Programmieren in R und LaTeX (FT 2019)

Dr. D. Steuer
steuer@hsu-hh.de, Tel. 2819, H1 R 1397

Rechnergestützte Statistik
Helmut-Schmidt-Universität Hamburg
Fakultät WiSo

April 2019

Struktur der Veranstaltung

- Veranstaltung prinzipiell am Rechner!
- Es ist erforderlich, sich mit
  - dem Rechner,
  - der Programmiersprache R ,
  - dem GUI RStudio,
  - dem Textsatzsystem TEXbzw. LATEX,
  - dem zugehörigen GUI texstudio und schließlich
  - dem R-Paket knitr, das alle Einzelteile zusammenführt,

  auseinanderzusetzen!
- Sprechstunde ist im Prinzip jederzeit, für ausführliche Beratung ($>$ 10min) bitte telefonisch oder per mail Termin ausmachen.
- Das Skript soll nach Möglichkeit jeweils am Donnerstag vor der Veranstaltung im Netz stehen.
- Tel 2819, steuer@hsu-hh.de

Ziel der Veranstaltung

- Sofort verwendbares Handwerkszeug für alle quantitativen Aspekte des Studiums. (R und LaTeX)
- Teil der Veranstaltung in V und Ü: Jeweils ab 11:30 Uhr Vorlesung, dann im Anschluss ab ca 13:15 Übung als Präsenzveranstaltung hier im Pool.
- Sie sollen sich in den vorgestellten Wergzeugen sicher bewegen können.
- Aufbau ist in drei Teile gegliedert: R, LaTeX und die simultane Nutzung beider Tools für Reproducible Research.
- Am Ende **unbenotete** Pflicht-Klausur am 28.6.2019.

Teil 0: Die HSU-Filebox

- Einbinden des Uni-Netzlaufwerks.
- Im Explorer „Netzlaufwerk verbinden“,
- \\filebox.unibw-hamburg.de\filebox als Adresse
- Universitätskennung zur Authentifizierung.
- Haken bei „automatisch einbinden“nicht vergessen.
- Dort abgelegte Dateien sind an allen Rechnern und auf der Stube verfügbar!

Teil 1: Die Programmiersprache R

- R ist eine Arbeitsumgebung für statistische Analyse und Grafik.
- R läßt sich sehr einfach erweitern! (packages)
- Mit `RStudio` und `knitr` (und LaTeX) erhalten Sie zusätzlich eine Softwarekomplettausstattung für die Erstellung wissenschaftlicher Arbeiten.
- Aufgabe: Installieren Sie R, RStudio, texlive (d.h. Miktex oder Mactex), texstudio auf ihrem privaten Rechner!

Warum R?

- R ist FOSS, *Freie Software*, kostenlos und open source.
- R ist Industriestandard!
- R ist plattformunabhängig, d.h. Sie nutzen weiter den Rechner und das Betriebssystem, das sie gewohnt sind, sei es Windows, MacOs oder Unix. Dasselbe gilt für `RStudio`.
- Kein Vendor-Lock-In! Im Gegensatz zu z.B. Stata, MS Office!
- Hervorragende Fähigkeiten: Immer mehr Firmen nutzen R, also bekommen Sie ein Werkzeug an die Hand, das Sie fast sicher im beruflichen Umfeld wieder sehen werden. R hat sich im universitären Bereich zur Standardsoftware entwickelt. Im industriellen Bereich ebenfalls überragende Bedeutung. (Survey of Datamining Tools 2013)
- Am 7.1.2009 der Durchbruch: R auf der Titelseite der NYT!
- *lingua franca* der Statistik!

Warum R?

- Hervorragende eingebaute Hilfefunktion!
- Lokalisiert in etlichen Sprachen.
- Professioneller (oder besser) Support über Mailinglisten!
- Professionelle (oder besser) Qualitätskontrolle der Software ('make check'). Validierung der Software und der Rechenergebnisse während der ganzen Entwicklung.
- Sehr gute Handbücher werden mitinstalliert (Reference Manual $>$ 3500 Seiten).
- Für Bachelor-, Master- oder Doktorarbeiten: sehr gute Integration mit LaTeX und OpenOffice. (MS Office ist auch ok.)
- Hervorragend geeignet für *Reproducible Research*.

Benutzerinterfaces für R

- Im Kern ist R ein Interpreter mit *read-eval-loop*, der über die Kommandzeile bedient wird!
- Empfehlenswert: Interface zu einem externen Editor (emacs mit ESS(!), winedt, etc.) oder GUI.
- Hier nutzen wir RStudio. Es gibt andere GUIs, diese werden in der Vorlesung nicht behandelt (Tinn-R, Revolution, Jaguar, rkward, StatEt).
- Batch mode (skriptgesteuert).
- etwas ausgefallener: R als Modul des Webservers oder als shared library aus anderen Programmiersprachen aufrufen (python, perl).

Kurzer Abriss der Geschichte

- R ist eine Implementierung des Sprachstandards **S**.
- **S** wurde seit Mitte der siebziger Jahren von Becker und Chambers bei ATT entwickelt (Version 1.0 Juli 1977).
- Ziel war, die damals neue explorative Datenanalyse im Rechner nutzen zu können.
- R Entstand ca. 1995 als Reimplementierung des Standards S auf Macintosh Rechnern in Auckland (*R*oss Ihaka und *R*obert Gentleman).
- Chambers erhielt 1998 den ACM Software System Award für Design und Implementierung der Sprache **S**.
- 2001 fand die erste R Konferenz statt, Gründung des Core Teams.
- Die Entwickler (Core-Team) bezeichnen R als „System und Sprache für Datenanalyse und statistische Grafik“.

Vorteile von R

- Freie Software (kostenlos und im Quelltext verfügbar)
- Betriebssystemsunabhängig!
- Industriestandard!
- Sehr flexibel im Datenhandling (kein lock-in!)
- Publikationsfähige Grafiken!
- Extrem leicht erweiterbar.

Einrichten der Arbeitsumgebung

- Einloggen
- Filebox einhängen
- RStudio starten (RStudio ist Frontend zu `R` und einigen anderen Tools.)
- Neues Projekt anlegen: Ordner `R_und_Latex` in Filebox anlegen!

Eine erste R-Sitzung

```
steuer@gaia> R

R version 3.5.3 (2019-03-11) -- "Great Truth"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-suse-linux-gnu (64-bit)

R ist freie Software und kommt OHNE JEGLICHE GARANTIE.
Sie sind eingeladen, es unter bestimmten Bedingungen weiter zu verbreiten.
Tippen Sie 'license()' or 'licence()' für Details dazu.

R ist ein Gemeinschaftsprojekt mit vielen Beitragenden.
Tippen Sie 'contributors()' für mehr Information und 'citation()',
um zu erfahren, wie R oder R packages in Publikationen zitiert werden können.

Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, oder
'help.start()' für eine HTML Browserschnittstelle zur Hilfe.
Tippen Sie 'q()', um R zu verlassen.
>
```

Zuerst:

```
> citation()
To cite R in publications use:
  R Core Team (2019). R: A language and environment for statistical
  computing. R Foundation for Statistical Computing, Vienna, Austria.
  URL http://www.R-project.org/.
A BibTeX entry for LaTeX users is
  @Manual{,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2019},
    url = {http://www.R-project.org/},
  }
We have invested a lot of time and effort in creating R, please cite it
when using it for data analysis. See also ‘citation("pkgname")’ for
citing R packages.
### Zitierung von R als Literaturstelle. R hat eine ISBN!
> contributors()
### Liste der Entwickler
```

Erste Schritte: interaktive Nutzung von R

R als Taschenrechner

```
> 3 + 4
[1] 7
> log(0)
[1] -Inf
> log(-1)
[1] NaN
Warning message:
NaNs were generated in: log(x)
> pi  #es kommt auf Groß- oder Kleinschreibunrg an
[1] 3.141593
> wert <- 3
> wert <- wert^2
> x <- .Last.value
###  '=' statt '<-' geht "neuerdings" auch, auch '->'
> ls()
[1] "wert" "x
> rm(x)
> sqrt(-1+0i)
[1] 0+1i
> q()
```

Mathematische Operatoren

Symbol	Funktion
^ oder **	Potenz
*, /, +, -	Multiplikation, Division, Addition, Subtraktion
%/%, %%	ganzzahlige bzw. modulo Division
%*%	Matrixmultiplikation

Natürlich gibt es alle üblichen mathematischen Operationen: `round()`, `sin()`, `abs()`, `sqrt()` etc. Wichtig für das Konzeptverständnis: Alle diese Operatoren sind gewöhnliche R-Funktionen:

```
> "+"(3,4)
[1] 7
```

Mathematische Operatoren

R kann auch auch (numerisch) integrieren und sogar einige Funktionen symbolisch differenzieren!

```
### Numerische Integration
> ?integrate
> integrate( function(x) x^2, 0, 1)
0.3333333 with absolute error < 3.7e-15

### Ableitungen
> D(expression(x^2), "x")
2 * x
```

Mathematische Operatoren

- Wichtig sind die Bezeichner für die speziellen Zahlen:
  - NaN : Not a Number,
  - Inf, -Inf : plus resp. minus unendlich,
  - NULL : nichts, leer,
  - TRUE, FALSE : Wahr oder falsch,
  - NA : not available, fehlender Wert, *missing value*.
- Achtung: `R` implementiert IEEE Arithmetik! Internationaler Standard.

```
> round(1.5) ; round(0.5)
[1] 2
[1] 0
```

- Achtung: `pi` ist nicht `PI`! `R` beachtet Groß- und Kleinschreibung!

Logische Operatoren

- == : als numerischer Vergleich: beide Objekt sind **identisch**,
- all.equal() testet auf numerische Gleichheit bis auf eine festgelegte Abweichung,
- `identical()` für Vergleich beliebiger Objekte,
- != : ungleich,
- <, > , <=, >= kleiner als, größer als (oder gleich),
- &, |, ! : (logisch) AND, OR, NOT .

Elementare Statistik (Statistik I)

Vielzahl eingebauter Funktionen!

- `mean()`, `var()`, `sd()`, `cor()` etc.
- `runif()`, `rnorm()` etc. Zufallszahlenerzeugung
- `fivenum()`, `range()`, `summary()`, `stem()` Tukey's numbers, Spannweite, Stem-and-leaf plot
- `boxplot()`, `pie()`, `hist()`, `barplot()` grundlegende grafische Darstellungen
- `lm()`, `t.test()` lineare Regression, t-Test

Umgang mit fehlenden Werten

- Wie behandelt man NAs? Was ist der Mittelwert von 1, 2, NA?
- `mean( c(1, 2, NA) )` ergibt `NA`
- `mean( c(1, 2, NA), na.rm=TRUE)` ergibt `1.5`
- Die wichtige Option `'na.rm'` legt fest, wie NAs in die Berechung des Mittelwertes eingehen.
- Global existiert eine Option `na.action`, die mit mittels `options() gesetzt werden kann.`

Die grundlegenden Konzepte von R

- R ist ein klassischer Interpreter, der in einer sogenannten *read-eval-print-loop* arbeitet.
- Es wird Zeile für Zeile eingelesen, jeweils bis der Interpreter das Ende eines Codeblocks erkennt. Das Einlesen kann auch direkt aus einer Datei geschehen! (siehe `source()`)
- Jede Evaluation geschieht auf genau einem sog. R Ausdruck, einer *expression* (siehe `eval(EXPR)`).
- Funktionen sind keine speziellen Sprachelemente, sondern einfache Objekte. Benutzerdefinierte Funktionen sind sehr leicht hinzuzufügen (lexical scoping!):

```
> datdobbelde <- function(x) {invisible(2*x)}
> x<-2
> datdobbelde(x)
### Keine Ausgabe!
> (datdobbelde(x)) # entspricht print(datdobbelde(x))
```

Atomare Datentypen in R

Datentyp	Beispiel
NULL	NULL
logical	FALSE
numeric	3.14
complex	3+1i
character	"Hello"
factor	Ford, GM, Mercedes

- Zahlen haben einen `mode` und einen `type`
- Man findet den Typ eines Objektes mit `is.logical()`, `is.numeric()` etc. heraus.
- Man kann in R Typumwandlung erzwingen durch `as.factor()`, `as.numeric()` etc.

Zusammengesetzte Datentypen in R

- R ist eine vektororientierte Sprache. Alle komplexeren Datentypen sind aus sogenannten generischen Vektoren zusammengesetzt.
- Der wichtigste Datentyp für Berechnungen ist der `vector`, eine spezielle, indizierte Menge von Elementen des selben Typs.
- Ein Vektor besteht immer aus Elementen eines Typs! Wenn bei der Konstruktion eines Vektors verschiedene Typen zusammengefasst werden, werden diese automatisch auf den einfachsten möglichen gemeinsamen Typen konvertiert!
- Das elementare Kommande heißt `c()` (**c**`ombine`).
- Skalare sind Vektoren der Länge 1.
- Um Elemente verschiedener Typen zusammenzufassen gibt es darüber hinaus die Listen, welche mit `list()` erzeugt werden.

Umgang mit Vektoren (Erzeugung)

```
> (x <- c(1, 3, 4.5))
[1] 1.0 3.0 4.5
> typeof(x)
[1] "double"
> (x <- c(1, x, 3))
[1] 1.0 1.0 3.0 4.5 3.0
> length(x) ### length of vector
[1] 3
> x <- c(1, 4, "Hello")
> t(x)       ### transposing (vectors / matrices)
[1,] "1" "4" "Hello"
```

Umgang mit Vektoren (Teilmengenauswahl)

```
> (x <- c(2, 3, 5, 7, 11, 13, 17, 19, 23))
[1]  2  3  5  7 11 13 17 19 23
> x[1]            ### Zugriff über den einfachen Index
[1] 2
> x[2:4]          ### mehrere Elemente auf einmal
[1] 3 5 7
> x[-(2:4)]       ### einige auslassen
[1]  2 11 13 17 19 23
> x[-c(1, 7, 9)] ## Indizes müssen nicht aufeinander folgen
[1]  3  5  7 11 13 19
> x[]             ### der komplette Vektor
[1]  2  3  5  7 11 13 17 19 23
```

Umgang mit Vektoren (bedingte Teilmengen)

```
> which(x < 10)
[1] 1 2 3 4
### Indizes, für die eine Bedingung erfüllen
> x
[1]  2  3  5  7 11 13 17 19 23
> x > 10
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> sum(x>10)
5
### Anzahl der Elemente, die eine Bedingung erfüllen
> x [ x > 10 ] # Indizieren über booleschen Vektor
[1] 11 13 17 19 23
### Äquivalent ist subset()
> subset(x, x<15)
[1] 2  3  5  7 11 13
```

Umgang mit Vektoren (spezielle Funktionen)

```
> sum(x)
[1] 100
> cumsum(x)
[1]   2   5  10  17  28  41  58  77 100
> mean(x)
[1] 11.11111
> sapply(x, sqrt)
[1] 1.414214 1.732051 2.236068 2.645751 3.316625 3.605551 4
[9] 4.795832
> prod(x)
> diff(x)
```

Erzeugung spezieller Vektoren seq(), rep()

```
> 1:5; 5:1      ### äquidistant, Distanz 1
[1] 1 2 3 4 5
[1] 5 4 3 2 1

> seq(1,4,2/3) ### äquidistant, Distanz != 1
[1] 1.000000 1.666667 2.333333 3.000000 3.666667

> seq(along=x) ### Numerierung von x
[1] 1 2 3 4 5 6 7 8 9

> rep(TRUE, 5) ### Wiederholungen
[1] TRUE TRUE TRUE TRUE TRUE

> rep(c("red","blue"),c(4,7))
[1] "red"  "red"  "red"  "red"  "blue" "blue" "blue" "blue"
[9] "blue" "blue" "blue"
```

Rechnen mit R-vektoren

```
> (x <- seq(3,7))
[1] 3 4 5 6 7

> 1+x    ### Recycling der 1
[1] 4 5 6 7 8

> 2*x    ### elementweise Berechnung
[1]  6  8 10 12 14

> x*x    ### elementweise Multiplication von Vektoren
[1]  9 16 25 36 49

> x%*%x  ### Skalarproduct, implizite Transposition!
     [,1]
[1,]  135
```

Gitter

Muss man auf einen Gitter in einem Koordinatensystem z.B. jeweils eine Funktion auswerten, so bietet sich `expand.grid()` an. Typische Anwendung in der Statistik auch in der ANOVA oder beim DOE.

```
> expand.grid(FaktorA=c("a", "b"), FaktorB=c(1, 2))
  FaktorA FaktorB
1       a       1
2       b       1
3       a       2
4       b       2
```

Übung 1 – Programmieren in R und Latex FT 2017

- Installieren Sie R und RStudio auf Ihrem Rechner.
- Probieren Sie das Hilfesystem `?sum` und `?prod`.
- Erzeugen Sie einen Vektor dreip der ersten fünf Potenzen von 3.
- Geben Sie die Länge des Vektors aus.
- Berechnen Sie die Differenzen aufeinanderfolgender Einträge von dreip.
- Erzeugen Sie einen Vektor der Zahlen von 1 bis 100.
- Berechnen Sie die Summe und das Produkt der Einträge dieses Vektors.

