Andreas Krause

Einführung in S und S–PLUS

Mit Aufgaben und vollständigen Lösungen

17. Februar 1997

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Vorwort

S-PLUS ist ein äußerst mächtiges System, um Daten zu analysieren. Die Stärken dieser Umgebung liegen in der interaktiven Datenanalyse, dem flexiblen Erstellen von Grafiken und der hochmodernen Entwicklungsumgebung, beruhend auf der S Language der AT&T Bell Labs (heute Lucent Technologies).

Die grafische Oberfläche erlaubt die Verwendung von Methoden und das Erstellen von Grafiken durch wenige Mausklicks. Eine erstellte Grafik kann (wiederum durch Mausklicks) in allen Elementen editiert werden. Ein Klick auf die Achse und ein weiterer Klick auf eine andere Farbe verändern den Farbton der Achse. Ein Klick auf eine statistische Methode führt die Methode aus.

Programmierer finden ihre Freude an dem offenen Konzept, fast alle Elemente des Systems sind in der S Language selbst geschrieben und im Quellcode zugreifbar. Neue Entwicklungen wie konsequente Umsetzung des Funktionskonzepts und Objektorientiertheit sind auch für Nicht– und Gelegenheits–Programmierer äußerst attraktiv.

Dieses Buch soll nicht die Handbücher ersetzen. Es soll kein besseres Handbuch entstehen, sondern eine kompakte Einführung, die trotzdem nicht bei den einfachen und gut funktionierenden Beispielen aufhört. Das vorliegende Buch ist aus Skripten zu Vorlesungen und Übungen zu "Einführung in S-PLUS" hervorgegangen, und dementsprechend soll eine schnelle, aber solide Einführung in die Konzepte des Systems und der Sprache S-PLUS, angereichert mit vielen Hinweisen aus langjähriger Praxis, das Ziel sein. Jedes Kapitel endet mit einer Auswahl von Problemstellungen, die das vorangegangene Material repetieren und anhand praktischer Fragestellungen vertiefen. Der Leser (wie auch die Leserin) soll angeregt werden, eine eigene Lösung zu erarbeiten, denn nie existiert nur eine einzige, richtige Variante. Im Anschluß daran kann ein Blick auf eine andere Lösung geworfen werden, denn zu jeder Aufgabe folgt eine im Detail ausgearbeitete Lösung.

Dadurch kann zum einen eine fertige Lösung nachgeschlagen werden, um mit der eigenen zu vergleichen, zum anderen sind diese Bausteine auch direkt verwendbar, um weitere Fragestellungen bearbeiten und lösen zu können.

Es sei noch einmal betont, daß hier nicht das im übrigen sehr lesenswerte Handbuch ersetzt werden soll, so daß nicht auf alle Besonderheiten und Optionen eingegangen wird, sondern vielmehr die wichtigsten herausgegriffen und anhand von Anwendungen vertieft werden. Dadurch entsteht ein kompakter Gesamtüberblick, der schnell auf die Elemente und Konzepte von S-PLUS zu sprechen kommt.

Nach einer Einführung in das System und die Konzepte von S-PLUS gehen wir auf die grafischen Möglichkeiten ein und behandeln diese ausführlich. Datenanalyse in S-PLUS und die zugehörigen Methoden, vielfach auch auf Grafiken basierend, werden erarbeitet, indem in S-PLUS bereits vorhandene reale Daten analysiert werden. Methoden und typische Auswertungen werden so gemeinsam dargestellt.

Im Kapitel über Programmieren werden die Möglichkeiten von S-PLUS als Programmiersprache diskutiert, das Funktionskonzept, typische Programmierbefehle wie Schleifen und Datenstrukturen wie Listen besprochen, und Fehlersuche (Debugging) behandelt. Daran anschließend werden ausführlich Ein– und Ausgabe von Daten behandelt und abschließend eine Sammlung von praktischen Hinweisen und Tips wie Einbindung von Grafiken in Textverarbeitung und vieles mehr diskutiert.

Das vorliegende Buch richtet sich an Anfänger, die einen schnellen Start in S-PLUS möchten, sowie an Benutzer, die bereits Erfahrung mit dem System S-PLUS gesammelt haben. Sie werden wertvolle Hinweise aus der Praxis finden. Es wird lediglich ein wenig Erfahrung im Umgang mit der eigenen Hardware und dem Betriebssystem vorausgesetzt.

Es werden nur elementarste Statistik–Kenntnisse vorausgesetzt. In gewissen Grenzen lassen sich für Nicht–Statistiker durchaus Einblicke gewinnen, wie Daten analysiert werden können durch Verwendung explorativer, grafischer und konfirmatorischer Methoden.

Die Struktur dieser Einführung ist durch seine Herkunft als Vorlesungsmanuskript geprägt. Dadurch kann es als solches direkt verwendet werden. Ein Kapitel kann jeweils als Stoff einer 90–minütigen Vorlesung dienen, und die Übungen vertiefen den Stoff. Aufgrund der

Flexibilität des Systems S-PLUS wird es immer viele verschiedene Wege geben, um das gesteckte Ziel zu erreichen, so daß die Bearbeitung der Aufgaben ohne vorheriges Ansehen der Lösung eine interessante Herausforderung sein sollte, die oft überraschende Vergleiche bringt.

Um die neuen Kenntnisse gleich umzusetzen, ist es sinnvoll, an einem Rechner zu sitzen oder nach einem Abschnitt eine Pause zu machen, um auszuprobieren und zu vertiefen. Wird das Buch als Nachschlagewerk benutzt, um einen Einblick in das System S-PLUS zu bekommen, genügen vermutlich ein paar Tage investierter Zeit. Die Art der Präsentation ist bewußt locker gehalten, so daß das Buch nicht "studiert" werden muß, sondern genausogut in der Freizeit gelesen werden kann.

Wenn diese Grundlagen einigermaßen gefestigt sind, sollten die Elemente von S-PLUS verstanden sein. Um weitere Probleme zu bearbeiten, die hier nicht abgedeckt werden, sollte ein Blick in die übrigens sehr guten Handbücher genügen.

Bereits jetzt möchte ich mich bei denjenigen Lesern entschuldigen, die alle EDV–Fachbegriffe in deutscher Sprache erwarten. S-PLUS ist ein amerikanisches System und die gesamte *Computer*(sic!)welt beruht auf englischer Terminologie. Auf mich wirken Begriffe wie Umschalt– und Steuerungstaste, Abrollmenu und Schaltfläche eher befremdlich und ich konnte vielfach feststellen, daß die Verwirrung geringer ist, wenn von Shift– und Control–Taste, Pulldown–Menus und Buttons die Rede ist.

Dieses Manuskript ist im Verlaufe vieler Jahre an den Universitäten von Dortmund und Basel entstanden. Viele Personen haben direkt oder indirekt, durch Ideen, Diskussionen und Kommentare, zu seinem Entstehen beigetragen, darunter eine große Zahl von Studenten des Fachbereichs Statistik der Universität Dortmund und des Wirtschaftswissenschaftlichen Zentrums der Universität Basel. Ihnen allen sei anonym gedankt.

Im weiteren Verlauf haben einige Kollegen Vorab–Versionen gelesen und zahlreiche Kommentare, Anregungen und Verbesserungsvorschläge gegeben. Zunächst danke ich denen, die ich unbeabsichtigt vergessen habe, die mir dafür aber hoffentlich nicht böse sind. Namentlich möchte ich Axel Benner (Heidelberg), Tom Cook (Wisconsin), Peter

Hartmann (St. Gallen) Thomas T. Kötter (Berlin) und Tony Rossini (South Carolina) danken.

Insbesondere danke ich Melvin "Skip" Olson (Genf). Die englische Ausgabe dieses Buches ist durch intensive Zusammenarbeit entstanden und bei Springer New York erschienen.

Die Entwickler und Distributoren von S-PLUS, die Data Analysis Products Division of MathSoft (früher Statistical Sciences, Inc.) in Seattle und GraS (Graphische Systeme) in Berlin in Person von Doug Martin, Brian Clark und Charlie Roosen sowie Reinhard Sy haben mich großzügig unterstützt und insbesondere eine Alpha–Version des letzten Release zur Verfügung gestellt.

