ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Introduction to S-PLUS for UNIX

(with exercises)

Diego Kuonen

Chair of Statistics
Department of Mathematics
Swiss Federal Institute of Technology
1015 Lausanne, Switzerland

`http://statwww.epfl.ch`

Fourth Edition

February 2001

Diego Kuonen
DMA–Ecublens
EPFL
CH–1015 Lausanne

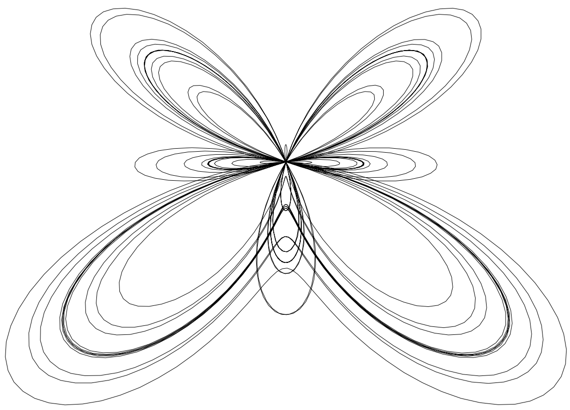
Phone: + 41 (0) 21 693 5508
Fax: + 41 (0) 21 693 4250

`Diego@Kuonen.com`

`http://stat.kuonen.com`

As the ability to cut and paste is very convenient in writing S-Plus code, the examples in this manuscript were developed by cut and paste between the LaTeX source and a window running on S-Plus. The figures were generated in PostScript, and were directly incorporated.

Preface



S-Plus is a powerful tool for interactive data examination, creation of graphs, and implementation of customized routines based on the S language of AT&T Bell Labs (Lucent Technologies). The modern concept together with its flexibility makes it appealing to data analysts from all scientific fields. Among its many strengths are its powerful but flexible graphics routines, and its facility for writing functions whereby you may either modify an existing function, or create one of your own. Most standard statistical techniques and many non-standard ones besides are already programmed in S-Plus, making exploratory and formal statistical analysis very easy.

This guide was written to give a solid but quick introduction to the S-Plus environment. If you have never used S-Plus before, this guide will get you up to speed quickly. So, the present guide is primarily intended for the S-Plus novice. With the provided groundwork in place, it is then possible to tackle the really sophisticated functions and to write your own. If a specific problem is not covered in this guide, you should be able to find the answer in one of the books and manuals on S-Plus.

This manuscript was written to accompany the introduction course to S-Plus given at the Department of Mathematics at the Swiss Federal Institute of Technology in Lausanne. Although it has been written for a particular environment, S-Plus 3.4 for UNIX, I have tried as far as practicable to make the guide independent of the computing environment and the exact version of S-Plus in use. This manuscript is also useful with R, a freely-available open-source statistical system 'not unlike S'. I have tried where possible to use code that works in all versions of S-Plus and in R. In addition to the first edition the present manuscript includes a new chapter on a short comparison of S-Plus 5.x with S-Plus 3.4. Moreover, it is assumed to work with a graphical user interface (GUI) such as Motif or Microsoft Windows, or at a X-terminal. Finally, the webpage of the introduction course to S-Plus and of the present manuscript is

`http://statwww.epfl.ch/splus/`

Many people influenced this manuscript and contributed directly or indirectly. They are far too numerous to mention. I would like to thank all of them. A special thank goes to Christian Sangiorgio from the Seminar für Statistik at the ETHZ in Zürich for providing me with their course material on their 'Einführung in S-Plus' course. Moreover, special thanks go to Prof. Anthony C. Davison and Reinhard Furrer for their valuables comments and suggestions.

Diego Kuonen,
Lausanne, February, 2001.

Typographical Conventions

Throughout this guide S-Plus language constructs and commands are set in typewriter font `like this`, or in using the S-Plus prompt '>' like

```
> S-Plus command
```

or

```
> S-Plus command     # comment on the S-Plus command
```

The prompt `$` is used as the standard prompt for the UNIX shell, e.g.

```
$ UNIX command
```

Finally, keyboard keys are written in capital letters, e.g. press the RETURN key.

Contents

Chapter 1

Introduction

1.1 The History of S and S-Plus

S is a very high level language and an environment for data analysis and graphics. S was written by Richard A. Becker, John M. Chambers, and Allan R. Wilks at Lucent Technologies (formerly AT&T Bell Laboratories) Statistics Research Department. S is a very general tool, so that applications are not restricted to any particular subject area. One way to think of it is to imagine the wide range of applications that can be handled by a spreadsheet program — but think of an even broader range of applications because S is much more flexible for complex computations. As examples, S has been used for computing in areas like business, finance and experimental science.

Of fundamental importance is that S is a language. This makes S much more useful than if it were merely a 'package' for statistics or graphics or mathematics. Imagine if English were not a language, but merely a collection of words that could only be used individually — a package. Then what is expressed in this sentence is far more complex than any meaning that could be expressed with the English package. That is why the authors of S prefer that you not call S a statistics package. Most of the people who use S have no attachment to statistics, and most of the S applications involve basic quantitative computations and graphics. In S, all functions are tightly integrated and controlled by the S language.

S-Plus is a value-added version of S sold by Insightful Corporation (formerly MathSoft, Inc.), and it is a fully supported and documented application which has been compiled and tested on numerous architectures. It is available in both UNIX and Windows versions. The latest version of S-Plus is even running on the most affordable platform: Linux. In all, S-Plus offers a very comprehensive set of functions with over 4,000 built-in functions.

S is a subset of S-Plus, and hence anything which may be done in S may be done in S-Plus. In addition S-Plus has extended functionality in a wide variety areas, including robust regression, modern nonparametric regression, time series, survival analysis, multivariate analysis, classical statistical tests, quality control, graphics drivers and many more. Add-on modules add additional capabilities for wavelet analysis, spatial statistics, financial analysis and design of experiments. In addition, the newer S-Plus versions for Windows introduces a full-featured graphical user interface.

An electronic source for information on S-Plus is the Insightful Corporation web site. The URL for this site is

`http://www.insightful.com`

1.2 Award-winning S Language

The S System was recently recognized with the prestigious 'Association for Computing Machinery (ACM) Software Systems Award'. The ACM named John M. Chambers of Bell Labs as the recipient of the 1998 'Software System Award' for developing the S System. The ACM's citation notes that Chambers's work '*forever altered the way people analyze, visualize, and manipulate data... S is an elegant, widely accepted, and enduring software system, with conceptual integrity, thanks to the insight, taste, and effort of John Chambers.*' Note that the 'System Software Award' recognizes those who develop software systems having a lasting influence. Software systems recognized by previous awards have included such cornerstones of modern computing as the UNIX operating system, the World Wide Web, TCP/IP, and the Postscript language.

1.3 Available Documentation

There are a variety of ways to accelerate your progress with S-Plus, and to build upon the work of others. This section describes the learning and support resources available to S-Plus users.

1.3.1 Online Help

S-Plus offers an online help system to make learning and using S-Plus easier. Moreover, the UNIX version of S-Plus 3.4 provides a convenient menu-driven help system, while in the Windows version help is provided through the Windows Help system. The latest UNIX version of S-Plus 5.1 introduces a new help system which displays help as HTML-formatted text (see also Chapter 10).

1.3.2 Printed and Online Manuals

All S-Plus versions come with several manuals. For instance the UNIX/Linux version comes with the 'User's Guide', the 'S-Plus Guide to Statistics (Part I and II)', and the 'S-Plus Installation and Maintenance Guide'. The entire S-Plus documentation for Windows and UNIX/Linux is also available from the following Insightful Corporation web site

`http://www.insightful.com/resources/doc/`

A collection of whitepapers and application notes on S-Plus you will at

`http://www.insightful.com/resources/whitepapers/`

and the 'S-Plus Instructor's Corner' — a resource for instructors teaching data analysis and statistics using S-Plus — under

`http://www.insightful.com/resources/inst-corner/`

An additional online book that is geared towards those who are programming in S is

`http://www.seanet.com/~pburns/Spoetry/`

Finally, a HTML-based help system for S-Plus 3.4 you will find at the URL

`www.uni-muenster.de/ZIV/Mitarbeiter/BennoSueselbeck/s-html/shelp.html`

1.3.3 StatLib

StatLib is a system for distributing statistical software, data sets, and information by electronic mail, FTP and the World Wide Web. It contains a wealth of user-contributed S-Plus functions. To access StatLib by FTP, open a connection to: `lib.stat.cmu.edu`. Login as `anonymous` and send your e-mail address as your password. The FAQ ('Frequently Asked Questions') is in `/S/FAQ`, or in HTML format at

`http://www.stat.math.ethz.ch/S-FAQ/`

To access StatLib with a web browser, visit

`http://lib.stat.cmu.edu`

To access StatLib by e-mail, send the message:

```
send index from S
```

to `statlib@lib.stat.cmu.edu`. You can then request any item in StatLib with the request `send item from S` where `item` is the name of the item.

1.3.4 S-News

S-news is an electronic mailing list by which S-PLUS users can ask questions and share information with other users. To get on this list, send a message with message body `subscribe` to

`s-news-request@wubios.wustl.edu`

To get off this list, send a message with body `unsubscribe` to the same address. Once enrolled on the list, you will begin to receive e-mail. To send a message to the S-news mailing list, send it to `s-news@wubios.wustl.edu`. Do not send subscription requests to the full list; use the `s-news-request` address shown above. The S-news mailing list archives are available at

`http://www.biostat.wustl.edu/s-news/s-news-archive/`

1.3.5 Other Books

The primary references for S are the books by its creators:

Becker R. A., Chambers J. M. and Wilks A. R. (1988), *The New S Language*, Chapman and Hall, London.

Chambers J. M. and Hastie T. J. (1992), *Statistical Models in S*, Chapman and Hall, London.

Chambers J. M. (1998), *Programming with Data*, Springer, New York.

Other books which discuss particular aspects of S and S-PLUS include the following:

Bowman A. W. and Azzalini A. (1997), *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations*, Oxford University Press, Oxford.

Cleveland W. (1993), *Visualizing Data*, Hobart Press, Summit, New Jersey.

Davison A. C. and Hinkley D. V. (1997), *Bootstrap Methods and Their Application*, Cambridge University Press, Cambridge.

Everitt B. (1994), *A Handbook of Statistical Analyses using S-Plus*, Chapman and Hall, London.

Härdle W. (1991), *Smoothing Techniques with Implementation in S*, Springer, New York.

Krause A. and Olson M. (1997), *The Basics of S and S-Plus*, Springer, New York.

Marazzi A. (1993), *Algorithms, Routines and S Functions for Robust Statistics*, Chapman and Hall, London.

Spector P. (1994), *An Introduction to S and S-Plus*, Duxbury Press, Belmont, California.

Venables W. N. and Ripley B. D. (1999), *Modern Applied Statistics with S-Plus*, Third Ed., Springer, New York.

Venables W. N. and Ripley B. D. (2000), *S Programming*, Springer, New York.

They go far beyond the coverage of this guide, including many topics (such as robust statistics, non-linear regressions, modern regression, survival analysis, tree-based models, time series and spatial statistics) not covered here, as well as in greater depth on what is covered. Moreover, for readers of German there are:

Böker F. (1997), *S-Plus – Learning by Doing: Eine Anleitung zum Arbeiten mit S-PLUS*, Verlag Lucius & Lucius, Stuttgart.

Krause A. (1997), *Einführung in S und S-Plus*, Springer Verlag, New York.

Suselbeck B. (1993), *S und S-Plus: Eine Einführung in Programmierung und Anwendung*, Gustav Fischer Verlag, Jenna.

For readers of French there is:

Baumgartner M. (1994), *Une introduction à S-Plus Version 3.1 pour Silicon Graphics Iris*, Département de Mathématiques, Chaire de Statistique, EPFL, CH-1015 Lausanne.

The later can be downloaded from the course home page:

`http://statwww.epfl.ch/splus/`

An up-to-date list of texts and books on S and S-Plus is:

`http://www.insightful.com/resources/biblio.html`

1.3.6 Statistics Related Information

Statistics related information you will find at

`http://www.statoo.com`

On this site you can find easily and quickly anything related to statistics.

1.4 Essentials

This section covers the basics of starting S-Plus, of constructing S-Plus expressions, of getting help by use of the built-in help system, and some general information.

1.4.1 Getting Started

To start S-Plus, type the following at the UNIX shell prompt, and press the RETURN key:

```
$ Splus
```

Only the 'S' is capitalized. When you press RETURN, a copyright message appears:

```
S-PLUS : Copyright (c) 1988, 1996 MathSoft, Inc.
S : Copyright AT&T.
Version 3.4 Release 1 for Silicon Graphics Iris, IRIX 5.3 : 1996
Working data will be in .Data
>
```

You get the S-Plus prompt '>', waiting for input from you. You can use S-Plus by typing expressions after the prompt and pressing the RETURN key. Among the simplest S-Plus expressions are arithmetic ones such as

```
> 8+9
[1] 17
> 8*9
[1] 72
```

The symbols '+' and '*' represent S-Plus operators for addition and multiplication, respectively. In addition to the usual arithmetic and logical operators, S-Plus has special operators for special purposes. For example, the colon operator ':' is used to obtain sequences:

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

The [1] in each of the output lines is the index of the first S-Plus response on the line of S-Plus output. If S-Plus is responding with a long vector of results, each line is preceded by the index of the first response of that line. The most common S-Plus expression is the function call. An example of a function in S-Plus is the c function, used for 'combining' comma-separated lists of items into a single item. Functions calls are always followed by a pair of parentheses, with or without any arguments in the parentheses. An example is given by

```
> c(1,5,8,9)
[1] 1 5 8 9
```

Let us now introduce some basic typing syntax and conventions in S-Plus. Note that S-Plus ignores most spaces, for example:

```
> 8+      9
[1] 17
```

However, do not put spaces in the middle of numbers or names. For example, if you wish to add `321` and `1`, the expression `32 1+1` causes an error. Also, when you type a RETURN and it is clear to S-Plus that an expression is incomplete (for example, the last character is an operator, or there is a missing parenthesis), S-Plus provides a continuation prompt to remind you to complete the expression. The default continuation prompt is '`+`'. Here are two examples of incomplete expressions

```
> 8*
+ 9
[1] 72
> c(1,5,
+ 8,
+ 9
+ )
[1] 1 5 8 9
```

In each of the above cases, the user completed the expression after the continuation prompt, '`+`', and then S-Plus responded with the result of the evaluation of the complete expression.

Sometimes you may want to stop the evaluation of an S-Plus expression. For example, you may suddenly realize you want to use a different command, or the output display of data on the screen is extremely long and you do not want to look at all of it. To interrupt S-Plus, use the UNIX interrupt command, which on most systems consists of CTRL-C (pressing the C key while holding down the CONTROL key).

Elementary commands consist of either expressions or assignments. If an expression is given as a command, it is evaluated, printed, and the value is discarded. An assignment also evaluates an expression and passes the value, which can be changed, but the default is assumed here. In fact it is kept in the (hidden) variable `.Last.value`:

```
> .Last.value
[1] 1 5 8 9
```

So far, in all of the examples, S-Plus has simply returned a value. To reuse the value of an S-Plus expression, you must assign it with the '`<-`' operator. For example, to assign the expression `c(1,5,8,9)` to an S-Plus object named `myvector`, you would type

```
> myvector <- c(1,5,8,9)
```

S-PLUS creates the object `myvector` and returns an S-PLUS prompt. To view the contents of the newly created object, just type its name

```
> myvector
[1] 1 5 8 9
```

Like most UNIX-based packages S-PLUS is case sensitive, and so all S-PLUS objects, arguments, names, and so on. In the following example, you get an error message if you do not type `myvector` exactly as stated, including matching all upper case and lower case letters.

```
> Myvector
Error: Object "Myvector" not found
> MYVECTOR
Error: Object "MYVECTOR" not found
```

To get a list of all the S-PLUS objects in your `.Data` directory — the S-PLUS working directory, where all the objects you create are stored — type

```
> ls()
[1] "myvector"
```

A more sophisticated version of `ls` is the function `objects`, which can be used to list only a subset of the objects. To find all the objects starting with '`m`' in the working directory, use

```
> objects(pattern="^m")
[1] "myvector"
```

To remove a object, say `myvector`, from this directory, type

```
> rm(myvector)
```

It is important to note that objects in your `.Data` directory will take precedence over system objects of the same name. This is a frequent cause of rather obscure errors, and can cause apparently correct behavior but erroneous results. Avoid using names such as `c`, `s`, `t`, etc. for your own objects. Say you want to create an object called `t`:

```
> t <- c(1,5,8,9)
> t
[1] 1 5 8 9
```

Everything seems to work fine, but in removing `t` from your `.Data` directory and in recalling `t`,

```
> rm(t)
> t
function(x)
UseMethod("t")
```

you see that this object already exists — as we will see later, it returns a matrix which is the transpose of the input matrix. Note that this is a generic function (`UseMethod`), meaning that the action depends on the type of object given as an argument. These functions are utility functions for object-oriented programming. The idea of object-oriented programming is simple, but carries a lot of weight. Here is the whole thing: if you told a group of people to dress for work, then you would expect each to put on clothes appropriate for that individual's job. Likewise it is possible for S objects to get dressed appropriately depending on what class of object they are. Object-orientation simplifies. If you want to print an object, you do not need to find out what type of object it is, then try to remember the proper function to use on that type of object, and then do it. You merely use `print` and the right thing happens, e.g.

```
> print(myvector)
[1] 1 5 8 9
```

It can also simplify programming. Programming is simplified for the reason above, and it also suggests a proper ensemble of functions to write for a particular application. Although technical information is more sparse than one would hope at the S-Plus site of MathSoft, there is a very useful article on this issue by Richard Calaway:

`http://www.mathsoft.com/spress/Oop.htm`

1.4.2 Quitting

Finally, to quit S-Plus and get back to UNIX, use the `q` function:

```
> q()
```

The '`()`' are required with the `q` command to quit S-Plus because `q` is an S-Plus function. Parentheses are required with all S-Plus functions.

1.4.3 The Help System

If you need help at any time during an S-Plus session, you can obtain it easily with the `?` and `help` functions. The `?` function has simpler syntax — it requires no parentheses in most instances:

```
> ?t
Matrix Transpose

DESCRIPTION:
       Returns a matrix which is  the  transpose  of  the  input.
       This  is  a  generic  function. In particular, there are t
       methods for various matrix classes in the Matrix library.

USAGE:
       t(x)
       t.default(x)

REQUIRED ARGUMENTS:
x:     matrix.  Missing values (NAs) are allowed.

VALUE:
       transpose of x (rows of x are columns of  the  result,  so
       that t(x)[j, i] is the same as x[i, j]).

SEE ALSO:
       aperm for the generalized transpose of arrays.  solve  for
       the  inverse  of  a  matrix.  library(help = Matrix) for a
       general description of the Matrix library.

EXAMPLES:
       amat <- author.count[1:2,]
       t(amat)
```

The same result is given by `help(t)`. Both `?` and `help` use the less pager (provided with S-Plus) to display the requested help. You can use the 'd' and 'u' keys to page down and up, respectively; use the 'q' key to exit help and return to the S-Plus prompt. The `?` command is particularly useful for obtaining information on classes and methods. If you use `?` with a function call, S-Plus offers documentation on the function name itself and on all methods that might be used with the function if evaluated. In particular, if the function call is `methods(name)`, where `name` is a function name, S-Plus offers documentation on all methods for `name` available in the current search list. For example,

```
> ?methods(t)
The following are possible methods for t
        Select any for which you want to see documentation:

1: t.data.frame
2: t.default
3: t.test
Selection:
```

You enter the number of the desired method and S-Plus prints the associated help file, if it exists — the `?` command does not check for the existence of the help files before constructing the menu. After each menu selection, S-Plus presents an updated menu showing the remaining choices. To get back to the S-Plus prompt from within a `?` menu, enter '`0`'. You call `help` with the name of an S-Plus function, operator, or data set as argument.