Matrizen und Arrays

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

Matrizen in R

```
>?matrix
#  matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
#                dimnames = NULL)
> X <- matrix( 1:100, nrow=10)
> dim(X)
> X[1,]
 [1]  1 11 21 31 41 51 61 71 81 91
> X[,7]
 [1] 61 62 63 64 65 66 67 68 69 70
> X[4,6]
[1] 54
```

Matrizen und Arrays

```
> X^2 # Achtung! Elementweise!
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1  121  441  961 1681 2601 3721 5041 6561  8281
 [2,]    4  144  484 1024 1764 2704 3844 5184 6724  8464
 [3,]    9  169  529 1089 1849 2809 3969 5329 6889  8649
 [4,]   16  196  576 1156 1936 2916 4096 5476 7056  8836
 [5,]   25  225  625 1225 2025 3025 4225 5625 7225  9025
 [6,]   36  256  676 1296 2116 3136 4356 5776 7396  9216
 [7,]   49  289  729 1369 2209 3249 4489 5929 7569  9409
 [8,]   64  324  784 1444 2304 3364 4624 6084 7744  9604
 [9,]   81  361  841 1521 2401 3481 4761 6241 7921  9801
[10,]  100  400  900 1600 2500 3600 4900 6400 8100 10000
```

```
> t(X)%*%X # Matrixmultiplikation
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
 [1,]  385   935  1485  2035  2585  3135  3685  4235  4785  5335
 [2,]  935  2485  4035  5585  7135  8685 10235 11785 13335 14885
 [3,] 1485  4035  6585  9135 11685 14235 16785 19335 21885 24435
 [4,] 2035  5585  9135 12685 16235 19785 23335 26885 30435 33985
 [5,] 2585  7135 11685 16235 20785 25335 29885 34435 38985 43535
 [6,] 3135  8685 14235 19785 25335 30885 36435 41985 47535 53085
 [7,] 3685 10235 16785 23335 29885 36435 42985 49535 56085 62635
 [8,] 4235 11785 19335 26885 34435 41985 49535 57085 64635 72185
 [9,] 4785 13335 21885 30435 38985 47535 56085 64635 73185 81735
[10,] 5335 14885 24435 33985 43535 53085 62635 72185 81735 91285
>
```

Matrizen und Arrays

```
> (B <- t(A) %*% A) ### Matrixmultiplikation
     [,1] [,2]
[1,]   84  100
[2,]  100  120
> dim(B)
[1] 2 2
> B <- rbind(B, c(1,2))  ### Zeile anhängen
> dim(B)
[1] 3 2                  ### jetzt 3 Zeilen
> B <- cbind(B, c(4,4,4)) ### Spalte anhängen
> dim(B)
[1] 3 3                  ### jetzt 3 Zeilen
> B
     [,1] [,2] [,3]
[1,]   84  100    4
[2,]  100  120    4
[3,]    1    2    4
```

Matrizen und Arrays

```
> nrow(B); ncol(B)

> rowSums(B); colSums(B)

### Jede Koordinate kann adressiert werden , wie ein Vektor
> B[1,2]          ### Elemte 1. Zeile, 2. Spalte
> B[, 2]          ### Die ganze 2. Spalte
> B[1:2, 2:3)]    ### Die Matrix aud den Elementen 1.+2. Zeile,
                  ### 3.+4. Spalte
> B[-2,]          ### Ohne die 2. Zeile
> B[ c(1,3), 2] ### 2. Spalte, 1.+3. Element

> (I3 <- diag(1,2))      ### Einheitsmatrix
>      [,1] [,2]
[1,]    1    0
[2,]    0    1

> det(B)
[1] 368
```

Matrizen und Arrays

```
> solve(B)           ### Berechnung der Inversen
      [,1]  [,2]
[1,]  1.50 -1.25
[2,] -1.25  1.05

> solve(B)%*%B
              [,1]          [,2]          [,3]
[1,]  1.000000e+00 -2.564615e-14  2.220446e-16
[2,] -3.941292e-15  1.000000e+00 -2.220446e-16
[3,] -1.942890e-15 -3.330669e-16  1.000000e+00

> solve(B, c(1,2,3)) ### B%*% x = c(1,2,3)
[1] -1.50  1.25  0.50
```

Matrizen und Arrays

```
> eigen(B)           ### Eigenwerte
$values
[1] 203.6684848   3.8638884   0.4676268

$vectors
            [,1]         [,2]       [,3]
[1,] 0.64140564  0.09209843  0.7562438
[2,] 0.76712455 -0.11337508 -0.6375846
[3,] 0.01089634  0.98927447  0.1469056
```

Gitter

Muss man auf einen Gitter in einem Koordinatensystem z.B. jeweils eine Funktion auswerten, so bietet sich `expand.grid()` an. Typische Anwendung in der Statistik auch in der ANOVA oder beim DOE.

```
> expand.grid(FaktorA=c("a", "b"), FaktorB=c(1, 2))
  FaktorA FaktorB
1       a       1
2       b       1
3       a       2
4       b       2
```

Matrizen und Arrays

```
### Arrays geben beliebig dimensionale Strukturen
### Matrizen sind ein Spezialfall

> Ar <- array(1, c(2,2)) ### 2x2 Matrix
> is.matrix(A)

> Ar <- array ( 1:6, c(2,3,4)) ### 4 2x3 Matrizen
```

Rechnen mit Matrizen

```
> X+X
> eigen(X)
> qr(X)
> ?solve # Lösen von Gleichungssystemen
> MM <- matrix(runif(9), ncol=3)
> solve(MM, c(1,1,1))  # MM * x = (1,1,1)
[1]  0.7405213  2.5615387 -0.9428870
> solve(MM) # MM^(-1)
           [,1]       [,2]       [,3]
[1,]  10.847011 -18.539994  8.4335048
[2,] -18.603636  30.304962 -9.1397866
[3,]   7.258846  -8.582219  0.3804861
> diag(1,3)
> colSums(X)
```

Vektorrecycling

```
> 6:1 - 1:3 ### der kürzere Vektor wird recycelt
[1]  5  3  1  2  0 -2

> 7:1 - 1:3 ### Nur, wenn man weiß. was man tut!
[1]  6  4  2  3  1 -1  0
Warning message:
longer object length
is not a multiple of shorter object length in: 7:1 - 1:3
```

Listen

- Im Gegensatz zu Vektoren, die nur Objekte identischen Typs beinhalten können, können Listen Objekte verschiedener Art in einem R-Objekt zusammen fassen.
- Erzeugung mit `list()`, z.B.

```
> liste <- list( Namen=c("Kevin","Doro") ,
                 Geschlecht=c("m",  "f"), c(12, 13)
```

- Namen sind optional!
- Keine Restriktionen für die einzelnen Listenelemente. Unterschiedlich lange Elemente möglich!

Listen

```
> liste <- list( Namen=c("Kevin","Doro") ,
                 Geschlecht=c("m", "f"),
                 Gewicht=c(95, 70), c(12, 13))
> liste
$Namen
[1] "Kevin" "Doro"
$Geschlecht
[1] "m" "f"
$Gewicht
[1] 95 70
[[4]]
[1] 12 13
> liste$Namen
> liste[[1]]
> liste$Geschlecht[2]
```

Dataframes

- Ein Dataframe („Datentabelle“) ist eine Liste von Vektoren gleicher Länge. (und dem Klassenattribut `data.frame`)
- **Der** Datentyp in R, um Beobachtungen in einem Objekt zusammenzufassen.
- Anschaulich ein rechteckiges Schema mit einer Beobachtung je Zeile und einer Variablen (Messgröße) je Spalte.
- Konstruktion über den `data.frame()` Befehl.
- Viele R Kommandos haben als Rückgabewert einen Dataframe, insbesondere Funktionen, die Daten einlesen!

Konstruktion und Zugriff auf Dataframes

```
> leute <- data.frame( Name=c("Kevin","Doro") ,
                Geschlecht=c("m",  "f"), Alter=c(12, 13))
> leute
   Name Ges chlecht Alter
1 Kevin           m     12
2  Doro           f     13
> rownnames(leute); colnames(leute)
> colnames(leute)[1] <- "Name"
> leute$Alter
> leute$Geschlecht == "f"
> is.data.frame(leute)
```

Zugriff auf die Daten

```
> leute[2, ]  ; leute[leute$Name=="Kevin", ]
              # Zugriff auf Zeile
> leute$Name ; leute[, 3]
              # Zugriff auf Spalte
> subset(leute, Name="Kevin")
> subset(leute, Name="Doro", Alter > 20)
              # leerer Dataframe
```

Allgemein `subset( dataframe, whichrows, whichcolumns)`.
Oft weniger zu tippen.

Explizite Schleifenkonstrukte in R

Es gibt in R drei explizite Schleifenkonstrukte, um *Codeblöcke* zu wiederholen, bis eine anzugebenden Abbruchbedingung erfüllt ist.

- Die häufigste Schleife ist die `for`-Schleife.

  ```
  #> for ( variable in menge  ) { codeblock }, z.B.
  > for ( i in 1:10 ) { print(i**2) }
  # Besteht ein Codeblock nur aus einem Befehl können
  # die {} weggelassen werden!
  > for ( i in 1:10 ) print(i**2)
  ```

  Die `for`-Schleife wird benutzt, wenn die Menge von Werten, die die sogenannten Lauf- oder Zählvariable annehmen soll, im Vorhinein bekannt ist.

Explizite Schleifenkonstrukte in R

- Eine zweite Variante ist die `while`-Schleife.

```
#> while ( bedingungerfuellt ) { codeblock }
> i <- 0 ; while( i < 11) { i <- i+1 ; print(i**2) }
```

- Eine dritte Möglichkeit bietet das `repeat` Kommando.

```
#> repeat{ codeblock }
```

- Um Schleifen explizit zu verlassen gibt es
  - `next`, um den aktuellen Durchlauf von *codeblock* abzubrechen und sofort den nächsten Durchlauf zu beginnen und
  - `break`, um die ganze Schleife zu verlassen. Offensichlich benötigt jede `repeat`-Schleife ein `break` Kommando um zu enden.
- Eine analoge `repeat`-Schleife wäre also:

```
> i <- 1 ; repeat { if (i > 10) break ;
                    print(i**2) ; i <- i+1  }
```

Die `while`- und `repeat`-Schleifen wird benutzt, wenn eine Abbruchbedingung bestimmt, wie häufig dafür ein Codeblock durchlaufen werden muss.

Übung 2

1. Erzeugen Sie einen Vektor `numbers` mit den Zahlen von 1 bis 1000!
2. Programmieren Sie eine Schleife mit der Zählvariablen `i` über die Zahlen von 2 bis `floor(sqrt(1000))` und ersetzten Sie jeweils die Einträge bei den Indizes, die Vielfache der Zählvariablen sind in `numbers` durch 0.
3. Geben Sie alle verbliebenen Zahlen > 0 in `numbers` aus! (Welche Zahlen sind das?)
4. Programmieren Sie dieselbe Schleife auch in einer `while` und einer `repeat` Variante!
5. Schreiben Sie eine Funktion, bei der die höchste Zahl im Vektor als Parameter übergeben werden kann, mit 1000 als Standardwert, und die dann dasselbe Verfahren wie in den Schritten 1-3 als Funktion implementiert.

Implizite Schleifen

- R ist durch seine Vektororientierung optimiert für implizite Schleifen. Jede Operation die auf allen Elementen eines Vektors (Liste, Matrix) in einem Rutsch wirkt, ist eine implizite Schleife. Sofern möglich, sollte man diese Möglichkeit nutzen, da in der Regel ein erheblicher Geschwindigkeitsvorteil zu erzielen ist!
- Mittels der Familie von `apply()`-Funktionen können beliebige Funktionen über implizite Schleifen angewendet werden.

```
> MM <- matrix(1:100, nrow=10)
> apply( MM, 1 , sum) # Zeilensummen
```

- Es gibt `lapply()` für Listen, `tapply()` für Tabellen, `sapply()` um Vektoren zurück zu bekommen und eine Hilfsfunktionen wie `sweep()` oder `aggregate()`

Geschwindigkeitsgewinn am Beispiel von Wertetabellen

- Wie viel schneller sind implizite Schleifen? Beispiel Multiplikationstabelle.
- Nicht im Geiste von R:

```
res <- matrix(ncol=10, nrow=10)
for (i in 1:10) for (j in 1:10) res[i,j] <- i*j
```

- The R-way (much, much faster)

```
res <- outer(1:10, 1:10, "*")
```

- Wie viel schneller? Faktor 1000!

```
> res <- matrix(ncol=1000, nrow=1000) ; system.time (
  for (i in 1:1000) for (j in 1:1000) res[i,j] <- i*j)
     User      System verstrichen
    2.542       0.009       2.564
> system.time(res <- outer(1:1000, 1:1000, "*"))
     User      System verstrichen
    0.002       0.008       0.010
```

Untersuchung vorhandener Objekte

- `str()` Struktur eines R Objektes
- `summary()` Erstes Beispiel für einen „Methodendispatch“. R versucht abhängig vom Objekt, dessen summary angeschaut wird, das Richtige(tm) zu tun!

```
> data(cars)
> summary(cars)
> summary( lm ( dist ~ speed, data=cars ) ) )
```

- Je nach Objekt führt ein und dieselbe Funktion unterschiedliche Aktionen durch!

