Einführung in die Programmierumgebung R

Michael Wenk [*†]

25.04.2018

Inhaltsverzeichnis

[*]Arbeitsgruppe „Cheminformatics and Computational Metabolomics“, Friedrich-Schiller-Universität Jena

1 Installation und Start von R

R ist eine freie Software-Umgebung und auf statistische Berechnungen sowie Grafiken ausgelegt. Sie kann von *https://www.r-project.org* heruntergeladen und anschließend installiert werden. Um nun die R-Programmierumgebung zu starten, wird das Programm **Terminal** geöffnet und der Befehl **R** eingegeben. Im Rahmen dieser Einführung wird jedoch die bereits installierte und grafische Benutzeroberfläche **RStudio** (*https://www.rstudio.com*) mit integrierter Entwicklungsumgebung für den weiteren Verlauf genutzt.

2 Grundlagen

2.1 Basisoperationen und Variablenzuweisung

Die vier Basisoperationen können mit + (Addition), − (Subtraktion), ∗ (Multiplikation) und / (Division) durchgeführt werden.

```
# Addition
3 + 4
```

```
## [1] 7
```

Das Speichern von Werten oder Ergebnissen erfolgt in Objekten, auch Variablen genannt. Für eine Zuweisung wird typischerweise ein „<“ und ein „−“ (Linkspfeil) verwendet. Die Ausgabe einer Variable geschieht mit der Eingabe des Variablennamens. Hierbei sind Groß- und Kleinschreibung zu beachten.

```
# Variable x1 und x2 als Ergebnis einer Addition bzw. Subtraktion
x1 <- 1 + 3
x2 <- 8 - 2
# gespeicherten Inhalt der Variablen x1 und x2 ausgeben
x1
x2
```

2.2 Datentypen

R unterscheidet zwischen nummerischen (*numeric*), alphabetischen (Zeichen, *character*) und logischen Datentypen. Zu den nummerischen Datentypen gehören zum Beispiel die zuvor erstellten Variablen *x1* und *x2*. Zeichen oder -ketten werden von R erkannt, indem Anführungszeichen bei der Wertzuweisung benutzt werden. Logische (Boolsche) Werte bestehen aus *TRUE* (wahr) oder *FALSE* (falsch).

```
# Zuweisung der Zeichenkette "Hallo" zur Variable x3
x3 <- "Hallo"
# Ausgabe von x3
x3
```

Variablenwerte sind überschreibbar. Bereits erstellte Variablen (auch anderer Datentypen) können dabei ebenfalls genutzt werden.

```
# Neuzuweisung eines Ergebnisse zu x2
x2 <- 2 * 5
x2
# x1 erhält den Wert von x2
x1 <- x2
x1
# x1 erhält den Wert von x3 (Änderung des Datentyps von x1)
x1 <- x3
```

```r
x1
# x2 erhält den Wert TRUE (Änderung des Datentyps)
x2 <- TRUE
x2
```

2.3 Objekttypen (Vektor und Matrix)

R beinhaltet eine Reihe von Objekttypen. Davon finden zum Beispiel Vektoren oder Matrizen häufig Anwendung. Vektoren sind eindimensionale Datenfelder und bieten die Möglichkeit, mehrere Einzelelemente in ein Objekt (Variable) zusammenzufassen. Bei der Zuweisung mehrerer Elemente zu einer Variable wird dabei **c()** (für *combine*) verwendet.

```r
# Speichern der Zahlen 3 bis 10 in den Vektor v
v <- c(3, 4, 5, 6, 7, 8, 9, 10)
v
```

Matrizen (**matrix()**) sind hingegen zweidimensionale Datenfelder des selben Datentyps, deren Größe durch ihre Anzahl von Zeilen (**nrow**) und Spalten (**ncol**) definiert wird. Es können auch Zeilen- und Spaltennamen vergeben werden.

```r
# Speichern der Zahlen 3 bis 10 (Vektor v) in die Matrix m
m <- matrix(data = v, ncol = 4, nrow = 2)
m
# Anzahl der Zeilen
nrow(m)
# Zeilen- und Spaltennamen vergeben
rownames(m) <- c("Zeile1", "Zeile2")
colnames(m) <- c("Spalte1", "Spalte2", "Spalte3", "Spalte4")
m
```

