R

Einführung durch angewandte Statistik

2., aktualisierte Auflage

Reinhold Hatzinger
Kurt Hornik
Herbert Nagel
Marco J. Maier

EXTRAS ONLINE

ALWAYS LEARNING

PEARSON

Darstellung des .txt-Formats in Editor oder aber in Textverarbeitungsprogrammen (wie z. B. Word) besser ist.

Alles bisher Dargestellte gilt in analoger Weise für LibreOffice Calc.

Das Einlesen von Dateien im .csv-Format in R erfolgt mittels der Befehle read.csv2() oder read.csv(). Die erste Version ist für Dateien, die aus einer deutschen Excel-Version stammen, die zweite für die englische Version. Abgesehen davon, dass in der deutschen Excel-Version ein Komma als Dezimalzeichen und nicht der in R benötigte Punkt verwendet wird, unterscheiden sich die Versionen noch bezüglich einiger anderer Eigenschaften. Wir nehmen an, dass die Fragebogendaten in der Datei fragebogen.csv aus einer deutschen Version stammen, und lesen sie in R so ein:

```
> fragdat1 <- read.csv2("fragebogen.csv", header = TRUE)
> head(fragdat1)
```

R

```
  id sex lalt gross mon date entsch proj i1 i2 i3 i4 i5
1 11   1    2   173 266    4      3    2  2  3  3  2  2
2 16   1    3   166 241    5      4    1  4  2  3  1  1
3 17   7    2   178 231    3      4    2  2  1  3  2  4
4 18   1    3   154 265    3      5    2  5  3  2  4  1
5 19   1    1   164 225    2      3    2  1  4  2  2  3
6 20   2    1   389 229    4      1    1  5  2  2  1  4
```

Das erste Argument ist der Dateiname (ggf. inkl. Pfadangabe), das zweite, *header* = TRUE, gibt an, ob in der ersten Zeile der Datei die Variablennamen angegeben sind (dies ist auch die Voreinstellung). Sollte das nicht der Fall sein, muss man *header* = FALSE angeben.

Für tabulatorgetrennte Dateien im .txt-Format verwendet man den Befehl read.table().

```
> fragdat2 <- read.table("fragebogen.txt", header = TRUE, sep = "\t",
+     dec = ",")
> head(fragdat2)
```

R

```
  id sex lalt gross mon date entsch proj i1 i2 i3 i4 i5
1 11   1    2   173 266    4      3    2  2  3  3  2  2
2 16   1    3   166 241    5      4    1  4  2  3  1  1
3 17   7    2   178 231    3      4    2  2  1  3  2  4
4 18   1    3   154 265    3      5    2  5  3  2  4  1
5 19   1    1   164 225    2      3    2  1  4  2  2  3
6 20   2    1   389 229    4      1    1  5  2  2  1  4
```

Die beiden Optionen *sep* und *dec* bedeuten hierbei Folgendes: *sep* gibt das Zeichen an, durch das die einzelnen Werte voneinander getrennt sind, *sep* = "\t" bedeutet Tabulator. *dec* spezifiziert das Zeichen für die Dezimalstelle, bei deutschen Versionen ist dies das Komma bzw. ",".

Bei beiden Varianten ist Folgendes zu beachten:

- Die Funktion read.csv2() (wie auch die englische Variante read.csv()) erzeugt ein Data Frame.
- Beim Einlesen werden character-Variablen standardmäßig automatisch in Faktoren umgewandelt.

- Je nachdem, ob die Variablennamen in der ersten Zeile stehen, muss man *header* = TRUE oder *header* = FALSE spezifizieren. Gibt es keine Variablennamen, dann werden die Variablen im erzeugten Dataframe mit den Namen V1, V2 etc. versehen.

Wenn Sie R unter Windows aufrufen, dann befinden Sie sich standardmäßig im Benutzerverzeichnis Eigene Dokumente (Documents), d. h., wenn Sie eine Datei einlesen wollen, erwartet R, dass die Datei in diesem Verzeichnis gespeichert ist. Meistens verwendet man aber ein anderes Verzeichnis. Es gibt nun in R die Möglichkeit, statt des Dateinamens die Funktion file.choose() anzugeben. Es öffnet sich dann ein Dialogfenster, in dem man wie in Windows gewohnt, die zu lesende Datei aussuchen kann. Alternative Möglichkeiten des Zugriffs auf Dateien in anderen Verzeichnispfaden zeigt Abschnitt 5.1.2.

