1000)
         # Dichte-Rasterkarte der WG

pcfcrossin <- envelope(hazpp, fun=pcfcross.inhom, nsim=99, nrank=1,"fs",
"wg", lambdaI = as.im(dimf), lambdaJ = as.im(dimw), r = seq(0,120000,
by=1000), kernel= "epanechnikov", correction = "translate")
```

- Das Ergebnis ist ein Graph, der erkennen lässt, ob Wagengräber bei irgendeinem Abstand r um einen Fürstensitz signifikante Abweichungen von einer Zufallsverteilung zeigen. Hier ist das Ergebnis: es gibt nur Zufallseffekte (Man plote wie bei 2.3.4.).

2.4. Bandbreitenoptimierte 3D-Kerndichteschätzung ('kernel density estimation', KDE) mit Paket ks

Eine Kerndichteschätzung (KDE) schätzt rechnerisch für beliebig kleine Ausschnitte des untersuchten Raumes die Punktdichte als Punkte (n) pro Raum. Im 2D-Fall ist der Raum eine Fläche der Größe A. Die Dichte ist also n/A. Dies funktioniert in 3D im Prinzip wie in 2D. Bei der 2D-Variante lässt sich die Funktionsweise mit einer Metapher leicht erklären: Jeder Punkt steuert den Wert 1 zu einer Dichteschätzung bei. Jetzt wird jedem Punkt quasi ein „runder Hut“ etwa in der Form einer Gaussglocke aufgesetzt. Das Volumen zwischen Hut und Fläche entspricht 1. Der Radius des Hutes steuert, wie stark der Beitrag von 1 in Fläche „verteilt“ wird. Die Höhe zwischen Hut und Fläche an einer bestimmten Koordinate ist die Dichteschätzung. Überlappen sich mehrere „Hüte“ an einer Koordinate, entspricht die Schätzung der Summe aus allen „Huthöhen“. Die Bandbreite ist ein Steuerungsparameter für die „Hutbreite“. Um sie nicht einfach willkürlich festzulegen, wurden in der Mathematik mehrere Verfahren zu ihrer, den Daten angepassten, optimierten Schätzung entwickelt. Hier wird das sog. Einsetzverfahren (plug-in) verwendet – es gibt aber auch andere.
In 3D sind es keine „Hüte“ sondern „Kugeln“ um die Punkte. Je mehr „Kugeln“ sich überlagern, desto dichter ist die Punktverteilung an einer bestimmten Stelle. Grafisch wird das in 3D mit zunehmend

undurchsichtigeren Raumbereichen umgesetzt. Man stelle sich das vor wie Regenwolken: je mehr Tröpfchen in einem Wolkenausschnitt vorhanden sind, desto dunkelgrauer wird die Wolke.

2.4.1. Pakete und Daten

- Vorbereitungen: Pakete installieren und laden

```
install.packages("ks", dependencies="Depends")
require(rgl)                  # Laden der Pakete
require(ks)
search()                      # Liste geladener Pakete
```

- Die Ausgangsdaten mit 3D-Einmessungen von Funden aus zwei Schnitten befinden sich im einzigen Objekt eines vorbereiteten Speicherabbildes.

```
load(file.choose())     # interaktives Einlesen des Speicher-Abbildes
ls()                    # Es gibt nur das Objekt 'd3dat'
str(d3dat)

   # Alternativ: Einlesen eigener ASCII-Datensatz mit Dezimalpunkt
   # read.table(file.choose(), header=TRUE, row.names=1) -> d3dat
   # str(d3dat)              # Aufbau des R-Datentabellenobjektes
```

- Die Daten bestehen aus 316 Fällen (=Funden) und 5 Spalten (=Variablen). Die erste Spalte namens 'bef' ist eine Nominalvariable (Factor), deren Ausprägungen die Schnittnamen beschreiben und nur die Möglichkeiten 'bef_1' oder 'bef_2' beinhalten. Die Spalte 'id' enthält eine Fundnummerierung, deren Zählung für jeden Schnitt neu beginnt. Die Spalten 3 bis 5 enthalten die D3-Koordinaten.

