

USER

I HATE THIS PROGRAM! THE SOFTWARE DEVELOPERS WHO WROTE THIS WERE DUMBASSES!



Passing the blame

PROGRAMMER

I HATE THIS LANGUAGE! THE DESIGNERS WHO CREATED THIS LANGUAGE WERE DUMBASSES!



ENGINEER

I HATE BUILDING THIS CIRCUITRY! THE PHYSICISTS WHO DISCOVERED THESE LAWS WERE DUMBASSES!



LANGUAGE DESIGNER

I HATE THIS ENVIRONMENT! THE ENGINEERS WHO CREATED THIS SYSTEM WERE DUMBASSES!



PHYSICIST

MAN, FUCK GRAVITY



ENGINEER

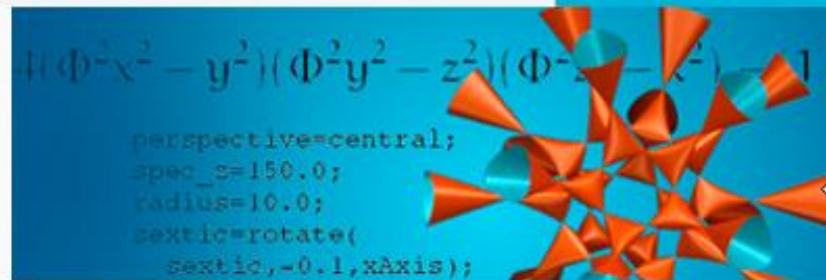
I HATE BUILDING THIS CIRCUITRY! THE PHYSICISTS WHO DISCOVERED THESE LAWS WERE DUMBASSES!





R Projekt for statistical computing

PD. Dr. Michael Thrun
Quirin Stier




Quellen

- Hauptseite:
 - <http://www.r-project.org/>
- Einführung zu R:
 - http://www.uni-ulm.de/fileadmin/website_uni_ulm/mawi.inst.110/lehre/ss08/stat1/R-skript.pdf
 - <http://adv-r.had.co.nz/>
 - <https://www.tutorialspoint.com/r/index.htm>

Einleitung

- R ist eine von Statistikern entwickelte Programmiersprache
- R ist **funktionale** und **vektorbasierte** Programmiersprache

Von Statistikern entwickelt

- Regression vertauscht Variablen
B/ $\text{lm}(y \sim x)$
- Bei bestimmten Plots sind y und x Achse vertauscht!
- Kategorische Variablen sind bekannt als Faktoren („factors“)
- Es gibt keine Strings, sondern nur „characters“
- Nicht-mathematische Operationen sind möglich
 - $\text{matrix}(2,2,2) * 1:2 = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 4 & 4 \end{pmatrix}$  **Nur in R!!!**
- Innerhalb von Funktionen können Standard-Funktionen überschrieben werden!
- Variablen können Funktionen überschreiben!

Warum funktional?

- Objektorientierte Paradigmen wurden mit S3 und S4 nachträglich eingeführt
- Es gibt keine „main“-Funktion
- Struktur durch Funktionen
 - Funktionen als Eingabe für Funktionen
 - Funktionen die Funktionen ausführen
- Die meisten Bibliotheken enthalten Funktionen ohne Objekte
- Viele Funktionen substituieren Objekte mit Listen um im Falle komplexere Sachverhalten alle Informationen kompakt zu sammeln
- Funktionen-Sammlungen für (fast) alles
 - „packages“ erhältlich über CRAN

Warum funktional?