To get the most information from the S-Plus help system, you should become familiar with the general arrangement of help files. Help files are organized as follows (not all files contain all sections):

- `DESCRIPTION`. A brief description of the function's main use.
- `USAGE`. Provides the correct syntax for a call to the function. Arguments for which just the argument name is given are required, while arguments stated in the form `name =` value are optional arguments, where the given value is the default value.
- `REQUIRED ARGUMENTS`. Lists arguments required in every call to the function. If not supplied, an error results.
- `OPTIONAL ARGUMENTS`. Lists arguments that may be supplied in a call to the function. If not supplied, default values are used.
- `SIDE EFFECTS`. Lists any effects of the function other than returning a value.
- `DETAILS`. Documents some of the computational details describing the implementation of the function.
- `REFERENCES`. References to scientific literature or books which describe in further detail the methodology or interpretation of the results of this function.
- `SEE ALSO`. Lists related S-Plus functions.
- `EXAMPLES`. Gives examples of use of the function.

Under the X11 window system, the help file can be forced to appear in its own window,

```
> help(t, window=T)
```

where `T` stands for TRUE. Assumed that the printer environment variable is set, the help page can be printed by

```
> help(t, offline=T)
```

For a feature specified by special characters care is needed. For example:

```
> help(+)
Syntax error: ")" used illegally at this point:
help(+)
```

In such cases the argument must be included in double (`""`), or single quotes (`''`), making it an entity known in S-Plus as character string. Two alternative ways of getting help on the addition operator are

```
> help("+")
> ?"+"
```

The S-Plus version 3.4 for UNIX has also a window system, which allows you to search for help files by topic and to display them. You can look at many help files at the same time and keep them on the screen as you continue working. Type

```
> help.start(gui='motif')
```

S-Plus 5.1 introduces a new help system to replace the nroff-based system used in earlier versions of S-Plus. The new help system displays help as HTML-formatted text, by default using a modified version of the Lynx terminal-based browser. If you have Netscape Navigator on the machine you use to run S-Plus, you can choose to have help displayed in your Navigator window. The `help.start` function has been restored, but now uses your Netscape Navigator window to display its menu of function and dataset help files. A short comparison of S-Plus 5.x with S-Plus 3.4 is given in Chapter 10.

1.4.4 General Information

All the objects you create in S-Plus are stored in a directory `.Data`. Note that as the directory name `.Data` begins with '`.`' it will normally be hidden in file listings from UNIX by `ls`. To see it, type the following at the UNIX shell prompt:

```
$ ls -al
```

If there is a `.Data` directory in the current directory when S-Plus is invoked, that directory is used rather than the directory of your home directory, or such a directory is created with a warning:

```
Initializing login directory for new S-PLUS user.
```

The following provides one way to organize your S-Plus, using separate directories for each project, e.g. suppose you want to have a new working directory for S-Plus, say `mysplus`:

```
$ mkdir mysplus
```

Now, change to this directory:

```
$ cd mysplus
```

and make a subdirectory of `mysplus` called `.Data`:

```
$ mkdir .Data
```

This subdirectory is for use by S-Plus itself. Note that this directory only has to be created once. An advantage of S-Plus is that all created objects are stored in the directory `.Data`, and they are retained until explicitly deleted. So, in in contrast to other statistical softwares, where you have to explicitly state what you want to save before quitting, the created S-Plus objects remain permanently even after quitting S-Plus. By the time your `.Data` directory may contain a few hundred objects, and so you should clean up your `.Data` directory from time to time in order to not be yelled by your system administrator for hogging all the disk-space.

S-Plus keeps a record of all commands in the `.Audit` file in your `.Data`. This is a hidden file and can grow rather fast as well. To clean out the `.Audit` file entirely, type

```
$ Splus TRUNC_AUDIT 0
```

The following output is produced:

```
Truncating audit file (.Data/.Audit) to most recent 0 characters
Audit file reset
```

If you like to keep the most recent 100000 characters, type

```
$ Splus TRUNC_AUDIT
```

1.4.5 Command Line Editing

Included with S-Plus is a command line editor that can help improve your productivity by enabling you to recall and edit previously issued S-Plus commands. The editor can do either emacs- or vi-style editing. To be valid, the value for the environment variable `S_CLEDITOR` must be 'vi' or 'emacs', i.e. in the UNIX shell type

```
$ setenv S_CLEDITOR emacs
```

To use the command line editor within S-PLUS, start S-PLUS with the following command:

```
$ Splus -e
```

Table 1.1 summarizes the most useful editing commands for the emacs mode of the command line editor.

Table 1.1: Command line editing for the emacs mode in S-PLUS.

Action	**Emacs keystrokes**
backward character	CTRL-B
forward character	CTRL-F
previous line	CTRL-P
next line	CTRL-N
beginning of line	CTRL-A
end of line	CTRL-E
forward word	ESC-F
backward word	ESC-B
kill char	CTRL-D
kill line	CTRL-K
delete word	ESC-D
search backward	CTRL-R
transpose chars	CTRL-T

Moreover, to avoid forgetting to involve these two commands in order to start S-PLUS, you can use an alias, which for C-shell users would look like:

```
alias S 'setenv S_CLEDITOR emacs; Splus -e'
```

As an example of using the command line editor, suppose you have started S-PLUS with the emacs option. Suppose you wanted to get some help on the `plot` function by typing the following:

```
> help(plto)
No documentation available for 'plto'
```

Type CTRL-P to recall the previous line, then use CTRL-B to return to the '`t`' in '`plto`'. Finally, type CTRL-T to transpose the '`t`' and the '`o`'. Press RETURN to issue the edited command.

Chapter 2

Introductory Session: How to Beat the Lottery

One of the best ways to getting acquainted with S-Plus is to use it to help you to understand a particular set of data. So let us consider data issued from the lottery, where you might be motivated to perform data analysis. The readers are invited to work through the following familiarization session and see what happens. First-time users may not yet understand every detail, but the best plan is to type what you see and observe what happens as a result.

This chapter is mainly based on Becker, Chambers and Wilks's book (1988, *The New S Language*, Chapter 1).

The specific data we will look at concerns the New Jersey Pick-It Lottery. Our data is for 254 drawings just after the lottery was started, from May, 1975 to March, 1976. Pick-It is a parimutuel game, meaning that the winners share a fraction of the money taken in for the particular drawing. Each ticket cost fifty cents and at the time of purchase the player picks a three-digit number ranging from 000 to 999. The money bet during the day is placed in a prize pool and anyone who picked the winning number shares equally in the pool.

The data available gives for each drawing the winning number and the payoff for a winning ticket. The winning numbers and the corresponding payoffs are:

```
> lottery.number    # print the winning numbers
> lottery.payoff    # print the payoffs
```

To get a description of the data, type

```
> help(lottery.number)
> help(lottery.payoff)
```

This data are an example of so-called built-in data, which come with S-Plus. A list of all built-in data is given in the printed S-Plus manuals.

From the print of the two data we notice that, for the first drawing, the winning number was 810 and it paid $190.00 to each winning ticket holder. In what follows we will try to examine the data. Numerical summaries provide a statistical synopsis of the data in a tabular format. Such a function is `summary`. The following displays a summary of the lottery payoffs:

```
> summary(lottery.payoff)
 Min. 1st Qu. Median  Mean 3rd Qu.  Max.
   83   194.2  270.2 290.4     364 869.5
```

We read from this that the mean payoff was $290.4, that the payoffs ranged from $83 to $869.5 and that 50% off all payoffs lay between $194.2 and $364. The quantiles for a set of data `x` can also be computed by means of `quantile(x, quantiles)`. For example,

```
> quantile(lottery.payoff, c(.25, .75))
    25% 75%
 194.25 364
```

This means that 25% of the values are less than 194, and 75% of the payoffs are less than 365.

A better way to understand the data is to look at it graphically. To do so, you have to turn on the graphics window. For the X11 window system a possible device is:

```
> motif()
```

A separate window should appear on your screen.

In our data, to detect long-term irregularities we will look at the winning numbers to see if they appear to be chosen at random. To do so we could produce a histogram of the lottery numbers:

```
> hist(lottery.number)
```

The histogram is shown in Figure 2.1. Since there are 10 bars, the count should be approximately 25 bar at the left represents the one time that 000 was the winning number. It looks fairly flat — no need to inform a jury.

Of course, most of our attention will probably be directed at the payoffs. Elementary probabilistic reasoning tells us that a single number we pick has a 1 in 1000 chance of winning. If we play many times, we expect about 1 winning number per 1000 plays. Since a ticket costs fifty cents, 1000 plays will cost $500, so we hope to win at least $500 each time we win, otherwise we will lose money in the long run. So, let us make a histogram

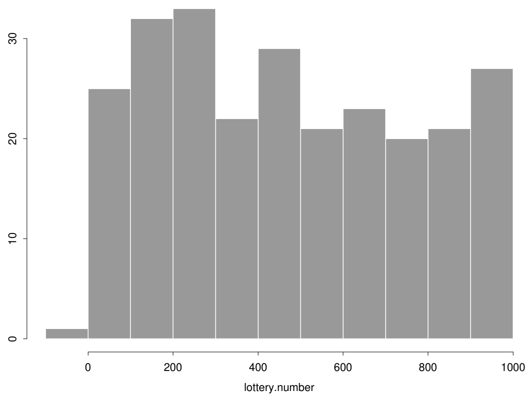


Figure 2.1: Histogram of the winning numbers from 254 lottery drawings.

of the payoffs.

```
> hist(lottery.payoff)
```

Figure 2.2 shows that payoffs range from less than \$100 to more than \$800, although the bulk of the payoffs are between \$100 and \$400, i.e. there were a number of payoffs larger than \$500 — perhaps we have a chance. The widely varying payoffs are primarily due to the parimutuel betting in the lottery: if you win when few others win, you will get a large payoff. If you are unlucky enough to win along with others, the payoff may be relatively small.

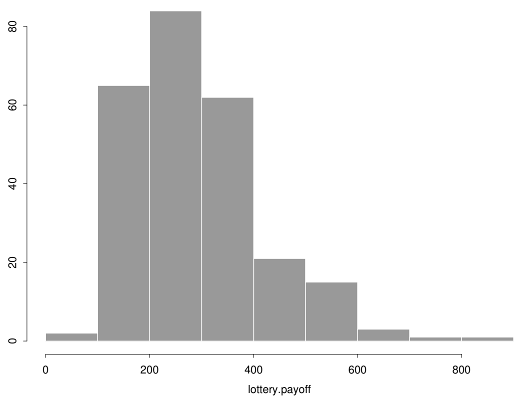


Figure 2.2: Histogram of the lottery payoffs.

Let us see what the largest and the smallest payoffs and corresponding winning numbers were:

```
> max(lottery.payoff)     # the largest payoff
[1] 869.5
> lottery.number[lottery.payoff==max(lottery.payoff)]
```

```
[1] 499
> min(lottery.payoff)     # the smallest payoff
[1] 83
> lottery.number[lottery.payoff==min(lottery.payoff)]
[1] 123
```

Winners who bet on '123' must have been disappointed; \$83 is not a very large payoff. On the other hand \$869.50 is very nice. Since the winning numbers and the payoffs come in pairs, a number and a payoff for each drawing, we can produce a scatterplot of the data to see if there is any relationship between the payoff and the winning number. S-PLUS provides a generic plotting function, `plot`, which produces different kinds of plots depending on the data passed to it. In its most common use, it produces a scatterplot of two numeric objects:

```
> plot(lottery.number, lottery.payoff)
```

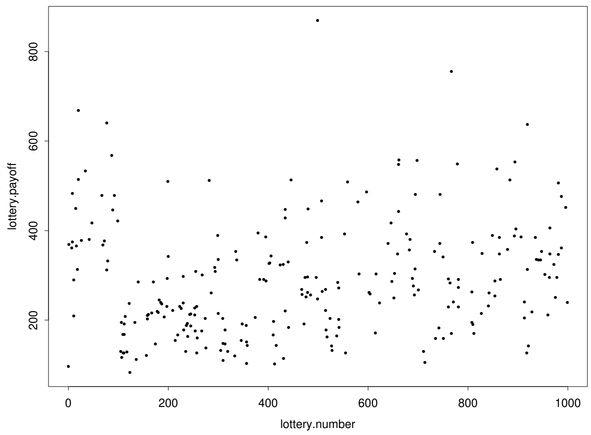


Figure 2.3: Scatterplot of winning number and payoff for the 254 drawings.

What do you see in the Figure 2.3? Does the payoff seem to depend on the position of the winning number? Perhaps it would help to add a 'middle' line that follows the overall pattern of the data:

```
> lines(lowess(lottery.number, lottery.payoff, f=.2))
```

This command superimposes a smooth curve on the winning number and payoff scatterplot. The result is given in Figure 2.4. Can you see the interesting characteristics now?

There are substantially higher payoffs for numbers with a leading zero, meaning fewer people bet on these numbers. Perhaps that reflects people's reluctance to think of numbers with leading zeros. After all, no one writes \$010 on a ten dollar check! Also note that, expect for the numbers with leading zeros, payoffs seem to increase as the winning

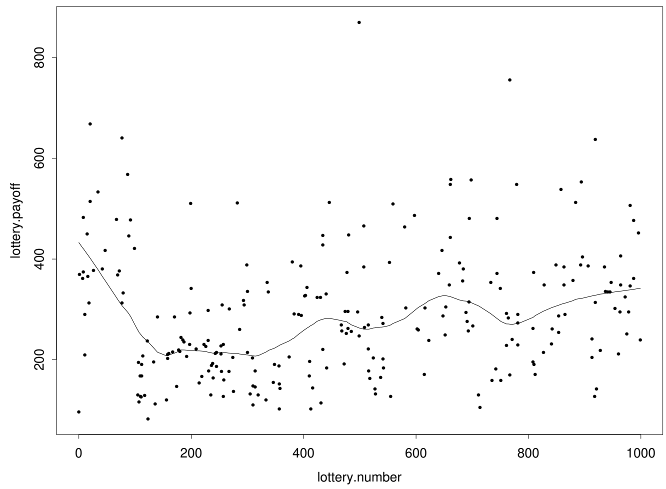


Figure 2.4: A smooth curve is superimposed on the winning number and payoff scatterplot.

number increases. It would be interesting to see exactly what numbers correspond to the large payoffs. Fortunately, with an interactive graphical input device, we can do that by simply pointing at the 'outliers':

```
> identify(lottery.number, lottery.payoff, lottery.number)
```

To identify a payoff with its corresponding winning number just click on a point using the left mouse button. Once you have pointed out the 'outliers' just click on the middle mouse button. Can you see in Figure 2.5 the pattern in the numbers with very high payoffs?

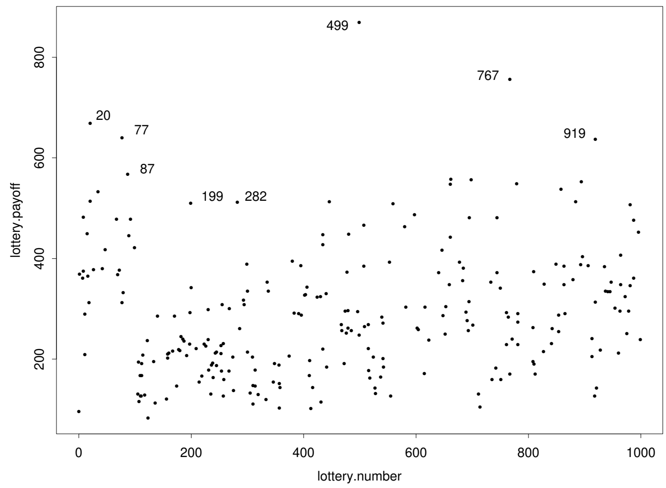


Figure 2.5: Outliers on the scatterplot are labelled with the actual winning number.

As a little help it is to say that the lottery has a mode of betting, called 'combination bets' where players win if the digits in their number appear in any order (Ticket 123 would win on 321, 231). The pattern in the numbers is that most of the numbers with high payoffs have duplicate digits. This results from the fact that payoffs for the numbers with duplicate digits are not shared with combination betters, and thus are higher.

Another method to look out for 'outliers' is to make a boxplot of the data. We noticed before that the payoffs seem to depend on the first digit of the winning number. So it would be interesting to draw boxes for the ten subsets of payoffs in a single plot to study this phenomenon graphically. Rather than extracting each set separately, we use the S-Plus function `split` to create a list, where each element of the list gives all of the payoffs that correspond to a particular first digit of the winning number. The `boxplot` function will draw a box for each element in the list.

```
> digit<-trunc(lottery.number/100)    # the first digits
> boxplot(split(lottery.payoff, digit))    # boxplot of data split by groups
> title(xlab="First Digit of Winning Number", ylab="Payoff")
> abline(h=500)    # horizontal line at 500
```

The box in a boxplot contains the middle half of the data; the whiskers extending from the box reach to the most extreme non-outlier; outlying points are plotted individually. Notice in Figure 2.6 the high payoffs for the first box. The graphic shows us as well that it is rare for a payoff to exceed $500. So, place your bet if you enjoy gambling. Do not expect to win.

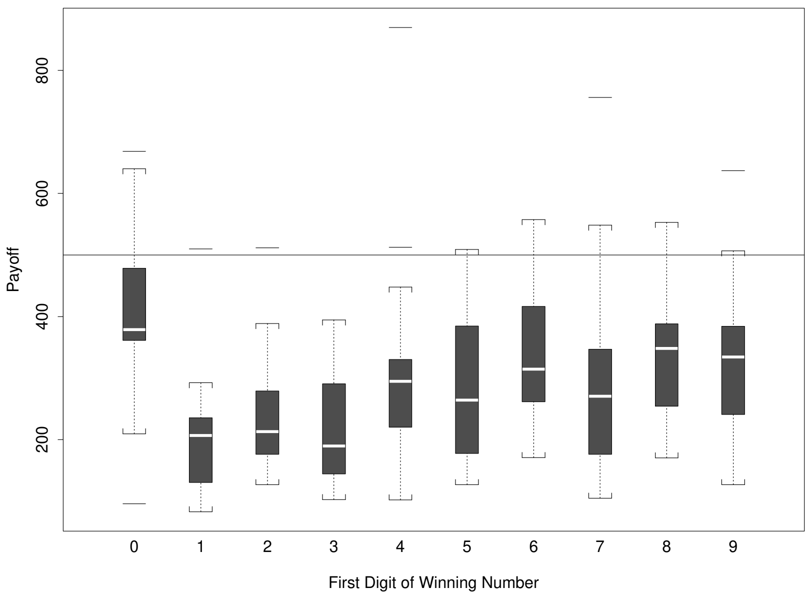


Figure 2.6: Boxplot of payoffs grouped by leading digit of the winning number.