Die Entwickler der Satzsysteme, mit denen dieses Buch entstand, haben Huldigung verdient. Donald Knuth und Leslie Lamport für LaTeX und Eberhard Mattes für emTeX, Thomas Rockiki für EPSF und den Entwicklern des CLMONO-Pakets beim Springer Verlag Heidelberg zolle ich Respekt.

Während der gesamten Entstehungsphase war stets eine gute Zusammenarbeit und äußerst professionelle Unterstützung durch den Springer–Verlag gegeben, daher ein großes Dankeschön an das Team in Heidelberg, Annette Klute, Claudia Müller, Ute McCrory, Karen Proff und Arnd Grimmer sowie an John Kimmel und Martin Gilchrist in New York.

Schlußendlich wäre dieses Projekt sicher nicht fertiggestellt worden ohne die fachliche und moralische Unterstützung meiner Frau, Michaela Jahn. Ihre permanente Hilfsbereitschaft und Ermutigung sowie jederzeitige Bereitschaft zu Diskussionen hat mich vorwärts getrieben.

Andreas Krause   Basel, im Februar 1997

Inhaltsverzeichnis

4. Die Zweite Sitzung

Im vorangegangenen Kapitel haben wir die elementaren Techniken gelernt, um mit S-PLUS als System umzugehen. Darauf aufbauend wollen wir uns in diesem Kapitel mit den etwas komplexeren Strukturen befassen, die in S-PLUS aufgebaut werden können. Wir gehen insbesondere auf Datenstrukturen wie Matrizen und mehrdimensionale Felder ein, behandeln Datensätze, die in sogenannten Data Frames abgespeichert sind und beschäftigen uns mit Listenstrukturen. Schließlich werden wir mit fehlenden Werten arbeiten und abschließend unsere Kenntnisse auf den Geysir–Datensatz anwenden. Die Übungsaufgaben vertiefen die Kenntnisse anhand praktischer Fragestellungen.

4.1 Datenstrukturen

Wie wir in der ersten Sitzung mit S-PLUS gelernt haben, sind Vektoren eine Datenstruktur, die die Verkettung mehrerer Objekte der gleichen Art zu einem Objekt (einer Variablen) erlauben. Zum Erstellen dieser Verkettung haben wir die Funktion `c` kennengelernt. Wenn mehrere solcher Vektoren zu einer neuen Struktur verknüpft werden, so ergeben sich zweidimensionale Strukturen, die wir im folgenden betrachten wollen. Mathematisch betrachtet sind zweidimensionale Strukturen Matrizen, S-PLUS kennt weitere Arten zweidimensionaler Datenstrukturen.
Die folgenden Abschnitte befassen sich mit Matrizen, Feldern (Arrays), Datenfeldern (Data Frames) und Listen. Zur besseren Übersicht stellen wir die Strukturen zu Beginn kurz vor.

Matrizen sind im Prinzip eine Reihe von vertikal dargestellten Vektoren, hintereinander angeordnet. Daraus ergibt sich eine rechteckige Form. Die Horizontalen werden als Zeilen bezeichnet, die Vertikalen als Spalten. Jede Spalte entspricht einem Vektor, jede Zeile einer Beobachtung.

Ordnet man die Daten Alter, Körpergewicht und Körpergröße von 10 Personen zu einer Matrix an, ergibt sich eine 10x3–Matrix mit 10 Zeilen und 3 Spalten. Jede Zeile enthält alle Daten zu einer Person, jede Spalte die Daten einer Variable zu allen Personen.
Matrizen können nur Variablen eines Typs, also entweder Zahlen oder Zeichenketten oder einen anderen Typ beinhalten.

Arrays sind eine etwas globalere Form von Matrizen, so daß jede Matrix auch ein Array ist. Ein Array kann in S-PLUS bis zu acht Dimensionen haben. Arrays können wie Matrizen nur einen Datentyp umfassen. Die wichtigsten Strukturen von Arrays haben spezielle Namen. Ein nulldimensionales Array wird auch Skalar genannt (entsprechend einem einzelnen Datum), ein eindimensionales Array heißt Vektor, ein zweidimensionales Array wird Matrix genannt und ein dreidimensionales Array wird auch als Würfel bezeichnet.
Wollten wir die im vorigen Abschnitt über Matrizen erwähnten Personendaten für mehrere Kontinente erheben, könnte man auf jedem der fünf Kontinente zehn Personen untersuchen und mit den Daten ein Array der Dimension 10x3x5 anlegen. Jeder Schnitt durch die zweite Dimension fixiert eine Variable, so daß, wenn die Variable Körpergröße betrachtet wird, wir eine Matrix der Dimension 10x5 erhalten. Schneiden wir durch die dritte Dimension, zum Beispiel, um alle Daten des Kontinents Afrika zu erhalten, bekommen wir eine 10x3–Matrix.

Data Frames sind aufgebaut wie Matrizen. Sie erlauben die Kombination verschiedener Datentypen zu einer Matrix, so daß wir in einem Data Frame bei unseren Personendaten noch zusätzlich die Variable Herkunftsland abspeichern können (bei einer Matrix oder einem Array wäre dies nicht möglich).

Listen sind die universellsten Datenstrukturen. Sie können in jedem Element eine beliebige Datenstruktur enthalten und erlauben verschiedene Größen für jedes Element der Liste. Die Flexibilität fordert allerdings manchmal den Preis eines etwas aufwendigeren Zugriffs.

Als erstes befassen wir uns mit Matrizen.

4.1.1 Matrizen

Eine Matrix ist eine rechteckige Struktur, die in jeder Zelle ein Datum enthält. Eine Zelle hat eine Adresse, die durch die Nummer der Zeile und der Spalte festgelegt ist, in der das Datum sich befindet. Ein solches Arrangement liegt beispielsweise auch Spreadsheets wie Excel zugrunde. Das Konzept abstrakt zu beschreiben ist weit komplizierter

als es anhand von Anwendungen zu verdeutlichen. Daher sehen wir uns einige Möglichkeiten an, Matrizen zu verwenden.
Zunächst muß eine Matrix erstellt werden. Dazu hält S-PLUS die Funktion `matrix` bereit. Die Syntax der Funktion `matrix` ist im wesentlichen die folgende:

`matrix (`*data*, *nrow*, *ncol*, `byrow=F)`

oder auf Deutsch

`matrix (`*Daten*, *Zeilenzahl*, *Spaltenzahl*, *zeilenweise*`=F)`

Zur Definition einer Matrix brauchen wir demnach vier Argumente. Unbedingt notwendig sind die Daten und zudem kann man die Anzahl der Zeilen und Spalten angeben sowie ob die Matrix zeilenweise oder spaltenweise aufgefüllt werden soll. Wenn die Voreinstellung des Parameters `byrow=F` nicht umgesetzt wird auf `byrow=T`, wird die Matrix spaltenweise gefüllt, also zuerst die erste Spalte, dann die zweite, usw. Wird `byrow=T` gesetzt, wird zeilenweise (horizontal) mit den Werten in *Daten* aufgefüllt.
Die Dimensionsangaben sind ebenfalls optional. Fehlen die Angaben *Zeilenzahl* und *Spaltenzahl*, wird eine eindimensionale Matrix erstellt, was dem Setzen von `ncol=1` entspricht.
Eine eindimensionale Matrix ist aber wiederum ein Vektor. Wir erstellen eine erste Matrix mit den Zahlen von 1 bis 6.

```
> matrix (1:6)
      1 2 3 4 5 6
```

Wenn nur eine der beiden Dimensionsangaben vorliegt, bestimmt S-PLUS die fehlende andere Dimension durch einfache Division. Liegen 6 Daten vor und die Anzahl der Zeilen ist gleich 2, so hat die Matrix 6/2=3 Spalten.

```
> matrix (1:6, nrow=2)
      1 3 5
      2 4 6
```

Natürlich können auch beide Dimensionen angegeben werden. Ist die Dimension so, daß die Datenzahl nicht zur Matrixstruktur paßt, verwendet S-PLUS die Daten nur teilweise oder mehrfach, informiert uns aber mit einer Warnung, daß dies geschehen ist.
Wir sehen, daß die Matrix zuerst in der ersten Spalte, dann in der zweiten und anschließend in der dritten Spalte gefüllt worden ist. Wollten

wir sie horizontal füllen, würden wir die Option `byrow=T` setzen (also "by rows", zeilenweise, füllen). Das `T` steht für den logischen Wert `TRUE`.

```
> matrix (1:6, nrow=2, byrow=T)
      1 2 3
      4 5 6
```

