Einführung in R

Björn Voß

Institut für Bioverfahrenstechnik - JP Computational Biology

Die Statistiksoftware R – http://www.r-project.org

- Softwareumgebung zur statistischen Datenanalyse
- Open Source $\Rightarrow$ frei verfügbar
- R-Studio ist eine gute Benutzeroberfläche für R
- R-Studio Server unter https://rstudio.rnabioinfo.de
- Anmeldedaten in ILIAS
- ILIAS: Einführungsskript von Andreas Handl, Universität Bielefeld

R als Taschenrechner

Die Grundrechenarten

```
> 2.1 + 2
[1] 4.1

> 2.1 - 2
[1] 0.1

> 2.1 * 2
[1] 4.2

> 2.1 / 2
[1] 1.05
```

Potenzieren & Wurzel ziehen

```
> 2.1^2
[1] 4.41

> 2^0.5
[1] 1.414214

> sqrt(2)
[1] 1.414214
```

Funktionen in R

`sqrt` ist eine **Funktion** die von `R` bereitgestellt wird.

Funktionen besitzen in der Regel ein oder mehrere Argumente, die der Funktion in runden Klammern hinter dem Funktionsnamen übergeben werden.

`R` gibt üblicherweise 6 Dezimalstellen an. Mit der Funktion `round()` kann man runden.

```
> round(sqrt(2))
[1] 1
```

Das war etwas zu viel des Guten. Wie kann man der Funktion `round()` mitteilen, dass man z.B. auf zwei Nachkommastellen runden möchte?

Die integrierte Hilfe

R verfügt über ein integriertes und sehr ausführliches Hilfesystem, welches mit der Funktion `help()` aufgerufen werden kann. Der folgende Aufruf hilft uns weiter:

```
> help(round)
```

Wie wir der Beschreibung entnehmen können, kann die Funktion mit einem zweiten Argument aufgerufen werde, welches die Zahl der Nachkommastellen angibt.

```
> round(sqrt(2), digits=2)
[1] 1.41
```

Funktionsargumente

Ist, wie im Fall der Funktion `round()`, die Zuordnung der Argumente eindeutig reicht auch folgendes:

```
> round(sqrt(2),2)
[1] 1.41
```

Will man sicher gehen und sich nicht um die Reihenfolge kümmern müssen, kann man die Argumente mit Namen übergeben, oder mit Abkürzungen, sofern diese eindeutig sind.

```
> round(digits=2,x=sqrt(2))
[1] 1.41
```

```
> round(d=2,x=sqrt(2))
[1] 1.41
```

Datenstrukturen

Bei statistischen Analysen hat man es meistens mit größeren Datenmengen zu tun, z.B. 20 Messungen der Länge eines Stiftes.

Solche Daten werde in R als Vektor gespeichert und die einzelnen Messungen bilden die Komponenten des Vektors.

Beispiel: Schallplattensammler
Ein Plattensammler hat im letzten halben Jahr fünf Langspielplatten (LPs) bei einem Händler gekauft und dafür folgende Preise in US$ bezahlt:

22 30 16 25 27

In R verwenden wir dazu fie Funktion c() (von combine) und erhalten die gezeigte Ausgabe:

```
> c(22,30,16,25,27)
[1] 22 30 16 25 27
```

Variablen

Um mit einem Vektor sinnvoll arbeiten zu können, speichern wir ihn in einer **Variablen**. Dies geschieht mit dem Zuweisungsoperator <-. Auf der linken Seite steht der Name der Variablen und rechts der Aufruf, dessen Ergebnis der Variablen zugewiesen werden soll.