Kleine Fallstricke

```
> a <- 3
> b <- 2.1/0.7
> a == b
[1] FALSE
```

Was passiert hier?

Kleine Fallstricke

```
> a <- 3
> b <- 2.1/0.7
> a == b
[1] FALSE
```

Was passiert hier? Als Gleitkommazahlen sind 3 und 2.1/0.7 im Rechner nicht identisch! Lösung in R: es gibt die Funktion `all.equal()`

```
> all.equal(a, b)
[1] TRUE
> ?all.equal
```

`all.equal()` überprüft die numerische Gleichheit bis auf ein ϵ
Standard: `sqrt( .Machine.double.eps )`

Kleine Fallstricke

Naiver Weise vermutet man, dass das Folgende funktioniert:

```
> a <- NA
> a == NA
```

oder

```
> a <- NaN
> a == NaN
```

Kleine Fallstricke

Naiver Weise vermutet man, dass das Folgende funktioniert:

```
> a <- NA
> a == NA

[1] NA
> a <- NaN
> a == NaN
[1] NA
```

Macht es aber nicht!
Für diese Fälle stellt R Folgendes zur Verfügung:

```
> a <- NA ; is.na(a)
> a <- NaN ; is.nan(a)
```

Aufräumen und Sichern

- Arbeiten Sie nur mit Kopien! **Nie** mit Originalen!
- Wo bin ich? `getwd()`, `setwd()`
- `ls()` Listen vorhandener Objekte, ausführlicher `ls.str()`. ,
- `rm()` Entfernen von Objekten
- `save(objekte, dateiname)` Speichern ausgewählter Objekte
- `saveimage(dateiname)` Speichern der aktuellen Sitzung
- `q("yes")` Automatische Sicherung am Sitzungsende in `.RData`.
- Tipp für quick'n'dirty: `saveimage( file.choose() )`

Externe Pakete

- In erheblichem Umfang zusätzliche Funktionalität in externen *packages* (oder *views*)
  `available.packages()` gibt eine Liste der aktuell vorhandenen Pakete
- Einfaches Einfügen in eine bestehende R Installation

  ```
  install.packages("ggplot2")
  install.packages("knitr")
  ```
- Laden in eine laufende R-Sitzung mit `library(ggplot2)` or `require(ggplot2)`
- Entfernen aus einer laufenden Sitzung
  `detach(package:ggplot2)`
- Deinstallieren aus der R-Umgebung `remove.packages()`

Eingebautes Hilfesystem

- Das cheat-sheet für R: http://cran.r-project.org/doc/contrib/Short-refcard.pdf
- `help` oder "?" sind äquivalent zu RTFM: Versuchen Sie `help(plot)` oder `?plot`.
- Wenn man das genaue Kommando nicht weiß oder `help()` nicht hilft, dann kann man `apropos()`, `find()` oder `help.search()` versuchen.
- Versteht man eine Hilfeseite nicht, dann kann man mit `example(`*command*`)` oder `demo(`*command*`)` versuchen, den Befehl und seine Nutzung am Beispiel zu lernen.

Eingebautes Hilfesystem

- `help.start()` zeigt die Dokumentation im Standard-Webbrowser an.
- Die meisten von Nutzern hinzugefügten Pakete enthalten eine sogenannte Vignette, eine kurzes Handbuch im PDF oder HTML Format. Mit dem Kommando `vignette()` kann man sich dieses anzeigen lassen.

Externe Hilfe

- Dokumentation auf CRAN: cran.r-project.org
  Sehr viel gut geschriebene Dokumentation!
  Installationshandbuch, Referenzhandbuch, Dokumentation für Datenaustausch, **FAQ** usw.
- Archive der Mailinglisten mit Suchinterface auf CRAN
  `http://cran.r-project.org/search.html`

Ultima ratio

- Selbst auf der Mailingliste r-help fragen. Unbedingt den posting guide beachten! Mehrere tausend Leser, mehr als 100 Mails am Tag. Es gibt praktisch auf jede vernünftig gestellte Frage ein fundierte Antwort.
- Bekommt man sein Problem gelöst, so sollte man sein Wissen teilen, in dem man die Antwort z.B. in das R-Wiki http://wiki.r-project.org/rwiki/doku.php einträgt.
- Es gibt auch eine eigene Gruppe auf `stackoverflow.com`: http://stackoverflow.com/questions/tagged/r. `Diese Gruppe ersetzt mehr und mehr die alte Mailliste.`

Datenein- und Ausgabe mit R

- In der Regel ist Datenaustausch mit anderen Programmen im Rahmen des Datenanalyseprozesses notwendig.
- R hat viele Möglichkeiten der Datenein- und -ausgabe implementiert. Manche davon allerdings in externen Paketen.
- Über das Paket `foreign` können beispielsweise SPSS-, SAS- oder auch Stata-Files gelesen werden.
- Excel-Files sind sicher die häufigste Datenquelle. Man kann direkt mit ihnen arbeiten, aber es gibt immer Schwierigkeiten.
- Es gibt das Paket `xlsReadWrite`. Dieses ist aber nur unter Windows verfügbar und kein Opensource!
- Das Excel-Datenformat ist nicht klar definiert!

- Wenn es unbedingt sein muss, kann man auf ein Excel-Format vor Excel 2007 gehen, um die Interoperabilität mit anderen Programmen zu verbessern.
- Zugriff über RODBC ist eine sichere Variante. Dabei wird jedes Arbeitsblatt als Tabelle einer Datenbank betrachtet.
- Dasselbe Paket bietet zusammen mit DBI einen sehr komfortablen Zugang zu fast allen aktuellen Datenbanksystemen. Es wird ein Interface zur Datenbanksprache SQL (*structured query language*) implementiert.
- Entweder Datenbanken oder CSV (*comma separated values*, Textfiles(!)).
- Für unstrukturierte Dateneingaben gibt es `scan()` oder `readline()`.

Einlesen von CSV Dateien

- Ganz allgemein lassen sich Dateien, die eine Datenmatrix enthalten, mit dem Kommando `read.table()` einlesen. Das Ergebnis ist jeweils ein Dataframe.
- Es verbirgt sich eine ganze Familie von Funktionen hinter `read.table()`.
- ```
  read.table(file, header = FALSE, sep = "", quote = "\"'",
   dec = ".", numerals = c("allow.loss", "warn.loss",
   "no.loss"), row.names, col.names,
   as.is = !stringsAsFactors, na.strings = "NA",
   colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
   fill = !blank.lines.skip, strip.white = FALSE,
   blank.lines.skip = TRUE, comment.char = "#",
   allowEscapes = FALSE, flush = FALSE,
   stringsAsFactors = default.stringsAsFactors(),
   fileEncoding = "", encoding = "unknown", text,
   skipNul = FALSE)
  ```
  ```

Komfortfunktionen für CSV Dateien

- `read.csv()` bzw. `read.csv2()` haben die Defaultparameter so voreingestellt, dass z.B. mit `read.csv2()` Dateien aus dem deutschsprachigen Raum korrekt eingelesen werden.
- Es handelt sich lediglich um Aliasse von `read.table()`!
- Trick: Pfad mit `fname <- file.choose()` bestimmen, dann im Sitzungs-Logfile `read.table(file=fname, ....)` nutzen!
- Dateipfade sind vor allem unter Windows elend zu tippen.

Datenausgabe in CSV Dateien

- Wie bei der Eingabe beherrscht `R` auch bei der Ausgabe viele externe Dateiformate.
- Aus Gründen der Portabilität bevorzuge ich jedoch auch für die Ausgabe CSV Dateien! Alle Tabellenkalkulationen können diese lesen!
- Wenig überraschend lautet das Kommando zum sichern eines Dataframes in eine Datei `write.table()` (oder `write.csv()` bzw. `write.csv2()`).
- ```
  write.table(x, file = "", append = FALSE,
              quote = TRUE, sep = " ", eol = "\n",
              na = "NA", dec = ".", row.names = TRUE,
              col.names = TRUE,
              qmethod = c("escape", "double")
              fileEncoding = "")
  ```
  ```

Exkurs: Datumsinformation in R

- Daten sind extrem wichtige Datentypen!
- Aber sehr schwierig im Rechner zu handhaben: Sommerzeit, Zeitzone, Rechnerzeit, Schaltjahre, Schaltsekunden etc.
- `?DateTimeClasses` implementiert POSIX konforme Daten- und Zeitklassen in R.
- Wichtigste Funktion: `strptime()` (string to posix time).
- Standardgenauigkeit auf allen Rechner ist 1s. Auf den meisten Rechnern heute eine Auflösung im Bereich einer Mikrosekunde implementiert.

Datumsinformationen in R (Beispielsitzung)

```
dates <- c("12/15/92","12/20/95","12/25/97")
times <- c("10:01:00","06:00:00","02:30:00")
x <- paste(dates, times)
x
(z <- strptime(x, "%m/%d/%y %H:%M:%S"))
class(z)
z[2]-z[1]
as.Date(z)
ISOdate ( 2008, 10, 9, 10, 30)
format(Sys.time(), "%a %b %d %H:%M:%S %Y")
```

Es gibt eine speziellen Datentyp time-series, der Daten mit zugehörigen Zeitstempeln, also Zeitreihen, umfasst und verwaltet.

Übung 3

- Laden Sie mitttels `data(iris)` einen der in R enthaltenen Datensätze. Überprüfen Sie, ob `iris` ein Dataframe ist.
- Welche Variablen enthält `iris`?
- Probieren Sie `str()` und `summary()` an diesem Dataframe. Verstehen Sie alles Ausgaben im Detail?
- Wählen Sie alle `setosa` und weisen diese einem Dataframe mit neuem Namen zu. Speichern Sie den neuen Dataframe mittels `save()` in eine Datei.

Übung 4 - Einlesen mit `read.table`

- Lesen Sie die Daten aus `sturmfluten.csv` von der Homepage der Veranstaltung ein.
- Speichern Sie den so eingelesen Dataframe mittels `save()` und `write.table()`.
- Was genau ist der Unterschied der beiden Speichermöglichkeiten?
- Lesen Sie die Daten aus `possum.csv` von der Homepage der Veranstaltung ein.
- Speichern Sie den so eingelesen Dataframe mittels `save()` und `write.table()`.
- Erzeugen Sie eine POSIX-Zeitobjekt mit Ihrem Geburtsdatum!

Einfache R-Grafik, traditionelle Grafik

One stop for all: `plot()`
`plot()` beherrscht Methodendispatch. Je nach Argument entscheidet die Funktion, was eine sinnvolle grafische Darstellung ist. (Dataframes, Regressionen, LDAs, PCAs, Funktionen etc.)

```
> data(iris); data(cars)
> plot(cars)
> plot(lm(dist ~ speed, data=cars))
> plot(cars) ; abline(lm(dist ~ speed, data=cars))
> plot(iris)
```

Das device-Modell der R-Grafik

- Gezeichnet wird **immer** in ein sogenanntes graphics device. Am einfachsten vorzustellen wie ein Blatt Papier. Jedoch mit verschiedenen Kontexten, je nach gewünschtem Ausgabeformat.
- Das Standarddevice ist ein Fenster auf dem Monitor. Je nach Betriebssystem mit `x11()`, `win()` oder `quartz()` zu öffnen.
- Ohne weitere Angaben beginnt plot() ein neues Bild im aktuellen Grafikdevice und öffnet bei Bedarf ein Fenster.
- ?Devices gibt aus, welche Ausgabeformate in der aktuellen R Version unterstützt sind.
- Z.B. `pdf()` lenkt die folgenden Zeichenbefehle in ein PDF-Device, also eine PDF-Datei um. Die Umleitung wird mit dev.off() beendet.
- dev.off() schreibt außerdem das Dateiendesymbol in die Datei, so dass das OS weiß, dass die Datei dort zuende ist.

Das `plot()`-Kommando

- Die Hauptdokumentation zu `plot`-Optionen findet sich in `?par`.
- `plot( x, y)` Erstellt einen neuen Plot. Die wichtigsten Optionen:
  - `type = both , points, lines, no`
    z.B. `plot(1:10, 1:10*1:10, type ="b")`
    - `main = 'Überschrift', sub = 'Untertitel'`
    - `xlab = 'X- bzw.', ylab = 'Y-Achsensbeschriftung'`
    - Diese Parameter können auch nachträglich mit `title()` in einen bestehenden Plot eingefügt werden!
  - `pch = 'x'` Das Symbol, das als Markierung eines Datenpunktes gewählt wird.
  - `lty`, `lwd` Bestimmen den Linientyp und die Strickstärke von Linienzügen.
  - `col = 'red'` Farbe mit der gezeichnet werden soll. `colors()` gibt eine Liste der Farbnamen, die R kennt.
  - `axes=FALSE` Ausschalten der automatischen Achsen, lassen sich mit `axis()` und `box()` nachträglich ergänzen!

Mehr Parameter: par()

- Mit par() setzen unzähliger weiterer Parameter (Ränder, Zeichensätze ...)
- Gute Praxis:

```
> old.paramerters <- par()
# Speichern der aktuellen Grafikparameter
> par( mfrow = c(2,2 ) ; par(ask=TRUE)
# Setzen gewünschter Paramter, hier z.B. 2x2 Bilder
# und nach jedem Plot auf Tastendruck warten,
# im jeweils aktuellen Device!
> plot( lm ( dist ~ speed, data = cars  )  )
# Alles auf einem Blatt!
> par(old.parameters)
# Zurücksetzen der Grafikparameter!
```

Grafikprimitve

Es gibt eine Reihe von Hilfsfunktionen, die in vielen Plotfunktionen genutzt werden:

- `xy.coordinates()` Leitet aus verschiedenen R-Objekten x- und y-Koordinaten ab. Argumente sind Vekoren oder Dataframes.
- `rect()` Zeichnet Rechtecke.
- `polygon()` Zeichnet Polygone, auch schraffiert oder eingefärbt.
- Funktionen für die Farbauswahl: `colors()`, `palette()`, `terraincolors()`, `rainbow()`, `gray()` and more...
- Funktionsplotter `curve()`: z.B. `curve(sin , -pi, pi)`.

Ergänzung bestehender Plots

- Der erste Plot legt die Koordinaten fest. Nachfolgende Kommandos müssen die Spannweite der Grafik in X- und Y-Richtung beachten. Sonst sieht man halt nichts ...
- `points()` ergänzt Punkte,
- `lines()` ergänzt Linienzüge. Jeweils müssen die X- und Y-Koordinaten angegeben werden.
- Übrige Parameter sind wie bei `plot()` bzw. `par()`

Verzierung und Beschriftung

- `legend()` Ergänzen einer Legende
- Mathematische Symbole wir $\sum$ oder Φ in Plots! LaTeX Schreibweise!

```
> x <- seq(-pi, pi, len = 65)
> plot(x, sin(x), type = "l", col = 2,
       xlab = expression(phi),
       ylab = expression(f(phi)))
> abline(h = -1:1, v = pi/2*(-6:6), col = "gray90")
> lines(x, cos(x), col = 3, lty = 2)
> ex.cs1 <- expression(plain(sin) * phi,
                       paste("cos", phi))
> legend(-3, .9, ex.cs1, lty = 1:2, adj = c(0, 0.6))
```

Die wichtigsten Grafikdevices

Die üblichsten Ausgabeformate sind:

- pdf() / postscript() Ausgabe in eine PDF Datei
- png() Ausgabe z.B. für Webseite eine Grafikdatei
- jpeg() Ausgabe in Datei als Fotoformat.
- tikz() Ausgabe in LaTeX Datei

Beispiel:

```
> pdf(file="cumsum.pdf")
> plot(1:10, cumsum(1:10), main="Kumulierte Summe")
> dev.off() # Nicht vergessen! Sonst leeres Bild!
```

Empfohlenes Vorgehen

- Wenn die Grafiken für Publikationen gedacht sind, empfiehlt es sich, die Höhe und Breite bereits bei der Erzeugung korrekt zu setzen. Dazu dienen die Optionen `widht` und `height` bei der Erzeugung eines devices. Achtung! Standardeinheit sind `inch`!
- Mittels `dev.print()` kam man auch das aktuelle Bild auf dem Bildschirm in eine Datei speichern. Dies ist jedoch nicht empfehlenswert für Publikationen! (Reproducible Research!)