Sie können auch Dateien aus dem Internet einlesen. Statt des Dateinamens geben Sie dann einfach die Internetadresse als Zeichenkette in der Funktion url() an, z. B. können Sie einen kleinen Datensatz von der Statlib-Webseite so einlesen:

```
> read.table(url("http://lib.stat.cmu.edu/datasets/Andrews/T02.1"),
+     header = FALSE)
```
R

5.5.2 Dateien aus anderen Statistikpaketen (z. B. SPSS)

Es gibt in R unter Verwendung des Package foreign (R Core Team, 2013b) die Möglichkeit, Systemdateien aus anderen Statistikpaketen (wie z. B. SPSS, Systat, SAS oder Stata) zu lesen. Wir wollen hier kurz auf SPSS eingehen. Die entsprechende Funktion ist read.spss(). Neben dem Dateinamen der SPSS-Datei mit der Endung .sav sollte die Spezifikation folgender Argumente in Betracht gezogen werden.

- *to.data.frame* = TRUE ist die wichtigste Option. Wenn sie nicht angegeben wird, wird kein Data Frame, sondern eine Liste erzeugt.
- *use.value.labels* = TRUE: Wenn diese Option mit TRUE spezifiziert wird, dann verwendet R die Werte-Labels, wie sie in SPSS definiert sind, und wandelt die entsprechenden Variablen in Faktoren um.
- *use.missings* = TRUE sollte gesetzt werden, damit sog. benutzerdefinierte fehlende Werte (engl. *user-defined missings*), die man in SPSS definieren kann, auch in R korrekt als fehlende Werte gekennzeichnet werden.

Ein Beispiel für das Lesen einer SPSS-Datei, nachdem man das Package foreign geladen hat (siehe auch Abschnitt 3.1.5), ist:

```
> library("foreign")
> frag3 <- read.spss("file.sav", to.data.frame = TRUE, use.value.labels = TRUE,
+     use.missings = TRUE)
```
R

Alternativ kann man auch wie oben in Excel (Abschnitt 5.5.1) vorgehen und zunächst in SPSS ein .csv oder ein tabulatorgetrenntes File erstellen und dieses dann ent-

sprechend einlesen. Manchmal, besonders bei sehr komplexen SPSS-Dateien, erweist sich diese Vorgehensweise als stabiler.

5.5.3 Direktes Kopieren – Einfügen über die Zwischenablage

Eine besonders einfache Möglichkeit, Dateien aus anderen Quellen nach R zu bringen, liefert die Verwendung der **ZWISCHENABLAGE** (engl. *clipboard*). Sie können hierbei in SPSS oder Excel, aber auch in anderen Programmen (wie etwa auf einer Webseite oder manchmal in einem PDF-File), in denen die Daten in Tabellenform dargestellt sind, so vorgehen:

- Zunächst markieren Sie den benötigten Bereich mit der Maus und verwenden dann Kopieren (Menü, Rechtsklick oder Strg + V). Die Daten sind jetzt in der Zwischenablage.
- In R verwenden Sie die Funktion `read.table()` genauso wie oben beschrieben (also wenn nötig mit den Optionen *`header`* bzw. *`dec`*), Sie schreiben aber statt des Dateinamens `"clipboard"`.

Als Beispiel wollen wir die Datei von der Statlib-Webseite, die wir vorher über die Internetadresse eingelesen haben, nun direkt kopieren. In einem Internetbrowser gehen wir zur Adresse *http://lib.stat.cmu.edu/datasets/Andrews/T02.1*, markieren dort die Daten und verwenden z. B. die Tastenkombination Strg + C zum Kopieren. In R schreiben wir:

```
> andrews <- read.table("clipboard", header = FALSE)
```
R

und haben die Daten dann im Data Frame `andrews` zur Verfügung.

