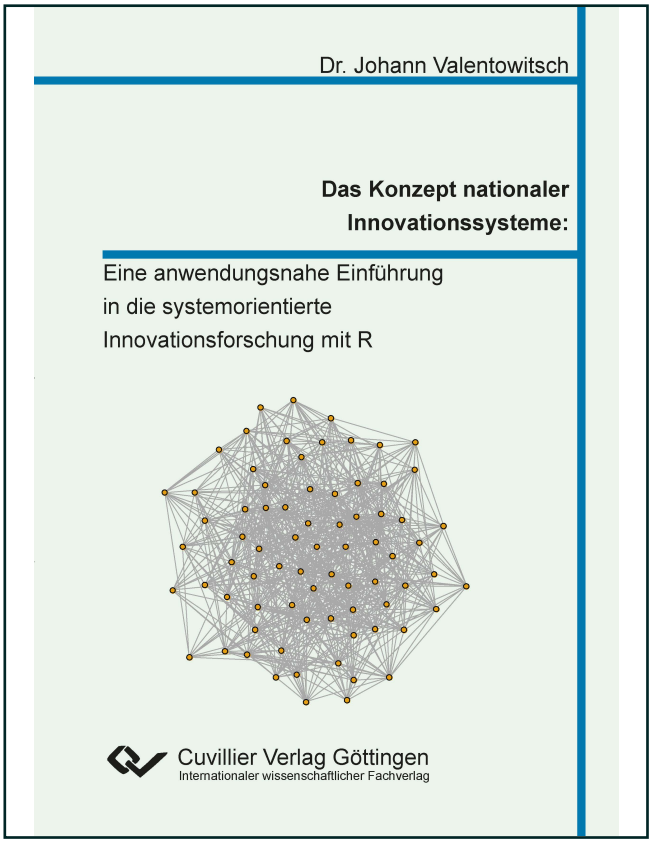
Johann Valentowitsch (Autor)

Das Konzept nationaler Innovationssysteme:

Eine anwendungsnahe Einführung in die systemorientierte Innovationsforschung mit R



https://cuvillier.de/de/shop/publications/8294

II. Eine Kurzeinführung in R

Die Analysesoftware R gehört heute zu den mit Abstand wichtigsten Statistik-Programmen in der angewandten empirischen Forschung. Seine Anfänge nahm das Programm bereits in den 1970er Jahren. Damals entwickelte ein von John Chambers geführtes Team in den zu AT&T gehörenden Bell Laboratories eine Programmiersprache, die speziell für simulationsbasierte Anwendungen optimiert und für statistische Analysezwecke konzipiert wurde. Heute ist die Sprache unter Fachleuten als die Programmiersprache S bekannt. Ihre Blütezeit hatte S vor allem in den späten 1980er Jahren als sie unter dem Markennamen S-Plus erfolgreich kommerzialisiert wurde. Durch die Kommerzialisierung von S-Plus wurde die Nutzung der zugrundeliegenden Programmiersprache für statistische Anwendungen allerdings lizenz- und gebührenpflichtig. Daher entstand in den frühen 1990er Jahren als Gegenentwurf zum kostenpflichtigen S-Plus Paket die auf Open-Source-Basis entwickelte Software R. Namensgebend für die Software waren die beiden Vornamen ihrer Entwickler, Ross Ihaka und Robert Gentleman. Beide waren zu jener Zeit Statistiker an der Universität von Auckland und bemängelten die lizenzrechtlichen Hürden von S-Plus, die ihnen das Offenlegen von Code-Fragmenten in den Vorlesungen und Übungen an der Universität erschwerten. Um ihre Probleme im akademischen Alltag zu lösen, entwickelten Ihaka und Gentleman daher noch im Jahr 1992 eine sehr frühe Version von R, die sich in ihren Grundzügen noch stark an der alten Programmiersprache S orientierte. Seitdem hat sich die neue Software aber konsequent weiterentwickelt, so dass R uns S heute zwar konzeptionell verwandt, jedoch lange nicht mehr identisch sind. Die Entwickler selbst bezeichnen R dabei nicht als eine Programmiersprache, sondern sprechen in Bezug auf die Software vielmehr von einer offenen Umgebung, in die unterschiedliche statistische Programme und Anwendungen eingebunden werden können. Dieses Selbstverständnis spiegelt das eigentliche Wesen der Software recht treffend wieder, weil R tatsächlich nicht als eine reine Programmiersprache verstanden werden sollte. Zwar können in R eigene Programme geschrieben und neue Pakete entwickelt werden, die große Stärke von R liegt aber eindeutig in der statistischen Analyse verortet. Für statistische Analysezwecke stellt die Software dabei eine ganze Fülle unterschiedlichster Verfahren bereit, die sich alle vortrefflich zum Aufspüren, Überprüfen und Visualisieren von auffälligen Mustern und Strukturen in den Daten eignen.