Spezielle Plotkommandos

- Histogramm: `hist()` bzw. `truehist()` aus dem Paket MASS. `truehist()` hat in der Nutzung weniger Überraschungen für den Nutzer. Die Wahl der Klassengrenzne ist insbesondere bei ganzzahligen Beobachtungen intuitiver.
- `barplot()` Darstellung von Anzahlen! Histogramm ist dafür ungeeignet!
- Boxplot: `boxplot()`
- Violinplot: `vioplot()` aus Paket vioplot.
- und unzähliges anderes ...

Klassierte Boxplots

```
> data(ToothGrowth)
> ?ToothGrowth
> boxplot( len ~ supp,  data=ToothGrowth  )
```

Violinen-Plots: boxplot+density

```
> install.packages(vioplot)
> with(ToothGrowth, vioplot(len[supp=="OJ"],
                            len[supp=="VC"]))
# see help(vioplot) for options
```

`with()` nimmt als erstes Argument einen Dataframe, so dass innerhalb des Aufrufs von `with()` die Komponenten direkt genutzt werden können. Vgl. `data=...` in z.B. `lm`.
Mit `attach()` und `detach()` können Dataframes so auch innerhalb einer R-Sitzung in den direkten Zugriff gebracht bzw. daraus entfernt werden.

3D-Grafik

3-D Grafik ist **keine** Stärke von R. Es gibt aber ein paar Funktionen, die entweder eine Projektion auf R^2 von dreidimensionalen Daten oder einen perspektivisch dargestellten dreidimensionalen Viewport darstellen.
Erweiterungen bietet die Pakete `plot3D` https://cran.r-project.org/web/packages/plot3D oder `scatterplot3d`.

- `persp()` Zeichnet eine 3D-Viewport (Würfel) mit 3D-Daten. Man wählt den Winkel und kann das gezeichnete Objekt so drehen.
- `contour()` Zeichnet eine Projektion auf eine Ebene mit Isolinien.

Übung 5 - `plot()` etc.

- Erstellen Sie einen Plot, der die Funktion $f(x) = x^2$ im Intervall $(-2, 2)$ enthält.
- Ergänzen Sie die Wurzelfunktion auf $(0, 2)$ in blau, berechnet an 100 Stützstellen im gegebenen Intervall. Zeichnen Sie die Punkte und die Linien ein.
- Ergänzen Sie eine entsprechend Legende.
- Schaffen Sie das Wurzelzeichen in die Legende zu bringen?
- Wie kann man den Bereich zwischen der Wurzelfunktion und der Parabel schraffieren bzw. färben?
- Speichern Sie die Grafik in einer PDF Datei ab.

Lösung Übung 5

```
> curve(x^2, -2, 2)
> xval <- seq(-2,2,length=101)
> lines(xval, sqrt(xval), col="blue")
> points(xval, sqrt(xval), pch=".")
> legend(-1,3, c(expression(x^2), expression(sqrt(x))),
                lty=c(1,1), col=c("black", "blue"))
> xval <- seq(0,1, 0.05)
> polygon( c(xval, rev(xval)),
           c(xval^2, sqrt(rev(xval))),
           density=10, angle=30   )
> pdf(file="curves.pdf")
#repeat plotting commands above
> dev.off()
```

Dynamische oder interaktive Grafik

Die Unterstützung für dynamische oder interaktive Grafik im Standard-R ist rudimentär. Es gibt Pakete, die versuchen diese Schwäche zu beheben. Zu nennen sind die Pakete `rgl` für bewegte Grafik und `shiny` um interaktive Webseiten mit R-Grafiken zu versehen.
Im Standard-R beschränkt sich die Interaktion auf die Funktion `locator()`, welche die Koordinaten eines Punktes in einer Grafik liefert, so dass einzelne Beobachtungen identifizierbar sind.

Bedingte Programmabläufe

- Der einfachste Fall ist eine Anweisung, die nur ausgeführt wird, wenn eine Bedingung erfüllt ist. Hierfür nutzt man das Kommando `if ( Bedingung ) Codeblock`.
  Ein Beispiel, wenn nur für positive Zahlen die Wurzelfunktion ausgewertet werden soll:
  ```
  if ( zahl > 0) print(sqrt(zahl)) .
  ```
- Oft soll je nach Wahrheitswert der Bedingung eine andere Funktion aufgerufen werden. Hierfür existiert das Schlüsselwort `else`.

```
betrag <- function(zahl) {
              if ( zahl >=0 ) { zahl }
              else {-zahl}
          }
```

Achtung: Die {} sind in diesem Falle optional.

Bedingte Programmabläufe

- `if()` arbeitet nicht auf Vektoren, sondern nutzt nur das erste Element!
- `ifelse( bedingung, wertbeitrue, wertbeifalse )` kann oft als kompakter, vektorwertiger Ersatz genutzt werden.

```
betrag <- function(zahl) {
            ifelse ( zahl >=0 , zahl, -zahl) }
```

- Für mehrwertige Verzweigungen gibt es in vielen Programmiersprachen eine Anweisung, in R heisst diese `switch(EXPR, ...)`. Hier wird EXPR ausgewertet und dann aus den restlichen Argumenten jenes mit der entsprechenden Nummer oder jenes mir dem entsprechenden Namen gewählt, je nachdem, ob EXPR sich zu einer ganzen Zahl oder einer Zeichenkettenkonstante berechnet. EXPR darf kein Vektor sein!

Bedingte Programmabläufe

```
# Beispiel fuer EXPR die Zeichenkette ergibt
centre <- function(x, type) {
        switch(type,
                mean = mean(x),
                median = median(x),
                trimmed = mean(x, trim = .1))
      }
      x <- rcauchy(10)
      centre(x, "mean")
      centre(x, "median")
      centre(x, "trimmed")

# Beispiel fuer EXPR mit ganzzahligem Wert
betrag <- function(zahl) switch( sign(zahl)+2, -zahl, 0, za
```

Übung 6 - Einlesen mit `read.table`, Daten aufbereiten!

- Lesen Sie die Daten aus `sturmfluten.csv` von der Homepage der Veranstaltung ein. (Fertig!)
- Fassen Sie alle Pegelstände in einer Spalte des Dataframes zusammen! Fügen Sie eine Spalte `Kategorie` hinzu, die als Faktorvariable die Art der Sturmflut enthält.
- Reduzieren Sie den Dataframe auf die Spalten mit Datum, Pegelstand und Kategorie.
- Fertigen Sie schließlich einen Plot der Daten an, bei dem die verschiedenen Kategorien von Sturmfluten verschieden eingefärbt sind.
- Sichern sie den Dataframe in einer Datei.
- Erzeugen Sie eine PDF-Datei mit der erstellten Grafik.

Lösung zu Übung 6

```
location <-
"http://fawn.hsu-hh.de/~steuer/downloads/FT2018/sturmfluten.csv"
fluten <- read.table(location, skip=2, header=TRUE, sep=";",as.is=TRUE)
fluten <- fluten[ 1:NROW(fluten) , 1:4  ]
fluten <- fluten[1:(which(fluten$Datum == "")[1] - 1) ,
                 1:NCOL(fluten)]
fluten$Datum <- strptime(fluten$Datum, "%d.%m.%Y")
Pegel <- rowSums(fluten[, 2:4], na.rm=TRUE)
fluten <- cbind(fluten, Pegel)
Kategorie <-  ifelse(! is.na(fluten$Sturmflut) , "normal",
  ifelse( ! is.na(fluten$schwere.Sturmflut), "schwer", "extrem"))
fluten <- cbind(fluten, Kategorie)
fluten <- fluten[, c("Datum", "Pegel", "Kategorie")]
write.table(fluten, "fluten.csv", row.names=FALSE)
pdf(file="fluten.pdf")
plot(fluten$Datum, fluten$Pegel, col=fluten$Kategorie,
main="Pegel St.Pauli",
sub="Pegelstände der Sturmfluten 1955 - 2005",
xlab="Datum", ylab="Pegel in cm")
lines(fluten$Datum, fluten$Pegel, lty = 3)
dev.off()
```

Eigene Funktionen in R

- Eigene Funktionen sind in R sehr einfach hinzuzufügen und lassen sich danach nutzen wie ein eingebaute Kommandos.
- Die Syntax einer Funktionsdefinition:

```
fname <- function( para1 [= default1],
                   [para2], ...  ){
                     R Code Block
                  }
```

- Der Wert einer Funktion ist der Wert des letzten Ausdrucks, der im R Code Block ausgeführt wurde.
- Aufruf einfach mit `fname( para1 [,..])`.

Modellspezifikationen in R: `formula()`

- In der statistischen Analyse gelangt man immer irgendwann an den Punkt, dass man ein statistisches Modell an Daten anpassen will. Beipiele sind z.B. `lm`, `glm`, `nls` etc
- Um solche Modelle kompakt zu spezifizieren gibt es den Operator $\sim$. Ein R Ausdruck der Form `y` $\sim$ `modell` beschreibt damit, dass eine Responsevariable y durch das Modell *modell* erklärt werden soll.
- *modell* selbst enthält Einflussgrößen getrennt mit '+'
- Eine Einflussgröße kann dabei einen einzelne Variable sein oder eine mit ':' oder '*' verknüpfte Liste von Einflussfaktoren.
- ':' bezeichnet eine Wechselwirkung $x_1 : x_2 = x_1 x_2$,
- '*' bezeichnet das Modell mit Haupteffekten und Wechselwirkungen: $x_1 * x_2 = x_1 + x_2 + x_1 x_2$.

Modellspezifikationen in R

- Um Modelle mit höheren Exponenten zu spezifizieren kann '^' genutzt werden. $((a+b)^2 = (a+b) * (a+b))$
- Besondere Bedeutung hat der „.“als Symbol! Im Rahmen eines `data` Arguments im Modell bedeutet es “alle noch nicht benutzten Variablen”. Im Rahmen der `update.formula()` Funktion “alle bisher in der Formel vorkommenden Terme”.
- Analog zu '+' kann '-' genutzt werden, um einzelne Terme aus einer Formel zu entfernen.
- Spezialfall Achsenabschnitt: $y \sim 0 + x$ und $y \sim x - 1$ spezifizieren beide ein Modell ohne Absolutglied.
- Werden Variablen transformiert, so kann man beispielsweise schreiben $y \sim log(x) + z$ um ein Modell an den Log von x anzupassen. Manche Transformationen erfordern eine Kapselung in `I()`, um syntaktische Eindeutigkeit zu erreichen. Möchte man z.B. y erklären mit der Summer zweier Variablen x und z, so schreibt man $Y \sim I(x + z)$ und nicht $y \sim x + z$.

Variablenselektion

- Die Modellspezifikationen müssen flexibel sein, da typischerweise eine Vielzahl von Modellen angepasst und diese miteinander verglichen werden, bis man zu einem zufriedenstellenden Ergebnis gelangt.
- Wenn die Anzahl der Einflussgrößen zunimmt, nimmt die Zahl der möglichem Modelle exponentiell(!) zu!
- Klassische Strategien sind forward- und backward-selection des passenden Modells. Forwardselection beginnt mit dem leeren Modell und nimmt immer die "signifikanteste" Variable mit ins Modell auf, Backwardselektion startet mit dem vollen Modell und streicht jeweils die unwichtigste Variable.
- Der Gütevergleich erfolgt in der Regel über sogenannte Informationsmaße (AIC, BIC, ...) oder einfach über das adjustierte R^2 und den p-Wert zu berücksichtigender Einflussgrößen.
- Gestoppt wird, wenn es zum festgelegten p-Wert keine weitere signifikante Variable gibt oder R^2 schlechter wird.

Das multiple Bestimmtheitsmaß R^2 und R_a^2

- Erinnerung: In der Regression (eine Einflußgröße) ist das Bestimmtheitsmaß R^2 definiert als

$$R^2 = 1 - \frac{SSR}{SST}.$$

- Das multiple $R^2 := 1 - \frac{SSR}{SST}$ genau, wie im Fall der einfachen linearen Regression.
- Das adjustiertes R_a^2 berücksichtigt die Anzahl der inkludierten Variablen. Dies ist sinnvoll, das jede zusätzliche Variable das R^2 erhöht. Es gilt:

$$R_a^2 = 1 - \frac{SSR/(n-p)}{SST/(n-1)} = 1 - \left(\frac{n-1}{n-p}\right)(1-R^2).$$

Modellauswahl durch *backward selection*

- Beispiel für *backward selection* aus Faraway.
- Daten: `state.x77`.
- Ziel: Modell für die Lebenserwartung aus den anderen Variablen herleiten.
- Beginnend mit dem vollen Modell, wird in jedem Schritt der Einflußfaktor entfernt, der den höchsten p-Wert größer als 0.05 hat.
- In der Praxis würde man den letzten Schritt rückgängig machen, da das R_a^2 abnimmt und die gesetzte 5% Grenze von Einflußfaktor `Population` nur sehr knapp überschritten wird.

Die einzelnen Schritte in R

```
library(DAAG)
data(state)
?state
statedata <- data.frame(state.x77, row.names=state.abb)
tmpmodel <- lm(Life.Exp ~ . , data=statedata )
summary(tmpmodel)
### größter p-Wert: Area

tmpmodel <- update(tmpmodel, . ~ . - Area)
summary(tmpmodel)
### größter p-Wert: Illiteracy

tmpmodel <- update(tmpmodel, . ~ . - Illiteracy)
summary(tmpmodel)
### größter p-Wert: Income

tmpmodel <- update(tmpmodel, . ~ . - Income)
summary(tmpmodel)
### größter p-Wert Population
```

```
tmpmodel <- update(tmpmodel, . ~ . - Population)
summary(tmpmodel)
> summary(tmpmodel)
#### das finale Modell
Call:
lm(formula = Life.Exp ~ Murder + HS.Grad + Frost, data = stateda

Residuals:
    Min      1Q  Median      3Q     Max
-1.5015 -0.5391  0.1014  0.5921  1.2268

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 71.036379   0.983262  72.246  < 2e-16 ***
Murder      -0.283065   0.036731  -7.706 8.04e-10 ***
HS.Grad      0.049949   0.015201   3.286  0.00195 **
Frost       -0.006912   0.002447  -2.824  0.00699 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.7427 on 46 degrees of freedom
Multiple R-squared: 0.7127,Adjusted R-squared: 0.6939
```

Übung 7

Das Beispiel nachvollziehen und forward selection durchspielen!