Variablennamen dürfen nur aus Buchstaben, Ziffern und dem Punkt bestehen. In unserem Beispiel soll die Variable `lp` heißen. Wir geben ein

```
> lp <- c(22,30,16,25,27)
```

Variablen

Eine Variable bleibt während der gesamten Sitzung erhalten. Man kann sie mit dem Befehl `rm` löschen. Mit dem Befehl `ls()` kann man alle Objekte auflisten. Den Inhalt einer Variablen kann man durch Eingabe des Namens sehen:

```
> ls()
[1] lp

> lp
[1]  22 30 16 25 27
```

`R` unterscheidet Groß- und Kleinschreibung. Die Namen `lp` und `Lp` bezeichnen daher verschiedene Objekte.

```
> Lp
Fehler: Objekt "Lp" nicht gefunden
```

Rechnen mit Variablen

Wir wollen nun die Preise der LPs von Dollar in Euro umrechnen und nehmen einen Kurs von 1 USD = 0.774 EURO an. Eigentlich muss man jeden Preis mit 0.774 multiplizieren, man kann aber auch einfach den Vektor `lp` mit 0.774 multiplizieren:

```
> lp * 0.774
[1] 17.028 23.220 12.384 19.350 20.898
```

Um das ganze auf zwei Stellen zu runden, machen wir folgendes:

```
> round(lp * 0.774,2)
[1] 17.03 23.22 12.38 19.35 20.90
```

Rechnen mit Variablen

Addition, Subtraktion, usw. funktionieren analog:

```
> lp + 3.4
[1] 25.4 33.4 19.4 28.4 30.4
```

Man kann die einzelnen Komponenten eines Vektors auch gezielt adressieren. Hierzu gibt man den Namen des Vektors gefolgt von eckigen Klammern (`[]`), zwischen denen die Nummer der Komponente steht, ein. Für den Preis der 2. LP also folgendes:

```
> lp[2]
[1] 30
```

Rechnen mit Variablen

Um den Preis der Platte zu erhalten, die man zuletzt gekauft hat, benötigt man die Länge des Vektors `lp`. Diese erhält man von der Funtion `length()`.

```
> length(lp)
[1] 5
> lp[length(lp)]
[1] 27
```

Man kann auch auf mehrere Komponenten gleichzeitig zugreifen:

```
> lp[c(1,2,3)]
[1] 22 30 16
```

Folgen

Für Vektoren mit aufeinander folgenden natürlichen Zahlen, gibt es eine Kurschreibweise in R und zwar den Operator `:`.

```
> 1:3
[1] 1 2 3
> 4:10
[1]  4  5  6  7  8  9 10
> 3:-2
[1]  3  2  1  0 -1 -2
```

Man kann also auch mit `> lp[1:3]` die ersten drei Komponenten des Vektors wählen.

Weitere Funktionen

Hier noch ein paar Funktionen zum Arbeiten mit Vektoren

```
> sum(lp)
[1] 120
> min(lp)
[1] 16
> max(lp)
[1] 30
> sort(lp)
[1] 16 22 25 27 30
> sort(lp, decreasing=TRUE)
[1] 30 27 25 22 16
```

Nicht-numerische Daten

Die Urliste des Geschlechts der 10 Teilnehmer eines Projektes ist:

`w m w m w m m m w m`

Wie können wir das in `R` eingeben?

Dazu verwendet man Zeichenketten, also eines oder mehrere Zeichen zwischen Hochkommata, wie z.B. `"Statistik"` oder `"Stuttgart"`. Wir nennen den Vektor Geschlecht:

```
> Geschlecht <- c("w","m","w","m","w","m","m","m","w","m")
> Geschlecht
 [1] "w" "m" "w" "m" "w" "m" "m" "m" "w" "m"
```

Faktoren – Qualitative Merkmale

Mit der Funktion `factor()` können wir die Komponenten in Ausprägungen eines qualitativen Merkmals (Faktor) umwandeln:

```
> Geschlecht <- factor(Geschlecht)
> Geschlecht
 [1] w m w m w m m m w m
Levels: m w
```

Mit Vektoren vom typ `factor` kann man ganz normal arbeiten:

```
> Geschlecht[2]
[1] m
Levels: m w
> Geschlecht[5:length(Geschlecht)]
[1] w m m m w m
Levels: m w
```

Tabellarische Daten

Bei statistischen Analysen hat man es meistens mit tabellarischen Daten zu tun. Z.B.