Im Unterschied zu vielen anderen Statistik-Programmen wie SPSS oder SAS enthält das Basis-Paket von R keine menü-geführte Bedienoberfläche. Die Befehlseingabe erfolgt in R daher ähnlich wie bei MS-DOS direkt über eine Kommando-Konsole. Was auf den ersten Blick für den bequemen User als Nachteil erscheinen mag, erweist sich bei genauerer Betrachtung als eine große Stärke, durch die gerade für Statistikneulinge substantielle Vorteile entstehen können. Das Fehlen einer menü-basierten Oberfläche erfordert nämlich eine viel intensivere Auseinandersetzung mit den zugrundeliegenden Funktionen und fördert somit das Verständnis von statistischen Sachverhalten. Ein ahnungsloses Durchklicken durch vordefinierte Menüs ist bei R also grundsätzlich nicht möglich. Das Fehlen einer menügeführten Oberfläche bringt jedoch nicht ausschließlich didaktische Vorteile mit sich. So erhöht sich durch die manuelle Eingabe von Befehlen unter anderem auch die Flexibilität bei der Datenanalyse, weil jeder einzelne Baustein im Code modifiziert und den eigenen Wünschen entsprechend angepasst werden kann. R-Nutzer finden sich also grundsätzlich in einer sehr bequemen Lage wieder: Sie können einerseits auf vordefinierte Anwendungen zurückgreifen, gleichzeitig können Sie diese aber auch jederzeit eigenständig modifizieren und erweitern, so dass am Ende stets passgenaue Lösungen realisiert werden können. Ein weiterer

Vorteil der R-Nutzung liegt in der Transparenz des Analysecodes begründet. Wer mit R statistische Funktionen programmiert, behält stets den Überblick über seine Arbeit und sieht genau, welche Berechnungsprozesse im Hintergrund ablaufen und wie sich diese schlussendlich auf den finalen Output auswirken.

Obwohl R unter Datenanalytikern durchaus mit anfänglichen Akzeptanzbarrieren zu kämpfen hatte und sich gegenüber anderen Programmen wie STATA, SAS oder SPSS durchsetzen musste, erfreut sich die Software heute unter Fachleuten einer immer größer werdenden Beliebtheit. So nimmt beispielsweise der Anteil aktiver R-Nutzer unter den als Data-Scientists bezeichneten Spezialisten seit Jahren kontinuierlich zu (vgl. Studie von Rexer Analytics 2015). Den Ergebnissen einer aktuellen Studie zur Folge, die unter mehr als tausend Fachleuten durchgeführt wurde, zählt R sogar zu den beliebtesten Statistikprogrammen überhaupt (vgl. Umfrage von Burtch Works 2017). Dabei ist die wachsende Beliebtheit der Software wenig verwunderlich, schließlich weist R ein in der Praxis nahezu unbegrenztes Anwendungspotential auf. Die paketbasierte Struktur der Software ermöglicht eine unkomplizierte Erweiterung des originären Funktionsumfangs. So kann das Basispaket jederzeit um diverse Spezialpakete erweitert werden, die maßgeschneiderte Analysewerkzeuge für ganz konkrete Problemstellungen liefern können. Ein Mediziner kann beispielsweise Pakete installieren, die eine für klinische Studien erforderliche Akkreditierung erhalten haben. Ein Sozialwissenschaftler kann Pakete nutzen, die speziell für die Abbildung von sozialen Netzwerken kreiert wurden und ein Geograph kann auf Pakete zugreifen, die eigens zur morphologischen Analyse von Oberflächenstrukturen konzipiert wurden. Kurzum: Die Anwendungsfelder dieser Software sind vielfältig und der Funktionsumfang des Programms aufgrund seiner Paketorientierung sehr mächtig (vgl. Abbildung 1).

Die Eingabe von Befehls- und Codezeilen erfolgt in R interaktiv. Ein wiederholtes Ausprobieren im Trial-and-Error-Stil gehört bei der Arbeit mit dem Programm daher zum Regelfall und macht ein häufiges Speichern funktionierender Code-Fragmente zur Sicherung des Arbeitsstandes unerlässlich. Auf den ersten Blick kann die codelastige Oberfläche des Programms unerfahrene Benutzer leicht abschrecken. Ist diese Anfangshürde jedoch genommen, gestaltet sich die Benutzung von R schnell intuitiv. Je häufiger Sie das Programm verwenden, desto leichter wird Ihnen seine Bedienung fallen. Ein regelmäßiges Üben und Praktizieren kann daher gerade für Anfänger ratsam sein.

