ulm university universität
uulm

Einführung in R

Marius Hofert
marius.hofert@uni-ulm.de
Ulm University

2007-10-18

1 Einleitung

Die Statistik Software R ist ähnlich zur Statistik Software S-Plus. Beide basieren auf der Sprache S und sind weit verbreitet. Der Hauptunterschied zwischen R und S-Plus besteht darin, dass R open source Software ist, d.h. sie ist frei verfügbar, während S-Plus kommerziell ist. Dadurch kann man für R sehr viele Zusatzpakete bekommen, die von Anwendern selbst entwickelt wurden und in vielen Fällen eine Problemlösung bieten, wie sie in kommerzieller Software nicht zu finden ist. Auch kann ein Code an eigene Bedürfnisse angepasst werden. Allerdings muss man im Hinterkopf behalten, dass open source Code fehlerhaft sein kann. Ein weitere Vorteil besteht trotz allem darin, dass der Code eingesehen werden kann und somit ist es auch bei komplexen Sachverhalten möglich, genau zu sehen, was wo und vor allem wie berechnet wird.

2 Allgemeines

- R homepage (download + manuals): http://www.r-project.org/
- R Grafik Gallerie (Grafik Beispiele): http://addictedtor.free.fr/graphiques/
- R ist auf den neueren Rechnern unserer Fakultät installiert und kann per remote login (ssh) genutzt werden:
  (1) Einloggen auf dem Rechner "dublin" via (mit entsprechendem Login)
  `ssh <login>@dublin.mathematik.uni-ulm.de`
  (2) Start von R durch Eingabe von: `R <Enter>`
- R Hilfe Aufruf: `help(<name>)` oder `?<name>`
- R Kommentare: `#`
- R ist case sensitive, d.h. Gross- und Kleinschreibung führen nicht zum selben Ergebnis.

3 Allgemeine Kommandos

- `source(foo)`: Lädt das Skript im Pfad `foo`.
- `library(foo)`: Lädt die Bibliothek `foo`.
- `ls()`: Zeigt alle gespeicherten Objekte an.
- `class(foo)`: Gibt die Klasse des Objektes `foo` an.
- `attributes(foo)`: Zeigt die Bestandteile des Objektes `foo` an. Damit kann auf einzelne Komponenten eines Objektes gezielt zugegriffen werden.
- `summary(foo)`: Gibt eine Zusammenfassung des Objektes `foo` an.
- `remove(foo)`: Löscht das Objekt `foo`.
- `cat("foo",bar,"\n")` bzw. `print(paste("foo",bar,"\n"))`: Aneinanderhängen des Strings `foo` und des Objektes `bar` mit anschliessendem Newline.

4 Kommandos zur Erzeugung und Bearbeitung von Datenstrukturen

4.1 Vektoren

Ein Vektor ist eine Zusammenfassung von Objekten gleicher Art zu einer endlichen Folge. Er kann z.B. mit folgenden Kommandos generiert werden.

- `c(foo,bar)`: Erstellt einen Vektor durch Angabe der Objekte `foo` und `bar`.
- `rep(foo,bar)`: Erstellt einen Vektor durch `bar`-fache Wiederholung des Objektes `foo`.
- `seq(from=,to=,by=)` bzw. `seq(from=,to=,length=)`: Erstellt einen Vektor durch bilden einer Folge von `from` bis `to` mit Schrittweite `by` bzw. Vektorlänge `length`.
- `numeric(foo)` bzw. `character(foo)`: Erstellt Vektoren der Länge `foo` die Objekte des Typs `numeric` (also Gleitpunktzahlen) bzw. `character` (also Zeichenstrings) enthalten. Andere Typen sind möglich.

Auf die Elemente eines Vektors kann wie üblich über Indizierung in eckigen Klammern zugegriffen werden. Dabei ist zu beachten, dass Vektoren in R 1-indiziert sind, d.h. dass das erste Element eines Vektors über den Index 1 erreicht werden kann (nicht etwa 0 wie in C/C++).

