Eine Einführung in R

0 Allgemein

0.1 R starten, R beenden

Wenn man R startet, beginnt man **einen Session**. Man beendet einen Session mit

```
q()
```

(Hier die Frage *Save workspace image* mit c (cancel) beantworten).

0.1 Getting out of trouble

Sie geben einen Befehl ein, und es passiert nichts.
Lösung: ESC-Taste (Windows, Mac) oder Steuerung-C

0.2 Objekte erzeugen und listen

q() ist eine Funktion, und das ist daran zu erkennen, dass () Klammern gleich hinter dem Namen stehen. Wenn der Funktionsname ohne Klammern eingegeben wird, dann wird der Inhalt von der Funktion selbst gezeigt (zB bei Eingabe von q). Die Funktion ls() listet die Objekte im Session:

```
ls()
```

Ein neues Objekt wird mit = (assign) oder <- erzeugt

```
x = 3
y <-4
ls()
```

Kommentare können nach dem Rautenzeichen # eingegeben werden und werden von R ignoriert:

```
# So erzeuge ich einen Schriftzeichen-Vektor mit Inhalt "Phonetik"
a = "Phonetik"

# Neue Objekte erzeugen mit dem selben Inhalt
x = y = z = 4
```

Woran erkenne ich die Eigenschaft vom Objekt?

```
class(x)
class(a)
class(ls)
```

0.3 Graphik

Ein graphisches Fenster erscheint, wenn eine graphische Funktion verwendet wird:

```
plot(1:10)
```

Um ein neues graphisches Fenster zu erzeugen, win.graph() in Windows, z.B. X11() in Linux, quartz() auf einem Macintosh. Siehe help(Devices) für weitere Details.

0.4 Objekte sichern

Objekte werden in R **gleich überschrieben ohne Warnmeldung**

```
y = 4
y
4
y ="Phonetik"
y
"Phonetik"
```

Wenn Objekte nicht mit der rm() Funktion entfernt werden, dann sind sie während des Sessions **im Workspace** vorhanden. Der Workspace-Verzeichnis ist in pfad/.Rdata (pfad\.Rdata unter Windows), wo pfad diesen Wert hat:

```
# Die Funktion get-working-dirctory
getwd()
```

Objekte werden durch y auf *Save workspace image* gespeichert (und sind im nächsten Session vorhanden) nachdem R mit q() beendet wird. Eine andere (bessere) Möglichkeit: **Objekte in einem Library** mit der save() Funktion speichern:

```
# Alle Objekte im Default-Verzeichnis werden in meinlib gespeichert
# im Verzeichnis H:/Eigene Dateien
save(list  =ls(),  file = "H:/meinlib")
```

Man kann dann n antworten auf Save Workspace Image nach q() . Im nächsten Session sind diese Objekte dann vorhanden mit:

```
attach("H:/meinlib")
```

0.5 Libraries

Wie schon oben gezeigt, werden vorhandene R-Funktionen und Objekte in verschiedenen **Libraries** zusammengepackt.

```
# Die im Session vorhandenen libraries
search()
```

Der erste Pfad in dieser Liste entspricht (per Default) dem Verzeichnis pfad/.Rdata (siehe 0.3 oben). Der Inhalt vom einem Library lässt sich mit ls(pos=*n*) feststellen (*n* ist die Reihenfolge in search()). z.B.

```
attach("H:/meinlib")
search()

ls(pos=2)
```

Die gespeicherten Objekte in allen diesen Libraries sind in einem R Session nach der Ganzzahl-Reihenfolge zugänglich (d.h. R sucht für ein Objekt zuerst in ls(pos=1), dann ls(pos=2) ....).

Die **auf dem Rechner bereits installierten (aber ggf. im Session noch nicht vorhanden)** Libraries sind hier:

```
library()
```

oder
```
.packages(all.available = TRUE)
```

library(x) ermöglicht Zugang zu einem installierten aber im Session noch nicht vorhandenen Library, z.B:

```
library(MASS)
```

Die **in R vorhandenen, und ggf. noch nicht installierten Libraries** können hier gesichtet werden: http://cran.r-project.org/ -> packages. In den neueren Versionen von R können diese Libraries (Packages) mit der Funktion install.packages() heruntergeladen werden. z.B[1].

```
install.packages("AlgDesign")
```

