

Einführung in R

Christoph Dalitz

Hochschule Niederrhein
Fachbereich Elektrotechnik & Informatik

August 2022

R: Benutzung

Debugging

Anhalten Ausführung im R Code mit Befehl `browser()`
Ausgabe von Variableninhalten und spezielle Befehle:

help: Liste der möglichen Befehle

c: Fortsetzen des Programms beim nächsten Statement

n: vollständiges Ausführen der nächsten Befehlszeile

s: nächster Schritt irgendwo im Programm

Q: Browser nebst Programm verlassen

Rstudio

- Button *Run* führt nur Zeile des Cursors aus
- deshalb immer den Button *Source* verwenden

Achtung: Rstudio legt u.a. Verzeichnis "R" in \$HOME an.
(keine Ahnung wie man das verhindert)

R: Benutzung

Aufruf

- interaktive R-Shell mittels *R* oder IDE *R Studio*
- Batchausführen Script mittels *Rscript*
- Beenden interaktive R-Shell mittels *q()* oder *Ctrl-D*
- Unterdrücken "Save Workspace?" durch Option *--no-save*
 - Linux, MacOS X: *alias R="R --no-save"* in *\$HOME/.profile*
 - Windows: *--no-save* in Desktop-Icon-Verknüpfung ergänzen

Ausführen Script

- am R-Prompt mit *source("script.r")*
- *Rscript script.r* schreibt Plots in Datei *Rplots.ps*

Abfrage Aufrufmodus über Funktion *interactive()*.

Beispiel zum Ändern Dateinamen *Rplots.ps* bei Batchaufruf:

```
if (!interactive()) { postscript("bla.ps") }
```

R: Packages

Benutzung von Packages

Funktionen sind in *Packages* zusammengefasst
Beim Start werden nur wenige Packages geladen

- *(.packages())* listet geladene Packages auf
- *library()* listet installierte Packages auf
- *library(package)* lädt *package* nach

Wie löst R Objektnamen auf?

- Suchpfad wird angezeigt mit *search()*
Parameter *pos=...* bei *library()* setzt Position im Suchpfad
- Package Namespace ggf. explizit voranstellen, z.B. *stats::sd()*

Nachinstallation von Packages

Installation von CRAN in der R-Shell mit

- *install.packages(c("pkg1", "pkg2"))*
- *remove.packages(c("pkg2"))*

R: Syntax

Allgemeines

- R ist casesensitiv
- Punkt in Bezeichner möglich, aber keine Zahl am Anfang
- Kommentare von # bis Zeilenende; keine mehrzeiligen Kommentare; ggf. Workaround *if (FALSE) { ... }*
- Kommandoabschluss Zeilenumbruch oder Semikolon
Kommando am Zeilenende noch nicht fertig ⇒ fortgesetzt

```
x <- 2*3 +      x <- 2*3
  4*5          # ok      + 4*5 # klappt nicht
```

Funktionsaufrufe

- Keyword-Argumente möglich, z.B. *plot(x, y, col="red")*
- Objektmethoden nicht über Zugriffoperator (".") in C++, sondern durch "parametrische Polymorphie": *method(obj)*

R: elementare Datentypen

Datentyp kann abgefragt werden mit *class()*

```
> class(pi)           > class(iris)
[1] "numeric"         [1] "data.frame"
```

Vektoren

- R kennt *keine skalaren* Datentypen:
⇒ jeder "Skalar" ist Vektor der Länge Eins

```
> length(pi)          > length("abc")
[1] 1                  [1] 1
```

- *numeric*: keine Unterscheidung int/float: $2 / 4 == 0.5$
- *character*: single oder double Quotes, Escape durch `\`
- Vektorkonstruktor ist *c(...)*:

```
x <- c(0.5,6,3); strings <- c('a','b','cde')
```

```
x <- c(x, c(3,4)) # -> 0.5 6 3 3 4 (Concatenation)
```

R: Variablen und Werte

Definition von Variablen

- Interpretersprache ⇒ keine Deklaration nötig
erste Zuweisung erzeugt Variable:

```
a <- 10 # Anlage Variable und Zuweisung
```
- eingebaute Variablen können überlagert werden;
ggf. Auflösung über Namespace-Prefix:

```
pi # -> 3.141593
pi <- 3 # überschreibt eingebaute Variable
pi # -> 3
base::pi # -> 3.141593
```

Verwaltung von Variablen

- *ls()* listet Objekte in aktueller Umgebung auf;
für sonstige Umgebungen im Suchpfad z.B. *ls(package:stats)*
- *rm(var)* löscht Variable aus aktueller Umgebung

R: elementare Datentypen

Vektoren (Forts.)