2.4 Zugriff auf Elemente

In dem Vektor v und der Matrix m sind nun mehrere Elemente enthalten, auf die auf ähnliche Weise zugegriffen werden kann.
Bei Vektoren reicht es aus, Indexwerte anzugeben, welche die Positionen der zu extrahierenden Elemente darstellen. Diese Werte stehen dabei in eckigen Klammern.

```r
# Extrahieren der Zahl 5 (dritte Position in Vektor v)
v[3]
# Extrahieren der Zahl 5, 7 und 10 (dritte, fünfte und achte Position in Vektor v)
v[c(3, 5, 8)]
```

Da Matrizen zweidimensionale Objekte sind, können hier je Dimension Indexwerte oder aber auch Zeilen-/Spaltennamen angegeben werden. Steht in einer Dimension kein Indexwert oder Zeilen-/Spaltenname, so werden alle Elemente dieser Dimension extrahiert.

```r
# Extrahieren der Zahl 5 (erste Zeile, zweite Spalte in Matrix m)
m[1, 2]
# Extrahieren der Zahlen 3 und 4 (erste Spalte)
m[, 1]
# Extrahieren der Zahlen 5, 7 und 9 (erste Zeile, letzten drei Spalten)
# Zugriff über Indizes
m[1, c(2, 3, 4)]
# Zugriff über Zeilen- und Spaltennamen
```

```r
m["Zeile1", c("Spalte2", "Spalte3", "Spalte4")]
# gemischter Zugriff
m[1, c("Spalte2", "Spalte3", "Spalte4")]
```

2.5 Objekttypen (Liste und *Data Frame*)

Listen (**list()**) sind Strukturen, welche verschiedene Daten- und Objekttypen zur selben Zeit beinhalten können.

```r
# Vektor v, Matrix m, TRUE und das Wort "Hallo" als eine Liste (ohne Namensgebung)
l <- list(v, m, TRUE, "Hallo")
l
# Extrahieren der Matrix m
l[[2]]
# nun eine Liste mit Namensgebung
l <- list(Vektor=v, Matrix=m, Wahrheitswert=TRUE, Zeichenkette="Hallo")
l
# nun zusätzliches Extrahieren der Matrix m mittels zugeordnetem Namen möglich
l$Matrix
# erste Zeile der Matrix m
l$Matrix[1,]
```

Bei *data frames* (**data.frame()**) handelt es sich um Strukturen, welche ähnlich einer Matrix aufgebaut sind und zur Speicherung von Datentabellen gedacht sind. Von einer Matrix unterscheidet sich dieser Objekttyp hauptsächlich darin, dass jede Spalte Elemente eines anderen Datentyps enthalten darf.

```r
# verschiedene Vektoren als "data.frame"-Objekt
zahlen <- c(1, 2.1, 0.4)
woerter <- c("Hallo", "Hello", "Bonjour")
wahrheitswerte <- c(TRUE, FALSE, TRUE)
df <- data.frame(Zahl=zahlen, Wort=woerter, Wahrheit=wahrheitswerte)
df
# Extrahieren der ersten Spalte
df$Zahl
```

3 Funktionen und Vergleichsoperatoren (Auswahl)

3.1 Integrierte Funktionen

In R sind zahlreiche fertige Funktionen bereits vorhanden. Sollten Funktionen aus einer bestimmten Bibliothek dennoch benötigt sein, so kann diese mit dem Befehl **install.packages(***Paketname***)** Pakete installiert und mit **library(***Paketname***)** geladen werden.
Eine Übersicht ausgewählter, bereits integrierter Funktionen ist in folgender Tabelle gegeben:

Funktionen	Beschreibung
length()	Anzahl der Elemente in einem Vektor
mean()	Mittelwert
sum()	Summe
prod()	Produkt
var()	Varianz
sd()	Standardabweichung
median()	Median
min(), max()	Minimum, Maximum
rnorm()	Erzeugen von normalverteilten Zufallszahlen
sort()	Ordnung von Elementen nach Größe
seq(m,n) oder m:n	Erzeugen einer nummerischen Sequenz von m bis n
rep(x,n)	Erzeugen von n Wiederholungen von Objekt x
plot(), boxplot(), hist(), barplot()	Erzeugung von: Streudiagramm, Boxplot, Histogramm, Barplot
read.csv	Einlesen von csv-Tabellen

Tabelle 1: Übersicht ausgewählter Funktionen

Anwendungsbeispiele hierfür sind:

```
# Länge der ersten Zeile der Matrix m
length(m[1, ])
# Vektor v neu definieren
v <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
# gleichbedeutend mit
v <- 1:9
v
# Länge des Vektors v
length(v)
# Mittelwert des ganzen Vektors v
mean(v)
# Mittelwert der ersten drei Werte in v
mean(v[1:3])
# Minimum
min(v)
```