Der package ist vorhanden mit:

```
library(AlgDesign)
```

und erscheint dann als Pfad in search()

0.6 Eigene Funktionen erstellen

Wir werden fast immer von vorhanden Funktionen Gebrauch machen -- und dann manchmal eigene Funktionen erstellen. Die einfachste Methode, eine erstellte Funktion in R zu laden ist mit cut-and-paste. zB diese Funktion ins R Fenster kopieren

```
meinef <- function(x="hello")
{
x
}

meinef()

meinef("etwas anders eingeben")
```

Die Funktion könnte mit meinef <- edit(meinef) editiert werden. Oder (besser) die Funktion könnte als Text irgendwo gespeichert werden und dann mit der source() Funktion eingelesen werden. zB speichern Sie die obigen Funktion in einer Textdatei in H:\fun.txt, dann:

---

[1] Auf manchem Betriebssystemen:
```
install.packages("AlgDesign", "pfad", "http://cran.r-project.org")
library(AlgDesign)
```

```
source("H:/fun.txt")
```

0.7 Dateneingabe, Datenausgabe
Vektoren, Matrizen usw. die Sie erstellt haben, können mit read.table() in R eingelesen werden. zB speichern Sie diese Daten als mat.txt in Eigene Dateien:

```
Vokal VPn F2
i: AB   2200
i: CD   2800
i: EF   2900
```

(NB ***Dateneingabe nach der letzten Zeile***)

Dann in R:

```
mat = read.table("H:/mat.txt", header=T)
mat
Vokal VPn   F2
1     i:  AB 2200
2     i:  CD 2800
3     i:  EF 2900
```

Mit write.table() können Objekte diverser Art außerhalb von R geschrieben werden

```
w = 0:10
write.table(w, "H:/w.txt", row.names=F, col.names=F)
```

Segmentlisten werden in R direkt mit emu.query() erzeugt, oder sie können mit read.emusegs() eingelesen werden, wenn sie bereits von Emu gespeichert worden sind. Segmentliste können mit write.emusegs() gespeichert werden:

```
write.emusegs(vowlax, "H:/vowlax.txt")
```

0.8 Help!

Siehe auch 0.2 oben, wenn R nicht mehr zu funktionieren scheint.

Für eine Einführung in R
help.start() -> 'An Introduction to R'
help(pnorm)
# oder ?pnorm

```
example(density)
```

```
apropos("spline")
```

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

Überblick der vorhanden R-Funktionen:
http://www.rpad.org/Rpad/Rpad-refcard.pdf

1. Vektoren

1.1 Die c() Funktion

```
# Numerisch
newdata  = 20
newdata

# Schriftzeichen
vek = "ips"

# Numerisches Vektor x mit 6 Elementen
x  = c(3, 4, 6, 89.3, 0, -10)

# Schriftzeichen-Vektor mit 4 Elementen
labs  =  c("E", "A", "E", "i:")

# Elemente 2 bis 4 von y mit 0 überschreiben
y[2:4] = 0
```

1.2 Indizierung

(Kein Komma in [ ] da es sich um einen Vektor handelt)

```
x[2]                    # Element 2 of x
x[2:4]                  # Elemente 2 bis  4
x[c(1,4)]               # Elemente 1 und 4
x[-2]                   # Alle Elemente außer Element 2
x[2:4] = 0              # Elemente 2 bis 4 auf  0 setzen

# Alle Elemente von x ohne den 2en und 5en?

# Elemente von x in der anderen Reihenfolge (also von 6 bis 1)?
```

1.3 Arithmetische Manipulationen von Vektoren (*, /, +, -)

```
a = c(10, 4, 20)
a * 10

b = c(5, 2, 7)
a + b
15  6 27
sum(a)                  # Summe aller Elemente
sqrt(a)                 # Wurzel pro Element
a^3                     # Jedes Element hoch 3
range(a)                # Bereich
max(a)                  # Maximum
mean(a)                 # Mittelwert
```

1.4 Einige nützliche Funktionen, um Vektoren zu manipulieren

length()

```
y = c("i", "i", "a", "a", "E", "E", "E", "E", "U")
length(y)
```

Wie könnte man das letzte Element von y listen? Das vorletzte? Das Vorletzte und Letzte zusammen?

