Statistische Software (R)

Paul Fink, M.Sc.

Institut für Statistik
Ludwig-Maximilians-Universität München

Vektoren, Matrizen, Listen und Data Frames

Grundlegende Operatoren und Funktionen

Aufruf der Hilfeseiten zu grundlegende Operatoren und Funktionen:

`?Arithmetic`	Grundlegende Operatoren für numerische Vektoren
`?Logic`	Operatoren für logische Vektoren
`?log`	Logarithmus und Exponens
`?Trig`	Trigonnometrische Funktionen
`?Special`	Z.B. Binomialkoeffizienten, Fakultät, etc.

Konstanten

Übersicht einiger Konstanten:

`pi`	Die Zahl π
`Inf`, `-Inf`	$\infty, -\infty$
`NaN`	Not a Number: z.B. 0/0[1] NaN
`NA`	Not available: fehlende Werte
`NULL`	leere Menge
`letters`	Kleinbuchstaben von a bis z
`LETTERS`	Großbuchstaben von A bis Z

Datentypen in R

DAS Datenobjekt ist ein Vektor mit Elementen des Typs

- **numeric**: ganzzahlige oder Gleitkomma-Werte,
- **character**: beliebige Zeichen,
- **logical**: die Zustände `TRUE` und `FALSE`,
- **list**: ein Objekt beliebigen Typs, auch wieder eine Liste (rekursive Datenstruktur!). Mehr dazu später.

Jeder Vektor besitzt Elemente (eines!!) Typs und hat eine Länge (`length()`).

Vektoren unterschiedlichem Typs

Vektor vom Typ **numeric**.

```
> num1 <- c(1.64, 1.96, 2.71, 3.14, 3.84)
> num1
[1] 1.64 1.96 2.71 3.14 3.84
```

Vektor vom Typ **character**.

```
> char1 <- c("Hallo", "Welt")
> char1
[1] "Hallo" "Welt"
```

Vektor vom Typ **logical**.

```
> logic1 <- c(TRUE, FALSE, FALSE, TRUE)
> logic1
[1]  TRUE FALSE FALSE  TRUE
> sum(logic1)
[1] 2
```


Vektoren unterschiedlichem Typs

Vektor vom Typ **list** kann einfache Vektoren, wie auch komplexere Objekte unterschiedlicher Klassen als Elemente enthalten.

```
> l1 <- list(numeric = num1, character = char1,
+   logical = logic1)
> l1
$numeric
[1] 1.64 1.96 2.71 3.14 3.84

$character
[1] "Hallo" "Welt"

$logical
[1]  TRUE FALSE FALSE  TRUE
```


Vektoren unterschiedlichem Typs

```
> l2 <- list(numeric = num1, character = char1,
+   logical = logic1, list = l1)
> l2
$numeric
[1] 1.64 1.96 2.71 3.14 3.84

$character
[1] "Hallo" "Welt"

$logical
[1]  TRUE FALSE FALSE  TRUE

$list
$list$numeric
[1] 1.64 1.96 2.71 3.14 3.84

$list$character
[1] "Hallo" "Welt"

$list$logical
[1]  TRUE FALSE FALSE  TRUE
```


Automatische Umwandlung

- Konstruktion einfacher Vektoren

```
> c(1,2,7)
> c("Hallo", "Welt")
```

- R wandelt den Typ eines Objektes *automatisch* um, wenn dies notwendig und möglich ist:

```
> TRUE + 2
[1] 3
> c("Hello", sqrt(2))
[1] "Hello"           "1.4142135623731"
> c(1:2, 3.14, exp(1i * pi))
[1]  1.00+0i  2.00+0i  3.14+0i -1.00+0i
```

Automatische Umwandlung

- Beispiele für häufige Umwandlungen sind

logisch	→	numerisch
logisch, numerisch	→	Text
numerisch	→	logisch

```
> as.numeric(rnorm(10) >= 1)
 [1] 0 0 0 0 0 0 1 1 1 0
> as.logical(c(0, pi))
[1] FALSE  TRUE
> c(2, "Hallo", TRUE)
[1] "2"     "Hallo" "TRUE"
```