Chapter 3

Data Manipulation

3.1 Basic Data Manipulation

The basic data objects in S-Plus are vectors, arrays, lists and data frames. In this section, each of them will be described in more detail.

3.1.1 Numbers and Vectors

S-Plus operates on named data structures. The simplest such structure is the vector, which is a single entity consisting of an ordered collection of numbers. For example,

```
> x.vector <- c(5.1, 6.3, 7.8, 9.3, 10.5)
```

is a vector consisting of five numbers, namely 5.1, 6.3, 7.8, 9.3 and 10.5. The previous S-Plus command is an assignment statement (`<-`) using the function `c`, which combines the five values into a vector. Note that the assignments can also be made in the other direction, using the obvious change in the assignment operator. So the same assignment could be made using

```
> c(5.1, 6.3, 7.8, 9.3, 10.5) -> x.vector
```

If an expression is used as a complete command, the value is printed and lost. So if we were to use the command

```
> 1/x.vector
```

the reciprocals of the five values would be printed and, of course, the value of `x.vector` would be unchanged,

```
> x.vector
[1]  5.1  6.3  7.8  9.3 10.5
```

Vectors can be used in arithmetic expressions, in which case the operations are performed element-by-element. The following elementary arithmetic operations work on each element of `x.vector`:

```
> x.vector - 1
[1] 4.1 5.3 6.8 8.3 9.5
> 2*(x.vector - 1)
[1]  8.2 10.6 13.6 16.6 19.0
> x.vector^2
[1]  26.01  39.69  60.84  86.49 110.25
```

where '^' stands for raising to a power. Any time you use an operator with a vector as one argument and a number as the other argument, the operation is performed on each component of the vector. Let us construct a second vector,

```
> y.vector <- c(1.3, 3.4, 8.1, 3.6, 5.7)
```

which has the same length as `x.vector`. Vectors occurring in the same expression need not all be of the same length. If they are not, the value of the expression is a vector with the same length as the longest vector which occurs in the expression. Shorter vectors in the expression are recycled as often as they need to be until they match the length of the longest vector; for this the length of the vectors have to be multiples. In particular a constant is simply repeated. So with the above assignments the command

```
> new.vector <- 4*x.vector - y.vector + 2
```

generates a new vector `new.vector` of length 5 constructed by adding together, element-by-element, `4*x.vector`, `- y.vector`, and 2 repeated five times. In addition all of the common arithmetic functions are available, e.g. `log`, `log10`, `exp`, `sin`, `cos`, `tan`, `sqrt`, and so on, all have their usual meaning. `max` and `min` select the largest and smallest elements of an vector respectively. `range` is a function whose value is a vector of length two, namely the vector containing the minimum and the maximum of all of the elements, e.g.

```
> range(x.vector)
[1]  5.1 10.5
> c(min(x.vector), max(x.vector))
[1]  5.1 10.5
```

The element-by-element maximum and minimum of two or more vectors are given by `pmax` and `pmin`. The function `length` gives the number of elements,

```
> length(x.vector)
[1] 5
```

Hence, valid indices for the vector `x.vector` are in the range `1:length(x.vector)`. Elements of a vector may be extracted by specifying the element in square brackets, '`[]`'. For example, `x.vector[3]` is the third component of `x.vector`, and

```
> x.vector[1:3]
```

selects the first three elements of `x.vector`, assuming `length(x.vector)` $\geq$ `3`. The same is obtained by `x.vector[c(1,2,3)]`. Of particular interest are the following two functions: `sum` gives the total of the elements and `prod` their product,

```
> sum(x.vector)
[1] 39
> prod(x.vector)
[1] 24472.46
```

So far, we have only seen vectors consisting of numbers, but a vector can also consist of an array of logical values:

```
> c(T, F, T, F, T)
```

where `T` stands for `TRUE` and `F` stands for `FALSE`. Or you can have a set of character strings:

```
> c("DMA", "EPFL")
```

S-Plus has a number of facilities for generating commonly used sequences of numbers. For example `1:10` is the vector

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
```

The colon operator has highest priority within an expression, so, for example `2*1:10` is the vector `c(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)`. To construct a backwards sequence, you can use `10:1`.

The function `seq` is a more general facility for generating sequences. From its help file

```
> ?seq
```

we see that it has five arguments. The first two arguments, if given, specify the beginning and end of the sequence, and if these are the only two arguments given the result is the same as the colon operator, i.e. `1:10` is the same vector as

```
> seq(1, 10)
```

So are `seq(from=1, to=10)` and `seq(to=10, from=1)`. The next two parameters to `seq` specify a step size and a length for the sequence respectively. If neither of these is given, the default is assumed. To generate a sequence from 0 to 1 with spacing of 0.1 between successive values, type

```
> seq(0, 1, 0.1)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Or you can specify the beginning, the increment, and the length with either the `length` argument or the `along` argument:

```
> seq(0, by=0.1, length=11)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(0, by=0.1, along=1:11)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

A related function is `rep` which repeats a value by specifying either a `times` argument or a `length` argument. If `times` is specified, the value is repeated the number of times specified (the value may be a vector):

```
> rep(1, 5)
[1] 1 1 1 1 1
> rep(c(1, 2, 3), 2)
[1] 1 2 3 1 2 3
```

If `times` is a vector with the same length as the vector of values being repeated, each value is repeated the corresponding number of times.

```
> rep(c("DMA", "EPFL"), c(4,2))
[1] "DMA"  "DMA"  "DMA"  "DMA"  "EPFL" "EPFL"
```

Moreover, the function `paste` can be used to create some special vectors, e.g.

```
> paste("no", 1:3, sep=".")
[1] "no.1" "no.2" "no.3"
```

The `order` function produces the vector of indices that will sort a vector. That is, `x.vector[order(x.vector)]` is the same as `sort(x.vector)`. The function `order` is useful when you want to sort several vectors in parallel:

```
> x.order <- order(x.vector)
> x.order
[1] 1 2 3 4 5
> x.sorted <- x.order[x.order]
> y.sorted <- y.vector[x.order]
> y.sorted
[1] 1.3 3.4 8.1 3.6 5.7
```

`x.sorted` is actually sorted; `y.sorted` probably is not sorted but it is in the order that made `x.vector` sorted. To reverse the order of a vector or list, one can use the `rev` function. So you might have

```
> x.vector[rev(order(x.vector))]
[1] 10.5  9.3  7.8  6.3  5.1
```

Additional functions for manipulating vectors are `unique` and `duplicated`, which returns the values of the input without any repetition, or a logical vector denoting which elements duplicate previous elements. For example, to get the sorted list of

```
> tosort.vector <- rep(c("DMA", "EPFL"), c(4,2))
```

with no duplicates, type

```
> sort(unique(tosort.vector))
[1] "DMA"  "EPFL"
```

3.1.2 Matrices and Arrays

Matrices

An important data object type in S-Plus is the two-way array, or matrix object. Matrices and their higher-dimensional analogues, arrays, are related to vectors, but have an extra structure imposed on them. Matrices are used to arrange values by rows and columns in a rectangular table. For data analysis, different variables are usually represented by different columns, and different cases or subjects are represented by different rows. Thus matrices are convenient for grouping together observations that have been measured on the same set of subjects and variables. Matrices differ from vectors by having a 'dimension', that is, the number of rows and columns. For example, the vector `x.vector` has no 'dimension':

```
> dim(x.vector)
NULL
```

To turn `x.vector` into a matrix one has simply to specify its dimension,

```
> dim(x.vector) <- c(1,5)
> x.vector
     [,1] [,2] [,3] [,4] [,5]
[1,]  5.1  6.3  7.8  9.3 10.5
```

In addition to `dim` the function `as.matrix` can be used as well:

```
> x.vector <- c(5.1, 6.3, 7.8, 9.3, 10.5)
> dim(x.vector)
NULL
> x.matrix <- as.matrix(x.vector)
> dim(x.matrix)
[1] 5 1
```

To create a matrix, one can use the `matrix` function. The `matrix` function takes as arguments a vector and two numbers which specify the number of rows, `nrow` and columns, `ncol`. For example:

```
> mymatrix <- matrix(1:15, nrow=3, ncol=5, byrow=T)
> mymatrix
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
```

The argument `byrow=T` assumes the data values to be the first row, then the second row, and finally the third row. More often, you need to combine several vectors or matrices into a single matrix. To combine vectors (and matrices) into matrices, one can use the functions `cbind` and `rbind`. The `cbind` function combines vectors column by column, and `rbind` combines vectors row by row.

```
> rbind(1:5, 6:10, 11:15)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15

> cbind(1:5, 6:10, 11:15)
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

When vectors of different lengths are combined using `cbind` or `rbind`, the shorter ones are replicated cyclically so that the matrix is 'filled in'. If matrices are combined, they must have matching numbers of rows when using `cbind` and matching numbers of columns when using `rbind`. Otherwise, S-PLUS prints an error message and the objects are not combined. For example, using `rbind`:

```
> rbind(mymatrix, 16:20)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
```

To name the rows or/and columns of a matrix, one can use the function `dimnames`. For example, suppose that the columns of `mymatrix` contain the colors, say, of different flowers. In typing

```
> dimnames(mymatrix)
NULL
```

one sees that this information is not included in the matrix, so one may like to pass a list of names to `mymatrix`,

```
> dimnames(mymatrix) <- list(NULL, c("red", "brown", "purple",
+ "silver", "blue"))
> mymatrix
     red brown purple silver blue
[1,]   1     2      3      4    5
[2,]   6     7      8      9   10
[3,]  11    12     13     14   15
```

An additional example on the use of the function `order` is to sort the columns of a matrix relative to one of the rows, say in the reversed order:

```
> mymatrix[, rev(order(mymatrix[1,]))]
     blue silver purple brown red
[1,]    5      4      3     2   1
[2,]   10      9      8     7   6
[3,]   15     14     13    12  11
```

Table 3.1: Access elements of the matrix `A.mat`.

Command	Description
`A.mat[i, j]`	Access the element in the `i`th row `j`th column
`A.mat[i, ]`	Access `i`th row
`A.mat[, j]`	Access `j`th column
`A.mat[-i, ]`	A.mat without `i`th row
`A.mat[, -j]`	A.mat without `j`th column
`A.mat[vec, ]`	Access the rows defined by the vector `vec`

Different ways of how to access matrix elements are given in Table 3.1. For matrix multiplication the operator '`%*%`' is used. For example

```
> matrix.1 <- matrix(1:4, 2)
> matrix.1
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> matrix.2 <- matrix(5:8, 2)
> matrix.2
[1,]    5    7
[2,]    6    8
```

are square matrices of the same size. So,

```
> matrix.1 * matrix.2
```

is the matrix of element by element product and

```
> matrix.1 %*% matrix.2
```

is the matrix product. Moreover, the function `t` produces the matrix transpose, `diag` either creates a diagonal matrix or extracts the diagonal elements of a matrix, `eigen` returns the eigenvalues and eigenvectors of a square matrix, `chol` returns the Choleski decomposition of a symmetric matrix, `qr` computes a QR matrix decomposition, and `solve` performs matrix inversion if given a single argument, or solves systems of linear equations if given two arguments.

Arrays

To create an array in S-Plus, one has to use the `array` function. The `array` function is analogous to `matrix`. It takes data and the appropriate dimensions as arguments, then produces the array. If no data is supplied, the array is filled with `NA`s. In the following example we create a 2 by 4 by 3 array,

```
> myvect <- c(1:8, 11:18, 111:118)
> newarray <- array(myvect, dim = c(2,4,3))
> newarray

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

, , 2
     [,1] [,2] [,3] [,4]
[1,]   11   13   15   17
[2,]   12   14   16   18

, , 3
     [,1] [,2] [,3] [,4]
[1,]  111  113  115  117
[2,]  112  114  116  118
```

The first dimension (the rows) is incremented first. This is equivalent to placing the values column by column. The second dimension (the columns) is incremented second. The third dimension is incremented by filling a matrix for each level of the third dimension. For creating arrays from existing vectors, the `dim` function works for arrays in the same way it works for matrices. For example, if the data above were stored in the vector `myvect`, you could create the above array by defining the 'dimension' with the vector `c(2,4,3)`:

```
> myvect
 [1]   1   2  3  4  5   6   7   8  11  12  13
[12]  14  15 16 17 18 111 112 113 114 115 116
[23] 117 118
> dim(myvect) <- c(2,4,3)
```

To name each level of each dimension, one can use the `dimnames` function in the same way as it is done for matrices.

3.1.3 Lists and Data Frames

Lists

An S-PLUS list is an object consisting of an ordered collection of objects known as its components. There is no particular need for the components to be of the same mode or type, and, for example, a list could consist of a numeric vector, a logical value, a matrix, a character array, a function, and so on. Components are always numbered and may always be referred to as such.

New lists may be formed from existing objects by the function `list`. Consider for example the following list

```
> mylist <- list("First element"=1,  "Second"=1:2, "Third"=1:4,
+ "Fourth"="This is a simple list.")
> mylist
$"First element":
[1] 1

$Second:
[1] 1 2

$Third:
[1] 1 2 3 4

$Fourth:
[1] "This is a simple list."
```

Components are always numbered and may always be referred to as such. The function `length(mylist)` gives the number of (top level) components it has: 4; specified as `mylist[[1]]`, `mylist[[2]]` and so on. And the command `mylist[1:3]` (!) gives the first three components. As we named the components of the list, the component may be referred to either by giving the component name as a character string in place of the number in double square brackets, or, more conveniently, by giving an expression of the form

```
> mylist$Second
```

when we want to list the component `Second` of `mylist`. This is a very useful convention as it makes it easier to get the right component if you forget the number, and is strongly advised. You can find out the names of the components by the function `names`,

```
> names(mylist)
```

This generates much less output that printing the object, which will achieve the same purpose. The names of components may be abbreviated down to the minimum number of letters needed to identify them uniquely. Most of the datasets are in fact lists (or can be treated as lists), so we could refer to the component `Second` of `mylist` as `mylist$S`. Moreover, note that lists can be attached, by means of the function `attach`, as well as directories, and this allows their components to be accessed as if they were stand-alone entities.

Data Frames

Data frames can be thought of as closely coupled lists of data vectors of the same length. Unlike matrices, the data vectors can be of different types, including character data. The main benefit of a data frame is that it allows you to mix data of different types into a single object in preparation for analysis and modeling. The idea of a data frame is to group data by variables (columns) regardless of their type. Then all the observations on a particular set of variables can be grouped into a single data frame. This is particularly useful in data analysis where it is typical to have a 'character' variable labeling each observation, one or more 'numeric' variables of observations, and one or more categorical variables for grouping observations. Consider the built-in data frame `catalyst`, which represents all possible combinations of two temperatures (`160`, `180`), two concentrations (`20`, `40`), and two catalysts (`A`, `B`). The fourth column represents the response variable `Yield`.

```
> catalyst
  Temp Conc Cat Yield
1  160   20   A    60
2  180   20   A    72
3  160   40   A    54
4  180   40   A    68
5  160   20   B    52
6  180   20   B    83
7  160   40   B    45
8  180   40   B    80
```

One remarks that this data frame has both row and column labels. The columns can be treated as components of a list:

```
> catalyst$Temp
[1] 160 180 160 180 160 180 160 180
```

and the structure can be treated as a two-dimensional array:

```
> catalyst[2, 4]
[1] 72
> catalyst[1, "Yield"]
[1] 60
> catalyst[1,]
  Temp Conc Cat Yield
1  160   20   A    60
```

Note how the row label is carried along. Data frames can be attached just as lists can, and this allows their columns to be accessed as if they were named vectors.

```
> attach(catalyst)
> Cat
[1] A A A A B B B B
```

A data frame can be created from vectors and matrices by the `data.frame` function. For example:

```
> treeframe <- data.frame(diameter=c(10,15,20,25),
                          height=c(1.5,1.6,1.2,2.1))
> treeframe
  diameter height
1       10    1.5
2       15    1.6
3       20    1.2
4       25    2.1
```

Finally, note that you can create data frames in several ways:

- `read.table` reads in data from an external file (see Section 3.2.1);
- `data.frame` binds together S-PLUS objects of various kinds; and
- `as.data.frame` coerces objects of a particular type to objects of class `data.frame`.

You can also combine existing data frames in several ways, using the `cbind`, `rbind`, and `merge` functions.

3.1.4 Logical and Missing Values

Logical Values

Comparisons and logical operations are frequently convenient for extracting subsets of data, and conditionals using logical comparisons play an important role in flow of control in functions. Table 3.2 lists the S-PLUS logical and comparison operators. The elements of a logical vector have just two possible values, represented formally as `F` (for 'false') and `T` (for 'true'). Note that `TRUE` and `FALSE` are also valid representations. Logical vectors are generated by conditions. For example

```
> z.vector <- 1:10
> z.vector > 5
[1] F F F F F T T T T T
```

Table 3.2: Logical and comparison operators in S-Plus.

Operator	Explanation
`==`	equal to
`!=`	not equal to
`>`	greater than
`<`	less than
`>=`	greater than or equal to
`<=`	less than or equal to
`&`	vectorized 'And'
`|`	vectorized 'Or'
`&&`	control 'And'
`||`	control 'Or'
`!`	not

produces a vector of the same length as `z.vector` with values `F` corresponding to elements of `z.vector` where the condition is not met and `T` where it is.

Subsets of a vector (or any expression that evaluates to a vector) may be selected by appending to the name of the vector an index vector in square brackets. Such index vectors can be any logical vector, which must be of the same length as the vector from which elements are to be selected. Values corresponding to `T` in the index vector are selected and those corresponding to `F` omitted. For example

```
> index <- (z.vector > 5)
> newz.vector <- z.vector[index]
> newz.vector
[1]  6  7  8  9 10
```

creates a vector `newz.vector` satisfying the condition `(z.vector > 5)`. Additional examples of the S-Plus logical and comparison operators listed in Table 3.2 follow:

```
> (z.vector == 5)
[1] F F F F T F F F F F
> (z.vector > 5)&(z.vector < 8)
[1] F F F F F T T F F F
> (z.vector < 5)|(z.vector > 8)
[1] T T T T F F F F T T
```

Moreover, the functions `all` and `any` can also be useful:

```
> all(z.vector<5)
[1] F
```

```
> any(z.vector<5)
[1] T
```

Missing Values

In some cases the components of a vector may not be completely known. When an element or value is 'not available' or a 'missing value' in the statistical sense, a place within a vector may be reserved for it by assigning it the special value `NA`. In general any operation on an `NA` becomes an `NA`. The motivation for this rule is simply that if the specification of an operation is incomplete, the result cannot be known and hence is not available. For example, let `na.vector` be the vector `c(1, 2, NA)`. The `is.na(na.vector)` gives a logical vector of the same size as `na.vector` with value `T` if and only if the corresponding element in `na.vector` is `NA`.

```
> na.vector <- c(1, 2, NA)
> is.na(na.vector)
[1] F F T
```

For example,

```
> no.na.vector <- na.vector[!is.na(na.vector)]
```

creates (or re-creates) an object `no.na.vector` which will contain the non-missing values of `na.vector`, in the same order. Note that as `na.vector` has missing values, `no.na.vector` is shorter than `na.vector`.

Infinite values are represented as `INF` and `is.infinite` is `TRUE` for values that are either plus or minus infinity.