Will man einen Vektor über die Zwischenablage einlesen, kann man den `scan()` Befehl verwenden. Auch hier kann man einfach den Dateinamen durch `"clipboard"` ersetzen, wobei man jedoch darauf achten muss, was in der Zwischenablage vorhanden ist. Standardmäßig erwartet `scan()` Zahlen und erzeugt einen Fehler, wenn Text eingefügt wird. Will man also Text einfügen, muss man das Argument *`what`* auf `"character"` setzen.

```
> scan_zahlen <- scan("clipboard")
> scan_text <- scan("clipboard", what = "character")
```
R

Der `scan()` Befehl eignet sich auch zur direkten Eingabe von Werten über die Tastatur. Ruft man die Funktion ohne Argumente auf, erscheint anstatt der Eingabeaufforderung `1:`. Hier kann man nun einzelne Werte eintippen und nach jedem Wert auf ↵ (Eingabe) drücken oder mehrere durch mindestens ein Leerzeichen getrennte Zahlen eingeben. Sie können dies aber auch mischen, d. h. ein paar Werte eingeben, dann auf die Eingabe-Taste drücken usw. Wenn Sie fertig sind, müssen Sie nochmals die Eingabe-Taste betätigen, bis eine Information über die Anzahl der eingelesenen Elemente ausgegeben wird und Sie wieder die Eingabeaufforderung sehen. Ein Beispiel ist in ▶ Abbildung 5.21 zu sehen.

Abbildung 5.21: Ausschnitt aus der R Console nach Eintippen einiger Zahlen, die mit `scan()` dem Vektor `x` zugewiesen werden

5.5.4 Schreiben von Dateien

In Abschnitt 4.2.2 haben wir schon den Befehl `save()` zum Speichern eines Data Frame in eine binäre Datei kennengelernt. Will man Daten in für Menschen lesbarer Form in eine Datei schreiben, also in einer Form, wie man sie mit `read.table()` wieder einlesen kann, dann verwendet man den Befehl `write.table()`.

Im einfachsten Fall kann man ein Data Frame, z. B. den vorher eingelesenen Datensatz `andrews`, mittels

```
> write.table(andrews, file = "andrews.txt", col.names = TRUE,
+     row.names = FALSE)
```
R

abspeichern. Das heißt, man erzeugt hier eine Datei `andrews.txt`, die in Windows mit dem Programm Editor geöffnet werden kann. Die Option *`row.names = FALSE`* unterdrückt das Abspeichern der Zeilennamen, die in diesem Fall nur eine Nummerierung der Zeilen wären, da wir keine Zeilennamen spezifiziert haben. Wenn Sie Zeilennamen abspeichern, kann es dazu kommen, dass diese keinen eigenen Spaltennamen bekommen und somit die Spaltennamen nicht mehr mit den tatsächlichen Inhalten übereinstimmen. Es ist daher ratsam, diese prinzipiell nicht mit abzuspeichern. Hätten wir die Option *`col.names = FALSE`* gewählt, dann wären die Variablennamen unterdrückt worden.

Mit einer weiteren Option (analog zu `read.table()`), nämlich *`sep = "\t"`*, können wir die Ausgabedatei so gestalten, dass man sie in Excel als tabulatorgetrennte Datei einlesen kann. Arbeitet man mit einer deutschen Version von Excel, so muss man zusätzlich den Dezimaltrenner auf ein Komma setzen, d. h. *`dec = ","`*, da sonst ein Punkt verwendet wird.

Will man eine `.csv`-Datei (für deutsche Excel-Versionen) schreiben, dann muss man das Trennzeichen Strichpunkt mittels *`sep = ";"`* und das Dezimalzeichen Komma mittels *`sep = ","`* spezifizieren. Der entsprechende R-Befehl wäre also

```
> write.table(andrews, file = "andrews.csv", row.names = FALSE,
+     sep = ";", dec = ",")
```
R

Die so erzeugte Datei `andrews.csv` können Sie durch Anklicken im Windows-Explorer direkt in Excel öffnen.

Eine etwas bequemere Variante bieten die Befehle `write.csv()` bzw. `write.csv2()`, bei denen man Trenn- und Dezimalzeichen nicht spezifizieren muss und die wie bei `read.csv()` bzw. `read.csv2()` für englische bzw. deutsche Programmvarianten von Excel funktionieren.

5.6 Statistische Funktionen

Eine der größten Stärken von R liegt in der umfassenden Verfügbarkeit statistischer Funktionen. Bei einer Software für Statistik ist das wenig verwunderlich, jedoch punktet R durch die Einfachheit und Vielfältigkeit der Anwendung im Vergleich zu anderen Paketen. Im Folgenden sollen kurz die wichtigsten Funktionalitäten – nämlich zur Simulation, Berechnung der Dichte, Quantile und kumulativer Wahrscheinlichkeiten – erläutert werden. Die Beispiele werden mit der Normalverteilung durchgeführt – andere Verteilungen werden am Ende aufgelistet und können analog verwendet werden.