- spezielle Vektoren:

```
x <- seq(1,2,b=0.5) # -> 1.0 1.5 2.0
x <- rep(9,3)       # -> 3 3 3
```

Matrizen

Zusammensetzen aus Vektoren

spaltenweise (*c(olumn)bind*) oder zeilenweise (*r(ow)bind*):

```
A <- cbind(c(1,-1), c(0,1)) # -> 1 0
A <- rbind(c(1,0), c(-1,1)) # -> -1 1
```

- spezielle Matrizen:

```
matrix(1, nrow=2, ncol=2) # -> 1 1
# 1 1
diag(c(1,1))              # -> 1 0
diag(1, nrow=2, ncol=2)  # 0 1
```

R: Operatoren

Zuweisung

wahlweise "<-" (lies: "gets") oder "="

Achtung: bei Zuweisung (auch Vektoren) wird immer Kopie erstellt

Rechenoperationen

- Potenzieren mit `**` oder `^`
- Operatoren `+`, `-`, `*`, `/` sind überladen:
bei Vektoren oder Matrizen *elementweise*
- Matrizenmultiplikation mit `%*%`, dyadisches Produkt mit `%o%`

Wahrheitswertige Operatoren

- Vergleichsoperatoren `==`, `<`, `<=`, `>`, `>=`, `!=`
- Boolesche Operatoren `!` (not), `|` (or), `&` (and), `||`, `&&`,
`c(TRUE,TRUE) & c(FALSE,TRUE) # -> FALSE TRUE`
`c(TRUE,TRUE) && c(FALSE,TRUE) # -> FALSE`
- dreiwertige Logik: Test auf NA mit `is.na()`

R: Datentyp Data.Frame

Ein *Data Frame* ist eine Tabelle mit benannten Spalten.

Anfang des eingebauten Data Frames *iris*:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa

Indizierung

- Spaltenauswahl durch `$` und Spaltennamen
oder durch eckige Klammern und Spaltenname als String
`iris$Sepal.Length`
`iris["Sepal.Length"]`
- Auch `[row,col]`-Indizierung wie bei Matrizen:
`# alle Spalten für Species == "setosa"`
`iris[(iris$Species == "setosa"),]`
`# nur Spalte Sepal.Length für Species == "virginica"`
`iris[(iris$Species == "virginica"),]$Sepal.Length`

R: Operatoren

Indexoperator [..]

- Indizes beginnen bei Eins
- Index kann auch Vektor sein:
`x <- c('a','b','c','d')`
`x[c(3,1)] # -> 'c' 'a'`
`x[seq(2,4)]; x(2:4) # -> 'b' 'c' 'd'`
Bemerkung: `2:4` ist eine Kurznotation für `seq(2, 4, by=1)`
- Index kann auch ein boolescher Vektor sein:
`x > 'b' # -> FALSE FALSE TRUE TRUE`
`x[x > 'b'] # -> 'c' 'd'`
- negative Indizes zum Ausblenden von Werten:
`x[-2] # -> 'a' 'c' 'd'`
`x[-2:-4] # -> 'a'`
- Matrizen-Indizierung mit `A[row,col]`
Auswahl erste drei Zeilen: `A[1:3,]`

R: Datentyp Data.Frame

Erzeugen von Data Frames (1)

Einlesen eines Data Frames aus einer CSV-Datei mittels

```
df <- read.table("data.csv", header=TRUE, sep=";")
```

- Spaltennamen werden aus erster Zeile übernommen,
wobei Leerzeichen durch einen Punkt ersetzt werden
- Spaltennamen alternativ (insbes. wenn `header=FALSE`)
über Parameter `col.names=c("col1","col2",...)` angegebbar

Explizites Anlegen eines Data Frames mittels

```
x <- c(1,2); y <- c('a','b')
```

```
df <- data.frame(col1=x, col2=y)
```

- ohne Spaltennamen (`col1`, `col2`) werden Namen der Variablen
(im Beispiel `x`, `y`) als Spaltennamen genommen
- Alternative: Parameter `col.names=c('col1','col2')`

R: Datentyp Data.Frame

Erzeugen von Data Frames (2)