Ausblick: Variablenselektion

- Die hier vorgestellten Verfahren sind “einfach“.
- Wie bereits bemerkt kann das schrittweise (*one-at-a-time*) Vorgehen dazu führen, dass die beste Teilmenge von Einflussgrößen nicht gefunden wird.
- Das Paket `leaps` stellt einige umfassendere Funktionen für ein allgemeineres Vorgehen zur Verfügung.
- Dazu gehört unter anderem die *erschöpfende Suche*, bei der **alle** möglichen Teilmengen von Größen untersucht werden.
- Zwar findet man so die beste Teilmenge von Größen, allerdings müssen 2^p mögliche Teilmengen von Einflußgrößen $X_1, \ldots, X_p$ untersucht werden!
- Auch heute noch nicht möglich für echte Probleme mit z.B. $p >> 50$!

Das Paket *leaps*

```
> install.packages("leaps") ; library(leaps)
> reg1 <- regsubsets(Life.Exp ~ Population + Income +
  Illiteracy +  Murder + HS.Grad + Frost + Area,
  data=statedata ,  method="backward")

### backward selection with leaps
### possible methods: exhaustive, backward, forward
> summary(reg1)
> plot(reg1, scale="adjr2")
```

Exkurs: Nicht-lineare Modelle in R

- Normalerweise ist für diesen Fall die Fehlerquadratsumme als R-Funktion zu definieren und dann auf einen der eingebauten Minimierungsalgorithmen der gewählten Programmiersprache zurückzugreifen.
- In R gibt es `optim`, `uniroot`, `nls` und `nlm` für diese numerische Schätzung von Parametern.
- Besonders `nls` (*nonlinear least squares*) ist extrem praktisch:

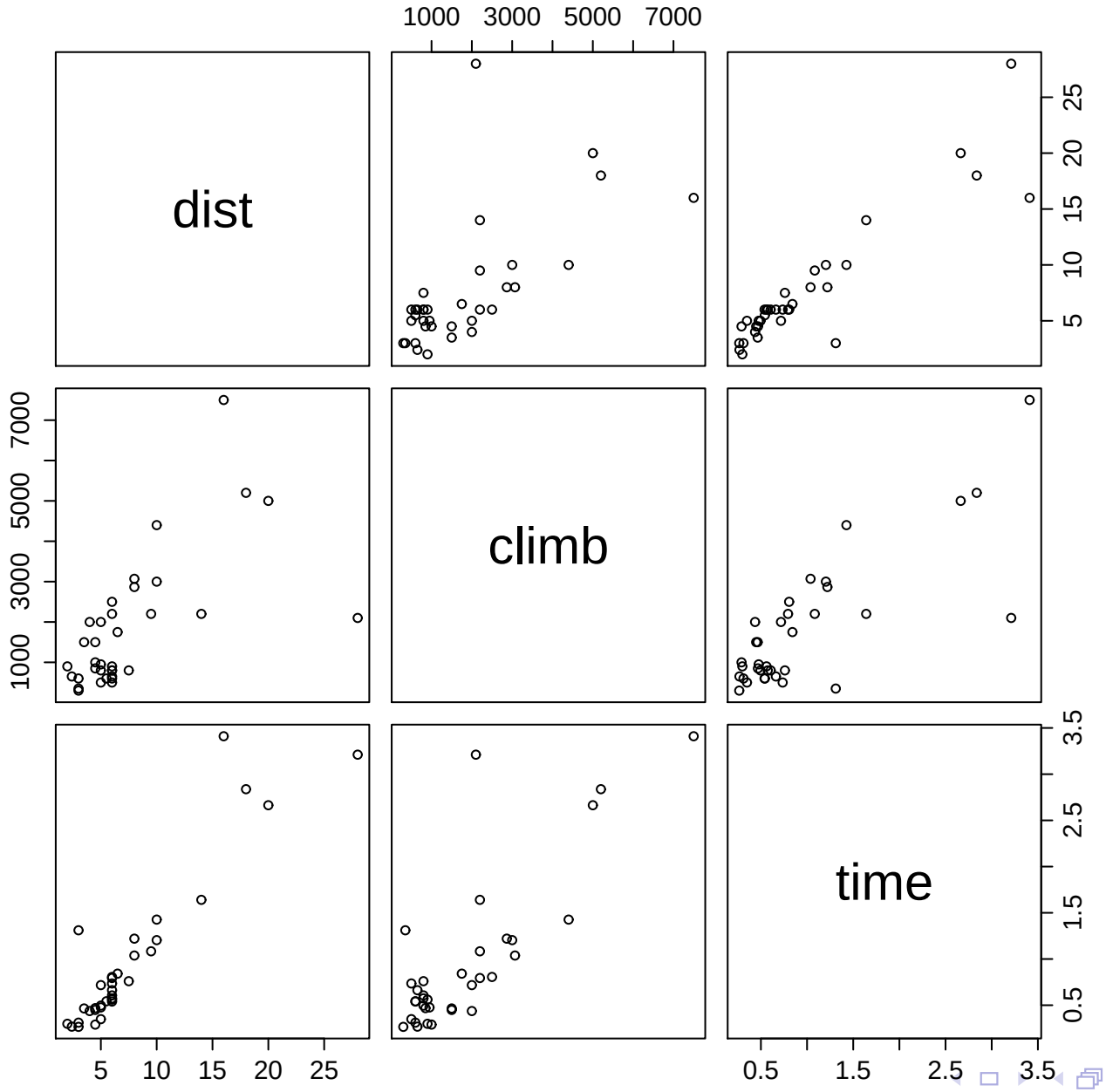
```
x <- -(1:100)/10; y <- 100 + 10 * exp(x / 2) + rnorm(x)/10
nlmod <- nls(y ~  Const + A * exp(B * x), trace=TRUE)
plot(x,y, main = "nls(*), data, true f() and fit, n=100")
curve(100 + 10 * exp(x / 2), col=4, add = TRUE)
lines(x, predict(nlmod), col=2)
```

Variablenselektion und Modellbildung am Beispiel

- Datensatz `hills` aus dem Paket `DAAG`
- Rekordzeiten für diverse schottische Bergläufe, Stand 1984
- 35 Strecken im Datensatz, jeweils Streckenlänge in Meilen (`dist`), Höhenmeter in Fuß (`climb`) und Rekordzeit in Stunden (`time`)
- Vorbereiten der Analyse:

```
> library(DAAG) ; data(hills)
> help(hills)
> hi.a <- hills
> pairs(hi.a)
```

Scatterplotmatrix zum `hills` Datensatz

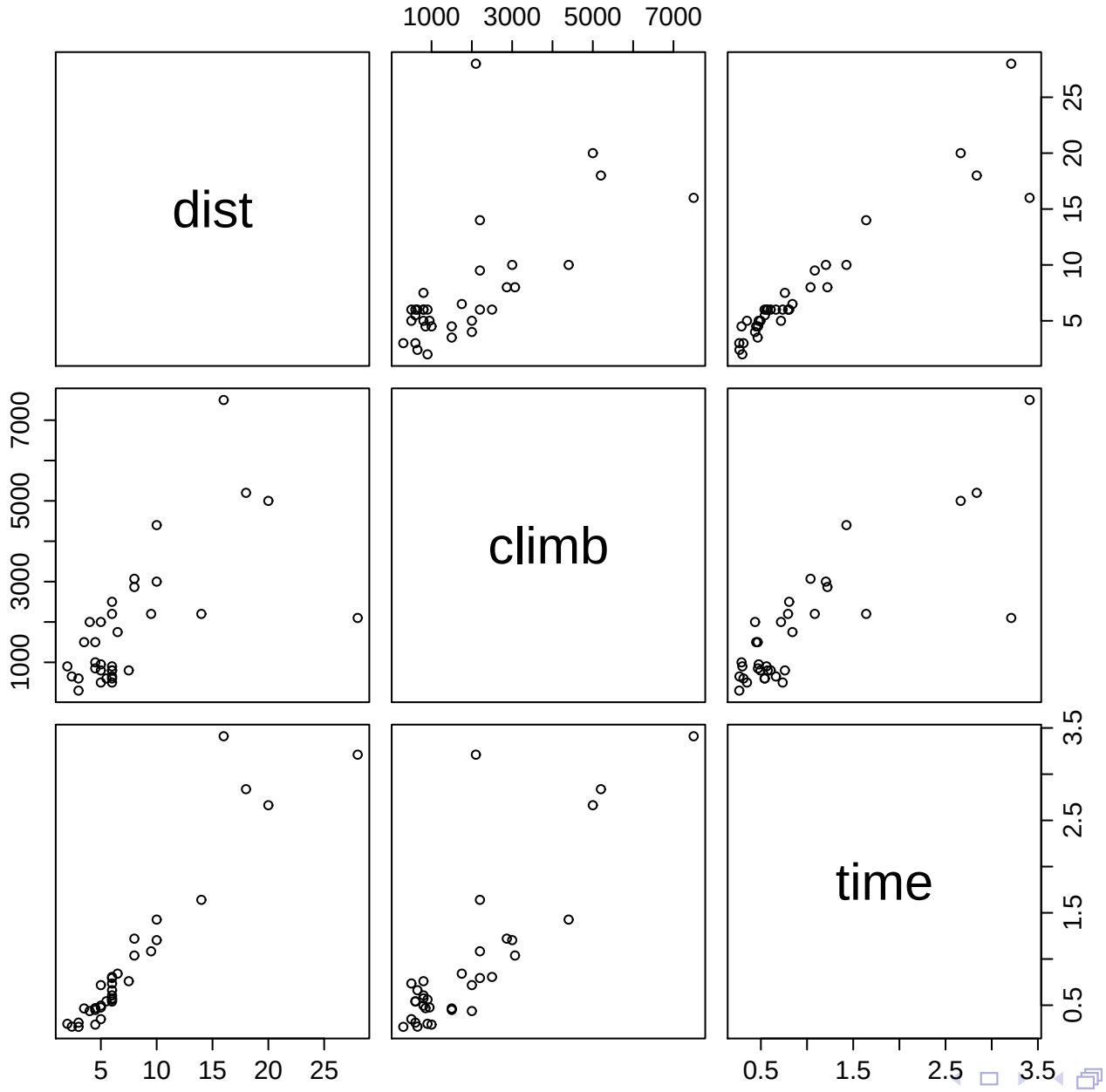


Analyse des Scatterplots

- Sind auffällige Punkte zu erkennen?
- Ja, die Beobachtung mit fast 1.5 h für 3 Meilen.
- Bebachtung finden und aus dem Analysedatensatz entfernen. (Nr. 18)

```
> hi.a
### in den Daten den Punkt suchen
> hi.a <- hi.a [-18,]
### und entfernen
> pairs(hi.a)
### Kontrolle!
```

Kontrolle des Scatterplots



Erster Modellansatz: Lineares Modell

- Inhaltliche Überlegung: Sowohl Länge als auch Höhenmeter sollten Einfluß auf die Gesamtzeit haben!
- Erste Idee: einfaches lineares Modell:
  $\text{time} = \beta_0 + \beta_1 * \text{dist} + \beta_2 * \text{climb}$
- in R:

```
> hi.a.lm <- lm(time~dist + climb , data=hi.a); summary(hi.a.lm)

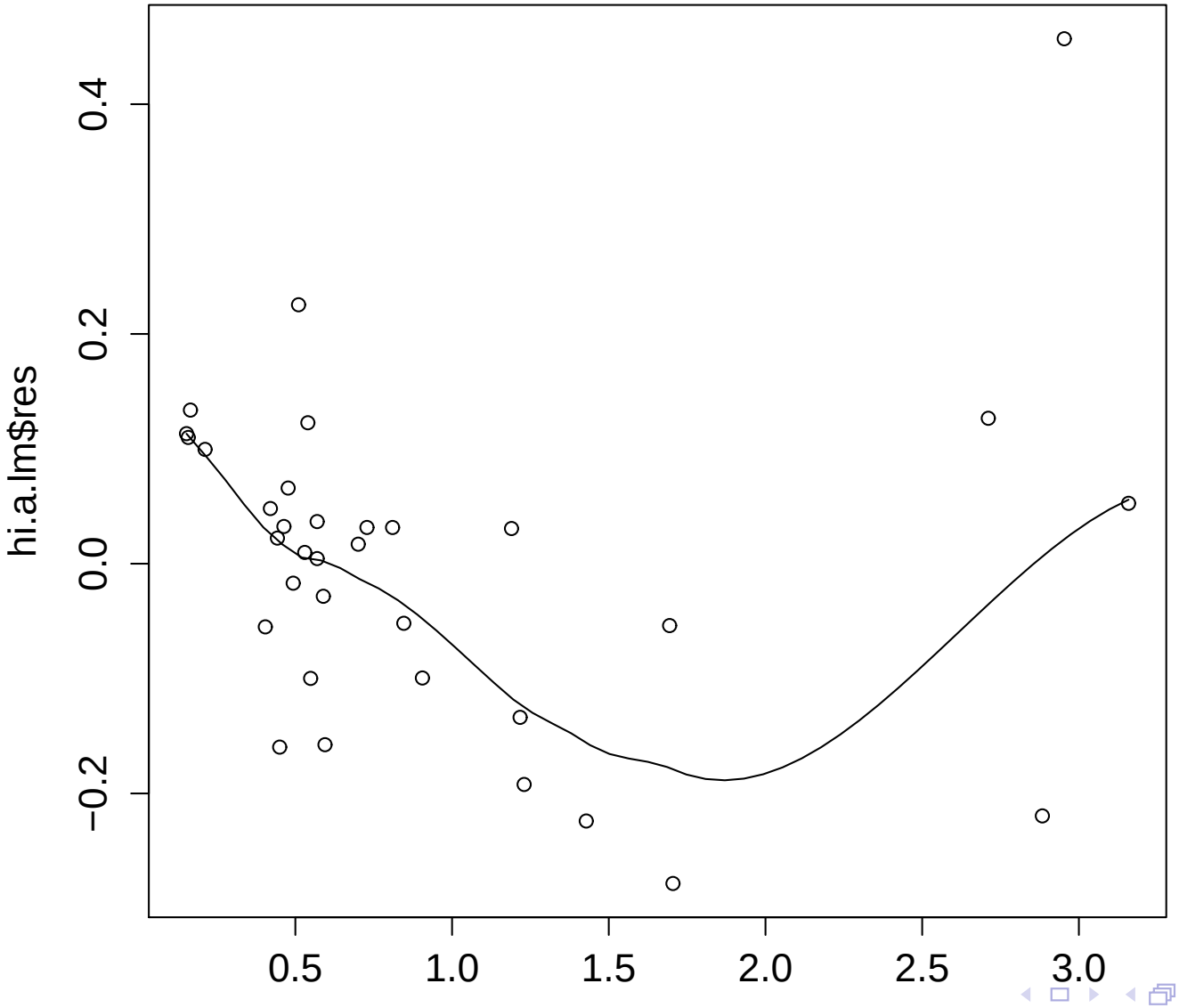
Residuals:
     Min       1Q   Median       3Q      Max
-0.27838 -0.08837  0.01962  0.06253  0.45695
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.247e-01  4.420e-02  -5.083 1.69e-05
dist         1.060e-01  6.026e-03  17.592  < 2e-16
climb        1.976e-04  2.062e-05   9.584 8.76e-11
Residual standard error: 0.147 on 31 degrees of freedom
Multiple R-squared: 0.9715,Adjusted R-squared: 0.9697
F-statistic: 529.1 on 2 and 31 DF,  p-value: < 2.2e-16
```

