Notes on \textbf{S-PLUS}:
A Programming Environment for
Data Analysis and Graphics

Bill Venables and Dave Smith
Department of Statistics
The University of Adelaide

Email: venables@stats.adelaide.edu.au


Version 2.2, September, 1992
Preface

These notes were originally intended only for local consumption at the University of Adelaide, South Australia. After some encouraging comments from students, the author decided to release them to a larger readership in the hope that in some small way they promote good data analysis. S (or S-PLUS) is no panacea, of course, but in my view it represents the single complete environment most conducive to good data analysis so far available.

The authors are indebted to many people for useful contributions, but in particular Lucien W. Van Elen, who did the basic \TeX to \BTeX conversion and Rick Becker who offered an authoritative and extended critique on an earlier version. Others to offer useful comments and corrections include Doug Bates, Ron Baxter, Ray Brownrigg, Sue Clancy, Kathy Haskard, Martin Maechler, James Pearce, Andreas Ruckstuhl and Catarina Savi.

Responsibility for this version, however, remains entirely with the authors, and the notes continue to enjoy a fully unofficial and unencumbered status.

These notes may be freely copied and redistributed provided the copyright notice remains intact. Where appropriate, a small charge to cover the costs of production and distribution, only, may be made.

Comments and corrections are always welcome. Please address email correspondence to the first author at venables@stats.adelaide.edu.au.

Suggestions to the reader

Most S-PLUS novices will start with the introductory session in Appendix A. This should give some familiarity with the style of S-PLUS sessions and more importantly some instant feedback on what actually happens.

Many users will come to S-PLUS mainly for its graphical facilities. In this case section 11 on the graphics facilities can be read at almost any time and need not wait until all the preceding sections have been digested.

Bill Venables and David Smith,
University of Adelaide,
Contents
1 Introduction and Preliminaries

1.1 The S environment

S is an integrated suite of software facilities for data manipulation, calculation and graphical display. Among other things it has

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either at a workstation or on hardcopy, and
- a well developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities. (Indeed most of the system supplied functions are themselves written in the S language.)

The term “environment” is intended to characterize it as a fully planned and coherent system, rather than an incremental accretion of very specific and inflexible tools, as is frequently the case with other data analysis software.

S is very much a vehicle for newly developing methods of interactive data analysis. As such it is very dynamic, and new releases have not always been fully upwardly compatible with previous releases. Some users welcome the changes because of the bonus of new technology and new methods that come with new releases; others seem to be more worried by the fact that old code no longer works. Although S is intended as a programming language, in my view one should regard programs written in S as essentially ephemeral.

The name S as with many names within the UNIX world, is not explained, but left as a cryptic puzzle, and probably a weak pun. However its authors insist it does not stand for “Statistics”!

1.2 The S-PLUS environment

These notes will be mainly concerned with S-PLUS, an enhanced version of S distributed by Statistical Sciences, Inc., Seattle, Washington. Most of what we will have to say, however, applies interchangeably to S and S-PLUS.

1.3 Reference manuals


It is not the intention of these notes to replace these manuals. Rather these notes are intended as a brief introduction to the S-PLUS programming language and a minor amplification of some important points. Ultimately the user of S-PLUS will need to consult this reference manual, probably frequently.

1.4 S-PLUS and X–windows

The most convenient way to use S-PLUS is at a high quality graphics workstation running a windowing system. Since these are becoming more readily available, these notes are aimed at
users who have this facility. In particular we will occasionally refer to the use of S-PLUS on an
X-window system, and even with the motif window manager, although the vast bulk of what is
said applies generally to any implementation of the S-PLUS environment.

Setting up a workstation to take full advantage of the customizable features of S-PLUS is a
straightforward if somewhat tedious procedure, and will not be considered further here. Users in
difficulty should seek local expert help.

1.5 Using S-PLUS interactively

When you use the S-PLUS program it issues a prompt when it expects input commands. The
default prompt is “>”, which is sometimes the same as the shell prompt, and so it may appear
that nothing is happening. However, as we shall see, it is easy to change to a different S-PLUS
prompt if you wish. In these notes we will assume that the shell prompt is “$ ”.

In using S-PLUS the suggested procedure for the first occasion is as follows:

1. Create a separate sub-directory, say work, to hold data files on which you will use S-PLUS
   for this problem. This will be the working directory whenever you use S-PLUS for this
   particular problem.

   $ mkdir work
   $ cd work

2. Place any data files you wish to use with S-PLUS in work.

3. Create a sub-directory of work called .Data for use by S-PLUS.

   $ mkdir .Data

4. Start the S-PLUS program editor with the command

   $ Splus -e

   (The -e flag is optional, but it allows the inbuilt line editor to be used, which is very handy
   for correcting typing errors.)

5. At this point S-PLUS commands may be issued (see later).

6. To quit the S-PLUS program the command is

   > q()
   $

The procedure is simpler using S-PLUS after the first time:

Make work the working directory and start the program as before:

   $ cd work
   $ Splus -e

Use the S-PLUS program, terminating with the q() command at the end of the session.

1.6 An introductory session

Readers wishing to get a feel for S-PLUS at a workstation (or terminal) before proceeding are
strongly advised to work through the model introductory session given in Appendix A, starting
on page 62.
1.7 S-PLUS and UNIX

S-PLUS allows escape to the operating system at any time in the session. If a command, on a new line, begins with an exclamation mark then the rest of the line is interpreted as a UNIX command. So for example to look through a data file without leaving S-PLUS you could use

```
> !more curious.dat
```

When you finish paging the file the S-PLUS session is resumed.

In fact the integration of S-PLUS into UNIX is very complete. For example, there is a command, `unix(...)`, that executes an UNIX command, (specified as a character string argument), and passes on any output from the command as a character string to the program. Essentially the full power of the operating system remains easily available to the user of the S-PLUS program during any session.

The only non-UNIX implementation of S-PLUS so far is one for the DOS operating system. Users should consult the appropriate user guides for more information.

1.8 Getting help with functions and features

S-PLUS has an inbuilt help facility similar to the `man` facility of UNIX. To get more information on any specific named function, for example `solve` the command is

```
> help(solve)
```

An alternative is

```
> ?solve
```

For a feature specified by special characters, the argument must be enclosed in double or single quotes, making it a ‘character string’:

```
> help("["])
```

Either form of quote mark may be used to escape the other, as in the string "It's important". Our convention in these notes is to use double quote marks for preference.

A much more comprehensive help facility is available with the X windows version of S-PLUS. The command

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

causes a "help window" to appear (with the “motif” graphical user interface). It is at this point possible to select items interactively from a series of menus, and the selection process again causes other windows to appear with the help information. This may be either scanned at the screen and dismissed, or sent to a printer for hardcopy, or both.

1.9 S-PLUS commands. Case sensitivity.

Technically S-PLUS is a function language with a very simple syntax. It is case sensitive as are most UNIX based packages, so `a` and `A` are different variables.

Elementary commands consist of either expressions or assignments. If an expression is given as a command, it is evaluated, printed, and the value is lost. An assignment also evaluates an expression and passes the value to the variable but the result is not automatically printed.

Commands are separated either by a semi-colon, `;`, or by a newline. If a command is not complete at the end of a line, S-PLUS will give a different prompt, for example
on second and subsequent lines and continue to read input until the command is syntactically complete. This prompt may also be changed if the user wishes. In these notes we will generally omit the continuation prompt and indicate continuation by simple indenting.

1.10 Recall and correction of previous commands

1.10.1 S-PLUS

S-PLUS (but not plain S) provides a mechanism for recall and correction of previous commands. For interactive use this is a vital facility and greatly increases the productive output of most people. To invoke S-PLUS with the command recall facility enabled use the -e flag:

> Splus -e

Within the session, command recall is available using either emacs-style or vi-style commands. The former is very similar to command recall with an interactive shell such as tcsh. Details are given in Appendix B of these notes, or they may be found in the reference manual or the online help documents.

1.10.2 S

With S no built-in mechanism is available, but there are two common ways of obtaining command recall for interactive sessions.

- Run the S session under emacs using S-mode, a major mode designed to support S. This is probably more convenient than even the inbuilt editor of S-PLUS in the long term, however it does require a good deal of preliminary effort for persons not familiar with the emacs editor. It also often requires a dedicated workstation with a good deal of memory and other resources.

- Run the S session under some front end processor, such as the public domain fep program, available from the public sources archives. This provides essentially the same service as the inbuilt S-PLUS editor, but with somewhat more overhead, (but a great deal less overhead than emacs requires.)

1.11 Executing commands from, or diverting output to, a file

If commands are stored on an external file, say commands.lis in the working directory work, they may be executed at any time in an S-PLUS session with the command

> source("commands.lis")

Similarly

> sink("record.lis")

will divert all subsequent output from the terminal to an external file, record.lis. The command

> sink()

restores it to the terminal once again.

1.12 Data directories. Permanency. Removing objects.

The entities that S-PLUS creates and manipulates are known as objects. These may be variables, arrays of numbers, character strings, functions, or more general structures built from such components.
All objects created during your S-PLUS sessions are stored as files, in a special form, in the .Data sub-directory of your working directory work, say.

Each object is held as a separate file of the same name and so may be manipulated by the usual UNIX commands such as rm, cp and mv. This means that if you resume your S-PLUS session at a later time, objects created in previous sessions are still available, which is a highly convenient feature.

This also explains why it is recommended that you should use separate working directories for different jobs. Common names for objects are single letter names like x, y and so on, and if two problems share the same .Data sub-directory the objects will become mixed up and you may overwrite one with another.

There is, however, another method of partitioning variables within the same .Data directory using data frames. These are discussed further in §6.4.

In S-PLUS, to get a list of names of the objects currently defined use the command

\> objects()

whose result is a vector of character strings giving the names.

When S-PLUS looks for an object, it searches in turn through a sequence of places known as the search list. Usually the first entry in the search list is the .Data sub-directory of the current working directory. The names of the places currently on the search list are displayed by the function

\> search()

The names of the objects held in any place in the search list can be displayed by giving the objects() function an argument. For example

\> objects(2)

lists the contents of the entity at position 2 of the search list. The search list can contain either data frames and allies, which are themselves internal S-PLUS objects, as well as directories of files which are UNIX objects.

Extra entities can be added to this list with the attach() function and removed with the detach() function, details of which can be found in the manual or the help facility.

To remove objects permanently the function rm is available:

\> rm(x, y, z, ink, junk, temp, foo, bar)

The function remove() can be used to remove objects with non-standard names. Also the ordinary UNIX facility, rm, may be used to remove the appropriate files in the .Data directory, as mentioned above.
2 Simple manipulations; numbers and vectors

2.1 Vectors and Assignments

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. To set up a vector named \( x \), say, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7, use the S-PLUS command

\[
> x \leftarrow c(10.4, 5.6, 3.1, 6.4, 21.7)
\]

This is an assignment statement using the function \( c() \) which in this context can take an arbitrary number of vector arguments and whose value is a vector got by concatenating its arguments end to end.\(^1\)

A number occurring by itself in an expression is taken as a vector of length one.

Notice that the assignment operator is not the usual = operator, which is reserved for another purpose. It consists of the two characters < ('less than') and - ('minus') occurring strictly side-by-side and it 'points' to the object receiving the value of the expression.\(^2\)

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

\[
> c(10.4, 5.6, 3.1, 6.4, 21.7) \rightarrow x
\]

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

\[
> 1/x
\]

the reciprocals of the five values would be printed at the terminal (and the value of \( x \), of course, unchanged).

The further assignment

\[
> y \leftarrow c(x, 0, x)
\]

would create a vector \( y \) with 11 entries consisting of two copies of \( x \) with a zero in the middle place.

2.2 Vector arithmetic

Vectors can be used in arithmetic expressions, in which case the operations are performed element by element. 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 need be (perhaps fractionally) until they match the length of the longest vector. In particular a constant is simply repeated. So with the above assignments the command

\[
> v \leftarrow 2*x + y + 1
\]

generates a new vector \( v \) of length 11 constructed by adding together, element by element, \( 2x \) repeated 2.2 times, \( y \) repeated just once, and 1 repeated 11 times.

The elementary arithmetic operators are the usual +, −, ×, ÷ and ^ for raising to a power. In addition all of the common arithmetic functions are available. \( \log, \exp, \sin, \cos, \tan, \sqrt{\text{sqrt}} \), and so on, all have their usual meaning. \( \max \) and \( \min \) select the largest and smallest elements of an vector respectively. \( \text{range} \) is a function whose value is a vector of length two, namely \( c(\min(x), \max(x)) \).

\(^1\)With other than vector types of argument, such as list mode arguments, the action of \( c() \) is rather different. See [6.2.1].

\(^2\)The underscore character, "_", is an allowable synonym for the left pointing assignment operator "<", however we discourage this option, as it can easily lead to much less readable code.
max(x). length(x) is the number of elements in x, sum(x) gives the total of the elements in x and prod(x) their product.

Two statistical functions are mean(x) which calculates the sample mean, which is the same as sum(x)/length(x), and var(x) which gives

\[
\text{var}(x) = \frac{\text{sum}((x - \text{mean}(x))^2)}{\text{length}(x) - 1}
\]

or sample variance. If the argument to var() is a matrix the value is a sample covariance matrix got by regarding the rows as independent vectors produced by doing the sort.

sort(x) returns a vector of the same size as x with the values arranged in increasing order; however there are other more flexible sorting facilities available (see order() or sort.list() which produce a permutation to do the sorting).

rnorm(x) is a function which generates a vector (or more generally an array) of pseudo-random standard normal deviates, of the same size as x.

### 2.3 Generating regular sequences

S-PLUS has a number of facilities for generating commonly used sequences of numbers. For example 1:30 is the vector c(1, 2, ..., 29, 30). The colon operator has highest priority within an expression, so, for example 2*1:15 is the vector c(2, 4, 6, ..., 28, 30). Put n <- 10 and compare the sequences 1:n-1 and 1:(n-1).

The construction 30:1 may be used to generate a sequence backwards.

The function seq() is a more general facility for generating sequences. It has five arguments, only some of which may be specified in any one call. 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. That is seq(2,10) is the same vector as 2:10.

Parameters to seq(), and to many other S-PLUS functions, can also be given in named form, in which case the order in which they appear is irrelevant. The first two parameters may be named from=value and to=value; thus seq(1,30), seq(from=1, to=30) and seq(to=30, from=1) are all the same as 1:30. The next two parameters to seq() may be named by=value and length=value, which specify a step size and a length for the sequence respectively. If neither of these is given, the default by=1 is assumed.

For example

```r
> seq(-5, 5, by=.2) -> s3
```

generates in s3 the vector c(-5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0). Similarly

```r
> s4 <- seq(length=51, from=-5, by=.2)
```

generates the same vector in s4.

The fifth parameter may be named along=vector, which if used must be the only parameter, and creates a sequence 1, 2, ..., length(vector), or the empty sequence if the vector is empty (as it can be).

A related function is rep() which can be used for replicating an object in various complicated ways. The simplest form is

```r
> s5 <- rep(x, times=5)
```

which will put five copies of x end-to-end in s5.
2.4 Logical vectors

As well as numerical vectors, S-PLUS allows manipulation of logical quantities. The elements of a logical vector have just two possible values, represented formally as \( \text{F} \) (for ‘false’) and \( \text{T} \) (for ‘true’).

Logical vectors are generated by conditions. For example

\[
> \text{temp} \leftarrow x > 13
\]

sets \( \text{temp} \) as a vector of the same length as \( x \) with values \( \text{F} \) corresponding to elements of \( x \) where the condition is \text{not met} and \( \text{T} \) where it is.

The logical operators are \(<,\leq,\geq,\neq\), \(\text{==}\) for exact equality and \(\neq\) for inequality. In addition if \( c_1 \) and \( c_2 \) are logical expressions, then \( c_1 \& c_2 \) is their intersection, \( c_1 \lor c_2 \) is their union and \( \neg c_1 \) is the negation of \( c_1 \).

Logical vectors may be used in ordinary arithmetic, in which case they are coerced into numeric vectors, \( \text{F} \) becoming 0 and \( \text{T} \) becoming 1. However there are situations where logical vectors and their coerced numeric counterparts are not equivalent, for example see the next subsection.

2.5 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 \( \text{NA} \). In general any operation on an \( \text{NA} \) becomes an \( \text{NA} \). 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.

The function \( \text{is.na(x)} \) gives a logical vector of the same size as \( x \) with value \( \text{T} \) if and only if the corresponding element in \( x \) is \( \text{NA} \).

\[
> \text{ind} \leftarrow \text{is.na(z)}
\]

Notice that the logical expression \( x == \text{NA} \) is quite different from \( \text{is.na(x)} \) since \( \text{NA} \) is not really a value but a marker for a quantity that is not available. Thus \( x == \text{NA} \) is a vector of the same length as \( x \) \text{all} of whose values are \( \text{NA} \) as the logical expression itself is incomplete and hence undecidable.

2.6 Character vectors

Character quantities and character vectors are used frequently in S-PLUS, for example as plot labels. Where needed they are denoted by a sequence of characters delimited by the double quote character. E.g. "x-values", "New iteration results".

Character vectors may be concatenated into a vector by the \( \text{c()} \) function; examples of their use will emerge frequently.

The \( \text{paste()} \) function takes an arbitrary number of arguments and concatenates them into a single character string. Any numbers given among the arguments are coerced into character strings in the evident way, that is, in the same way they would be if they were printed. The arguments are by default separated in the result by a single blank character, but this can be changed by the named parameter, \( \text{sep=string} \), which changes it to \( \text{string} \), possibly empty.

For example

\[
> \text{labs} \leftarrow \text{paste(c("X","Y"), 1:10, sep=""})
\]

makes \( \text{labs} \) into the character vector

\[ ("X1", "Y2", "X3", "Y4", "X5", "Y6", "X7", "Y8", "X9", "Y10") \]
2.7 Index vectors. Selecting and modifying subsets of a data set

Note particularly that recycling of short lists takes place here too; thus \( c("x", "y") \) is repeated 5 times to match the sequence 1:10.

2.7 Index vectors. Selecting and modifying subsets of a data set

Subsets of the elements of a vector may be selected by appending to the name of the vector an index vector in square brackets. More generally any expression that evaluates to a vector may have subsets of its elements similarly selected by appending an index vector in square brackets immediately after the expression.

Such index vectors can be any of four distinct types.

1. A logical vector. In this case the index vector 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

\[
y \leftarrow x[!is.na(x)]
\]

creates (or re-creates) an object \( y \) which will contain the non-missing values of \( x \), in the same order. Note that if \( x \) has missing values, \( y \) will be shorter than \( x \). Also

\[
(x+1)[(!is.na(x)) & x>0] \rightarrow z
\]

creates an object \( z \) and places in it the values of the vector \( x+1 \) for which the corresponding value in \( x \) was both non-missing and positive.

2. A vector of positive integral quantities. In this case the values in the index vector must lie in the set \(-1, 2, \ldots, \text{length}(x)\)’. The corresponding elements of the vector are selected and concatenated, in that order, in the result. The index vector can be of any length and the result is of the same length as the index vector. For example \( x[6] \) is the sixth component of \( x \) and

\[
x[1:10]
\]

selects the first 10 elements of \( x \), (assuming \( \text{length}(x) \geq 10 \)). Also

\[
c("x","y") \leftarrow \text{rep}(c(1,2,2,1), \text{times}=4)
\]

(an admittedly unlikely thing to do) produces a character vector of length 16 consisting of "x", "y", "y", "x" repeated four times.

3. A vector of negative integral quantities. Such an index vector specifies the values to be excluded rather than included. Thus

\[
y \leftarrow x[-(1:5)]
\]

gives \( y \) all but the first five elements of \( x \).

4. A vector of character strings. This possibility only applies where an object has a names attribute to identify its components. In this case a subvector of the names vector may be used in the same way as the positive integral labels in 2. above.

\[
\text{fruit} \leftarrow c(5, 10, 1, 20)
\]
\[
\text{names(fruit)} \leftarrow c(\text{"orange", "banana", "apple", "peach"})
\]
\[
\text{lunch} \leftarrow \text{fruit}[c(\text{"apple","orange")}]
\]

The advantage is that alphanumeric names are often easier to remember than numeric indices. This option is particularly useful in connection with data frames, as we shall see later.

An indexed expression can also appear on the receiving end of an assignment, in which case the assignment operation is performed \textit{only on those elements of the vector}. The expression must be
of the form `vector[index_vector]` as having an arbitrary expression in place of the vector name does not make much sense here.

The vector assigned must match the length of the index vector, and in the case of a logical index vector it must again be the same length as the vector it is indexing.

For example

```r
> x[is.na(x)] <- 0
```

replaces any missing values in `x` by zeros and

```r
> y[y<0] <- -y[y<0]
```

has the same effect as

```r
> y <- abs(y)^3
```

---

\(^3\)Note that `abs()` does not work as expected with complex arguments. The appropriate function for the complex modulus is `Mod()`.
3 Objects, their modes and attributes

3.1 Intrinsic attributes: mode and length

The entities S-PLUS operates on are technically known as objects. Examples are vectors of numeric (real) or complex values, vectors of logical values and vectors of character strings. These are known as ‘atomic’ structures since their components are all of the same type, or mode, namely numeric, complex, logical and character respectively.

Vectors must have their values all of the same mode. Thus any given vector must be unambiguously either logical, numeric, complex or character. The only mild exception to this rule is the special “value” listed as NA for quantities not available. Note that a vector can be empty and still have a mode. For example the empty character string vector is listed as character(0) and the empty numeric vector as numeric(0).

S-PLUS also operates on objects called lists, which are of mode list. These are ordered sequences of objects which individually can be of any mode. lists are known as ‘recursive’ rather than atomic structures since their components can themselves be lists in their own right.

The other recursive structures are those of mode function and expression. Functions are the functions that form part of the S-PLUS system along with similar user written functions, which we discuss in some detail later in these notes. Expressions as objects form an advanced part of S-PLUS which will not be discussed in these notes, except indirectly when we discuss formulae used with modelling in S-PLUS.

By the mode of an object we mean the basic type of its fundamental constituents. This is a special case of an attribute of an object. The attributes of an object provide specific information about the object itself. Another attribute of every object is its length. The functions mode(object) and length(object) can be used to find out the mode and length of any defined structure.

For example, if z is a complex vector of length 100, then in an expression mode(z) is the character string "complex" and length(z) is 100.

S-PLUS caters for changes of mode almost anywhere it could be considered sensible to do so, (and a few where it might not be). For example with

> z <- 0:9

we could put

> digits <- as.character(z)

after which digits is the character vector ("0", "1", "2", ..., "9"). A further coercion, or change of mode, reconstructs the numerical vector again:

> d <- as.numeric(digits)

Now d and z are the same.\(^5\) There is a large collection of functions of the form as.something() for either coercion from one mode to another, or for investing an object with some other attribute it may not already possess. The reader should consult the help file to become familiar with them.

3.2 Changing the length of an object

An “empty” object may still have a mode. For example

> e <- numeric()

\(^4\)numeric mode is actually an amalgam of three distinct modes, namely integer, single precision and double precision, as explained in the manual.

\(^5\)In general coercion from numeric to character and back again will not be exactly reversible, because of roundoff errors in the character representation.
makes e an empty vector structure of mode numeric. Similarly character() is a empty character vector, and so on. Once an object of any size has been created, new components may be added to it simply by giving it an index value outside its previous range. Thus

> e[3] <- 17

now makes e a vector of length 3, (the first two components of which are at this point both NA). This applies to any structure at all, provided the mode of the additional component(s) agrees with the mode of the object in the first place.