Rechnen mit Vektoren

Wie in Linearer Algebra komponentenweise Addition und Subtraktion

```
> x <- 1:4
> y <- c(4,10,2,0)
> x + y
[1]  5 12  5  4
```

Achtung: Multiplikation/Division auch komponentenweise!!

```
> x * y
[1]  4 20  6  0
```

Wichtig: Die meisten Operationen von 2 Vektoren werden komponentenweise durchgeführt!!

Recycling-Regel

R erlaubt auch Rechnen mit Vektoren unterschiedlicher Länge.

```
> x
[1] 1 2 3 4
> x + c(1, 2)
[1] 2 4 4 6
```

entspricht

```
> x + c(1, 2, 1, 2)
[1] 2 4 4 6
```

Fehlende Werte werden aus bestehenden „recycled“.

Recycling-Regel

Funktioniert auch wenn Vektorlängen nicht Vielfache sind, allerdings mit Warnung

```
> x + c(1, 2, 4) # x + c(1, 2, 4, 1)
[1] 2 4 7 5

Warning message:
In x + c(1, 2, 4) :
longer object length is not a multiple of shorter object length
```

Zugriff auf Vektorelemente

1. Vektor von positiven Zahlen

```
> letters[1:3]
[1] "a" "b" "c"
> letters[ c(2,4,6) ]
[1] "b" "d" "f"
```

2. Logischer Vektor

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x[ (x>5) ]
[1]  6  7  8  9 10
> x[ (x%%2==0) ]
[1]  2  4  6  8 10
```


Zugriff auf Vektorelemente

3. Vektor von negativen Zahlen

```
> x <- 1:10
> x[-(1:5)]
[1]  6  7  8  9 10
```

4. Vektor von Zeichenketten

   Die Elemente eines Vektors kann man mit Namen versehen. Mittels dieses Namens kann auf die Elemente zugegriffen werden.

```
> x <- c(Wasser=1, Saft=2, Limonade=3 )
> names(x)
[1] "Wasser"   "Saft"     "Limonade"
> x["Saft"]
Saft
   2
```


Zugriff auf Vektorelemente

5. Leerer Index. Diesen haben wir bereits ständig verwendet!

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


Faktoren

Nominale oder ordinale Daten werden in R „Faktoren“ genannt. Intern ist dies ein ganzzahliger Vektor, wo jeder Zahl ein „Label“ zugeordnet ist.

```
> x <- factor( c("Saft", "Saft", "Limonade", "Saft", "Wasser") )
> x
[1] Saft     Saft     Limonade Saft     Wasser
Levels: Limonade Saft Wasser
```

Die einzelnen Stufen werden dabei gemäß der lexikographischen Ordnung angelegt:

Limonade $<$ Saft $<$ Wasser

Den einzelnen Stufen werden Zahlenwerte zugeordnet, wie der `str` Befehl zeigt:

```
> str(x)
 Factor w/ 3 levels "Limonade","Saft",..: 2 2 1 2 3
```

Faktoren

Soll die Zuordnung anders geschehen, so kann das durch das Argument `levels` erfolgen:

```
> x <- factor( c("Saft", "Saft", "Limonade", "Saft", "Wasser"),
+   levels=c("Saft", "Wasser", "Limonade") )
> x
[1] Saft     Saft     Limonade Saft     Wasser
Levels: Saft Wasser Limonade
> str(x)
 Factor w/ 3 levels "Saft","Wasser",..: 1 1 3 1 2
> levels(x)
[1] "Saft"     "Wasser"   "Limonade"
```

Möchte man ein spezielles Level als erstes verwenden, so geht das mittels `relevel`

```
> x <- relevel(x, "Wasser")
> str(x)
 Factor w/ 3 levels "Wasser","Saft",..: 2 2 3 2 1
```


Faktoren

Ein Vektor kann in einen Faktor mittels `as.factor` umgewandelt werden:

```
> x <- c("Apfel","Birne","Apfel","Traube","Traube","Kiwi")
> x <- as.factor(x)
> x
[1] Apfel  Birne  Apfel  Traube Traube Kiwi
Levels: Apfel Birne Kiwi Traube
```


Sequenzen

Der Befehl `seq()`

- Absteigende Sequenz mit gleicher Schrittweite

```
> seq(from=3, to=-2, by=-0.5)
 [1]  3.0  2.5  2.0  1.5  1.0  0.5  0.0 -0.5 -1.0 -1.5 -2.0
```

- Standardschrittweite ist $+1$ oder -1:

```
> seq(from=2, to=4)
[1] 2 3 4
> seq(from=4, to=2)
[1] 4 3 2
> 2:4
[1] 2 3 4
```


Sequenzen

- Sequenzen mit vorgegebener Länge

```
> seq(to=10, length=10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(from=10, length=10)
 [1] 10 11 12 13 14 15 16 17 18 19
> seq(from=10, length=10, by=0.1)
 [1] 10.0 10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8 10.9
```

Wiederholungen

Der Befehl `rep()`

- n-malige Wiederholung eines Objekts

```
> rep(3.5, times = 10)
 [1] 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5
> rep(1:4, times = 2)
[1] 1 2 3 4 1 2 3 4
> (anz <- seq(from = 2, to = 8, by = 2))
[1] 2 4 6 8
> rep(1:4, times = anz)
 [1] 1 1 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4 4
> rep(3.5, 10)
 [1] 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5
```

- Jedes Vektorelement wird mehrmals hintereinander wiederholt:

```
> rep(1:4, each = 2)
[1] 1 1 2 2 3 3 4 4
> rep(1:4, each = 2, times = 3)
 [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```


Aufgaben

1. Erstellen Sie einen Vektor `ung`, welcher die ersten 10 ungeraden Zahlen enthält die größer als 100 sind!

2. Geben Sie vom Vektor `ung` diejenigen Zahlen aus, die durch 3 teilbar sind und bilden sie die Summe daraus!

3. Erstellen Sie einen Faktor `faecher`, der 24 mal das Wort `Statistik`, 1 mal `Informatik` und 5 mal `Mathematik` enthält. Statistik, soll dabei an erster Stelle der Faktorlevels stehen.

Matrizen

Matrix ist ein spezieller Vektor in R!!!
⇒ ein Vektor mit Dimensionen

Erzeugen einer Matrix:

```
> x <- matrix( nrow = 4, ncol = 2,
+   data = c(1, 2, 3, 4, 5, 6, 7, 8) )
> x
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

Auf ein einzelnes Element kann mittels der Notation `x[i,j]` zugegriffen werden (i ist die Zeile und j ist die Spalte):

```
> x[3, 2]
[1] 7
```


Matrizen

Was machen die Argumente in der oben angegebenen Funktion?

`?matrix`

Matrizen

Wir können nun bestimmte *Eigenschaften* der Matrix abfragen:

```
> dim(x)
[1] 4 2
> nrow(x)
[1] 4
> ncol(x)
[1] 2
```

D.h., neben den Elementen selbst werden zusätzliche, abfragbare Eigenschaften im *Objekt* Matrix abgelegt (Dimension, Anzahl der Zeilen, Anzahl der Spalten).

Matrizen

Der Hilfe zum Befehl `matrix` entnimmt man, dass man durch das Argument `byrow = TRUE` eine zeilenweise Belegung der Matrix mit den Daten erreicht

```
> x <- matrix( nrow = 4, ncol = 2,
+   data = 1:8, byrow=TRUE )
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
```


Matrizen

Spaltenweise Vektoren und Matrizen verbinden mit `cbind`

