



TECHNISCHE
UNIVERSITÄT
WIEN
VIENNA
UNIVERSITY OF
TECHNOLOGY

Einführung in R

Bettina Grün und Matthias Templ

Oktober 2003

Wie komme ich zu R

- Die allgemeine Informationswebseite ist <http://www.r-project.org/>
(virtueller Name für <http://www.ci.tuwien.ac.at/R-project/>).
- CRAN - the Comprehensive R Archive Network
 - Die Hauptseite ist <http://cran.r-project.org/>
(virtueller Name für <http://www.ci.tuwien.ac.at/R/>).
 - Mirror Seiten sind verfügbar in vielen Ländern,
z.B. <http://cran.us.r-project.org/>.
- Auf den CRAN Seiten gibt es Binary Distributionen für Windows 95, 98, ME, NT4 und 2000 auf Intel, für Macintosh (System 8.6 bis 9.1 und MacOS X), und für mehrere Linux Distributionen.

Allgemeines

R ist ein Software-Paket für Statistical Computing. Es ist die Open Source Implementierung der Sprache S, die von John Chambers und Kollegen in den Bell Laboratories entwickelt worden ist.

Eine weitere, jedoch kommerzielle Implementierung von S ist S-PLUS.

- R ist ursprünglich von Ross Ihaka und Robert Gentleman an der Universität von Auckland entwickelt worden.
- derzeit Wartung und Weiterentwicklung durch eine weltweite Gruppe von Freiwilligen aus Forschung und Wirtschaft
- primäre Verbreitung über Webseiten (www.r-project.org) und Archive (cran.r-project.org).
- R kann über Packages erweitert werden. Einige Packages sind in der R Distribution enthalten, weitere befinden sich auf CRAN.

Installieren von R

Auf Windows:

- Lade den Installer `rw1080.exe` von <http://cran.r-project.org/bin/windows/base/> herunter und führe die Datei aus
- Setzen des persönlichen Arbeitsverzeichnis:
 - rechte Maustaste bei Start-Icon: Eigenschaften → Ausführen in
 - Setzen im GUI über Menü: File → Change dir ...
 - Setzen über die Command Line mit `setwd()`

Kontrolle mit `getwd()`

Installieren von R

- Starten von R:
 - über den Start-Icon
 - über die Datei `Rgui.exe`

Auf den Terminals:

- Setzen des persönlichen Arbeitsverzeichnis:
 - Starten von R im gewünschten Verzeichnis
 - Setzen über die Command Line mit `setwd()`

Kontrolle mit `getwd()`

- Starten von R:
 - Eintippen von `R[RET]` in einen Terminal

Wichtige R Funktionen

Wenn der Funktionsname und anschließend `[RET]` getippt wird, wird die Funktion angezeigt. Um eine Funktion aufzurufen, muss der Funktionsname zusammen mit der Argumentenliste innerhalb von `()` eingetippt werden, auch wenn diese leer ist. D.h. `q()[RET]` anstatt `q[RET]` ist notwendig, um R zu verlassen.

q() Beende die Session. Es wird gefragt, ob der Workspace (im File `.Rdata`) gespeichert werden soll.

help() Bekomme Hilfeseite für eine Funktion oder ein Objekt

help.start() Verwende einen Webbrowser für das Lesen der Hilfe

ls() Liste die Objekte im Workspace auf

rm() Lösche Objekte im Workspace

save.image() Speichern des Workspace-Inhalts in einem File

save() Speichern eines Objekts in einem File

load() Laden von mit R gespeicherten Daten

example() Exekutiere das Beispiel auf der Hilfeseite einer Funktion

summary() Fasse die Information über ein Objekt kurz zusammen

library() Laden von zusätzlichen (schon installierten) Packages

Informationsquellen über R

- Die Webseite <http://www.r-project.org/> und CRAN

- Die frequently asked questions (FAQ) Listen auf CRAN.

- Manuals: z.B.

- An Introduction to R

- R Data Import/Export

sind Teil der R Installation bzw. befinden sich auf CRAN. Unter Windows können diese im Menü unter `Help` → `Manuals` gefunden werden.

R Basics

Hilfe:	<code>help(topic)</code> <code>?topic</code> <code>help.search("topic")</code> <code>help.start()</code>	Zuweisung:	<code>x = 5</code> <code>x <- 5</code> <code>5 -> x</code>
Operatoren:	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> see <code>help("+")</code>	Vergleiche:	<code>==</code> , <code>!=</code> , <code>></code> , <code>>=</code> , <code><</code> , <code><=</code> see <code>help("==")</code>
Namen:	case sensitive, Buchstabe zu Beginn, kein Underscore	Kommentare:	alles nach <code>#</code>

R Basics

Elementare Datentypen:

Logical : TRUE, FALSE, T, F

Integer : 1, 100, 326, ...