- Wir arbeiten mit einem Workspace
- Wir arbeiten uns Schritt für Schritt durch Workflows
 - Dabei ist jeder Schritt einzeln ausführbar (siehe Jupyter Notebooks)
 - In jedem Schritt nehmen wir uns es frei, Visualisierungen einzubauen, Veränderungen vorzunehmen und den Output zu kontrollieren
- Durch die funktionale Bauweise können wir schnell einzelne Funktionen eines Pakets einsehen, eine Kopie davon erstellen, die Kopie verändern, die neue Variante laden (`source(„abc.R“)`), durch das Laden die Funktion aus dem Paket überschreiben (bei Bedarf kann man die Funktion wieder aus dem Workspace entfernen um diesen Schritt rückgängig zu machen), um damit schlussendlich eine neue Funktionalität zu testen
 - Debugging, Test neuer Funktionalitäten, Überprüfung einzelner Rechenschritte

Datenstrukturen

- Wir legen Variablen an, ohne sie zu definieren (kein int, float, double, etcpp)
- zB. „i = 1“
- Aber auch: A = list(), B = rbind(), C = matrix(0, 2, 2)
- Wie wissen wir, welcher Datentyp vorliegt?
 - Allgemein: typeof(abc)
 - Spezifisch: is.matrix(), is.vector(), is.numeric()
- Problem: Die numerische Matrix ist vom Typ double (typeof), damit man weiß, dass sie numerisch und auch eine Matrix ist, muss man das spezifisch abfragen
- c(a,b): konkateniert Vektoren a und b
- Rbind, cbind
 - „Rowbind“, „columnbind“: „Klebt“ passende Vektoren oder Matrizen oder gemischt zusammen (Zeilen- oder Spaltenweise)

Warum vektorbasiert?

- Vektor im mathematischen Sinn
 - Ansammlung von Elementen
 - mit Reihenfolge und Index
 - Elemente je Vektor von **einem** Datentyp
- Vektor als Basis aller Datenstrukturen
 - Zahl z :
 - Array v
 - Matrix m

Initialisierung und Zugriff

- Vektor als Basis aller Datenstrukturen

- Initialisierung: `z=42` Zugriff: `z`
- Initialisierung: `v=c(1,2,z)` Zugriff: `v, v[i]`
- Initialisierung: `m=matrix(v,3,3)` Zugriff: `m, m[1,], m[i,j]`
- Grundoperationen: `+, -, *, /, **` (Potenzieren) elementweise
- `cbind(c(1,2,3),c(2,3,4))` Spaltenweise binden
- `rbind()` analog Zeilenweise
- `which(m==42, arr.ind = TRUE)`
- `sort()` und `order()`
- `match` und `x %in% y`
- `t()` für's transponieren

Vorteile

- Funktion auf Vektor anwenden
 - schneller programmiert + ausgeführt als `for`-Schleife -> `apply()`
- Funktion auf ausgewählte Vektorelemente
 - schneller programmiert + ausgeführt als `if-then-else`
- echt paralleles Rechnen möglich, später mehr
- schnelle + einfache Matrizenrechnung

Mengen

Uniform distribution, 100 samples

`x=10*runif(100)`

Sample 10 times from values between 1 and 100 with replacement

`x=sample(x = 1:100, size = 10, replacement = TRUE)`

Zahlen von 1:100

`1:100`

Vektor mit 100 Nullen

`rep(0, 100)`

Sequenz von 0 bis 1 in 0.1 Schritte

`seq(0, 1, 0.1)`

Indexmengen

Alle Werte kleiner 5

`x[x<5]`

Alle Werte kleiner 5 und größer 0

`x[x<5 & x>0]`

Werte schnell sortieren

`x[order(x)]`

Teste die Bedingung elementweise

`ifelse(c(TRUE, FALSE, FALSE), 1, 0) => c(1,0,0)`

Mengendifferenz zwischen allen Zahlen von 1 bis 100 und der Menge bestehend aus den Zahlen 1 bis 10 vereint mit Zahlen von 90 bis 100