Interpretation des multiplen linearen Modells

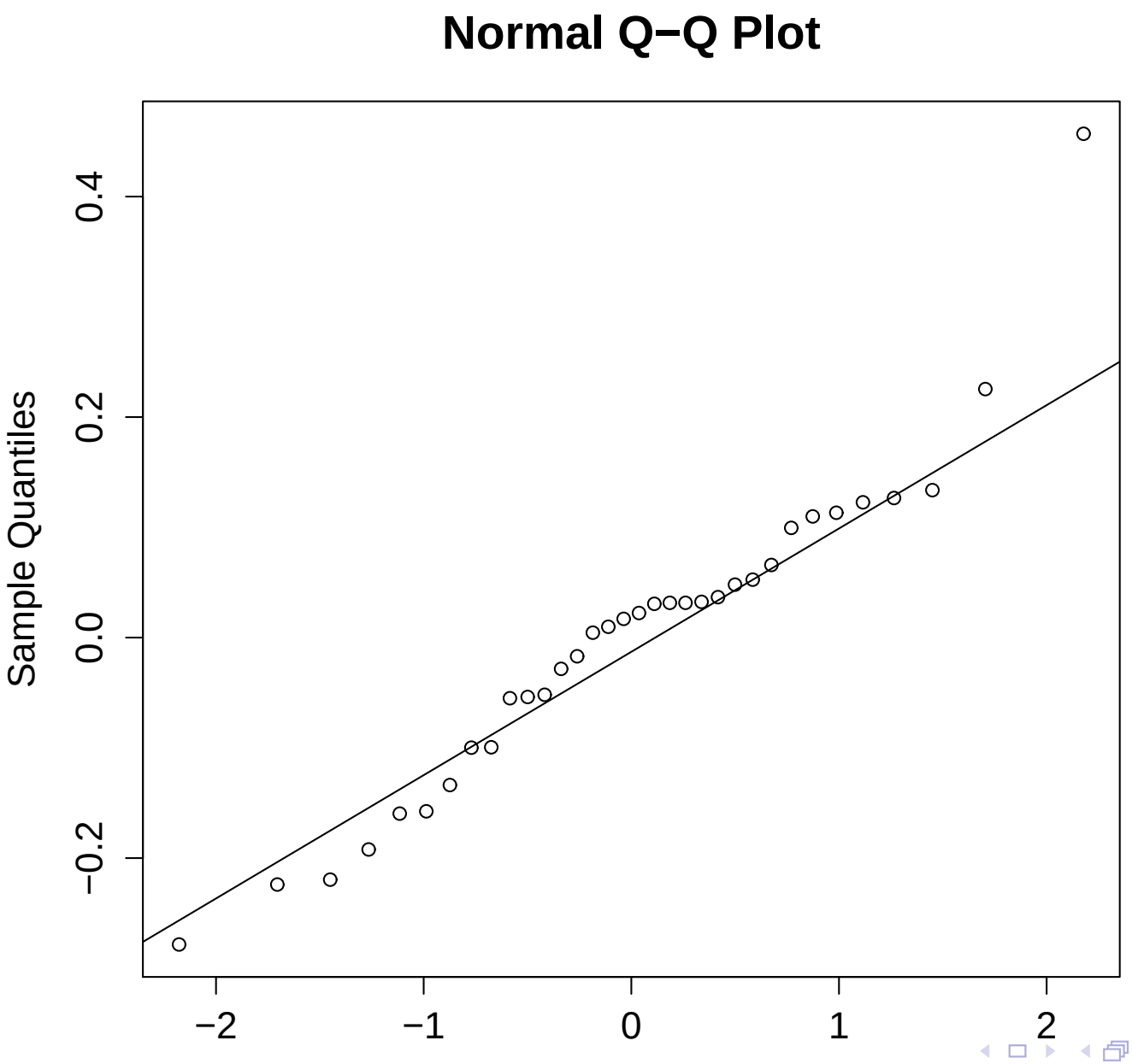
- Sehr hohes R_a^2. Dies spricht für das Modell.
- Allerdings: Die Grafik der angepassten Werte gegen die Modellfehler zeigt klar eine Struktur, genau wie der QQ-Plot der Fehler.
- in R:

```
scatter.smooth(hi.a.lm$fit , hi.a.lm$res)
qqnorm(hi.a.lm$res); qqline(hi.a.lm$res)
```

Diagnostische Plots I



Diagnostische Plots II



Modellverfeinerung

- Muss evtl eine Wechselwirkung zwischen `dist` und `climb` berücksichtigt werden?
- Neues Modell:time $= \beta_0 + \beta_1 * \text{dist} + \beta_2 * \text{climb} + \beta_3 * \text{dist:climb}$
- in R:

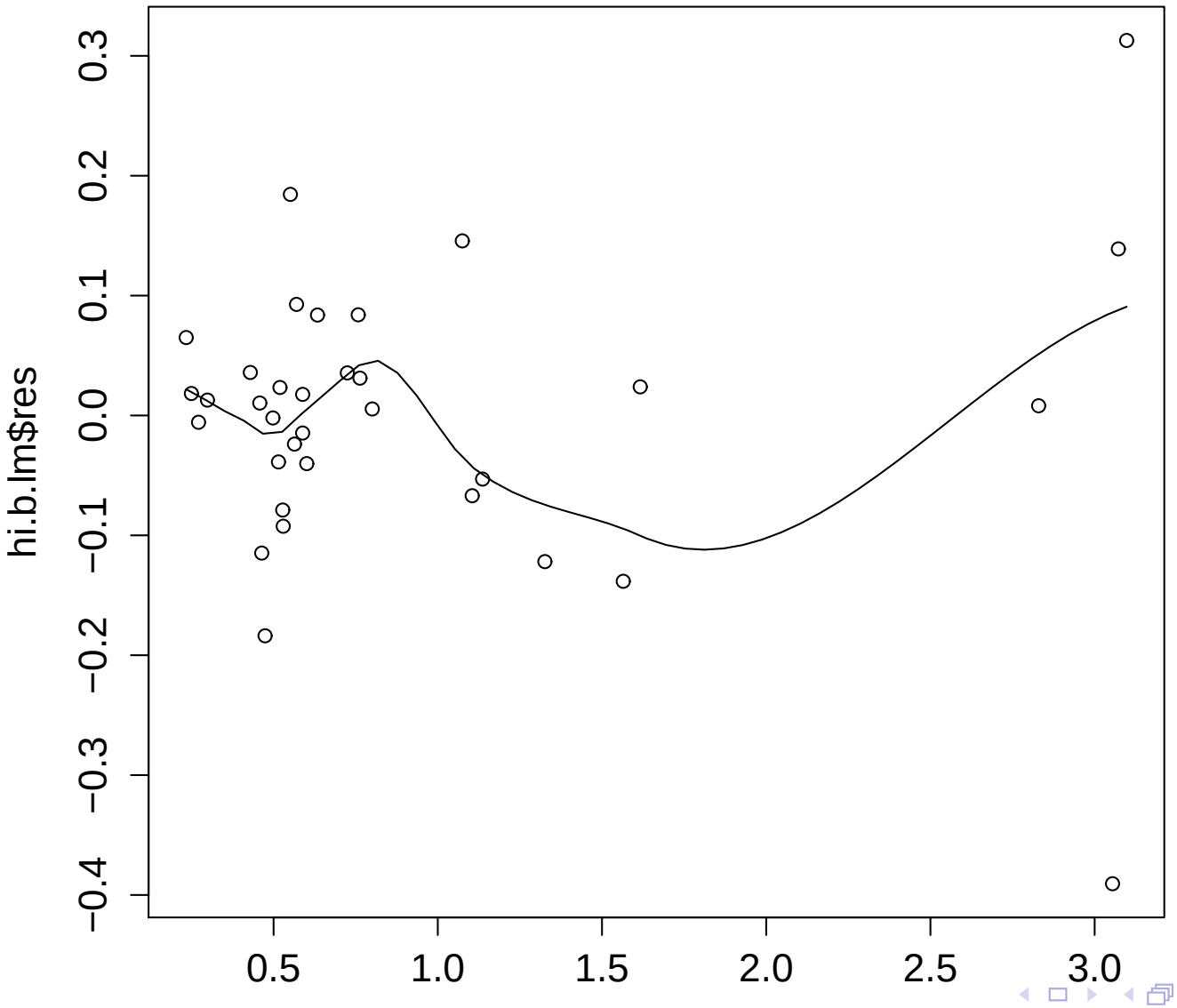
```
> hi.b.lm <- lm(time ~ dist + climb + dist:climb, data=hi.a)
> summary(hi.b.lm)
Residuals:
     Min       1Q   Median       3Q      Max
-0.38684 -0.05109  0.01201  0.03721  0.31571
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.130e-02  6.946e-02  -0.163    0.872
dist         8.257e-02  8.207e-03  10.061 3.97e-11
climb        6.132e-05  4.125e-05   1.487    0.148
dist:climb   1.104e-05  3.028e-06   3.646    0.001
Residual standard error: 0.1244 on 30 degrees of freedom
Multiple R-squared: 0.9803,Adjusted R-squared: 0.9783
F-statistic:   497 on 3 and 30 DF,  p-value: < 2.2e-16
```

Modellverfeinerung II

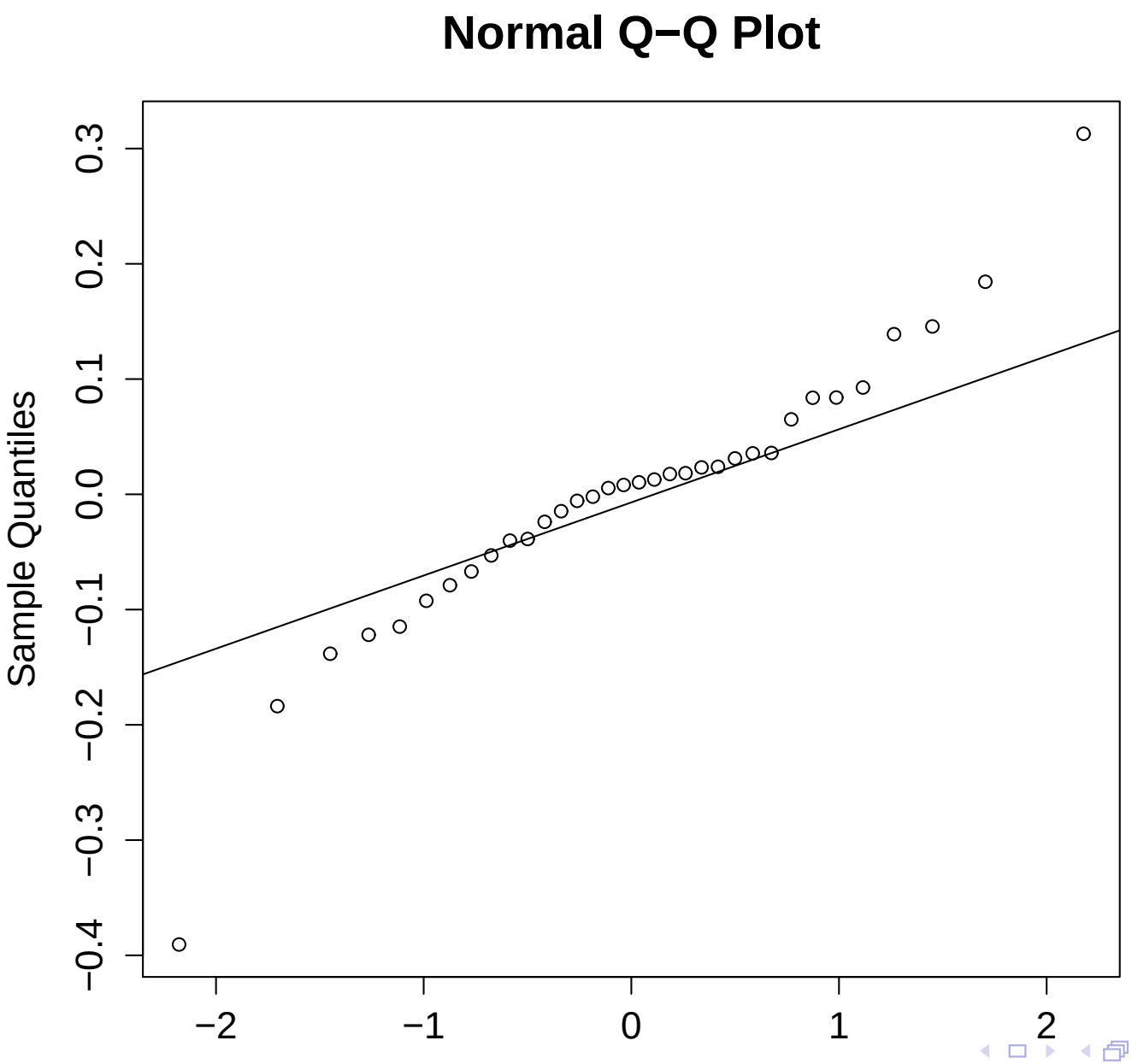
- R^2 ist gewachsen und der Intercept ist nicht mehr signifikant. (Sinnvolle Modellannahme!)
- Also: Achsenabschnitt aus dem Modell entfernen! In R:

```
>hi.b.lm <- lm(time ~ -1 + dist + climb + dist:climb, data=hi.a)
>summary(hi.b.lm)
Call:
lm(formula = time ~ -1 + dist + climb + dist:climb, data = hi.a)
Residuals:
     Min       1Q   Median       3Q      Max
-0.39059 -0.04982  0.00924  0.03577  0.31281
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
dist       8.147e-02  4.592e-03  17.742  < 2e-16
climb      5.590e-05  2.394e-05   2.336   0.0262
dist:climb 1.146e-05  1.605e-06   7.137 5.07e-08
Residual standard error: 0.1224 on 31 degrees of freedom
Multiple R-squared: 0.9915,Adjusted R-squared: 0.9907
F-statistic:  1202 on 3 and 31 DF,  p-value: < 2.2e-16
```

Diagnostische Plots III



Diagnostische Plots IV



Zwischenfazit

- Wer bis hierher kommt, kann schon viel mehr als die meisten. Jetzt noch die Kür!
- Der QQ-Plot der Residuen ist noch nicht optimal.
- Welche Beobachtungen sind die Abweichler im QQ-Plot?

```
> which.max(hi.b.lm$res); which.min(hi.b.lm$res); hi.a
```

- Es sind lange, steile Rennen!
- Evtl ist der Zusammenhang nicht rein linear?

Modellverfeinerung (Kür!)