3.2 Importing and Exporting Data

The actual version 5.x of S-Plus for UNIX makes it easy to read data from virtually any source. Its comprehensive import/export capabilities (like the functions `importData`, `exportData`, `openData`, `readNextDataRows`) free you from spending time moving data from source to source, allowing you to focus on your analysis. You can directly import and export data in the following formats and databases: Oracle, Sybase, Informix, SAS, SPSS, Excel, ODBC databases, Text (ASCII) files, Quattro Pro, Lotus 1-2-3, dBase, Formatted ASCII, Systat, Stata, Gauss, and MatLab.

But, unfortunately, this functions does not exist for S-Plus version 3.4 for UNIX. I refer readers using S-Plus version 5.x for UNIX to the 'S-Plus 5 for UNIX User's Guide' and to Chapter 10 of the present manuscript. Note that while `importData` is the recommended method for reading data files into S-PLUS, there are several other functions that you can use to read ASCII data into S-PLUS. These functions are commonly used by other functions in S-PLUS and are already part of S-PLUS 3.4, so it is a good idea even for S-PLUS 5.x users to familiarize with them. In S-PLUS version 3.4 for UNIX, the principal tools for importing/exporting data are the following functions: `source`, `scan`, `read.table`, `dump`, `restore`, `cat`, `write`, `write.table` and `sink`.

3.2.1 Notes on Importing

Commands can be stored in an external file and executed using the `source` function. For example, if a series of S-PLUS instructions were stored on a file `commands.q` they would be caused to run by the command:

```
> source("commands.q")
```

The `read.table` function reads in a file and creates a data frame. Data frames in S-PLUS were designed to resemble tables. They must have a rectangular arrangement of values and typically have row and column labels. Data frames arise frequently in designed experiments and other situations. If you have a text file with data arranged in the form of a table, you can read it into S-PLUS using the `read.table` function. For example, consider the data file `auto.dat`:

```
        Model    Price   Country Reliab Mileage Type
 AcuraIntegra4   11950   Japan   5      NA      Small
 Audi1005        26900   Germany NA     NA      Medium
 BMW325i6        24650   Germany 94     NA      Compact
 ChevLumina4     12140   USA     NA     NA      Medium
 FordFestiva4     6319   Korea   4      37      Small
 Mazda929V6      23300   Japan   5      21      Medium
 MazdaMX-5Miata  13800   Japan   NA     NA      Sporty
 Nissan300ZXV6   27900   Japan   NA     NA      Sporty
 OldsCalais4      9995   USA     2      23      Compact
 ToyotaCressida6 21498   Japan   3      23      Medium
```

All fields are separated by spaces and the first line is a header line. To create a data frame from this data file, use `read.table` as follows:

```
> auto <- read.table("auto.dat", header=T)
> auto
                Price Country Reliab Mileage    Type
  AcuraIntegra4 11950   Japan      5      NA   Small
       Audi1005 26900 Germany     NA      NA  Medium
       BMW325i6 24650 Germany     94      NA Compact
```

```
    ChevLumina4 12140     USA      NA       NA  Medium
   FordFestiva4  6319   Korea       4       37   Small
     Mazda929V6 23300   Japan       5       21  Medium
 MazdaMX-5Miata 13800   Japan      NA       NA  Sporty
  Nissan300ZXV6 27900   Japan      NA       NA  Sporty
    OldsCalais4  9995     USA       2       23 Compact
ToyotaCressida6 21498   Japan       3       23  Medium
```

The `scan` function, which can read from either standard input or from a file, is commonly used to read data from keyboard input. By default, `scan` expects numeric data separated by white space, although there are options with `scan` that let you specify the type of data being read and the separator. When using `scan` to read data files, it is helpful to think of each line of the data file as a record, or case, with individual observations as fields. For example, the following expression creates a matrix named `test.file` from a data file specified by the user:

```
> test.file <- matrix(scan("filename"), ncol = 6, byrow = T)
```

Here the data file is assumed to have 6 columns of numeric data; the matrix contains a number of observations for each of these ten variables. To read in a file of character data, use `scan` with the `what` argument:

```
> test.file <- matrix(scan("filename", what = ""), ncol = 6, byrow = T)
```

The `what` argument to `scan` can also be used to read in data files of mixed type, for example, a file containing both numeric and character data, as in the following sample file, `table.dat`:

```
 Tom  93 37
 Joe  47 42
 Dave 18 43
```

In this case, you provide a list as the value for `what`, with each list component corresponding to a particular field:

```
> test.file <- scan("table.dat", what=list("",0,0))
> test.file
[[1]]:
[1] "Tom" "Joe" "Dave"

[[2]]:
[1] 93 47 18

[[3]]:
[1] 37 42 43
```

S-PLUS creates a list with separate components for each field specified in the `what` list. You can turn this into a matrix, with the subject names as column names, as follows:

```
> test.matrix <- rbind(test.file[[2]], test.file[[3]])
> dimnames(test.matrix) <- list(NULL, test.file[[1]])
> test.matrix
     Tom Joe Dave
[1,]  93  47   18
[2,]  37  42   43
```

You can scan files containing multiple line records by using the argument `multi.line=T`.

3.2.2 Notes on Exporting

When you want to export data to share with another S-PLUS user, use the to S-PLUS `data.dump` function:

```
> data.dump("test")
```

By default, the data object `test` is exported to the file `dumpdata` in your S-PLUS startup directory. You can specify a different output file with the `file` argument to `data.dump`:

```
> data.dump("test", file="test.dmp")
```

To bring back into S-PLUS dumped objects, i.e. objects created by `data.dump`, you have to use the `restore` function.

The inverse operation to the `scan` function is provided by the `cat` and `write` functions. Similarly, the inverse operation to `read.table` is provided by `write.table`. The result of either `write` or `cat` is just an ASCII file with data in it. There is no S-PLUS structure written in. Of the two commands, `write` has an argument for specifying the number of columns and thus is more useful for retaining the format of a matrix. By default, `write` writes matrices column by column, five values per line. If you want the matrix represented in the ASCII file in the same form it is represented in S-PLUS, transform the matrix first with the `t` function and specify the number of columns in your original matrix:

```
> mymatrix
     [,1] [,2] [,3] [,4] [,5]
[1,]  5.1  6.3  7.8  9.3 10.5
[2,]  5.1  6.3  7.8  9.3 10.5
[3,]  5.1  6.3  7.8  9.3 10.5

> write(t(mymatrix), "mymatrix", ncol=5)
```

You can view the resulting file with a text editor or pager; it contains the following three lines:

```
 5.1 6.3 7.8 9.3 10.5
 5.1 6.3 7.8 9.3 10.5
 5.1 6.3 7.8 9.3 10.5
```

The `cat` function is a general-purpose writing tool in S-PLUS, used for writing to the screen as well as writing to files. To use `cat` to write to a file, simply specify a file name with the file argument:

```
> test.vector <- 1:1000
> cat(test.vector, file="test.vector", fill=T)
```

The argument `fill=T` limits line length in the output file to the width specified in your options object. The files written by `cat` and `write` do not contain S-Plus structure information; to read them back into S-Plus you must reconstruct this information! The `write.table` function can be used to export a data frame, say the built-in data frame `fuel.frame`, into an ASCII text file:

```
> write.table(fuel.frame, "fuel.txt")
```

Finally, the S-Plus command

```
> sink("sink.txt")
```

will divert all subsequent output from the terminal to an external file `sink.txt`. The command `sink()` will restore output to the terminal.

3.3 Exercises

Exercise 3.1. For the numbers

```
7.3   6.8   0.005   9   12   2.4   18.9   0.9
```

(a) Find their mean.

(b) Substract the mean from each number.

(c) Calculate the square roots of the numbers.

(d) Print those numbers which are larger than their square roots.

Exercise 3.2. Put `n <- 10` and compare the sequences `1:n-1` and `1:(n-1)`.

Exercise 3.3. What is the value of the expression `1:6 * 1:2`. Explain how this particular result is computed.

Exercise 3.4. Generate a sequence from 1 to 10 with spacing 0.5 in three different ways using the `seq` function.

Exercise 3.5. Use the `seq` and `rep` functions to produce a vector containing:

(a) the values: $1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4$;

(b) the values: $4, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1, 1, 1$;

(c) the values: $1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5$; and

(d) the values: $1, 1, 3, 3, 5, 5, 7, 7, 9, 9$.

Exercise 3.6. Use the function `paste` to create the following character vector:

`"X1" "Y2" "X3" ... "X9" "Y10"`

Exercise 3.7. Suppose you have an object, `X`, containing some numbers ranging from 0 to 100 (maybe repeated). Now you would like to create another object `Y` from `X` in such a way that all numbers which are less than 10 or greater than 90 must be excluded. How can do you that?

Exercise 3.8. Suppose you have an object `A` containing the following characters:

`"M","M",...,"M","A","A",...,"A","N","N",...,"N","Y","Y",...,"Y"`

Create an object `B` from `A` containing only the four characters `"M","A","N","Y"`

Exercise 3.9. Consider the two matrices `M1`

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

and `M2`

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

(a) Create `M1` and `M2` in S-Plus.

(b) Multiply `M1` by itself element by element, and by matrix multiplication.

(c) Multiply `M1` by `M2`, and multiply the transpose of `M2` by `M1`.

Exercise 3.10. Consider the following matrix `Exo.mat`:

```
      [,1] [,2] [,3]
[1,]    12   10    3
[2,]    15    2    2
[3,]     6    9    1
```

(a) Create `Exo.mat` in S-Plus.

(b) Extract the diagonal elements of `Exo.mat`, and create a square matrix with this elements.

(c) Invert the matrix `Exo.mat`, and calculate the eigenvalues of `Exo.mat`.

(d) Let `b` the vector `c(9,5,14)`, and solve the following system of linear equations: `Exo.mat` $x =$ `b`. Verify the solution.

Exercise 3.11. Create the following matrices as simple as possible:

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

```
      [,1] [,2] [,3] [,4] [,5]
[1,]     0   -1   -2   -3   -4
[2,]     1    0   -1   -2   -3
[3,]     2    1    0   -1   -2
[4,]     3    2    1    0   -1
[5,]     4    3    2    1    0
```

Exercise 3.12. Solve the following system of linear equations:

```
2x +  3y -   z = 4
5x - 10y + 2z = 0
 x +   y - 4z = 5
```

Exercise 3.13. Consider the built-in data frame `catalyst`:

```
> catalyst
  Temp Conc Cat Yield
1  160   20   A    60
2  180   20   A    72
```

```
3  160  40  A  54
4  180  40  A  68
5  160  20  B  52
6  180  20  B  83
7  160  40  B  45
8  180  40  B  80
```

(a) Reconstruct this data frame by means of functions like `rep`, `data.frame`, etc.

(b) Export the created data frame by means of `write.table` to a file named `catalyst.dat`, and view the resulting file with a text editor (e.g. emacs, jot, nedit) or a pager (e.g. more, cat, less).

(c) Import the file `catalyst.dat` into S-PLUS using `read.table` under the name `catalyst.two`

(d) Extract the entries for temperature `160`, and those for concentration `20`.

Exercise 3.14. Consider the built-in data frame `kyphosis`, which represents data on 81 children who have had corrective spinal surgery.

(a) Store the entries for children, which had a postoperative deformity (`Kyphosis`), in `child.yes`.

(b) Store the entries for children, which had no postoperative deformity (`Kyphosis`), in `child.no`.

(c) Extract the entries for one month (`age`) old children.

Exercise 3.15 (LaTeX data). To get this data, which is not built-in, you must attach the library `s+intro` by typing

```
> library("s+intro")
```

This library contains the data frame `LaTeX.dat`, which gives information on the use of LaTeX for writing PhD thesis at the EPFL for the years 1993-1998.

```
> LaTeX.dat
    Year Dept LaTeX
  1 1997   DP     F
  2 1993   DP     F
  3 1993  DGM     F
  4 1994  DGC     F
 ...
563 1993   DP     T
564 1993   DP     F
565 1996  DGC     F
566 1997  DGC     F
```

The data frame has 566 rows and three columns: the `Year` of the PhD thesis, the department (`Dept`) in which the thesis was made, and the third column represents the response variable `LaTeX`, i.e. if the thesis has been written in LaTeX or not.

(a) From `LaTeX.dat` create a data frame, or a matrix, containing only the entries for 1997, and denore it by `Y1997.dat`.

(b) From `Y1997.dat` create a data frame, or a matrix, containing only the entries for a particular department.

(c) For this subset of the data calculate the number of thesis for which LaTeX was used, and compute its percentage.

(d) From `LaTeX.dat` create a data frame, or a matrix, containing only the `LaTeX=="T"` entries, and calculate for each department the number of thesis using LaTeX.

Exercise 3.16. Consider the data from the New Jersey Pick-it Lottery, which are in S-Plus under the names `lottery.number` and `lottery.payoff`.

(a) Find which winning numbers had payoffs of more than $500.

(b) Find the 10 smallest payoffs and the corresponding 'unlucky' numbers. Hint: the function `order` is helpful.

Chapter 4

Graphics

Note that S-Plus 4.0 for Windows introduced a completely separate style of graphics based on menus and toolbars with simple command-line equivalents. The style of the interface is designed for intuitive exploration by experienced Windows users; most of the options are set from dialog boxes brought up by selecting and double-clicking or right-clicking elements on the plot. There are separate dialog boxes for different plot elements. This graphical system works in a new graphical device called a *Graph Sheet*. As this guide is intended for S-Plus for UNIX users, this device will not be treated in this guide.

4.1 Introduction

Graphics are an integral part of S-Plus. The topic deserves a book of its own, and is only skimmed in this chapter. Two principles that S-Plus graphics follow is that they are device independent, and that each graph can be built up with numerous commands. Device independence means that the same commands are used to create a graph on a PostScript printer as on a monitor running Motif — the only difference is what type of graphics device the user has set to listen for graphics commands. The ability to add to plots means that complex graphics can be created easily. A graphics device is a S-Plus function that arranges for graphics to be rendered. For example, the `postscript` function makes it so that a file of PostScript commands will result when a graphics command is given. Graphics devices are put into one of three categories. *Hard-copy devices* are for creating a physical picture; the most common is the `postscript` device. *Window devices* produce graphics when you are running a window system on the network where S-Plus is running; the S-Plus `motif` device is an example. Finally there are *terminal devices* that are used when S-Plus is run remotely. To inform you of graphics devices available to you, type

```
> help(Devices)
```

A graphics device has a state that is queried and modified by the `par` function. The state is described by a reasonably large number of graphics parameters, which will be described later on. Graphics functions that say what to draw, as opposed to graphics devices, are divided into *high-level* and *low-level* functions. High-level functions, like `plot` and `barplot`, create an entire new figure. Low-level functions (`points` and `lines`, for example) merely add to the existing figure. Some functions have an `add` argument so that they can function in either capacity.

Note that in addition to the graphics features described in this chapter, S-Plus includes the 'Trellis Graphics' library. You will find information at:

`http://www.research.att.com/~rab/trellis/`

During this chapter, a graphics device is assumed to be running; the S-Plus `motif` device is an example,

```
> motif()
```

4.2 High-level Plotting Commands

High-level plotting functions are designed to generate a complete plot of the data passed as arguments to the function. Appropriate, axes, labels and titles are automatically generated (unless you request otherwise). Note that high-level plotting commands always start a new plot, erasing the current plot if necessary.

The most frequently used S-Plus plotting function is `plot`. This is a generic function: the type of plot produced is dependent on the type or class of the first argument. When you call a plotting function, the S-Plus graphics window displays the requested plot:

```
> plot(car.miles)
```

This plot is shown in Figure 4.1. The argument `car.miles` is a S-Plus built-in vector data object. Since there is no other argument to `plot`, the data are plotted against their natural index or observation numbers, 1 through 120. Since you may be interested in your gas mileage, you may want to plot `car.miles` against `car.gals`. This is also easy to do:

```
> plot(car.gals, car.miles)
```

The result is shown in Figure 4.2.

You can use many high-level plotting functions besides `plot` to display graphical results in the S-Plus graphics window. These functions are listed in Table 4.1.

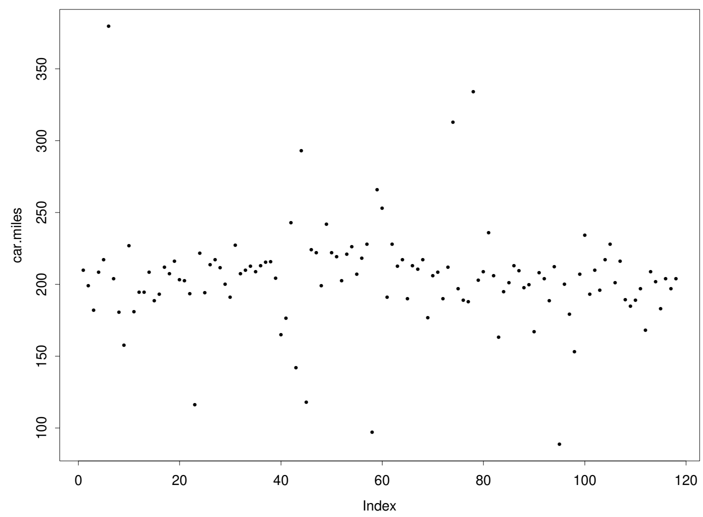


Figure 4.1: Default plot for the `car.miles` vector data object.

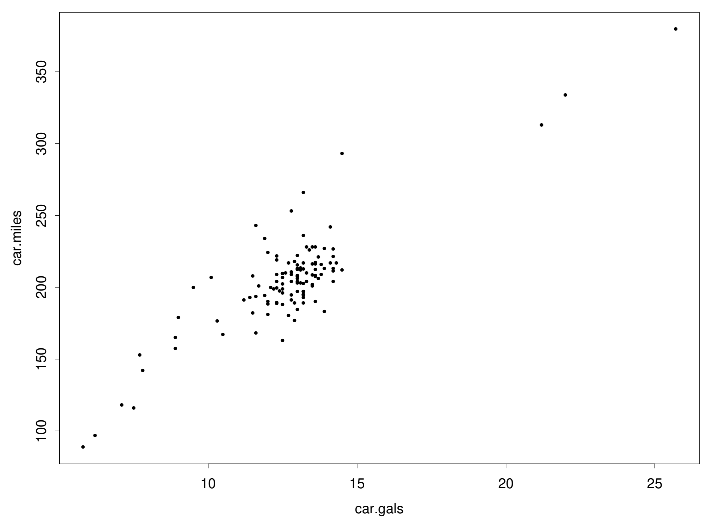


Figure 4.2: Scatterplot of `car.miles` against `car.gals`.

A funny example is given in Figure 4.3, where each observation of the `cereal.attitude` data, giving consumer attitudes towards breakfast cereals, is represented as a face. The following S-Plus command was used:

```
> faces(t(cereal.attitude), labels=dimnames(cereal.attitude)[[2]],
+ ncol=3)
```

Which cereal did the consumers like best? You may find the response in Chernoff (1973, 'The Use of Faces to Represent Points in k-Dimensional Space Graphically', *Journal of the American Statistical Association*, 68, 361-368).

Table 4.1: Common high-level plotting functions in S-Plus.