```
table(d3dat$bef)  # zeigt wie viele Funde pro Schnitt vorhanden sind.
```

- Für die weiteren Berechnungen ist es sinnvoll, je ein Objekt für jeden Schnitt zu erzeugen. Mit *subset()* werden aus der vorhandenen Datentabelle alle Zeilen (Fälle) ausgewählt (s.o. 1.2.5.), die eine Bedingung erfüllen. Hier lautet die Bedingung *d3dat$bef=="bef_1"*. Dies ergibt einen logischen Vektor mit so vielen Elementen wie die Datentabelle Zeilen hat, und der den Eintrag TRUE enthält, wenn die Bedingung erfüllt ist. Man beachte: die doppelten (!) Ist-Gleich-Zeichen und die Anführungsstriche um den Level des Factor; schließlich werden als beizubehaltende Spalten die Spalten2 bis 5 gewählt, also 'id' und die Koordinaten.

```
bef1 <- subset(d3dat, d3dat$bef=="bef_1", 2:5)
bef2 <- subset(d3dat, d3dat$bef=="bef_2", 2:5)
```

2.4.2. Explorative 2D-Grafik (=anspruchsvolle Grafikausgestaltung)

- Zunächst betrachte man den 2D-Ausdruck der X- und Y-Koordinaten für Schnitt I. *ylim=c()* setzt die Grenzen des Ausdrucks auf 0 bis 6, und da das X-Y-Verhältnis mit *asp=1* auf 1 gesetzt ist, entsprechen die Grenzen in der X-Dimension diesen Grenzen.

```
plot(bef1[,2:3], asp=1, axes=FALSE,pch=21,cex=.5,bg=8,ylim=c(0,6), ylab="")
```

- Die Grabungsgrenzen bestehen aus einem einfachen Rechteck, das mit *rect(xminimum, yminimum, xmaximum, ymaximum)* gezeichnet wird. Mit *lty=4* wird der Linientyp (LineTYpe) Strichpunkt gewählt. Das Argument *border=* legt als Farbe einen dunklen Grauwert *gray(.3)* fest; *gray(.01)* wäre dunkelstes Grau, *gray(.99)* hellstes Grau.

```
rect(0,0,3.5,5, lty=4, border=gray(.3))
```

- Die Achsen werden extra definiert. Zunächst die X-Achse: *side=* wählt die Seite der Achse (1=unten, 2=links usw.). *at=* wählt die Position der Achsenbeschriftungen; hier ist es eine Sequenz *seq(0, 3.5, .5)* von 0 bis 3,5 im Abstand von 0,5 Einheiten. Die Position *pos=* wählt mit einem negativen Wert den Abstand der Achse vom Grafikrand nach Außen in Normalzeilenhöheneinheiten (NZE), hier 0,2 NZE. *cex.axis=* schließlich wählt die Größe der Achsenbeschriftung wiederum in NZE.

```
axis(side=1, at=seq(0,3.5,.5), pos=-.2, cex.axis=.7)
axis(side=2, at=seq(0,5,.5), pos=-.2, cex.axis=.7, las=1) # Y-Achse
# 'las=1' sorgt für waagrechte Achsenbeschriftung der Y-Achse.
```

```
      title(line=-3,main="Aufsicht auf die Fundverteilung in Schnitt I")
         # Ein von oben eingerückter Titel

 title(line=-3, ylab="y") # an Grafik herangerückte Y-Achsen-Beschriftung
```

- Das Gleiche geht für Schnitt 2. Man beachte die veränderten Koordinaten und Fenstergrenzen