Wenn Daten als Datei vorliegen und diese in eine Matrixstruktur eingelesen werden sollen, geschieht dies typischerweise zeilenweise.
Was wird von den folgenden Befehlen erstellt ? (Erst überlegen und dann ausprobieren !)

```
> x <- 3:8
> matrix (x, 3, 2)
> matrix (x, ncol=2)
> matrix (x, ncol=3)
> matrix (x, ncol=3, byrow=T)
```

Hinweis Matrizen können nur aus einem Datentyp bestehen. In unserem Fall besteht die Matrix aus Zahlen, sie kann aber auch aus Zeichenketten bestehen, jedoch nicht aus beiden Typen gleichzeitig. Für diesen Fall sind Data Frames die bessere Datenstruktur, auf die wir in Kapitel 4.1.3 eingehen. ◁

Wir erstellen im weiteren Verlauf eine Anwendung, die wir nach und nach verfeinern. Wir wollen Daten in einer Matrix speichern, die für verschiedene Länder deren Bruttosozialprodukt (BSP), Bevölkerung (in Millionen) und die Inflationsrate (in Prozent für 1995) angibt.[1]

```
> daten <- c(2075, 81.1, 1.8, 1355, 57.7, 1.7,
+ 197,7.9,2.2)
> landesdaten <- matrix (data, nrow=3, byrow=T)
> landesdaten
       2075  81.1  1.8
       1355  57.7  1.7
        197   7.9  2.2
```

Die Daten sind als Variable in `landesdaten` abgelegt, aber in dieser Form sind sie nicht sehr brauchbar. Wenn wir keine weitere Beschreibung haben, wissen wir nicht, welche Zahl welches Land und

[1] Die Daten sind dem Buch "The Economist Pocket World in Figures, 1997 Edition" entnommen.

welche Variable angibt. Wir können die Matrix "beschriften", indem wir Label (Beschreibungen) zu den Zeilen und Spalten der Matrix hinzufügen.
Die Label müssen mit der Anzahl Zeilen und Spalten übereinstimmen, so daß wir einen Vektor der Länge 3 für die Zeilen und einen weiteren Vektor der Länge 3 für die Spalten erstellen.

Beispiel 4.1. Label einer Matrix erstellen
Wir weisen der Matrix `landesdaten` Label (Beschriftungen) für Zeilen und Spalten zu. Dazu verwenden wir die Funktion `dimnames`.

```
> dimnames (landesdaten)
    NULL                        # Bestehende Namen
> dim (landesdaten)
    3 3                         # Dimension der Matrix
> laender <- c("Deutschland", "Frankreich",
+  "Oesterreich")
> variablen <- c("BSP", "Bev", "Inflation")
```

Wir können zunächst nur den Zeilen Label zuweisen, die aus den Landesnamen bestehen. Um auszudrücken, daß die Spalten kein Label bekommen sollen, verwenden wir die leere Variable `NULL`.

```
> dimnames (landesdaten) <- list (laender, NULL)
> landesdaten
      Deutschland  2075  81.1  1.8
       Frankreich  1355  57.7  1.7
      Oesterreich   197   7.9  2.2
```

Damit wäre uns schon geholfen, die Länder zu identifizieren. Analog können nur den Spalten Label zugewiesen werden oder Zeilen und Spalten beschriftet werden.

```
> dimnames (landesdaten) <- list (NULL, variablen)
> landesdaten
       BSP   Bev  Inflation
      2075  81.1        1.8
      1355  57.7        1.7
       197   7.9        2.2
> dimnames (landesdaten) <- list (laender, variablen)
> landesdaten
                    BSP   Bev  Inflation
      Deutschland  2075  81.1        1.8
       Frankreich  1355  57.7        1.7
      Oesterreich   197   7.9        2.2
```

Mit der gleichen Funktion, die zur Zuweisung von Labels verwendet wurde, können wir die Label abfragen. Wir lassen dazu die Zuweisung auf der rechten Seite weg und erhalten die Zeilen– und Spaltenlabels zurück.

```
> dimnames (landesdaten)
      [[1]]:
      "Deutschland"  "Frankreich"  "Oesterreich"
      [[2]]:
      "BSP"          "Bev"         "Inflation"
```
◁

Die Namen werden als Liste eingegeben (`list` (*Zeilennamen*, *Spaltennamen*)). Listen behandeln wir noch ausführlich, für den Moment genügt, daß Matrixlabel mit zwei Elementen über die Funktion `dimnames` angegeben werden. Der spezielle Werte `NULL` entfernt Namen oder gibt an, daß keine vorhanden sind.

Zugreifen auf Elemente von Matrizen

Daten werden nicht abgelegt, um sie zu speichern, sondern um mit ihnen zu arbeiten. Demzufolge wollen wir auf die Daten, die wir in der Matrix `landesdaten` gespeichert haben, zugreifen.
Wenn die Indizes der interessierenden Daten in der Matrix bekannt sind, können wir direkt auf sie zugreifen. Wir wissen beispielsweise, daß die Daten von Österreich sich in der dritten Zeile befinden und die Bevölkerung in der zweiten Spalte. Daher können wir die Bevölkerung Österreichs extrahieren.

```
> landesdaten[3, 2]
     7.9
```

Österreichs Einwohnerzahl beträgt also etwa 7.9 Millionen.

Hinweis Auf Daten einer Matrix wird mit Angabe der Zeilennummer und der Spaltennummer (in dieser Reihenfolge) zugegriffen. ◁

Wir können mehr als nur eine Zahl auf einmal extrahieren, indem als Index ein Vektor angegeben wird. So extrahieren wir alle Daten zu Frankreich, indem wir die zweite Zeile und dort die Spalten 1 bis 3 anfordern.

```
> landesdaten[2, 1:3]
       BSP   Bev  Inflation
      1355  57.7        1.7
```

Wenn alle Daten einer Zeile oder einer Spalte gefragt sind, kann der entsprechende Index ganz weggelassen werden. Eine kürzere Eingabe für alle Daten von Frankreich wäre daher

```
> landesdaten[2, ]
       BSP   Bev  Inflation
      1355  57.7        1.7
```

Wenn eine Matrix Labels besitzt, können diese zur Adressierung der Zellen verwendet werden. Indizes sind zwar schneller einzutippen, die Benutzung der Labels macht Programme aber wesentlich besser lesbar und erlaubt auch, daß Zeilen und Spalten eventuell neu sortiert werden, ohne daß das Programm geändert werden muß.

```
> landesdaten["Deutschland", "Inflation"]
     1.8
> landesdaten[, "Inflation"]
      Deutschland  Frankreich  Oesterreich
              1.8         1.7          2.2
```

Dies soll genügen, um das Prinzip der Adressierung von Daten zu erläutern.

Hinweis Mit der vorgestellten Adressierung können Daten nicht nur extrahiert, sondern auch neu zugewiesen werden. So kann der Stand der Bevölkerung von Frankreich auf 58 (Millionen) erhöht werden durch

```
> landesdaten["Frankreich", "Bev"] <- 58
```

◁

Rechnen mit Matrizen

Matrizen unterliegen Rechenregeln. Wir wollen diese kurz rekapitulieren und betrachten, wie diese Regeln in S-PLUS umgesetzt sind.

Um Matrizen zu addieren oder subtrahieren, müssen beide Matrizen von der gleichen Dimension sein, gleiche Zeilen– und Spaltenzahl haben. Ein Spezialfall ist eine einzelne Zahl, die bei Addition oder Subtraktion zu jedem Element der Matrix addiert oder von jedem Element subtrahiert wird. (S-PLUS rechnet auch mit Matrizen, deren Dimensionen ein Vielfaches der Dimension einer anderen Matrix sind. Von einem absichtlichen Benutzen dieser Funktionalität ist außer bei einzelnen Zahlen im allgemeinen abzuraten.)