`setdiff(1:100, c(1:10, 90:100))`

Übersicht: Vektorfunktionen

Funktion	Erklärung
<code>c()</code>	Zusammenfügen von Elementen/Vektoren
<code>seq()</code>	Sequenzen
<code>rep()</code>	Wiederholungen
<code>max()</code> , <code>min()</code>	Extremwerte
<code>sum()</code> , <code>prod()</code>	Summe/ Produkt
<code>cumsum()</code> , <code>cumprod()</code>	kumulative Summe/ Produkt
<code>length()</code>	Länge
<code>sort()</code>	Sortiert den Vektor
<code>order()</code>	Indizes nach denen der Vektor sortiert wäre...
<code>x[order(x)]==sort(x)</code>	...ist also überall <i>TRUE</i>
<code>y[order(x)]</code>	Sortiert y nach x
<code>rank()</code>	Rangzahlen
<code>which(expression)</code>	Indizes der Elemente auf die <i>expression</i> zutrifft.
<code>a %in% b</code>	<i>TRUE</i> für alle Elemente in <i>a</i> die auch in <i>b</i> sind

Beispiel Matrizenoperationen

```
MyMatrix = matrix(round(100*rnorm(9)),3,3)
MyMatrix
diag(MyMatrix) = 0
MyMatrix
colnames(MyMatrix) = c("Col1", "Col2", "Col3")
MyMatrix
MyMatrix[, "Col1"]
rownames(MyMatrix) = c("Row1", "Row2", "Row3")
MyMatrix
MyMatrix["Row1",]
MyBooleanMatrix = lower.tri(MyMatrix, diag = FALSE)
MyBooleanMatrix
MyMatrix[MyBooleanMatrix]
colSums(MyMatrix)
sum(MyMatrix)
dim(MyMatrix)
which(MyMatrix < 50, arr.ind = FALSE)
which(MyMatrix < 50, arr.ind = TRUE)
MyCube = array(MyMatrix, c(3,3,3))
MyCube
dim(MyCube)
```

Scope in R: Grob erklärt

- Funktionen
 - Suche nach Funktionen erst im Workspace, dann im Rest (R Packages, Base R)
- Variablen
 - Zuerst Variablen im nächsten Scope (z.B. in einer for-loop, dann innerhalb der Funktion, dann im Workspace)
- Workspace sauber halten!
 - Überlädt Funktionen (das heißt: ihr überschreibt und baut damit Varianten von den bestehenden Funktionen)
 - Vorsicht: Ihr könnt Funktionen programmieren und mit Variablen ausführen, die nicht in der Funktion selber definiert werden! Dadurch können falsche Ergebnisse oder fälschlicherweise richtige Ergebnisse vorgegaukelt werden

Funktionen in R

- Input
 - Partiel sensitiv gegenüber Reihenfolge
 - Reihenfolge kann mit deklaration des parameters umgangen werden
 - `my_function = function(a, b, c){...}`
 - `my_function(b=1, 2, 3)`
 - Hier kommt b vor a, definierte Reihenfolge von a und c bleibt, also a=2 und c = 3
- Output
 - Return ist letzte Anweisung oder via Befehl `return()`
 - Gibt immer genau 1 Objekt zurück
 - Falls mehrere Variablen zurückgegeben wollten, dann auflisten
 - `return(list(„a“=2, b=3))`
 - Bei return mit Listen können die Namen der Ouput-parameter in Anführungszeichen sein oder nicht – am besten aber einheitlich

Grafiken mit R

Grafikbefehle	Erklärung
<code>plot()</code> <code>hist()</code> , <code>boxplot()</code> , <code>barplot()</code> , <code>pie()</code> <code>plot(density())</code> , <code>qqnorm()</code> <code>ts.plot()</code> , <code>curve()</code> <code>persp()</code> , <code>contour()</code> , ...	allgemeine plot-Funktion, meist Scatterplot Histogramm, Boxplot Säulendiagramm, Kuchendiagramm Kerndichteschätzer, Q-Q-Plot Zeichnen von Zeitreihen, Funktionen dreidimensionaler Plot, Contour Plot
Häufige Optionen	
<code>main</code> , <code>sub</code> , <code>xlab</code> , <code>ylab</code> , <code>xlim</code> , <code>ylim</code> <code>col</code> , <code>pch</code> , <code>lty</code> <code>axes=TRUE</code> :	Titel, Untertitel, Achsenbeschriftung Skalierung der Achsen Farbe, Symbol, Linientyp sollen Achsen gezeichnet werden?
Graphische Parameter mit <code>par()</code>	
<code>mfrow=c(a,b)</code> <code>new=TRUE</code> :	Darstellung von $a \times b$ Grafiken gleichzeitig neue Grafik wird zu bestehender hinzugefügt fast alles kann eingestellt werden ...
Hinzufügen von	
<code>lines()</code> , <code>abline()</code> , <code>polygon()</code> <code>points()</code> , <code>legend()</code> , <code>text()</code> <code>axis()</code> , <code>title()</code>	Kurve, Gerade, Polygonzug Punkt, Legende, Text Achse, Titel