```
plot(bef2[,2:3], asp=1, axes=FALSE, pch=21, cex=.5, bg=8, ylim=c(10,15))

rect(2,10,7,15, lty=4, border=gray(.3)) # Die veränderten Grabungsgrenzen

         axis(side=1, at=seq(2,7,.5), pos=9.8, cex.axis=.7)
         # und die veränderten Beschriftungs- und Positionskoordinaten

         axis(side=2, at=seq(10,15,.5), pos=1.8, cex.axis=.7, las=1)

title(main="Aufsicht (X-Y-Ebene) auf die Fundverteilung in Schnitt II")
```

2.4.3. Interaktive 3D-Grafik

- Man beginnt mit einer interaktiven 3D-Grafik der Fundverteilung zur visuellem Inspektion, ob die Z-Werte in Ordnung sind. *aspect="iso"* sorgt für ein einheitliches Achsen-Verhältnis, *box=FALSE* verhindert einen unschöne 3D-Kanten-Kubus. Als grafische Objekte werden an den Koordinaten durch *type="s"* sog. Sphären – das sind Kugeln mit einem bestimmten Durchmesser – mit einem Durchmesser von 5 cm, *radius=.025*, gezeichnet Die Kugelfarbe ist blau und die Z-Koordinaten der Grafik liegen zwischen 0 und -2 m.

```
plot3d(bef1[,2:4],aspect="iso",box=FALSE,type="s",radius=.025,col=4,zlim=c(0
,-2))
```

2.4.4. Berechnung und Darstellung der 3D-KDE

- Es gibt nun mehrere Schätzverfahren für die optimale Bandbreite. Hier wird das sog. Einsetzverfahren ("plug-in") verwendet. Für andere Verfahren siehe die ks-Hilfe-Einträge zu *Hbcv*, *Hlscv* und *Hscv*. Benötigt wird ein als Matrix interpretierbares Objekt – hier es eine dreispaltige Datentabelle.
  ```
  (h_bef1 <- Hpi(bef1[,2:4]))
  ```
- Das Ergebnis ist eine Matrix mit Bandbreiten für jede Dimension auf der Diagonalen und den jeweiligen Verzerrungen der "Bandbreitenkugel". Mit dieser Bandbreitenmatrix wird die KDE berechnet. Das erste Argument sind Koordinatenspalten aus der Datentabelle, das zweite ist die Bandbreitenmatrix und das dritte legt die Auflösung in X-,Y-, und Z-Richtung fest – hier sind es 2,5 cm zwischen den 3D-Voxeln.
  ```
  kd_bef1 <- kde(bef1[,2:4],  h_bef1, gridsize=c(141,201,81))
  ```
- Der eigentliche Ausdruck der 3-D-KDE wird der bestehenden Grafik hinzugefügt. *cont=seq(10,90,20)* legt RELATIVE Dichtekonturen/"Dichte-Isolinien" fest auf Bereiche mit den oberen 90% der Dichtewerte (das ist die 10 bei der Sequenz!), den oberen 70% der Dichtewerte (das wäre eine 30, der zweite Sequenzwert !) usw. ; stünde hier die Sequenz *seq(20,65,15)* wäre die äußerste, durchsichtigste Isolinie die der oberen 80% der Dichtewerte, dann kämen die oberen 65 %, die oberen 50 % und die oberen 35 %. Die Isolinien sind so immer der Schätzung angepasst weil relativ definiert. Die Farben der "Isolinien" sind fünf Regenbogenfarben in umgekehrter Reihenfolge. Die Punkte werden nicht noch einmal gezeichnet *drawpoints=FALSE.* Die Durchsichtigkeit der fünf "Isolinien" liegt bei 95 %, 85 % usw. was mit *alphavec=seq(.05,.45,.1)* gesteuert wird; ein Wert von 1 wäre undurchsichtig bzw. eine Durchsichtigkeit von 0 %. Schließlich wird die eigentlich als eigenständiger Ausdruck generierte Grafik noch der bestehenden hinzugefügt *add=TRUE*.

```
plot(kd_bef1,aspect="iso", box=FALSE, cont=seq(10,90,20), colors=rev(rainbow
(5)) , drawpoints=FALSE, alphavec=seq(.05,.45,.1 ), add=TRUE)
```

- Um leichter lernend zu experimentieren hier der Grafik-Code nochmals für eine 3D-KDE in geringerer Auflösung.