Addition und Subtraktion von Matrizen wird elementweise durchgeführt, so daß zu jedem Element einer Matrix A der Dimension 5x4 das korrespondierende Element der gleichen Zelle in Matrix B addiert bzw. davon subtrahiert wird. Eine Matrix C der Dimension 4x5 kann nicht zu einer Matrix A der Dimension 5x4 addiert werden.

```
> A <- matrix (0:5, 2, 3)
> B <- matrix (seq (0, 10, 2), 2, 3)
> A+B
       0  6  12
       3  9  15
```

Auf die gleiche Art und Weise können elementweise Matrizen multipliziert und dividiert werden.
Matrixmultiplikation im mathematischen Sinne ist komplizierter. Die Multiplikation einer Matrix A der Dimension mxn mit einer Matrix B der Dimension pxq ergibt eine Matrix C der Dimension mxq. Dabei muß die Anzahl der Spalten der ersten Matrix (A) der Anzahl der Zeilen der zweiten Matrix (B) entsprechen, d.h. n muß gleich p sein.
Der Operator für Matrixmultiplikation besteht in S-PLUS aus drei Symbolen, die sich zusammensetzen zum Operator `%*%`.

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

Matrixdivision im eigentlichen Sinne ist nicht möglich. Die Umkehroperation zur Matrixmultiplikation ist die Inversenbildung, so daß eine (beliebige passende) Matrix multipliziert mit einer Matrix D und wieder multipliziert mit der Inversen von D die Ausgangsmatrix ergibt. Zur Bildung der inversen Matrix bietet S-PLUS die Funktion `solve` an, die aufgrund ihrer weiteren Fähigkeiten, Gleichungssysteme zu lösen, diesen Namen erhalten hat. Der Aufruf zur Invertierung einer Matrix A würde demzufolge lauten:

```
> A.Invers <- solve(A)
```

Matrizen verschmelzen

Häufig sollen zu bestehenden Datensätzen weitere Daten hinzugefügt werden. Für Matrizen wie in unserem Fall für `landesdaten` bedeutet dies, eine neue Zeile für ein neues Land oder eine neue Spalte für eine weitere Variable hinzuzufügen.
Ein sehr einfacher Weg um Daten zu editieren besteht darin, den eingebauten Daten–Editor von S-PLUS zu verwenden, der über das Kontextmenu im Object Browser erreichbar ist.

Zur Zeit erlaubt dieser Editor die Veränderung der generellen Struktur eines Datensatzes nicht, so daß keine weiteren Zeilen oder Spalten ergänzt werden können. Wir erstellen daher eine neue Matrix, die die bisherigen Daten und die neuen Daten enthält.
Um zu einer Matrix eine neue Zeile anzubinden, wird die Funktion `rbind` verwendet, für Zeilen verwendet man `cbind` (`rbind` - row bind, `cbind` - column bind). Wir nehmen in unseren Datensatz `landesdaten` die Fläche der einzelnen Staaten auf. Dazu erstellen wir einen Vektor `Flaeche`, der die Zahlen enthält, und fügen ihn den Landesdaten hinzu.

```
> Flaeche <- c(358, 544, 84)
> landesdaten <- cbind (landesdaten, Flaeche)
> landesdaten
                  BSP   Bev  Inflation  Flaeche
     Deutschland  2075  81.1        1.8      358
      Frankreich  1355  57.7        1.7      544
     Oesterreich   197   7.9        2.2       84
```

Man beachte, daß S-PLUS automatisch den Namen der Variable als neues Label hinzugefügt hat.
Eine neue Zeile für die Daten der Schweiz wird in genau der gleichen Weise hinzugefügt. Wir verwenden dazu die Funktion `rbind`.

```
> Schweiz <- c(265, 7.1, 1.8, 41)
> landesdaten <- rbind (landesdaten, Schweiz)
> landesdaten
                  BSP   Bev  Inflation  Flaeche
     Deutschland  2075  81.1        1.8      358
      Frankreich  1355  57.7        1.7      544
     Oesterreich   197   7.9        2.2       84
         Schweiz   265   7.1        1.8       41
```

In dieser Art können auch Matrizen miteinander verbunden werden. Wenn `rbind` verwendet wird, müssen die Zeilen gleich lang sein, im Falle von `cbind` gilt das gleiche für die Spaltenlängen.

4.1.2 Arrays

Arrays oder Felder sind eine mehrdimensionale Erweiterung von Matrizen. Matrizen sind zweidimensionale Strukturen, charakterisiert durch Zeilen und Spalten. Ein Array kann ein Vektor sein, der nur durch einen Index charakterisiert ist, eine Matrix, oder eine dreidimensionale Struktur, die einem Würfel ähnlich ist. Mehr als drei Di-

mensionen sind schwierig vorstellbar, in S-PLUS als Datenstruktur aber realisierbar.
Die allgemeinen Regeln, die wir für Matrizen eingeführt haben, gelten auch für Arrays.
Ein Array wird folgendermaßen erstellt:

`array` (*data*, *dim*)

oder auf Deutsch

`array` (*Daten*, *Dimension(en)*)

Beide Parameter müssen geschlossene Ausdrücke sein, die nicht durch Komma getrennt aufgezählt werden. So kann *data* ein Ausdruck der Form `c(1,4,11,7,4,4)` sein und *dim* ein Vektor der Form `c(2,3)`. Ein dreidimensionales Array der Dimension 3x2x4 (drei Zeilen, 2 Spalten und 4 "Scheiben"), das die Zahlen 1 bis 24 enthält, wird erstellt durch

```
> x <- array (1:24, c(3,4,2))
```

Wenn ein Array auf dem Bildschirm ausgegeben werden soll, kann es nur in zwei Dimensionen dargestellt werden. S-PLUS läßt zuerst den höchsten Index laufen, dann den zweithöchsten, usw., um zweidimensionale Scheiben der mehrdimensionalen Struktur darzustellen.

Beispiel 4.2. Ausgabe von Arrays
Wir erstellen ein dreidimensionales Array und betrachten das Ausgabeformat, das den höchsten Index am schnellsten laufen läßt, während je zwei Indizes vollständig ausgegeben werden, um eine zweidimensionale Scheibe anzuzeigen.

```
> x <- array (1:24, c(3,4,2))
> x
       , , 1
                   [,1]  [,2]  [,3]  [,4]
             [1,]     1     4     7    10
             [2,]     2     5     8    11
             [3,]     3     6     9    12
```

```
, , 2
          [,1]  [,2]  [,3]  [,4]
    [1,]    13    16    19    22
    [2,]    14    17    20    23
    [3,]    15    18    21    24
```

Wenn wir die zweite Scheibe der zweiten Dimension des Arrays sehen wollten, könnten wir dies tun, indem der zweite Index fixiert wird.

```
> x[,2,]                          # Fixiere 2. Index
          [,1]  [,2]
    [1,]     4    16
    [2,]     5    17
    [3,]     6    18
```

◁

Hinweis An Arrays können Label angefügt werden wie an Matrizen. Die Funktion `dimnames` kann analog wie zuvor bei Matrizen verwendet werden. ◁

apply

Die Funktion `apply` bietet eine sehr elegante Möglichkeit, mit Arrays (und daher auch mit Matrizen) zu arbeiten. Eine Funktion kann auf alle Elemente eines Arrays angewandt werden, und der Benutzer ist in der Lage, zu spezifizieren, wie durch das Array gelaufen werden soll. So kann bei einer Matrix Spalte für Spalte der Mittelwert ausgerechnet werden oder in einem Array für jede Matrix, die sich ergibt, wenn durch den dritten Index gelaufen wird, der Mittelwert subtrahiert und durch die Standardabweichung dividiert werden, um die Daten zu standardisieren.
Die Syntax der Funktion `apply` lautet:

`apply (`*data*`,` *dim*`,` *function*`, ...)`

oder auf Deutsch

`apply (`*Daten*`,` *Dimension(en)*`,` *Funktion*`, ...)`

Wir können uns ein besseres Verständnis für die Anwendung von `apply` verschaffen, indem wir die Funktion auf den zuvor erstellten Datensatz `landesdaten` anwenden.

Beispiel 4.3. Anwendungen von `apply`
Mit der Funktion `apply` berechnen wir das Maximum über jede einzelne Spalte des Datensatzes `landesdaten`, den wir zuvor erstellt haben. Wir errechnen somit das größte Bruttosozialprodukt, die höchste Bevölkerungszahl, die höchste Inflation und die größte Fläche.
Daß die Anwendung auf die Spalten der Matrix erfolgen soll, wird durch die Zahl 2 ausgedrückt, die sich auf die zweite Dimension des Arrays bezieht.

```
> apply (landesdaten, 2, max)
      BSP   Bev  Inflation  Flaeche
     2075  81.1        2.2      544
```

◁

Für alle Spalten der Matrix ruft `apply` der Reihe nach `max` auf. Von `max` wird je ein Wert pro Aufruf zurückgegeben, und alle Werte gemeinsam werden von `apply` als Vektor retourniert.
In dieser Art kann jede Funktion in S-PLUS, die die entsprechende Eingabe akzeptiert, mittels `apply` aufgerufen und angewandt werden.

