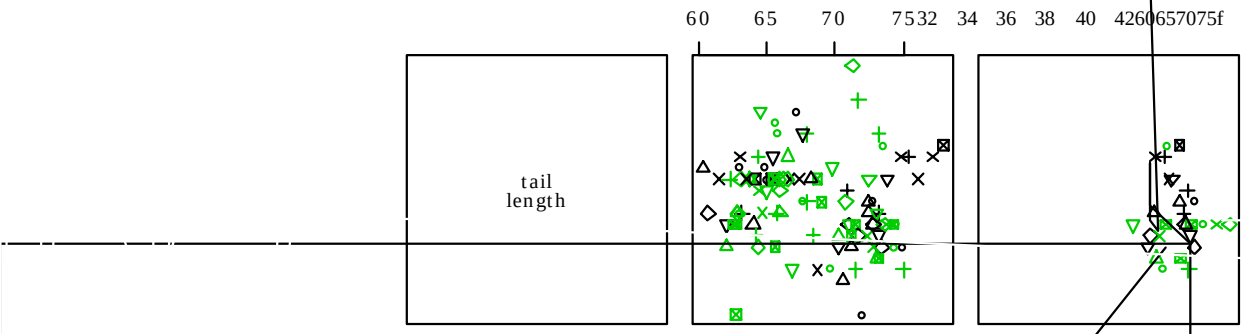
Using R for Data Analysis and Graphics
An Introduction

J H Maindonald

Statistical Consulting Unit of the Graduate School,
Australian National University.

Languages shape the way we think, and determine what we can think about (Benjamin Whorf.).

25 June 2001


60
65
70
75
32
34
36
38
40
42
60
65
70
75f
tail
length

Contents

Introduction

R implements a dialect of the S language that was developed at AT&T Bell Laboratories by Rick Becker, John Chambers and Allan Wilks. Versions of R are available, at no cost, for 32-bit versions of Microsoft Windows for Linux, for Unix and for Macintosh systems 8.6 or later. It is available through the Comprehensive R Archive Network (CRAN). Web addresses are given below.

The citation for John Chambers' 1998 Association for Computing Machinery Software award stated that S has "forever altered how people analyze, visualize and manipulate data." The R project enlarges on the ideas and insights that generated the S language.

Here are points relating to the use of R that potential users might consider:

1. R has extensive and powerful graphics abilities, that are tightly linked with its analytic abilities.
2. Although there is no official support for R, its informal support network, accessible from the r-help mailing list, can be highly effective.
3. Simple calculations and analyses can be handled straightforwardly, albeit (in the current version) using a command line interface. Chapters 1 and 2 are intended to give the flavour of what is possible without getting deeply into the R language. If simple methods prove inadequate, there can be recourse to the huge range of more advanced abilities that R offers. Adaptation of available abilities allows even greater flexibility.
4. The R community is widely drawn, from application area specialists as well as statistical specialists. It is a community that is sensitive to the potential for misuse of statistical techniques and suspicious of what might appear to be mindless use. Expect scepticism of the use of models that are not susceptible to some minimal form of data-based validation.
5. Because R is free, users have no right to expect attention, on the r-help list or elsewhere, to queries. Be grateful for whatever help is given.

There is no substitute for experience and expert knowledge, even when the statistical analysis task may seem straightforward. Neither R nor any other statistical system will give the statistical expertise that is needed to use sophisticated abilities, or to know when naïve methods are not enough. Experience with the use of R is however, more than with most systems, likely to be an educational experience.

While R is as reliable as any statistical software that is available, and exposed to higher standards of scrutiny than most other systems, there are traps that call for special care. Many of the model fitting routines in R are leading edge. There may be a limited tradition of experience of the limitations and potential pitfalls of some of the newer abilities. Whatever the statistical system, and especially when there is some element of complication, check each step with care.

Hurrah for the R development team!

The Use of these Notes

1. Starting Up

R must be installed on your system! If it is not, follow the installation instructions appropriate to the operating system. Installation is now especially straightforward for Windows users. Copy down the latest *SetupR.exe* from the relevant *base* directory on the nearest CRAN site, click on its icon to start installation, and follow instructions. Libraries that do not come with the base distribution must be downloaded and installed separately.

It pays to have a separate workspace directory for each major project. For more details. see the README file that is included with the R distribution. Users of Microsoft Windows may wish to create a separate icon for each such workspace. First create the directory that will be used for the new workspace. Then right click|copy[1] to

Linix/Unix than under Windows. Under Microsoft Windows, an attractive option is to use a utility that is designed for use with the shareware WinEdt editor[4].

1.2 Using the Console (or Command Line) Window

Fig. 1 showed the console window when it was first opened.