Meist liegen Daten nicht als Spalten, sondern als Zeilen vor
neue Zeile sei `row <- data.frame(col1=3.4, col2='d')`

- Lösung für wenige Datensätze:

```
# lege leeren Data Frame an
result <- data.frame()
# hänge Zeile an und weise wieder dem Data Frame zu
result <- rbind(result, row)
```

Nachteil: bei n Zeilen $O(n^2)$ Kopiervorgänge durch Zuweisung

- Lösung für $n \gg 1$ Datensätze:

```
# preallokiere Speicher (siehe ?numeric, ?character)
result <- data.frame(col1=numeric(n),
                    col2=character(n), stringsAsFactors=FALSE)
# weise einzelne Zeile zu
result[3,] <- row
```

Dalitz: Einführung in R. -12-

R: eingebaute Funktionen

allgemeine Funktionen

- `help(func)` - Online-Hilfe zu `func`; Kurzform: `?func`
- `help.search("term")` - Durchsuchen Online-Hilfe nach `term`
- `getwd()` - get working directory
- `setwd(..)` - set working directory; Pfadname kann relativ sein
- `options(..)` - Darstellungsoptionen für Ergebnisse

Matrix- und Vektor-Funktionen

- `length(x)` - Länge des Vektors x
- `sort(x)` - gibt sortierten Vektor zurück (x bleibt unverändert)
- `diag(x)` - erzeugt Diagonalmatrix mit x auf Diagonale
`diag(A)` - Diagonale von A als Vektor
- `dim(A)` - Dimension (nrows, ncols) der Matrix A
- `t(A)` - Transposition der Matrix A

Dalitz: Einführung in R. -14-

R: weitere Datentypen

List

Wie Vektoren, aber verschiedenartige Elemente möglich
(Aufpassen bei Indizierung!)

```
a <- list("a", 1, c(1,2,3))
a[2] # gibt 2. Element als Liste zurück
a[[2]] # gibt 2. Element als Wert zurück
```

Factor

Wie Vektor, aber für Kategorien (nominale Werte)
typisches Beispiel: Klassenlabel, z.B. `iris$Species`

```
classes <- factor(c("lachs", "barsch", "barsch"))
levels(classes) # -> "barsch" "lachs"
```

- Nachteil: riskant wenn Inputdateien nicht alle alle Levels enthalten \Rightarrow selbes Level erhält in R unterschiedliche Kennung. Lösung: `read.csv(..., stringsAsFactors=FALSE)`

Dalitz: Einführung in R. -13-

R: eingebaute Funktionen

Wahrscheinlichkeitsverteilungen

Jede eindimensionale Verteilung hat in R vier Funktionen, die sich aus dem Stammnamen und einem Präfix zusammensetzen:

- d - "density" für die Wahrscheinlichkeitsdichte
- p - "probability" für die CDF $F(x) = P(X \leq x)$
- q - "quantile" für die inverse CDF $F^{-1}(p)$
- r - "random" für Zufallsgenerator

Wichtige Verteilungen sind `norm` (Normalverteilung),
`binom` (Binomialverteilung), `chisq` (χ^2 -Verteilung),
`t` (Student's t -Verteilung), `unif` (stetige Gleichverteilung)

Für die mehrdimensionale ("multivariate") Normalverteilung:

- Zufallswerte mit `mvrnorm` aus Package `MASS`
- Wahrscheinlichkeitsdichte als Formel direkt eingeben

Dalitz: Einführung in R. -15-

R: eingebaute Funktionen

Ein- und Ausgabe

Variable + *Enter* gibt den Wert aus

Formatierung kann über *options()* gesteuert werden

- *sink("datei")* - leitet Ausgabe in *datei* um
sink() setzt Ausgabe wieder auf *stdout*
- *print(..)* - gibt einen String oder Variable aus
- *cat(..)* - gibt mehrere Werte hintereinander aus, z.B.
`cat("\n", x, "\n", sep="")`

Stringfunktionen

- *sprintf(..)* - gibt formatierten String zurück, z.B.
`s <- sprintf("x=%4.2f", x)`
- *paste(s1, s2, sep=)* - Konkatenation von *s1* und *s2*
- *substr(s, start=n1, stop=n2)* - Extraktion Substring
- *as.numeric(s)* - Umwandlung in Zahl

R: Grafik

Hinzufügen Legende

Muss manuell mit *legend(..)* hinzugefügt werden:

```
legend("topright",           # Position
       c("first", "second"),  # Beschriftungen
       col=c("red","blue"),   # Farbe und Linetype
       lty=c(1,1) )          # der Liniensamples
```