4.1.3 Data Frames

Data Frames sind eine weitere Verallgemeinerung von Matrizen. Sie erlauben insbesondere, verschiedene Datentypen wie Zahlen, Zeichenketten, Faktoren und andere in einer Matrixstruktur abzuspeichern. Bisher könnten wir nicht Geschlecht (als "m" und "w" codiert) und Alter in einer Matrix ablegen, doch mit Data Frames ist dies möglich. SAS–Benutzer kennen diesen Datentyp vielleicht bereits als Data Set.
Ein Data Frame wird erstellt, indem die zugehörigen Elemente, im allgemeinen Vektoren, Matrizen oder Data Frames, aufgezählt werden. Die Syntax lautet

`data.frame (`*Element1*`, `*Element2*`, ....)`

Wir führen unsere Länderdaten fort und erweitern sie um eine neue Variable, die nicht als Zahl ausgedrückt wird. Wir können festhalten, welche Staaten zur Europäischen Union gehören und welche nicht (Stand 1997). Die Staaten Deutschland, Frankreich und Österreich erhalten das Attribut "EU", die Schweiz bekommt das Attribut "Nicht–EU".

Wenn wir die neue Variable mit `cbind` zur bestehenden Matrix `landesdaten` hinzufügen, passiert folgendes:

```
> EU <- c("EU", "EU", "EU", "nicht-EU")
> landesdaten.neu <- cbind (landesdaten, EU)
> landesdaten.neu
                                    Infla
                   BSP       Bev     tion  Flaeche          EU
 Deutschland    "2075"    "81.1"    "1.8"    "358"        "EU"
   Frankreich   "1355"    "57.7"    "1.7"    "544"        "EU"
 Oesterreich     "197"     "7.9"    "2.2"     "84"        "EU"
      Schweiz    "265"     "7.1"    "1.8"     "41"  "nicht-EU"

> apply (landesdaten.neu, 2, max)
      Error in FUN(x):
      Numeric summary undefined for mode "character"
      Dumped
```

Die Matrix wurde erstellt, aber S-PLUS weigert sich, auf Zeichenketten einen Mittelwert zu errechnen. Es folgt die gleiche Anwendung mit einem Data Frame.

Beispiel 4.4. Erstellen eines Data Frames
Wir erstellen einen Data Frame der Länderdaten einschließlich der Angabe EU/nicht–EU.

```
> EU <- c("EU", "EU", "EU", "nicht-EU")
> laender.dataframe <- data.frame (landesdaten, EU)
> laender.dataframe
                               Infla
                 BSP    Bev     tion   Flaeche         EU
  Deutschland   2075   81.1      1.8       358         EU
   Frankreich   1355   57.7      1.7       544         EU
  Oesterreich    197    7.9      2.2        84         EU
      Schweiz    265    7.1      1.8        41   nicht-EU
```

Jetzt können wir auf die neu erstellten Daten die Funktion `max` spaltenweise anwenden.

```
> apply (laender.dataframe, 2, max)
       BSP   Bev  Inflation  Flaeche
      2075  81.1        2.2      544
```

Man sieht, daß S-PLUS automatisch erkennt, daß über die Variable EU kein Maximum zu bilden ist. Das Ergebnis zeigt lediglich die Maxima aller numerischen Variablen an. ◁

Hinweis Innerhalb eines Data Frames können numerische Funktionen nur auf numerische Daten angewandt werden. Im allgemeinen erkennt S-PLUS die Anwendbarkeit. ◁

Die Variablen unseres Datensatzes können in Matrixnotation adressiert werden. Auf das Bruttosozialprodukt kann zugegriffen werden mit `laender.dataframe [, "BSP"]`, auf die Bevölkerungszahlen mit `laender.dataframe [, "Bev"]`, usw. Ebenso können die Variablen in Listennotation adressiert werden durch `laender.dataframe$BSP`, `laender.dataframe$Bev`, etc. Zu Listen mehr im folgenden Kapitel. Um auf die Variablen direkt mit ihrem Namen zugreifen zu können, bietet S-PLUS die Funktion `attach` mit ihrem Gegenstück `detach` an. Nachdem ein Datensatz "attached" ist, kann auf ihn zugegriffen werden wie auf ein Verzeichnis, er taucht im Suchpfad auf und seine Variablen sind S-PLUS bekannt.

Beispiel 4.5. Bearbeiten eines Data Frames mit `attach` und `detach`
Der Datensatz `laender.dataframe` kann seine Variablen direkt zur Verfügung stellen, wenn sie mit `attach` zugreifbar gemacht werden.

```
> attach (laender.dataframe)
```

Auf die Variable `Bev` greifen wir nun nur mit ihrem Namen zu, ohne `laender.dataframe [, "Bev"]` verwenden zu müssen.

```
> Bev
      Deutschland  Frankreich  Oesterreich  Schweiz
             81.1        57.7          7.9      7.1
> EU
      "EU" "EU" "EU" "nicht-EU"
```

Für die Variable `EU` werden keine Ländernamen angezeigt wie zuvor bei `Bev`. Jetzt gilt es aufzupassen! Wir haben zuvor eine Variable mit dem Namen `EU` erstellt, die sich als separate Variable in der interaktiven Sitzung befindet. Zusätzlich können wir mit dem gleichen Namen auf die Variable `EU` zugreifen, die Teil des Datensatzes `laender.dataframe` ist! Hier zeigt S-PLUS die Daten an, die zuerst gefunden werden, in unserem Falle die interaktiv definierten Daten. Eine Variable namens `Bev` hatten wir nicht separat definiert.

Wir sollten sicherheitshalber ein `detach` durchführen, die nicht mehr benötigten Daten `EU` löschen und danach `laender.dataframe` erneut attachen.

```
> detach()
> rm (EU)                          # Löschen der Variablen
> attach (laender.dataframe)
> EU                               # neuer Versuch
    Deutschland  Frankreich  Oesterreich   Schweiz
             EU          EU           EU  nicht-EU
> detach()
> EU
     Error: Object "EU" not found
```

◁

Hinweis Wenn eine Variable gleichen Namens mehrfach vorkommt, als S-PLUS–Funktion, als eigenständige Variable und als Teil eines Data Frames, der mit `attach` bearbeitet wird, findet S-PLUS das erste Vorkommen dieses Namens. Globale Variablen werden vor Variablen innerhalb eines Data Frames gefunden. ◁

Ein weiterer Quervergleich für SAS–Anwender: Der set–Befehl in SAS hat eine Reihe von Ähnlichkeiten zum Befehl `attach` in S-PLUS.

4.1.4 Listen

Listen sind die flexibelste Datenstruktur, um auch völlig heterogene Daten in einer Variablen zusammenzubinden. Eine Liste besteht aus einer Anzahl von Elementen, die nichts gemeinsam haben müssen. So kann das erste Element einer Liste eine Matrix der Dimension 100x7 sein und Zahlen enthalten, das zweite Element kann ein Vektor der Länge 17 mit Zeichenketten sein und das dritte Element könnte wieder eine Liste sein, die aus mehreren Elementen besteht.
Der Befehl, um eine Liste zu erstellen, lautet

`list (`*Element1*`, `*Element2*`, ....)`

Der Befehl `list` erstellt eine Liste, die die Teilelemente zu einer neuen Variablen zusammenfügt, ohne die Elemente direkt miteinander zu verknüpfen. Elemente werden hintereinander angefügt und mittels

doppelter Klammern adressiert. Das erste Element wird durch *Variablenname*`[[1]]` angesprochen, das zweite durch *Variablenname*`[[2]]`, etc.
Wird eine Liste durch die Eingabe von

`list (`*Name1*`=`*Element1*`, `*Name1*`=`*Element2*`, ....)`

erstellt, können die Elemente auch durch *Variablenname*`$`*Name1*, *Variablenname*`$`*Name2*, etc. adressiert werden.
Wir erweitern unsere Daten der europäischen Länder, die wir bisher erstellt haben. Wir fügen die Information hinzu, welche Länder in Europa Deutsch als Amtssprache haben. Die Länder sind Belgien, Deutschland, Liechtenstein, Luxemburg, Österreich und Schweiz. Wir erstellen einen Vektor, der die Namen der Länder enthält, und fügen den Vektor und die bisherigen Daten, die in einem Data Frame abgespeichert sind, in eine Liste ein.