5.6.1 Die drei Funktionen statistischer Verteilungen

Die meisten statistischen Funktionen, die in R implementiert sind, haben einen Namen wie `*norm` (siehe `?Normal`), wobei statt des $*$ ein Buchstabe für eine spezielle Funktion der Verteilung hinzugefügt wird.

Dichte (d)

Die Wahrscheinlichkeitsdichtefunktion oder Dichtefunktion (*probability density function*, PDF) erhält man, indem man ein `d` vor den Verteilungsnamen stellt, beispielsweise `dnorm()`. Diese Funktion hat die Argumente `dnorm(x, mean = 0, sd = 1, log = FALSE)`, das `x` ist ein Wert, für den die Dichte einer Standardnormalverteilung $\mathcal{N}(\mu = 0, \sigma = 1)$ berechnet wird. Mit dem Argument `log` kann man die logarithmierte Dichte und somit die Log-Likelihood berechnen. Wenn man die Werte im Intervall $[-5, +5]$ plottet, erhält man Plots wie in ► Abbildung 5.22.

```
> par(mfrow = c(1, 2))
> curve(dnorm(x), -5, 5, xlab = "x-Wert", ylab = "Dichte", main = "dnorm(x)")
> curve(dnorm(x, log = TRUE), -5, 5, xlab = "x-Wert", ylab = "log(Dichte)",
+     main = "dnorm(x, log = TRUE)")
```

R

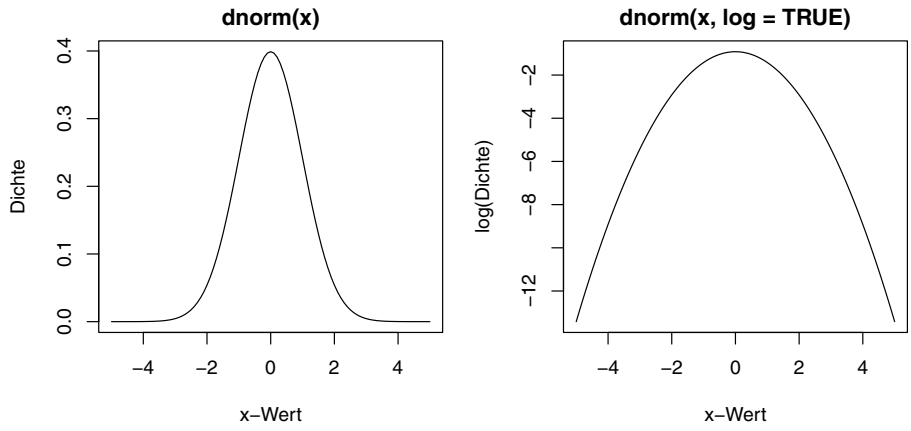

Abbildung 5.22: (Logarithmierte) Dichtefunktion einer Standardnormalverteilung

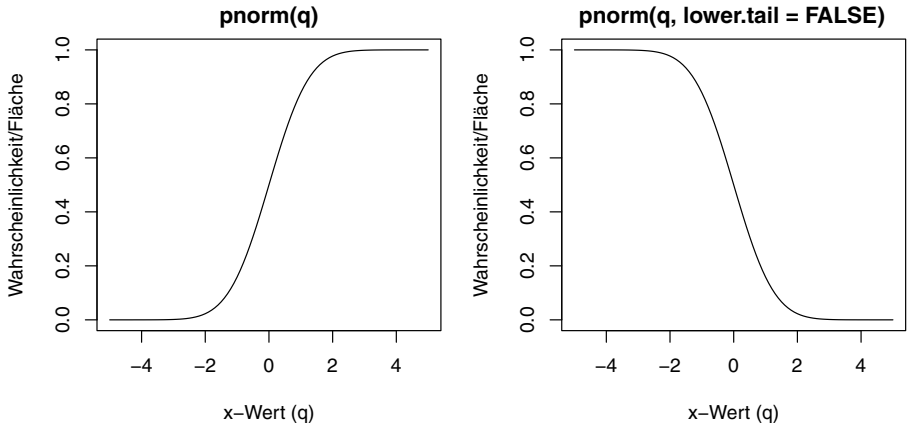

Abbildung 5.23: Kumulative Wahrscheinlichkeitsfunktionen von links und rechts kommend