```
      kd_bef1_2 <- kde(bef1[,2:4],  h_bef1, gridsize=c(71,101,41))
      # zunächst schneller verarbeitbare KDE mit geringerer Auflösung

plot3d(bef1[,2:4], aspect="iso", box=FALSE, type="s", radius=.025, col=4,
zlim=c(0,-2))

plot(kd_bef1_2,aspect="iso",box=FALSE,cont=seq(10,90,20),colors=rev(rainbow
(5)), drawpoints= FALSE, alphavec=seq(.05,.45,.1 ), add=TRUE)
```

- Einen Bildschirmschnappschuss im .png-Format exportiert man so:
  ```
  rgl.snapshot( "3D_KDE_01.png", fmt="png")
  # Beachte Wahl neuen Dateinamens bei weiteren Schnappschüssen.
  ```

2.5. Literatur zu Methoden

Hier werden ein- bzw. weiterführende Werke zu den Methoden von Teil 2. aufgeführt. Die Zitate zu den verwendeten Paketen wurden oben an entsprechenden Stellen im Skript schon als .txt-Dateien aus R erzeugt. Die Kommentare in eckigen Klammern sind völlig subjektive Meinungen von Roth.

2.5.1. Korrespondenzanalyse (CA)

M. Greenacre, Correspondence Analysis in Practice (Boca Raton 2007, 2. Aufl.). [DAS CA-Buch!]
P. Legendre/L. Legendre, Numerical Ecology (Amsterdam 1998). [Erschöpfend aber unlesbar]
I. Leyer/K. Wesche, Multivariate Statistik in der Ökologie (Berlin 2007). [DAS Multivariat-Lehrbuch!]

2.5.2. Kanonische Korrespondenzanalyse (CCA)

D. Borcard/Fr. Gillet/P. Legendre, Numerical Ecology with R (New York 2011). [multivariate Zukunft]
P. Legendre/L. Legendre, Numerical Ecology (Amsterdam 1998). [Erschöpfend aber unlesbar]
I. Leyer/K. Wesche, Multivariate Statistik in der Ökologie (Berlin 2007). [DAS Multivariat-Lehrbuch!]

2.5.3. Punktfeldstatistik (PFS bzw. englisch *point pattern analysis*)

A. Baddeley, Analysing spatial point patterns in R. CSIRO and University of Western Australia. Workshop Notes Version 4.1, December 2010 [www.csiro.au/resources/pf16h]
J. Illian/A. Penttinen/H. Stoyan/D. Stoyan, Statistical Analysis and Modelling of Spatial Point Patterns (Chichester 2008). [Leider ohne Anbindung an R]
H. Stoyan/D. Stoyan, Fraktale – Formen – Punktfelder. Methoden der Geometrie-Statistik (Berlin 1992). [DAS dt. Lehrbuch zur PFS, damals gab es noch kein R...]

2.5.4. Kerndichteschätzung

Literatur zum 3D-Grafik-Paket 'rgl':
D. Adler, Interactive visualization of multi-dimensional data in R using OpenGL (Diplomarbeit Universität Göttingen 2002). rgl.neoscientists.org/arc/doc/RGL_MINITHESIS.pdf

Literatur zur 3D-KDE:
A. Bowman/A. Azzalini, Applied Smoothing Techniques for Data Analysis. The Kernel Approach with S-Plus Illustrations (Oxford 1997). [älter aber gut]
T. Duong, ks: Kernel Density Estimation and Kernel Discriminant Analysis for Multivariate Data in R. Journal of Statistical Software Volume 21, Issue 7, 2007. www.jstatsoft.org/v21/i07/paper

****

Wir bedanken uns für Ihre Teilnahme am Bamberg-Kurs!
Viel Erfolg und Spaß beim eigenen Nachrechnen und Ausprobieren!
Bitte kontaktieren Sie uns, wenn Sie auf Code-Fehler gestoßen sind. Danke.

Georg Roth und Jörg Wicke
(Köln und Halle im Februar 2012)

groth[at]uni-koeln.de
joerg.wicke[at]praehist.uni-halle.de