This automatic adjustment of lengths of an object is used often, for example in the scan() function for input. (See §7.2.) Conversely to truncate the size of an object requires only an assignment to do so. Hence if alpha is an object of length 10, then

> alpha <- alpha[2 * 1:5]

makes it an object of length 5 consisting of just the former components with even index. The old indices are not retained, of course.

3.3 attributes() and attr()

The function attributes(object) gives a list of all the non-intrinsic attributes currently defined for that object. The function attr(object, name) can be used to select a specific attribute. These functions are rarely used, except in rather special circumstances when some new attribute is being created for some particular purpose, for example to associate a creation date or an operator with an S-PLUS object. The concept, however, is very important.

3.4 The class of an object

A special attribute known as the class of the object has been introduced in the August 1991 release of S and S-PLUS to allow for an object oriented style of programming in S-PLUS.

For example if an object has class data.frame, it will be printed in a certain way, the plot() function will display it graphically in a certain way, and other generic functions such as summary() will react to it as an argument in a way sensitive to its class.

To remove temporarily the effects of class, use the function unclass(). For example if winter has the class data.frame then

> winter

will print it in data frame form, which is rather like a matrix, whereas

> unclass(winter)

will print it as an ordinary list. Only in rather special situations do you need to use this facility, but one is when you are learning to come to terms with the idea of class and generic functions.

Generic functions and classes will be discussed further in §9.7, but only briefly.
4 Categories and factors

A category is a vector object used to specify a discrete classification of the components of other vectors of the same length. A factor is similar, but has the class factor, which means that it is adapted to the generic function mechanism. Whereas a category can also be used as a plain numeric vector, for example, a factor generally cannot.

4.1 A specific example

Suppose, for example, we have a sample of 30 tax accountants from the all states and territories\(^6\) and their individual state of origin is specified by a character vector of state mnemonics as

\[
\]

For some purposes it is convenient to represent such information by a numeric vector with the distinct values in the original (in this case the state labels) represented by a small integer. Such a numeric vector is called a category. However at the same time it is important to preserve the correspondence of these new integer labels with the originals. This is done via the levels attribute of the category.

More formally, when a category is formed from such a vector the sorted unique values in the vector form the levels attribute of the category, and the values in the category are in the set 1, 2, ..., k where k is the number of unique values. The value at position j in the factor is i if the ith sorted unique value occurred at position j of the original vector.

Hence the assignment

\[
\text{> stcode <- category(state)}
\]

creates a category with values and attributes as follows

\[
\text{> stcode}
\]

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

attr(, "levels"):

[1] "act" "nsw" "nt" "qld" "sa" "tas" "vic" "wa"

Notice that in the case of a character vector, “sorted” means sorted in alphabetical order.

A factor is similarly created using the factor() function:

\[
\text{> statef <- factor(state)}
\]

The print() function now handles the factor object slightly differently:

\[
\text{> statef}
\]

[1] tas sa qld nsw nt wa wa qld vic nsw vic qld qld sa

[16] tas sa nt wa vic qld nsw wa sa act nsw vic vic act

If we remove the factor class, however, using the function unclass(), it becomes virtually identical to the category:

\[
\text{> unclass(statef)}
\]

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

attr(, "levels"):

[1] "act" "nsw" "nt" "qld" "sa" "tas" "vic" "wa"

\(^6\)Foreign readers should note that there are eight states and territories in Australia, namely the Australian Capital Territory, New South Wales, the Northern Territory, Queensland, South Australia, Tasmania, Victoria and Western Australia.
4.2 The function `tapply()` and ragged arrays

To continue the previous example, suppose we have the incomes of the same tax accountants in another vector (in suitably large units of money)

```r
> incomes <- c(60, 49, 40, 61, 64, 60, 59, 54, 62, 69, 70, 42, 56,
              61, 61, 61, 58, 51, 48, 65, 49, 49, 41, 48, 52, 46,
              59, 46, 58, 43)
```

To calculate the sample mean income for each state we can now use the special function `tapply()`:

```r
> incmeans <- tapply(incomes, statef, mean)
```

giving a means vector with the components labelled by the levels

```r
> incmeans
  act    nsw    nt    qld    sa    tas    vic    wa
 44.5  57.333  55.5  53.6  60.5   56   52.25
```

The function `tapply()` is used to apply a function, here `mean()`, to each group of components of the first argument, here `incomes`, defined by the levels of the second component, here `statef`, as if they were separate vector structures. The result is a structure of the same length as the levels attribute of the factor containing the results. The reader should consult the help document for more details.

Suppose further we needed to calculate the standard errors of the state income means. To do this we need to write an S-PLUS function to calculate the standard error for any given vector. We discuss functions more fully later in these notes, but since there is an inbuilt function `var()` to calculate the sample variance, such a function is a very simple one liner, specified by the assignment:

```r
> stderr <- function(x) sqrt(var(x)/length(x))
```

(Writing functions will be considered later in §9.) After this assignment, the standard errors are calculated by

```r
> incster <- tapply(incomes, statef, stderr)
```

and the values calculated are then

```r
> incster
  act    nsw    nt    qld    sa    tas    vic    wa
   1.5  4.3102  4.5  4.1061  2.7386  0.5  5.244  2.6575
```

As an exercise you may care to find the usual 95% confidence limits for the state mean incomes. To do this you could use `tapply()` once more with the `length()` function to find the sample sizes, and the `qt()` function to find the percentage points of the appropriate $t$-distributions.

The function `tapply()` can be used to handle more complicated indexing of a vector by multiple categories. For example, we might wish to split the tax accountants by both state and sex. However in this simple instance what happens can be thought of as follows. The values in the vector are collected into groups corresponding to the distinct entries in the category. The function is then applied to each of these groups individually. The value is a vector of function results, labelled by the levels attribute of the category.

The combination of a vector and a labelling factor or category is an example of what is called a **ragged array**, since the subclass sizes are possibly irregular. When the subclass sizes are all the same the indexing may be done implicitly and much more efficiently, as we see in the next section.
5 Arrays and matrices

5.1 Arrays

An array can be considered as a multiply subscripted collection of data entries, for example numeric. S-PLUS allows simple facilities for creating and handling arrays, and in particular the special case of matrices.

A dimension vector is a vector of positive integers. If its length is \( k \) then the array is \( k \)-dimensional. The values in the dimension vector give the upper limits for each of the \( k \) subscripts. The lower limits are always 1.

A vector can be used by S-PLUS as an array only if it has a dimension vector as its \( \text{dim} \) attribute. Suppose, for example, \( z \) is a vector of 1500 elements. The assignment

\[
g > \text{dim}(z) \leftarrow c(3,5,100)
\]

gives it the \( \text{dim} \) attribute that allows it to be treated as a \( 3 \times 5 \times 100 \) array.

Other functions such as \( \text{matrix()} \) and \( \text{array()} \) are available for simpler and more natural looking assignments, as we shall see in §5.4.

The values in the data vector give the values in the array in the same order as they would occur in Fortran, that is ‘column major order’, with the first subscript moving fastest and the last subscript slowest.

For example if the dimension vector for an array, say \( a \) is \( c(3,4,2) \) then there are \( 3 \times 4 \times 2 = 24 \) entries in \( a \) and the data vector holds them in the order \( a[1,1,1], a[2,1,1], \ldots, a[2,4,2], a[3,4,2] \).

5.2 Array indexing. Subsections of an array

Individual elements of an array may be referenced, as above, by giving the name of the array followed by the subscripts in square brackets, separated by commas.

More generally, subsections of an array may be specified by giving a sequence of index vectors in place of subscripts; however if any index position is given an empty index vector, then the full range of that subscript is taken.

Continuing the previous example, \( a[2,\ldots] \) is a \( 4 \times 2 \) array with dimension vector \( c(4,2) \) and data vector containing the values

\[
a[2,1,1], a[2,2,1], a[2,3,1], a[2,4,1], a[2,1,2], a[2,2,2], a[2,3,2], a[2,4,2],
\]
in that order. \( a[\ldots] \) stands for the entire array, which is the same as omitting the subscripts entirely and using \( a \) alone.

For any array, say \( Z \), the dimension vector may be referenced explicitly as \( \text{dim}(Z) \) (on either side of an assignment).

Also, if an array name is given with just one subscript or index vector, then the corresponding values of the data vector only are used; in this case the dimension vector is ignored. This is not the case, however, if the single index is not a vector but itself an array, as we next discuss.

5.3 Index arrays

As well as an index vector in any subscript position, an array may be used with a single index array in order either to assign a vector of quantities to an irregular collection of elements in the array, or to extract an irregular collection as a vector.
A matrix example makes the process clear. In the case of a doubly indexed array, an index matrix may be given consisting of two columns and as many rows as desired. The entries in the index matrix are the row and column indices for the doubly indexed array. Suppose for example we have a \( 4 \times 5 \) array \( X \) and we wish to do the following:

- Extract elements \( X[1,3], X[2,2] \) and \( X[3,1] \) as a vector structure, and
- Replace these entries in the array \( X \) by 0s.

In this case we need a \( 3 \times 2 \) subscript array, as in the example given in Figure 1.

```r
> x <- array(1:20, dim=c(4,5))  # Generate a 4 x 5 array.
> x
[1,]  1  5  9 13 17
[2,]  2  6 10 14 18
[3,]  3  7 11 15 19
[4,]  4  8 12 16 20
> i <- array(c(1:3,3:1),dim=c(3,2))
> i
   [,1] [,2]
[1,]  1  3
[2,]  2  2
[3,]  3  1
> x[i]  # Extract those elements
[1]  9  6  3
> x[i] <- 0  # Replace those elements by zeros.
> x
[1,]  1  5  0 13 17
[2,]  2  0 10 14 18
[3,]  0  7 11 15 19
[4,]  4  8 12 16 20
```

Figure 1: Using an index array

As a less trivial example, suppose we wish to generate an (unreduced) design matrix for a block design defined by factors `blocks` (b levels) and `varieties`, (v levels). Further suppose there are \( n \) plots in the experiment. We could proceed as follows:

```r
> Xb <- matrix(0, n, b)
> Xv <- matrix(0, n, v)
> ib <- cbind(1:n, blocks)
> iv <- cbind(1:n, varieties)
> Xb[ib] <- 1
> Xv[iv] <- 1
> X <- cbind(Xb, Xv)
```

Further, to construct the incidence matrix, \( N \) say, we could use

```r
> N <- crossprod(Xb, Xv)
```

However a simpler direct way of producing this matrix is to use `table()`:

```r
> N <- table(blocks, varieties)
```
5.4 The `array()` function

As well as giving a vector structure a `dim` attribute, arrays can be constructed from vectors by the `array` function, which has the form

\[
Z \leftarrow \text{array}(\text{data\.vector}, \text{dim\.vector})
\]

For example, if the vector `h` contains 24, or fewer, numbers then the command

\[
Z \leftarrow \text{array}(h, \text{dim}=c(3,4,2))
\]

would use `h` to set up \(3 \times 4 \times 2\) array in `Z`. If the size of `h` is exactly 24 the result is the same as

\[
\text{dim}(Z) \leftarrow c(3,4,2)
\]

However if `h` is shorter than 24, its values recycled from the beginning again to make it up to size 24. See §5.4.1 below. As an extreme but common example

\[
Z \leftarrow \text{array}(0, c(3,4,2))
\]

makes `Z` an array of all zeros.

At this point `dim(Z)` stands for the dimension vector `c(3,4,2)`, and `Z[1:24]` stands for the data vector as it was in `h`, and `Z[0]` with an empty subscript or `Z` with no subscript stands for the entire array as an array.

Arrays may be used in arithmetic expressions and the result is an array formed by element by element operations on the data vector. The `dim` attributes of operands generally need to be the same, and this becomes the dimension vector of the result. So if `A`, `B` and `C` are all similar arrays, then

\[
D \leftarrow 2*A*B + C + 1
\]

makes `D` a similar array with data vector the result of the evident element by element operations. However the precise rule concerning mixed array and vector calculations has to be considered a little more carefully.

5.4.1 Mixed vector and array arithmetic. The recycling rule

The precise rule affecting element by element mixed calculations with vectors and arrays is somewhat quirky and hard to find in the references. From experience I have found the following to be a reliable guide.

- The expression is scanned from left to right.
- Any short vector operands are extended by recycling their values until they match the size of any previous (or subsequent) operands.
- As long as short vectors and arrays, only, are encountered, the arrays must all have the same `dim` attribute or an error results.
- Any vector operand longer than some previous array immediately converts the calculation to one in which all operands are coerced to vectors. A diagnostic message is issued if the size of the long vector is not a multiple of the (common) size of all previous arrays.
- If array structures are present and no error or coercion to vector has been precipitated, the result is an array structure with the common `dim` attribute of its array operands.

5.5 The outer product of two arrays

An important operation on arrays is the outer product. If \(a\) and \(b\) are two numeric arrays, their outer product is an array whose dimension vector is got by concatenating their two dimension
vectors, (order is important), and whose data vector is got by forming all possible products of elements of the data vector of a with those of b. The outer product is formed by the special operator %o%:

\[
> \text{ab} \leftarrow \text{a} \ %o\% \ \text{b}
\]

An alternative is:

\[
> \text{ab} \leftarrow \text{outer(a, b, '*)}
\]

The multiplication function can be replaced by an arbitrary function of two variables. For example if we wished to evaluate the function

\[
f(x, y) = \frac{\cos(y)}{1 + x^2}
\]

over a regular grid of values with \(x\)– and \(y\)–coordinates defined by the S-PLUS vectors \(x\) and \(y\) respectively, we could proceed as follows:

\[
> \text{f} \leftarrow \text{function(x, y) cos(y)/(1 + x^2)}
\]

\[
> \text{z} \leftarrow \text{outer(x, y, f)}
\]

In particular the outer product of two ordinary vectors is a doubly subscripted array (that is a matrix, of rank at most 1). Notice that the outer product operator is of course non-commutative. Defining your own S-PLUS functions will be considered further in Chapter 9.

5.5.1 An example: Determinants of \(2 \times 2\) digit matrices

As an artificial but cute example, consider the determinants of \(2 \times 2\) matrices

\[
\begin{bmatrix}
 a & b \\
 c & d
\end{bmatrix}
\]

where each entry is a non-negative integer in the range 0, 1, \ldots, 9, that is a digit.

The problem is to find the determinants, \(ad-\text{bc}\), of all possible matrices of this form and represent the frequency with which each value occurs as a high density plot. This amounts to finding the probability distribution of the determinant if each digit is chosen independently and uniformly at random.

A neat way of doing this uses the \texttt{outer()} function twice:

\[
> \text{d} \leftarrow \text{outer(0:9, 0:9)}
\]

\[
> \text{fr} \leftarrow \text{table(outer(d, d, "-"))}
\]

\[
> \text{plot(as.numeric(names(fr)), fr, type="h",}
\]

\[
\quad \text{xlab="Determinant", ylab="Frequency"})
\]

Notice the coercion of the \texttt{names} attribute of the frequency table to numeric in order to recover the range of the determinant values. The “obvious” way of doing this problem with \texttt{for}-loops, to be discussed in §8.2, is so inefficient as to be impractical.

It is also perhaps surprising that about 1 in 20 such matrices is singular.

5.6 Generalized transpose of an array

The function \texttt{aperm(a, perm)} may be used to permute an array, \texttt{a}. The argument \texttt{perm} must be a permutation of the integers \(-1, 2, \ldots, k\), where \(k\) is the number of subscripts in \texttt{a}. The result of the function is an array of the same size as \texttt{a} but with old dimension given by \texttt{perm[j]} becoming the new \(j\)th dimension. The easiest way to think of this operation is as a generalization of transposition for matrices. Indeed if \(A\) is a matrix, (that is, a doubly subscripted array) then \(B\) given by

\[
> \text{B} \leftarrow \text{aperm(A, c(2,1))}
\]
is just the transpose of \( A \). For this special case a simpler function \( t() \) is available, so we could have used \( B \leftarrow t(A) \).

### 5.7 Matrix facilities. Multiplication, inversion and solving linear equations.

As noted above, a matrix is just an array with two subscripts. However it is such an important special case it needs a separate discussion. S-PLUS contains many operators and functions that are available only for matrices. For example \( t(X) \) is the matrix transpose function, as noted above. The functions \( \text{nrow}(A) \) and \( \text{ncol}(A) \) give the number of rows and columns in the matrix \( A \) respectively.

The operator \( \%*\% \) is used for matrix multiplication. An \( n \times 1 \) or \( 1 \times n \) matrix may of course be used as an \( n \)-vector if in the context such is appropriate. Conversely vectors which occur in matrix multiplication expressions are automatically promoted either to row or column vectors, whichever is multiplicatively coherent, if possible, (although this is not always unambiguously possible, as we see later).

If, for example, \( A \) and \( B \) are square matrices of the same size, then

\[
> A * B
\]

is the matrix of element by element products and

\[
> A \%*\% B
\]

is the matrix product. If \( x \) is a vector, then

\[
> x \%*\% A \%*\% x
\]

is a quadratic form.\(^7\)

The function \( \text{crossprod()} \) forms “crossproducts”, meaning that

\[
> \text{crossprod}(X, y) \text{ is the same as } t(X) \%*\% y
\]

but the operation is more efficient. If the second argument to \( \text{crossprod()} \) is omitted it is taken to be the same as the first.

Other important matrix functions include \( \text{solve}(A, b) \) for solving equations, \( \text{solve}(A) \) for the matrix inverse, \( \text{svd()} \) for the singular value decomposition, \( \text{qr()} \) for QR decomposition and \( \text{eigen()} \) for eigenvalues and eigenvectors of symmetric matrices.

The meaning of \( \text{diag()} \) depends on its argument. \( \text{diag}(\text{vector}) \) gives a diagonal matrix with elements of the vector as the diagonal entries. On the other hand \( \text{diag}(\text{matrix}) \) gives the vector of main diagonal entries of \( \text{matrix} \). This is the same convention as that used for \( \text{diag()} \) in MATLAB. Also, somewhat confusingly, if \( k \) is a single numeric value then \( \text{diag}(k) \) is the \( k \times k \) identity matrix!

A surprising omission from the suite of matrix facilities is a function for the determinant of a square matrix, however the absolute value of the determinant is easy to calculate for example as the product of the singular values. (See later.)

### 5.8 Forming partitioned matrices. \( \text{cbind()} \) and \( \text{rbind()} \).

As we have already seen informally, matrices can be built up from other vectors and matrices by the functions \( \text{cbind()} \) and \( \text{rbind()} \). Roughly \( \text{cbind()} \) forms matrices by binding together

\(^7\)Note that \( x \%*\% x \) is ambiguous, as it could mean either \( x'x \) or \( xx' \), where \( x \) is the column form. In such cases the smaller matrix seems implicitly to be the interpretation adopted, so the scalar \( x'x \) is in this case the result. The matrix \( xx' \) may be calculated either by \( \text{cbind}(x) \%*\% x \) or \( x \%*\% \text{rbind}(x) \) since the result of \( \text{rbind()} \) or \( \text{cbind()} \) is always a matrix.
matrices horizontally, or column-wise, and `rbind()` vertically, or row-wise.

In the assignment

```r
> X <- cbind(arg1, arg2, arg3, ...)
```

the arguments to `cbind()` must be either vectors of any length, or matrices with the same column size, that is the same number of rows. The result is a matrix with the concatenated arguments `arg1, arg2, ...` forming the columns.

If some of the arguments to `cbind()` are vectors they may be shorter than the column size of any matrices present, in which case they are cyclically extended to match the matrix column size (or the length of the longest vector if no matrices are given).

The function `rbind()` does the corresponding operation for rows. In this case any vector argument, possibly cyclically extended, are of course taken as row vectors.

Suppose `X1` and `X2` have the same number of rows. To combine these by columns into a matrix `X`, together with an initial column of 1s we can use

```r
> X <- cbind(1, X1, X2)
```

The result of `rbind()` or `cbind()` always has matrix status. Hence `cbind(x)` and `rbind(x)` are possibly the simplest ways explicitly to allow the vector `x` to be treated as a column or row matrix respectively.

### 5.9 The concatenation function, `c()`, with arrays.

It should be noted that whereas `cbind()` and `rbind()` are concatenation functions that respect `dim` attributes, the basic `c()` function does not, but rather clears numeric objects of all `dim` and `dimnames` attributes. This is occasionally useful in its own right.

The official way to coerce an array back to a simple vector object is to use `as.vector()`

```r
> vec <- as.vector(X)
```

However a similar result can be achieved by using `c()` with just one argument, simply for this side-effect:

```r
> vec <- c(X)
```

There are slight differences between the two, but ultimately the choice between them is largely a matter of style (with the former being preferable).

### 5.10 Frequency tables from factors. The `table()` function

Recall that a factor defines a partition into groups. Similarly a pair of factors defines a two way cross classification, and so on. The function `table()` allows frequency tables to be calculated from equal length factors. If there are `k` category arguments, the result is a `k`-way array of frequencies.

Suppose, for example, that `statef` is a factor giving the state code for each entry in a data vector. The assignment

```r
> statefr <- table(statef)
```

gives in `statefr` a table of frequencies of each state in the sample. The frequencies are ordered and labelled by the levels attribute of the category. This simple case is equivalent to, but more convenient than,

```r
> statefr <- tapply(statef, statef, length)
```
Further suppose that `incomef` is a category giving a suitably defined “income class” for each entry in the data vector, for example with the `cut()` function:

```r
> factor(cut(incomes,breaks=35+10*(0:7))) -> incomef
```

Then to calculate a two-way table of frequencies:

```r
> table(incomef,statef)
  act nsw nt qld sa tas vic wa
35+ thru 45 1 1 0 1 0 0 1 0
45+ thru 55 1 1 1 1 2 0 1 3
55+ thru 65 0 3 1 3 2 2 2 1
65+ thru 75 0 1 0 0 0 0 1 0
```

Extension to higher way frequency tables is immediate.
6 Lists, data frames, and their uses

6.1 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 complex vector, a character array, a function, and so on. Here is a simple example of how to make a list:

```r
> Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
```

Components are always numbered and may always be referred to as such. Thus if Lst is the name of a list with four components, these may be individually referred to as Lst[[1]], Lst[[2]], Lst[[3]] and Lst[[4]]. If, further, Lst[[4]] is a vector subscripted array then Lst[[4]][1] is its first entry.

If Lst is a list, then the function `length(Lst)` gives the number of (top level) components it has.

Components of lists may also be named, and in this case 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

```r
> name$component.name
```

for the same thing.

This is a very useful convention as it makes it easier to get the right component if you forget the number.

So in the simple example given above:

Lst$name is the same as Lst[[1]] and is the string "Fred";
Lst$wife is the same as Lst[[2]] and is the string "Mary";
Lst$child.ages[1] is the same as Lst[[4]][1] and is the number 4.

It is very important to distinguish Lst[[1]] from Lst[1]. "[[...]]" is the operator used to select a single element, whereas "[...]" is a general subscripting operator. Thus the former is the first object in the list Lst, and if it is a named list the name is not included. The latter is a sublist of the list Lst consisting of the first entry only. If it is a named list, the name is transferred to the sublist.

The names of components may be abbreviated down to the minimum number of letters needed to identify them uniquely. Thus Lst$coefficients may be minimally specified as Lst$coe and Lst$covariance as Lst$cov.

The vector of names is in fact simply an attribute of the list like any other and may be handled as such. Other structures besides lists may, of course, similarly be given a names attribute also.