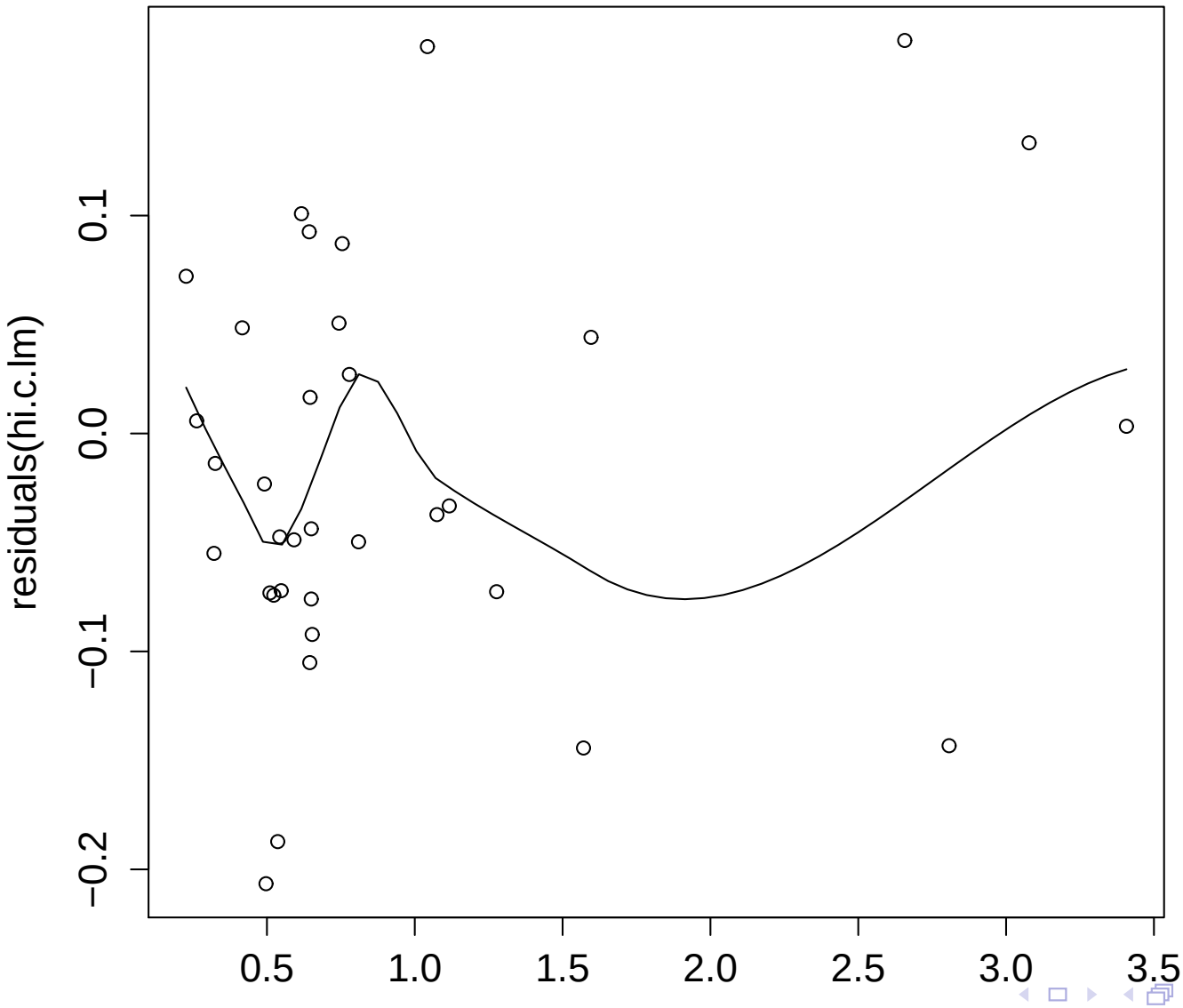
- Annahme: Die Länge geht im Wesentliche linear in die Zeit ein, die Steigung hat aber überproportionalen Einfluß auf die Endzeit.
- Das Modell:

$$\text{time} = \beta \cdot \text{dist} + \gamma \cdot \text{climb}^{\delta}$$

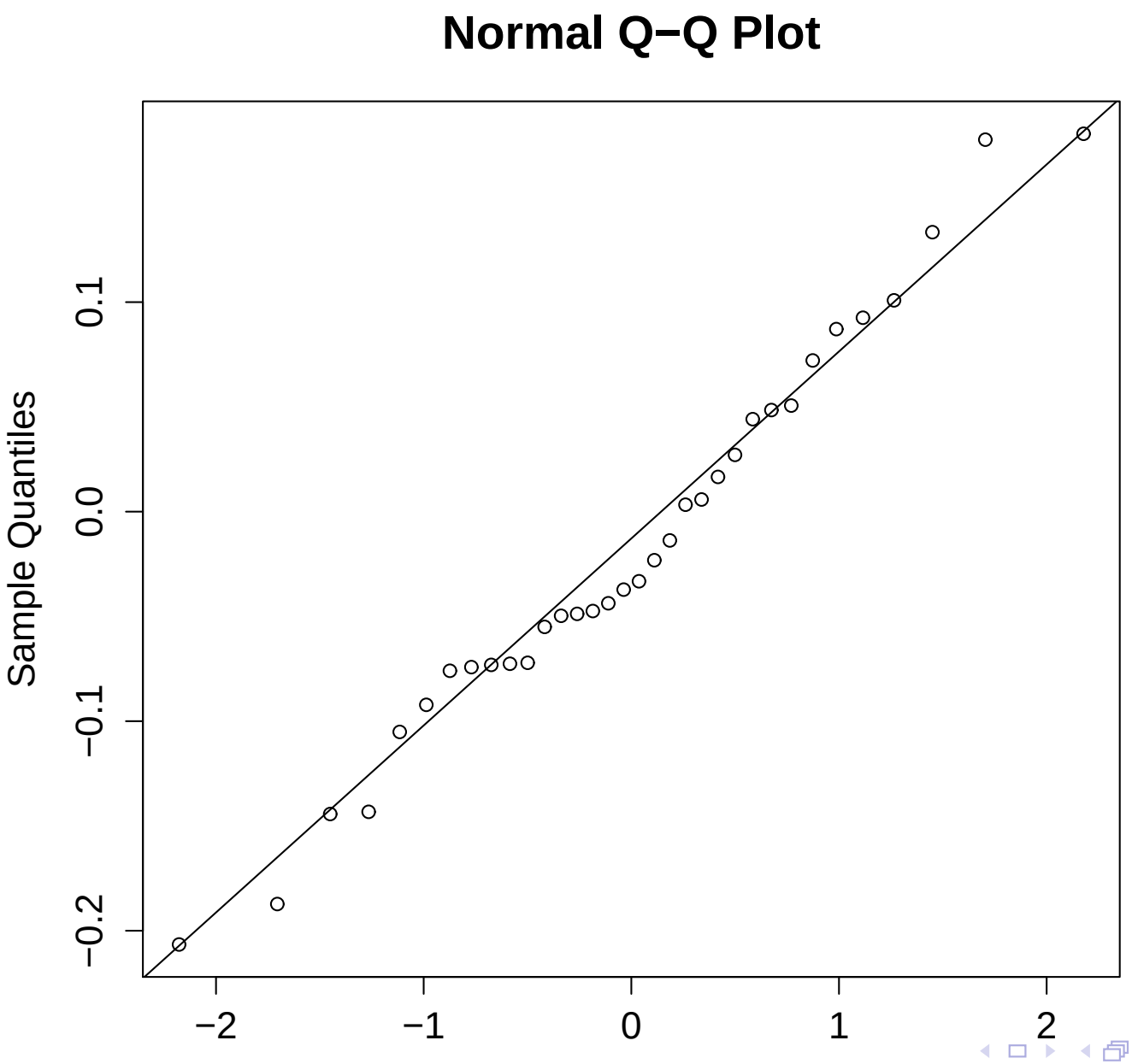
- In R:

```
hi.c <- hi.a ; hi.c$climb <- hi.c$climb/5280
### Damit X'X gut konditioniert ist!
hi.c.lm <- nls(time ~ (beta*dist) +
                gamma*(climb^delta) ,
                start= c(beta=1, gamma=1, delta=1), data=hi
1 - var(residuals(hi.c.lm))/  var(hi.a$time) ### r.squared
[1] 0.98
```

Diagnostische Plots V



Diagnostische Plots VI



Was war mit Beobachtung 18?

- Das Modell scheint nunmehr den Annahmen zu entsprechen!
- Mit der schließlichen Modellanpassung ergibt sich, dass Strecke 18

```
> predict(hi.c.lm, data.frame(dist=hills[18,"dist"],
            climb=hills[18,"climb"]/5280))
[1] 0.3213
```

  Stunden gedauert haben sollte. Am Wahrscheinlichsten ist also eine Fehleingabe, bei der statt 0.3 Stunden 1.3 Stunden eingegeben wurden.
- Als Pedant könnte man die ganze Analyse an dieser Stelle mit den korrigierten Daten wiederholen.

Teil 2 LaTeX

- LaTeX ist ein Makropaket für TeX geschrieben von Leslie Lamport (Turing-Preisträger 2013!) in den 1980er Jahren.
- TeX wurde von Donald Knuth ab 1977 entwickelt, da er mit dem Satz mathematischer Konstrukte nicht zufrieden war.
- TeX gilt als praktisch bugfrei.
- Unerreicht: Formelsatz
- Vorbildlich: perfekte Silbentrennung seit 30 Jahren!
- Mehr Historie: zu LaTeX und zu TeX

Was ist das Konzept?

- Ansatz in Office-Pakten: WYSIWYG.
- Ansatz in LaTeX: WYGIWYM (logische Auszeichnung der Textelemente).
- LaTeX-Dokumente sind reine Textdokumente, die Auszeichnungen für die Struktur (Überschriften, Listen, Tabellen) enthalten, die aber als Text nicht so aussehen.
- Für die Druckausgabe wird das Textdokument "kompiliert". Aus `dokumentenname.tex` wird `dokumentenname.pdf` erstellt.
- Die Idee ist, möglichst wenig manuell in die Formatierung einzugreifen. Die Software kennt die bewährten Regeln des Buchsatzes!

Technik - Was braucht man?

- TEX-Distribution
  - Linux: Texlive in den wichtigen Distributionen vorhanden.
  - Windows: Miktex
  - MacOS: Mactex
  - Es gibt natürlich Alternativen, aber das sind die Hauptakteure
  - Die Distributionen bestehen aus den eigentlichen Kompilern (tex, latex, **pdflatex** etc.) und einer Unzahl von Paketen.
  - Das CRAN ist nach Vorbild des CTAN (*comprehensive TEXarchive network*) aufgebaut worden.
- Editor/IDE
  - TEX-Studio unterstützt alle Betriebssysteme
  - Alternativen: emacs, TeXnicCenter, winedt, Eclipse/TeXclipse usw.
  - Interessant: LyX, ein Mittelding zwischen Office und purem TEX, im Hintergrund aber reines TEX.

Topics

- Minimales LaTeX-Dokument
  - Anpassungen für Deutsch als Sprache
  - empfohlene Pakete
- Gliederungskommandos
  - Titelseite
  - Inhaltsverzeichnis
- Formeln
- Bilder
- Tabellen
- Literaturverzeichnis
- Indizes
- Onlineressourcen

Minimalbeispiel

- Die Textdatei sollte einen Namen haben, der mit der Endung `.tex` endet, z.B. `minimal.tex`.
- Es gibt eine Zweiteilung des Dokuments
  - Header
    - Dokumentenklasse
    - Pakete
    - Individuelle Makros bzw. Erweiterungen
  - Body: Text mit Gliederungskommandos und LaTeX-Makros.
- Minimaldokument

```
\documentclass{article}
% keine Pakete, keine Benutzermakros
\begin{document}
So einfach?
\end{document}
```

Alternative Dokumentenklassen

- Vorträge: `\documentclass{beamer}`
- Bücher: `\documentclass{book}` (zusätzliche Gliederungsebene `part`)
- Briefe: `\documentclass{scrlttr2}` oder `g-brief2`

Anpassungen für Deutschland

- Alles nur Vorschläge!
- `\documentclass[a4paper]{scrartcl}`
  Dokumentenklasse aus dem KOMA Paket und DIN A4 Papier.
- ```
  \usepackage[ngerman]{babel}    % deutsche Rechtschreibung,
                                 % Trennregeln
  \usepackage[utf8]{inputenc}    % Kodierung des Textes
  \usepackage[T1]{fontenc}       % Kodierung der Fonts
                                 %(schöner am Monitor)
  \usepackage{amsmath,amsfonts,amssymb} % Mathepakete
  \usepackage{graphicx}          % Einbindung von Grafik
  \usepackage{hyperref}          % Links aus PDF Dateien
  \usepackage{listings}          % Quelltext
  \usepackage{biblatex}          % Literaturverzeichnis
  ```
  ```

Titelseite

- Definition möglichst weit vorn, auch als Teil des Vorspanns möglich.
- Beispiel:

```
\title{\LaTeX-Crash}
\author{Dr. D. Steuer \\
        \href{mailto:steuer at hsu-hh.de}{steuer@hsu-hh.de},
        Tel. 2819, H1 R 1397}

\date{April 2015} % oder \date { \today }
```

- Aufruf dann im Dokument an gewünschter Stelle mit

```
\maketitle
```

Gliederungskommandos

- Möchte man ein automatisch korrektes Inhaltsverzeichnis, so ist

  `\tableofcontents`

  an passender Stelle im Dokument einzufügen.
- Aufgenommen werden dort die Kommandos

  ```
  \chapter
  \section bzw. \section*
  \subsection bzw. \subsection*
  \paragraph bzw. \paragraph*
  ```
- Nummerierung wird automatisch aktuell gehalten, auch bei komplizierten Änderungen!
- Absätze werden mit zwei Zeilenvorschüben getrennt!

Aufzählungen

- Es gibt Umgebungen für nicht numerierte Listen und für numerierte Listen.
- Unnumeriert Listen werden in der `\itemize` Umgebung gesetzt,
- numerierte in der `\enumerate` Umgebung.
- Einträge in den Listen folgen jeweils dem Schlüsselwort `\item`.
- Die Umgebungen können geschachtelt werden.

Formeln

- Zwei Sorten Formel: Text (inline) und abgesetzte Formeln
- Im Text mit `$..$` abgetrennt:
  `$\int_{-\infty}^{\infty} x dx$` ergibt $\int_{-\infty}^{\infty} xdx$
- Abgesetzt mit `\[...\]` :

$$\int_{-\infty}^{\infty} xdx$$

- Nummeriert mit `\begin{equation} ... \end{equation}` :

$$F(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt \tag{1}$$

- Reicher Vorrat an mathematischen Zeichen (Menü oder direkt per Kommando!)

Einbinden von Bildern

- Bei Verwendung von pdflatex sollten auch PDF Grafiken verwendet werden.
- `\includegraphics{}: \includegraphics{bild.pdf}`
- Stimmt die Größe nicht, so kann nachträglich mit `\scalebox{factor}{\includegraphics{...}}` das Bild skaliert werden.
- Kann in die `figure` Umgebung eingebunden werden, siehe `\begin{figure}...\end{figure}`
- ```
  \begin{figure}[h]  \centering
   \includegraphics{...}
   \caption{...}
   \label{...}
  \end{figure}
  ```
- Der Parameter `h` drückt den Wunsch aus, dass das Bild an genau dieser Stelle erscheint. Bilder und Tabellen sind sogenannte floats, für die LaTeX versucht die optimale Position zu finden.
  ```

Tabellen

- Tabellen werden ebenfalls über ihre Struktur beschrieben.

```
\begin{tabular}{clr|} % center, left, right alinged,
                      % vertikaler Trenner
Zeile 1, Feld 1 & zeile 1, feld 2 & ..., feld 3 \\ \hline
Zeile 2, Feld 1 & ... & unten rechts \\
\end{tabular}
```

ergibt

Zeile 1, Feld 1	zeile 1, feld 2	..., feld 3
Zeile 2, Feld 1	...	unten rechts

- Um eine Tabelle analog zu einem Bild zu behandeln gibt es die `table` Umgebung.