Intervalle: seq()

```
seq(10, 20, length=5)  # 5 Intervalle zwischen 10 und 20
seq(10, 20, by=1.5)    # In Intervallen von 1.5
```

Wiederholdung: rep()

```
a = c(10, 4, 20)
rep(a, 4)
rep(a, each=2)
```

unique()

```
y = c("i", "i", "a", "a", "E", "E", "E", "E", "U")
unique(y)
 "i" "a" "E" "U"
```

Tabelle, table()

```
table(y)
y
a E i U
2 4 2 1
```

paste()

```
paste(y, "V", sep=".")
 "i.V" "i.V" "a.V" "a.V" "E.V" "E.V" "E.V" "E.V" "U.V"
```

String manipulation: nchar(), substring()

```
cities = c("London", "Paris", "Hamburg", "Verona", "New York")

# Anzahl der Schriftzeichen pro Element
nchar(cities)
 6 5 7 6 8

# Die ersten 3 Schriftzeichen pro Element
substring(cities, 1, 3)
```

2. Matrizen

2.1 Allgemein

```
a = c(10, 3, 8, 7)
b = c(11, 45, 20, -1)
x = rbind(a, b)        # 2 x 4 Matrix
y = cbind(a, b)        # 4 x 2 Matrix
nrow(x)                # Reihenanzahl
ncol(x)                # Spaltenanzahl
```

```
dim(x)                  # Dimensionen

rownames(x) = c("subject1", "subject2")        # Reihen-namen
colnames(x)  = c("A", "B", "C", "D")           # Spalten-namen
x
```

2.2 Anwendung arithmetischer Funktionen

```
x -20
x
   [,1] [,2] [,3] [,4]
     10    3    8    7
     11   45   20   -1
y = x*2
y
   [,1] [,2] [,3] [,4]
     20    6   16   14
     22   90   40   -2

x + y
   [,1] [,2] [,3] [,4]
     30    9   24   21
     33  135   60   -3
```

2.3 Anwendung von Vektoren-Funktionen auf Matrizen

Viele Funktionen werden auf **alle** Elemente von einer Matrix angewandt. z.B.

```
a = c(1, 5, 9)
mean(a)

# Mittelwert aller Elemente der obigen Matrix
mean(x)
```

Die apply() Funktion wendet eine Vektoren-Funktion **entweder auf die Reihen (1) oder die Spalten (2)**. zB Mittelwert der Reihen

```
apply(x, 1, mean)
# Maximum der Spalten
apply(x, 2, max)
```

2.4 Indizierung

> ***Was vor dem Komma erscheint, bezieht sich auf die Reihen; was nach dem Komma erscheint, auf die Spalten***

```
x = 1:4
y = c(-10, 0, 20, 30)
z = c(1.1, 1.2, 1.3, 1.4)
newmat = rbind(x, y, z)          # Eine 3 x 4 Matrix
newmat[2:3,]                     # Reihen 2 bis 3
newmat[,2:4]                     # Spalten 2 bis 4
newmat[2:3,3:4]                  # R. 2 bis 3 von S. 3 bis 4
```

newmat[,c(2,4)] # S. 2 *und* 4
Reihe 1 von Spalte 2 und 4

Spalten 1 bis 3 aller Reihen, außer Reihe 2

Reihen 2 und 3 aller Spalten, außer Spalten 2 und 4

3. Segmentlisten
Eine Emu-Segmentliste ist ein sogenannter Data-Frame, in dem Schriftzeichen und numerische Vektoren verbunden werden können, **die sich wie eine Matrix behandeln lässt**. Daher können sehr viele Matrix-Funktionen auf Segmentlisten angewandt werden, insbesondere diejenigen für die Indizierung. Bitte daher auch weiterhin merken:

> ***Was vor dem Komma erscheint, bezieht sich auf die Reihen; was nach dem Komma erscheint, auf die Spalten***

Eine Emu-Segmentliste hat immer 4 Spalten, die entweder mit Indizierung [,n] oder mit den folgenden Funktionen extrahiert werden können:

- Die Etikettierungen: label()
- Start-Zeiten (linke Zeigrenzen) start()
- End-Zeiten (rechte Zeitgrenzen): end()
- Die Äußerungen : utt()

(Ausprobieren an der eingebauten Segmentliste vowlax)

Die Dauer kann berechnet werden entweder mit start(x) - end(x) oder mit mudur(x)[2], wo x eine Segmentliste ist

Erste 10 Segmente in vowlax?