6.2 Constructing and modifying lists

New lists may be formed from existing objects by the function list(). An assignment of the form

```r
> Lst <- list(name1=object1, name2=object2, ..., name_m=object_m)
```

sets up a list Lst of \( m \) components using \( object_1, \ldots, object_m \) for the components and giving them names as specified by the argument names, (which can be freely chosen). If these names are omitted, the components are numbered only. The components used to form the list are copied when forming the new list and the originals are not affected.
Lists, like any subscripted object, can be extended by specifying additional components. For example

> List[5] <- list(matrix=M)

### 6.2.1 Concatenating lists

When the concatenation function c() is given list arguments, the result is an object of mode list also, whose components are those of the argument lists joined together in sequence.

> list.ABC <- c(list.A, list.B, list.C)

Recall that with vector objects as arguments the concatenation function similarly joined together all arguments into a single vector structure. In this case all other attributes, such as dim attributes, are discarded.

### 6.3 Some functions returning a list result

Functions and expressions in S-PLUS must return a single object as their result; in cases where the result has several component parts, the usual form is that of a list with named components.

#### 6.3.1 Eigenvalues and eigenvectors

The function eigen(Sm) calculates the eigenvalues and eigenvectors of a symmetric matrix Sm. The result of this function is a list of two components named values and vectors. The assignment

> ev <- eigen(Sm)

will assign this list to ev. Then ev$vals is the vector of eigenvalues of Sm and ev$vec is the matrix of corresponding eigenvectors. Had we only needed the eigenvalues we could have used the assignment:

> evals <- eigen(Sm)$values

evals now holds the vector of eigenvalues and the second component is discarded. If the expression

> eigen(Sm)

is used by itself as a command the two components are printed, with their names, at the terminal.

#### 6.3.2 Singular value decomposition and determinants

The function svd(M) takes an arbitrary matrix argument, M, and calculates the singular value decomposition of M. This consists of a matrix of orthonormal columns U with the same column space as M, a second matrix of orthonormal columns V whose column space is the row space of M and a diagonal matrix of positive entries D such that $M = U \times D \times V^t$. D is actually returned as a vector of the diagonal elements. The result of svd(M) is actually a list of three components named d, u and v, with evident meanings.

If M is in fact square, then, it is not hard to see that

> abdetM <- prod(svd(M)$d)

calculates the absolute value of the determinant of M. If this calculation were needed often with a variety of matrices it could be defined as an S-PLUS function

> abdet <- function(M) prod(svd(M)$d)
after which we could use `absdet()` as just another S-PLUS function. As a further trivial but potentially useful example, you might like to consider writing a function, say `tr()`, to calculate the trace of a square matrix. [Hint: You will not need to use an explicit loop. Look again at the `diag()` function.]

Functions will be discussed formally later in these notes.

### 6.3.3 Least squares fitting and the QR decomposition

The function `lsfit()` returns a list giving results of a least squares fitting procedure. An assignment such as

```r
> ans <- lsfit(x, y)
```

gives the results of a least squares fit where `y` is the vector of observations and `x` is the design matrix. See the help facility for more details, and also for the follow-up function `ls.diag()` for, among other things, regression diagnostics. Note that a grand mean term is automatically included and need not be included explicitly as a column of `x`.

Another closely related function is `qr()` and its allies. Consider the following assignments

```r
> Xplus <- qr(x)
> b <- qr.coef(Xplus, y)
> fit <- qr.fitted(Xplus, y)
> res <- qr.resid(Xplus, y)
```

These compute the orthogonal projection of `y` onto the range of `x` in `fit`, the projection onto the orthogonal complement in `res` and the coefficient vector for the projection in `b`, that is, `b` is essentially the result of the MATLAB ‘backslash’ operator.

It is not assumed that `x` has full column rank. Redundancies will be discovered and removed as they are found.

This alternative is the older, low level way to perform least squares calculations. Although still useful in some contexts, it would now generally be replaced by the statistical models features, as will be discussed in §10.

### 6.4 Data frames

A *data frame* is a list with class `data.frame`. There are restrictions on lists that may be made into data frames, namely

- The components must be vectors (numeric, character, or logical), factors, numeric matrices, lists, or other data frames.
- Matrices, lists, and data frames provide as many variables to the new data frame as they have columns, elements, or variables, respectively.
- Numeric vectors and factors are included as is, and non-numeric vectors are coerced to be factors, whose levels are the unique values appearing in the vector.
- Vector structures appearing as variables of the data frame must all have the same length, and matrix structures must all have the same row size.

Data frames may in many ways be regarded as a matrix with columns possibly of differing modes and attributes. It may be displayed in matrix form, and its rows and columns extracted using matrix indexing conventions.
6.4 Data frames

6.4.1 Making data frames

Objects satisfying the restrictions placed on the columns (components) of a data frame may be used to form one using the function `data.frame`:

```r
> accountants <- data.frame(home=statef, loot=income, shot=incomef)
```

A list whose components conform to the restrictions of a data frame may be coerced into a data frame using the function `as.data.frame()`

The simplest way to construct a data frame from scratch is to use the `read.table()` function to read an entire data frame from an external file. This is discussed further in §7.

6.4.2 `attach()` and `detach()`

The `§` notation, such as `accountants$statef`, for list components is not always very convenient. A useful facility would be somehow to make the components of a list or data frame temporarily visible as variables under their component name, without the need to quote the list name explicitly each time.

The `attach()` function, as well as having a directory name as its argument, may also have a data frame. Thus suppose `lentils` is a data frame with three variables `lentils$u`, `lentils$v`, `lentils$w`. The attach

```r
> attach(lentils)
```

places the data frame in the search list at position 2, and provided there are no variables `u`, `v` or `w` in position 1, `u`, `v` and `w` are available as variables from the data frame in their own right. At this point an assignment such as

```r
> u <- v+w
```

does not replace the component `u` of the data frame, but rather masks it with another variable `u` in the working directory at position 1 on the search list. To make a permanent change to the data frame itself, the simplest way is to resort once again to the `§` notation:

```r
> lentils$u <- v+w
```

However the new value of component `u` is not visible until the data frame is detached and attached again.

To detach a data frame, use the function

```r
> detach()
```

More precisely, this statement detaches from the search list the entity currently at position 2. Thus in the present context the variables `u`, `v` and `w` would be no longer visible, except under the list notation as `lentils$u` and so on.

**NOTE:** With the current release of S-PLUS the search list can contain at most 20 items. Avoid attaching the same data frame more than once. Always detach the data frame as soon as you have finished using its variables.

6.4.3 Working with data frames

A useful convention that allows you to work with many different problems comfortably together in the same working directory is

- gather together all variables for any well defined and separate problem in a data frame under a suitably informative name;
• when working with a problem attach the appropriate data frame at position 2, and use the working directory at level 1 for operational quantities and temporary variables;

• before leaving a problem, add any variables you wish to keep for future reference to the data frame using the $ form of assignment, and then detach();

• finally remove all unwanted variables from the working directory and keep it as clean of left-over temporary variables as possible.

In this way it is quite simple to work with many problems in the same directory, all of which have variables named x, y and z, for example.

6.4.4 Attaching arbitrary lists

attach() is a generic function that allows not only directories and data frames to be attached to the search list, but other classes of object as well. In particular any object of mode list may be attached in the same way:

> attach(any.old.list)

It is also possible to attach objects of class pframe, to so-called parametrized data frames, needed for nonlinear regression and elsewhere.

Being a generic function it is also possible to add methods for attaching yet more classes of object should the need arise.
7 Reading data from files

Large data objects will usually be read as values from external files rather than entered during an S-PLUS session at the keyboard. S-PLUS input facilities are simple and their requirements are fairly strict and even rather inflexible. There is a clear presumption by the designers of S-PLUS that you will be able to modify your input files using other tools, such file editors and the UNIX utilities sed and awk to fit in with the requirements of S-PLUS. Generally this is very simple.

There is, however, a function make.fields() that can be used to convert a file with fixed width, non separated, input fields into a file with separated fields. There is also a facility count.fields() that will count the number of fields on each line of such a file. Occasionally for very simple conversion and checking problems these may be adequate to the task, but in most cases it is better to do the preliminary spade work before the S-PLUS session begins.

If variables are to be held mainly in data frames, as we strongly suggest, they should be, an entire data frame can be read directly with the read.table() function. There is also an older, more primitive input function, scan(), that is still occasionally useful in special circumstances.

7.1 The read.table() function

To read an entire data frame directly, the external file will normally have a special form.

- The first line of the file should have a name for each variable in the data frame.
- Each additional line of the file has its first item a row label and the values for each variable.

If the file has one fewer item in its first line than in its second, this arrangement is presumed to be in force. So the first few lines of a file to be read as a data frame might look as in Figure 2. By default numeric items (except row labels) are read as numeric variables and non-numeric

<table>
<thead>
<tr>
<th></th>
<th>Price</th>
<th>Floor</th>
<th>Area</th>
<th>Rooms</th>
<th>Age</th>
<th>Cent.heat</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>52.00</td>
<td>111.0</td>
<td>830</td>
<td>5</td>
<td>6.2</td>
<td>no</td>
</tr>
<tr>
<td>02</td>
<td>54.75</td>
<td>128.0</td>
<td>710</td>
<td>5</td>
<td>7.5</td>
<td>no</td>
</tr>
<tr>
<td>03</td>
<td>57.50</td>
<td>101.0</td>
<td>1000</td>
<td>5</td>
<td>4.2</td>
<td>no</td>
</tr>
<tr>
<td>04</td>
<td>57.50</td>
<td>131.0</td>
<td>690</td>
<td>6</td>
<td>8.8</td>
<td>no</td>
</tr>
<tr>
<td>05</td>
<td>59.75</td>
<td>93.0</td>
<td>900</td>
<td>5</td>
<td>1.9</td>
<td>yes</td>
</tr>
</tbody>
</table>

...  

Figure 2: Input file form with names and row labels

variables, such as Cent.heat in the example, as factors. This can be changed if necessary.

The function read.table() can then be used to read the data frame directly

> HousePrice <- read.table("houses.data")

Often you will want to omit including the row labels directly and use the default labels. In this case the file may omit the row label column as in Figure 3. The data frame may then be read as

> HousePrice <- read.table("houses.data", header=T)

where the header=T option specifies that the first line is a line of headings, and hence, by implication from the form of the file, that no explicit row labels are given.

7.2 The scan() function

Suppose the data vectors are of equal length and are to be read in in parallel. Further suppose that there are three vectors, the first of mode character and the remaining two of mode numeric,
and the file is `input.dat`. The first step is to use `scan()` to read in the three vectors as a list, as follows:

```r
> in <- scan("input.dat", list("", 0, 0))
```

The second argument is a dummy list structure that establishes the mode of the three vectors to be read. The result, held in `in`, is a list whose components are the three vectors read in. To separate the data items into three separate vectors, use assignments like

```r
> label <- in[[1]]; x <- in[[2]]; y <- in[[3]]
```

More conveniently, the dummy list can have named components, in which case the names can be used to access the vectors read in. For example

```r
> in <- scan("input.dat", list(id="", x=0, y=0))
```

If you wish to access the variables separately they may either be re-assigned to variables in the working frame:

```r
> label <- in$id; x <- in$x; y <- in$y
```

or the list may be attached at position 2 of the search list, (see §6.4.4).

If the second argument is a single value and not a list, a single vector is read in, all components of which must be of the same mode as the dummy value.

```r
> X <- matrix(scan("light.dat", 0), ncol=5, byrow=T)
```

There are more elaborate input facilities available and these are detailed in the manual.

### 7.3 Other facilities; editing data

Once a data set has been read, there is an X-window based facility in S-PLUS for making small changes. The command

```r
> xnew <- data.ed(xold)
```

will allow you to edit your data set `xold` using a spreadsheet-like environment in a separate editing window, and on completion the changed object is assigned to `xnew`. `xold`, and hence `xnew`, can be any matrix, vector, data frame, or atomic data object.
8 More language features. Loops and conditional execution

8.1 Grouped expressions

S-PLUS is an expression language in the sense that its only command type is a function or expression which returns a result. Even an assignment is an expression whose result is the value assigned, and it may be used wherever any expression may be used; in particular multiple assignments are possible.

Commands may be grouped together in braces, \{expr_1; expr_2; \ldots; expr_m\}, 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.

8.2 Control statements

8.2.1 Conditional execution: if statements

The language has available a conditional construction of the form

\[
\text{if (expr_1) expr_2 else expr_3}
\]

where \(\text{expr}_1\) must evaluate to a logical value and the result of the entire expression is then evident.

8.2.2 Repetitive execution: for loops, repeat and while

There is also a \texttt{for}-loop construction which has the form

\[
\text{for (name in expr_1) expr_2}
\]

where \textit{name} is the loop variable. \textit{expr}_1 is a vector expression, (often a sequence like \texttt{1:20}), and \textit{expr}_2 is often a grouped expression with its sub-expressions written in terms of the dummy \textit{name}. \textit{Expr}_2 is repeatedly evaluated as \textit{name} ranges through the values in the vector result of \textit{expr}_1.

As an example, suppose \textit{ind} is a vector of class indicators and we wish to produce separate plots of \(y\) versus \(x\) within classes. One possibility here is to use \texttt{coplot()} to be discussed later, which will produce an array of plots corresponding to each level of the factor. Another way to do this, now putting all plots on the one display, is as follows:

\[
\text{yc <- split(y, ind); xc <- split(x, ind)}
\]
\[
\text{for (i in 1:length(yc)) (plot(xc[[i]], yc[[i]]);}
\]
\[
\text{abline(lsfit(xc[[i]], yc[[i]]))}
\]

(Note the function \texttt{split()} which produces a list of vectors got by splitting a larger vector according to the classes specified by a category. This is a useful function, mostly used in connection with boxplots. See the \texttt{help} facility for further details.)

\textbf{WARNING:} \texttt{for()} loops are very slow in the present version of S-PLUS and should be avoided if possible. Many functions, such as \texttt{apply()}, \texttt{tapply()}, \texttt{sapply()} and others, are written primarily to avoid using explicit \texttt{for()} loops.

Other looping facilities include the

\[
\text{repeat expr}
\]

statement and the
> while (condition) expr
statement.

The break statement can be used to terminate any loop, possibly abnormally. For repeat loops, this is the only way to terminate them.

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 §9, and where more examples will emerge.
9 Writing your own functions

As we have seen informally along the way, the S-PLUS language allows the user to create objects of mode function. These are true S-PLUS functions that are stored in a special internal form and may be used in further expressions and so on. In the process the language gains enormously in power, convenience and elegance, and learning to write useful functions is one of the main ways to make your use of S-PLUS comfortable and productive.

It should be emphasized that most of the functions supplied as part of the S-PLUS system, such as \texttt{mean()}, \texttt{var()}, \texttt{postscript()} and so on, are themselves written in S-PLUS and thus do not differ materially from user written functions.

A function is defined by an assignment of the form

\begin{verbatim}
> name <- function(arg1, arg2, ...) expression
\end{verbatim}

The expression is an S-PLUS expression, (usually a grouped expression), that uses the arguments, \texttt{arg1}, 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 \texttt{name(expr1, expr2, ...)} and may occur anywhere a function call is legitimate.

9.1 Simple examples

As a first example, consider a function to calculate the two sample \(t\)-statistic, showing "all the steps". This is an artificial example, of course, since there are other, simpler ways of achieving the same end.

The function is defined as follows:

\begin{verbatim}
> twosam <- function(y1, y2) {
    n1 <- length(y1); n2 <- length(y2)
    yb1 <- mean(y1); yb2 <- mean(y2)
    s1 <- var(y1); s2 <- var(y2)
    s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
    tst <- (yb1 - yb2)/sqrt(s2*(1/n1 + 1/n2))
    tst
}
\end{verbatim}

With this function defined, you could perform two sample \(t\)-tests using a call such as

\begin{verbatim}
> tstat <- twosam(datamale, datafemale); tstat
\end{verbatim}

As a second example, consider a function to emulate directly the MATLAB backslash command, which returns the coefficients of the orthogonal projection of the vector \(y\) onto the column space of the matrix, \(X\). (This is ordinarily called the least squares estimates of the regression coefficients). This would ordinarily be done with the \texttt{qr()} function; however this is sometimes a bit tricky to use directly and it pays to have a simple function such as the following to use it safely.

Thus given a vector \(y^{n\times1}\) and a matrix \(X^{n\times p}\) then

\[
X'y =^\text{def.} (X'X)^{-} X'y
\]

where \((X'X)^{-}\) is a generalised inverse of \(X'X\).

\begin{verbatim}
> bslash <- function(X, y) {
    X <- qr(X)
    qr.coef(X, y)
}
\end{verbatim}
After this object is created it is permanent, like all objects, and may be used in statements such as

```r
> regcoef <- bslash(Xmat, yvar)
```

and so on.

The classical S-PLUS function lsfit() does this job quite well, and more. It in turn uses the functions qr() and qr.coef() in the slightly counterintuitive way above to do this part of the calculation. Hence there is probably some value in having just this part isolated in a simple to use function if it is going to be in frequent use. If so, we may wish to make it a matrix binary operator for even more convenient use.

### 9.2 Defining new binary operators.

Had we given the `bslash()` function a different name, namely one of the form

```r
%anything%
```

it could have been used as a binary operator in expressions rather than in function form. Suppose, for example, we choose ! for the internal character. The function definition would then start as

```r
> "%!" <- function(x, y) {... }
```

(Note the use of quote marks.) The function could then be used as `X %!% y`. (The backslash symbol itself is not a convenient choice as it presents special problems in this context.)

The matrix multiplication operator, `%^%`, and the outer product matrix operator `%o%` are other examples of binary operators defined in this way.

### 9.3 Named arguments and defaults. “…”

As first noted in §2.3 if arguments to called functions are given in the “name=object” form, they may be given in any order. Furthermore the argument sequence may begin in the unnamed, positional form, and specify named arguments after the positional arguments.

Thus if there is a function `fun1` defined by

```r
> fun1 <- function(data, data.frame, graph, limit) {...}
```

Then the function may be invoked in several ways, for example

```r
> ans <- fun1(d, df, 20, T)
> ans <- fun1(d, df, graph=T, limit=20)
> ans <- fun1(data=d, limit=20, graph=T, data.frame=df)
```

are all equivalent.

In many cases arguments can be given commonly appropriate default values, in which case they may be omitted altogether from the call when the defaults are appropriate. For example, if `fun1` were defined as

```r
> fun1 <- function(data, data.frame, graph=T, limit=20) {...}
```

it could be called as

```r
> ans <- fun1(d, df)
```

which is now equivalent to the three cases above, or as

---

8This ellipsis is used as a customary typographical device to mean an abridgment. This is not the case in the latter part of this section.
9.4 Assignments within functions are local. Frames.

> ans <- fun1(d, df, limit=10)

which changes one of the defaults.

It is important to note that defaults may be arbitrary expressions, even involving other arguments to the same function; they are not restricted to be constants as in our simple example here.

Another frequent requirement is to allow one function to pass on argument settings to another. For example many graphics functions use the function par() and functions like plot() allow the user to pass on graphical parameters to par() to control the graphical output. (See §11.4.1 for more details on the par() function.) This can be done by including an extra argument, literally "...", of the function, which may then be passed on. An outline example is given in Figure 4.

fun1 <- function(data, data.frame, graph=T, limit=20, ...) {
  # omitted statements
  if (graph)
    par(pch="*", ...)
  # more omissions
}

Figure 4: Use of the ellipsis argument, "..."

Note here that the ellipses, "..." are literal S-PLUS, not a typographical device.

9.4 Assignments within functions are local. Frames.

Note that any ordinary assignments done within the function are local and temporary and are lost after exit from the function. Thus the assignment \( x \leftarrow \text{qr}(x) \) does not affect the value of the argument in the calling program.

To understand completely the rules governing the scope of S-PLUS assignments the reader needs to be familiar with the notion of an evaluation frame. This is a somewhat advanced, though hardly difficult, topic and is not covered further in these notes.

If global and permanent assignments are intended within a function, then the 'superassignment' operator, '<<-' can be used. See the help document for details, and also see the synchronize() function.

9.5 More advanced examples

9.5.1 Efficiency factors in block designs

As a more complete, if a little pedestrian, example of a function, consider finding the efficiency factors for a block design. (Some aspects of this problem have already been discussed in §5.3.)

A block design is defined by two factors, say blocks (b levels) and varieties, (v levels). If \( R^{v \times v} \) and \( K^{b \times b} \) are the replications and block size matrices, and \( N^{b \times v} \) is the incidence matrix, then the efficiency factors are defined as the eigenvalues of the matrix

\[
E = I_v - R^{-1/2}N^T K^{-1} N R^{-1/2} = I_v - A' A
\]

where \( A = K^{-1/2}NR^{-1/2} \). One way to write the function is as in Figure 5.
It is numerically slightly better to work with the singular value decomposition on this occasion rather than the eigenvalue routines.

The result of the function is a list giving not only the efficiency factors as the first component, but also the block and variety canonical contrasts, since sometimes these give additional useful qualitative information.

9.5.2 Dropping all names in a printed array

For printing purposes with large matrices or arrays, it is often useful to print them in close block form without the array names or numbers. Removing the dimnames attribute will not achieve this effect, but rather the array must be given a dimnames attribute consisting of empty strings. For example to print a matrix, X

```r
> temp <- X
> dimnames(temp) <- list(rep("", nrow(X)), rep("", ncol(X))
> temp; rm(temp)
```

This can be much more conveniently done using a function, no.dimnames(), shown in Figure 6, as a “wrap around” to achieve the same result. It also illustrates how some effective and useful user functions can be quite short. With this function defined, an array may be printed in close format using

```r
no.dimnames <- function(a){
  #
  # Remove all dimension names from an array for compact printing.
  #
  d <- list()
  l <- 0
  for(i in dim(a)) {
    d[[l <- l + 1]] <- rep("", i)
  }
  dimnames(a) <- d
  a
}
```

Figure 6: A function for printing arrays in compact form
> no.dimnames(X)

This is particularly useful for large integer arrays, where patterns are the real interest rather than the values.

### 9.5.3 Recursive numerical integration

Functions may be recursive, and may themselves define functions within themselves. Note, however, that such functions, or indeed variables, are not inherited by called functions in higher evaluation frames as they would be if they were on the search list.

The example in Figure 7 shows a naive way of performing one dimensional numerical integration. The integrand is evaluated at the end points of the range and in the middle. If the one-panel trapezium rule answer is close enough to the two panel, then the latter is returned as the value. Otherwise the same process is recursively applied to each panel. The result is an adaptive integration process that concentrates function evaluations in regions where the integrand is farthest from linear. There is, however, a heavy overhead, and the function is only competitive with other algorithms when the integrand is both smooth and very difficult to evaluate.

The example is also given partly as a little puzzle in S-PLUS programming.

```r
area <- function(f, a, b, eps = 1.0e-06, lim = 10)
{
  fun1 <- function(f, a, b, fa, fb, a0, eps, lim, fun)
  {
    d <- (a + b)/2
    h <- (b - a)/4
    fd <- f(d)
    a1 <- h * (fa + fd)
    a2 <- h * (fd + fb)
    if(abs(a0 - a1 - a2) < eps || lim == 0)
      return(a1 + a2)
    else {
      return(fun(f, a, d, fa, fd, a1, eps, lim - 1, fun) +
             fun(f, d, b, fb, a2, eps, lim - 1, fun))
    }
  }
  fa <- f(a)
  fb <- f(b)
  a0 <- ((fa + fb) * (b - a))/2
  fun1(f, a, b, fa, fb, a0, eps, lim, fun1)
}
```

Figure 7: A recursive function within a function

### 9.6 Customising the environment. .First and .Last

Any function named .First() in the .Data directory has a special status. It is automatically performed at the beginning of an S-PLUS session and may be used to initialise the environment. For example, the definition in Figure 8 alters the prompt to $ and sets up various other useful things that can then be taken for granted in the rest of the session. Similarly a function .Last(), if defined, is executed at the very end of the session. An example is given in Figure 9.
> .First <- function() {
  options(prompt=" $ ", continue="+\t")  # $ is the prompt
  options(digits=5, length=999)  # custom numbers and printout
  options(gui="motif")  # default graphics user interface
  teX4014()  # for terminal work
  par(pch = "+")  # plotting character
  attach(paste(unix("echo $HOME"), "/.Data", sep = "))  # Home of my personal library
  library(examples)  # attach also the system examples
}

Figure 8: An example of a .First() function

> .Last <- function() {
  graphics.off()  # a small safety measure.
  cat(paste(unix("date"),"\nAdios\n"))  # Is it time for lunch?
}

Figure 9: An example of a .Last() function

9.7 Classes, generic functions and object orientation

The class of an object determines how it will be treated by what are known as generic functions. Put the other way round, a generic function performs a task or action on its arguments specific to the class of the argument itself. If the argument lacks any class attribute, or has a class not catered for specifically by the generic function in question, there is always a default action provided.