Double : 1.0, 3.1415297, 2.718282, NaN, Inf, -Inf

Complex : 1+0i, 1i, 3+5i

Character : "Hello", "How are you?"

Missing Values : NA

Methoden zur Konstruktion

- Die Funktionen zur Konstruktion haben dieselben Namen wie die entsprechenden Datentypen. Umwandlung kann mithilfe von `as.type` erfolgen, Checken des Typs mithilfe von `is.type`.

```
> x <- matrix(1, nrow = 5, ncol = 2)
```

```
> is.matrix(x)
```

```
[1] TRUE
```

```
> as.vector(x)
```

```
[1] 1 1 1 1 1 1 1 1 1 1
```

```
> x <- list(a = "Hello", b = 1:10, pi = 3.1415927)
```

- Konstruktion einfacher Vektoren:

```
> c(1, 2, 7)
```

```
[1] 1 2 7
```

```
> c("Hello", "World")
```

```
[1] "Hello" "World"
```

R Basics

Zusammengesetzte Datentypen:

vector : von Elementen desselben elementaren Typs

array : Vektor mit Dimensionsattribut (beliebige Anzahl an Dimensionen erlaubt).

matrix : 2-dimensionaler Array. Spezialfall, um die üblichen Matrixoperationen wie z.B. Matrixmultiplikation durchführen zu können.

factor : Spezieller Vektor für das Kodieren von Klassen.

list : Liste von Elementen von verschiedenen Datentypen (sowohl elementar als auch zusammengesetzt)

data.frame : „Mischung“ aus Matrix und Liste

Zugreifen auf Elemente

Vektoren: Elemente können über Nummern oder Namen angesprochen werden

```
> x <- c(5, 3, 7)
```

```
> names(x) <- c("apple", "banana", "orange")
```

```
> x["apple"]
```

```
apple
```

```
5
```

```
> x[1:2]
```

```
apple banana
```

```
5 3
```

```
> length(x)
```

```
[1] 3
```

Zugreifen auf Elemente

Arrays: analog wie bei Vektoren, wobei die Dimensionen durch Kommas unterteilt werden

```
> x <- matrix(1:8, ncol = 2)
> colnames(x) <- c("First", "Second")
> x[1:2, ]
      First Second
[1,]     1      5
[2,]     2      6
> x[, "Second"]
[1] 5 6 7 8
> x[-3, ]
      First Second
[1,]     1      5
[2,]     2      6
[3,]     4      8
> dim(x)
[1] 4 2
```

Data Frames

- Können betrachtet werden als
 - Matrix, wo jede Spalte einen unterschiedlichen Datentyp besitzt
 - Liste, wo jedes Element ein Vektor derselben Länge ist
- Zugriff auf die Elemente analog wie bei Matrizen ODER Listen
- Beobachtungen mit Missing Values können mithilfe von `na.omit()` entfernt werden, wodurch die gesamte Zeile gelöscht wird

Zugreifen auf Elemente

Listen: Dollar Zeichen \$ oder Nummer in eckigen Klammern

```
> x <- list(a = "Hello", b = 1:10, pi = 3.1415927)
> x$a
[1] "Hello"
> x[["a"]]
[1] "Hello"
> x[[1]]
[1] "Hello"
> x[1]
$a
[1] "Hello"
> length(x)
[1] 3
```

Funktionen

```
> moment <- function(x, n = 2) {
+   sum(x^n)/length(x)
+ }
```

- Es wird nur ein Wert zurückgegeben (letztes Statement oder Argument von `return`). Wenn mehrere Werte zurückgegeben werden sollen, dann können diese in einer Liste zusammengefasst werden.
- Argumente mit Namen: Bei Aufruf einer Funktion können die Argumente in einer beliebigen Reihenfolge übergeben werden (z.B. `moment(n=3, x=x)`).
- Default Werte: Argumente mit Default Werten können beim Aufruf der Funktion weggelassen werden.

Einlesen von Daten

In R gibt es die Möglichkeit, die Daten aus den verschiedensten Formaten einzulesen. Eine nähere Beschreibung befindet sich in *R Data Import/Export*.

Die Befehle, die wir benötigen, sind:

- `read.table()`: Einlesen von Daten in Tabellen-Format aus einer Datei und Erzeugen eines Data Frames
- `load()`: Laden von Daten (und auch Objekten), die in R mit `save` gespeichert worden sind

Erstellen von Skripten

Wenn nur die Ergebnisse am Bildschirm mitverfolgt werden sollen:

```
> source("skript.R", print.eval = TRUE)
```

Mithilfe von `sink` kann die Ausgabe vom Bildschirm in eine Datei umgeleitet werden:

```
> sink("sink.txt")
> source("skript.R", echo = TRUE)
> sink()
```

Erstellen von Skripten

Um Reproduzierbarkeit der R Sessions zu erreichen, können die einzelnen Befehle in ein File geschrieben werden und daraus nach R geladen werden.