Kumulative Dichte (p)

Die kumulative Dichtefunktion oder Verteilungsfunktion (*cumulative probability distribution*, CDF) erhält man, indem man ein `p` vor den Verteilungsnamen setzt. `pnorm()` hat die Argumente *`mean`* und *`sd`*, die Mittelwert und Standardabweichung der Verteilung bestimmen. Das Resultat dieser Funktion ist die kumulative Wahrscheinlichkeit von links bis zu einem Punkt *q* – setzt man *`lower.tail = FALSE`* erhält man die Gegenwahrscheinlichkeit, also von rechts kommend bis *q*. Folgender Code erzeugt die Graphen in ▶ Abbildung 5.23.

```
> par(mfrow = c(1, 2))
> curve(pnorm(x), -5, 5, xlab = "x-Wert (q)", ylab = "Wahrscheinlichkeit/Fläche",
+     main = "pnorm(q)")
> curve(pnorm(x, lower.tail = FALSE), -5, 5, xlab = "x-Wert (q)",
+     ylab = "Wahrscheinlichkeit/Fläche", main = "pnorm(q, lower.tail = FALSE)")
```

R

Quantilfunktion (q)

Die Quantilfunktion ist (im einfachsten Fall, bei stetigen Verteilungen) die Umkehrfunktion der kumulativen Dichtefunktion und wird mit dem Präfix `q` spezifiziert. `qnorm()` gibt den Wert zurück, unter dem die Fläche *p* liegt – mit *`lower.tail = FALSE`* erhält man das Quantil, über dem *p* liegt. Diese Funktion ist wichtig, wenn man beispielsweise kritische Grenzen für ein Konfidenzintervall sucht: 95% Konfidenz erhält man, indem man links und rechts 2.5% abschneidet, d. h., bei einer Standardnormalverteilung erhält man die bekannten Werte:

```
> qnorm(c(0.025, 0.975))
```

R

```
[1] -1.96  1.96
```

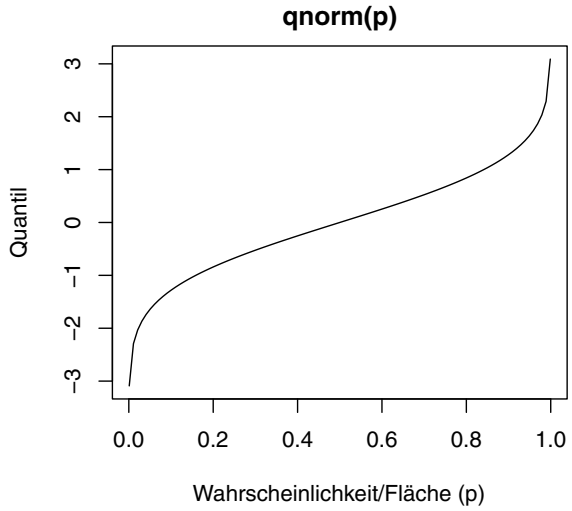

Abbildung 5.24: Quantilfunktion der Standardnormalverteilung

Folgender Code generiert die Grafik in ► Abbildung 5.24.

```
> curve(qnorm(x), 0.001, 0.999, xlab = "Wahrscheinlichkeit/Fläche (p)",
+     ylab = "Quantil", main = "qnorm(p)")
```

R

Zusammenhänge zwischen den Funktionen

Die grundlegende Funktion $f(x \mid \theta)$ jeder statistischen Verteilung ist die Dichtefunktion bei stetigen Variablen (bzw. die Wahrscheinlichkeitsfunktion bei diskreten Variablen). Diese beruht auf den Daten x und den Funktionsparametern θ. Um die kumulative Dichte (bzw. Masse) zu erhalten, muss man die Funktion bis zu einem gewissen Datenpunkt integrieren (bzw. summieren). Will man wissen, wo genau der Punkt ist, unter dem ein gewisser Flächenanteil liegt, benötigt man die Quantilfunktion, die ihrerseits die Umkehrfunktion der kumulativen Dichte/Masse ist.