Es gibt zahlreiche Operationen auf Vektoren. Ein paar sind im folgenden Beispiel demonstriert.

Beispiel

```
> x=c(1,3,6,4,2,5) #Zuweisung. Alternativ auch: x<-c(1,3,6,4,2,5)
> class(x) #liefert die Klasse des Objektes x
[1] "numeric"
> x #Stellt x dar
[1] 1 3 6 4 2 5
> x[2] #liefert das 2. Element von x
[1] 3
> length(x) #liefert die Laenge von x
[1] 6
> sum(x) #summiert
[1] 21
> min(x) #bildet Minimum
[1] 1
> max(x) #bildet Maximum
[1] 6
> mean(x) #bildet das Stichprobenmittel
[1] 3.5
> var(x) #bildet die Stichprobenvarianz
[1] 3.5
> sort(x) #sortiert
[1] 1 2 3 4 5 6
> x[order(x)] #sortiert auch
[1] 1 2 3 4 5 6
> x[x>=3] #liefert einen Vektor, der nur die Werte von x enthaelt, die
    groesser oder gleich 3 sind
[1] 3 6 4 5
```

4.2 Matrizen

Matrizen in R unterscheiden sich von Vektoren nur durch die Darstellung der Elemente, nämlich in einem Rechteckschema, statt einer Zeile/Spalte. Insbesondere enthalten Matrizen auch nur Objekte gleicher Art. Sie können mit folgenden Kommandos generiert werden.

- `matrix(data,nrow=foo,ncol=bar,byrow=T)`: Erstellt eine (`foo`×`bar`) Matrix aus den Daten `data`. Der Parameter `byrow=T` deutet an, dass die Daten zeilenweise in die Matrix geschrieben werden. Dabei steht `T` für "true".
- `cbind(foo,bar)` bzw. `rbind(foo,bar)`: Fügt die Vektoren `foo` und `bar` spalten- bzw. zeilenweise zusammen. Dies funktioniert auch mit anderen Datentypen.

Im Folgenden wieder ein Beispiel, das den Gebrauch von Matrizen erläutert.

Beispiel

```
> A=matrix(1:10,nrow=2,byrow=T) #bildet eine Matrix aus den Daten 1 bis
    10 mit 2 Zeilen
> class(A) #liefert die Klasse des Objektes A
[1] "matrix"
> A
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
```

```
[2,]    6    7    8    9   10
> A[1,2] #liefert das Element der 1. Zeile und 2. Spalte
[1] 2
> A[1,] #liefert die 1. Zeile von A
[1] 1 2 3 4 5
> A[,3] #liefert die 3. Spalte von A
[1] 3 8
> t(A) #transponiert die Matrix A
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
> A%*%t(A) #liefert das Matrixprodukt AA^T
     [,1] [,2]
[1,]   55  130
[2,]  130  330
```

Ähnlich zu Matrizen existieren in R noch Arrays. Diese können mehr als 2 Dimensionen abbilden, sind aber eher selten im Gebrauch.

4.3 Listen

Listen sind ähnlich wie Vektoren eine Zusammenfassung von Objekten zu einer endlichen Folge. Der Hauptunterschied zu Vektoren besteht darin, dass Listen verschiedene Objektklassen zulassen, z.B. können Strings und Zahlen in der gleichen Liste auftreten. Listen können folgendermassen erzeugt werden.

- `list(foo,bar)`: Erstellt eine Liste aus den Objekten `foo` und `bar`.

Das folgende Beispiel demonstriert den Gebrauch von Listen.

Beispiel

```
> L=list(foo=c(1,2,3),bar=c("bla","blubb"))
> class(L)
[1] "list"
> L
$foo
[1] 1 2 3
$bar
[1] "bla"   "blubb"
> L$foo #liefert die Objekte in foo
[1] 1 2 3
> L[[1]] #wie oben
[1] 1 2 3
> L$foo[1] #liefert erste Wert im Vektor foo
[1] 1
> L[[1]][1] #wie oben
[1] 1
> attributes(L) #liefert die vorhandenen Attribute der Liste
```

```
$names
[1] "foo" "bar"
```

4.4 Data Frames

Daten liegen in der Anwendung oft als Data Frames vor. Ein Data Frame kann man sich vorstellen als Liste von Vektoren gleicher Länge, die aber unterschiedliche Objekttypen enthalten können. Dies ist der in R gängigste Datentyp. Ein Data Frame wird entweder durch Einlesen von Daten mit `read.table()` (s.u.) oder wie folgt erzeugt.