```
> y <- c(12, 3, 4, 1)
> cbind(x, y)
          y
[1,] 1 2 12
[2,] 3 4  3
[3,] 5 6  4
[4,] 7 8  1
> cbind(y, y)
      y  y
[1,] 12 12
[2,]  3  3
[3,]  4  4
[4,]  1  1
```


Matrizen

Zeilenweise Vektoren und Matrizen verbinden mit `rbind`

```
> rbind(c(100, 0), x)
     [,1] [,2]
[1,]  100    0
[2,]    1    2
[3,]    3    4
[4,]    5    6
[5,]    7    8
> rbind(x, x)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    1    2
[6,]    3    4
[7,]    5    6
[8,]    7    8
```

Listen

Listen können beliebige Objekte enthalten, auch Objekte verschiedenen Typs. Beispielsweise können Listen als Objekte Matrizen enthalten:

```
> x1 <- matrix(nrow = 2, ncol = 2, data = 1:4, byrow = TRUE)
> x2 <- matrix(nrow = 2, ncol = 2, data = 5:8, byrow = TRUE)
> matlist <- list(matrix1=x1, matrix2=x2)
> matlist[[1]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> matlist[[2]]
     [,1] [,2]
[1,]    5    6
[2,]    7    8
```


Listen

Beispiel einer Liste, die verschiedene Objekte enthält:

```
> l1 <- list( c("Wasser", "Saft", "Limonade"), rep(1:4, each = 2),
+   matrix(data = 5:8, nrow = 2, ncol = 2, byrow=TRUE) )
> l1
[[1]]
[1] "Wasser"   "Saft"     "Limonade"

[[2]]
[1] 1 1 2 2 3 3 4 4

[[3]]
     [,1] [,2]
[1,]    5    6
[2,]    7    8
```


Zugriff auf Listenelemente

Der Zugriff auf die Elemente einer Liste sollte über den `[[]]` Operator erfolgen.

```
> l1[[1]][2]
[1] "Saft"
```

Sind die Element mit Namen versehen, kann man auch über `$` darauf zugreifen.

```
> matlist$matrix1
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```


Unterschied [] und [[]]

- x[1] liefert das Objekt an der ersten Stelle vom selben Datentyp wie x zurück
- x[[1]] liefert das Objekt an der ersten Stelle mit dessen Datentyp zurück
- x[z] liefert `NA` zurück, wenn z > `length(x)`
- x[[z]] gibt Fehler aus, wenn z > `length(x)`

Datenmatrix

Die wohl wichtigste Struktur zur Haltung von Daten im üblichen Rechteckschema, wo die Beobachtungen in den Zeilen und die Variablen in den Spalten dargestellt werden, ist die Datenmatrix. In R wird diese `data.frame` genannt.

Data frames sind spezielle Listen, deren Elemente wiederum Vektoren gleicher Länge sind. Data frames sind **DIE** typische Datenstruktur in R. Data frames können komplette Datensätze aufnehmen, die (meist) mit anderen Programmen erstellt wurden (Spreadsheet– Dateien, SPSS–Dateien, tab–delimited ASCII Dateien, etc.).

Datenmatrix

Beispiel *mtcars* (10 Kennzahlen zu 32 Autos im Jahr 1974)

```
> mtcars

                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
.........
```


Datenmatrix

Man kann die Zeilennamen und Variablennamen bekommen mit

```
> rownames(mtcars)
 [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
 [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
 [7] "Duster 360"          "Merc 240D"           "Merc 230"
[10] "Merc 280"            "Merc 280C"           "Merc 450SE"
[13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
[16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
[19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
[22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
[25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
[28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
[31] "Maserati Bora"       "Volvo 142E"
> colnames(mtcars)
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

Achtung, die Zeilennamen sind keine eigene Variable!

Zugriff auf Elemente

- Man kann es wie eine Matrix behandeln, also über []

```
> mtcars[1:4,]
                 mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4       21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag   21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710      22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive  21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
```

- Einzelne Variablen auch über $ wie bei Listen

```
> mtcars$cyl
 [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```