Da R auf der GNU-Lizenz basiert, steht die Verwendung und Erweiterung der Software jedem User weitgehend frei. Die Offenheit der Software hat im Laufe der Zeit zur Entstehung einer riesigen Nutzer- und Fangemeinde beigetragen. Generelle Fragen zum Code, aber auch Fragen zu konkreten statistischen Problemen werden daher rege im Internet diskutiert (hier nur eine kleine Auswahl besonders beliebter Treffpunkte im Netz: R-bloggers, RStudio-Community, stackoverflow). Die Hilfsbereitschaft dieser Community kann gerade für Anfänger von besonderem Vorteil sein. Aus persönlicher Erfahrung kann ich sagen, dass es nahezu keine statistische Problemstellung gibt, die im Netz noch nicht diskutiert und/oder gelöst worden wäre. Aus diesem Grund entfällt bei der Arbeit mit R oftmals das lästige Wälzen dicker Benutzerhandbücher. Stattdessen hilft bei der Lösung neuer Probleme häufig eine kurze Recherche im Internet. Wer R aktiv nutzt, investiert also nicht selten viel Zeit in Online-Recherchen, probiert Code-Zeilen anderer User aus und setzt die Lösung seines spezifischen Problems aus vielen Teillösungen anderer User im Netz zusammen. Die Arbeit mit R zeichnet sich also durch eine große interaktive Komponente aus.