- `dev.new()` # neues Fenster wird geöffnet

Plotly

- Use built-in functions:
- `plotly::plot_ly(x = ~rnorm(50), type = "histogram")`
- Build your own custom design visualizations:
- `x = seq(0,1,0.01)`
- `y = x**2`
- `MyPlot = plotly::plot_ly(type = "scatter", mode = "lines")`
- `MyPlot = plotly::add_lines(p = MyPlot, x = x, y = y)`
- `MyPlot = plotly::layout(p = MyPlot, xaxis = list(title = "x"), yaxis = list(title = "y"))`
- `MyPlot`

Funktionale Programmierung in R (II)

- Funktional Programmieren mit “apply” (apply, lapply, sapply, parapply)
- apply
 - `apply(matrix(1:4, 2, 2), 1, function(x) x**2)`
 - $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 9 \\ 4 & 16 \end{pmatrix}$
 - `apply(matrix(1:4, 2, 2), 1, sum)` (row sum)
 - $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \Rightarrow c(4, 6)$
 - `apply(matrix(1:4, 2, 2), 2, sum)` (column sum)
 - $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \Rightarrow c(3, 7)$

Funktionale Programmierung in R (II)

- `apply/lapply`
 - `lapply(1:10, function(x) if(!(x %% 2)){x**2}else{x**3})`
 - `c(1,2,3,4,5,6,7,8,9,10) => c(1,4,27,16,125,36,343,64,729,100)`
 - Zahlen 1 bis 10, falls gerade dann Quadrat, sonst ungerade und dann 3te Potenz
- `apply` liefert einen Vektor zurück
- `lapply` liefert eine Liste zurück
- `parApply/parSapply/parLapply` ist die parallelisierte Version von `apply/sapply/lapply`

Laufzeitmessung in R

```
x = Sys.time()
MyVec = c()
for(i in 1:10000){
  MyVec = c(MyVec, i)
}
y = Sys.time()
diff = y-x
```

Rprof('test1.txt')

lapply(MyVec, function(x) x+1)

Rprof(NULL)

summaryRprof('test1.txt')

```
> summaryRprof('test1.txt')
$by.self
      self.time self.pct total.time total.pct
"tryCatchOne"    0.02    100      0.02    100

$by.total
      total.time total.pct self.time self.pct
"tryCatchOne"    0.02    100      0.02    100
".rs.callAs"     0.02    100      0.00     0
"Rprof"          0.02    100      0.00     0
"tryCatch"       0.02    100      0.00     0
"tryCatchList"   0.02    100      0.00     0
"withCallingHandlers" 0.02    100      0.00     0

$sample.interval
[1] 0.02

$sampling.time
[1] 0.02

>
```