- `data.frame(foo,bar)`: Erstellt ein Data Frame aus den Objekten `foo` und `bar`.

Das folgende Beispiel dient zur Demonstration des Gebrauchs von Data Frames.

Beispiel

```
> D=data.frame(Student=c("Mustermann","Musterfrau"),Alter=c(24,21))
> class(D)
[1] "data.frame"
> D
     Student Alter
1 Mustermann    24
2 Musterfrau    21
> D$Alter
[1] 24 21
> attributes(D)
$names
[1] "Student"     "Alter"
$row.names
[1] 1 2
$class
[1] "data.frame"
> names(D) #liefert die Spaltenbeschriftung
[1] "Student"     "Alter"
> D=cbind(D,Studiengang=c("WiWi","WiMa")) #fuegt Studiengang hinzu
> D
     Student Alter Studiengang
1 Mustermann    24        WiWi
2 Musterfrau    21        WiMa
```

5 Einlesen/Schreiben von Daten aus/in Dateien

Liegt ein Datensatz in einer externen Datei vor, so kann er mit Hilfe folgender Funktionen als Data Frame in R eingelesen werden.

- `read.table(file,header=T,sep="\t",dec=".",row.names,col.names)`: Erstellt ein Data Frame aus den Daten die in der Datei (mit Pfad) `file` sind. Dabei gibt `header` an, ob in der Datei Spaltenüberschriften sind oder nicht. Defaultmässig wird `header` auf "true" gesetzt, genau dann wenn die erste Zeile der Datei weniger Spalten hat als die nachfolgenden Zeilen. Das optionale Argument `sep` gibt an, wie

die Daten im Input File getrennt sind (default: "white space", d.h. u.a. ein oder mehrere Leerzeichen, Tabs oder Newlines). Das optionale Argument `dec` gibt an, wie der "Dezimalpunkt" aussieht (default: Punkt, nicht Komma). Das Kommando `read.table()` hat noch zahlreiche weitere optionale Argumente (s. Hilfe), u.a. `row.names` bzw. `col.names` für Zeilen- bzw. Spaltennamen.

- `matrix(scan(file,sep="\t"),ncol,byrow=T)`: Das Kommando `scan` ist nicht so nutzerfreundlich wie read.table, dafür allerdings flexibler. Das Kommando `read.table` ist genau genommen lediglich ein Interface, das die wichtigsten Eigenschaften von `scan` mit geeignetem Parametersatz aufruft. In Verbindung mit `matrix` eignet sich `scan` vor allem zum Einlesen grosser Dateien von Input gleichen Datentyps (z.B. lauter Zahlen).

Zum Schreiben von Daten in Dateien können folgende Kommandos verwendet werden.

- `write.table(foo,file)`: Schreibt das Data Frame `foo` in die Datei `file`. Ähnlich zu `read.table`, beeinflussen zahlreiche optionale Parameter das Layout der geschriebenen Datei.
- `write.matrix(foo,file)`: Zum Schreiben grosser Dateien von Input gleichen Datentyps (z.B. lauter Zahlen) eignet sich das Kommando `write.matrix` (in der Library MASS), da es die Datei zeilenweise schreiben kann und daher deutlich schneller sein kann als `write.table`.

Für das folgende Beispiel verwenden wir die Datei exampledata.dat, die wie folgt aufgebaut ist.