Beispiel 4.6. Erstellen einer Liste

```
> deutschsprachig <- c("Belgien", "Deutschland",
+ "Liechtenstein", "Luxemburg", "Oesterreich",
+ "Schweiz")
> laender.liste <- list (Basisdaten=laender.dataframe,
+ sprache.d=deutschsprachig)
> laender.liste

$Basisdaten:
              BSP   Bev  Inflation  Flaeche        EU
 Deutschland  2075  81.1        1.8      358        EU
 Frankreich   1355  57.7        1.7      544        EU
 Oesterreich   197   7.9        2.2       84        EU
 Schweiz       265   7.1        1.8       41  nicht-EU

$sprache.d:
            "Belgien"    "Deutschland"  "Liechtenstein"
            "Luxemburg"  "Oesterreich"  "Schweiz"
> length(laender.liste$sprache.d)
      6
> laender.liste$sprache.d[length(laender.liste$sprache.d)]
      "Schweiz"
```

Wir greifen auf einzelne Elemente der Liste zu, indem wir die Indizes der Liste mit Doppelklammern adressieren, wie in `[[1]]` oder `[[2]]`, und die Indizes der Listenelemente wie gewohnt als Matrix– oder Vektorkoordinaten eingeben.

```
> laender.liste[[1]]["Oesterreich", "Bev"]
     7.9
> laender.liste[[1]]["Oesterreich", "EU"]
     EU
     Levels:
     "EU" "nicht-EU"
```

Natürlich können wir auf Listenelemente alle Funktionen anwenden, die wir anwenden könnten, wären die Daten nicht in einer Liste. Wie zuvor verwenden wir auch hier `apply`, um das Maximum jeder Variable zu errechnen.

```
> apply (laender.liste[[1]], 2, max)
        BSP   Bev  Inflation  Flaeche
       2075  81.1        2.2      544
```

◁

Diese Einführung soll zunächst genügen, um mit Listen zu arbeiten. Da Listen eine große Flexibilität bieten, die in vielen Anwendungen genutzt wird, gehen wir in Kapitel 8.7 (Seite 240) ausführlich auf weitere Möglichkeiten ein.

4.2 Fehlende Werte: Eine Kurze Einführung

Fehlende Werte können von S-PLUS durch nicht erlaubte Berechnungen generiert werden, sie können aber auch vom Benutzer eingegeben werden, um zu dokumentieren, daß ein Wert nicht zur Verfügung steht. Würden wir beispielsweise das Bruttosozialprodukt von Österreich nicht zur Verfügung haben, müßten wir diese Zelle mit der Angabe "nicht vorhanden" füllen können. Zu diesem Zweck kennt S-PLUS eine spezielle Variable namens `NA`, eine Abkürzung für Not Available oder nicht vorhanden.
Mit fehlenden Werten läßt sich umgehen wie mit anderen Daten, S-PLUS erkennt diese Codierung und verarbeitet sie entsprechend. So können wir das Bruttosozialprodukt von Österreich als fehlend markieren, indem wir das entsprechende Feld auf den Wert `NA` setzen.

```
> laender.dataframe["Oesterreich", "BSP"] <- NA
```

```
> laender.dataframe[ , "BSP"]
       Deutschland  Frankreich  Oesterreich  Schweiz
              2075        1355           NA      265
```

Wir gehen zu einem späteren Zeitpunkt im Detail auf fehlende Werte ein. Viele Funktionen in S-PLUS reagieren speziell auf `NA`–Werte, so daß es sinnvoll ist, bereits jetzt einen kurzen Einstieg in das Thema zu haben.
Nachdem in den BSP–Daten einer der Werte fehlend ist, reagiert die Funktion `max` darauf.

```
> max (laender.dataframe[,"BSP"])
    NA
```

Die dahinterstehende Regel lautet, daß alle Operationen, in denen ein oder mehrere Daten `NA`–Werte sind, als Ergebnis ein `NA` zurückgeben. Damit der Programmieraufwand nicht zu groß wird, indem zuerst alle fehlenden Werte aus einem Datensatz gefiltert werden, bevor das Maximum errechnet wird, besitzt die Funktion `max` (wie viele andere auch) einen Parameter `na.rm`, dessen Default auf `F` (`FALSE`) gesetzt ist. Wird er auf `T` gesetzt, berechnet S-PLUS das Maximum nach Entfernung aller fehlenden Werte.

```
> max (laender.dataframe[ , "BSP"], na.rm=T)
    2075
```

Fehlende Werte können entdeckt und entfernt werden, indem jeder Wert eines Datensatzes mit der Funktion `is.na` darauf überprüft wird, ob er ein `NA` ist oder nicht.

```
> x <- laender.dataframe[,"BSP"]
> x.na <- is.na(x)
```

Den Vektor `x.na` können wir nun als Index–Selektor verwenden.

```
> max (x)
    NA
> max (x[-x.na])
    2075
> max (x[!x.na])
    2075
```

Hier haben wir den Vektor `x.na` als Indexselektor benutzt, um alle Werte zu selektieren, die nicht gleich `NA` sind (daher die Negation `!`, die auch durch das negative Vorzeichen dargestellt werden kann).

Weitere Details zu fehlenden Werten folgen in Kapitel 6.5 (Seite 159), nachdem wir uns weitere Grundlagen angeeignet haben.

4.3 Anwendungen

Wie im vorherigen Kapitel wollen wir das soeben Gelernte konzentriert wiederholen und auf den Geysir–Datensatz anwenden, um zu sehen, was mit der Kombination von Befehlen erreicht werden kann. Die Stärke der Anwendung von S-PLUS besteht schlußendlich darin, daß man flexibel mit dem System umgehen kann. Dazu sollten die Grundlagen vorhanden sein, um nicht bei jedem Detail im Handbuch nachschlagen zu müssen.
Als erstes hatten wir im vorherigen Kapitel die Daten, die in der Variablen `geyser` gespeichert sind, in zwei separate Vektoren kopiert. In diesem Kapitel haben wir den Befehl `attach` kennengelernt, der auf Data Frames ebenso anwendbar ist wie auf Listen. Nach der Eingabe von `attach` stehen uns die in `geyser` vorhandenen Elemente als eigenständige Daten zur Verfügung.

```
> attach (geyser)
```

Der Datensatz `geyser` ist im Übrigen als Liste abgespeichert, die die Elemente `waiting` und `duration` hat. Ab sofort können wir die Variablen `waiting` und `duration` verwenden. Als erstes erstellen wir eine Matrix mit zwei Spalten. S-PLUS beschriftet die Spalten automatisch mit den Namen `waiting` und `duration`. Wir können die Namen `Wartezeit` und `Dauer` vergeben.

```
> geyser.matrix <- cbind (waiting, duration)
> dim (geyser.matrix)           # Dimension abfragen
    299 2                       # 299 Zeilen und 2 Spalten
> dimnames(geyser.matrix) <-  list (NULL,
+  c("Wartezeit", "Dauer"))
```

Um einen Eindruck von den Daten zu bekommen, fragen wir nach Minimum und Maximum. Wir berechnen Minimum und Maximum für jede Spalte der Matrix, indem wir die Funktion `apply` verwenden.

```
> apply (geyser.matrix, 2, min)
      Wartezeit     Dauer
             43  0.833333
```

```
> apply (both, 2, max)
      Wartezeit  Dauer
            108   5.45
```

Wir könnten vermuten, daß die minimale Wartezeit und die minimale Ausbruchsdauer zu einer Beobachtung gehören, also in einer Zeile der Matrix anzutreffen sind. Dies können wir überprüfen, indem wir den Index der minimalen Wartezeit suchen und die entsprechende Zeile der Matrix selektieren.

```
> Wartezeit.min <- waiting==min(waiting)
> geyser.matrix[Wartezeit.min, ]
      Wartezeit     Dauer
             43  4.333333
```

Die minimale Wartezeit und die minimale Ausbruchslänge gehören daher nicht zusammen. Vielleicht gehört die minimale Wartezeit zur maximalen Ausbruchsdauer ? Eventuell aber auch nur zur zweitlängsten Eruption. Daher wollen wir zwei Datensätze erstellen, die jeweils die kürzesten und längsten Wartezeiten mit den zugehörigen Eruptionsdauern beinhalten.