5.6.2 Erzeugen von (Pseudo-)Zufallszahlen

Eine besonders mächtige Funktionalität von R ist die Erzeugung von Pseudozufallszahlen (*pseudo random number generation*, pRNG). Bei einer Normalverteilung stellt man einfach `r` vor den Funktionsnamen, so kann man mit der Funktion `rnorm()` normalverteilte Zufallszahlen erzeugen. Will man beispielsweise 1000 Daten, die $\mathcal{N}(\mu = 1, \sigma = 2)$ verteilt sind, kann man diese mit

```
> set.seed(5)
> normal1000 <- rnorm(1000, mean = 1, sd = 2)
```

R

erzeugen und eine Berechnung des Mittelwerts und der Standardabweichung zeigt, dass die Ergebnisse sehr knapp an den Simulationsparametern liegen

```
> c(mean(normal1000), sd(normal1000))
```
R

```
[1] 1.0348 2.0240
```

Die verwendete Funktion `set.seed()` ist gleichzeitig die Antwort, warum eingangs von *Pseudo*zufallszahlen gesprochen wurde. Natürlich sind Computer deterministische Maschinen und es kommen (mehr oder minder) ausgefeilte Algorithmen zum Einsatz, die möglichst „gute“ Zufallszahlen erzeugen. Der Vorteil, dass der Erzeugungsmechanismus einer Formel folgt, ist, dass man Simulationen reproduzierbar machen kann, und der Schlüssel dazu ist der sog. *seed*. Das ist ein (ganzzahliger) „Startwert“ für den Zufallszahlengenerator, d. h., wenn Sie die zwei Zeilen oben eingeben, sollten Sie im Großen und Ganzen dieselben Zahlen mit kleinen Abweichungen je nach Prozessor etc. erhalten.

5.6.3 Verfügbare Verteilungsfunktionen

► Tabelle 5.1 enthält eine kleine Liste gängiger Verteilungen. Eine umfassendere Liste von Verteilungen, die in Packages implementiert sind, findet sich hier: *http://cran.r-project.org/web/views/Distributions.html*

Tabelle 5.1: Einige Verteilungen in R (`*` wird für die gewünschte Funktionalität durch `d`, `p`, `q` oder `r` ersetzt)

Verteilungen	
`*beta()`	Betaverteilung
`*binom()`	Binomialverteilung
`*cauchy()`	Cauchy-Verteilung
`*chisq()`	χ^2-Verteilung
`*exp()`	Exponentialverteilung
`*f()`	*F*-Verteilung
`*gamma()`	Gammaverteilung
`*geom()`	Geometrische Verteilung
`*hyper()`	Hypergeometrische Verteilung
`*lnorm()`	Log-Normalverteilung
`*multinom()`	Multinomialverteilung
`*nbinom()`	Negativ-Binomialverteilung
`*norm()`	Normalverteilung
`*pois()`	Poissonverteilung
`*t()`	*t*-Verteilung
`*unif()`	Gleichverteilung
`*weibull()`	Weibull-Verteilung

5.7 Rechnen mit Matrizen

Das Rechnen mit Matrizen ist in der Statistik immens wichtig, daher sind auch entsprechende Funktionen in R implementiert, wobei wir auf die wichtigsten kurz eingehen wollen.

5.7.1 Transponieren

Die Funktion `t()` transponiert eine Matrix (bzw. ein Data Frame), d. h., die Elemente werden so umgeordnet, dass aus Spalten Zeilen werden. Folgendes Beispiel sollte diese Funktionalität demonstrieren.

```
> (A <- matrix(1:6, 2))
```
R

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
> t(A)
```
R

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

Wir sehen, dass A zuerst die Dimension 2×3 und die transponierte Matrix A^T (oder A') dann 3×2 hat.

5.7.2 Matrixmultiplikation

Ein grundlegendes Konzept der Matrixalgebra ist die Multiplikation zweier Matrizen. Angenommen, wir haben zwei Matrizen $\boldsymbol{A}$ und $\boldsymbol{B}$ mit den Dimensionen $n \times m$ und $m \times k$, so können wir diese nur $\boldsymbol{M} = \boldsymbol{AB}$ multiplizieren, da die Spaltenanzahl der ersten Matrix der Zeilenanzahl der anderen entsprechen muss. Das Ergebnis dieser Multiplikation ist dann eine $n \times k$ Matrix. In R verwenden wir den Operator %*% und schreiben `A %*% B`. Als Beispiel verwenden wir `A` von vorhin und erstellen noch eine 3×2 Matrix `B`.