An example makes things clearer. In a sense the print() function has always been generic, since its action is to adopt a style of output appropriate to its arguments. Thus a matrix appears as a matrix, a vector as a vector, and so on. (Note that the print() function can be called explicitly, or implicitly by giving an expression as a complete command.)

The August 1991 release of S-PLUS increases the number of such functions, alters the mechanism by which they are implemented and via the class mechanism offers the user the facility of designing and writing generic functions for special purposes. Among the other new, or newly generic functions are plot() for displaying objects graphically, summary() for summarising analyses of various types, and anova() for comparing statistical models.

The number of generic functions that can treat a class in a specific way can be quite large. For example, the functions that can accommodate in some fashion objects of class data.frame include

[ , [<-, detach, dimnames<-, pairs, signif, [<-, aperm, dim, formula, plot, summary, [ [-, atan, dimnames, ordered<-, print, t,]

A currently complete list can be got by using the methods() function:

> methods(class="data.frame")

Conversely the number of classes a generic function can handle can also be quite large. For example the plot() function has variants for classes of object

data.frame, default, glm, pgram, surv.fit, design, factor, lm, preloess, tree, formula, gam, loess, profile, tree.sequence,
and perhaps more. A complete list can be got again by using the `methods()` function:

```r
> methods(plot)
```

The reader is referred to the official references for a complete discussion of this mechanism.
10 Statistical models in S-PLUS

This section presumes the reader has some familiarity with statistical methodology, in particular with regression analysis and the analysis of variance. Later we make some rather more ambitious presumptions, namely that something is known about generalized linear models and nonlinear regression.

The requirements for fitting statistical models are sufficiently well defined to make it possible to construct general tools that apply in a broad spectrum of problems. Since the August 1991 release S-PLUS provides an interlocking suite of facilities that make fitting statistical models very simple. However these are not at the same high level as those in, say, Genstat, especially in the form of the output which in keeping with general S-PLUS policy is rather minimal.

10.1 Defining statistical models; formulae

The template for a statistical model is a linear regression model with independent, homoscedastic errors

\[ y_i = \sum_{j=0}^{p} \beta_j x_{ij} + \epsilon_i, \quad \epsilon_i \sim \text{NID}(0, \sigma^2), \quad i = 1, 2, \ldots, n \]

In matrix terms this would be written

\[ y = X\beta + e \]

where the \( y \) is the response vector, \( X \) is the model matrix or design matrix and has columns \( x_0, x_1, \ldots, x_p \), the determining variables. Very often \( x_0 \) will be a column of 1s defining an intercept term.

Examples.

Before giving a formal specification, a few examples may usefully set the picture.

Suppose \( y, x, x_0, x_1, x_2, \ldots \) are numeric variables, \( X \) is a matrix and \( A, B, C, \ldots \) are factors. The following formulae on the left side below specify statistical models as described on the right.

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y \sim x )</td>
<td>Both imply the same simple linear regression model of ( y ) on ( x ). The first has an implicit intercept term, and the second an explicit one.</td>
</tr>
<tr>
<td>( y \sim 1 + x )</td>
<td>Simple linear regression of ( y ) on ( x ) through the origin, (that is, without an intercept term).</td>
</tr>
<tr>
<td>( y \sim -1 + x )</td>
<td>Multiple regression of the transformed variable, ( \log(y) ), on ( x_1 ) and ( x_2 ) (with an implicit intercept term).</td>
</tr>
<tr>
<td>( \log(y) \sim x_1 + x_2 )</td>
<td>Polynomial regression of ( y ) on ( x ) of degree 2. The first form uses orthogonal polynomials, and the second uses explicit powers, as basis.</td>
</tr>
<tr>
<td>( y \sim \text{poly}(x, 2) )</td>
<td>Multiple regression ( y ) with model matrix consisting of the matrix ( X ) as well as polynomial terms in ( x ) to degree 2.</td>
</tr>
<tr>
<td>( y \sim A )</td>
<td>Single classification analysis of variance model of ( y ), with classes determined by ( A ).</td>
</tr>
<tr>
<td>( y \sim A + x )</td>
<td>Single classification analysis of covariance model of ( y ), with classes determined by ( A ), and with covariate ( x ).</td>
</tr>
</tbody>
</table>
10.1 Defining statistical models; formulae

\[
y \sim A \times B
\]
Two factor non-additive model of \( y \) on \( A \) and \( B \). The first two specify the same crossed classification and the second two specify the same nested classification. In abstract terms all four specify the same model subspace.

\[
y \sim (A + B + C)^2
\]
Three factor experiment but with a model containing main effects and two factor interactions only. Both formulae specify the same model.

\[
y \sim A \times x
\]
Separate simple linear regression models of \( y \) on \( x \) within the levels of \( A \), with different codings. The last form produces explicit estimates of as many different intercepts and slopes as there are levels in \( A \).

\[
y \sim A + B + \text{Error}(C)
\]
An experiment with two treatment factors, \( A \) and \( B \), and error strata determined by factor \( C \). For example a split plot experiment, with whole plots, (and hence also subplots), determined by factor \( C \).

The operator " \( \sim \) " is used to define a model formula in S-PLUS. The form, for an ordinary linear model, is

\[
\text{response} \sim [\pm] \text{term}_1 \pm \text{term}_2 \pm \text{term}_3 \pm \ldots
\]

\( \text{response} \) is a vector or matrix, (or expression evaluating to a vector or matrix) defining the response variable(s).

\( \pm \) is an operator, either \( + \) or \( - \), implying the inclusion or exclusion of a term in the model, (the first is optional).

\( \text{term} \) is either

- a vector or matrix expression, or 1,
- a factor, or
- a formula expression consisting of factors, vectors or matrices connected by formula operators.

In all cases each term defines a collection of columns either to be added to or removed from the model matrix. A 1 stands for an intercept column and is by default included in the model matrix unless explicitly removed.

The formula operators are similar in effect to the Wilkinson and Rogers notation used by such programs a Glim and Genstat. One inevitable change is that the operator " \( \cdot \) " becomes " \( : \) " since the period is a valid name character in S-PLUS. The notation is summarised as in the Table 1 (based on Chambers & Hastie, p. 29).

Note that inside the parentheses that usually enclose function arguments all operators have their normal arithmetic meaning. The function \( I() \) is an identity function used only to allow terms in model formulae to be defined using arithmetic operators.

Note particularly that the model formulae specify the columns of the model matrix, specification of the parameters is implicit. This is not the case in other contexts, for example in fitting nonlinear models.
<table>
<thead>
<tr>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y \sim M$</td>
<td>$Y$ is modelled as $M$</td>
</tr>
<tr>
<td>$M_1 + M_2$</td>
<td>Include $M_1$ and $M_2$</td>
</tr>
<tr>
<td>$M_1 - M_2$</td>
<td>Include $M_1$ leaving out terms of $M_2$</td>
</tr>
<tr>
<td>$M_1 : M_2$</td>
<td>The tensor product of $M_1$ and $M_2$. If both terms factors, then the</td>
</tr>
<tr>
<td></td>
<td>“subclasses” factor.</td>
</tr>
<tr>
<td>$M_1 %!!!!!!!!!!!% M_2$</td>
<td>Similar to $M_1 : M_2$, but with a different coding.</td>
</tr>
<tr>
<td>$M_1 * M_2$</td>
<td>$M_1 + M_2 + M_1 : M_2$</td>
</tr>
<tr>
<td>$M_1 / M_2$</td>
<td>$M_1 + M_2 %!!!!!!!!!!!% M_1$</td>
</tr>
<tr>
<td>$M^n$</td>
<td>All terms in $M$ together with “interactions” up to order $n$</td>
</tr>
<tr>
<td>$I(M)$</td>
<td>Insulate $M$. Inside $M$ all operators have their normal arithmetic</td>
</tr>
<tr>
<td></td>
<td>meaning, and that term appears in the model matrix.</td>
</tr>
</tbody>
</table>

Table 1: Summary of model operator semantics

10.2 Regression models; fitted model objects

The basic function for fitting ordinary multiple models is `lm()`, and a streamlined version of the call is as follows:

```r
> fitted.model <- lm(formula, data=data.frame)
```

For example

```r
> fm2 <- lm(y ~ x1 + x2, data=production)
```

would fit a multiple regression model of $y$ on $x_1$ and $x_2$ (with implicit intercept term).

The important but technically optional parameter `data=production` specifies that any variables needed to construct the model should come first from the production data frame. This is the case regardless of whether data frame production has been attached to the search list or not.

10.3 Generic functions for extracting information

The value of `lm()` is fitted model object; technically a list of results of class `lm`. Information about the fitted model can then be displayed, extracted, plotted and so on by using generic functions that orient themselves to objects of class `lm`. A full list of these at the present time is

```
add1 coef effects kappa predict residuals
alias deviance family labels print summary
anova drop1 formula plot proj
```

A brief description of the most commonly used ones is given in Table 2.

10.4 Analysis of variance; comparing models

The model fitting function `aov(formula, data=data.frame)` operates at the simplest level in a very similar way to the function `lm()`, and most of the generic functions listed in Table 2 apply.
### 10.4 Analysis of variance: comparing models

<table>
<thead>
<tr>
<th>Function</th>
<th>Value or Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>anova(object1, object2)</code></td>
<td>Compare a submodel with an outer model and product an analysis of variance table.</td>
</tr>
<tr>
<td><code>coefficients(object)</code></td>
<td>Extract the regression coefficient (matrix). Short form: <code>coef(object)</code>.</td>
</tr>
<tr>
<td><code>deviance(object)</code></td>
<td>Residual sum of squares, weighted if appropriate.</td>
</tr>
<tr>
<td><code>formula(object)</code></td>
<td>Extract the model formula.</td>
</tr>
<tr>
<td><code>plot(object)</code></td>
<td>Produce two plots, one of the observations against the fitted values, the other of the absolute residuals against the fitted values.</td>
</tr>
<tr>
<td><code>predict(object, newdata=data.frame)</code></td>
<td>The data frame supplied must have variables specified with the same labels as the original. The value is a vector or matrix of predicted values corresponding to the determining variable values in <code>data.frame</code>. <code>predict.gam()</code> is a safe alternative to <code>predict()</code> that can be used for <code>lm</code>, <code>glm</code> and <code>gam</code> fitted objects. It must be used, for example, in cases where orthogonal polynomials are used as the original basis functions, and the addition of new data implies different basis functions to the original.</td>
</tr>
<tr>
<td><code>print(object)</code></td>
<td>Print a concise version of the object. Most often used implicitly.</td>
</tr>
<tr>
<td><code>residuals(object)</code></td>
<td>Extract the (matrix of) residuals, weighted as appropriate. Short form: <code>resid(object)</code>.</td>
</tr>
<tr>
<td><code>summary(object)</code></td>
<td>Print a comprehensive summary of the results of the regression analysis.</td>
</tr>
</tbody>
</table>

Table 2: Commonly used generic functions on class `lm` objects

It should be noted that in addition `aov()` allows an analysis of models with multiple error strata such as split plot experiments, or balanced incomplete block designs with recovery of inter-block information. The model formula

\[
\text{response} \sim \text{mean.formula} + \text{Error(strata.formula)}
\]

specifies a multi-stratum experiment with error strata defined by the `strata.formula`. In the simplest case, `strata.formula` is simply a factor, when it defines a two strata experiment, namely between and within the levels of the factor.

For example, with all determining variables factors, a model formula such as that in:

```r
> fm <- aov(yield ~ v + n*p*k + Error(farms/blocks), data=farm.data)
```

would typically be used to describe an experiment with mean model \( v + n*p*k \) and three error strata, namely “between farms”, “within farms, between blocks” and “within blocks”.

#### 10.4.1 ANOVA tables

Note also that the analysis of variance table (or tables) are for a sequence of fitted models. The sums of squares shown are the decrease in the residual sums of squares resulting from an inclusion
of that term in the model at that place in the sequence. Hence only for orthogonal experiments will the order of inclusion be inconsequential.

For multistratum experiments the procedure is first to project the response onto the error strata, again in sequence, and to fit the mean model to each projection. For further details, see Chambers and Hastie, §5.

A more flexible alternative to the default full ANOVA table is to compare two or more models directly using the anova() function.

> anova(fitted.model.1, fitted.model.2, 

The display is then an ANOVA table showing the differences between the fitted models when fitted in sequence. The fitted models being compared would usually be an hierarchical sequence, of course. This does not give different information to the default, but rather makes it easier to comprehend and control.

10.5 Updating fitted models. The ditto name “.”

The update() function is largely a convenience function that allows a model to be fitted that differs from one previously fitted usually by just a few additional or removed terms. Its form is

> new.model <- update(old.model, new.formula)

In the new.formula the special name consisting of a period, “.”, only, can be used to stand for “the corresponding part of the old model formula”. For example

> fm05 <- lm(y ~ x1 + x2 + x3 + x4 + x5, data=production)
> fm6  <- update(fm05, . ~ . + x6)
> smf6 <- update(fm6, sqrt(.) ~ .)

would fit a five variate multiple regression with variables (presumably) from the data frame production, fit an additional model including a sixth regressor variable, and fit a variant on the model where the response had a square root transform applied.

Note especially that if the data= argument is specified on the original call to the model fitting function, this information is passed on through the fitted model object to update() and its allies.

The name “.” can also be used in other contexts, but with slightly different meaning. For example

> fmfull <- lm(y ~ , data=production)

would fit a model with response y and regressor variables all other variables in the data frame production.

Other functions for exploring incremental sequences of models are add1(), drop1(), step() and stepwise(). The names of these give a good clue to their purpose, but for full details see the help document.

10.6 Generalized linear models; families

Generalized linear modelling is a development of linear models to accommodate both non-normal response distributions and transformations to linearity in a clean and straightforward way. A generalized linear model may be described in terms of the following sequence of assumptions:

- There is a response, \( y \), of interest and stimulus variables \( x_1, x_2, \ldots \) whose values influence the distribution of the response.
• The stimulus variables influence the distribution of $y$ through a single linear function, only. This linear function is called the linear predictor, and is usually written

$$\eta = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p,$$

hence $x_i$ has no influence on the distribution of $y$ if and only if $\beta_i = 0$.

• The distribution of $y$ is of the form

$$f_Y(y; \mu, \varphi) = \exp \left[ \frac{A}{\varphi} \{ y \lambda(\mu) - \gamma(\lambda(\mu)) \} + \tau(y, \varphi) \right]$$

where $\varphi$ is a scale parameter, (possibly known), and is constant for all observations, $A$ represents a prior weight, assumed known but possibly varying with the observations, and $\mu$ is the mean of $y$. So it is assumed that the distribution of $y$ is determined by its mean and possibly a scale parameter as well.

• The mean, $\mu$, is a smooth invertible function of the linear predictor:

$$\mu = m(\eta), \quad \eta = m^{-1}(\mu) = \ell(\mu)$$

and this inverse function, $\ell(.)$ is called the link function.

These assumptions are loose enough to encompass a wide class of models useful in statistical practice, but tight enough to allow the development of a unified methodology of estimation and inference, at least approximately. The reader is referred to any of the current reference works on the subject for full details, such as

*Generalized linear models* by Peter McCullagh and John A Nelder, 2nd edition, Chapman and Hall, 1989, or


### 10.6.1 Families

The class of generalized linear models handled by facilities supplied in S-PLUS includes gaussian, binomial, poisson, inverse gaussian and gamma response distributions and also quasi-likelihood models where the response distribution is not explicitly specified. In the latter case the variance function must be specified as a function of the mean, but in other cases this function is implied by the response distribution.

Each response distribution admits a variety of link functions to connect the mean with the linear predictor. Those automatically available are as in Table 3.

The combination of a response distribution, a link function and various other pieces of information that are needed to carry out the modelling exercise is called the family of the generalized linear model.

### 10.6.2 The glm function

Since the distribution of the response depends on the stimulus variables through a single linear function only, the same mechanism as was used for linear models can still be used to specify the linear part of a generalized model. The family has to be specified in a different way.

The S-PLUS function to fit a generalized linear model is `glm()` which uses the form

```r
> fitted.model <- glm(formula, family = family.generator, data = data.frame)
```

The only new feature is the `family.generator`, which is the instrument by which the family is described. It is the name of a function that generates a list of functions and expressions that
Table 3: Families and the link functions available to them

<table>
<thead>
<tr>
<th>Link Function</th>
<th>family (n)</th>
<th>binomial</th>
<th>gaussian</th>
<th>Gamma</th>
<th>inverse.gaussian</th>
<th>poisson</th>
<th>quasi</th>
</tr>
</thead>
<tbody>
<tr>
<td>logit</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>probit</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>cloglog</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>identity</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>inverse</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>log</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>1/mu^2</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>sqrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

together define and control the model and estimation process. Although this may seem a little complicated at first sight, its use is quite simple.

The names of the standard, supplied family generators are given under “Family Name” in Table 3. Where there is a choice of links, the name of the link may also be supplied with the family name, in parentheses as a parameter. In the case of the quasi family, the variance function may also be specified in this way.

Some examples make the process clear.

The gaussian family

A call such as

```r
> fm <- glm(y ~ x1+x2, family=gaussian, data=sales)
```

achieves the same result as

```r
> fm <- lm(y ~ x1+x2, data=sales)
```

but much less efficiently. Note how the gaussian family is not automatically provided with a choice of links, so no parameter is allowed. If a problem requires a gaussian family with a nonstandard link, this can usually be achieved through the quasi family, as we shall see later.

The binomial family

Consider a small, artificial example.

On the Greek island of Kalythos the male inhabitants suffer from a congenital eye disease, the effects of which become more marked with increasing age. Samples of islander males of various ages were tested for blindness and the results recorded. The data is shown in Table 4.

The problem we consider is to fit both logistic and probit models to this data, and to estimate for each model the LD50, that is the age at which the chance of blindness for a male inhabitant is 50%.

If $y$ is the number of blind at age $x$ and $n$ the number tested, both models have the form

$$y \sim B(n, F(\beta_0 + \beta_1 x))$$
where for the probit case, \( F(z) = \Phi(z) \) is the standard normal distribution function, and in the logit case, (the default), \( F(z) = e^z/(1 + e^z) \). In both cases the LD50 is

\[
LD50 = -\hat{\alpha}/\hat{\beta}
\]

that is, the point at which the argument of the distribution function is zero.

The first step is to set the data up as a data frame

\[
\begin{array}{c|cccccc}
\text{Age} & 20 & 35 & 45 & 55 & 70 \\
\hline
\text{No. tested} & 50 & 50 & 50 & 50 & 50 \\
\text{No. blind} & 6 & 17 & 26 & 37 & 44 \\
\end{array}
\]

Table 4: The Kalythos blindness data

To fit a binomial model using \texttt{glm()} there are two possibilities for the response:

- If the response is a vector it is assumed to hold binary data, and so must be a 0,1 vector.
- If the response is a two column matrix it is assumed that the first column holds the number of successes for the trial and the second holds the number of failures.

Here we need the second of these conventions, so we add a matrix to our data frame:

\[
\begin{array}{c|c}
\text{y} & \text{x} \\
\hline
50 & \text{c(20,35,45,55,70)} \\
50 & \text{rep(50,5)} \\
6 & \text{c(6,17,26,37,44)} \\
\end{array}
\]

To fit the models we use

\[
\begin{array}{c}
\text{glm}(y ~ x, \text{family=binomial(link=probit)}, \text{data=kalythos}) \\
\text{glm}(y ~ x, \text{family=binomial}, \text{data=kalythos}) \\
\end{array}
\]

Since the logit link is the default the parameter may be omitted on the second call. To see the results of each fit we could use

\[
\begin{array}{c}
\text{summary(fmp)} \\
\text{summary(fml)} \\
\end{array}
\]

Both models fit (all too) well. To find the LD50 estimate we can use a simple function:

\[
\begin{array}{c}
\text{ld50 <- function(b) } -b[1]/b[2] \\
\text{ldp <- ld50(coef(fmp)); ld1 <- ld50(coef(fml)); c(ldp, ld1)} \\
\end{array}
\]

The actual estimates from this data are 43.663 years and 43.601 years respectively.

**Poisson models**

With the poisson family the default link is the log, and in practice the major use of this family is to fit surrogate poisson log-linear models to frequency data, whose actual distribution is often multinomial. This is a large and important subject we will not discuss further here. It even forms a major part of the use of non-gaussian generalized models overall.

Occasionally genuinely poisson data arises in practice and in the past it was often analysed as gaussian data after either a log or a square-root transformation. As a graceful alternative to the latter, a poisson generalized linear model may be fitted as in the following example:

\[
\text{fmod <- glm(y ~ A+B + x, \text{family=poisson(link=sqrt)}, \text{data=worm.counts})}
\]
Quasi-likelihood models

For all families the variance of the response will depend on the mean and will have the scale parameter as a multiplier. The form of dependence of the variance on the mean is a characteristic of the response distribution; for example for the poisson distribution $\text{Var}[y] = \mu$.

For quasi-likelihood estimation and inference the precise response distribution is not specified, but rather only a link function and the form of the variance function as it depends on the mean. Since quasi-likelihood estimation uses formally identical techniques to those for the gaussian distribution, this family provides a way of fitting gaussian models with non-standard link functions or variance functions, incidently.

For example, consider fitting the non-linear regression

$$ y = \frac{\theta_1 z_1}{z_2 - \theta_2} + e \quad (1) $$

this may be written alternatively as

$$ y = \frac{1}{\beta_1 x_1 + \beta_2 x_2} + e $$

where $z_1 = z_2/z_1$, $x_2 = -1/x_1$, $\beta_1 = 1/\theta_1$ and $\beta_2 = \theta_2/\theta_1$. Supposing a suitable data frame to be set up we could fit this non-linear regression as

```r
> nlfit <- glm(y ~ x1 + x2 - 1, family = quasi(link = inverse, variance = constant), data = biochem)
```

The reader is referred to the manual and the help document for further information, as needed.

10.7 Nonlinear regression models; parametrized data frames

S-PLUS provides two functions to fit nonlinear models that do not conform even to the partially linear paradigm of generalized linear models. These are `nls` for arbitrary minimization problems where the objective functions is a sum of similar terms, and `nls()` for conventional nonlinear least squares estimation of normal nonlinear regression models.

In this brief introduction we only consider the nonlinear regression function `nls()` and leave `ms()` for the reader to pursue as needed.

10.7.1 Changes to the form of the model formula

In specifying a linear, or generalized linear model we could allow the regression parameters to be defined implicitly, and to be given names by transference from the column of the model matrix that they multiply.

In arbitrary nonlinear models no such simplicity applies and we have to specify the model as an ordinary expression that includes both determining variables and parameters together. For example to specify a model for a nonlinear regression such as 1 above, we would use

$$ y \sim t1*x1/(x2 - t2) $$

where $y$ is the response variable, $x1$ and $x2$ are determining variables and $t1$ and $t2$ are scalar parameters.