Name	Description
`barplot`, `hist`	Bar graph, histogram
`boxplot`	Boxplot
`brush`	Brush pair-wise scatter plots; spin 3-D axes
`contour`, `image`, `persp`	3-D plots
`coplot`	Conditioning plot
`dotchart`	Dotchart
`faces`, `stars`	Display multivariate data
`pairs`	Plot all pair-wise scatter plots
`pie`	Pie chart
`qqnorm`, `qqplot`	Normal and general QQ-plots
`scatter.smooth`	Scatter plot with a smooth curve
`tsplot`	Plot a time series
`usa`	Plot the boundary of the U.S.

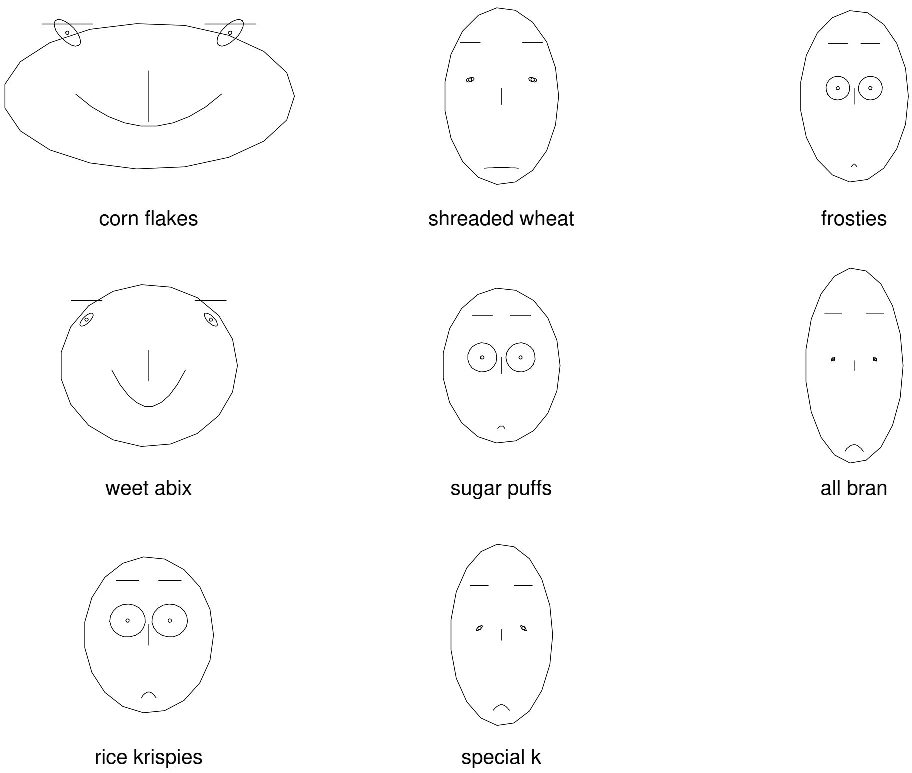


Figure 4.3: Plot of symbolic faces for the `cereal.attitude` data.

Note that S-PLUS provides very useful functions for representing multivariate data, which will be illustrated in Section 5.1. Moreover, there are a number of arguments which may be passed to high-level graphics functions. A selected list is given in Table 4.2, and Figure 4.4 illustrates different `type` arguments in order to control the type of plot.

Table 4.2: Selected list of arguments to high-level plotting functions.

Argument	**Description**
`add=T`	Forces the function to act as a low-level graphics function, superimposing the plot on the current plot (some functions only).
`axes=F`	Suppresses generation of axes — useful for adding your own custom axes with the `axis` function. The default, `axes=T`, means include axes.

`log="x"`
`log="y"`
`log="xy"`
	Causes the x, y or both axes to be logarithmic. This will work for many, but not all, types of plot.

`type="p"`	Plot individual points (the default)
`type="l"`	Plot lines
`type="b"`	Plot points connected by lines (both)
`type="o"`	Plot points overlaid by lines
`type="h"`	Plot vertical lines from points to the zero axis (high-density)
`type="s"` `type="S"`	Step-function plots. In the first form, the top of the vertical defines the point; in the second, the bottom.
`type="n"`	No plotting at all. However axes are still drawn (by default) and the coordinate system is set up according to the data. Ideal for creating plots with subsequent low-level graphics functions.
`xlab="string"` `ylab="string"`	Axis labels for the x and y axes. Use these arguments to change the default labels, usually the names of the objects used in the call to the high-level plotting function.
`main="string"`	Figure title, placed at the top of the plot in a large font.
`sub="string"`	Sub-title, placed just below the x-axis in a smaller font.

4.3 Low-level Plotting Commands

Sometimes the high-level plotting functions do not produce exactly the kind of plot you desire. In this case, low-level plotting commands can be used to add extra information to the current plot. Some of the more useful low-level plotting functions are listed in Table 4.3. Low-level plotting functions usually require some positioning information (e.g. x and y coordinates) to determine where to place the new plot elements. Coordinates are

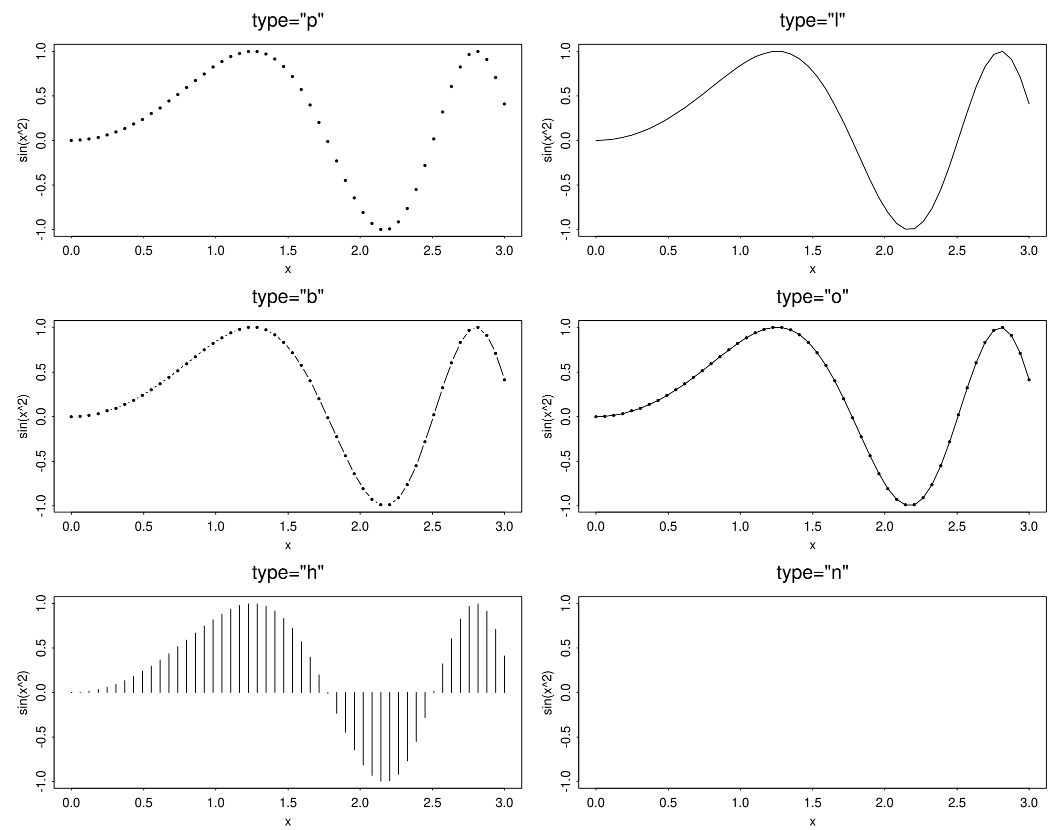


Figure 4.4: Different `type` arguments to high-level plotting functions.

given in terms of 'user coordinates' which are defined by the previous high-level graphics command and are chosen based on the supplied data.

Table 4.3: Common low-level plotting functions in S-Plus.

Name	**Description**
`abline`	Add line in intercept-slope form
`axis`	Add axis
`box`	Add a box around plot
`legend`	Add a legend to the plot
`lines`, `points`	Add lines or points to a plot
`mtext`, `text`	Add text in the margin or in the plot
`stamp`	Add date and time information to the plot
`symbols`	Add 3-D information to plot
`title`	Add title to the plot

4.4 Interactive Graphics Functions

S-Plus also provides functions which allow users to extract or add information to a plot using a mouse. The simplest of these is the `locator` function, which waits for the user to select locations on the current plot using the left mouse button. The `type` argument allows for plotting at the selected points and has the same effect as for high-level graphics commands; the default is no plotting. `locator` returns the locations of the points selected as a list with two components `x` and `y`. Note that `locator` is usually called with no arguments. It is particularly useful for interactively selecting positions for graphic elements such as legends or labels when it is difficult to calculate in advance where the graphic should be placed.

Moreover, the `identify` functions performs no plotting itself, but simply allows the user to move the mouse pointer and click the left mouse button near a point, i.e. the mouse is used to identify points on a graph.

4.5 Graphics Parameters

You can customize almost every aspect of the display using so-called graphics parameters. S-Plus maintains a large list of graphics parameters which control things such as line style, colors, figure arrangement and text justification among many others. Every graphics parameter has a name (such as `col`, which controls colors) and a value (a color number, for example). A separate list of graphics parameters is maintained for each active device, and each device has a default set of parameters when initialized. Graphics parameters can be set in two ways: either permanently, affecting all graphics functions which access the current device; or temporarily, affecting only a single graphics function call. To get the entire list, containing all graphical parameters, type

```
> help(par)
```

The `par` function is used to access and modify the list of graphics parameters for the current graphics device. Instead of describing in more detail all of the commonly-used graphical parameters, consult the S-Plus help documentation for the `par` function, which provides a concise summary, including `adj`, `cex`, `pch`, `ask`, `col` and `lwd`. Moreover, as examples, I will describe in this section only `pty` and `mfrow`.

When you use a S-Plus plotting function, the default shape of the box enclosing the plot is rectangular. Sometimes you prefer to have a square box around your plot. You get a square box by using the global graphics parameter function par as follows:

```
> par(pty="s")
```

All subsequent plots are made with a square box around the plot. If you want to return to making rectangular plots, use `par(pty="")`. The `pty` stands for 'plot type' and the `"s"` stands for square. You may also want to display more than one plot on your screen or on a single page of paper. To do so, you use the layout parameter `mfrow` to control the layout of the plots. If you want to set up a four-plot layout, with two rows of two plots each, type

```
> par(mfrow=c(2,2))
```

When you are ready to return to one plot per figure, use `par(mfrow=c(1,1))`; a similar parameter is `mfcol`. As a final example, the various possibilities of the `pch`, the number of a plotting symbol to be drawn when plotting points, and `lty`, the device dependent line type, arguments are given in Figure 4.5.

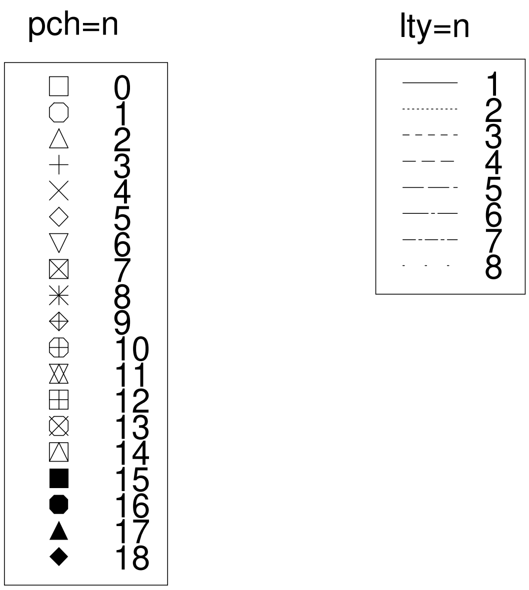


Figure 4.5: Various possibilities of the `pch` and `lty` arguments.

4.6 Exercises

Exercise 4.1. Create a sequence, `x`, from -10 to 10 with spacing 0.5.

(a) Create a vector, `y`, containing the squares of `x`.

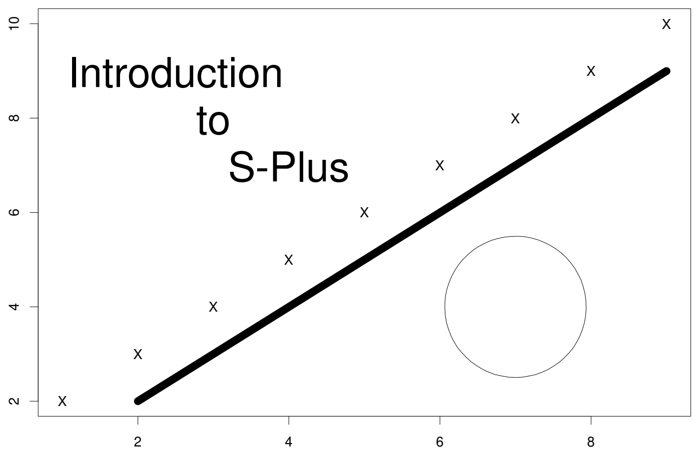
(b) Make a scatterplot of the two vectors.

(c) Reproduce additional scatterplots by changing its arguments, like `type`, `xlab`, `ylab`, `lwd`, `col`.

Exercise 4.2. Plot a pie chart with 15 colors on the background color, color 0, for a total of 16 colors. This pie chart is of particular interest as the default color map for graphsheet has sixteen colors: fifteen foreground and one background color. So, this chart can be used to see all the colors in the default color map.

Exercise 4.3. Plot `sin(x)` against `x`, using 200 values of `x` between `-pi` and `pi`, but do not plot any axes yet (use parameter `axes=F` in the call to plot). Add an y-axis passing through the origin using the 'extended' style and horizontal labels. Add an x-axis with tick-marks from `-pi` to `pi` in increments of `pi/4`, twice the usual length.

Exercise 4.4. Reproduce the following figure.



Exercise 4.5. Try out the various features of the `motif` device. To do so, use the following S-PLUS commands to generate an easily-reproducible graphic:

```
> plot(corn.rain, corn.yield, type="n", main="Plot Example")
> points(corn.rain, corn.yield, pch="*", col=2)
> lines(lowess(corn.rain, corn.yield), lty=2, col=3)
> legend(12, 23, c("Color 1", "Color 2", "Color 3"), pch="*",
+ lty=c(1, 0, 2), col=c(1, 2, 3))
```

Try to understand this commands and play around with its arguments.

Exercise 4.6. The data frame `survey.dat` from the library `s+intro` contains the results of a survey of 237 first-year Statistics students at Adelaide University. This data is taken from Venables and Ripley (1999).

(a) For a graphical summary of all the variables, use `plot(survey.dat)`. What kind of plots are produced?

(b) One component of this data frame, `Exer`, is a factor object containing the responses to a question asking how often the students exercised.

(i) Produce a barchart of these responses.

(ii) Use `table` and `pie` (or `piechart`) to create a pie chart of the responses. Do you like this better than the barplot? Which is more informative? Which gives a better picture of exercise habits of students?

(iii) The `pie` function takes an argument `names` which can be used to put labels on each pie slice. Redraw the pie chart with labels.

(iv) Alternatively, you could add a legend to identify the slices.

(c) Try the same things with the `Smoke` variable, which records responses to the question: 'How often do you smoke?'

Hint: note that `table` and `levels` ignore missing values; if you wish to include non-respondents in your chart use `summary` to generate the values, and `names` on the summary object to generate the labels.

Chapter 5

Exploring Data

5.1 Multivariate Data Exploration

The ethanol data (stored in the data frame `ethanol`) records 88 measurements for three variables `NOx`, `C`, and `E` from an experiment in which ethanol was burned in a single cylinder automobile test engine. A short description can be obtained by

```
> ?ethanol
```

A summary of this data is given by

```
> summary(ethanol)
      NOx              C                E
 Min.   :0.370   Min.   : 7.500   Min.   :0.5350
 1st Qu.:0.953   1st Qu.: 8.625   1st Qu.:0.7618
 Median :1.754   Median :12.000   Median :0.9320
 Mean   :1.957   Mean   :12.030   Mean   :0.9265
 3rd Qu.:3.003   3rd Qu.:15.000   3rd Qu.:1.1100
 Max.   :4.028   Max.   :18.000   Max.   :1.2320
```

S-Plus provides two very useful functions for representing multivariate data. First, to produce a pairwise scatterplot matrix of the variables defined by the columns of `ethanol`, you can use

```
> par(pty="s")     # square plotting regions
> pairs(ethanol)
```

The resulting plot appears as in Figure 5.1. Using the function `persp`, a perspective plot for the ethanol data can be easily computed:

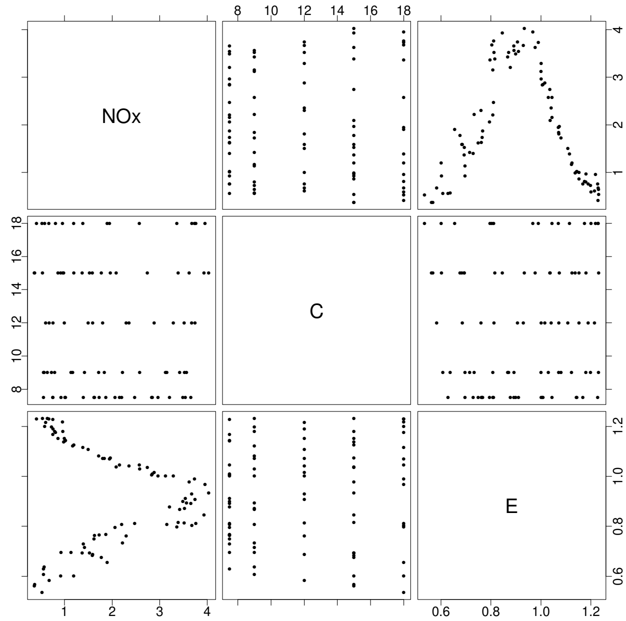


Figure 5.1: Pairwise scatterplots of the variables in the ethanol data.

```
> attach(ethanol)
> inter <- interp(E, C, NOx)     # interpolate the value of NOx
> persp(inter, xlab="E", ylab="C", zlab="NOx")
```

Note the the actual S-Plus 3D-plots need to be on a evenly spaced grid, i.e. the third variable needs to be interpolated onto an evenly spaced grid (!) of the first two variables. Figure 5.2 shows this three-dimensional perspective plot, whereas two-dimensional plots,

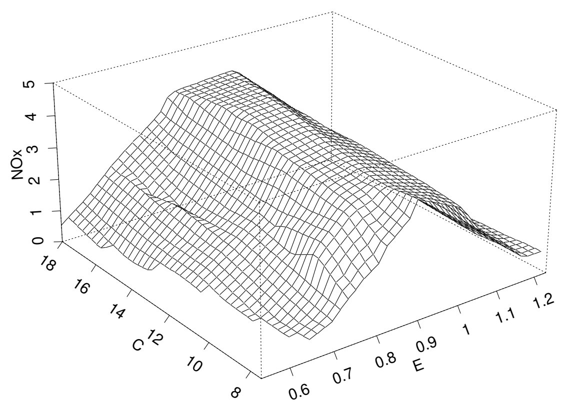


Figure 5.2: Perspective plot for the ethanol data.

namely contour and image, are represented in Figure 5.3. For the later, the following S-PLUS commands were used:

```
> par(mfrow=c(1,2), pty="s")
> contour(inter, xlab="E", ylab="C")
> title("Contour Plot")
> image(inter, xlab="E", ylab="C")
> title("Image Plot")
```

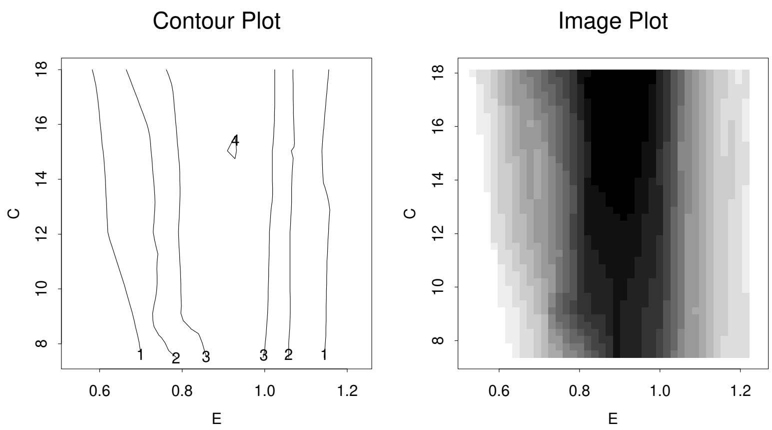


Figure 5.3: Contour and image plots for the ethanol data.