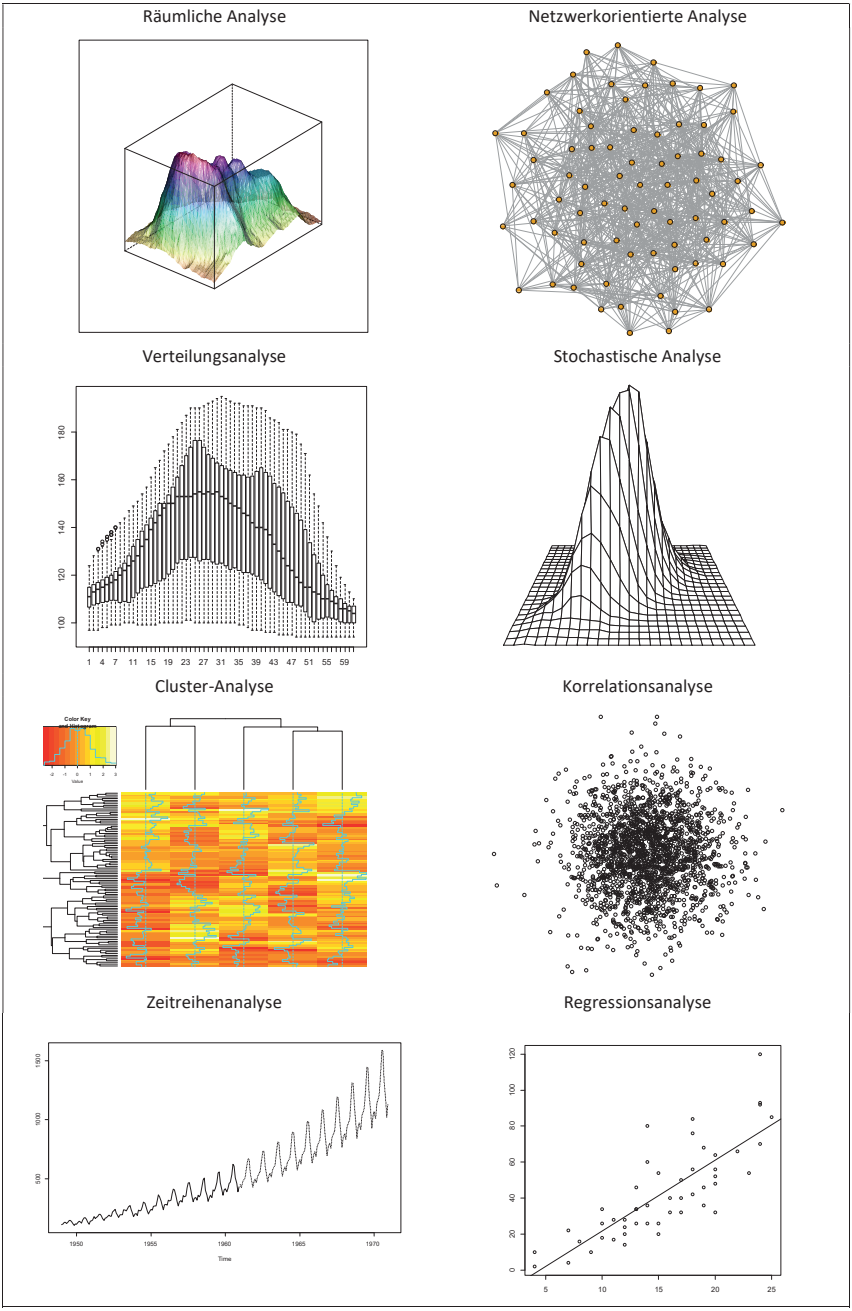


Abbildung 1: Zur Anwendungsvielfalt von R

Installation und erste Schritte

Damit Sie die Software an ihrem Arbeitsplatz nutzen können, müssen Sie zunächst das frei verfügbare Basis-Paket, das alle grundlegenden Funktionen bereitstellt, herunterladen und auf ihrem Rechner installieren. Die Download-Seite kann über den folgenden Link erreicht werden: `https://cran.r-project.org`. Nach dem Download lässt sich die Software unkompliziert einrichten. Folgen Sie hierzu einfach der Anleitung des Installationsclients. Die Nutzung von `R` ist nicht an ein spezielles Betriebssystem gekoppelt. Sie können das Programm daher sowohl auf einem Windows-System (32/64 bit) als auch auf einem Linux- oder Mac-OS-X-Rechner verwenden. Achten Sie beim Download darauf, die für ihr System passende Version auszuwählen. Nach der erfolgreichen Installation lässt sich die Software über einen Doppelklick auf die `Rgui.exe` starten. Hiernach erscheint die so genannte Kommando-Zeile von `R`, die auf eine Eingabe erster Befehle und Code-Zeilen wartet (vgl. Abbildung 2). Diese können in R standardmäßig nach dem `>` Zeichen gesetzt werden. Die Eingabe wird durch das Drücken der `Enter`-Taste beendet. Die Ausführung des Befehls erfolgt dann im Weiteren, wenn in ihrem Code-Fragment keine logischen Fehler festgestellt werden, automatisch. Sie können die Ausführung eines Befehls aber auch manuell unterbrechen (kann vor allem bei rechenintensiven Vorgängen notwendig sein), indem Sie wahlweise entweder die Tastenkombination `CTRL` und `C` oder `CTRL` und `BREAK` drücken.

Wie alle unixähnlichen Systeme unterscheidet auch `R` zwischen Groß- und Kleinschreibung. Ein großes `X` und ein kleines `x` stellen also für `R` grundsätzlich verschiedene Ausdrücke dar. Zur Vergabe von Bezeichnungen können in `R` alle alphabetischen und numerischen Zeichen herangezogen werden. Die Auswahl der zugelassenen Zeichen kann sich allerdings je nach Betriebssystem und Ländereinstellung deutlich unterscheiden. Um bestimmte, beispielsweise deutsche Spracheinstellungen zu erzwingen, kann der `Sys.setlocale`-Befehl verwendet werden:

```
> Sys.setlocale("LC_TIME", "German")       # Für Windows
> Sys.setlocale("LC_TIME", "de_DE.utf8")   # Für Linux
```