Download auf CRAN und Github

- <https://mthrun.github.io/rpackages>

Erste Schritte

Home Projects R Packages Lectures Publications Contact Me

R Packages on Cran and Github

Name	Licence	Link	Authors
DataVisualizations	GPL-3	CRAN	Michael Thrun, Felix Pape, Onno Hansen-Goos, Dirk Eddelbuettel, Craig Varrichio, Alfred Ultsch
DatabionicSwarm	GPL-3	CRAN	Michael Thrun
ProjectionBasedClustering	GPL-3	CRAN	Michael Thrun, Florian Lerch, Felix Pape, Kristian Nybo, Jarkko Venna
GeneralizedUmatrix	GPL-3	CRAN	Michael Thrun, Alfred Ultsch
FCPS: Fundamental Clustering Problems Suite	GPL-3	CRAN	Michael Thrun, Peter Nahrgang, Felix Pape, Vasyl Pihur, Guy Brock, Susmita Datta, Somnath Datta, Luis Winkelmann, Alfred Ultsch, Quirin Stier
Umatrix	GPL-3	CRAN	Michael Thrun, Michael Thrun, Alfred Ultsch
AdaptGauss: Gaussian Mixture Models (GMM)	GPL-3	CRAN	Michael Thrun, Onno Hansen-Goos, Rabea Griesse, Catharina Lippmann, Florian Lerch, Jörn Lötsch, Alfred Ultsch
ABCanalysis	GPL-3	CRAN	Michael Thrun, Florian Lerch, Jörn Lötsch, Alfred Ultsch
Pareto Density Estimation	GPL-3	CRAN	Michael Thrun, Onno Hansen-Goos, Rabea Griesse, Catharina Lippmann, Jörn Lötsch, Alfred Ultsch
DataIO	GPL-3	Github	Michael Thrun, Florian Lerch, Michael Thrun, Catharina Lippmann, Felix Pape, Onno Hansen-Goos, Quirin Stier, Sabine Herda
TimeSeries	GPL-3	Github	Michael Thrun
Classifiers	GPL-3	Github	Michael Thrun, Rabea Griesse, Alfred Ultsch
Distances	GPL-3	Github	Raphael Paebst, Felix Pape, Michael Thrun
Hidden Markov Models (RHmm)	GPL-3	Github	Olivier Taramso, Sebastian Bauer, Maintainer: Michael Thrun
ImageProcessing	GPL-3	Github	Michael Thrun
Multiresolution Forecasting Using Wavelets	GPL-3	CRAN	Quirin Stier, Michael Thrun
Nonparametric naiv bayes classifier using pareto density estimation and parametric naiv bayes classifier	GPL-3	Github	Michael Thrun, Quirin Stier
AdaptGauss2D: Interactive Gaussian Mixture Modeling in 2D	GPL-3	Github	Quirin Stier, Michael Thrun

If you use these packages, please cite me accordingly. If you find bugs, please feel free to contact me.

Installation von Paketen

- Installation von Github direkt nach R
- Ihr braucht das Paket „remotes“
 - Eingabe ins R Terminal/Terminal von RStudio
 - `install.packages("remotes")`
- Installation eines beliebigen Packages von Github dann in folgendem Stil (Beispiel dbt.DataIO)
 - `remotes::install_github("Mthrun/dbt.DataIO")`

Data Input Output - DataIO

- Wir definieren unsere eigenen Datenstrukturen
- Wir sorgen damit für eine strukturierte und standardisierte Arbeitsweise
- Dafür brauchen wir eine eigene Lese- und Schreibroutine
- Dafür ist die dbt.DataIO da (Databionic Team – Data Input Output)
- Meistens arbeiten wir mit numerischen Matrizen
 - $N \times d$
 - (Hochdimensionaler) Datenraum – reelle Zahlen
- Prinzip der Trennung der Belange:
 - Daten und Klassen sind getrennt zu speichern
 - Daten und Zusatzinformationen getrennt

DataIO: LRN („Learn“ – Numerische Matrizen)



file:///F:/PUB/ZFileFormatDocuments/lrn.html

Erste Schritte

[Prev](#)

5.8. File format *.lrn
Chapter 5. File Formats

5.8. File format *.lrn

The .lrn file contains numeric data,

```
# comments
# all comment lines start with #, all header lines with %
% n
% m
% 9
%KeyName      d1      d2      ..      dm
Name1      Name2      ..      Namem
k1          x11      x12      ..      x1m
k2          x21      x22      ..      x2m
.           .        .        .        .
.           .        .        .        .
kn          xn1      xn2      ..      xnm
```

Table 5.5. Elements of *.lrn file

n	Number of rows in Key resp. data
m	Number of columns in file, including Key column
ki	Key for data set row i
di	Description of column, 1 for valid data column
Namei	Name for the i-th data column, must not contain whitespace
xij	Elements of the data matrix; Decimal Point denoted by '.'