Wir wählen als untere Grenze 48 Minuten und als obere Grenze 95 Minuten. Nun selektieren wir alle Zeilen der Matrix, in denen die Wartezeit zum einen kürzer als 48 Minuten, zum anderen länger als 95 Minuten ist.

```
> geyser.matrix [waiting < 48, ]
      Wartezeit     Dauer
             43  4.333333
             45  4.416667
             47  4.983333
             47  4.766667
> geyser.matrix [waiting > 95, ]
      Wartezeit     Dauer
            108  1.950000
             96  1.833333
             98  1.816667
             96  1.800000
```

Es wird erkennbar, daß eine hohe Wartezeite mit einer niedrigen Dauer verknüpft ist und umgekehrt.

Hinweis In dieser Anwendung ist zu beachten, daß die Liste namens `geyser` die Variablen `waiting` und `duration` enthält, die wir für unsere Berechnungen verwenden. Die Matrix namens `geyser.matrix`, die als Datengrundlage dient, enthält dagegen zwei Spalten namens `Wartezeit` und `Dauer`. Ein `attach(geyser.matrix)` würde uns erlauben, auf die letzteren zwei Variablen zuzugreifen. Wir wollen aber nicht völlig durcheinandergeraten. ◁

Die beiden Teilmatrizen mit den kleinsten und größten Werten der Wartezeit können wir zu einer Matrix verschmelzen, indem wir Variablen erstellen und die entstehenden zwei Matrizen mit `rbind` zu einer zusammenfügen.

```
> geyser.warte.kurz <- geyser.matrix [waiting < 48, ]
> geyser.warte.lang <- geyser.matrix [waiting > 95, ]
> geyser.extrem <- rbind (geyser.warte.kurz,
+ geyser.warte.lang)
> geyser.extrem
      Wartezeit     Dauer
             43  4.333333
             45  4.416667
             47  4.983333
             47  4.766667
            108  1.950000
             96  1.833333
             98  1.816667
             96  1.800000
```

Schließlich stellen wir fest, daß die Daten der Matrix `geyser.extrem` nicht sortiert sind. Wir würden sie gern nach der ersten Spalte, den Werten für Wartezeit, sortieren. Nur die Wartezeiten zu sortieren, wäre ein einfaches Unterfangen. Wir könnten die Funktion `sort` verwenden.

```
> sort (geyser.extrem[, 1])
     43 45 47 47 96 96 98 108
```

Wir möchten aber die Matrix nach der ersten Spalte sortieren. In diesem Fall können die Ordnungszahlen der Variable `waiting` errechnet und als Index für beide Spalten verwendet werden. Dazu verwenden wir die Funktion `order`.

```
> geyser.extrem.ordnung <- order (geyser.extrem[, 1],
+ geyser.extrem[, 2])
> geyser.extrem.ordnung
     1 2 4 3 8 6 7 5
```

Diese Zahlen geben uns an, daß der kleinste Wert sich in der ersten Zeile befindet, der zweitkleinste in der zweiten und der drittkleinste in der vierten Zeile. Der acht-kleinste oder größte Wert findet sich in Zeile 5. Diese Ordnungszahlen als Index verwendet ergeben eine sortierte Matrix.

```
> geyser.extrem[geyser.extrem.ordnung, ]
      Wartezeit     Dauer
             43  4.333333
             45  4.416667
             47  4.766667
             47  4.983333
             96  1.800000
             96  1.833333
             98  1.816667
            108  1.950000
```

Wir haben der Funktion `order` zwei Argumente übergeben, um zu bewirken, daß zuerst nach der ersten Spalte der Matrix `geyser.extrem` sortiert werden soll. Wenn gleiche Werte auftreten, wird nach der zweiten Spalte sortiert. Für die Wartezeiten 47 und 96 sind zwei Werte vorhanden, so daß jeweils zuerst der Wert mit der kleineren Eruptionsdauer vorkommt.

Für den Moment haben wir genügend neue Dinge gelernt, um eine kleine Pause zu machen. Anschließend wollen wir das in diesem Kapitel gelernte anwenden. Die Aufgaben vertiefen das neue Wissen, indem die Daten der europäischen Länder durch Kombination verschiedener Befehle weiter untersucht werden.

4.4 Aufgaben

Aufgabe 4.1

Der bisher erstellte Datensatz der europäischen Länder beinhaltet eine Reihe interessanter Informationen. Die flächenmässig großen Länder Deutschland und Frankreich sind mit den kleineren Staaten Schweiz und Österreich verglichen worden, ohne irgendeine Normierung auf die Größe vorzunehmen.
Daher soll nun das Bruttosozialprodukt pro Kopf berechnet werden.
Die neue Variable soll Teil des bereits erstellten Data Frames werden, den wir `laender.dataframe` genannt hatten.
Berechne das Maximum für jede der Variablen des Data Frames.
Welches Land hat das höchste Bruttosozialprodukt pro Kopf?
Setze das BSP für Frankreich auf einen fehlenden Wert. Schließe Frankreich von den Berechnungen der Maxima aus und errechne erneut die Maximalwerte.
Welches Land wird nun als das Land mit der größten Fläche angezeigt – stimmt die Angabe?
Wie müssen die Maxima berechnet werden, um die korrekte Lösung zu erhalten?

Aufgabe 4.2

Wir setzen die bisherige Datenanalyse fort.
Der Datensatz `laender.dataframe`, der nun die Variable BSP pro Kopf beinhaltet, soll nach dieser Variable sortiert werden.
Sortiere die Zeilen der Variablen so, daß die Länder in aufsteigender Reihenfolge des BSP pro Kopf angezeigt werden (das Land mit dem kleinsten Bruttosozialprodukt pro Kopf zuerst).
Was muß eingegeben werden, um die Länder in absteigender Reihenfolge des Bruttosozialprodukts zu sortieren? Benutze die von S-PLUS angebotene Hilfe, um die Sortierung in absteigender Reihenfolge vorzunehmen.

Aufgabe 4.3

Wir wollen die Liste der Länder (`laender.liste`) um einen weiteren Eintrag ergänzen. Der Eintrag soll ein Vektor von Zeichenketten sein, der die Länder umfaßt, in denen Französisch die oder eine der Amtssprachen ist.
Als nächstes wollen wir mit diesen Daten von S-PLUS anzeigen lassen, in welchen Ländern Deutsch und Französisch offizielle Sprachen sind.
Dazu soll ein gemeinsamer Vektor der Länder mit Amtssprachen Deutsch und Französisch erstellt werden, der darauf überprüft wird, welche Länder doppelt vorkommen. Diese müssen Deutsch und Französisch als Amtssprache haben.
Hilfestellung: Die Länder, in denen Französisch offizielle Amtssprache ist, sind Belgien, Frankreich, Luxemburg und Schweiz.

4.5 Lösungen

Lösung zu Aufgabe 4.1

Wir wollen das Bruttosozialprodukt pro Kopf für jedes Land berechnen und die neue Variable dem Datensatz `laender.dataframe` hinzufügen. Sie entsteht durch Teilen des Gesamt-Bruttosozialproduktes durch die Bevölkerungszahl.

```
> attach (laender.dataframe)   # Variablen direkt zugreifen
> laender.neu <- data.frame (laender.dataframe,
+ BSP/Bev)
> detach ()                    # und wieder schliessen
> laender.neu
                           Infla  Flae
               BSP   Bev   tion   che        EU   BSP.Bev
 Deutschland  2075  81.1    1.8   358        EU  25.58570
  Frankreich  1355  57.7    1.7   544        EU  23.48354
 Oesterreich   197   7.9    2.2    84        EU  24.93671
     Schweiz   265   7.1    1.8    41  nicht-EU  37.32394
```

Die neue Variable hat einen merkwürdigen Namen bekommen, da S-PLUS versucht hat, diesen aus der Division herauszufiltern. Wir können den Namen neu zuweisen durch die Funktion `dimnames`.

```
> dimnames (laender.neu)[[2]][6] <- "BSP.pro.Kopf"
```

Das gleiche Ergebnis erzielt man, indem beim Zusammenstellen gleich der Variablenname auf die linke Seite eines Gleichheitszeichens geschrieben wird.

```
> laender.neu <- data.frame (laender.dataframe,
+ BSP.pro.Kopf=BSP/Bev)
> laender.neu
                           Infla  Flae             BSP.pro.
               BSP   Bev   tion   che        EU       Kopf
 Deutschland  2075  81.1    1.8   358        EU  25.58570
  Frankreich  1355  57.7    1.7   544        EU  23.48354
 Oesterreich   197   7.9    2.2    84        EU  24.93671
     Schweiz   265   7.1    1.8    41  nicht-EU  37.32394
```

Das Maximum für jede Variable (Spalte) wird mit den Funktionen `apply` und `max` berechnet. Dabei darf die 5. Spalte nicht berücksichtigt werden, da diese keine numerischen Werte enthält.

```
> apply (laender.neu[, -5], 2, max)
      BSP   Bev  Inflation  Flaeche  BSP.pro.Kopf
     2075  81.1        2.2      544      37.32394
```

Für den Wert Fläche ist der Maximalwert von 544 angegeben.