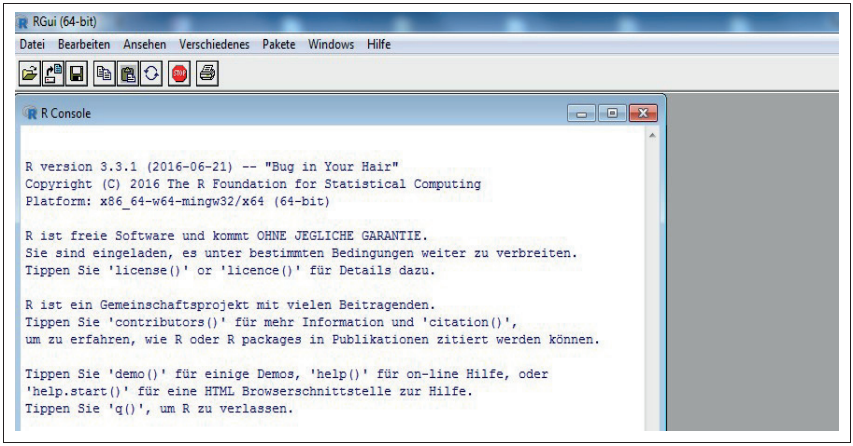


Abbildung 2: Die R-Konsole

Der Funktionsumfang des Basis-Pakets kann durch das Laden spezialisierter Pakete jederzeit erweitert werden. Geladen werden Pakete entweder manuell über die am oberen Fensterrand befindliche Menüleiste (vgl. Abbildung 2) oder über den `install.packages`-Befehl. An dieser Stelle wollen wir zu Übungszwecken das Paket `lmtest` laden, das zahlreiche Zusatzfunktionen bereitstellt, die im Rahmen von statistischen Testverfahren häufig benötigt werden. Hierzu geben wir zunächst `install.packages("lmtest")` in die `R`-Konsole ein, wählen dann einen CRAN-Spiegel aus und warten, bis R die erforderlichen Daten aus dem Internet bezogen und auf unserem System eingerichtet hat. Nachdem das Paket erfolgreich entpackt wurde, muss dieses allerdings noch in die Bibliothek eingelesen werden. Das Einlesen erfolgt im Programm über den `library`-Befehl. Wir tippen daher in die Kommando-Zeile `library("lmtest")` ein. War das Einlesen erfolgreich, stehen uns fortan die Zusatzfunktionen des `lmtest`-Pakets in vollem Umfang zur Verfügung.

Der Umfang von Paketen kann in `R` grundsätzlich sehr stark variieren. Manche Pakete beinhalten nur rudimentäre Ergänzungen, andere hingegen können den Funktionsumfang für ganze Modellklassen bereitstellen. Um den Überblick über die verschiedenen Funktionen eines neu geladenen Pakets zu behalten, kann es ratsam sein, die von den Entwicklern des Pakets zur Verfügung gestellte Dokumentation zu sichten. Standardmäßig erscheint zu jedem neuen Zusatzpaket auf `https://cran.r-project.org` eine ausführliche Beschreibung seines Funktionsumfangs. Die Beschreibung zu dem von uns geladenem Paket `lmtest` listet beispielsweise 27 zusätzliche Funktionserweiterungen auf.

Neben der Einrichtung notwendiger Pakete kann auch das Festlegen eines Arbeitsverzeichnisses zu Beginn einer neuen Sitzung sinnvoll sein. Dies geschieht im Programm über die Eingabe des gewünschten Pfades mit Hilfe des `setwd`–Befehls. Notwendig wird dieser Schritt insbesondere dann, wenn Datensätze aus gespeicherten Dateien eingelesen oder gesicherte Sitzungen wieder fortgeführt werden sollen. Sind Sie sich unsicher, welche Einstellungen `R` aktuell verwendet, können Sie sich den derzeit gültigen Pfad auch mit `getwd()` anzeigen lassen:

```
> setwd("C:/Users/Mein-R-Ordner")
```

Grundlegende mathematische Funktionen

Ist der Arbeitspfad den eigenen Wünschen entsprechend eingestellt, kann die eigentliche Arbeit mit `R` beginnen. `R` beherrscht in der Basisvariante alle gängigen mathematischen Rechenoperatoren sowie Funktionen. Eine kleine Auswahl der wichtigsten Befehle ist in Tabelle 1 zusammengefasst. Die Eingabe dieser einfachen Befehle ist intuitiv, so dass die Bedienung von `R` in gewisser Weise einem überdimensionierten Taschenrechner ähnelt:

```
> 2*2+3
[1] 7

> 1+2+3-4
[1] 2

> (1/2)*(1/2)
[1] 0.25
```

Hinweis: Die Schreibweise in R orientiert sich am US-amerikanischen Standard, Kommazahlen müssen somit mit einem Punkt anstelle des Kommas geschrieben werden. Wir müssen also beispielsweise `0.9` statt `0,9` schreiben.

Nachkommastellen werden in R bei der Ausgabe begrenzt. Wenn wir uns beispielsweise die Kreiszahl π anzeigen lassen, weist R uns das Ergebnis auf sechs Nachkommastellen begrenzt aus. Wollen wir eine größere Zahlenfolge sehen, so müssen wir die Standardeinstellungen von R ändern. Diese Manipulation können wir mit dem Befehl `options(digits)` vornehmen. Setzen wir den `digits`-Wert beispielsweise auf `10`, dann bekommen wir künftig alle Ergebnisse auf `9` Nachkommastellen genau angezeigt:

```
> options(digits=10)

> pi
[1] 3.141592654
```

Alle Rechenergebnisse lassen sich in R bestimmten Objekten oder Variablen zuweisen. Die Zuweisung erfolgt dabei stets mit Hilfe einer, dem Pfeil-Symbol `<-` nachempfundenen, Tastenkombination. In neueren Versionen von R kann die pfeilähnliche Zuweisung auch durch das Setzen eines Gleichheitszeichens an entsprechender Stelle ersetzt werden. Da aber in vielen Statistiklehrbüchern nach wie vor die alte Pfeil-Schreibweise dominiert, werden wir sie auch im Rahmen dieses Buches beibehalten. Um die Funktionsweise der Objektzuweisung zu verdeutlichen, wollen wir im Folgenden die Rechnerergebnisse aus vorheriger Eingabe den Variablen `x` und `y` zuweisen:

```
> x<-2*2+3

> y<-1+2+3-4
```