Matrikelnummer	Alter
123456	24
234567	21
345678	23

Beispiel

```
> read.table("/Users/mhofert/mhofert/job/stat/misc/introduction\ to\
    r/contents/exampledata.dat") #einfaches Einlesen mit default
    Werten fuer die Argumente
             V1    V2
1 Matrikelnummer Alter
2         123456    24
3         234567    21
4         345678    23
> read.table("/Users/mhofert/mhofert/job/stat/misc/introduction\ to\
    r/contents/exampledata.dat",col.names=c("Matrikelnummer","Alter"))
     #Einlesen mit Vergabe von Spaltennamen
  Matrikelnummer Alter
1 Matrikelnummer Alter
2         123456    24
3         234567    21
```

```
4         345678    23
> read.table("/Users/mhofert/mhofert/job/stat/misc/introduction\ to\
    r/contents/exampledata.dat",skip=1,col.names=c("Matrikelnummer","
    Alter")) #Ueberspringe erste Zeile in der Datei
      Matrikelnummer Alter
1         123456    24
2         234567    21
3         345678    23
> T=matrix(scan("/Users/mhofert/mhofert/job/stat/misc/introduction\
    to\ r/contents/exampledata.dat",skip=1),ncol=2,byrow=T) #Einlesen
    mit scan (mit Ueberspringen der ersten Zeile)
Read 6 items
> T
       [,1] [,2]
[1,] 123456   24
[2,] 234567   21
[3,] 345678   23
> library(MASS) #Laden der Library MASS
> write.matrix(T,"/Users/mhofert/mhofert/job/stat/misc/introduction\
    to\ r/contents/output.dat") #Schreiben der Daten in die Datei
    output.dat
```

6 Programmieren in R

6.1 Kontrollstrukturen

Eine einfache Kontrollstruktur vom Typ If-Else wird in R wie folgt realisiert.

```
if(foo){
  print("foo")
} else{
  print("bar")
}
```

Logische Operatoren sind gegeben durch `&&` für "Und" und `||` für "Oder".

6.2 Schleifen

Eine For-Schleife wird in R folgendermassen verwendet.

```
for(i in 1:10){
  print(i)
}
```

Eine While-Schleife ist folgendermassen implementiert.

```
i=0
while(i<10){
  i=i+1
  print(i)
}
```

Eine Repeat-Schleife funktioniert auf folgende Weise.

```
i=0
repeat{
  i=i+1
  print(i)
  if(i==10){
    break
  }
}
```

6.3 Funktionen

Das Grundgerüst einer Funktion in R ist wie folgt. Die hier dargestellte Funktion bekommt einen Parameter x und hat einen optionalen Parameter y. Falls nur x übergeben wird, liefert die Funktion x+2, andernfalls (x und y werden übergeben) liefert die Funktion die Summe aus x und y.

```
myfunction=function(x,y=2){
  result=x+y
  return(result)
}
```

Als Beispiel spielen wir nun etwas mit der oben definierten Funktion. Dabei stellen wir fest, dass x oder y (oder beide) auch Vektoren sein dürfen.

Beispiel

```
> myfunction=function(x,y=2){
+   result=x+y
+   return(result)
+ }
> myfunction(1)
[1] 3
> myfunction(1,4)
[1] 5
> myfunction(c(1,2),4)
[1] 5 6
> myfunction(c(1,2),c(2,4))
[1] 3 6
```

7 Grafiken

Grafiken in R sind eine Wissenschaft für sich. Es gibt zahlreiche Parameter mit denen Grafiken verändert werden können. Wir werden hier nur wenige grundlegende Dinge auflisten, die wir häufig benötigen. Erstellt werden können Grafiken prinzipiell auf zwei unterschiedliche Arten. Zum Einen gibt es grundlegende Plot-Funktionen aus denen man eine Grafik zusammenstellen kann. Hierzu gehören folgende Funktionen:

- `plot(x,y)`: Plottet die Punkte in den Vektoren `x` und `y` ohne Verbindungslinien. Zahlreiche Argumente können die Eigenschaften des Plots verändern, z.B. die Option `type="l"` für "lines", d.h. die Punkte werden durch Linien verbunden. Siehe R Hilfe für andere Parameter.
- `plot(myfunction,a,b)`: Plottet die Funktion `myfunction` im Intervall zwischen `a` und `b`. Parameter wie oben.
- `barplot()`,`hist()` usw.: Erzeugt Balkendiagramme, Histogramme usw.
- `lines(x,y)`: Plottet einen Polygonzug durch die Punkte der Vektoren `x` und `y`.
- `abline`: Zeichnet Geraden.