Alter	Alter der Mutter	Alter des Vaters
29	58	61
26	53	54

Für solche Fälle verwendet man in `R` eine Matrix, die aus r Zeilen und s Spalten besteht. Die Funktion `matrix()` wird folgendermaßen aufgerufen:

```
matrix(data, nrow=1, ncol=1, byrow=F)
```

`data` ist der Vektor mit den Elementen der Matrix, `nrow` die Zeilen- und `ncol` die Spaltenzahl.

Matrizen

Standardmäßig wird eine Matrix spaltenweise eingegeben, d.h.

```
> alter<-matrix(c(29,26,58,53,61,54),2,3)
> alter
     [,1] [,2] [,3]
[1,]   29   58   61
[2,]   26   53   54
```

Soll zeilenweise aufgefüllt werden, setzt man `byrow=TRUE`:

```
> alter<-matrix(c(29,58,61,26,53,54),2,3,byrow=TRUE)
> alter
     [,1] [,2] [,3]
[1,]   29   58   61
[2,]   26   53   54
```

Zugriff auf Matrizen

Der Zugriff auf Elemente einer Matrix erfolgt in vergleichbarer Weise wie bei Vektoren, nur dass man zwei Positionen angeben muss. Um z.B. auf das erste Element in der zweiten Spalte zuzugreifen, gibt man ein:

```
> alter[1,2]
[1] 58
```

Die ganze erste Zeile bzw. zweite Spalte erhält man mit:

```
> alter[1,]
[1] 29 58 61
```

```
> alter[,2]
[1] 58 53
```

Man kann auch die Gesamt-, Zeilen- und Spaltensumme berechnen:

```
> sum(alter)
[1] 281
```

```
> rowSums(alter)
[1] 148 133
```

```
> colSums(alter)
[1]  55 111 115
```

Arbeiten mit Matrizen

Allgemein kann man mit `apply(x,margin,fun)` eine Funktion `fun` auf die Zeilen (`margin=1`) bzw. Spalten (`margin=2`) anwenden. Man kann also die Zeilen- und Spaltensummen auch folgednermaßen berechnen:

```
> apply(alter,1,sum)
[1] 148 133
```

```
> apply(alter,2,sum)
[1]  55 111 115
```

Der Vorteil von `apply()` ist, dass ich beliebige andere Funktionen verwenden kann:

```
> apply(alter,1,min)
[1] 29 26
```

```
> apply(alter,2,max)
[1] 29 58 61
```

Datensätze mit quantitativen und qualitativen Merkmalen

Will man Datensätze wie

Alter	29	26	24
Geschlecht	m	w	m

abspeichern, dann verwendet man in R dazu sog. Datentabellen, die mit der Funktion `data.frame()` erzeugt werden:

```
> sexage <- data.frame(sex=c("m","w","m"),age=c(29,26,24))
> sexage
  sex age
1   m  29
2   w  26
3   m  24
```

Arbeiten mit Datentabellen

Der Zugriff auf die Elemente erfolgt wie bei Matrizen

```
> sexage[2,2]
[1] 26
```

```
> sexage[2,]
  sex age
2   w  26
```

```
> sexage[,1]
[1] m w m
Levels: m w
```

Das letzte Beispiel zeigt die automatische Umwandlung von Zeichenketten zu Faktoren.