Nach der Zuweisung können die neu gebildeten Objekte selbst wieder in mathematischen Funktionen eingebaut und, wie das untenstehende Beispiel zeigt, auch mit anderen Objekten verrechnet werden:

```
> (x*y)/y+(log(x)-sqrt(y))
[1] 7.531697
```

Arithmetische Operatoren		Mathematische Funktionen	
+	Addition	`min()`	Minimaler Wert
-	Subtraktion	`max()`	Maximaler Wert
/	Division	`abs()`	Absoluter Wert
*	Multiplikation	`sqrt()`	Wurzel-Funktion
^	Exponentiation	`exp()`	Exponential-Funktion
`%*%`	Matrix-Multiplikation	`log()`	Natürlicher Logarithmus

Tabelle 1: Ausgewählte mathematische Operatoren und Funktionen

Vektoren und Matrizen

Für viele statistische Anwendungen müssen Vektoren und Matrizen definiert werden. In R lassen sich Vektoren am einfachsten durch eine Aneinanderreihung von bestimmten Zahlen oder Variablen definieren. Hierzu wird der so genannte concatenate-Befehl verwendet, der im Programm über ein kleines c abgekürzt wird:

```
> a<-c(1,2,3)
> a
[1] 1 2 3
> b<-c(2,1,5)
> b
[1] 2 1 5
```

Hinweis: Je nach Zielsetzung kann auch die Verwendung anderer Befehle zur Erzeugung von Vektor-Objekten sinnvoll sein. Eine kleine Übersicht der wichtigsten Befehle ist in Tabelle 2 zusammengefasst.

Die standardmäßig im Programm implementierten arithmetischen und mathematischen Funktionen können ohne Einschränkungen auch auf alle Skalar- und Vektorobjekte angewendet werden. Für die Vektorrechnung werden in R also keine weiteren Befehle oder spezialisierten Ausdrücke mehr benötigt. Die korrekte Anwendung arithmetischer Funktionen setzt allerdings eine gewisse Kenntnis der linearen Vektoralgebra voraus. Wenn wir also beispielsweise Vektor a quadrieren a^2, wird in R automatisch jedes einzelne Element des Datenvektors quadriert. Besondere Beachtung verdient in diesem Kontext die Recycling-Eigenschaft von R. Dank dieser Eigenschaft können in R viele Befehle kompakter und übersichtlicher formuliert werden. So genügt zum Beispiel anstelle einer ausführlichen Rechenanweisung wie dieser:

```
> c(1,2,3,4,5,6,7,8,9,10)*c(1,2,1,2,1,2,1,2,1,2)
[1]  1  4  3  8  5  12  7  16  9  20
```

eine kurze Formulierung der folgenden Form, um im Ergebnis dieselbe Rechenoperation auszuführen:

```
> c(1,2,3,4,5,6,7,8,9,10)*c(1,2)
[1]  1  4  3  8  5  12  7  16  9  20
```

Mehrere Vektoren lassen sich in R zu Matrizen zusammenfassen. Wir können beispielsweise eine neue Matrix mit der Bezeichnung m1 erstellen, in der die beiden zuvor definierten Vektoren a und b zusammengefasst werden sollen. Hierzu führen wir den matrix-Befehl aus und definieren über die uns bereits bekannte concatenate-Funktion, aus welchen Vektoren die neue Matrix gebildet werden soll. Anschließend legen wir noch fest, wie viele Zeilen die neue Matrix vorweisen soll:

```
> m1<-matrix(c(a,b),3)
> m1
     [,1] [,2]
[1,]    1    2
[2,]    2    1
[3,]    3    5
```

Befehle	Beschreibung	Beispiele
`rep`	Replikation numerischer Werte	`> rep(5,5)` `[1] 5 5 5 5 5`
`seq`	Bildung einer numerischer Folge	`> seq(5)` `[1] 1 2 3 4 5`
`numeric`	Erzeugung eines numerischen Vektors	`> numeric(5)` `[1] 0 0 0 0 0`
`a:b`	Bildung numerischer Folge von a bis b	`> 1:5` `[1] 1 2 3 4 5`

Tabelle 2: Ausgewählte Befehle zur Erzeugung von Vektoren

Im Ergebnis erhalten wir eine 3×2-Matrix, die sich aus den beiden Spaltenvektoren `a` und `b` mit insgesamt drei Zeilen zusammensetzt. Wie der obige R-Auszug zeigt, fehlen der von uns definierten Matrix allerdings noch die Spaltenüberschriften. Diese können wir nachträglich mit Hilfe des `colnames`-Befehls manuell ergänzen:

```
> colnames(m1)<-c("Spalte a", "Spalte b")
> m1
     Spalte a Spalte b
[1,]        1        2
[2,]        2        1
[3,]        3        5
```

Matrizen lassen sich in R addieren, subtrahieren, multiplizieren, transponieren und invertieren. Hierbei müssen allerdings einige Besonderheiten der Matrix-Algebra beachtet werden, die wir im Folgenden kurz ansprechen wollen.