Des Weiteren gibt es noch die Funktionen `title()` (Erzeugung eines Titels), `text()` (Schreiben eines Textes an eine bestimmte Stelle) und `legend()` (Erstellen einer Legende). Manche Einstellungsparameter können den Grafikfunktionen nicht direkt mitgegeben werden. Dies geschieht dann über die Funktion `par()`. Mit dieser Funktion kann man auch angeben, dass z.B. 2 Plots nebeneinander erscheinen sollen oder dass die y-Achsen Beschriftung gedreht werden soll und vieles mehr.

Beispiel

```
> pdf("/Users/mhofert/mhofert/job/stat/misc/introduction\ to\ r/
    contents/exampleplot1.pdf") #Oeffnen eines pdf-Device zum
    Schreiben der Grafik in eine pdf-Datei
> plot(pnorm,-2,2) #Plotten der Std. Normalverteilung
    Verteilungsfunktion auf [-2,2]
> par(col=2) #nachfolgende Plots sind rot
> plot(dnorm,-2,2) #Plotten der Normalverteilungsdichte auf [-2,2]
> dev.off() #Schliessen des pdf-Device
null device
          1
> pdf("/Users/mhofert/mhofert/job/stat/misc/introduction\ to\ r/
    contents/exampleplot2.pdf")
> par(mfrow=c(1,2)) #Unterteile das Bild in eine (1x2)-Matrix und
    fuelle diese mit den nachfolgenden Plots auf
> plot(pnorm,-2,2)
> par(col=2) #nachfolgende Plots sind rot
> plot(dnorm,-2,2) #Plotten der Std. Normalverteilungsdichte auf
    [-2,2]
> dev.off()
null device
          1
> pdf("/Users/mhofert/mhofert/job/stat/misc/introduction\ to\ r/
    contents/exampleplot3.pdf")
> plot(pnorm,-2,2)
> par(col=2)
> plot(dnorm,-2,2,add=T) #Plotten der Std. Normalverteilungsdichte
    auf [-2,2] ins selbe Bild
> dev.off()
null device
          1
```

Das folgende Beispiel generiert den Plot einer empirischen Verteilungsfunktion basierend auf 100 standardnormalverteilten Zufallszahlen.

Beispiel

```
> data=rnorm(100) #Erzeuge 100 N(0,1) Zufallszahlen
> library(stats) #Fuer empirische Verteilungsfunktion ecdf
> pdf("/Users/mhofert/mhofert/job/stat/misc/introduction\ to\ r/
    contents/exampleplot4.pdf")
> plot(pnorm,xlab="x",ylab="Fn(x)",xlim=c(-4,4)) #Plotte die Std.
    Normalverteilung Verteilungsfunktion auf [-4,4]
> par(col=2)
> plot(ecdf(data),do.points=F,verticals=T,add=T) #Fuege die ecdf
    basierend auf den Daten data (ohne Punkte an den Sprungstellen
    aber mit vertikalen Linien) hinzu
> title("Vergleich CDF und ECDF einer N(0,1) Vert.") #Titel
> par(col=1) #Zuruecksetzen der Farbe auf schwarz
> legend(0,0.2,c("cdf","ecdf (100 Beob.)"),col=c(1,2),lty=c(1,1)) #
    Legende
> dev.off()
null device
          1
```

Wenn man verschiedene Komponenten hochdimensionaler Daten gegeneinander plotten möchte (conditioning plots), so kann man dies sehr effizient mit Trellis Grafiken verwirklichen. Trellis Grafiken (in der Library lattice) können sehr gut gewisse Aspekte von Daten hervorheben, sind allerdings nicht einfach zu bedienen. Wir betrachten hierzu das folgende Beispiel.