In such model formulae all operators have their usual arithmetic expression meaning, and the useful facility of expanding factors and forming cross and nested structures is no longer available. All parameters must be explicitly defined in the formula, even if they come from a linear part of the model.
10.7.2 Specifying the parameters

Since the model formula now contains both determining variables and parameters, there has to be some mechanism for specifying which are which. But of course once the parameters have been specified the remaining variates in the model formula must be variables.

As well as specifying which are the parameters, it is also necessary in this case to specify an initial approximation for each with which to start the iterative estimation procedure.

There are two ways of specifying this information:

- If the call to \texttt{nls()} has a \texttt{start=} parameter specified, its value must be a list of named components. The names of the list specify the names of the parameters and the values specify the starting values.
- If the data is held in a data frame, the parameters may similarly be defined as a parameters attribute of the data frame.

Since our policy is generally to work with data frames as much as possible, we show the second possibility in the next example.

Example

Consider again a nonlinear regression of the form \(1\). An easy way to find initial estimates for the parameters is to regress \(x_2y\) on \(x_1\) and \(x_2\):

\[
> \text{fm0} \leftarrow \text{lm}(x2*y \sim x1 + x2 - 1, \text{data=biochem})
\]

\[
> \text{th} \leftarrow \text{coef(fm0)}
\]

To name the parameters and associate them with the biochem is done as follows:

\[
> \text{parameters(biochem)} \leftarrow \text{list(t1=th[1], t2=th[2])}
\]

Now to fit the nonlinear regression model:

\[
> \text{fm} \leftarrow \text{nls}(y \sim t1*x1/(x2 - t2), \text{data=biochem})
\]

At this point we could use the \texttt{summary()} function and most of the other generics to investigate the model and display information. To extract the coefficients we could now use, for example

\[
> \text{th} \leftarrow \text{coef(fm)}
\]

and to make these least squares estimates the new values of the parameters associated with biochem we could simply repeat the step

\[
> \text{parameters(biochem)} \leftarrow \text{list(t1=th[1], t2=th[2])}
\]

Note that the function \texttt{parameters()} may either be used as an expression, in which case it extracts the list of parameters from a data frame, or it may be used as the target for an assignment, in which case it accepts a parameter list for a specified data frame. In this respect it is very similar to the \texttt{attributes()} function. There is also a function \texttt{params()} analogous to \texttt{attr()}, which handles one parameter at a time under a character string name.

10.8 Some non-standard models

We conclude this section with just a brief mention of some of the other facilities available in S-PLUS for special regression and data analysis problems.

**Local approximating regressions.** The \texttt{loess()} function fits a nonparametric regression by using a locally weighted regression. Such regressions are useful for highlighting a trend in messy data or for data reduction to give some insight into a large data set.
Robust regression  There are several functions available for fitting regression models in a way resistant to the influence of extreme outliers in the data. The most sophisticated of these is \texttt{rreg()}, but others include \texttt{lmsfit()} for least median squares regression and \texttt{lfit()} for regression using the $L_1$-norm. However these do not as yet have the facility of using formulae to specify the model function, for example, and conform to an older protocol, which makes them sometimes rather tedious to use. There is also a \texttt{robust()} facility to change a \texttt{glm} family object into a robust version for use with the \texttt{glm()} model fitting function.

Generalized additive models.  This technique aims to construct a regression function from smooth additive functions of the determining variables, usually one for each determining variable. The function \texttt{gam()} is in many ways similar to the other model fitting functions outlined above. In addition there are other model fitting functions that do a similar job. These include \texttt{avas()} and \texttt{ace()}. On the other hand \texttt{ppreg()} is available for projection pursuit regression, but this technique is still very much in need of a complete theoretical treatment and further practical experience. These latter functions are again conforming to an older protocol for model fitting functions and lack the convenience of the newer functions.

Tree based models  Rather than seek an explicit global linear model for prediction or interpretation, tree based models seek to bifurcate the data, recursively, at critical points of the determining variables in order to partition the data ultimately into groups that are as homogeneous as possible within, and as heterogeneous as possible between. The results often lead to insights that other data analysis methods tend not to yield.

Models are again specified in the ordinary linear model form. The model fitting function is \texttt{tree()}, but many other generic functions such as \texttt{plot()} and \texttt{text()} are well adapted to displaying the results of a tree-based model fit in a graphical way.
11 Graphical procedures

The graphical facilities are an important and extremely versatile component of the S-PLUS environment. Best results are obtained when S-PLUS is used with a high quality graphics system such as X-windows, although even a simple ASCII terminal can be quite effective for some purposes.

Before the graphical facilities of S-PLUS may be used, it is necessary to inform S-PLUS what type of device is being used by starting a *device driver*. In an X-windows environment, the command to do this may be

> X11()

(which creates a separate window in which high-quality graphical output will appear,) or for a simpler graphics terminal the command

> tek4014()

may be appropriate.

Once a device driver is running S-PLUS plotting commands can be used to construct and display graphical objects. Plotting commands are divided into three basic groups:

**High-level** plotting functions create a new plot on the graphics device, possibly with axes, labels, titles and so on.

**Low-level** plotting functions add more information to an existing plot, such as extra points, lines and labels.

**Interactive** graphics functions allow you interactively add information to, or extract information from, an existing plot, using a pointing device such as a mouse.

Furthermore, S-PLUS maintains a list of *graphical parameters* which allow you to customise your plots.

11.1 High-level plotting commands

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

11.1.1 The *plot()* function

One of the most frequently used plotting functions in S-PLUS is the *plot()* function. This is a *generic* function: the type of plot produced is dependent on the type or *class* of the first argument.

| *plot(x,y)* | If *x* and *y* are vectors, *plot(x,y)* produces a scatterplot of *x* against *y*. The same effect can be produced by supplying one argument (second form) as either a list containing two elements *x* and *y* or a two-column matrix. |
| *plot(xy)* |
| *plot(x)* | Produces a time series plot if *x* is a numeric vector or time series object, or an Argand diagram if *x* is a complex vector. |
| *plot(f)* | *f* is a factor object, *y* is a numeric vector. The first form generates a bar plot of *f*; the second form produces boxplots of *y* for each level of *f*. |
plot(df)
plot(~ expr)
plot(y ~ expr)
df is a data frame, y is any object, expr is a list of object names separated by ‘+’ (e.g. a + b + c). The first two forms produce distributional plots of the variables in a data frame (first form) or of a number of named objects (second form). The third form plots y against every object named in expr.

11.1.2 Displaying multivariate data

S-PLUS provides two very useful functions for representing multivariate data. If X is a numeric matrix or data frame, the command

> pairs(X)

produces a pairwise scatterplot matrix of the variables defined by the columns of X, that is, every column of X is plotted against every other column of X and the resulting n(n-1) plots are arranged in a matrix with plot scales constant over the rows and columns of the matrix.

When three or four variables are involved a coplot may be more enlightening. If a and b are numeric vectors and c is a numeric vector or factor object (all of the same length), then the command

> coplot(a ~ b | c)

produces a number of scatterplots of a against b for given values of c. If c is a factor, this simply means that a is plotted against b for every level of c. When c is numeric, it is divided into a number of conditioning intervals and for each interval a is plotted against b for values of c within the interval. The number and position of intervals can be controlled with given.values= argument to coplot() — the function co.intervals() is useful for selecting intervals. You can also use two given variables with a command like

> coplot(a ~ b | c + d)

which produces scatterplots of a against b for every joint conditioning interval of c and d.

The coplot() and pairs() function both take an argument panel= which can be used to customise the type of plot which appears in each panel. The default is points() to produce a scatterplot but by supplying some other low-level graphics function of two vectors x and y as the value of panel= you can produce any type of plot you wish. An example panel function useful for coplots is panel.smooth().

11.1.3 Display graphics

Other high-level graphics functions produce different types of plots. Some examples are:

tspplot(x1,x2,...) Plots any number of time series on the same scale. This automatic simultaneous scaling feature is also useful when the x_i's are ordinary numeric vectors, in which case they are plotted against the numbers 1,2,3,....

qnorm(x) Distribution-comparison plots. The first form plots the numeric vector x against the expected Normal order scores (a normal scores plot.) The second form plots the quantiles of x against those of y to compare their respective distributions.
### 11.1 High-level plotting commands

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hist(x)</code></td>
<td>Produces a histogram of the numeric vector x. A sensible number of classes is usually chosen, but a recommendation can be given with the <code>nclass=</code> argument. Alternatively, the breakpoints can be specified exactly with the <code>breaks=</code> argument. If the <code>probability=T</code> argument is given, the bars represent relative frequencies instead of counts.</td>
</tr>
<tr>
<td><code>dotchart(x,...)</code></td>
<td>Construct a dotchart of the data in x. In a dotchart the y—axis gives a labelling of the data in x and the x—axis gives its value. For example it allows easy visual selection of all data entries with values lying in specified ranges.</td>
</tr>
<tr>
<td><code>pie(slices, names, explode=...)</code></td>
<td>Make a pie diagram, including the possibility of some pieces displaced or <code>exploded</code> out from the centre. (Pie diagrams are especially good for showing to administrators and bosses, but not much else, in my opinion.)</td>
</tr>
</tbody>
</table>

#### 11.1.4 Arguments to high-level plotting functions

There are a number of arguments which may be passed to high-level graphics functions, as follows:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add=T</code></td>
<td>Forces the function to act as a low-level graphics function, superimposing the plot on the current plot (some functions only).</td>
</tr>
<tr>
<td><code>axes=F</code></td>
<td>Suppresses generation of axes — useful for adding your own custom axes with the <code>axis()</code> function. The default, <code>axes=T</code>, means include axes.</td>
</tr>
<tr>
<td><code>log=“x”</code></td>
<td>Causes the x, y or both axes to be logarithmic. Only works for scatterplots (and variants).</td>
</tr>
<tr>
<td><code>type=“p”</code></td>
<td>Plot individual points (the default)</td>
</tr>
<tr>
<td><code>type=“l”</code></td>
<td>Plot lines</td>
</tr>
<tr>
<td><code>type=“b”</code></td>
<td>Plot points connected by lines (<em>both</em>)</td>
</tr>
<tr>
<td><code>type=“o”</code></td>
<td>Plot points overlaid by lines</td>
</tr>
<tr>
<td><code>type=“h”</code></td>
<td>Plot vertical lines from points to the zero axis (<em>high-density</em>)</td>
</tr>
<tr>
<td><code>type=“s”</code></td>
<td>Step-function plots. In the first form, the top of the vertical defines the point; in the second, the bottom.</td>
</tr>
<tr>
<td><code>type=“n”</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>xlab=“string”</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>ylab=“string”</code></td>
<td>Figure title, placed at the top of the plot in a large font.</td>
</tr>
<tr>
<td><code>sub=“string”</code></td>
<td>Sub-title, placed just below the x-axis in a smaller font.</td>
</tr>
</tbody>
</table>
11.2 Low-level plotting commands

Sometimes the high-level plotting functions don’t produce exactly the kind of plot you desire. In this case, low-level plotting commands can be used to add extra information (such as points, lines or text) to the current plot.

Some of the more useful low-level plotting functions are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>points(x,y)</code></td>
<td>Adds points or connected lines to the current plot. <code>plot()</code>’s <code>type=</code> argument can also be passed to these functions (and defaults to &quot;p&quot; for <code>points()</code> and &quot;l&quot; for <code>lines()</code>).</td>
</tr>
<tr>
<td><code>lines(x,y)</code></td>
<td>Add text to a plot at points given by <code>x</code>, <code>y</code>. Normally <code>labels</code> is an integer or character vector in which case <code>labels[i]</code> is plotted at point <code>(x[i], y[i])</code>. The default is <code>1:length(x)</code>. Note: This function is often used in the sequence <code>&gt; plot(x, y, type=&quot;n&quot;); text(x, y, names)</code> The graphics parameter <code>type=&quot;n&quot;</code> suppresses the points but sets up the axes, and the <code>text()</code> function supplies special characters, as specified by the character vector <code>names</code> for the points.</td>
</tr>
<tr>
<td><code>abline(a, b)</code></td>
<td>Adds a line of slope <code>b</code> and intercept <code>a</code> to the current plot. <code>h=y</code> may be used to specify <code>y</code>-coordinates for the heights of horizontal lines to go across a plot, and <code>v=x</code> similarly for the <code>x</code>-coordinates for vertical lines. Also <code>lm.obj</code> may be list with a <code>$coefficients</code> component of length 2 (such as the result of model-fitting functions,) which are taken as an intercept and slope, in that order.</td>
</tr>
<tr>
<td><code>polygon(x,y,</code></td>
<td>Draws a polygon defined by the ordered vertices in <code>(x,y)</code>, and (optionally) shade it in with hatch lines, or fill it if the graphics device allows the filling of figures.</td>
</tr>
<tr>
<td>...)`</td>
<td></td>
</tr>
<tr>
<td><code>legend(x,y,</code></td>
<td>Adds a legend to the current plot at the specified position. Plotting characters, line styles, colours etc. are identified with the labels in the character vector <code>legend</code>. At least one other argument <code>v</code> (a vector the same length as <code>legend</code>) with the corresponding values of the plotting unit must also be given, as follows:</td>
</tr>
<tr>
<td><code>legend(...)</code></td>
<td></td>
</tr>
<tr>
<td><code>angle=v</code></td>
<td>Shading angles</td>
</tr>
<tr>
<td><code>density=v</code></td>
<td>Shading densities</td>
</tr>
<tr>
<td><code>fill=v</code></td>
<td>Colours for filled boxes</td>
</tr>
<tr>
<td><code>col=v</code></td>
<td>Colours in which points or lines will be drawn</td>
</tr>
<tr>
<td><code>lty=v</code></td>
<td>Line styles</td>
</tr>
<tr>
<td><code>pch=v</code></td>
<td>Plotting characters (character vector)</td>
</tr>
<tr>
<td><code>marks=v</code></td>
<td>Plotting symbols, as obtained when using a numeric argument to <code>pch=</code> (numeric vector).</td>
</tr>
</tbody>
</table>
11.3 Interactive graphics functions

**title(main,sub)** Adds a title `main` to the top of the current plot in a large font and optionally a sub-title `sub` at the bottom in a smaller font.

**axis(side,...)** Adds an axis to the current plot on the side given by the first argument (1 to 4, counting clockwise from the bottom.) Other arguments control the positioning of the axis within or beside the plot, and tick positions and labels. Useful for adding custom axes after calling `plot()` with the `axes=F` argument.

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 given in terms of `user coordinates` which are defined by the previous high-level graphics command and are chosen based on the supplied data. Where `x` and `y` arguments are required, it is also sufficient to supply a single argument being a list with elements named `x` and `y`. Similarly a matrix with two columns is also valid input. In this way functions such as `locator()` (see below) may be used to specify positions on a plot interactively.

11.3 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:

**locator(n,type)** Waits for the user to select locations on the current plot using the left mouse button. This continues until `n` (default 500) points have been selected, or the middle mouse button is pressed. 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`.

`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. For example, to place some informative text near an outlying point, the command

```
> text(locator(1), "Outlier", adj=0)
```

may be useful. `locator()` will still work if the current device does not support a mouse; in this case the user will be prompted for `x` and `y` coordinates.

**identify(x,y,labels)** Allow the user to highlight any of the points defined by `x` and `y` (using the left mouse button) by plotting the corresponding component of `labels` nearby (or the index number of the point if `labels` is absent). Returns the indices of the selected points when the middle button is pressed.
Sometimes we want to identify particular points on a plot, rather than their positions. For example, we may wish the user to select some observation of interest from a graphical display and then manipulate that observation in some way. Given a number of \((x,y)\) coordinates in two numeric vectors \(x\) and \(y\), we could use the \texttt{identify()} function as follows:

\[
\begin{align*}
&> \text{plot}(x,y) \\
&> \text{identify}(x,y)
\end{align*}
\]

The \texttt{identify()} function performs no plotting itself, but simply allows the user to move the mouse pointer and click the left mouse button near a point. The point nearest the mouse pointer will be highlighted with its index number (that is, its position in the \(x/y\) vectors) plotted nearby. Alternatively, you could use some informative string (such as a case name) as a highlight by using the \texttt{labels} argument to \texttt{identify()}, or disable highlighting altogether with the \texttt{plot=} \texttt{F} argument. When the middle button is pressed, \texttt{identify()} returns the indices of the selected points; you can use these indices to extract the selected points from the original vectors \(x\) and \(y\).

### 11.4 Using graphics parameters

When creating graphics, particularly for presentation or publication purposes, S-PLUS does not always produce exactly what is required. You can, however, customise almost every aspect of the display using \textit{graphics parameters}. S-PLUS maintains a list of a large number of graphics parameters which control things such as line style, colours, figure arrangement and text justification among many others. Every \textit{graphics parameter} has a name (such as ‘\textit{col’}, which controls colours,) and a value (a colour 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 initialised. 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.

#### 11.4.1 Permanent changes: The \texttt{par()} function

The \texttt{par()} function is used to access and modify the list of graphics parameters for the current graphics device.

\[
\begin{align*}
\text{\texttt{par()}} & \quad \text{Without arguments, returns a list of all graphics parameters and their values for the current device.} \\
\text{\texttt{par(c("col", "lty"))}} & \quad \text{With a character vector argument, returns only the named graphics parameters (again, as a list.)} \\
\text{\texttt{par(col=4,lty=2)}} & \quad \text{With named arguments (or a single list argument), sets the values of the named graphics parameters, and returns the original values of the parameters as a list.}
\end{align*}
\]

Setting graphics parameters with the \texttt{par()} function changes the value of the parameters \textit{permanently}, in the sense that all future calls to graphics functions (on the current device) will be affected by the new value. You can think of setting graphics parameters in this way as setting ‘default’ values for the parameters, which will be used by all graphics functions unless an alternative value is given.
Note that calls to \texttt{par()} \emph{always} affect the global values of graphics parameters, even when \texttt{par()} is called from within a function. This is often undesirable behaviour — usually we want to set some graphics parameters, do some plotting, and then restore the original values so as not to affect the user’s S-PLUS session. You can restore the initial values by saving the result of \texttt{par()} when making changes, and restoring the initial values when plotting is complete.

\begin{verbatim}
> oldpar <- par(col=4,lty=2)
...plotting commands ...
> par(oldpar)
\end{verbatim}

### 11.4.2 Temporary changes: Arguments to graphics functions

Graphics parameters may also be passed to (almost) any graphics function as named arguments. This has the same effect as passing the arguments to the \texttt{par()} function, except that the changes only last for the duration of the function call. For example:

\begin{verbatim}
> plot(x,y,pch="*")
\end{verbatim}

produces a scatterplot using a plus sign as the plotting character, without changing the default plotting character for future plots.

### 11.5 Graphics parameters list

The following sections detail many of the commonly-used graphical parameters. The S-PLUS help documentation for the \texttt{par()} function provides a more concise summary; this is provided as a somewhat more detailed alternative.

Graphics parameters will be presented in the following form:

\begin{verbatim}
name=value
\end{verbatim}

\texttt{name} is the name of the parameter, \texttt{value} is a typical value you might use when setting the parameter.

<table>
<thead>
<tr>
<th>name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pch=&quot;*&quot;</td>
<td>Character to be used for plotting points. The default varies with graphics drivers, but it is usually ‘*’ for terminals or window devices, and ‘•’ for \texttt{PostScript} devices. Plotted points tend to appear slightly above or below the appropriate position unless you use ‘.’ as the plotting character, which produces centred points.</td>
</tr>
<tr>
<td>pch=4</td>
<td>When \texttt{pch} is given as an integer between 0 and 18 inclusive, a specialised plotting symbol is produced. To see what the symbols are, use the command \texttt{legend(locator(1), as.character(0:18), marks=0:18)}</td>
</tr>
</tbody>
</table>
\textbf{lty=2} \hspace{1cm} \text{Line types. Alternative line styles are not supported on all graphics devices (and vary on those that do) but line type 1 is always a solid line, and line types 2 and onwards are dotted or dashed lines, or some combination of both.}

\textbf{lw=2} \hspace{1cm} \text{Line widths. Desired width of lines, in multiples of the 'standard' line width. Affects axis lines as well as lines drawn with \texttt{lines()}, etc.}

\textbf{col=2} \hspace{1cm} \text{Colours to be used for points, lines, text, filled regions and images. Each of these graphic elements has a list of possible colours, and the value of this parameter is an index to that list. Obviously, this parameter applies only to a limited range of devices.}

\textbf{font=2} \hspace{1cm} \text{Font to use for text. The appropriate value of this parameter is dependent on the graphics device being used; for the \texttt{postscript()} device this is an index to the system dataset \texttt{ps.fonts}.}

\textbf{adj=-0.1} \hspace{1cm} \text{Justification of text relative to the plotting position. 0 means left justify, 1 means right justify and 0.5 means to centre horizontally about the plotting position. The actual value is the proportion of text that appears to the left of the plotting position, so a value of -0.1 leaves a gap of 10\% of the text width between the text and the plotting position.}

\textbf{cex=1.5} \hspace{1cm} \text{Character expansion. The value is the desired size of text characters (including plotting characters) relative to the default text size.}

### 11.5.2 Axes and tick marks

Many of S-PLUS’s high-level plots have axes, and you can construct axes yourself with the low-level \texttt{axis()} graphics function. Axes have three main components: the \textit{axis line} (line style controlled by the \texttt{lty} graphics parameter), the \textit{tick marks} (which mark off unit divisions along the axis line) and the \textit{tick labels} (which mark the units.) These components can be customised with the following graphics parameters.

\textbf{lab=c(5,7,12)} \hspace{1cm} \text{The first two numbers are the desired number of tick intervals on the \textit{x} and \textit{y} axes respectively. The third number is the desired length of axis labels, in characters (including the decimal point.) Choosing a too-small value for this parameter may result in all tick labels being rounded to the same number!}

\textbf{las=1} \hspace{1cm} \text{Orientation of axis labels. 0 means always parallel to axis, 1 means always horizontal, and 2 mean always perpendicular to the axis.}

\textbf{mgp=c(3,1,0)} \hspace{1cm} \text{Positions of axis components. The first component is the distance from the axis label to the axis position, in text lines. The second component is the distance to the tick labels, and the final component is the distance from the axis position to the axis line (usually zero). Positive numbers measure outside the plot region, negative numbers inside.}
11.5 Graphics parameters list

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tck=0.01</td>
<td>Length of tick marks, as a fraction of the size of the plotting region. When tck is small (less than 0.5) the tick marks on the x and y axes are forced to be the same size. A value of 1 gives grid lines. Negative values give tick marks outside the plotting region. Use tck=0.01 and mgp=c(1,-1.5,0) for internal tick marks.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xaxs=&quot;a&quot;</td>
<td>Axis styles for the x and y axes, respectively. With styles &quot;a&quot; (standard) and &quot;e&quot; (extended) the smallest and largest tick marks always lie outside the range of the data. Extended axes may be widened slightly if any points are very near the edge. This style of axis can sometimes leave large blank gaps near the edges. With styles &quot;i&quot; (internal) and &quot;x&quot; (the default) tick marks always fall within the range of the data, however style &quot;x&quot; leaves a small amount of space at the edges. Setting this parameter to &quot;d&quot; (direct axis) locks in the current axis and uses it for all future plots (or until the parameter is set to one of the other values above, at least.) Useful for generating series of fixed-scale plots.</td>
</tr>
</tbody>
</table>

11.5.3 Figure margins

A single plot in S-PLUS is known as a figure and comprises a plot region surrounded by margins (possibly containing axis labels, titles, etc.) and (usually) bounded by the axes themselves. A typical figure appears in Figure 10. Graphics parameters controlling figure layout include:

![Figure 10: Anatomy of an S figure](image-url)
maic(1.0.5,0.5,0)  Widths of the bottom, left, top and right margins, respectively, measured in inches.

mar=c(4,2,2,1)  Similar to mai, except the measurement unit is text lines.

mar and mai are equivalent in the sense that setting one changes the value of the other. The default values chosen for this parameter are often too large; the right-hand margin is rarely needed, and neither is the top margin if no title is being used. The bottom and left margins must be large enough to accommodate the axis and tick labels. Furthermore, the default is chosen without regard to the size of the device surface: for example, using the postscript() driver with the height=4 argument will result in a plot which is about 50% margin unless mar or mai are set explicitly. When multiple figures are in use (see below) the margins are reduced by half, however this may not be enough when many figures share the same page.