When three or four variables are involved a coplot may be more enlightening.

```
> par(pty="n")
> E.intervals <- co.intervals(E, 9, 0.25)
> coplot(NOx ~ C | E, given.values=E.intervals, data=ethanol,
+ panel=function(x, y) panel.smooth(x, y, span=1, degree=1))
```

NOx ~ C | E means that the function produces a number of scatterplots of NOx, the response, against C, a predictor, for given values of E, a predictor; see Section 7.1. This plot provides a graphical look at cross-sectional relationships, which enable you to assess potential interaction affects. Figure 5.4 clearly shows the effect of the interaction between C and E on values of NOx.

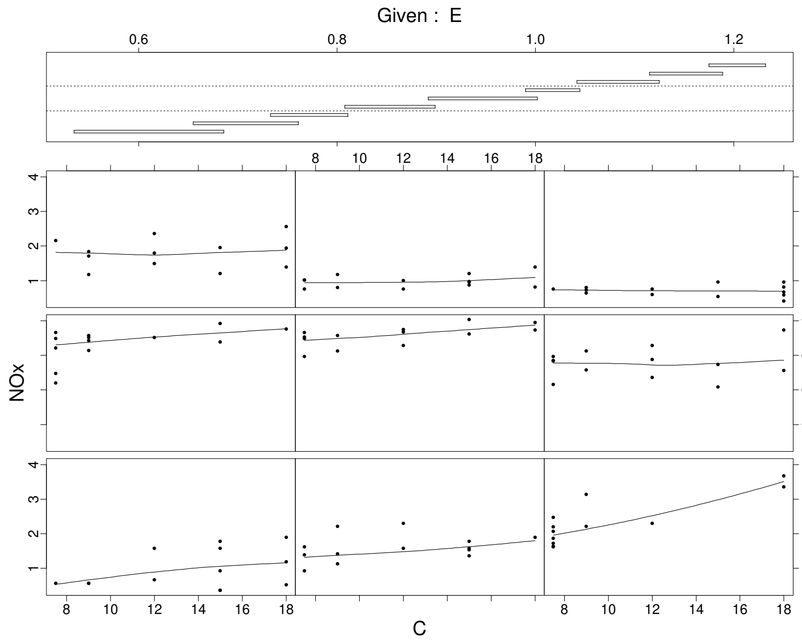


Figure 5.4: Coplot of response and predictors for the ethanol data.

5.2 Exercises

Exercise 5.1 (LaTeX data). Consider the data frame `LaTeX.dat` described in Exercise 3.15.

(a) By means of `summary` compute a summary of the data.
Hint: use the argument `maxsum`.

(b) Compute the overall percentage of the use of LaTeX.

(c) Produce a summary of the 1996 entries, and calculate the percentages for this particular year.

(d) Make a plot of the data `LaTeX.dat`:

```
> par(mfcol=c(3,1))
> plot(LaTeX.dat, ask=F)
```

What happens if you omit the argument `ask=F`?
Comment the produced graphics.

Exercise 5.2 (Phone data). Consider the artificial data frame `phone.dat` from the course library `"s+intro"`. This data set consists of the length in seconds of 200 phone calls.

(a) Determine what the distribution looks like by using the stem and leaf display.

(b) Make a histogram showing 10 classes, i.e. bars. First, by using the `breaks` argument, and second by specifying the `nclass` argument.

(c) Produce a boxplot of the data, and compute the 10%, 25%, 50%, 75% and 90% quantiles, as well as the mean, the interquartile distance and the standard deviation. What do you conclude?

Exercise 5.3 (Old Faithful geyser data). The 'Old Faithful Geyser Data', `geyser`, contains the waiting time between eruptions, `waiting`, and the duration of the eruption, `duration`, for the Old Faithful geyser in Yellowstone National Park, Wyoming. The data were collected continuously from August 1st until August 15th, 1985. There are a total of 299 observations. The waiting time is the time interval between starts of successive eruptions. The times are measured in minutes. If you type

```
> geyser
```

you see that this data is represented in form of list containing the two variables `waiting` and `duration`. Our aim is to look at some simple descriptive techniques to help understand the relationship between the interval between eruptions and the duration of the eruption.

(a) Make a summary for the two variables `waiting` and `duration`. Comments?

(b) Produce histograms, normal probability plots, boxplots and summary statistics for the duration and interval between eruptions.

(c) Produce a scatterplot of the data, showing the relationship between the time between eruptions and the duration of the eruption.

Is there a relationship between the interval between eruptions and the duration of the next eruption? How does the length of the interval effect the duration and does this account for the bimodal histograms found?

Exercise 5.4 (Car data). Do an analysis of the car data set provided with S-Plus. The data set, stored under `car.all`, contains a set of cars plus a variety of their attributes, like fuel consumption, price, and many more.

(a) Before starting the analysis, determine which car models are contained in the data and what the variables given in the data set are.

(b) Extract the fuel consumption (miles per gallon) and determine for each type of car (small, medium, compact, large, van, sporty) the model with the lowest and the highest fuel consumption.

(c) On the basis of average fuel consumption and tank size, calculate the maximum travel distance without a gas refill.

(d) Take the variables maximum travel distance, tank size, mileage, and horsepower, and display them graphically.

(e) Try to determine, using only graphics, the two variables with the strongest dependency. Check your guess by calculating the correlation matrix (using the function `cor`) for these variables.

Exercise 5.5 (U.S. States data). The data frame `state.x77` contains the following statistics for the States of the U.S.: population estimate as of July 1, 1975; per capita income (1974); illiteracy (1970, percent of population); life expectancy in years (1969-71); murder and non-negligent manslaughter rate per 100,000 population (1976); percent high-school graduates (1970); mean number of days with minimal temperature smaller than 32 degrees (1931-1960) in capital or large city; and land area in square miles.

S-Plus provides a very useful function for representing such multivariate data: `brush`. This functions creates a matrix of all two-dimensional scatterplots of the data plus optional histograms and three-dimensional spinning plot, all of which may have points highlighted interactively. By typing

```
> brush(state.x77, hist=T)
```

you should see a full-screen graphics window appear with the title 'Brush'.

(a) Try to understand the graphics given.

(b) Highlight several points with the left mouse button. What happens?

(c) Above to the right of the scatter plots, there is a separate sub-window with three-dimensional coordinate axes and a 3-D point cloud. Rotate or spin the 3-D point cloud. Do you see any particular structure?

To exist `brush` at any time, click on the 'Quit' button. Note that the sub-window with the three-dimensional coordinate axes and the 3-D point cloud can also be obtained by typing

```
> spin(state.x77)
```

Comments?

Chapter 6

Statistical Functions

6.1 Densities and Distributions

S-Plus has built-in functions to compute the density, cumulative distribution function (CDF), quantile function (the inverse of the CDF) and to generate random samples from many standard distributions. In all, there are four functions for each distribution. The function names begin with one of the four letters:

- `d`: is for density, e.g. `dnorm`;
- `p`: stands for probability , e.g. `pnorm`;
- `q`: is for quantile, e.g. `qnorm`; and
- `r`: stands for random, e.g. `rnorm`.

The function names end with a code for the distribution. The random functions generate numbers from the distribution, the probability functions give the probability of being less than or equal to the value given (probability distribution function), and the quantile functions return the value where the distribution function achieves the probability given. The list of available distributions in S-Plus is given in Table 6.1.

Table 6.1: Available standard distributions in S-PLUS.

Name	**Distribution**	**Parameter(s)**
`beta`	Beta	`shape1`, `shape2`
`binom`	Binomial	`size`, `prob`
`cauchy`	Cauchy	`location`, `scale`
`chisq`	χ^2	`df=` degrees of freedom
`exp`	Exponential	`rate`
`f`	Fisher's F	`df1=` degrees of freedom for the numerator `df2=` degrees of freedom for the denominator
`gamma`	Gamma	`shape`
`geom`	Geometric	`prob`
`hyper`	Hypergeometric	`m=` number of red balls in the urn `n=` number of black balls in the urn `k=` number of balls drawn from an urn with `m` red and `n` black balls
`lnorm`	Lognormal	`meanlog=` log of the mean `sdlog=` log of the standard deviation
`logis`	Logistic	`location`, `shape`
`nbinom`	Negative binomial	`size=` number of successes `prob=` probability of a success
`norm`	Normal (Gaussian)	`mean=` mean `sd=` standard deviation
`pois`	Poisson	`lambda`
`t`	Student's t	`df=` degrees of freedom
`unif`	Uniform	`min`, `max`
`weibull`	Weibull	`shape`
`wilcox`	Wilcoxon	`m=` number of observations in first sample `n=` number of observations in second sample

6.1.1 Randomness

Pseudo-random functions within S-PLUS use a single random number generator. The seed for the generator is stored in `.Random.seed`, which is a vector of 12 small integers. So, if you want to reproduce results of random functions, then you need to save the seed and restore it just before the call that is to reproduce the results:

```
save.seed <- .Random.seed
> runif(4)
[1] 0.5354037 0.5107591 0.1341979 0.3324095
> runif(4)
[1] 0.6010147 0.2729012 0.8917196 0.3833721
> .Random.seed <- save.seed
> runif(4)
[1] 0.5354037 0.5107591 0.1341979 0.3324095
```

Another way of ensuring reproducibility is to use the function `set.seed`, which puts the random number generator in a reproducible state. `set.seed` takes an integer that should be between 1 and 1024, and sets the random seed to a certain value based on the integer given.

```
> set.seed(1023)
> print(.Random.seed)
> [1] 21 14 49 16 62 1 32 22 36 23 28 3
> set.seed(-1)
> print(.Random.seed)
> [1] 21 14 49 16 62 1 32 22 36 23 28 3
```

The example above shows that `set.seed` returns the same answer for arguments that are the same modulo 1024.

6.1.2 QQ-Plots

One of the best ways to compare the distribution of a sample, `x.sample`, with a distribution is to use a QQ-plot, of which the normal probability plot is the best-known example. QQ-plots can also be used to compare two samples. The function `qqplot` plots the quantile functions of two samples `x.sample` and `y.sample` against each other, and so compares two samples. The function `qqnorm` replaces one of the samples by a sample at the quantiles of a standard normal distribution. This idea can be applied quite generally, see the `ppoints` function. Finally, the command `qqline` helps assess how straight a `qqnorm` plot is by plotting a straight line through the upper and lower quartiles. The examples, which are shown in Figure 6.1, are

```
> par(mfcol=c(1,2), pty="s")
> qqplot(lottery.payoff, lottery3.payoff)
> abline(0, 1)

> qqnorm(lottery.payoff)
> qqline(lottery.payoff)
```

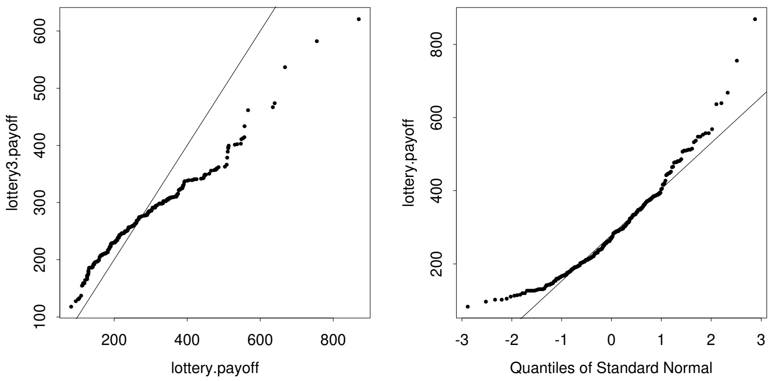


Figure 6.1: Two examples of the use of QQ-plots in S-Plus.

The left panel in Figure 6.1 suggests a basically different distribution for the two sets of data. If both data would have a similar distribution the presented line would match the plot. In the right panel of Figure 6.1 one notices that the payoffs are not normal distributed, especially in the tails; whereas the middle part of the payoffs seems to be. If they were, the plot would be fairly linear matching the line resulting from the `qqline` command.

6.2 Basic Statistics

S-Plus includes functions for calculating all the standard summary statistics for a data set, together with a variety of robust and/or resistant estimators of location and scale. Table 6.2 gives a list of the most common functions for summary statistics.

Table 6.2: Common functions for summary statistics in S-Plus.

Statistic	Description
`cor`	Correlation coefficient
`cummax`, `cummin`	Cumulative maximum and minimum
`cumprod`, `cumsum`	Cumulative product and sum
`diff`	Create sequential differences
`max`, `min`	Maximum and minimum
`pmax`, `pmin`	Maxima and minima of several vectors
`mean`	Arithmetic mean
`median`	50th percentile
`prod`	Product of elements of a vector
`quantile`	Compute empirical quantiles
`range`	Returns minimum and maximum of a vector
`sample`	Random sample or permutation of a vector
`sum`	Sum elements of a vector
`summary`	Summarize an object
`var`	Variance and covariance

Consider as example the `sample` function. In its simplest use, it returns a permutation of its argument:

```
> names.vector <- c("Introduction", "to", "S-PLUS", "for", "Unix")
> sample(names.vector)
[1] "S-PLUS" "to" "Introduction" "Unix" "for"
> sample(1:10)
[1]  6  5  1  2  3  8  7 10  4  9
```

An important argument is `replace` which tells whether or not items are to be replaced after they have been selected:

```
> sample(names.vector, replace=T)
[1] "Introduction" "for" "Introduction" "S-PLUS" "S-PLUS"
> sample(1:10, replace=T)
[1] 10 10  1 10  2  5  3  6  7 10
```

There is also a probability argument, `prob`, that takes the relative probabilities for each item in the population. This need not sum to one, sample will normalize it.

```
> sample(names.vector, replace=T, prob=1:5)
[1] "S-PLUS" "Unix" "for" "Unix" "Unix"
```

6.3 Standard Tests

S-Plus contains a number of functions for doing classical hypothesis testing; see Table 6.3.

Table 6.3: S-Plus functions for hypothesis testing.

Test	Description
`binom.test`	Exact binomial test
`chisq.gof`	χ^2 goodness-of-fit test
`chisq.test`	Pearson's χ^2 test for 2-D contingency table
`cor.test`	Test for zero correlation
`fisher.test`	Fisher's exact test for 2-D contingency table
`friedman.test`	Friedman rank sum test
`kruskal.test`	Kruskal-Wallis rank sum test
`ks.gof`	Kolmogorov-Smirnov test
`mantelhaen.test`	Mantel-Haenszel χ^2 test
`mcnemar.test`	McNemar's χ^2 test
`prop.test`	Proportions test
`t.test`	Student's one-or two-sample t-test
`var.test`	F test to compare two variances
`wilcox.test`	Wilcoxon rank sum and signed-rank sum tests

The following example illustrates how to use `t.test` to perform a two-sample t-test to detect a difference in means. This example uses two random samples generated from $N(0,1)$ and $N(1,1)$ distributions. The random number seed is set with the function `set.seed`, so this example is reproducible:

```
> set.seed(17)
> x.sample <- rnorm(10)
> y.sample <- rnorm(5, mean=1)
> t.test(x.sample, y.sample)

         Standard Two-Sample t-Test

data:  x.sample and y.sample
t = -1.3353, df = 13, p-value = 0.2047
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.4926916  0.3522959
sample estimates:
 mean of x mean of y
 0.1375302  0.707728
```

As the `p-value` of this two-sample t-test is `0.2047`, there is no significant difference between both means.

As an additional example, suppose you toss a coin 1000 times, with heads coming up 473 times. To test the coin for 'fairness'; that is, to test that the probability of heads equals 0.5, you can use the following:

```
> binom.test(473, 1000)$p.value

         Exact binomial test

data:  473 out of 1000
number of successes = 473, n = 1000, p-value = 0.0937
alternative hypothesis: true p is not equal to 0.5
```

This test suggests that you do have a fair coin. You may not want to take another coin.

6.4 Exercises

Exercise 6.1. Generate 100 and 1000 random numbers, both samples from a normal distribution with mean value 3 and variance 5. Draw histograms with bin width 0.5, 1 and 2 for each of the two samples. Remember that all figures have exactly the same underlying distribution. Plot them all in a single graphics window and label them accordingly. What is visible?

Exercise 6.2. Generate a sample of size 50 from a Poisson distribution with rate 2.

(a) Put this sample into a 10×5 matrix `A`.

(b) Name the rows of `A` with `r1,..., r10` and the columns with `c1,..., c5`.

(c) Divide each element in a row of `A` by the sum of the corresponding row.

Exercise 6.3. Generate a 20×3 matrix of data using the `lrnorm` function. Produce a 'bubble plot' of the data, i.e. a scattergram of the values in the first two columns which includes the values in the third column represented by circles with appropriate radii. Hint: use the `symbols` function.

Exercise 6.4. Generate a sample of size 100 from a t-distribution with 5 degrees of freedom. Produce a QQ-plot of this sample and assess how straight the produced plot is by plotting a straight line. Moreover, in order to reproduce the same plots, set the random seed.

Exercise 6.5. In Chapter 2 we noticed that the distribution of payoffs for winning numbers with a leading zero digit was different from the rest of the payoffs. Use a QQ-plot to investigate this difference.

Exercise 6.6. Suppose you play roulette in Las Vegas and bet on red. You expect your probability of winning to be close to, but slightly less than, 0.5, because a roulette wheel has 18 red slots, 18 black slots, and two additional slots labeled '0' and '00', for a total of 38 slots into which the ball can fall. Thus, for a 'fair' wheel, you expect the probability of red to be $p_0 = 18/38 = 0.474$. You hope that the casino is not cheating you altering the roulette wheel so that the probability of red is less than 0.474.

Suppose that you bet on red 100 times and red comes up 42 times. Test the hypothesis that $p = p_0$, using the `binom.test` function,

(a) against the two-sided alternative that $p \neq 0.474$; and

(b) against the one-sided alternative $p < p_0$, i.e. you are concerned that the casino might cheat you.

Finally, as the function `binom.test` does not compute a confidence interval for the probability p of success, compute a confidence interval for p by using the `prop.test` function. Comments?

Exercise 6.7 (IQ data). To get this data, which is not built-in, you must attach the library `s+intro` by typing `library("s+intro")`. This library contains the data frame `iq.data`, which consists of the IQ scores and behavioural problem (BP) scores of children of age five, labelled according to whether or not their mothers had suffered an episode of postnatal depression.