3.2 Eigene Funktionen schreiben

Für einen übersichtlichen und effizienten Ablauf eines Programms ist es nahezu unumgänglich eigene Funktionen zu erstellen und zu verwenden. Besonders nützlich sind sie, wenn man ein und die selbe Berechnung mehrfach durchführen und den dafür vorgesehenen Quellcode nicht jedesmal neu eintippen muss. Der Befehl **function()** wird zur Funktionsdeklarierung genutzt.
Die bereits in R integrierte Mittelwertfunktion **mean()** kann zum Beispiel leicht nachgebaut werden. Die folgende Funktion **Mittelwert()** erwartet einen nummerischen Vektor **v** als Parameter, berechnet die Summe und teilt diese durch die Anzahl der Elemente. Das Ergebnis wird über den **return()**-Befehl zurückgegeben.

```
Mittelwert <- function(v){
  summe <- sum(v)
  mittelwert <- summe/length(v)
  return(mittelwert)
}
```

3.3 Vergleichsoperatoren

Zu den wichtigsten Vergleichoperatoren gehören >, >=, <, <= sowie == und ! =. Möchte man nun aus Vektor **v** Werte mit bestimmten Eigenschaften extrahieren, so können diese Operatoren beispielsweise wie folgt angewendet werden:

```
# Vektor v neu definieren
v <- -3:8
# Welche Werte in v sind größer 4? (Ausgabe Boolscher Werte)
v > 4
# Welche Werte in v sind größer 4? (Ausgabe der Indizes)
which(v > 4)
# Ausgabe der Elemente, die größer 4 sind (unter Verwendung der Indizes)
v[which(v > 4)]
# Ausgabe der Elemente, die größer 4 sind (unter Verwendung Boolscher Werte)
v[v > 4]
# An welcher Position befindet sich der Wert 2?
which(v == 2)
# Welche Positionen sind ungleich dem Wert 2?
which(v != 2)
# Alle Werte ungleich dem Wert 2 ausgeben
v[v != 2]
```

Unter Verwendung Boolscher Werte können Bedingungen auch miteinander geknüpft werden. Dabei kommen die Operatoren & (UND) und | (ODER) zum Einsatz.

```
# Alle Werte größer gleich 0 und kleiner 3
v[(v >= 0) & (v < 3)]
# Alle Werte kleiner 0 oder größer gleich 3
v[(v < 0) | (v >= 3)]
```

3.4 Hilfe-Funktion

Meistens ist es sehr nützlich zu erfahren, welche Parameter eine Funktion benötigt bzw. welche es zusätzlich noch gibt. So kann mit **help(***Funktionsname***)** die Funktionsweise sowie deren Nutzung nachgelesen werden.

```
# Beispiel für die read.csv-Funktion
help("read.csv")
```

4 Plot-Funktionen (Beispiele)

Ein Hauptmerkmal von R ist die Vielfalt an Plot-Möglichkeiten (deskriptive Statistik). Es existieren unter anderem das Streudiagramm **plot()**, das Säulendiagramm **barplot()**, das Histogramm **hist()** und der Boxplot **boxplot()**.

4.1 Streudiagramm

Möchte man beispielweise die Sinusfunktion darstellen, kann der **plot()**-Befehl zu diesem Zweck genutzt werden. Hierbei belaufen sich die Mindestangaben auf x- (Eingabewert) und y-Werte (berechnete Sinuswerte). Die x-Werte werden mittels **seq()** von 1 bis 10 und einem jeweiligen Abstand von 0.1 erzeugt. Die y-Werte werden dann mit **sin()** berechnet.

```
v <- seq(0,10,0.1)
plot(x = v, y = sin(v))
```

Bei Plot-Funktionen ist eine Vielzahl von grafischen Einstellungsmöglichkeiten gegeben. Dazu zählen beispielsweise die Darstellung der Daten als Punkte oder Linien. Auch die Änderung der Farbe sowie Achsenbeschriftungen/-skalierungen oder Überschriften gehören mit dazu.

```
# zeichne die Sinusfunktion als rote Linie mit einer Hauptüberschrift
# und anderen Achsenbeschriftungen
plot(x = v, y = sin(v), type = "l", col = "red",
     main = "Das ist eine Sinusfunktion",
     xlab = "Eingabewert", ylab = "Sinuswert")
```