Die Addition und Subtraktion von Matrizen setzt voraus, dass beide Matrizen in ihrer Spalten- und Zeilenzahl übereinstimmen. Es lassen sich also nur diejenigen Matrizen addieren oder subtrahieren, die in ihren Dimensionen identisch sind. Die Dimensionen einer Matrix lassen sich in R über den `dim`-Befehl kontrollieren. Für unsere Matrix `m1` weist der `dim`-Befehl die Dimensionen `3` und `2` aus, was der Anzahl an Zeilen (`3`) und Spalten (`2`) in der betrachteten Matrix entspricht. Wir wollen nun zu Übungszwecken unsere Matrix `m1` mit sich selbst addieren und subtrahieren. Die Eingabe der entsprechenden Befehle erfolgt weitgehend intuitiv:

```
> m1+m1
     [,1] [,2]
[1,]    2    4
[2,]    4    2
[3,]    6   10
> m1-m1

     [,1] [,2]
[1,]    0    0
[2,]    0    0
[3,]    0    0
```

Anders als bei der Addition und Subtraktion setzt die Multiplikation von Matrizen voraus, dass die Anzahl der Spalten der ersten Matrix mit der Zeilenanzahl der zweiten Matrix übereinstimmen muss. Zu Übungszwecken wollen wir an diese Stelle die uns bekannte 3×2-Matrix m1 mit einer neuen 2×2-Matrix m2 multiplizieren:

```
> m2<-matrix(c(1,2,3,4),2)

> m1%*%m2
     [,1] [,2]
[1,]    5   11
[2,]    4   10
[3,]   13   29
```

Die Transposition von Matrizen erfolgt in R über den transpose-Befehl, der bei der Eingabe mit einem kurzen t abgekürzt wird. Wie das nachfolgende Beispiel zeigt, bewirkt die Transposition einer Matrix dabei eine Umwandlung ihrer Spaltenvektoren in Zeilenvektoren:

```
> m1
     [,1] [,2]
[1,]    1    2
[2,]    2    1
[3,]    3    5

> t(m1)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    1    5
```

Hinweis: Durch die Transposition ändern sich die Dimensionen der Matrizen. In unserem Beispiel wurde dabei eine 3×2-Matrix in eine 2×3-Matrix umgewandelt.

Die Inverse einer Matrix wird in R über den solve-Befehl berechnet. Das Invertieren einer Matrix ist dabei grundsätzlich nur dann möglich, wenn die betrachtete Matrix quadratisch ist. Von einer quadratischen Matrix wird im Allgemeinen immer dann gesprochen, wenn sie genauso viele Spalten wie auch Zeilen aufweist. Die von uns erzeugte Matrix m2 erfüllt diese Voraussetzung und lässt sich daher invertieren:

```
> solve(m2)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

Wie bei allen anderen Rechenvorgängen auch, lassen sich die durch Transposition und Inversion erzeugten Matrizen bestimmten Objekten im Workspace zuweisen. Durch die Zuweisung findet dabei nicht nur ein Transfer aller numerischen Werte in das neue Objekt statt, sondern auch die Übertragung der Matrixeigenschaft. Das Programm erkennt das neue Objekt also nach der Zuweisung eigenständig als ein Matrixobjekt, ohne dass wir dies noch spezifisch deklarieren oder kennzeichnen müssen. Falls eine solche Kennzeichnung aber explizit erforderlich wird, können wir zum Beispiel die Funktion as.matrix verwenden, die jeden Dateninput in eine Matrix konvertiert. Analog hierzu existiert in R auch der Befehl as.vector, der beliebige Inputs in Vektoren umwandeln kann.

Logische Operatoren, Schleifen und Funktionen

In R können eigene Programme und Funktionen erstellt werden. Wie in anderen Programmiersprachen auch, werden hierzu Operatoren verwendet, die sowohl logische Vergleiche als auch logische Verknüpfungen zwischen bestimmten Ausdrücken zulassen (vgl. Tabelle 3). Zudem können in R die üblichen `for`-, `while`- und `repeat`-Schleifen erstellt werden.

Als `For`-Schleifen werden im einschlägigen Jargon Kontrollstrukturen bezeichnet, mit deren Hilfe bestimmte Anweisungen mit einer vordefinierten Anzahl von Wiederholungen ausgeführt werden können. Eine einfache `for`-Schleife kann zum Beispiel wie folgt definiert werden:

```
> n<-3
> for(i in 0:n) {print(i);}
[1] 0
[1] 1
[1] 2
[1] 3
```

Über den `if`-Befehl kann die oben abgebildete `for`-Schleife um eine Variablenbedingung ergänzt werden. In unserer Schleife wollen wir zum Beispiel die Ausgabe auf den Indexwert 2 beschränken, wir definieren daher die Bedingung `if(i==2)` und setzen diese in die ursprüngliche `for`-Schleife ein:

```
> for(i in 0:n) {if(i==2) print(i);}
[1] 2
```

Die Ausführung der Schleife kann auch an spezifische Vorbedingungen (Initialbedingungen) geknüpft werden. Die Anweisungen innerhalb der Schleife werden also erst ausgeführt, wenn die initialen Bedingungen geprüft und als wahr identifiziert werden. Solche an eine Anfangsbedingung geknüpften Schleifen werden, wie auch in anderen Programmiersprachen üblich, mit Hilfe des `while`-Arguments formuliert. Das nachfolgende Beispiel soll das Funktionsprinzip dieser Schleifen verdeutlichen:

```
> i<-1
> while(i<3) {print(i); i=i+1}
[1] 1
[1] 2
```

Zum Verständnis: Die oben gezeigte `while`-Schleife führt die Berechnung `i=i+1` iterativ aus, solange die von uns formulierte Bedingung `i<3` erfüllt ist.

Als letztes wollen wir uns kurz mit der Funktionsweise von so genannten `repeat`-Schleifen vertraut machen. Wie es die Namensgebung schon vermuten lässt, werden `repeat`-Schleifen zur Wiederholung von bestimmten Anweisungen benötigt. Da die Wiederholung dabei unendlich oft ausgeführt wird, benötigt die `repeat`-Schleife zwingend eine Abbruchsbedingung. Diese auch als „Break Condition“ bezeichnete Bedingung legt exakt fest, unter welchen Umständen die `repeat`-Schleife abgebrochen wird. Zum setzen solcher Abbruchsbedingungen kann der `if`-Befehl verwendet werden. Alles in allem ähnelt die logi-

Logische Vergleiche		Logische Verknüpfungen	
Operator	Beschreibung	Operator	Beschreibung
==	gleich	x | y	x ODER y
!=	ungleich	x & y	x UND y
<	kleiner	!x	NICHT x
>	größer		
<=	keiner-gleich		
>=	größer-gleich		

Tabelle 3: Logische Operatoren in R

sche Struktur einer `repeat`-Schleife somit einer auf den Kopf gestellten `while`-Schleife. Ein einfaches Beispiel soll die Funktionsweise der `repeat`-Schleife veranschaulichen:

```
> i<-1
> repeat {print(i); i=i+1; if(i==3){break}}
[1] 1
[1] 2
```

Als Break Condition haben wir in unserem Beispiel die `if`-Bedingung `i==3` definiert. Die Berechnungsvorschrift `i=i+1` wird also solange iterativ wiederholt, bis die in der `if`-Klammer festgelegte Abbruchsbedingung erreicht ist. Das ist in unserer `repeat`-Schleife, wie der obige Code-Auszug deutlich macht, nach genau zwei Wiederholungen der Fall.

Neben den bereits gezeigten Schleifen können in `R` auch eigene Funktionen erstellt werden. Hierfür steht in `R` der Befehl `function` zur Verfügung. Der Grundaufbau einer Funktion wird dabei wie folgt definiert: `function(Parameter){Anweisung}`. Zu Verdeutlichung wollen wir zunächst eine einfache Funktion definieren, die sich aus den Parametern `x` und `y` zusammensetzen soll. Dabei wollen wir erreichen, dass aus beiden Parametern eine Summe gebildet und `z` zugewiesen wird (erste Anweisung). Anschließend soll `z` über den `print`-Befehl ausgegeben werden (zweite Anweisung). Die konkrete Definition der Funktion sieht dann wie folgt aus:

```
> myfunc<-function(x,y){z<-x+y; print(z)}
```

Nachdem unsere Funktion definiert und dem Objekt `myfunc` zugewiesen wurde, können wir die korrekte Funktionsweise unserer neu gebildeten Funktion über die Abfrage von willkürlichen Werten überprüfen:

```
> myfunc(1,1)     # mit x=1, y=1 und z=x+y
[1] 2
```

Wie die obige Wertabfrage zeigt, führt unsere Funktion die vordefinierten Anweisungen korrekt aus.