(a) Store the IQ scores of children of non-depressed mothers in `iqnd`, and those of children of depressed mothers in `iqd`.

The main questions of interest about these data are as follows:

(b) Do the two groups of children have different IQ and/or BP scores? Most commonly a t-test is used to assess these hypothesis. The test assumes that the observations:

   1. are independent of one another (here they are);
   2. come from populations having normal distributions (verify with histograms, boxplots, and probability plots and exclude the outliers); and
   3. come from populations having the same variance (verify with the `var.test` function).

   Assess the hypothesis that the two groups have the same population mean with with the S-PLUS function `t.test` and with the Wilcoxon rank-sum test (`wilcox.test`).

(c) Is there any evidence of a relationship between the two variables? Plot BP versus IQ scores and add a linear regression line. Moreover, the correlation between the two variables can be calculated, and the hypothesis that the population value is zero tested, using the `cor.test` function.

(d) Assess whether there is any evidence of an association between mother's mental state at the birth of her child, and the child's behavioural problems scores at five, in particular whether this score is above 8 or not.
Hint: make a χ^2-test by means of `chisq.test`, followed by Fisher's exact test (`fisher.test`).

Chapter 7

Statistical Models

Most of the statistical modeling functions in S-Plus follow a unified modeling paradigm in which the input data are represented as a data frame and the model to be fit is represented as a formula. Formulas can be saved as separate S-Plus objects and supplied as arguments to the modeling functions. A partial listing of S-Plus modeling functions is given in Table 7.1.

Table 7.1: Listing of some S-Plus modeling functions.

Function	**Description**
`aov`, `manova`, `raov`	Analysis of variance models
`lm`	Linear model (regression)
`glm`	Generalized linear model
`gam`	Generalized additive model
`loess`	Local regression model
`tree`	Classification and regression tree models
`nls`, `ms`	Nonlinear models
`lme`, `nlme`	Mixed-effects models
`factanal`	Factor analysis
`princomp`	Principal components analysis

7.1 Basic Formulas

In a formula, you specify the response variable first, followed by a tilde, '~', and the terms to be included in the model, separated by operators. Variables in formulas can be any expression that evaluates to a numeric vector, a factor or ordered factor, or a matrix. Table 7.2 gives a summary of the formula syntax.

Table 7.2: Summary of the S-Plus formula syntax.

Expression	Meaning
`A ~ B`	`A` is modeled as `B`
`B + C`	Include both `B` and `C` in the model
`B - C`	Include all of `B` except what is in `C` in the model
`B:C`	The interaction between `B` and `C`
`B*C`	Include `B`, `C`, and their interaction in the model
`C %in% B`	`C` is nested within `B`
`B/C`	Include `B` and `C %in% B` in the model

An example of a simple formula linking your chosen response variable to the predictor variable is given by:

```
> cost ~ age
```

Both, `cost` and `age`, are variables from the `claims` data. So, the previous formula specifies that the variable `cost` is modeled as a function of `age`.

7.2 Linear Models: `lm`

Simple linear regression explores relationships that are readily described by straight lines. The classical simple regression uses the method of least squares to fit a continuous, univariate response as a linear function of a single predictor variable. In the method of least squares, we fit a line to the data so as to minimize the sum of the squared residuals.

As example, consider the air pollution data in the built-in data set `air`, which is taken from an environmental study that measured the four variables `ozone`, solar `radiation`, `temperature`, and `wind` speed for 111 consecutive days. A scatter plot of `temperature` against `ozone` is shown in Figure 7.1;

```
> attach(air)
> plot(temperature, ozone)
```

From the scatter plot, we may hypothesize a linear relationship between `temperature` and `ozone` concentration. We choose `ozone` as the response, and `temperature` as the single predictor. The choice of response and predictor variables is driven by the subject matter in which the data arise, rather than by statistical considerations. To fit the model, we can use the `lm` function as follows:

```
> ozone.lm <- lm(ozone ~ temperature, data=air)
```

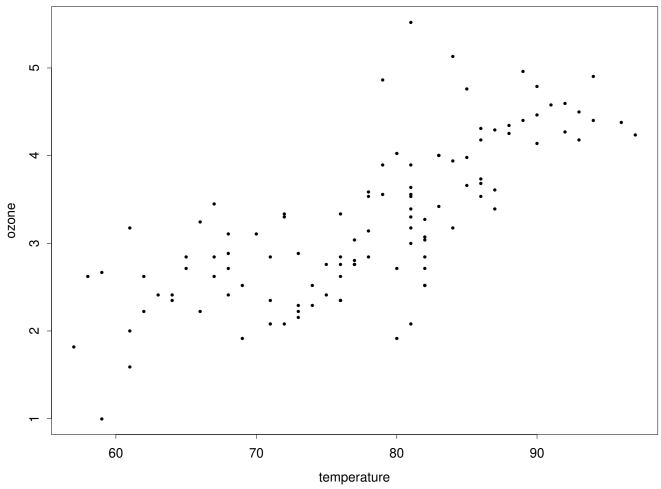


Figure 7.1: Scatterplot of `temperature` against `ozone` for the `air` data.

The first argument, `ozone ~ temperature`, is the formula specifying that the variable `ozone` is modeled as a function of `temperature`. The second argument specifies that the data for the linear model is contained in the data frame `air`. The `summary` function can be used to obtain a summary of the fitted model:

```
> summary(ozone.lm)

Call: lm(formula = ozone ~ temperature, data = air)
Residuals:
   Min      1Q  Median     3Q   Max
 -1.49 -0.4258 0.02521 0.3636 2.044

Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept)  -2.2260   0.4614    -4.8243   0.0000
temperature   0.0704   0.0059    11.9511   0.0000

Residual standard error: 0.5885 on 109 degrees of freedom
Multiple R-Squared: 0.5672
F-statistic: 142.8 on 1 and 109 degrees of freedom, the p-value is 0

Correlation of Coefficients:
            (Intercept)
temperature -0.9926
```

The `Value` column under `Coefficients` gives the coefficients of the linear model, allowing us to read off the estimated regression line: `ozone = -2.2260 + 0.0704 temperature`. The estimated standard error is given in the column labelled `Std. Error`, and the term

`Multiple R-Squared` tells us that the model explains about 57% of the variation in `ozone`. The `F-statistic` is the ratio of the mean square of the regression to the estimated variance; if there is no relationship between `temperature` and `ozone`, this ratio has an F distribution with 1 and 109 degrees of freedom. The ratio here is clearly significant, so the true slope of the regression line is probably not zero.

But how good is the fitted regression model? To answer this question the simplest and most informative method consists in looking at the model graphically. Therefore we can use the default plotting method for `lm` objects by means of:

```
> par(mfrow=c(2,3))
> plot(ozone.lm, ask=F)
```

If you are not interested in all of the plots created by the default plotting method, call `plot` with the argument `ask=T`. As shown in Figure 7.2, this yields the following six plots:

1. *Residuals against fitted values.* This plot reveals unexplained structure in the residuals.

2. *Square root of absolute residuals against fitted values.* This plot is useful in identifying outliers and assessing variability in the residuals.

3. *Response against the fitted values.* This gives a good idea of how well the model has captured the broad outliers of the data.

4. *Normal quantile plot of residuals.* This plot provides a visual test of the assumption that the model's errors are normally distributed.

5. *Residual-Fit spread plot.* This plot compares the spread of the fitted values with the spread of the residuals. Since the model is an attempt to explain the variation in the data, one hopes that the spread in the fitted values is much greater than that in the residuals.

6. *Cook's distance plot.* Cook's distance is a measure of the influence of individual observations on the regression coefficient.

The line shown as a dashed line in the third plot (far right of top row in Figure 7.2) is in the case of simple regression visually equivalent to the regression line. The regression line appears to model the trend of the data reasonably well. The residuals plots (left and center, top row) show no obvious pattern, although five observations appear to be outliers. By default, as in Figure 7.2, the three most extreme values are identified in each of the residuals plots and the Cook's distance plot; see `id.n` argument in the call to `plot`. Another useful diagnostic plot is the normal plot of residuals (left plot, bottom row). The normal plot gives no reason to doubt that the residuals are normally distributed. The Residual-Fit spread plot, on the other hand (middle plot, bottom row), shows a weakness in this model; the spread of the residuals is actually greater than the spread in the original data. However, if we ignore the five outlying residuals, the residuals are more

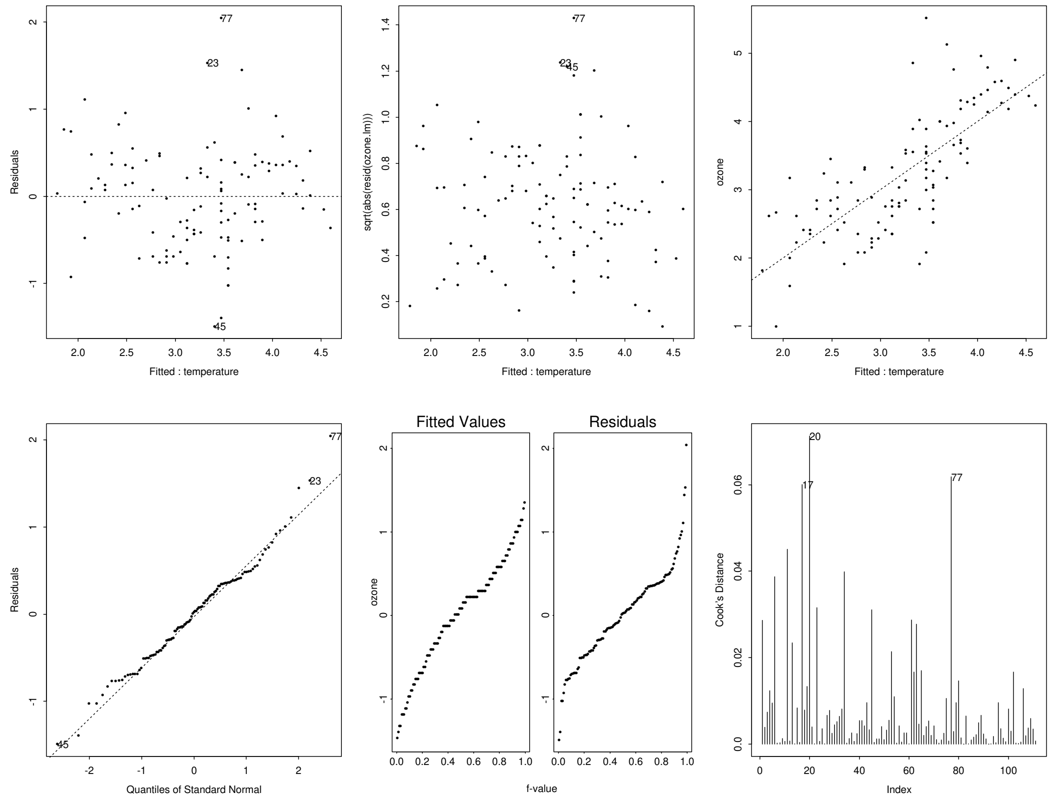


Figure 7.2: Use of the default plotting method for `lm` objects for `ozone.lm`.

tightly bunched than the original data. The Cook's distance plot shows four or five heavily influential observations. As the regression line fits the data reasonably well, the regression is significant, and the residuals appear normally distributed, we feel justified in using the regression line as a way to estimate the ozone concentration for a given temperature.

One important issue remains — the regression line explains only 57% of the variation in the data. Therefore, we may want to consider the effect of other variables on the ozone concentration. Constructing linear models involving more than one predictor is as easy in S-Plus as models with a single predictor. One possible way is to use the `lm` function with the formula `ozone ~ temperature + wind + radiation`.

7.3 Others Models

A variety of statistical models are analyzed in Becker and Hastie's book (1992, *Statistical Models in S*), including analysis of variance, generalized linear models, additive models, local regression, nonlinear models, mixed-effects models, and tree-based models. As they go far beyond the coverage of this guide, only the names of the corresponding S-Plus functions are listed in Table 7.1. In addition, you may want to look at the excellent book by Venables and Ripley (1999, *Modern Applied Statistics with S-Plus*).

7.4 Exercises

Exercise 7.1 (Simple linear regression). Consider the cell survival data, which is in the course library `s+intro`. The two variables are `dose`, the dose of drug given to each cell, and `logsurv`, the logarithm of the resulting survival time.

```
> cell.dat
     dose    logsurv
 1  1.175 -0.8209806
 2  1.175 -0.5978370
 3  2.350 -1.8325815
 4  2.350 -2.0402208
 5  4.700 -3.2188758
 6  4.700 -3.9322257
 7  4.700 -2.7936081
 8  7.050 -5.2983174
 9  7.050 -5.7446045
10  9.400 -6.8124451
11  9.400 -8.8048753
12  9.400 -8.5684865
13 14.100 -4.9618451
14 14.100 -9.7211660
```

(a) Make a scatter plot of the data, i.e. plot `dose` against `logsurv`.

(b) Do a simple linear regression with S-Plus in choosing `logsurv` as the response and `dose` as the predictor variable. Fit the model by means of the function `lm`. Produce a summary of the fitted model. Finally, plot the fitted line using the function `abline`. Comments?

(c) Redo (b) by using `cell.dat` without the 13th observation, and compare with (b). What do you conclude?

Exercise 7.2 (Multiple Regression). Consider the Scottish hill races data, which is in the course library `s+intro`. The data set is taken from Staudte and Sheather (1990, *Robust Estimation and Testing*, Wiley and Sons, New York). The data is in the data frame `race.dat`.

(a) Fit a linear model in taking as dependent variable the record time in minutes (`RecTime`) and as independent variables the distance in miles (`Dist`) and the climb in feet (`Climb`).

(b) Use the S-Plus function `summary` to obtain the summary of the fitted model. What model do you read off?

(c) Look at the fit graphically. To do this use the `plot` function. In considering the resulting plots, what do you conclude?

(d) Try to fit a generalized additive model. Do you think the linear model is appropriate?

Exercise 7.3 (Advanced). *Nonparametric regression with the Great Barrier Reef data.* In a survey of the fauna on the sea bed in an area lying between the coast of northern Queensland and the Great Barrier Reef, data were collected at a number of locations. Data and the needed functions are in the S-Plus library `sm`, which comes with Bowman and Azzalini (1997). This library can be downloaded directly from

`http://www.stats.gla.ac.uk/~adrian/sm/`

Try the following commands :

```
> library(sm)
> provide.data(trawl)
> par(mfrow=c(2,2))
> plot(Longitude, Latitude, type="n")
> points(Longitude[Zone == 1], Latitude[Zone == 1])
> text(Longitude[Zone == 0], Latitude[Zone == 0], "o")
> Zone93 <- (Year == 1 & Zone == 1)
> Position <- cbind(Longitude - 143, Latitude)
> sm.regression(Latitude[Zone93], Score1[Zone93], h=0.1)
> sm.regression(Position[Zone93,], Score1[Zone93],
              h=c(0.1, 0.1), eye=c(8,-6, 5), xlab="Longitude - 143")
> sm.regression(Longitude[Zone93], Score1[Zone93], h=0.1)
> par(mfrow=c(1,1))
```

The top left panel displays the sampling points. The dots are taken from a closed region, where commercial fishing is not allowed to protect the nature. The circles refer to areas outside, which were surveyed to allow comparisons. Due to large numbers and types of species captured in the survey, the response variable is expressed as a score, on a log weighted scale, which combines information across species. The two right panels of the plot show nonparametric estimates of the relationship between the catch score within the closed zone and the spatial coordinates, latitude and longitude, for the 1993 survey.

(a) Can we observe a change in the catch score with latitude ?

(b) Can we observe a change in the catch score with longitude ?

Exercise 7.4 (Advanced). *Generate artificial data.* Generate data using the following code:

```
> n <- 100
> x <- sort(runif(n))
> m <- (sin(2*pi*x^3))^3
> y <- m + rnorm(n)*sqrt(0.1)
> plot(x, y, main="Generated data")
> lines(x, y)
```

Apply smoothing techniques like `ksmooth`, `lowess`, `loess` and `smooth.spline` to the generated data. Choose different smoothing parameters and observe their influence on the smoother.

Exercise 7.5 (Advanced). *Kernel or wavelet smoothing.* In this exercise we propose to compare the predictive performance of the kernel smoother with that of the wavelet smoother, based on simulated data. We generate data according to the additive model

$$y_i = f(x_i) + \epsilon_i,$$

where $\epsilon \overset{iid}{\sim} N(0, 1)$. Here we use the following non-smooth `block` function for $f(\cdot)$:

```
blocks <- function(x, snr = 5)
{
 pos <- c(0.1, 0.13, 0.15, 0.23, 0.25, 0.4, 0.44, 0.65, 0.76, 0.78, 0.81)
 hgt <- c(4, -5, 3, -4, 5, -4.2, 2.1, 4.3, -3.1, 2.1, -4.2)
 y <- rep(0, length(x))
 for(j in 1:length(pos))  y <- y + ((1 + sign(x - pos[j])) * hgt[j])/2
 (snr * y)/1.914
}
```

The following commands create a plot in which the dots are the noisy data y_i and the dotted line is the true `block` function.

```
> n <- 256
> x <- seq(0, 1, length=n)
> ytrue <- blocks(x)
> ynoise <- ytrue + rnorm(n)
> plot(ynoise)
> lines(ytrue, lty=2)
```

(a) First we try the kernel smoother `ksmooth`. After typing the following commands, choose your 'best' bandwidth among $(0.001, 0.01, 0.1, 1)$ to estimate `block`:

```
for(bandwidth in c(0.001, 0.01, 0.1, 1))
{
```

```
 plot(ynoise, pch=".")
 kernel.estimate <- ksmooth(x, ynoise, bandwidth=bandwidth)
 lines(kernel.estimate$y)
 lines(ytrue, lty=2)
 title(paste("bandwidth=", as.character(bandwidth)))
}
```

You may want to set `par(mfrow=c(2,2))`.

(b) Second we try the wavelet smoother **waveshrink**, assuming you have the S+Wavelets module:

`http://www.insightful.com/products/wavelets/`

Again choose your 'best' threshold among $(0.1, 1, 5, 10)$:

```
module(wavelets)
for(vshrink in c(0.1,0 1, 5, 10))
{
 plot(ynoise, pch=".")
 wavelet.estimate <- waveshrink(ynoise, wavelet="haar", vshrink=vshrink)
 lines(wavelet.estimate)
 lines(ytrue, lty=2)
 title(paste("Threshold=", as.character(vshrink)))
}
module(wavelets, unload=T)
```

(c) Finally put your 'best' bandwidth and threshold in place of `???` in the following commands:

```
> par(mfrow=c(2,2))
> kernel.best.estimate <- ksmooth(x, ynoise, bandwidth=???)
> plot(ytrue, type="l", ylim=c(-6,16))
> lines(kernel.best.estimate$y, lty=2)
> title("Kernel")
> plot((ytrue-kernel.best.estimate$y)^2, type="l", ylim=c(0,25),
       ylab="errors")
> ME <- round(mean((ytrue-kernel.best.estimate$y)^2), 2)
> title(paste("Residuals. ME=", as.character(ME)))

> module(wavelets)
> wavelet.estimate <- waveshrink(ynoise, wavelet="haar", vshrink=???)
> module(wavelets, unload=T)
> plot(ytrue, type="l", ylim=c(-6,16))
```

```
> lines(wavelet.estimate, lty=2)
> title("Wavelet")
> plot((ytrue-wavelet.estimate)^2, type="l", ylim=c(0,25), ylab="errors")
> ME <- round(mean((ytrue-wavelet.estimate)^2), 2)
> title(paste("Residuals. ME=", as.character(ME)))
```

The 'Model Error', ME $= \sum_{i=1}^{n}(\hat{f}(x_i) - f(x_i))^2/n$, was calculated for both estimates. Based on that measure of goodness-of-fit and on the two left residuals plots, which estimate gives the best predictive performance?