[Prev](#)

5.7. Key format

NOTE: there can be only ONE column with defined ==9 i.e. the column containing the Key

NOTE2: Keys with a value higher than 10^{10} get written out in Scientific Notation and might lose their uniqueness by that.

[Up](#)
[Home](#)

DataIO – NAMES (Matrix für Zeichenketten)



file:///F:/PUB/ZFileFormatDocuments/names.html

Erste Schritte

5.10. File format *.names
Chapter 5. File Formats

[Prev](#)

5.10. File format *.names

The *.names file contains names (=teststrings) and longer descriptions for the index keys otherwise used in e.g. *.lmn or *.cls files.

```
# comments (optional)
# all comment lines start with #, the line containing the number of datalines lines starts with %
% n
k1      NameText1      Description1
k2      NameText2      Description2
..      ..
..      ..
..      ..
kn      NameTextn      Descriptionn
```

Table 5.6. Elements of *.names file

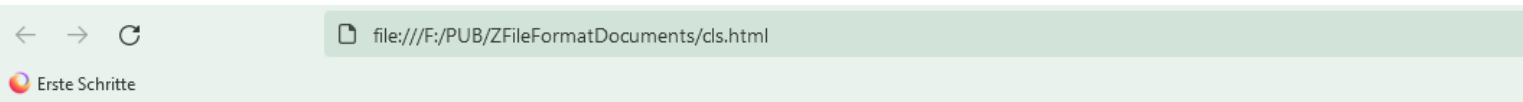
n	number of names (valid rows) in the following
kI	key (=unique Integer) for the i-th name (row)
NameTextI	a string; usually a short identifier, for example a GeneName
DescriptionI	longer description corresponding to NameTextI

[Prev](#)

5.9. File format *.data

[Up](#)
[Home](#)

DataIO – CLS (Classification)


[Prev](#)

5.3. File format *.cls
Chapter 5. File Formats

5.3. File format *.cls

The *.cls file usually contains a classification of data, which is stored in a corresponding *.lm file

```
# comments (optional)
#
# NOTE only the % n header line is required in the *.cls file, all others are optional.
# The optional lines describe the name of the class and the color to be assigned to the class
% n
%c_1 s_1 r_1 g_1 b_1
%c_2 s_2 r_2 g_2 b_2
%...
%c_m s_m r_m g_m b_m
k1 c1_1
k2 c1_2
.. ..
.. ..
.. ..
kn c1_n
```

Table 5.2. Elements of *.cls file

n	Number of data-points.
m	Number of classes.
c_j	Class number.
s_j	Class name.
r_j	Red component of class color.
g_j	Green component of class color.
b_j	Blue component of class color.
ki	Key of i-th data-point.
cl_i	Classification (=Integer) of i-th data-point.

[Prev](#)

5.2. File format *.bm

[Up](#)
[Home](#)

DataIO

- Datenformate in ZFileFormats nachschlagen
- Beispiel LRN am Defaultdatensatz „iris“ (verfügbar über Base R)
- `data(iris)`
- `Data = iris`
- `WriteLRN(FileName = "Iris", Data = Data, Header = colnames(Data), Key = 1:dim(Data)[1], OurDirectory = NULL)`
- `V5 = ReadLRN(FileName = "IrisTestX", InDirectory = NULL)`
- `V5$Data`

Fragen?

- Beim nächsten Mal wird im Tutorium die Lösung mit dem empfohlenen Workflow vorgestellt
 - Das nächste Tutorium ist in 2 Wochen
- Auf die anderen Pakete, die in Zukunft dazukommen, wird eingegangen, sobald es in den Aufgaben erforderlich ist
- Ihr müsst für die nächsten Aufgaben erstmal R aufsetzen, Rstudio installieren, die dbt.DataIO installieren und einen Datensatz vorbereiten (Mehr dazu gleich)
- Bei Schwierigkeiten -> Quirin_Stier@gmx.de