4.2 Säulendiagramm

Im Folgenden wird eine Strichprobe **v** mit 50 zufällig, aber gleichverteilt generierten (**runif()**) und auf ganzzahlig abgerundeten (**floor()**) Werten betrachtet. Der Befehl **table()** zählt die Anzahl aller vorkommenden Werte, also deren Häufigkeiten. **barplot()** kann nun genutzt werden, um diese Häufigkeiten in Form eines Säulendiagramms darzustellen.

```
v <- floor(runif(n = 50, min = 0, max = 20))
haeufigkeiten <- table(v)
barplot(haeufigkeiten, col = colors(),
        ylab = "Häufigkeiten", xlab = "Wert")
```

4.3 Histogramm

Im Folgenden wird eine Strichprobe **v** mit 50000 zufällig, aber exponentialverteilt generierten (**rexp()**) Werten betrachtet. Mithilfe des **hist()**-Befehls kann nun genutzt werden, um diese Häufigkeiten in Form eines Histogramms darzustellen.

```
v <- rexp(n = 50000)
hist(v, breaks = 100, col = "green",
     ylab = "Häufigkeit", xlab = "Wert",
     main = "Histogramm über expontialverteilten Werten")
```

4.4 Boxplot

Für eine Ermittlung einer möglichen Verteilung der vorliegenden Daten und möglicher Ausreißer eignen sich beispielsweise sogenannte Boxplots. In diesem Beispiel sollen die Werte 0, 10 bis 40 und 70 untersucht werden.

```
v <- c(0, 10:40, 70)
boxplot(v)
```

5 Übungen

Im Folgenden sollen einige Übungen durchgeführt werden. Manuelle Eingaben von Werten, wie z.B. die Erstellung eines Vektors mit Werten von 1 bis 10, sind schnell eingetippt. Jedoch sollte weitestgehend mit bereits erwähnten Funktionen zur Generierung solcher Werte gearbeitet werden.
Die **help()**-Funktion ist stets zu empfehlen, wenn nähere Informationen zu einer Funktion benötigt werden.

5.1 Vektor erstellen und sortieren

Erstellen Sie einen Vektor **v**, welcher nur die Zahlen zwischen -10 und 10 sowie zwischen 20 und 30 enthält. Anschließend soll dieser aufwärts und danach abwärts sortiert werden.

5.2 Matrix erstellen und filtern

Nun soll eine Matrix **m** bestehend aus den Werten von Vektor **v** gebildet werden. Zudem soll diese eine Spaltenanzahl von 4 besitzen.
Des Weiteren sollen die Indizes von **m** ausgegeben werden, wo die Werte kleiner als 0 oder größer als 20 sind.

5.3 Eigene Funktion und Zufallszahlen

Schreiben Sie eine Funktion **f**, die als Parameter eine Zahl **n** entgegennimmt. Des Weiteren soll diese Funktion einen Vektor **v** mit n zufällig gezogenenen und normalverteilten Werten erzeugen. Diese Werte sollen sich alle um den Mittelwert 10 bewegen. Anschließend ist ein Histogramm der Werte aus Vektor **v** mit Parameter *break = 100* zu erstellen.
Rufen Sie diese Funktion jeweils mit $n = 10$, $n = 100$, $n = 1000$, $n = 10000$, $n = 100000$, $n = 1000000$ auf. Was fällt Ihnen hier auf?

5.4 Testdaten

Die letzte Übung handelt um den Umgang mit einem Testdatensatz. Dieser beinhaltet chemische Verschiebungen aus 13C-NMR-Experimenten. Bei der ersten Spalte handelt es sich dabei um Verschiebungen von Kohlenstoffen, die eine Einfachbindung mit einem Sauerstoff aufweisen (C-O). In der zweiten Spalte geht es um Verschiebungen mit einer Doppelbindung (C=O).
Laden Sie zunächst den Datensatz **~/Downloads/testdaten.csv** über den Befehl **read.csv()** ein und speichern Sie diesen in eine Variable *data*. Achten darauf, dass der Parameter *header = TRUE* gesetzt wird. Erstellen Sie nun für jede Bindungsart (Spalte) ein Histogramm, wobei der Parameter *break = 100* gesetzt wird.
Anschließend soll ein Boxplot für *data* erstellt und dieser gleichzeitg in die Variable q gespeichert werden. Über **q$stats** kann nun auf Minimum und Maximum der im Boxplot dargestellten Bereiche zugegriffen werden, wieder je Bindungsart. Nutzen Sie diese Minima und Maxima, um die Indizes der außerhalb der Antennenbereiche liegenden, potenziellen Ausreißer in *data* je Bindungsart ausgeben zu lassen.