11.5.4 Multiple figure environment

S-PLUS allows you to create an n x m array of figures on a single page. Each figure has its own margins, and the array of figures is optionally surrounded by an outer margin as shown in Figure 11.

The graphical parameters relating to multiple figures are as follows:

mfcol=c(3,2)  Set size of multiple figure array. The first value is the number of rows; the second is the number of columns. The only difference between these two parameters is that setting mfcol causes figures to be filled by column; mfrow fills by rows. The arrangement in Figure 11 would have been created by setting mfrow=c(3,2); the figure shows the page after four plots have been drawn.

mfg=c(2,2,3,2)  Position of current figure in a multiple figure environment. The first two numbers are the row and column of the current figure; the last two are the number of rows and columns in the multiple figure array. Set this parameter to jump between figures in the array. You can even use different values for the last two numbers than the true values for unequally-sized figures on the same page.

fig=c(4.9,1,4)/10  Position of the current figure on the page. Values are the positions of the left, right, bottom and top edges respectively, as a percentage of the page measured from the bottom left corner. The example value would be for a figure in the bottom right of the page. Set this parameter for arbitrary positioning of figures within a page.

oma=c(2,0,3,0)  Size of outer margins. Like mar and mai, the first measures in text lines and the second in inches, starting with the bottom margin and working clockwise.

Outer margins are particularly useful for page-wise titles, etc. Text can be added to the outer margins with the mtext() function with argument outer=T. There are no outer margins by default, however, so you must create them explicitly using oma or omi.
11.6 Device drivers

S-PLUS can generate graphics (of varying levels of quality) on almost any type of display or printing device. Before this can begin, however, S-PLUS needs to be informed what type of device it is dealing with. This is done by starting a device driver. The purpose of a device driver is to convert graphical instructions from S-PLUS (‘draw a line,’ for example) into a form that the particular device can understand.

Device drivers are started by calling a device driver function. There is one such function for every device driver: type help(Devices) for a list of them all. For example, issuing the command

> postscript()

causes all future graphics output to be sent to the printer in PostScript format. Some commonly-used device drivers are:
motif() For use with the X11 or Open Windows window systems.
openlook() X11()
suntools() For use with the SunView windowing system.

postscript() For printing on PostScript printers, or creating PostScript graphics files.
printer() crt() For terminals with little or no graphics capabilities. ASCII-based graphics are generated.

When you have finished with a device, be sure to terminate the device driver by issuing the command

> dev.off()

This ensures that the device finishes cleanly; for example in the case of hardcopy devices this ensures that every page is completed and has been sent to the printer.

11.6.1 PostScript diagrams for typeset documents.

By passing the file argument to the postscript() device driver function, you may store the graphics in PostScript format in a file of your choice. The plot will be in portrait orientation unless the horizontal=T argument is given, and you can control the size of the graphic with the width and height arguments (the plot will be scaled as appropriate to fit these dimensions.) For example, the command

> postscript("file.ps", height=4)

will produce a file containing PostScript code for a figure four inches high, perhaps for inclusion in a document.\(^9\) It is important to note that if the file named in the command already exists, it will be overwritten. This is the case even if the file was only created earlier in the same S-PLUS session.

11.6.2 Multiple graphics devices

In advanced use of S-PLUS it is often useful to have several graphics devices in use at the same time. Of course only one graphics device can accept graphics commands at any one time, and this is known as the current device. When multiple devices are open, they form a numbered sequence with names giving the kind of device at any position.

The main commands used for operating with multiple devices, and their meanings are as follows:

motif() Each new call to a device driver function opens a new graphics device, thus extending by one the device list. This device becomes the current device, to which graphics output will be sent.
postscript() ...

dev.list() returns the number and name of all active devices. The device at position 1 on the list is always the null device which does not accept graphics commands at all.

\(^9\)Warning: The PostScript code produced by the postscript() device driver is not Encapsulated PostScript, and thus including it in a document electronically (as opposed to physical cut-and-paste) can be rather problematic. For this type of application, a better solution is to use the fig() driver (available from statlib) and use a conversion program, such as fig2dev, to convert the resultant fig code to Encapsulated PostScript.
### 11.6 Device drivers

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dev.next()</td>
<td>returns the number and name of the graphics device next to, or previous to the current device, respectively.</td>
</tr>
<tr>
<td>dev.prev()</td>
<td></td>
</tr>
<tr>
<td>dev.set(which=k)</td>
<td>can be used to change the current graphics device to the one at position k of the device list. Returns the number and label of the device.</td>
</tr>
<tr>
<td>dev.off(k)</td>
<td>Terminate the graphics device at point k of the device list. For some devices, such as postscript devices, this will either print the file immediately or correctly complete the file for later printing, depending on how the device was initiated.</td>
</tr>
<tr>
<td>dev.copy(device, ..., which=k)</td>
<td>Make a copy of the device k. Here device is a device function, such as postscript, with extra arguments, if needed, specified by .... dev.print is similar, but the copied device is immediately closed, so that end actions, such as printing hardcopies, are immediately performed. (See also printgraph()).</td>
</tr>
<tr>
<td>graphics.off()</td>
<td>Terminate all graphics devices on the list, except the null device.</td>
</tr>
</tbody>
</table>
# A S-PLUS: An introductory session

The following session is intended to introduce to you some features of the S-PLUS environment by using them. Many features of the system will be unfamiliar and puzzling at first, but this will soon disappear.

<table>
<thead>
<tr>
<th>login:</th>
<th>Login, start your windowing system (ask a demonstrator if you need help), and check that your working directory has a subdirectory .Data, which in turn contains the files .First, .Last and possibly .Audit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>You should also have the file morley.data in your working directory. If not, see a demonstrator. If you have, proceed.</td>
</tr>
<tr>
<td>&gt; ls -a</td>
<td>Start Splus with the inbuilt command line editor enabled.</td>
</tr>
<tr>
<td>&gt; ls -a .Data</td>
<td>The S-PLUS program begins, with a banner.</td>
</tr>
<tr>
<td>help.start()</td>
<td>(Within S-PLUS the prompt on the left hand side will not be shown to avoid confusion.)</td>
</tr>
<tr>
<td>motif()</td>
<td>Start the help facility. You should briefly explore the features of this facility with the mouse. Standard X window conventions apply.</td>
</tr>
<tr>
<td>x &lt;- rnorm(50)</td>
<td>Iconify the help window and move on to the next part.</td>
</tr>
<tr>
<td>y &lt;- rnorm(x)</td>
<td>Turn on the graphics window. You may need to re-position and re-size to make it convenient to work with both windows.</td>
</tr>
<tr>
<td>hull &lt;- hull(x, y)</td>
<td></td>
</tr>
<tr>
<td>plot(x, y)</td>
<td></td>
</tr>
<tr>
<td>polygon(x[hull], y[hull], dens=15)</td>
<td></td>
</tr>
<tr>
<td>objects()</td>
<td></td>
</tr>
<tr>
<td>rm(x,y,hull)</td>
<td></td>
</tr>
</tbody>
</table>

| x <- 1:20 | Generate two pseudo random normal vectors of x- and y-coordinates. |
| w <- 1 + sqrt(x)/2 | Find their convex hull in the plane. |
| dummy <- data.frame(x=x, y=x + rnorm(x)*w) | Plot the points in the plane, and mark in their convex hull. |
| dummy | |
| fm <- lm(y~x, data=dummy) | See which S-PLUS objects are now in the .Data directory. |
| summary(fm) | Remove objects no longer needed. (Clean up). |
| f1m <- lm(y~x, data=dummy, weight=1/w^2) | |
| summary(f1m) | |
| lrf <- loess(y~x, dummy) | |
| attach(dummy) | |
| plot(x, y) | |
| lines(x, fitted(lrf)) | Make a nonparametric local regression function. |
| Standard point plot. | Make the columns in the data frame visible as variables. |
| Add in the local regression. |
abline(0, 1, lty=3)  The true regression line: (intercept 0, slope 1).
abline(coef(fm))  Unweighted regression line.
abline(coef(fm1), lty=4)  Weighted regression line.

detach()  At any time you can make a hardcopy of the graphics window
plot(fitted(fm), by clicking on the Graph section of the window and selecting
resid(fm), the Print option.
xlab="Fitted values", A standard regression diagnostic plot to check for heteroscedasticity. Can you see it?
ylab="Residuals", main= "Residuals vs Fitted")
qqnorm(resid(fm), main= "Residuals Rankit Plot")  A normal scores plot to check for skewness, kurtosis and outliers. (Not very useful here.)
rm(fm, fm1, lrf, x, dummy)  Clean up again.

!more morley.data  The next section will look at data from the classical experiment
mm <- read.table( of Michelson and Morley to measure the speed of light.
"morley.data")  Optional. Temporarily interrupt S-PLUS and look at the file.  
mm  This is a standard way to escape to the operating system.
attach(mm, 1)  Read in the Michelson and Morley data as a data frame, and
objects()  look at it. There are five experiments (col. Expt) and each has
Expt <- factor(Expt)  20 runs (col. Run) and s1 is the recorded speed of light, suitably
Run <- factor(Run)  coded.
detach(1, save="mm")  Place mm on the top of the search list, (position 1).
attach(mm)  Change Expt and Run into factors.
plot(Expt, Speed, main= "Speed of Light Data",
"Experiment No.")  Compare the five experiments with simple boxplots.
fm <- aov(Speed~Run+Expt, data=mm)  Analyse as a randomized block, with ‘runs’ and ‘experiments’
summary(fm)  as factors.
fm0 <- update(fm, 
."~.Run)  Fit the sub-model omitting ‘runs’, and compare using a formal
anova(fm0, fm)  analysis of variance.
detach()  Clean up before moving on.
rm(fm, fm0)

x <- seq(-pi, pi, len=50)  We now look at some more graphical features: contour and
y <- x  3-dimensional perspective plots.
x is a vector of 50 equally spaced values in \(-\pi \leq x \leq \pi\).  y is the same.
f <- outer(x, y, function(x,y) cos(y)/(1+x^2))

oldpar <- par()
par(pty="s")
contour(x, y, f)
contour(x, y, f, nint=15, add=T)
fa <- (f-t(f))/2
contour(x, y, fa, nint=15)

par(oldpar)
persp(x, y, f)
persp(x, y, fa)
image(x, y, f)
image(x, y, fa)

objects(); rm(x,y,f,fa) and clean up before moving on.

th <- seq(-pi, pi, len=100)
z <- exp(1i*th)

par(pty="s")
plot(z, type="l")
w <- rnorm(100) + rnorm(100)*1i

w <- ifelse(Mod(w) > 1, 1/w, w)
plot(w, xlab="x", ylab="y")
lines(z)
w <- sqrt(runif(100))*exp(2*pi*runif(100)*1i)
plot(w, xlab="x", ylab="y")
lines(z)
rm(th,w,z)

par(oldpar)

S-PLUS can do complex arithmetic, also. 1i is used for the complex number i

Plotting complex arguments means plot imaginary versus real parts. This should be a circle.

Suppose we want to sample points within the unit circle. One method would be to take complex numbers with standard normal real and imaginary parts...

and to map any outside the circle onto their reciprocal.

All points are inside the unit circle, but the distribution is not uniform.

The second method uses the uniform distribution. The points should now look more evenly spaced over the disc.

Clean up again.

par(oldpar)

An old favourite. Take a hardcopy if you wish.

Clean up again.

Quit the S-PLUS program...

... and return to UNIX.
B The Inbuilt Command Line Editor in S-PLUS

B.1 Preliminaries

The August 1991 release of S-PLUS has inbuilt command line editor that allows recall, editing and re-submission of prior commands.

To use it, start the S-PLUS program with

$ Splus -e

Inside the editor either emacs or vi conventions are available, according to the shell environment variable $SLEDITOR. To get the emacs conventions use (in csh and variants)

$ setenv SLEDITOR emacs

and for the vi conventions to apply, put vi instead of emacs. This statement would normally be included in your .login file (or equivalent) and would then be done automatically at login time. To avoid forgetting to include the -e a handy alias for your .cshrc file is, say

alias S+ '"Splus -e'

after which S+ is the command to start S-PLUS with command line editor.

The usual typographical conventions apply: "M means Hold the Control down while you press the m key, but Esc m means First press the Esc key and then the m key. Note that case is significant after Esc.

B.2 Editing Actions

The S-PLUS program keeps a history of the commands you type, including the error lines, and commands in your history may be recalled, changed if necessary, and re-submitted as new commands. In emacs style command line editing any straight typing you do while in this editing phase causes the characters to be inserted in the command you are editing, displacing any characters to the right of the cursor. In vi mode character insertion mode is started by Esc i or Esc a, characters are typed and insertion mode is finished by typing a further Esc .

Pressing the Return command at any time causes the command to be re-submitted.

Other editing actions are summarised in the following table.

Unfortunately it does not seem to be possible to bind the motion keys, for example, to the arrow keys, which is something of a nuisance.
B.3 Command Line Editor Summary

1. Command recall and vertical motion

<table>
<thead>
<tr>
<th>emacs style</th>
<th>vi style</th>
</tr>
</thead>
<tbody>
<tr>
<td>^P</td>
<td>Esc k</td>
</tr>
<tr>
<td>^N</td>
<td>Esc j</td>
</tr>
<tr>
<td>^R text</td>
<td>Esc ? text</td>
</tr>
</tbody>
</table>

2. Horizontal motion of the cursor

<table>
<thead>
<tr>
<th>emacs style</th>
<th>vi style</th>
</tr>
</thead>
<tbody>
<tr>
<td>^A</td>
<td>Esc ^</td>
</tr>
<tr>
<td>^E</td>
<td>Esc $</td>
</tr>
<tr>
<td>Esc b</td>
<td>Esc b</td>
</tr>
<tr>
<td>Esc f</td>
<td>Esc w</td>
</tr>
<tr>
<td>^B</td>
<td>Esc h</td>
</tr>
<tr>
<td>^F</td>
<td>Esc l</td>
</tr>
</tbody>
</table>

3. Editing and re-submission

<table>
<thead>
<tr>
<th>emacs style</th>
<th>vi style</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Esc i text Esc</td>
</tr>
<tr>
<td>^R text</td>
<td>Esc a text Esc</td>
</tr>
<tr>
<td>Delete</td>
<td>Esc shift-x</td>
</tr>
<tr>
<td>^D</td>
<td>Esc x</td>
</tr>
<tr>
<td>Esc d</td>
<td>Esc dw</td>
</tr>
<tr>
<td>^K</td>
<td>Esc shift-d</td>
</tr>
<tr>
<td>^Y</td>
<td>Esc shift-y</td>
</tr>
<tr>
<td>^T</td>
<td>Esc xp</td>
</tr>
<tr>
<td>Esc l</td>
<td></td>
</tr>
<tr>
<td>Esc c</td>
<td>Return</td>
</tr>
<tr>
<td>Return</td>
<td>Return</td>
</tr>
</tbody>
</table>

NOTE: With vi style commands the Esc need only be issued before the first recall command, and to terminate insert and append commands, as is usual in vi.

The final Return terminates the command line editing sequence for commands of either style.
C Exercises

C.1 The cloud point data

Source: Draper & Smith, *Applied Regression Analysis*, p. 162
Category: Polynomial regression. Simple plots.

Description

The cloud point of a liquid is a measure of the degree of crystallization in a stock that can be measured by the refractive index. It has been suggested that the percentage of $I_8$ in the base stock is an excellent predictor of cloud point using the second or third order model:

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + E, \quad E \sim N(0, \sigma^2)$$

Data

The following data was collected on stocks with known percentage of $I_8$:

<table>
<thead>
<tr>
<th>$I_8$</th>
<th>Cloud Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>21.9</td>
</tr>
<tr>
<td>0</td>
<td>22.1</td>
</tr>
<tr>
<td>0</td>
<td>22.8</td>
</tr>
<tr>
<td>1</td>
<td>24.5</td>
</tr>
<tr>
<td>2</td>
<td>26.0</td>
</tr>
<tr>
<td>5</td>
<td>28.9</td>
</tr>
<tr>
<td>6</td>
<td>29.8</td>
</tr>
<tr>
<td>6</td>
<td>30.0</td>
</tr>
<tr>
<td>6</td>
<td>30.3</td>
</tr>
<tr>
<td>7</td>
<td>30.4</td>
</tr>
<tr>
<td>8</td>
<td>31.4</td>
</tr>
<tr>
<td>8</td>
<td>31.5</td>
</tr>
<tr>
<td>9</td>
<td>31.8</td>
</tr>
<tr>
<td>10</td>
<td>33.1</td>
</tr>
</tbody>
</table>

The data may be read from file cloud.data in a form suitable to construct a data frame.

Suggested analysis

Fit polynomial regression models using the lm() function, and choose the degree carefully.

C.2 The Janka hardness data

Category: Polynomial regression. Transformations.

Description

The Janka hardness is an important structural property of Australian timbers, which is difficult to measure. It is, however, related to the density of the timber, which is relatively easy to measure. A low degree polynomial regression is suggested as appropriate.

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + E$$

where $Y$ is the hardness and $x$ the density.
Data

The following data comes from samples of 36 Australian Eucalypt hardwoods.

<table>
<thead>
<tr>
<th>D</th>
<th>H</th>
<th>D</th>
<th>H</th>
<th>D</th>
<th>H</th>
<th>D</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.7</td>
<td>484</td>
<td>30.3</td>
<td>587</td>
<td>39.4</td>
<td>1210</td>
<td>42.9</td>
<td>1270</td>
</tr>
<tr>
<td>24.8</td>
<td>427</td>
<td>32.7</td>
<td>704</td>
<td>39.9</td>
<td>989</td>
<td>45.8</td>
<td>1180</td>
</tr>
<tr>
<td>27.3</td>
<td>413</td>
<td>35.6</td>
<td>979</td>
<td>40.3</td>
<td>1160</td>
<td>46.9</td>
<td>1400</td>
</tr>
<tr>
<td>28.4</td>
<td>517</td>
<td>38.5</td>
<td>914</td>
<td>40.6</td>
<td>1010</td>
<td>48.2</td>
<td>1760</td>
</tr>
<tr>
<td>28.4</td>
<td>549</td>
<td>38.8</td>
<td>1070</td>
<td>40.7</td>
<td>1100</td>
<td>51.5</td>
<td>1710</td>
</tr>
<tr>
<td>29.0</td>
<td>648</td>
<td>39.3</td>
<td>1020</td>
<td>40.7</td>
<td>1130</td>
<td>51.5</td>
<td>2010</td>
</tr>
</tbody>
</table>

The data may be read as a data frame from file janka.data.

Suggested analysis

Fit polynomial regression models, choosing the degree carefully. Examine the residuals and see if the data has any obvious outliers or heteroscedasticity. Check to see what effect a square root or log transformation has on the residual pattern when plotted against fitted values.

More advanced: Consider a quasi-likelihood model with variance proportional to the mean and a square root link.

C.3 The Tuggeranong house price data

Source: Dr Ray Correll, Personal communication
Category: Multiple regression, coplots.

Description

Before buying a house in Tuggeranong in February, 1987, a cautious potential householder collected some data on houses on the market. The data for 20 such houses is shown in the table and is available as the file house.dat. The variables collected are, in order, price, total floor area, block area, number of main rooms, age of house and whether or not the house was centrally heated.

Data

The data is given in Table 5 and is available as the file house.data.

Suggested analysis

Explore the data with coplot() using Age and CentHeat as conditioning variables.

Choose a multiple regression model carefully and check for outliers, (that is, for “bargains” and “rip-offs”).
### C.4 Yorke Penninsula wheat yield data

**Source:** K. W. Morris (private communication)

**Category:** Multiple regression.

**Description**

The annual yield of wheat in a marginal wheat growing district on the Yorke Penninsula, South Australia, together with the rainfall for the three growing months, for the years 1931–1955. The year itself is potentially a surrogate predictor to allow for improvements in varieties and farm practice. Yield is in bushels per acre, and rainfall is in inches.

**Data**

The data is given in Table 6 and may be read as a data frame from file `sawheat.data`.

**Suggested analysis**

Fit a multiple regression model and check the results, via residual plots especially.
<table>
<thead>
<tr>
<th>Year</th>
<th>Rain0</th>
<th>Rain1</th>
<th>Rain2</th>
<th>Yield</th>
<th>Year</th>
<th>Rain0</th>
<th>Rain1</th>
<th>Rain2</th>
<th>Yield</th>
</tr>
</thead>
<tbody>
<tr>
<td>1931</td>
<td>.05</td>
<td>1.61</td>
<td>3.52</td>
<td>.31</td>
<td>1944</td>
<td>3.30</td>
<td>4.19</td>
<td>2.11</td>
<td>4.60</td>
</tr>
<tr>
<td>1932</td>
<td>1.15</td>
<td>.60</td>
<td>3.46</td>
<td>.00</td>
<td>1945</td>
<td>.44</td>
<td>3.41</td>
<td>1.55</td>
<td>.35</td>
</tr>
<tr>
<td>1933</td>
<td>2.22</td>
<td>4.94</td>
<td>3.06</td>
<td>5.47</td>
<td>1946</td>
<td>.50</td>
<td>3.26</td>
<td>1.20</td>
<td>.00</td>
</tr>
<tr>
<td>1934</td>
<td>1.19</td>
<td>11.26</td>
<td>4.91</td>
<td>16.73</td>
<td>1947</td>
<td>.18</td>
<td>1.52</td>
<td>1.80</td>
<td>.00</td>
</tr>
<tr>
<td>1935</td>
<td>1.40</td>
<td>10.95</td>
<td>4.23</td>
<td>10.54</td>
<td>1948</td>
<td>.80</td>
<td>3.25</td>
<td>3.55</td>
<td>2.98</td>
</tr>
<tr>
<td>1936</td>
<td>2.96</td>
<td>4.96</td>
<td>.11</td>
<td>5.89</td>
<td>1949</td>
<td>7.08</td>
<td>5.93</td>
<td>.93</td>
<td>11.89</td>
</tr>
<tr>
<td>1937</td>
<td>2.68</td>
<td>.67</td>
<td>2.17</td>
<td>.03</td>
<td>1950</td>
<td>2.54</td>
<td>4.71</td>
<td>2.51</td>
<td>6.56</td>
</tr>
<tr>
<td>1938</td>
<td>3.66</td>
<td>8.49</td>
<td>11.95</td>
<td>16.03</td>
<td>1951</td>
<td>1.08</td>
<td>3.37</td>
<td>4.02</td>
<td>1.30</td>
</tr>
<tr>
<td>1939</td>
<td>5.15</td>
<td>3.60</td>
<td>2.18</td>
<td>6.57</td>
<td>1952</td>
<td>.22</td>
<td>3.24</td>
<td>4.93</td>
<td>.03</td>
</tr>
<tr>
<td>1940</td>
<td>6.44</td>
<td>2.69</td>
<td>1.37</td>
<td>8.43</td>
<td>1953</td>
<td>.55</td>
<td>1.78</td>
<td>1.97</td>
<td>.00</td>
</tr>
<tr>
<td>1941</td>
<td>2.01</td>
<td>6.88</td>
<td>.92</td>
<td>8.68</td>
<td>1954</td>
<td>1.65</td>
<td>3.22</td>
<td>1.65</td>
<td>3.09</td>
</tr>
<tr>
<td>1942</td>
<td>.73</td>
<td>3.30</td>
<td>3.97</td>
<td>2.49</td>
<td>1955</td>
<td>.72</td>
<td>3.42</td>
<td>3.31</td>
<td>2.72</td>
</tr>
<tr>
<td>1943</td>
<td>2.52</td>
<td>1.93</td>
<td>1.16</td>
<td>.98</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Yorke Penninsula wheat yield data

C.5 The Iowa wheat yield data

Category: Multiple regression; diagnostics.