R-Intern sind Datentabellen Listen, und daher kann man auf Datentabellen wie auf Listen zugreifen. Auf Komponenten einer Liste kann man über doppelte, eckige Klammern (`[[]]`) oder über `Name_Liste$Name_Komponente` zugreifen:

```
> sexage[[1]]
[1] m w m
Levels: m w
```

```
> sexage$sex
[1] m w m
Levels: m w
```

```
> sexage[[2]]
[1] 29 26 24
```

```
> sexage$age
[1] 29 26 24
```

Die Funktionen `attach()` und `detach()`

Wenn man nur mit einer Datentabelle arbeitet ist es lästig immer den Namen der Datentabelle gefolgt von dem Merkmal einzugeben. Mit der Funktion `attach()` kann man die Variablen unter ihrem Namen direkt ansprechbar machen. `detach()` macht das wieder rückgängig.

```
> attach(sexage)
> age
[1] 29 26 24
> sex
[1] m w m
Levels: m w
```

```
> detach(sexage)
> age
Fehler: Objekt 'age' nicht gefunden
> sex
Fehler: Objekt 'age' nicht gefunden
```

Daten aus Dateien

Größere Datensätze will man natürlich nicht von Hand eingeben, insbesondere wenn Sie bereits als Datei vorliegen.

`R` bietet einige Funktionen zum Einlesen von Dateien. Liegt die Datei als reine Textdatei (ASCII) vor, so kann sie mit der Funktion `read.table()` eingelesen werden.

Geschlecht	Alter	Mutter	Vater	Geschwister
m	29	58	61	1
w	26	53	54	2
m	24	49	55	1

Wichtig: Die Spalten sind durch Leerzeichen (eines oder mehrere) getrennt.

Daten aus Dateien

Mit der Funktion `read.table()` lassen sich die Daten aus einer Datei Beispiel_Daten.txt folgendermaßen einlesen:

```
> bidaten <- read.table("Beispiel_Daten.txt",header=TRUE)
> bidaten
> bidaten
  Geschlecht Alter Mutter Vater Geschwister
1          m    29     58    61           1
2          w    26     53    54           2
3          m    24     49    55           1
```

Wichtig: Korrekter Pfad zur Datei, Name in Anführungszeichen, `header=TRUE` falls 1. Zeile Spaltennamen enthält.

Konflikte

```
> attach(bidaten)
The following object is masked _by_ .GlobalEnv:
    Geschlecht
```

Hier gab es ein Problem, da es bereits eine Variable mit dem Namen Geschlecht gibt. Wir müssen die "alte" Variable erst löschen oder umbenennen.

```
> Geschlecht
 [1] w m w m w m m m w m
Levels: m w
> Ges <- Geschlecht
> rm(Geschlecht)
> Geschlecht
 [1] m w m w w w m m m m w m m m w m w w m w
Levels: m w
```

Selektieren und Filtern

Wenn wir für die Daten das Alter zwischen den Geschlechtern vergleichen wollen, müssen wir die Werte des Alters bei denen das Geschlecht den Wert w hat und die Werte des Alters bei denen das Geschlecht den Wert m hat getrennt selektieren. Wir wollen also Bedingungen überprüfen. Dafür gibt es in R folgende Operatoren:

```
== gleich      <  kleiner              >  groesser
!= ungleich    <= kleiner o. gleich    >= groesser o. gleich
```

Mit diesen kann man zwei Objekte vergleichen:

```
> 3<4          > 3>4          > 2+2 == 4
[1] TRUE       [1] FALSE      [1] TRUE
```

Bedingungen überprüfen – Vektoren

Man kann auch Vektoren mit Skalaren vergleichen:

```
> lp
[1] 22 30 16 25 27
> lp >25
[1] FALSE TRUE FALSE FALSE TRUE
```

Bei Letzterem spricht man auch von einem **logischen Vektor**. Nimmt man einen solchen logischen Vektor `v` und indiziert damit einen gleichlangen Vektor `x` durch `x[v]`, werden alle Komponenten aus `x` ausgewählt, die in `v` den Wert `TRUE` haben.

```
> lp [lp >=25]
[1] 30 25 27
```

liefert daher die Preise der Langspielplatten die mindestens 25 USD gekostet haben.