Beispiel

```
> data=rnorm(100)
> library(lattice) #Library fuer Trellis Grafiken
> trellis.device(pdf,file="/Users/mhofert/mhofert/job/stat/misc/
    introduction\ to\ r/contents/exampleplot5.pdf")
> x=seq(-4,4,0.001) #Erzeuge eine Folge von Punkten an denen die
    Verteilungsfunktion ausgewertet wird
> xyplot(pnorm(x)~x,type="l",scales=list(relation="free"), #Erzeuge
    die Grafik
+   panel=function(...){
+     panel.xyplot(...,col=1) #Plot 1
+     llines(sort(data),(1:length(data))/length(data),type="s",col=2)
     #Plot 2
+     if(min(data)>=min(x)){ #Ergaenze linken Teil, falls noetig
+       llines(c(min(x),min(data),min(data)),c(0.0,0.0,1/length(data)
    ),col=2)
+     }
+     if(max(data)<=max(x)){ #Ergaenze rechten Teil, falls noetig
+       llines(c(max(data),max(x)),c(1.0,1.0),col=2)
+     }
+   },
+   xlab="x",ylab="Fn(x)", #Achsenbeschriftung
```

```
+    main="Vergleich CDF und ECDF einer N(0,1) Vert.", #Titel
+    key=list(x=0.55,y=0.2,lines=list(lty=c(1,1),col=c(1,2)),text=list
    (c("cdf","ecdf (100 Beob.)")),align=T,transparent=T) #Legende
+ )
> dev.off()
null device
          1
```

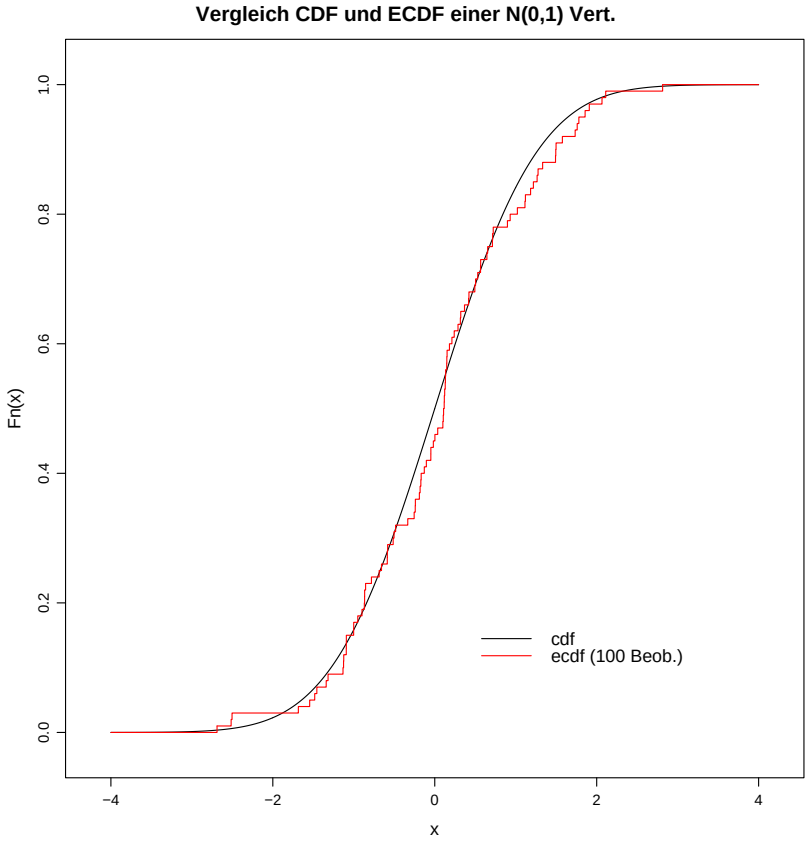

Figure 1 Trellis Plot Beispiel

Zur Demonstration ausgefallener Grafiken sei auf den Link zur R Grafik Gallerie (s.o.) verwiesen.