Description

The data gives the pre-season and three growing months’ precipitation, the mean temperatures for the three growing months and harvest month, the year, and the yield of wheat for the USA state of Iowa, for the years 1930–1962.

Data

The data is given in Table 7 and may be read as a data frame from file iowheat.data.

Suggested analysis

Fit a multiple regression model and select carefully the predictors. Work either by backward elimination of forward selection. Examine the residuals by plotting them in turn against each predictor variable.

Consider the effect of adding quadratic terms in the predictors.

It is interesting to compare this set of data with the Yorke Penninsula data for a similar period.

C.6 The gasoline yield data

Source: Estimate gasoline yields from crudes by Nilon H. Prater, Petroleum Refiner, 35, #5.
Category: Analysis of variance, covariance, and multiple regression.
Table 7: The Iowa historical wheat yield data

Modern regression.

Description

The data gives the gasoline yield as a percent of crude oil, say $y$, and four independent variables which may influence yield. These are

$x_1$: The crude oil gravity, in °API,
\( x_2 \): The crude oil vapour pressure,
\( x_3 \): The crude oil 10\% point, ASTM,
\( x_4 \): The gasoline end point.

The data comes as 10 separate samples, and within each sample the values for \( x_1, x_2, \) and \( x_3 \) are constant.

**Data**

The data is shown in Table 8, and is available as the file oil.data in a form suitable for constructing a data frame.

<table>
<thead>
<tr>
<th>Sample</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31.8</td>
<td>0.2</td>
<td>316</td>
<td>365</td>
<td>8.5</td>
</tr>
<tr>
<td>1</td>
<td>31.8</td>
<td>0.2</td>
<td>316</td>
<td>379</td>
<td>14.7</td>
</tr>
<tr>
<td>1</td>
<td>31.8</td>
<td>0.2</td>
<td>316</td>
<td>428</td>
<td>18.0</td>
</tr>
<tr>
<td>2</td>
<td>32.2</td>
<td>2.4</td>
<td>284</td>
<td>351</td>
<td>14.0</td>
</tr>
<tr>
<td>2</td>
<td>32.2</td>
<td>2.4</td>
<td>284</td>
<td>424</td>
<td>23.2</td>
</tr>
<tr>
<td>3</td>
<td>32.2</td>
<td>5.2</td>
<td>236</td>
<td>267</td>
<td>10.0</td>
</tr>
<tr>
<td>3</td>
<td>32.2</td>
<td>5.2</td>
<td>236</td>
<td>360</td>
<td>24.8</td>
</tr>
<tr>
<td>3</td>
<td>32.2</td>
<td>5.2</td>
<td>236</td>
<td>402</td>
<td>31.7</td>
</tr>
<tr>
<td>4</td>
<td>38.1</td>
<td>1.2</td>
<td>274</td>
<td>285</td>
<td>5.0</td>
</tr>
<tr>
<td>4</td>
<td>38.1</td>
<td>1.2</td>
<td>274</td>
<td>365</td>
<td>17.6</td>
</tr>
<tr>
<td>4</td>
<td>38.1</td>
<td>1.2</td>
<td>274</td>
<td>444</td>
<td>32.1</td>
</tr>
<tr>
<td>5</td>
<td>38.4</td>
<td>6.1</td>
<td>220</td>
<td>235</td>
<td>6.9</td>
</tr>
<tr>
<td>5</td>
<td>38.4</td>
<td>6.1</td>
<td>220</td>
<td>300</td>
<td>15.2</td>
</tr>
<tr>
<td>5</td>
<td>38.4</td>
<td>6.1</td>
<td>220</td>
<td>365</td>
<td>26.0</td>
</tr>
<tr>
<td>5</td>
<td>38.4</td>
<td>6.1</td>
<td>220</td>
<td>410</td>
<td>33.6</td>
</tr>
<tr>
<td>6</td>
<td>40.0</td>
<td>6.1</td>
<td>217</td>
<td>212</td>
<td>7.4</td>
</tr>
<tr>
<td>6</td>
<td>40.0</td>
<td>6.1</td>
<td>217</td>
<td>272</td>
<td>18.2</td>
</tr>
<tr>
<td>6</td>
<td>40.0</td>
<td>6.1</td>
<td>217</td>
<td>340</td>
<td>30.4</td>
</tr>
<tr>
<td>7</td>
<td>40.3</td>
<td>4.8</td>
<td>231</td>
<td>307</td>
<td>14.4</td>
</tr>
<tr>
<td>7</td>
<td>40.3</td>
<td>4.8</td>
<td>231</td>
<td>367</td>
<td>26.8</td>
</tr>
<tr>
<td>7</td>
<td>40.3</td>
<td>4.8</td>
<td>231</td>
<td>395</td>
<td>34.9</td>
</tr>
<tr>
<td>8</td>
<td>40.8</td>
<td>3.5</td>
<td>210</td>
<td>218</td>
<td>8.0</td>
</tr>
<tr>
<td>8</td>
<td>40.8</td>
<td>3.5</td>
<td>210</td>
<td>273</td>
<td>13.1</td>
</tr>
<tr>
<td>8</td>
<td>40.8</td>
<td>3.5</td>
<td>210</td>
<td>347</td>
<td>26.6</td>
</tr>
<tr>
<td>9</td>
<td>41.3</td>
<td>1.8</td>
<td>267</td>
<td>235</td>
<td>2.8</td>
</tr>
<tr>
<td>9</td>
<td>41.3</td>
<td>1.8</td>
<td>267</td>
<td>275</td>
<td>6.4</td>
</tr>
<tr>
<td>9</td>
<td>41.3</td>
<td>1.8</td>
<td>267</td>
<td>358</td>
<td>16.1</td>
</tr>
<tr>
<td>9</td>
<td>41.3</td>
<td>1.8</td>
<td>267</td>
<td>416</td>
<td>27.8</td>
</tr>
<tr>
<td>10</td>
<td>50.8</td>
<td>8.6</td>
<td>190</td>
<td>205</td>
<td>12.2</td>
</tr>
<tr>
<td>10</td>
<td>50.8</td>
<td>8.6</td>
<td>190</td>
<td>275</td>
<td>22.3</td>
</tr>
<tr>
<td>10</td>
<td>50.8</td>
<td>8.6</td>
<td>190</td>
<td>345</td>
<td>34.7</td>
</tr>
<tr>
<td>10</td>
<td>50.8</td>
<td>8.6</td>
<td>190</td>
<td>407</td>
<td>45.7</td>
</tr>
</tbody>
</table>

Table 8: The gasoline recovery data

**Suggested analysis**

Using EndPt as a covariate, check to see if differences between samples can be accounted for by regression models on the other predictors.

More advanced: Fit a two stratum ANOVA model using between and within samples as the two strata.

**C.7 The Michaelson and Morley speed of light data**

**Source:** Weekes: A Genstat Primer.

**Category:** Analysis of Variance.
C.8  The rat genotype data

Description

Data from a foster feeding experiment with rat mothers and litters of four different genotypes: A, F, I and J. The measurement is the litter weight gain after a trial feeding period.

Data

The data is given in Table 10 and may be read as a data frame from file genotype.data.

---

Table 9: The Michaelson and Morley speed of light data

<table>
<thead>
<tr>
<th>Runs 1-10</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Runs 11-20</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E₁</td>
<td>E₂</td>
<td>E₃</td>
<td>E₄</td>
<td>E₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>850</td>
<td>960</td>
<td>880</td>
<td>890</td>
<td>890</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>740</td>
<td>940</td>
<td>880</td>
<td>810</td>
<td>840</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>900</td>
<td>960</td>
<td>880</td>
<td>810</td>
<td>780</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1070</td>
<td>940</td>
<td>860</td>
<td>820</td>
<td>810</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>930</td>
<td>880</td>
<td>720</td>
<td>800</td>
<td>760</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>850</td>
<td>800</td>
<td>720</td>
<td>770</td>
<td>810</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>950</td>
<td>850</td>
<td>620</td>
<td>760</td>
<td>790</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>980</td>
<td>880</td>
<td>860</td>
<td>740</td>
<td>810</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>980</td>
<td>900</td>
<td>970</td>
<td>750</td>
<td>820</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>880</td>
<td>840</td>
<td>950</td>
<td>760</td>
<td>850</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>830</td>
<td>880</td>
<td>910</td>
<td>870</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>980</td>
<td>790</td>
<td>910</td>
<td>920</td>
<td>870</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>930</td>
<td>810</td>
<td>850</td>
<td>890</td>
<td>810</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>650</td>
<td>880</td>
<td>870</td>
<td>860</td>
<td>740</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>760</td>
<td>880</td>
<td>840</td>
<td>880</td>
<td>810</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>810</td>
<td>830</td>
<td>840</td>
<td>720</td>
<td>940</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>800</td>
<td>850</td>
<td>840</td>
<td>950</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>790</td>
<td>840</td>
<td>850</td>
<td>800</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>960</td>
<td>760</td>
<td>840</td>
<td>850</td>
<td>810</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>960</td>
<td>800</td>
<td>840</td>
<td>780</td>
<td>870</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Suggested analysis

Using an single classification ANOVA model check for differences between experiments and summarise your conclusions.
<table>
<thead>
<tr>
<th>Litter’s Genotype</th>
<th>Mother’s Genotype</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>F</td>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>A</td>
<td>61.5</td>
<td>55.0</td>
<td>52.5</td>
<td>42.0</td>
</tr>
<tr>
<td></td>
<td>68.2</td>
<td>42.0</td>
<td>61.8</td>
<td>54.0</td>
</tr>
<tr>
<td></td>
<td>64.0</td>
<td>60.2</td>
<td>49.5</td>
<td>61.0</td>
</tr>
<tr>
<td></td>
<td>65.0</td>
<td>52.7</td>
<td>48.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>59.7</td>
<td></td>
<td></td>
<td>39.6</td>
</tr>
<tr>
<td>F</td>
<td>60.3</td>
<td>50.8</td>
<td>56.5</td>
<td>51.3</td>
</tr>
<tr>
<td></td>
<td>51.7</td>
<td>64.7</td>
<td>59.0</td>
<td>40.5</td>
</tr>
<tr>
<td></td>
<td>49.3</td>
<td>61.7</td>
<td>47.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>48.0</td>
<td>64.0</td>
<td>53.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>62.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>37.0</td>
<td>56.3</td>
<td>39.7</td>
<td>50.0</td>
</tr>
<tr>
<td></td>
<td>36.3</td>
<td>69.8</td>
<td>46.0</td>
<td>43.8</td>
</tr>
<tr>
<td></td>
<td>68.0</td>
<td>67.0</td>
<td>61.3</td>
<td>54.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>55.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>55.7</td>
</tr>
<tr>
<td>J</td>
<td>59.0</td>
<td>59.5</td>
<td>45.2</td>
<td>44.8</td>
</tr>
<tr>
<td></td>
<td>57.4</td>
<td>52.8</td>
<td>57.0</td>
<td>51.5</td>
</tr>
<tr>
<td></td>
<td>54.0</td>
<td>56.0</td>
<td>61.4</td>
<td>53.0</td>
</tr>
<tr>
<td></td>
<td>47.0</td>
<td></td>
<td>42.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54.0</td>
</tr>
</tbody>
</table>

Table 10: The rat genotype data

Suggested analysis

Fit a double classification model. Check for interaction using both a formal analysis and graphically using `interaction.plot()`. Test the main effects and summarise.

C.9  Fisher’s sugar beet data


Category: Analysis of variance and covariance.

Description

A classical $3 \times 2^3$ randomized block experiment in four blocks of size 24. The response is the total weight of sugarbeet roots off the plot, but this is accompanied by the number of roots measured. The suggestion is that number of roots should be a covariate to allow for varying plot size.

The factors are Variety. (3 levels, a, b and c), and N, P and K each at 2 levels, present or absent.
Data

The data is given in Table 11 and may be read from the file sugar.data in a form suitable to construct a data frame.

<table>
<thead>
<tr>
<th>V</th>
<th>N</th>
<th>P</th>
<th>K</th>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
<th>Block 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>No Wt</td>
<td>No Wt</td>
<td>No Wt</td>
<td>No Wt</td>
</tr>
<tr>
<td>a</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>124 162</td>
<td>133 162</td>
<td>114 127</td>
<td>127 158</td>
</tr>
<tr>
<td>a</td>
<td>-</td>
<td>-</td>
<td>k</td>
<td>131 152</td>
<td>161 164</td>
<td>130 141</td>
<td>145 188</td>
</tr>
<tr>
<td>a</td>
<td>-</td>
<td>p</td>
<td>-</td>
<td>115 173</td>
<td>134 175</td>
<td>134 142</td>
<td>109 162</td>
</tr>
<tr>
<td>a</td>
<td>-</td>
<td>p</td>
<td>k</td>
<td>126 140</td>
<td>133 158</td>
<td>106 148</td>
<td>132 160</td>
</tr>
<tr>
<td>a</td>
<td>n</td>
<td>-</td>
<td>-</td>
<td>136 184</td>
<td>134 178</td>
<td>127 168</td>
<td>139 199</td>
</tr>
<tr>
<td>a</td>
<td>n</td>
<td>-</td>
<td>k</td>
<td>134 112</td>
<td>156 193</td>
<td>101 171</td>
<td>138 191</td>
</tr>
<tr>
<td>a</td>
<td>n</td>
<td>p</td>
<td>-</td>
<td>132 190</td>
<td>104 166</td>
<td>119 157</td>
<td>132 193</td>
</tr>
<tr>
<td>a</td>
<td>n</td>
<td>p</td>
<td>k</td>
<td>120 175</td>
<td>147 155</td>
<td>107 139</td>
<td>148 192</td>
</tr>
<tr>
<td>b</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>145 133</td>
<td>147 130</td>
<td>139 138</td>
<td>127 128</td>
</tr>
<tr>
<td>b</td>
<td>-</td>
<td>-</td>
<td>k</td>
<td>156 117</td>
<td>152 137</td>
<td>107 121</td>
<td>147 147</td>
</tr>
<tr>
<td>b</td>
<td>-</td>
<td>p</td>
<td>-</td>
<td>152 140</td>
<td>138 101</td>
<td>125 124</td>
<td>120 143</td>
</tr>
<tr>
<td>b</td>
<td>-</td>
<td>p</td>
<td>k</td>
<td>137 127</td>
<td>145 132</td>
<td>125 132</td>
<td>143 139</td>
</tr>
<tr>
<td>b</td>
<td>n</td>
<td>-</td>
<td>-</td>
<td>124 163</td>
<td>138 159</td>
<td>140 166</td>
<td>159 174</td>
</tr>
<tr>
<td>b</td>
<td>n</td>
<td>-</td>
<td>k</td>
<td>136 143</td>
<td>142 144</td>
<td>133 142</td>
<td>148 159</td>
</tr>
<tr>
<td>b</td>
<td>n</td>
<td>p</td>
<td>-</td>
<td>140 168</td>
<td>142 150</td>
<td>133 118</td>
<td>138 157</td>
</tr>
<tr>
<td>b</td>
<td>n</td>
<td>p</td>
<td>k</td>
<td>146 144</td>
<td>135 160</td>
<td>138 155</td>
<td>140 153</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>113 122</td>
<td>138 132</td>
<td>119 123</td>
<td>127 146</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
<td>-</td>
<td>k</td>
<td>91 107</td>
<td>149 171</td>
<td>118 142</td>
<td>129 151</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
<td>p</td>
<td>-</td>
<td>123 118</td>
<td>139 142</td>
<td>127 120</td>
<td>124 138</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
<td>p</td>
<td>k</td>
<td>129 140</td>
<td>126 115</td>
<td>129 130</td>
<td>142 152</td>
</tr>
<tr>
<td>c</td>
<td>n</td>
<td>-</td>
<td>-</td>
<td>121 118</td>
<td>141 152</td>
<td>127 149</td>
<td>127 165</td>
</tr>
<tr>
<td>c</td>
<td>n</td>
<td>-</td>
<td>k</td>
<td>126 148</td>
<td>128 152</td>
<td>107 147</td>
<td>110 136</td>
</tr>
<tr>
<td>c</td>
<td>n</td>
<td>p</td>
<td>-</td>
<td>103 112</td>
<td>144 175</td>
<td>102 152</td>
<td>143 173</td>
</tr>
<tr>
<td>c</td>
<td>n</td>
<td>p</td>
<td>k</td>
<td>120 162</td>
<td>125 160</td>
<td>129 173</td>
<td>137 185</td>
</tr>
</tbody>
</table>

Table 11: Fisher’s sugar beet data

Suggested analysis

Analyse the data as a randomised block experiment with Wt as the response and No as a covariate. Prune the model of all unnecessary interaction terms and summarise.

C.10 A barley split plot field trial

Source: Unknown. Traditional data.
Category: Multistratum analysis of variance.

Description

An experiment involving barley varieties and manure (nitrogen) was conducted in 6 blocks of 3 whole plots.

Each whole plot was divided into 4 subplots. Three varieties of barley were used in the experiment with one variety being sown in each whole plot, while the four levels of manure (0, 0.01, 0.02, and 0.04 tons per acre) were used, one level in each of the four subplots of each whole plot. In the above table $V_i$ denotes the $i$th variety and $N_j$ denotes the $j$th level of nitrogen.

Data

<table>
<thead>
<tr>
<th>Block</th>
<th>Variety</th>
<th>$N_1$</th>
<th>$N_2$</th>
<th>$N_3$</th>
<th>$N_4$</th>
<th>Block</th>
<th>Variety</th>
<th>$N_1$</th>
<th>$N_2$</th>
<th>$N_3$</th>
<th>$N_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$V_1$</td>
<td>111</td>
<td>130</td>
<td>157</td>
<td>174</td>
<td>IV</td>
<td>$V_1$</td>
<td>74</td>
<td>89</td>
<td>81</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>$V_2$</td>
<td>117</td>
<td>114</td>
<td>161</td>
<td>141</td>
<td></td>
<td>$V_2$</td>
<td>64</td>
<td>103</td>
<td>132</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>$V_3$</td>
<td>105</td>
<td>140</td>
<td>118</td>
<td>156</td>
<td></td>
<td>$V_3$</td>
<td>70</td>
<td>89</td>
<td>104</td>
<td>117</td>
</tr>
<tr>
<td>II</td>
<td>$V_1$</td>
<td>61</td>
<td>91</td>
<td>97</td>
<td>100</td>
<td>V</td>
<td>$V_1$</td>
<td>62</td>
<td>90</td>
<td>100</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>$V_2$</td>
<td>70</td>
<td>108</td>
<td>126</td>
<td>149</td>
<td></td>
<td>$V_2$</td>
<td>80</td>
<td>82</td>
<td>94</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td>$V_3$</td>
<td>96</td>
<td>124</td>
<td>121</td>
<td>144</td>
<td></td>
<td>$V_3$</td>
<td>63</td>
<td>70</td>
<td>109</td>
<td>99</td>
</tr>
<tr>
<td>III</td>
<td>$V_1$</td>
<td>68</td>
<td>64</td>
<td>112</td>
<td>86</td>
<td>VI</td>
<td>$V_1$</td>
<td>53</td>
<td>74</td>
<td>118</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>$V_2$</td>
<td>60</td>
<td>102</td>
<td>89</td>
<td>96</td>
<td></td>
<td>$V_2$</td>
<td>89</td>
<td>82</td>
<td>86</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>$V_3$</td>
<td>89</td>
<td>129</td>
<td>132</td>
<td>124</td>
<td></td>
<td>$V_3$</td>
<td>97</td>
<td>99</td>
<td>119</td>
<td>121</td>
</tr>
</tbody>
</table>

Table 12: A split plot barley field trial

The data is given in Table 12 and may be read as a data frame from file barley.data.

Suggested analysis

Analyse as a split plot field experiment and summarise.

C.11 The snail mortality data

Source: Zoology Department, The University of Adelaide.
Category: Generalized Linear Modelling.

Description

Groups of 20 snails were held for periods of 1, 2, 3 or 4 weeks (exposure) in carefully controlled conditions of temperature (3 levels) and relative humidity (4 levels). There were two species of snail, A and B, and the experiment was designed as a $4 \times 3 \times 4 \times 2$ completely randomized design. At the end of the exposure time the snails were tested to see if they had survived; this process itself is fatal for the animals. The object of the exercise was to model the probability of survival in terms of the stimulus variables, and in particular to test for differences between species.

The data is unusual in that in most cases fatalities during the experiment were fairly small.
C.12 The Kalythos blindness data

Data

<table>
<thead>
<tr>
<th>Species</th>
<th>Exposure</th>
<th>Relative Humidity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>60.0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Temperature</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0 1 1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1 4 5</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7 7 7</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0 3 2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7 11 11</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>12 14 16</td>
</tr>
</tbody>
</table>

Table 13: The snail mortality data

The data is given in Table 13 and may be read as a data frame from file snails.data.

Suggested analysis

The data is interesting in that although it has many extremely small cell counts there is every indication that some of the likelihood ratio large sample theory is quite safe.

Fit Binomial models to the data with either a logit or a probit link. Show that a model with parallel regressions on Temp, Humid, Exposure and Exposure^2 for each species (in the logit/probit scale) is reasonable, and summarise.

C.12 The Kalythos blindness data

Source: S. D. Silvey: Statistical Inference. (Fictitious?)
Category: Generalized linear modelling

Description

On the Greek island of Kalythos the male inhabitants suffer from a congenital eye disease, the effects of which become more marked with increasing age. Samples of islander males of various ages were tested for blindness and the results recorded.

Data

<table>
<thead>
<tr>
<th>Age:</th>
<th>20</th>
<th>35</th>
<th>45</th>
<th>55</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. tested:</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>No. blind:</td>
<td>6</td>
<td>17</td>
<td>26</td>
<td>37</td>
<td>44</td>
</tr>
</tbody>
</table>

Table 14: The Kalythos blindness data

The data is given in Table 14 and may be read as a data frame from file kalythos.data.
Suggested analysis

Using an logit or probit model estimate the LD50, that is, the age at which the probability of blindness is \( p = \frac{1}{2} \), together the standard error. Check how different the logit and probit models are in this respect.

C.13 The Stormer viscometer calibration data

Source: E. J. Williams: Regression Analysis, Wiley, 1959

Category: Nonlinear regression, special regression.

Description

The stormer viscometer measures the viscosity of a fluid by measuring the time taken for an inner cylinder in the mechanism to perform a fixed number of revolutions in response to an actuating weight. The viscometer is calibrated by measuring the time taken with varying weights while the mechanism is suspended in fluids of accurately known viscosity. The data comes from such a calibration, and theoretical considerations suggest a nonlinear relationship between time, weight and viscosity of the form

\[
T_i = \frac{\beta v_i}{w_i - \theta} + E_i
\]

where \( \beta \) and \( \theta \) are unknown parameters to be estimated.