Bedingungen auf Vektoren

Wenn wir wissen wollen, welche LPs einen Preis ≥ 25 USD haben nutzen wir die Funktion `which()`. Mit `any()` und `all()` überprüfen wir ob mindestens eine bzw. alle Komponenten, die Bedingung erfüllen:

```
> which(lp>=25)        > any(lp>=25)        > all(lp>=25)
[1] 2 4 5              [1] TRUE             [1] FALSE
```

Verknüpfte Bedingungen

Bedingungen lassen sich mit den Operatoren & und | verknüpfen. Das logische & liefert TRUE wenn beide Bedingungen erfüllt sind. Das logische | liefert TRUE, wenn mindestens eine Bedingung erfüllt ist.

```
> 25 < 30 & 30 > 4
[1] TRUE
```

```
> 17 > 27 | 44 < 182
[1] TRUE
```

Man kann verknüpfte Bedingungen auch zum selektieren verwenden:

```
> lp[lp < 30 & lp > 25]
[1] 27
```

```
> lp[lp < 30 | lp > 25]
[1] 22 30 16 25 27
```

Selektieren

Wählen wir nun das Alter getrennt nach Geschlecht aus den Beispiel-Daten:

```
> alter.w <- Alter[Geschlecht=="w"]
> alter.w
[1] 26 25 25 23 26 23 24 23 23
> alter.m <- Alter[Geschlecht=="m"]
> alter.m
 [1] 29 24 23 27 25 24 24 29 28 24 24
```

Man kann auch die Funktion `split()` verwenden:

```
> split(Alter, Geschlecht)
$m
 [1] 29 24 23 27 25 24 24 29 28 24 24
$w
 [1] 26 25 25 23 26 23 24 23 23
```

Selektieren in Datentabellen

Eine weitere Möglichkeit zur Auswahl in Datentabellen bietet `subset()`:

```
> subset(bidaten, Geschlecht=="w")
  Geschlecht Alter Mutter Vater Geschwister
2          w    26     53    54           2
4          w    25     56    63           3
5          w    25     49    53           0
```

Mit dem Argument `select=` kann man einzelne Spalten selektieren:

```
> subset(bidaten, Geschlecht=="w", select=Mutter)
  Mutter
2     53
4     56
5     49
```

Grafiken in R

R ist ein mächtiges Werkzeug um Grafiken zu erzeugen.

```
> par(mfrow=c(2,2))                # 4 Grafiken in einem Bild
> x = c(0,15)                      # X-Koordinaten der Punkte
> y = c(3,6)                       # Y-Koordinaten der Punkte
> plot(x,y)                        # Zeichne die Punkte
> plot(x,y,pch=16)                 # Gefuellte Kreise
> plot(x,y,pch=16,type='l')        # Zeichne Linie zw. x und y
> plot(Alter,Mutter,pch=16)        # Streudiagramm Alter ~ Mutter
```

Man kann quasi alles an einem Plot verändern, also z.B. Achsenbeschriftung, Farbe, Titel und Legende. Details liefert das Einführungsskript.

Diagramme von Funktionen können mit der Funktion `curve()` gezeichnet werden:

```
> curve(x^2, from=-4, to=4)          # Quadratfunktion
> curve(sin(x), from=-pi, to=pi)     # Sinus

Die Dichtefunktion der Standardnormalverteilung:
> curve(1/sqrt(2*pi)*exp(-0.5*x^2), from=-4, to=4)
```

Eigene Funktionen definieren

Es gibt zwar sehr viele vordefinierte Funktionen in R, dennoch ist es manchmal hilfreich eigene Funktionen zu definieren. Dafür verwendet man `function()`:

```
> fun_name <- function(Argumente) {
    Koerper der Funktion, also was macht die Funktion
    return(Ergebnis)
  }
```

Eine Funktion Mittelwert würde man demnach folgendermaßen deklarieren:

```
> mittelwert <- function(x) {
    mw = sum(x)/length(x)
    return(mw)
  }
```

Viel Spaß mit R!