```
> (B <- matrix(11:16, 3))
```
R

```
     [,1] [,2]
[1,]   11   14
[2,]   12   15
[3,]   13   16
```

```
> M <- A %*% B
> M
```

R

```
     [,1] [,2]
[1,]  112  139
[2,]  148  184
```

Wir sehen, dass die entstandene Matrix `M` nun 2×2 ist.

Für die Spezialfälle $X^T Y$ (`t(X) %*% Y`) und XY^T (`X %*% t(Y)`) gibt es die Funktionen `crossprod()` und `tcrossprod()`, die etwas effizienter und schneller als die entsprechenden Ausdrücke mit `t()` und `%*%` arbeiten.

5.7.3 Matrixinversion

Quadratische Matrizen mit der Dimension $n \times n$ (also mit gleich vielen Zeilen wie Spalten) kann man unter bestimmten Bedingungen mit der Funktion `solve()` invertieren. Die invertierte Matrix $\boldsymbol{M}^{-1}$ hat die Eigenschaft, dass die Multiplikation $\boldsymbol{M}\boldsymbol{M}^{-1} = \boldsymbol{M}^{-1}\boldsymbol{M} = \boldsymbol{I}$ eine Einheitsmatrix (siehe nächster Abschnitt) ergibt.

```
> Mminus <- solve(M)
> round(M %*% Mminus, 5)
```

R

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

5.7.4 Lösen von Gleichungssystemen

Allgemein ist `solve(Y, Z)` eine Funktion zum Lösen von Gleichungssystemen der Form $\boldsymbol{Y}\boldsymbol{X} = \boldsymbol{Z}$. Gibt man nur eine quadratische Matrix an, wird für `Z` eine Einheitsmatrix angenommen, wodurch als Ergebnis $\boldsymbol{X}$ die Inverse berechnet wird.

Als Beispiel wollen wir folgendes Gleichungssystem lösen:

$$\underbrace{\begin{pmatrix} 5 & 2 & 3 \\ 9 & 8 & 1 \\ 7 & 6 & 4 \end{pmatrix}}_{\boldsymbol{Y}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{\boldsymbol{X}} = \underbrace{\begin{pmatrix} 24.5 \\ 48.5 \\ 43.5 \end{pmatrix}}_{\boldsymbol{Z}}$$

Wir definieren also die Matrix `Y`. `Z` kann auch als einfache Folge angegeben werden.

```
> (Y <- matrix(c(5, 9, 7, 2, 8, 6, 3, 1, 4), nrow = 3))
```

R

```
     [,1] [,2] [,3]
[1,]    5    2    3
[2,]    9    8    1
[3,]    7    6    4
```

```
> Z <- c(24.5, 48.5, 43.5)
```

Wir lösen das Gleichungssystem mit `solve()`, weisen die Lösung dem Objekt `loesung` zu und geben dieses aus.

```
> (loesung <- solve(Y, Z))
```

```
[1] 2.5 3.0 2.0
```

Diese Lösung kann man überprüfen, da $\boldsymbol{YX}$ wiederum $\boldsymbol{Z}$ ergeben muss, was hier der Fall ist.

```
> Y %*% loesung
```

```
     [,1]
[1,] 24.5
[2,] 48.5
[3,] 43.5
```

5.7.5 Spezielle Funktionen für quadratische Matrizen

Quadratische Matrizen (Dimensionen $n \times n$) treten in der Statistik z. B. als Kovarianz- oder Korrelationsmatrizen auf. Bei Ersteren ist die Hauptdiagonale besonders interessant, da hier die Varianzen enthalten sind. Die Funktion `diag()` bietet mehrere Funktionen (siehe auch `?diag`). Unter anderem kann man damit die Elemente der Hauptdiagonale extrahieren

```
> diag(M)
```

```
[1] 112 184
```

aber auch ersetzen

```
> diag(M) <- c(1, 2)
> M
```

```
     [,1] [,2]
[1,]    1  139
[2,]  148    2
```

Weiters kann man Einheitsmatrizen (eine quadratische Matrix mit Einsen in der Hauptdiagonale und Nullen außerhalb davon) erstellen, indem man eine ganze Zahl angibt

Copyright