Tabellen

- ```
  \begin{table}[h]
  \centering
  \begin{tabular}{clr|} % center, left, right alinged,
                        % vertikaler Trenner
  Zeile 1, Feld 1 & zeile 1, feld 2 & ..., feld 3 \\ \hline
  Zeile 2, Feld 1 & ... & unten rechts \\
  \end{tabular}
  \caption{Beispieltabelle}
  \label{tab:beispiel}
  \end{table}
  ```
- Ergibt

Zeile 1, Feld 1	zeile 1, feld 2	..., feld 3
Zeile 2, Feld 1	...	unten rechts

Tabelle: 1 Beispieltabelle
  ```

Tabellen

- Es sind selbstverständlich auch komplizierte Formatierungen möglich, auch das Zusammenfassen mehrer Spalten.
- `\listoftables` fügt eine Tabellenverzeichnis an entsprechender Stelle im Dokument ein, analog `\listoffigures` eine Abbildungsverzeichnis.
- Bezüge auf vergebene Labels mit `\ref{label}}` bzw. `\pageref{}`, z.B, ergibt Tabelle `\ref{tab:beispiel}` auf Seite `\pageref{tab:beispiel}` das Folgende: Tabelle 1 auf Seite 136

Textauszeichung

- `\texttt{}` Schreibmaschinenschrift
- `\textit{}`, `\textbf{}` *Italic*, **bold**
- Schriftgrößen können mittels

  `\begin{schriftgrad}`

  `...`

  `\end{schriftgrad}`

  gesetzt werden. Standardmäßig gibt es `tiny`, `footenotesize`, `small`, `normal`, `large`, `Large`, `huge`, `Huge`.
- ...viel, viel mehr ...

Was alles einfach so noch geht!

- Literaturverzeichnis (`\thebibliography`), Aufruf im Dokument mit `\cite{keylist}` Vorbereiten der Literaturdatenbank am Besten mit einem externen Tool. Beliebt und gut ist jabref.
- Fußnoten mit `\footnote{Text der Fussnote}` einfach in den Text schreiben. Das Satzprogramm übernimmt den Rest.
- Stichwortverzeichnisse, Bildverzeichnisse etc. pp.

Online Resourcen

- Recht umfassendes Wikibook
- A not so short introduction to LaTeX
- Essential LaTeX
- LaTeXReferenz
- Einführung auf deutsch.

Übung 8 zur LaTeX Einführung

Beschreiben Sie die Erstellung des plots der Sturmfluten in einer LaTeXDatei! Nutzen Sie die Klasse `scrartcl`. Binden Sie Bilder aus PDF Dateien ein! Tabellieren Sie die Anzahl der Sturmflutkategorien. Verweisen Sie im Text auf Bilder und Tabellen.

Die Kombination von R und LaTeX ergibt Reproducible Research!

- Ursprung liegt im **literate programming** von Donald Knuth bereits um 1980. „Programs are meant to be read by humans, and only incidentally for computers to execute.“
- Im Kern geht es darum Analysen in ihrer Gesamtheit einem zukünftigen Rezipienten an die Hand zu geben. Die Wiederholbarkeit wird erreicht indem sowohl die Daten **vollständig** vorhanden sind, als auch die Analyse **vollständig** und **vollständig dokumentiert** vorliegt. Die Analyse umfasst (idealerweise) auch die das *computational environments*, welches zur Analyse genutzt wird.

Warum Reproducible Research?

- „Non-reproducible single occurrences are of no significance to science.“(Karl Popper)
- Im Jahr 2012 wurde eine Studie von Begley und Ellis in Nature veröffentlicht, die ein Jahrzehnt Forschung untersuchte. Diese Studie fand, dass 47 von 53 Papieren zu medizinischer Krebsforschung nicht reproduzierbar waren.
- Es gibt eine wirkliche Krise der wissenschaftlichen Methode.
- In der Pharmakologie ist die Wiederholbarkeit der Schlüssel zur Zulassung neuer Medikamente.

Was hindert bisher an RepRes?

- Zeitdruck
- Daten in Excel Sheets
- Point and Klick Interfaces (SPSS)
- Proprietäre Datenformate (STATA, SAS)
- Proprietäre Dateiformate für den Report (.docx)
- Fehlende Tools, um den Anreiz zu schaffen, aus diesen Fallen zu entkommen.

Wie kann man WiFo in den Arbeitsalltag von Wissenschaftlern bringen?

- Es muss einfach sein! So einfach, dass man nicht zurück will, zu den proprietären Formaten.
- Die Vorteile müssen offensichtlich sein.

Was ist bei der Auswahl der Werkzeuge zu beachten?

Anforderungen (und Konsequenzen)

- Jede am Prozess beteiligte Datei muss **von Menschen lesbar** sein. Beliebte lock-ins z.B. MS Office, SAP, STATA
- Damit wird jede Art von Vendor-Lock-in vermieden
- Daten z.B. als .csv oder Ähnliches
- Datenbanken: SQL Query Texte
- Programmcode und Analyseerbegnisse als Textfiles. Grafiken als Programm zu ihrer Erzeugung.
- Gandrud (2014): Die verschiedenen Dateien einer Analyse (Daten, Bilder, Tabellen, Code, Prosa) müssen **explizit** miteinander verbunden werden. Wenn sich z.B. an den Daten sich etwas ändert, sollen sich Tabellen und Graphen automatisch mitändern. Keine implizite Abhängigkeit, sondern explizite!

Konsequenzen!

- Damit fallen schon die üblichen Office-Produkte als Arbeitswerkzeuge aus. Ein Excel Worksheet ist nicht transparent. Natürlich muss in beide Richtungen mit Standardprogrammen kommuniziert werden können!
- Wenn z.B. Reviewer beurteilen sollen, ob ein Artikel angenommen wird, dann muss er in der Lagen sein, die Rechnungen zu reproduzieren. Was ist mit Software,die mit hohen Kosten verbunden ist (Matlab, SAS, Mathematica)?
- Was ist mit alten Versionen von Software? Z.B. STATA hat mal das Datenformat inkompatibel geändert.
- Eigentlich landet man zwangsläufig bei OpenSource oder mindestens bei „free as free beer“ Software, wenn
- RStudio ist ein perfektes Beispiel für die Power von Open Source und entsprechender Lizenzierung! Im Wesentlichen ein hervorragendes Interface zu einem ganzen Haufen FOSS!

Grenzen des Konzepts

- Wie konserviert man Rechenumgebungen? Vmware? Docker? Hardware einlagern?
- Wie konserviert man Daten? Wie kann die Integrität eines extrahierten Datensatzes gesichert werden?
- Revision Control Systems: svn, git etc. Extrem nützlich, sprengt aber den Rahmen eines solch kurzen Kurses

Die Werkzeuge (standing on the shoulders of giants)

- TEXbzw. LATEX(Donald Knuth)
- pandoc (John MacFarlane)
- R (The R-core team)
- RStudio (Hadley Wickham, ggplot, dplyr, devtools etc.)
- knitr (Yihui Xie)
- SWeave (Fritz Leisch)
- git (Linus Torvalds)

Arbeitsschritte eine quantitativen Arbeit

- Daten sammeln und aufbereiten Es gehört schon hier dazu, einen Plan zu haben, wo und wie man die Daten Forschungspartnern, insbesondere zukünftigen(!), zugänglich machen möchte. (Homepage, Owncloud, Dropbox..)
- Angst vor der Transparenz ist schlechte wissenschftliche Praxis!
- Analyse
- Präsentation

Erinnerung: Alle drei Schritte müssen reproduzierbar sein! Hoffnungsvollerweise fallen die letzten beiden Schritte zusammen, wenn man es richtig macht.

Geschichte

- Literate Programming (1979, Donald Knuth) LLet us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to humans what we want the computer to do."Donald E. Knuth, Literate Programming, 1984
- Hier Literate Data Analysis!
- Sweave (2002, Fritz Leisch)
- knitr (2011, Yihui Xie)
- org-mode mit babel (2011, Eric Schulte)
  Add-on für [emacs](https://www.gnu.org/software/emacs/), welches über das org-babel Modul eine Interface zu vielen Programmiersprachen bietet (maxima, gnuplot, R, python, C ...) Inklusive Exportfunktionen nach LaTeX, ODT, Markdown etc.

Die Idee der Literate Data Analysis

- Man möchte gern den Progammtext und den Analysetext in einer Datei haben. Alles was man dazu braucht, ist eine Syntax (und die Tools), um aus diesem **gemischten** Text den Programmcode bei Bedarf zu extrahieren und auszuführen.
- Bei Knuth war die Ausführung des Programms in situ noch nicht vorgesehen.
- Heute hat man den Ausgangstext, in dem Freitext und Programmtext wechselweise auftreten, und durch „weaven“wird ein Enddokument erzeugt, das Freitext und Ausgabe des Programmtexts, wahlweise auch den Quelltext, enthält.
- Praktische Probleme machen Bilder und Tabellen.
- Konvention: Dateiendung `.Rnw` für „R no web“

Begriff des Code-Chunks

Die Codeabschnitte im Text heißen traditionell „chunks“. Durch spezielle Syntax wird dem Weavingtool begreiflich gemacht, dass ein Textabschnitt als Programm zu interpretieren ist.
Bei der Mischung von LaTeXund R sieht das wie folgt aus:

```
Umgebender Text
<<opt.chunkname chunkoptions>>=
Programmcode
@
Umgebender Text
```

Die chunks in einem Textdokument werden in einem gesonderten R Prozess der Reihenfolge im Text entsprechend wie ein fortlaufendes R Programm ausgeführt!

Computation Control

- eval = TRUE | FALSE ; Der Codeblock wird ausgeführt (oder nicht)
- echo = TRUE | FALSE ; Der Quelltext wird in das Ergebnisdokument eingefügt (oder nicht)
- results = markup | asis | ... ; Die Art der Übernahmen der Ausgaben in das Ergebnisdokument
- error: (TRUE; logical) ; Stoppt nicht bei Fehlern!

Caching

Spezielle Option für aufwendige Rechnungen. knitr kann verfolgen, ob sich Codeblock ändern oder nicht. Wenn sich nichts geändert hat, kann über den `cache` Parameter festgelegt werden, dass die Werte der Variablen ohne Neuberechnung übernommen werden können.

- cache = TRUE | FALSE oder feiner abgestuft numerisch; Die Ergebnisse des Codeblocks werden vor unnützer Neuberechnung geschützt.
- dependson: Chunkname (NULL; character or numeric) ; Im Falle von cache = TRUE kann über dependson eine explizite Abhängigkeit von einem Block definiert werden.
- cache.vars: (NULL); erlaubt eine Eingrenzung des Cachemechanismus auf bestimmte Variablen.

Abbildungen

- dev: ('pdf' for LaTeX output and 'png' for HTML/markdown; character) ; Auswahl des Grafikformats. Alle Devices, die R bietet sind möglich. Z.B. ist `dev=c('pdf', 'png')` möglich!
- `fig.width, fig.height, out.width, out.height`: Skalierungen für Abbildungen. Relatives Skalieren ist möglich, z.B. `fig.width=.8\linewidth` in LaTeX

Sonstiges

- Alterativ innerhalb des Codeblocks z.B.

```
opts_chunk\$set(comment=NA, fig.width=6,
                                fig.height=6))
opts_chunk\$set(dev = c("pdf", "jpg"))
```

- Werte für die Optionen müssen in einer Zeile eingegeben werden. Die Ausdrücke müssen stets gültige R Ausdrücke sein.
- Punkte und Leerzeichen in Namen und Verzeichnisnamen vermeiden!

Tabellen

- Tabellen kopieren und formatieren gehört zu den langweiligsten und deshalb fehleranfälligsten Tätigkeiten überhaupt. Eine automatische Erzeugung ist deshalb in jedem Falle vorzuziehen.
- R-Pakete für schöne Tabellen: xtable, stargazer, apsrtable, knitr::kable
- Wichtige Chunkoptionen: `asis, hide, markup`

```
<< tabellen, results='asis'>>=
library(xtable)
set.seed(17041967)
learnhours <- sample(100:200,30)
result <- learnhours + rbinom(30,30,0.5) -15
reg1 <- lm(result~ learnhours)
nice.table <- xtable(
  reg1,caption="Von nichts kommt nichts")
print.xtable(nice.table,type="latex")
@
```

Tabellen (Fort.)

Verschiedene Optionen können global gesetzt werden, z.B.
`options("xtable.type" = "html")`
Eigentlich, wenn mit knitr, dann mit knitr::kable, da dieses Kommando automatisch das richtige Ausgabeformat erzeugt!

```
<< kable>>=
  knitr::kable(data.frame(
          cbind(learnhours[1:5],result[1:5])))
@
```

Die Möglichkeiten des Styling werden vom Paket kableExtra stark verbessert.

Abbildungen

- Nützliche Chunkoptionen sind z.B: fig.align='center', out.width out.height
- Unabhängig vom Grafiksystem werden die üblichen R Kommandos in den chunk geschrieben, um die Abbildung zu konstruieren. Der Rest wird im Hintergrund von knitr erledigt! (Temporäre Dateien anlegen, device auswählen, etc.)

Reproduzierbare Simulationen

Um Wifo auch im Bereich der Monte-Carlo-Methoden richtig anzuwenden ist `set.seed()` das unbedingt nötige Kommando. Es erzwingt einen Startwert für die Erzeugung der Pseudozufallszahlen und somit die Wiederholung des kompletten Streams

```
<< random_number>>=
set.seed (17041967)
mean(runif(100))
mean(runif(100))
set.seed(17041967)
mean(runif(100))
@
```

RStudio - Betriebssystem für WiFo

- RStudio ist im Prinzip nicht nötig, da es lediglich das Setup aller Werkzeuge übernimmt. Alle Kommandos sind auch aus von der R Kommandozeile möglich. Allerdings ist z.B. das Setup von pandoc nicht trivial.
- „Lediglich“ ist aber die Lüge an dieser Aussage. RStudio löst ein großes Problem!
- Die eigentliche Einbettung von Analyseergebnisse übernimmt ein R-Paket namens `knitr`.
- Es sind auch andere Quellformate als LaTeXund Rmöglich, z.B. RMarkdown mit einfacherer Syntax, welches Export desselben Quellcodes nach HTML, Word und PDF ermöglicht!
- Optimaler Einstieg über die Seite des Autors selbst.

Literatur

- Hain, Statistik mit R, RRZN Hannover 2011 (über die Uni erhältlich)
- Dalgaard, Introductory statistics with R, Springer (elektronisch über die Bibliothek verfügbar)
- Ligges, Programmieren in R, Springer (elektronisch über die Bibliothek verfügbar)
- Soetaert K (2013) plot3D: Plotting multi-dimensional data. R package version 1.0
- Yihui Xie (2012). knitr: A general-purpose package for dynamic report generation in R. R package version 0.6. http://CRAN.R-project.org/package=knitr
- Reichhaltige Informationen im Netz!