Wir setzen das BSP für Frankreich auf fehlend und berechnen die Maximalwerte neu.

```
> laender.neu[2,1] <- NA
> apply (laender.neu[, -5], 2, max)
      BSP   Bev  Inflation  Flaeche  BSP.pro.Kopf
       NA  81.1        2.2      544      37.32394
```

Mit den nächsten Befehlen werden die Zeilen identifiziert, die fehlende Werte enthalten, und von der erneuten Berechnung des Maximums ausgeschlossen.

```
> BSP.fehlt <- is.na (laender.neu[, "BSP"])
> apply (laender.neu[!BSP.fehlt, -5], 2, max)
      BSP   Bev  Inflation  Flaeche  BSP.pro.Kopf
     2075  81.1        2.2      358      37.32394
```

Bei der Berechnung des Maximums werden die Zeilen nicht berücksichtigt, die einen oder mehrere fehlende Werte beinhalten. In unserem Datensatz gibt es einen fehlenden Wert für Frankreich. Damit werden alle Angaben für dieses Land bei der Berechnung des Maximums nicht berücksichtigt, auch wenn für andere Parameter keine fehlenden Werte vorliegen. Das flächenmäßig größte Land ist Deutschland mit einer Fläche von 358'000 Quadratkilometern. Bei der Betrachtung der Rohdaten ist zu sehen, daß eigentlich Frankreich das größte Land ist. Grund für diese Diskrepanz ist der fehlende Wert beim BSP für Frankreich. Eigentlich wäre nur der Ausschluß des fehlenden Wertes bzgl. des BSP nötig und nicht der Ausschluß der gesamten Zeile. Die Hilfstexte geben für die Funktion `max` an, daß mit dem Parameter `na.rm` die Behandlung von fehlenden Werten gesteuert werden kann und einzelne Werte von der Analyse ausgeschlossen werden können.

```
> help (max)                       # Hilfe aufrufen
> apply (laender.neu[, -5], 2, max, na.rm=T)
      BSP   Bev  Inflation  Flaeche  BSP.pro.Kopf
     2075  81.1        2.2      544      37.32394
```

Lösung zu Aufgabe 4.2

Im folgenden werden wir den Datensatz aus der vorherigen Aufgabe, `laender.neu`, aufsteigend und absteigend nach BSP pro Kopf sortieren. Die Funktion `order` liefert die Indexzahlen der unsortierten Variablen so, daß wenn diese Zahlen als Indizes verwendet werden, die Reihe aufsteigend sortiert ist. Benutzt man die Indizes als Zeilenindizes einer Matrix, wird die gesamte Matrix nach der Variable sortiert, die an `order` übergeben wurde. Wir erstellen die Variable `BSP.order`, die die Indexzahlen enthält, die zu `BSP.pro.Kopf` gehören.

```
> BSP.order <- order (laender.neu[, "BSP.pro.Kopf"])
> BSP.order                          # Vektor sortierter Indizes
     3 1 2 4
```

Werden die Ordungszahlen als Zeilenindex eines neuen Data Frames eingesetzt, resultiert ein Data Frame in gewünschter Folge.

```
> laender.neu.sortiert <- laender.neu[BSP.order, ]
> laender.neu.sortiert
                           Infla  Flae            BSP.pro.
               BSP    Bev   tion   che        EU      Kopf
  Frankreich  1355   57.7    1.7   544        EU  23.48354
 Oesterreich   197    7.9    2.2    84        EU  24.93671
 Deutschland  2075   81.1    1.8   358        EU  25.58570
     Schweiz   265    7.1    1.8    41  nicht-EU  37.32394
```

Es kann nicht häufig genug betont werden, daß die Help–Funktion in S-PLUS benutzt werden sollte. Zu einem Befehl existiert meist ein Hyperlink zu anderen in Verbindung stehenden Befehlen. Zum Beispiel gibt es für die Funktion `order` einen Hyperlink zu der Funktion `rev`. Mit ein wenig Vorstellungskraft kann man erahnen, daß diese Funktion die Reihenfolge umkehrt. Verfolgt man den Hyperlink, erkennt man, daß genau dies die Funktionsweise der `rev`–Funktion ist.

```
> help (order)
> laender.neu [ rev(BSP.order), ]

                           Infla  Flae            BSP.pro.
               BSP    Bev   tion   che        EU      Kopf
     Schweiz   265    7.1    1.8    41  nicht-EU  37.32394
 Deutschland  2075   81.1    1.8   358        EU  25.58570
 Oesterreich   197    7.9    2.2    84        EU  24.93671
  Frankreich  1355   57.7    1.7   544        EU  23.48354
```

Lösung zu Aufgabe 4.3

Wir wollen die Daten der europäischen Länder um die französischsprachigen ergänzen und anschließend mit S-PLUS die Länder suchen, die Deutsch und Französisch als offizielle Amtssprache haben.
Französisch ist in fünf europäischen Ländern Amtssprache oder eine der Hauptsprachen. Selbstverständlich kann die Aufgabe auch ohne dieses Wissen bearbeitet werden.
Die bereits bestehende Liste namens `laender.liste` fassen wir mit dem Vektor der deutschsprachigen Länder und dem Vektor der französischsprachigen Länder zu einer neuen Liste zusammen.

```
> franz.sprachig <- c("Belgien", "Frankreich",
+ "Luxemburg","Schweiz")
> amtssprachen <- list (laender.liste$Basisdaten,
+ deutschsprachig, franz.sprachig)
> amtssprachen
      [[1]]:
                                Infla
                 BSP    Bev      tion   Flaeche           EU
  Deutschland   2075   81.1      1.8        358           EU
   Frankreich   1355   57.7      1.7        544           EU
  Oesterreich    197    7.9      2.2         84           EU
      Schweiz    265    7.1      1.8         41    nicht-EU

      [[2]]:
      "Belgien" "Deutschland" "Liechtenstein"
      "Luxemburg" "Oesterreich "Schweiz"
      [[3]]:
      "Belgien" "Frankreich" "Luxemburg"
      "Schweiz"
```

Es gibt nicht nur einen Lösungsweg um herauszufinden, in welchen Ländern Deutsch und Französisch gesprochen wird. Hier ist einer der vielen Wege.
Wir erstellen einen neuen Vektor von Ländern, der alle Länder aus dem Vektor `deutschsprachig` und zudem alle Länder aus dem Vektor `franz.sprachig` enthält. Die Länder, die beide Sprachen als offizielle Sprachen führen, müssen im neuen Vektor doppelt vorkommen. Ob ein Element in einem Vektor mehr als einmal vorkommt, kann mit der Funktion `duplicated` ermittelt werden.

```
> d.oder.f <- c(deutschsprachig, franz.sprachig)
> doppelt <- d.oder.f [duplicated (d.oder.f)]
```

```
> doppelt
      "Belgien", "Luxemburg" "Schweiz"
> laender.liste2$Basisdaten[doppelt, ]
          BSP  Bev  Inflation  Flaeche        EU
           NA   NA         NA       NA        NA
           NA   NA         NA       NA        NA
Schweiz   265  7.1        1.8       41  nicht-EU
```

Wir erhalten zwei Zeilen, die nur fehlende Werte (`NA`) enthalten, da für die Länder Belgien und Luxemburg keine Basisdaten über Bevölkerung, Bruttosozialprodukt, etc. vorliegen. Die dritte Zeile gibt die Daten des dritten mehrsprachigen Landes an, der Schweiz.