Segmente 10, 15, 18 von vowlax?

Anzahl der Reihen in vowlax?

Der letzte Segment von vowlax?

Die Etikettierungen der ersten 20 Segmente?

Eine Tabellierung der Etikettierungen der ersten 100 Segmente?

Die Dauern der Segmente

Die durchschnittliche Dauer der ersten 45 Segmente

---

[2] dur(x) in den neueren Emu-R Versionen

4. Kurze Zusammenfassung der wichtigsten Funktionen aus 0-3, die für die Fragen benötigt werden

Allgemein

q() R beenden
ESC wenn R nichts mehr zurückgibt/hängen bleibt

```
ls() oder ls(pos=3)                    # Objekte listen
a = 4                                  # Objekt erzeugen
a = "Phonetik"                         # Objekt erzeugen
(Schriftz.)
class(a)                               # Eigenschaft vom Objekt
a
help.start() , help(Funktion)          # Hilfe Funktionen
```

Vektoren

```
x  = c(3, 4, 6, 89.3, 0, -10)
x[2]                    # Element 2
x[2:4]                  # Elemente 2 bis  4
x[c(1,4)]               # Elemente 1 und 4
x[-2]                   # Alle Elemente außer  2
sum(a)                  # Summe
a^3                     # a hoch 3
```

Vektoren-Funktionen

```
range(a)                # Bereich
max(a)                  # Maximum
mean(a)                 # Mittelwert
median(a)               # Median
sd(a)                   # Standardabweichung
log(a, base=10)         # Logarithmus Basis 10
length(a)               # Anzahl der Elemente
```

Matrizen

```
nrow(x)                 # Reihenanzahl
ncol(x)                 # Spaltenanzahl
dim(x)                  # Dimensionenanzahl
newmat[2:3,]            # Reihen 2 bis 3
newmat[,2:4]            # Spalten 2 bis 4
newmat[2:3,3:4]         # R. 2 bis 3 von S. 3 bis 4
newmat[,c(2,4)]         # S. 2 und 4
```

Segmentliste

```
label(x)                # Etikettierungen
start(x)                # Startzeiten
end(x)                  # Endzeiten
utt(x)                  # Äußerungen
mudur(x)                # Dauer
```

Indizierung: Siehe Matrizen

5. Fragen

1. Für die vorhandene Segmentliste von Diphthongen, dip, erstellen Sie einen Vektor der Dauern[3]:
d = mudur(dip)

Schreiben Sie R-Befehle für:

1.1 Die Dauer von 10en Segment
d[10]

1.2 Die Dauern der 12en und 14en Segmente
1.3 Die Dauern der Segmente 20-25, 29, und 40-45.
1.4 Die Dauern aller Segmente außer dem zweiten
1.5 Die Dauern aller Segmente außer den ersten 10
1.6 Die Dauern aller Segmente außer den letzten 2
1.7 Die Dauern aller Segmente minus 50 ms
1.8 Die Median-Dauer aller Segmente
1.9 Die Standardabweichung der Dauern in Frage 1.3
1.10 Die Summe der Dauern über alle Segmente
1.11 Die Dauern der Segmente minus den Dauer-Mittelwert.

2. vowlax.fdat.5 ist eine vierspaltige Matrix von F1-F4 zum zeitlichen Mittelpunkt von 410 Vokalen. Benennen Sie dieses Objekt um auf diese Weise (damit Sie nicht so viel zum Tippen haben)

form = vowlax.fdat.5

Schreiben Sie R-Befehle für die folgenden Fälle
2.1 F1-F4 der ersten 5 Segmente
form[1:5,]

2.2 F1-F4 von Segmenten 20, 22, 28
2.3 F1 und F2 von Segmenten 20-25
2.4 F2 und F4 von Segmenten 30, 32, 33
2.5 F1, F2, and F4 von Segmenten 1, 10, 20
2.6 F2-F4 der letzten 3 Segmente
2.7 F2+F3 aller Segmente (also ein Wert, F2+F3 pro Segment)

2.8. Logarithmus Basis 10 von F2/F1 aller Segmente

2.9 $$\frac{F2 - F2_m}{F2_s}$$

über alle Segmente berechnet. $F2_m$ und $F2_s$ sind hier der F2-Mittelwert und F2-Standardabweichung.

---

[3] d = dur(dip) in den neueren Emu-R Versionen