Beispiel: Sinus- und Kosinus-Kurve in einem Plot:

```
x <- seq(0,2*pi,by=0.01)
plot(x, sin(x), type='l', col="red")
lines(x, cos(x), lty=2, col="blue")
legend("bottomleft", c("sin", "cos"),
      lty=c(1,2), col=c("red","blue"))
```

Bemerkung: *sin* und *cos* sind für Vektoren überladen:

```
sin( c(0, pi/2, pi) ) # -> 0 1 0
cos( c(0, pi/2, pi) ) # -> 1 0 -1
```

R: Grafik

Zeichnen von Kurven

Zeichnen der Werte in *y* als Funktion von *x*:

```
plot(x, y, type='l', lty=1, col="red")
```

- *type='l'* (klein L) für Verbindung mit Linien ('p' = Punkte)
- Überblick weitere Parameter (*lty*, *col*, ...) mittels *?par*
wichtige Parameter sind *xlab*, *ylab* (x/y-Achsenlabel)
und *xlim*, *ylim* (Achsen-Wertebereiche)

plot überschreibt immer den alten Plot

Hinzufügen zu bestehendem Plot erfolgt mit

- *lines(x,y,col="blue")* - mit Linien verbundene Punkte
- *points(x,y,pch=19)* - Einzelpunkte (*pch* = "point character")

Neues Grafikfenster mit *dev.new()*

R: Grafik

Spezielle Plots

- Histogramm:
`hist(iris$Sepal.Width, breaks=12, border="red")`
- Box-Plot aus Data Frame:
`boxplot(iris$Sepal.Length)`
aufgliedert nach Kategorie `iris$Species`
`boxplot(Sepal.Length ~ Species, data=iris)`

Export und Druck der Grafik

Setzen anderes *Graphics Device*, z.B. *postscript*, *pdf*, *xfig*, *wmf*

- Lösung 1: Ändern Device vor *plot*-Befehlen
`pdf("p.pdf")` # folgende Plots als PDF nach "p.pdf"
... (plot-Befehle)
`dev.off()` # schreibt Datei und setzt Device zurück
- Lösung 2: aktuell *angezeigten* Plot speichern
`dev.copy(pdf, "plot.pdf", width=8, height=6)`
`dev.off()`

R: Grafik

Formatierungen des Plots mit *par* setzbar

Mehrere Plots nebeneinander

- Zeilen und Spalten mit *mfrow* definieren:
`par(mfrow=c(1,2))` # 1 Zeile, 2 Spalten
- *plot*-Befehle schreiben nacheinander in die Felder

Ränder setzen

- Ränder innerhalb der Plots:
 - ▶ `par(mar=c(bottom, left, top, right))`
 - ▶ Einheit ist Lines of Text:
`par(mar=c(4,4,0.1,0.1))`
- Ränder außerhalb aller Plots,
z.B. um Titel für mehrere Plots auf einer Seite zu ergänzen
 - ▶ `par(oma=c(bottom, left, top, right))`
 - ▶ Einheit ebenfalls Lines of Text

R: Benutzerdefinierte Funktionen

Beispiel: Funktion die zwei Zahlen addiert

```
addfunc <- function(x, y, double=FALSE) {  
  z <- x + y  
  if (double) z <- 2*z  
  return(z)  
}
```

Bemerkungen:

- Parameter werden als Kopie übergeben ("Call by value")
- Funktionen sind selbst Objekte und können wieder als Parameter an andere Funktionen übergeben werden
- Funktionsname ohne Klammern listet Sourcecode auf
- statt *return* kann auch *invisible* verwendet werden
unterschiedliches Verhalten bei Aufruf ohne Zuweisung:
return ⇒ Ergebnis ausgegeben; *invisible* ⇒ nicht
auch bei *invisible* erzwingbar mit `(addfunc(2,3))`

R: Kontrollfluss

Bedingte Verzweigung

```
if (a > 1) {  
  print("a > 1")  
} else {  
  print("a <= 1")  
}
```

Schleifen mit for und while

```
for (symbol in sequence) {      while (condition) {  
  expressions                    expressions  
}
```

Iterationen über Vektor *x* also alternativ mit

```
for (xi in x)      { ... } # Elementzugriff: xi  
for (i in 1:length(x)) { ... } # Elementzugriff: x[i]
```