Chapter 8

S-Plus Functions

In this chapter, the basics of S-Plus functions are described briefly, like using loops and conditional execution, the syntax and the use of the `apply` functions. For an additional and a more detailed account of S-Plus functions, consult books given in Section 1.3.5.

8.1 Loops and Conditional Execution

8.1.1 Grouped Expressions

Commands may be grouped together in braces, { `expression1; expression2; ...` }, in which case the value of the group is the result of the last expression in the group evaluated. Since such a group is also an expression it may, for example, be itself included in parentheses and used a part of an even larger expression, and so on.

8.1.2 Control Statements

Conditional execution: `if` statements

S-Plus has a conditional construction of the form

```
if (expression1) expression2 else expression3
```

where `expression1` must evaluate to a logical value and the result of the entire expression is then evident. For example, suppose that for a vector, `x.vector`, we want to calculate its logarithm by first checking if the vector is larger than zero. This could be done by

```
> if (min(x.vector)>0) log(x.vector) else cat("min(x.vector)<=0.")
```

With `x.vector <- c(-1,0,1)` this would produce the following output:

```
min(x.vector)<=0.
```

Repetitive execution: `for` loops, `repeat` and `while`

There is also a `for` loop construction which has the form

```
for (name in expression1) expression2
```

where `name` is the loop variable. `expression1` is a vector expression, often a sequence like `1:10`, and `expression2` is often a grouped expression with its sub-expressions written in terms of the dummy `name`. `expression2` is repeatedly evaluated as `name` ranges through the values in the vector result of `expression1`.

As an example, suppose that for a vector, `x.vector`, we want to check element by element if its components are all larger than zero. For example, we could use the following code:

```
for (i in 1:length(x.vector))
  {
  if (x.vector[i]>0) cat("Element", i, "is >0.\n")
  else cat("Element", i, "is <=0.\n")
  }
```

With `x.vector <- c(-1,0,1)` this would produce the following output:

```
Element 1 is <=0.
Element 2 is <=0.
Element 3 is >0.
```

Another example of a `for` loop is

```
for (i in 1:length(x.vector)) x.vector <- x.vector + i
```

Note that the use of `for` loops will result in relatively slow evaluation. Therefore, they should be avoided if possible. Many functions, such as `apply`, `tapply`, `sapply` and others, are written primarily to avoid using explicit `for` loops; see Section 8.3. Other looping facilities include the statement

```
repeat expression
```

and the statement

```
while (condition) expression
```

The `break` statement can be used to terminate any loop, possibly abnormally. This is the only way to terminate `repeat` loops. The `next` can be used to discontinue one particular cycle and skip to the 'next'. Control statements are most often used in connection with `functions`, which are discussed in the next section.

8.2 Function Syntax

A function is defined by an assignment of the form

```
name <- function(argument1, argument2, ...)    expression
```

The `expression` is an S-Plus expression (usually a grouped expression) that uses the arguments, `argument1, argument2, ...`, to calculate a value. The value of the expression is the value returned for the function. A call to the function then usually takes the form `name(expression1, expression2, ...)` and may occur anywhere a function call is legitimate. An example is given by the function `mean`:

```
> mean
function(x, trim = 0, na.rm = F){
        if(na.rm) {
                wnas <- which.na(x)
                if(length(wnas))  x <- x[ - wnas]
        }
        if(mode(x) == "complex") {
                if(trim > 0)  stop("trimming not allowed for complex data")
                return(sum(x)/length(x))
        }
        x <- as.double(x)
        if(trim > 0) {
                if(trim >= 0.5) return(median(x, na.rm = F))
                if(!na.rm && length(which.na(x))) return(NA)
                n <- length(x)
                i1 <- floor(trim * n) + 1
                i2 <- n - i1 + 1
                x <- sort(x, unique(c(i1, i2)))[i1:i2]
        }
        sum(x)/length(x)
}
```

Easier examples of S-Plus functions are given by

```
std.dev <- function(x){
                   std.x <- sqrt(var(x))
                   std.x
}

mean.var <- function(x){
                    x.mean <- mean(x)
                    x.var <- var(x)
                    return(x.mean, x.var)
}

sum.squares <- function(x, y = 0){
                       (x + y)^2
}
```

The function `std.dev` returns the standard deviation of a vector `x`, `mean.var` its mean and variance, and `sum.squares` calculates the sum of squares of the sum of `x` and `y`. Note that the default value of `y` is zero. Possible uses and outputs of these three functions are

```
> std.dev(x=1:10)
[1] 3.02765
> std.dev(1:10)
[1] 3.02765

> mean.var(1:10)
$x.mean:
[1] 5.5

$x.var:
[1] 9.166667

> sum.squares(1:4)
[1]  1  4  9 16
> sum.squares(1:4, 5)
[1] 36 49 64 81
```

8.3 The apply Family

The functions `apply`, `lapply`, `sapply` and `tapply` all apply a function to sections of some object. These are used instead of a `for` loop. In general, the appropriate `apply` function is faster than the equivalent `for` loop. Each of the four `apply` functions have an argument named FUN which can be either a function or the name of a function. When you use apply, the MARGIN argument specifies the dimensions that you want to keep. For example,

```
> mymatrix <- matrix(1:10, nrow=2)
> mymatrix
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

> apply(mymatrix, 2, "mean")
[1] 1.5 3.5 5.5 7.5 9.5
```

The result is a vector of length 5, representing the average of each column. The function lapply applies a function to each component of a list, and sapply is used in the same situations as lapply, but generally returns a simplified object. sapply produces a matrix when all of the results are the same length. If the results vary in length, then sapply is the same as lapply. Finally, tapply is used in situations where you have one or more categories, but there is not necessarily an equal number of points in each combination of categories.

Another interesting feature of these apply functions is that optional arguments can be supplied and are passed unchanged (including their names) to each call of FUN. Suppose that there is a NA in mymatrix,

```
> mymatrix <- matrix(c(1:9, NA), nrow=2)
> mymatrix
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   NA
```

Calculating the average of each column as before would result in the following output

```
> apply(mymatrix, 2, "mean")
[1] 1.5 3.5 5.5 7.5  NA
```

If one wants to remove the missing value before computation, one can use

```
> apply(mymatrix, 2, "mean", na.rm=T)
[1] 1.5 3.5 5.5 7.5 9.0
```

where na.rm is an optional argument to the S-PLUS function mean.

8.4 Debugging in S-Plus

When writing your own functions, you may want to have some means of exploring the computations, say if either something is wrong or you are not sure that everything is right. Very good S-PLUS functions for this are `browser`, `debugger`, `trace` and `inspect`; see the corresponding help documentations. Note that the `inspect` interactive debugger is not available in S-PLUS 5.x and there have been several changes to the `browser` and the related function `debugger`; see Section 10.3.

8.5 Exercises

Exercise 8.1. Using the `tapply` function, compute a mean murder rate by region in the `state.x77` data, which contains information about the States of the U.S.

Exercise 8.2. Consider the data `testscores`, which contains test scores for 25 students in each of five subject areas.

(a) Compute the total score for each student. What about the average score? Which student was the best?

(b) Calculate the average score for each of the five subject areas.

Exercise 8.3. Write a function that takes two arguments, `x` and `y`, and returns an object whose elements are the larger of the corresponding elements of the arguments. For example, with $x = (1, 2)$ and $y = (2, 1)$ this function should return the vector $(2, 2)$.

Exercise 8.4. Write a function that takes x- and y-coordinate data and a vector of labels and produces a scatterplot with the labels at the points.

Exercise 8.5. Write a function that takes two vector arguments `x` and `power` and produces `x^power` if `power` is non-zero, but produces `log(x)` if `power` is zero.

Exercise 8.6. Write a recursive function to compute Fibonacci numbers, based on the following facts: `Fib(0)==0`, `Fib(1)==1` and otherwise `Fib(n)` is `Fib(n-1)+Fib(n-2)`. Trace its evaluation in computing `Fib(5)`.

Exercise 8.7. Write a function to compute the factorial of a number. Trace its evaluation in computing 10!, i.e `Fact(10)`.

Chapter 9

Miscellany

9.1 Use a Library

An advantage of S-PLUS is that a lot of recent methodology is available as free libraries that seamlessly extend S-PLUS. To use such a library, type

```
> library(name)
```

which attaches it as a data directory at the end of the search list. Thus libraries cannot over-ride standard functions nor your own functions. To make a library over-ride the system functions, use

```
> library(name, first=T)
```

which attaches the library `name` at position 2 (after the `.Data` directory). An example of such a library is the course library, which is a collection of useful functions and datasets to accompany the introduction course:

```
> library("s+intro")
```

To get the list of its contents, you need the position of the library `s+intro` by using:

```
> search()
[1] ".Data"
[2] "/usr/local/splus/splus/.Functions"
[3] "/usr/local/splus/stat/.Functions"
[4] "/usr/local/splus/s/.Functions"
[5] "/usr/local/splus/s/.Datasets"
[6] "/usr/local/splus/stat/.Datasets"
[7] "/usr/local/splus/splus/.Datasets"
```

```
[8] "/usr/local/splus/library/trellis/.Data"
[9] "/usr/local/splus/library/s+intro/.Data"
```

You note that the `s+intro` library was attached in the ninth position, therefore `ls(pos=9)` provides you with a listing of its contents.

9.2 Load C or Fortran Code

A powerful feature of S-Plus is that it allows you to extend its functionality, enabling you to interface with other languages, namely, C and Fortran. Interfaces to other languages allow you to combine the speed and efficiency of compiled code with the robust, flexible programming environment of S-PLUS; see also Section 10.4 regarding S-Plus 5.x.

9.3 ESS: Emacs Speaks Statistics

ESS is an emacs package which provides a standard interface between statistical programs and statistical processes. It is intended to provide assistance for interactive statistical programming and data analysis, and was is based on and extends the capabilities of S-mode. The code is freely available but is not in the public domain. It is distributed under the GNU General Public License (GNU GPL). With ESS you are able to communicate directly with a running S-Plus process: you can use key sequences to send lines from the S-mode file to the inferior S-Plus process, and you can edit S-Plus input files and transcripts of previous S-Plus sessions. The latest versions of ESS are available from:

`http://ess.stat.wisc.edu`

9.4 R: the 'GNU S'

R, also known as 'GNU S', is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files. R implements a language which is not entirely unlike the S language. Indeed in the absence of a R manual, you can (mostly) get along by using the S manual. R is very similar in appearance to S; see the R FAQ for a discussion of the differences between R and S:

`http://cran.r-project.org/doc/FAQ/R-FAQ.html`

R is available as free software under the terms of Free Software Foundation's GNU GPL. CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for the R statistical package:

`http://cran.r-project.org`

or

`http://www.r-project.org`

Chapter 10

Comparison of S-Plus 5.x with S-Plus 3.4

S-Plus 5.x is the first product that is based on the new version of the S language from Lucent Technologies: version 4 of S. This is a major revision of S designed by John Chambers to improve its usefulness at every stage of the programming process. The following page gives some of the features of Version 4 of S

`http://cm.bell-labs.com/cm/ms/departments/sia/project/S/`

You can keep up to date with the latest in S programming by visiting the 'Programming with Data' web site at

`http://cm.bell-labs.com/stat/Sbook/`

10.1 Backward Compatibility Issues

S-Plus 5.x for UNIX uses a completely reengineered implementation of the S language. Objects created with S-Plus 5.x cannot be used with earlier versions of S-Plus. You should always run S-Plus 5.x in an S-Plus 5.x working directory; this ensures that it contains only objects created in S-Plus 5.x. Do not run S-Plus 5.x using a `.Data` directory created by S-Plus 3.x as your working directory. If you have extensive collections of user-written S-Plus functions, you should run the function `convertOldLibrary` on them before using them with S-Plus 5.x for UNIX. As S-Plus 5.1 includes a new help system (see Section 10.2) you have to convert old-style help files for use with S-Plus 5.1 and later with `convertOldDoc`.

Additional changes include the object-oriented programming model, the functional programming model, the equals sign, '`=`', as an assignment operator. For more details I refer to 'Release Notes: S-Plus 5.x For UNIX'.

10.2 The New Help System

S-Plus 5.1 introduces a new help system to replace the nroff-based system used in earlier versions of S-Plus. The new help system displays help as HTML-formatted text, by default using a modified version of the Lynx terminal-based browser. If you have Netscape Navigator on the machine you use to run S-Plus, you can choose to have help displayed in your Navigator window. The `help.start` function has been restored, but now uses your Netscape Navigator window to display its menu of function and dataset help files. The new help system uses help files written in SGML, which are converted to HTML for display in the browser.

To call up the help system, type

```
> help.start()
```

If you are currently running Netscape Navigator, the S-Plus help menus appear in the running Navigator window. Otherwise, a new Navigator window is launched with the S-Plus help menus. To turn off the help system, type

```
> help.off()
```

At this point, help requests will no longer be sent to the Navigator help window, but the Navigator window will remain open until you close it.

The S-Plus help window contains two frames, which at start-up contain two lists of hyperlinks, the 'Category list' and the 'Function list'. The 'Category list', labeled 'Categories', is on the right side of the Netscape browser. The 'Function list' is on the left side of the browser, and lists all the functions under a selected category or all the functions whose descriptions contain a user-specified 'Topic keyword'. Getting help on a topic can easily obtained through scanning the 'Category list' on the right side of the help window until you find the desired category, selecting the category, and scrolling through the 'Function list' in order to select the function. To print the formatted file, click the 'Print button' on your Navigator toolbar. If none of the categories in the 'Category list' seems to cover what you want, you can use the 'Search option' (at the top of the 'Category list') to provide your own topics.

You can access help easily at the S-Plus prompt with the `?` and `help` functions. By default, both `?` and `help` use the `slynx` browser (a version of the freely available lynx browser) to display the requested help, which is formatted for display using HTML. You can use the ARROW keys to page down and up, respectively; use the '`q`' key to exit help and return to the S-PLUS prompt, or use the ARROW keys to navigate through the hyperlinks. You can specify a different help pager by using, for example,

```
> options(help.pager="vi")
```

Since `vi` is just a text editor, if you use `vi` to view your help files, you will see all the HTML formatting codes.

10.3 Changes in Debugging

The `inspect` interactive debugger is no more available in this new S-PLUS release for UNIX or Linux, and there have been several changes to the `browser` and the related function `debugger`. In addition, a new function, `recover`, can be used to provide interactive debugging as an error action. To use `recover`, set your error action as follows:

```
> options(error=expression(if(interactive()) recover() else dump.calls()))
```

Then, for those type of errors which would normally result in the message

```
Problem in ... Dumped
```

you are instead asked

```
Debug? Y/N
```

If you answer `Y`, you are put into recover s interactive debugger, with a `R>` prompt. Type `?` at the `R>` prompt to see the available commands. Use UP to move up the frame list, DOWN to move down the list. As you move to each frame, recover provides you with a list of local variables. Just type the local variable name to see its current value.

The `browser` function now works much like the `recover` function: you navigate using the UP and DOWN functions, see available commands and local variables with `?`, and exit with `q` (in S-PLUS 3.4 one had to use `0` to exit).

10.4 Migrating C and Fortran Code

'Dynamic Linking Dynamic' loading (that is, the `dyn.load` function) and static loading (the `LOAD` utility) are no longer supported in S-PLUS 5.x. Compiled code is now added to S-PLUS by means of dynamic linking, using the `CHAPTER` mechanism as described in Chambers (1998) or in the S-PLUS 5.x documentation.

Chapter 11

Conclusion

The main aim of this guide was to give a solid but quick introduction to the S-Plus environment. I hope that you are now ready to tackle the really sophisticated functions and to write your own functions. Note that space did not permit to cover recent developments in graphics, like the Trellis library, and to treat additional or new statistical functionality, like bootstrapping, particular statistical models, time series analysis, survival analysis, spatial statistics, neural networks, and so on. So, if a specific problem is not covered in the present guide, you should be able to find the answer in one of the books given in Section 1.3.5; especially in the books by Venables and Ripley (1999, *Modern Applied Statistics with S-Plus*; 2000, *S Programming*); see

`http://www.stats.ox.ac.uk/pub/MASS3/`

or

`http://www.stats.ox.ac.uk/pub/MASS3/Sprog/`

You will find on this page additional exercises, including selected answers, and on-line complements of their book. Moreover, the authors have written several software libraries which enhance S-Plus; these and all the data sets used are available as well. An up-to-date list of texts and books on S and S-Plus is:

`http://www.insightful.com/resources/biblio.html`

If you really should not find what you are searching for, post your query to the S-news mailing list or consult its archive:

`http://www.biostat.wustl.edu/s-news/`

Finally, note that the new version of S-Plus, S-Plus 6 for Solaris and Linux, is in its beta stage. New features in S-Plus 6 include Java-based graphical user interface, and

Java connectivity; multiple imputation library for analysis of data with missing values; and a new robust statistics library.

Bibliography

Baumgartner M. (1994), *Une introduction à S-Plus Version 3.1 pour Silicon Graphics Iris*, Département de Mathématiques, Chaire de Statistique, EPFL, CH-1015 Lausanne.

Becker R. A., Chambers J. M. and Wilks A. R. (1988), *The New S Language*, Chapman and Hall, London.

Böker F. (1997), *S-Plus – Learning by Doing: Eine Anleitung zum Arbeiten mit S-PLUS*, Verlag Lucius & Lucius, Stuttgart.

Bowman A. W. and Azzalini A. (1997), *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations*, Oxford University Press, Oxford.

Chambers J. M. (1998), *Programming with Data*, Springer, New York.

Chambers J. M. and Hastie T. J. (1992), *Statistical Models in S*, Chapman and Hall, London.

Cleveland W. (1993), *Visualizing Data*, Hobart Press, Summit, New Jersey.

Davison A. C. and Hinkley D. V. (1997), *Bootstrap Methods and Their Application*, Cambridge University Press, Cambridge.

Everitt B. (1994), *A Handbook of Statistical Analyses using S-Plus*, Chapman and Hall, London.

Härdle W. (1991), *Smoothing Techniques with Implementation in S*, Springer, New York.

Krause A. (1997), *Einführung in S und S-Plus*, Springer, New York.

Krause A. and Olson M. (1997), *The Basics of S and S-Plus*, Springer, New York.

Marazzi A. (1993), *Algorithms, Routines and S Functions for Robust Statistics*, Chapman and Hall, London.

Spector P. (1994), *An Introduction to S and S-Plus*, Duxbury Press, Belmont, California.

Suselbeck B. (1993), *S und S-Plus: Eine Einführung in Programmierung und Anwendung*, Gustav Fischer Verlag, Jenna.

Venables W. N. and Ripley B. D. (1999), *Modern Applied Statistics with S-Plus*, Third Ed., Springer, New York.

Venables W. N. and Ripley B. D. (2000), *S Programming*, Springer, New York.

List of Figures

List of Tables