Data

<table>
<thead>
<tr>
<th>Viscosity</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
</tr>
<tr>
<td>14.7</td>
<td>35.6</td>
</tr>
<tr>
<td>27.5</td>
<td>54.3</td>
</tr>
<tr>
<td>42.0</td>
<td>75.6</td>
</tr>
<tr>
<td>75.7</td>
<td>121.2</td>
</tr>
<tr>
<td>89.7</td>
<td>150.8</td>
</tr>
<tr>
<td>146.6</td>
<td>229.0</td>
</tr>
<tr>
<td>158.3</td>
<td>270.0</td>
</tr>
<tr>
<td>161.1</td>
<td>92.2</td>
</tr>
<tr>
<td>298.3</td>
<td>187.2</td>
</tr>
</tbody>
</table>

Table 15: The Stormer viscometer calibration data

The data is given in Table 15 and may be read as a data frame from file stormer.data.
Suggested analysis

Estimate the nonlinear regression model using the \texttt{nlr()} software. A suitable initial value may be obtained by writing the regression model in the form
\[ w_i T_i = \beta v_i + \theta T_i + (w_i - \theta) E_i \]
and regressing \( w_i T_i \) on \( v_i \) and \( T_i \) using ordinary linear regression.

C.14 The chlorine availability data

Category: Nonlinear regression

Description

The following set of industrial chemical data shows the amount of chlorine available in a certain product at various times of testing after manufacture. A nonlinear regression model for the chlorine decay of the form
\[ Y = \beta_0 + \beta_1 \exp(-\theta x) \]
has been suggested on theoretical grounds, with \( Y \) the amount remaining at time \( x \).

Data

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Percent available</th>
<th>Weeks</th>
<th>Percent available</th>
<th>Weeks</th>
<th>Percent available</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>49, 49</td>
<td>20</td>
<td>42, 43, 42</td>
<td>32</td>
<td>40, 41</td>
</tr>
<tr>
<td>10</td>
<td>47, 47, 48, 48</td>
<td>22</td>
<td>40, 41, 41</td>
<td>34</td>
<td>40</td>
</tr>
<tr>
<td>12</td>
<td>43, 45, 46, 46</td>
<td>24</td>
<td>40, 40, 42</td>
<td>36</td>
<td>38, 41</td>
</tr>
<tr>
<td>14</td>
<td>43, 43, 45</td>
<td>26</td>
<td>40, 41, 41</td>
<td>38</td>
<td>40, 40</td>
</tr>
<tr>
<td>16</td>
<td>43, 43, 44</td>
<td>28</td>
<td>40, 41</td>
<td>40</td>
<td>39</td>
</tr>
<tr>
<td>18</td>
<td>45, 46</td>
<td>30</td>
<td>38, 40, 40</td>
<td>42</td>
<td>39</td>
</tr>
</tbody>
</table>

Table 16: The chlorine availability data.

The data is given in Table 16 and may be read as a data frame from file \texttt{chlorine.data}.

Suggested analysis

Fit the nonlinear regression model using the \texttt{nlr()} function. A simple way to find an initial value is to guess a value for \( \theta \), say \( \theta = 1 \) and plot \( Y \) against \( \exp(-\theta x) \). Now repeatedly either double or halve \( \theta \) until the plot is near to linear. (This can be done very simply with the inbuilt line editor.) Once an initial value for \( \theta \) is available, initial values for the others can be got by linear regression.

Check the fitted model for suspicious data points.

C.15 The saturated steam pressure data

Source: Quoted in Draper & Smith: *Applied Regression Analysis*..
Category: Nonlinear regression.
Description

The data gives the temperature (°C) and pressure (Pascals) in a saturated steam driven experimental device. The relationship between pressure, \( Y \), and temperature, \( x \), in saturated steam can be written as

\[
Y = \alpha \exp \left( \frac{\beta x}{\gamma + x} \right) + E
\]

However a more realistic model may have the experimental errors multiplicative rather than additive, in which case an analysis in the log scale using the model

\[
\log Y = \log \alpha + \left( \frac{\beta x}{\gamma + x} \right) + E
\]

may be more appropriate.

Data

<table>
<thead>
<tr>
<th>Temp</th>
<th>Press</th>
<th>Temp</th>
<th>Press</th>
<th>Temp</th>
<th>Press</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.14</td>
<td>50</td>
<td>98.76</td>
<td>90</td>
<td>522.78</td>
</tr>
<tr>
<td>10</td>
<td>8.52</td>
<td>60</td>
<td>151.13</td>
<td>95</td>
<td>674.32</td>
</tr>
<tr>
<td>20</td>
<td>16.31</td>
<td>70</td>
<td>224.74</td>
<td>100</td>
<td>782.04</td>
</tr>
<tr>
<td>30</td>
<td>32.18</td>
<td>80</td>
<td>341.35</td>
<td>105</td>
<td>920.01</td>
</tr>
<tr>
<td>40</td>
<td>64.62</td>
<td>85</td>
<td>423.36</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 17: Temperature and pressure in saturated steam

The data is given in Table 17 and may be read as a data frame from file `steam.data`.

Suggested analysis

Fit both models and compare. Initial values may be got by a similar method to that employed for the Chlorine data, since again there is only one nonlinear parameter.

C.16 Count Rumford’s friction data

Source: Bates & Watts: *Nonlinear Regression Analysis*...

Category: Nonlinear regression

Description

Data on the amount of heat generated by friction was obtained by Lord Rumford in 1798. A bore was fitted into a stationary cylinder and pressed against the bottom by a screw. The bore was turned by a team of horses for 30 minutes, after which Lord Rumford “suffered the thermometer to remain in its place nearly three quarters of an hour, observing and noting down, at small intervals of time, the temperature indicated by it”.

Newton’s law of cooling suggests a nonlinear regression model of the form

\[
Y = \beta_0 + \beta_1 \exp(-\theta x)
\]

where \( Y \) is the temperature and \( x \) is the time in minutes.
Data

<table>
<thead>
<tr>
<th>Time (min.)</th>
<th>Temp (°F)</th>
<th>Time (min.)</th>
<th>Temp (°F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>126</td>
<td>24.0</td>
<td>115</td>
</tr>
<tr>
<td>5.0</td>
<td>125</td>
<td>28.0</td>
<td>114</td>
</tr>
<tr>
<td>7.0</td>
<td>123</td>
<td>31.0</td>
<td>113</td>
</tr>
<tr>
<td>12.0</td>
<td>120</td>
<td>34.0</td>
<td>112</td>
</tr>
<tr>
<td>14.0</td>
<td>119</td>
<td>37.5</td>
<td>111</td>
</tr>
<tr>
<td>16.0</td>
<td>118</td>
<td>41.0</td>
<td>110</td>
</tr>
<tr>
<td>20.0</td>
<td>116</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 18: The Rumford friction cooling data

The data is given in Table 18 and may be read as a data frame from file rumford.data.

Suggested analysis

This data is mainly of historical interest. Handle similarly to the Chlorine data above.

C.17  The jellyfish data

Category: Bivariate, two sample data.

Description

Two samples of jellyfish, from Danger Island and Salamander Bay respectively, were measured for length and width.

Data

The data is given in Table 19 and may be read as a data frame from file jellyfish.data.

Suggested analysis

Plot the two samples and mark in their convex hulls. Test for differences using Hotelling's $T^2$. (A simple way of conducting the analysis is to regress a dummy variable for Location on Length and width and to test the significance of both regression coefficients simultaneously.

C.18  The Archaeological pottery data

Category: Multivariate analysis
<table>
<thead>
<tr>
<th>Width</th>
<th>Length</th>
<th>Width</th>
<th>Length</th>
<th>Width</th>
<th>Length</th>
<th>Width</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0</td>
<td>9.0</td>
<td>11.0</td>
<td>13.0</td>
<td>12.0</td>
<td>14.0</td>
<td>16.0</td>
<td>20.0</td>
</tr>
<tr>
<td>6.5</td>
<td>8.0</td>
<td>11.0</td>
<td>14.0</td>
<td>13.0</td>
<td>17.0</td>
<td>16.0</td>
<td>20.0</td>
</tr>
<tr>
<td>6.5</td>
<td>9.0</td>
<td>11.0</td>
<td>14.0</td>
<td>14.0</td>
<td>16.5</td>
<td>16.0</td>
<td>21.0</td>
</tr>
<tr>
<td>7.0</td>
<td>9.0</td>
<td>12.0</td>
<td>13.0</td>
<td>14.0</td>
<td>19.0</td>
<td>16.5</td>
<td>19.0</td>
</tr>
<tr>
<td>7.0</td>
<td>10.0</td>
<td>13.0</td>
<td>14.0</td>
<td>15.0</td>
<td>16.0</td>
<td>17.0</td>
<td>20.0</td>
</tr>
<tr>
<td>7.0</td>
<td>11.0</td>
<td>14.0</td>
<td>16.0</td>
<td>15.0</td>
<td>17.0</td>
<td>18.0</td>
<td>19.0</td>
</tr>
<tr>
<td>8.0</td>
<td>9.5</td>
<td>15.0</td>
<td>16.0</td>
<td>15.0</td>
<td>18.0</td>
<td>18.0</td>
<td>19.0</td>
</tr>
<tr>
<td>8.0</td>
<td>10.0</td>
<td>15.0</td>
<td>16.0</td>
<td>15.0</td>
<td>18.0</td>
<td>18.0</td>
<td>20.0</td>
</tr>
<tr>
<td>8.0</td>
<td>10.0</td>
<td>15.0</td>
<td>19.0</td>
<td>15.0</td>
<td>19.0</td>
<td>19.0</td>
<td>20.0</td>
</tr>
<tr>
<td>8.0</td>
<td>11.0</td>
<td>16.0</td>
<td>16.0</td>
<td>15.0</td>
<td>21.0</td>
<td>19.0</td>
<td>22.0</td>
</tr>
<tr>
<td>9.0</td>
<td>11.0</td>
<td></td>
<td></td>
<td>16.0</td>
<td>18.0</td>
<td>20.0</td>
<td>22.0</td>
</tr>
<tr>
<td>10.0</td>
<td>13.0</td>
<td></td>
<td></td>
<td>16.0</td>
<td>19.0</td>
<td>21.0</td>
<td>21.0</td>
</tr>
</tbody>
</table>

Table 19: The jellyfish data – Danger Island and Salamander Bay

**Description**

The data arises from a chemical analysis of 26 samples of pottery found at Romano-British kiln sites in Wales, Gwent and the New Forest. The variables describe the composition, in terms of various metals, and are expressed as percentages of the oxides of the metals.

The metals are aluminium, iron, magnesium, calcium and sodium and the sites are

- L: Llanederyn
- C: Caldicot
- I: Island Thorns
- A: Ashley Rails

**Data**

The data is given in Table 20 and may be read as a data frame from file pottery.data.

**Suggested analysis**

Investigate both numerically and graphically using a simple discriminant analysis. Exhibit the four samples using the first two discriminant functions as coordinate axes. Summarise.

**C.19 The Beaujolais quality data**

**Source:** Quoted in Weekes: *A Genstat Primer*

**Category:** Multivariate analysis

**Description**

Table 20: The pottery composition data

<table>
<thead>
<tr>
<th>Site</th>
<th>Al</th>
<th>Fe</th>
<th>Mg</th>
<th>Ca</th>
<th>Na</th>
<th>Site</th>
<th>Al</th>
<th>Fe</th>
<th>Mg</th>
<th>Ca</th>
<th>Na</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>14.4</td>
<td>7.00</td>
<td>4.30</td>
<td>0.15</td>
<td>0.51</td>
<td>C</td>
<td>11.8</td>
<td>5.44</td>
<td>3.94</td>
<td>0.30</td>
<td>0.04</td>
</tr>
<tr>
<td>L</td>
<td>13.8</td>
<td>7.08</td>
<td>3.43</td>
<td>0.12</td>
<td>0.17</td>
<td>C</td>
<td>11.6</td>
<td>5.39</td>
<td>3.77</td>
<td>0.29</td>
<td>0.06</td>
</tr>
<tr>
<td>L</td>
<td>14.6</td>
<td>7.09</td>
<td>3.88</td>
<td>0.13</td>
<td>0.20</td>
<td>I</td>
<td>18.3</td>
<td>1.28</td>
<td>0.67</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>L</td>
<td>11.5</td>
<td>6.37</td>
<td>5.64</td>
<td>0.16</td>
<td>0.14</td>
<td>I</td>
<td>15.8</td>
<td>2.39</td>
<td>0.63</td>
<td>0.01</td>
<td>0.04</td>
</tr>
<tr>
<td>L</td>
<td>13.8</td>
<td>7.06</td>
<td>5.34</td>
<td>0.20</td>
<td>0.20</td>
<td>I</td>
<td>18.0</td>
<td>1.50</td>
<td>0.67</td>
<td>0.01</td>
<td>0.06</td>
</tr>
<tr>
<td>L</td>
<td>10.9</td>
<td>6.26</td>
<td>3.47</td>
<td>0.17</td>
<td>0.22</td>
<td>I</td>
<td>18.0</td>
<td>1.88</td>
<td>0.68</td>
<td>0.01</td>
<td>0.04</td>
</tr>
<tr>
<td>L</td>
<td>10.1</td>
<td>4.26</td>
<td>4.26</td>
<td>0.20</td>
<td>0.18</td>
<td>A</td>
<td>17.7</td>
<td>1.12</td>
<td>0.56</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>L</td>
<td>11.6</td>
<td>5.78</td>
<td>5.91</td>
<td>0.18</td>
<td>0.16</td>
<td>A</td>
<td>18.3</td>
<td>1.14</td>
<td>0.67</td>
<td>0.06</td>
<td>0.05</td>
</tr>
<tr>
<td>L</td>
<td>13.4</td>
<td>6.92</td>
<td>7.23</td>
<td>0.28</td>
<td>0.20</td>
<td>A</td>
<td>16.7</td>
<td>0.92</td>
<td>0.53</td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>L</td>
<td>12.4</td>
<td>6.13</td>
<td>5.69</td>
<td>0.22</td>
<td>0.54</td>
<td>A</td>
<td>14.8</td>
<td>2.74</td>
<td>0.67</td>
<td>0.03</td>
<td>0.05</td>
</tr>
<tr>
<td>L</td>
<td>13.1</td>
<td>6.64</td>
<td>5.51</td>
<td>0.31</td>
<td>0.24</td>
<td>A</td>
<td>19.1</td>
<td>1.64</td>
<td>0.60</td>
<td>0.10</td>
<td>0.03</td>
</tr>
<tr>
<td>L</td>
<td>12.7</td>
<td>6.69</td>
<td>4.45</td>
<td>0.20</td>
<td>0.22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>12.5</td>
<td>6.44</td>
<td>3.94</td>
<td>0.22</td>
<td>0.23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 21: Quality measurements on young Beaujolais wine samples

The data is given in Table 21 and may be read as a data frame from file beaujolais.data.

Suggested analysis

Look at ways of exhibiting the data graphically. Consider a principal component analysis using the correlation matrix and look for any wild outliers.

C.20 The painters data of de Piles

Category: Multivariate Analysis: Discriminant Analysis.

Description

The data shows the subjective assessment, on a 0-20 integer scale, of 54 classical painters. The painters were assessed on four characteristics: composition, drawing, colour and expression. The data is due to the Eighteenth century art critic, de Piles.

The School to which a painter belongs is indicated by a letter code as follows:

<table>
<thead>
<tr>
<th>A</th>
<th>Renaissance</th>
<th>E</th>
<th>Lombard</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Mannerist</td>
<td>F</td>
<td>Sixteenth Century</td>
</tr>
<tr>
<td>C</td>
<td>Seicento</td>
<td>G</td>
<td>Seventeenth Century</td>
</tr>
<tr>
<td>D</td>
<td>Venetian</td>
<td>H</td>
<td>French</td>
</tr>
</tbody>
</table>

Data

The data is given in Table 22 and may be read as a data frame from file painters.data.

Suggested analysis

Using a multivariate analysis of variance check for differences between schools. Use the likelihood ratio test. Also find the canonical $F$-statistics and discriminant functions.

Plot the painters on the first two discriminant function axes and use the school symbol as a plotting character. Mark in the convex hulls of the schools. Using identify() find interactively some of the painters that appear to lie towards the extremes of the plot, or who deviate considerably from their school mean.
<table>
<thead>
<tr>
<th>Artist</th>
<th>Composition</th>
<th>Drawing</th>
<th>Colour</th>
<th>Expression</th>
<th>School</th>
</tr>
</thead>
<tbody>
<tr>
<td>Da Udine</td>
<td>10</td>
<td>8</td>
<td>16</td>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td>Da Vinci</td>
<td>15</td>
<td>16</td>
<td>4</td>
<td>14</td>
<td>A</td>
</tr>
<tr>
<td>Del Piombo</td>
<td>8</td>
<td>13</td>
<td>16</td>
<td>7</td>
<td>A</td>
</tr>
<tr>
<td>Del Sarto</td>
<td>12</td>
<td>16</td>
<td>9</td>
<td>8</td>
<td>A</td>
</tr>
<tr>
<td>Fr. Penni</td>
<td>0</td>
<td>15</td>
<td>8</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>Guilio Romano</td>
<td>15</td>
<td>16</td>
<td>4</td>
<td>14</td>
<td>A</td>
</tr>
<tr>
<td>Michelangelo</td>
<td>8</td>
<td>17</td>
<td>4</td>
<td>8</td>
<td>A</td>
</tr>
<tr>
<td>Perino del Vaga</td>
<td>15</td>
<td>16</td>
<td>7</td>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>Perugino</td>
<td>4</td>
<td>12</td>
<td>10</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>Raphael</td>
<td>17</td>
<td>18</td>
<td>12</td>
<td>18</td>
<td>A</td>
</tr>
<tr>
<td>F. Zucarro</td>
<td>10</td>
<td>13</td>
<td>8</td>
<td>8</td>
<td>B</td>
</tr>
<tr>
<td>Fr. Salviata</td>
<td>13</td>
<td>15</td>
<td>8</td>
<td>8</td>
<td>B</td>
</tr>
<tr>
<td>Parmigiano</td>
<td>10</td>
<td>15</td>
<td>6</td>
<td>6</td>
<td>B</td>
</tr>
<tr>
<td>Primaticcio</td>
<td>15</td>
<td>14</td>
<td>7</td>
<td>10</td>
<td>B</td>
</tr>
<tr>
<td>T. Zucarro</td>
<td>13</td>
<td>14</td>
<td>10</td>
<td>9</td>
<td>B</td>
</tr>
<tr>
<td>Volterra</td>
<td>12</td>
<td>15</td>
<td>5</td>
<td>8</td>
<td>B</td>
</tr>
<tr>
<td>Barocci</td>
<td>14</td>
<td>15</td>
<td>6</td>
<td>10</td>
<td>C</td>
</tr>
<tr>
<td>Cortona</td>
<td>16</td>
<td>14</td>
<td>12</td>
<td>6</td>
<td>C</td>
</tr>
<tr>
<td>Josepin</td>
<td>10</td>
<td>10</td>
<td>6</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>L. Jordans</td>
<td>13</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>C</td>
</tr>
<tr>
<td>Testa</td>
<td>11</td>
<td>15</td>
<td>0</td>
<td>6</td>
<td>C</td>
</tr>
<tr>
<td>Vanini</td>
<td>15</td>
<td>15</td>
<td>12</td>
<td>13</td>
<td>C</td>
</tr>
<tr>
<td>Bassano</td>
<td>6</td>
<td>8</td>
<td>17</td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>Bellini</td>
<td>4</td>
<td>6</td>
<td>14</td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>Giorgione</td>
<td>8</td>
<td>9</td>
<td>18</td>
<td>4</td>
<td>D</td>
</tr>
<tr>
<td>Murillo</td>
<td>6</td>
<td>8</td>
<td>15</td>
<td>4</td>
<td>D</td>
</tr>
<tr>
<td>Palma Giovane</td>
<td>12</td>
<td>9</td>
<td>14</td>
<td>6</td>
<td>D</td>
</tr>
<tr>
<td>Palma Vecchio</td>
<td>5</td>
<td>6</td>
<td>16</td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>Pordenone</td>
<td>8</td>
<td>14</td>
<td>17</td>
<td>5</td>
<td>D</td>
</tr>
<tr>
<td>Tintoretto</td>
<td>15</td>
<td>14</td>
<td>16</td>
<td>4</td>
<td>D</td>
</tr>
<tr>
<td>Titian</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>6</td>
<td>D</td>
</tr>
<tr>
<td>Veronese</td>
<td>15</td>
<td>10</td>
<td>16</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>Albani</td>
<td>14</td>
<td>14</td>
<td>10</td>
<td>6</td>
<td>E</td>
</tr>
<tr>
<td>Caravaggio</td>
<td>6</td>
<td>6</td>
<td>16</td>
<td>0</td>
<td>E</td>
</tr>
<tr>
<td>Corregio</td>
<td>13</td>
<td>13</td>
<td>15</td>
<td>12</td>
<td>E</td>
</tr>
<tr>
<td>Domenichino</td>
<td>15</td>
<td>17</td>
<td>9</td>
<td>17</td>
<td>E</td>
</tr>
<tr>
<td>Guercino</td>
<td>18</td>
<td>10</td>
<td>10</td>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>Lanfranco</td>
<td>14</td>
<td>13</td>
<td>10</td>
<td>5</td>
<td>E</td>
</tr>
<tr>
<td>The Carracci</td>
<td>15</td>
<td>17</td>
<td>13</td>
<td>13</td>
<td>E</td>
</tr>
<tr>
<td>Durer</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>F</td>
</tr>
<tr>
<td>Holbein</td>
<td>9</td>
<td>10</td>
<td>16</td>
<td>13</td>
<td>F</td>
</tr>
<tr>
<td>Poussin</td>
<td>4</td>
<td>15</td>
<td>6</td>
<td>6</td>
<td>F</td>
</tr>
<tr>
<td>Van Leyden</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>F</td>
</tr>
<tr>
<td>Diepenbeck</td>
<td>11</td>
<td>10</td>
<td>14</td>
<td>6</td>
<td>G</td>
</tr>
<tr>
<td>J. Jordans</td>
<td>10</td>
<td>8</td>
<td>16</td>
<td>6</td>
<td>G</td>
</tr>
<tr>
<td>Otho Venius</td>
<td>13</td>
<td>14</td>
<td>10</td>
<td>10</td>
<td>G</td>
</tr>
<tr>
<td>Rembrandt</td>
<td>15</td>
<td>6</td>
<td>17</td>
<td>12</td>
<td>G</td>
</tr>
<tr>
<td>Rubens</td>
<td>18</td>
<td>13</td>
<td>17</td>
<td>17</td>
<td>G</td>
</tr>
<tr>
<td>Teniers</td>
<td>15</td>
<td>12</td>
<td>13</td>
<td>6</td>
<td>G</td>
</tr>
<tr>
<td>Van Dyck</td>
<td>15</td>
<td>10</td>
<td>17</td>
<td>13</td>
<td>G</td>
</tr>
<tr>
<td>Bourdon</td>
<td>10</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>H</td>
</tr>
<tr>
<td>Le Brun</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>16</td>
<td>H</td>
</tr>
<tr>
<td>Le Sueur</td>
<td>15</td>
<td>15</td>
<td>4</td>
<td>15</td>
<td>H</td>
</tr>
<tr>
<td>Poussin</td>
<td>15</td>
<td>17</td>
<td>6</td>
<td>15</td>
<td>H</td>
</tr>
</tbody>
</table>

Table 22: The subjective assessment data of de Piles