Verwende einen externen Editor zum Schreiben eines Skripts:
Notepad, Emacs, ...

Einlesen des Skripts in R:

```
> source("skript.R")
```

Wenn die Befehle und Ergebnisse am Bildschirm mitverfolgt werden sollen:

```
> source("skript.R", echo = TRUE)
```

Erstellen von Grafiken

Befehle zum Erzeugen von Grafiken sind:

- `plot()`: verwendet die Default Printmethode für Datentyp
- `hist()`: Histogramm
- `boxplot()`: Boxplot
- ...

Speichern oder Drucken der Grafiken

Windows :

- Klicken mit rechter Maustaste auf den Plot oder unter Menü File: Wähle Save as Metafile ... oder Print ...
- Vor dem Erstellen der Grafik `win.metafile(file = "Rplots.wmf")` und danach `dev.off()`

Terminals :

- `dev.print()`: Drucke aktives Device
- Vor dem Erstellen der Grafik `postscript(file = "Rplots.ps")` und danach `dev.off()`

Weitere Möglichkeiten zur Abspeicherung von Grafiken können mit `?Devices` ermittelt werden.

Beispiel

```
> dim(faithful)
```

```
[1] 272  2
```

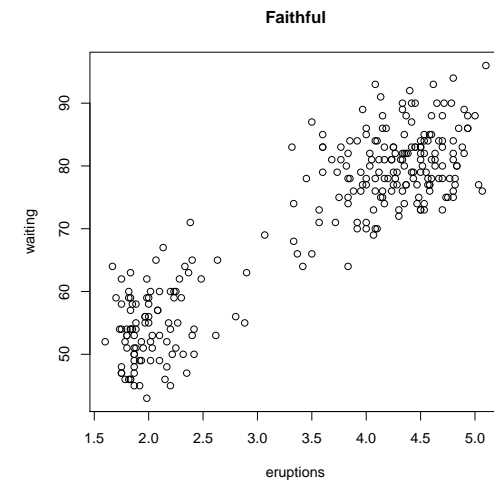
```
> faithful <- na.omit(faithful)
> plot(faithful, main = "Faithful")
```

Beispiel

```
> data(faithful)
> help(faithful)
> save(faithful, file = "faithful.Rdata")
> load("faithful.Rdata")
> write.table(faithful, file = "faithful.dat", quote = FALSE,
+   row.names = FALSE)
> faithful <- read.table("faithful.dat", header = TRUE)
> summary(faithful)
```

eruptions	waiting
Min. :1.600	Min. :43.0
1st Qu.:2.163	1st Qu.:58.0
Median :4.000	Median :76.0
Mean :3.488	Mean :70.9
3rd Qu.:4.454	3rd Qu.:82.0
Max. :5.100	Max. :96.0

Beispiel



Beispiel

Speichern der Grafik in einer Datei innerhalb eines Skripts:

Windows :

```
> win.metafile(file = "faithful.wmf")
> plot(faithful, main = "Faithful")
> dev.off()
```

Terminals :

```
> postscript(file = "faithful.eps")
> plot(faithful, main = "Faithful")
> dev.off()
```

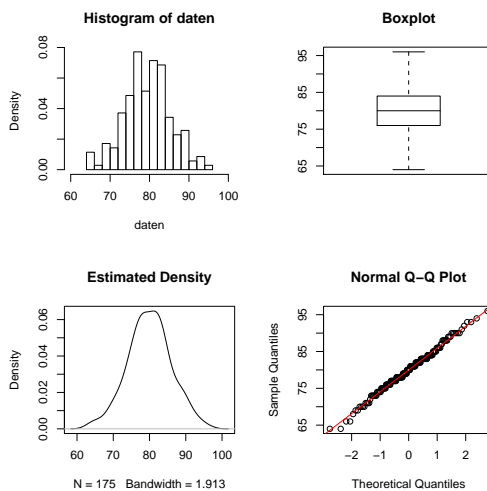
Beispiel

```
> daten <- faithful[faithful$eruptions > 3, 2]
> length(daten)
```

```
[1] 175
```

```
> par(mfrow = c(2, 2))
> hist(daten, freq = FALSE, breaks = 20, xlim = c(60,
+ 100))
> boxplot(daten, main = "Boxplot")
> plot(density(daten), main = "Estimated Density")
> qqnorm(daten)
> qqline(daten, col = "red")
```

Beispiel



Beispiel

```
> var(daten)
```

```
[1] 35.9309
```

```
> sd(daten)
```

```
[1] 5.994239
```

```
> moment(daten, n = 2)
```

```
[1] 6433.897
```