The command line prompt, i.e. the **>**, is an invitation to start typing in your commands. For example, type in **2+2** and press the **Enter** key. Here is what I get on my screen:

```
> 2+2
[1] 4
>
```

Here the result is 4. The **[1]** says, a little strangely, "first requested element will follow". Here, there is just one element. The **>** indicates that R is ready for another command.

The exit or quit command is

```
> q()
```

Alternatives are to click on the **File** menu and then on **Exit**, or to click on the ✕ in the top right hand corner of the R window. There will be a message asking whether to save the workspace image. Clicking **Yes** (the safe option) will save all the objects that remain in the workspace – any that were there at the start of the session and any that have been added since.

1.3 A Short R Session

We will read into R a file that holds the population figures for Australian states and territories, and the total

We will now do a plot of the ACT population between 1917 and 1997. We will first of all remind ourselves of the colum nams:

```
> names(austpop)
 [1] "Year"  "NSW"   "Vic."  "Qld"   "SA"    "WA"    "Tas."  "NT"
 [9] "ACT"   "Aust."
```

A simple way to gt thlot is:

```
> plot(ACT ~ Year, data=austpop, pch=16)
```

The option **pch=16** sets tholid black. Fig 4 shs thaph

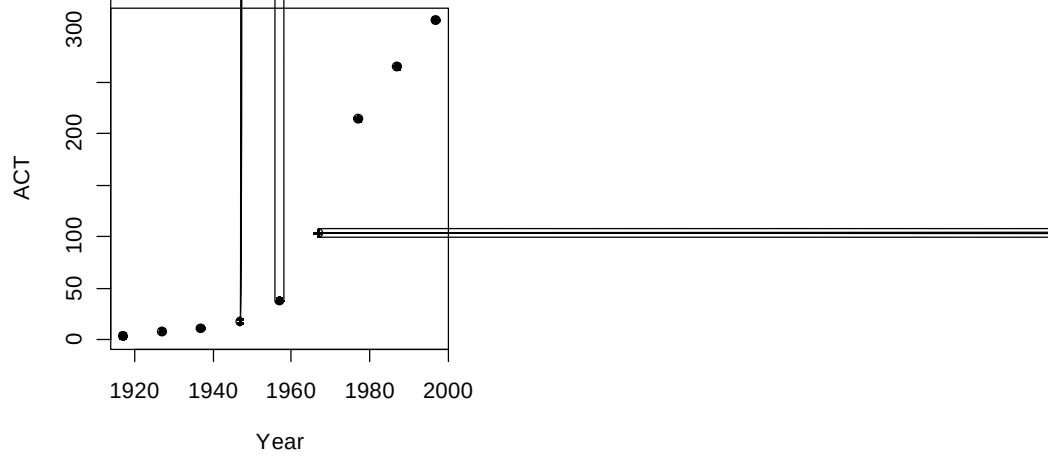


Figure 4: ACT population, at various times between 1917 and 1997.

This plot can be improved greatly. We can specify informative axis labels, change size of the text and of the plotting symbol, and so on

1.3.1 Entry of Data at the Command Line

A data frame is a rectangular array of columns of data. Here we have two columns, and both columns will be numeric. The following data gives, for each number, by which an elastic band is stretched over the end of a ruler, the distance that the band moved when released:

Stretch (m	
46	148
54	182
48	173
50	166
44	109
42	141
52	166

One can use **data.frame()** to input these (or other) data directly at the command line. We will give the data frame the name **elasticband**:

```
elasticband <- data.frame(stretch=c(46,54,48,50,44,42,52),
  distance=c(148,182,173,166,109,141,166))
```

1.3.2 Options for use of read.table()

The function **read.table()** takes, optionally various parameters additional to the file name that holds the data. Specify **header=TRUE** if there is an initial row of header names. The default is **header=FALSE**. In addition t9.5Sf7.2to0ty9.1(e)here sep5(e)-0.9(u)ra aracrs fa(at)7.6(aracrs T)C-5(f)19.8(e)19.8(e)-0.9(u)6(to-5(it alrs 7(a)-tiv(eres

space are **`sep=","`** and **`sep="\t"`**. This last choice makes tabs separators. Similarly, users can control over the choice of missing value character or characters, which by default is **`NA`**. If the missing value character is a period (“.”), specify **`na.strings="."`**.

R has several variants of **`read.table()`** that differ only in having different default parameter settings. Note in particular **`read.csv()`**, which has settings that are suitable for comma delimited (csv) files that have been generated from Excel spreadsheets.

If **`read.table()`**

1.6 Exercise

1. In the data frame **elasticband** from section 1.3.1, plot **distance** against **stretch**.

2. The following ten observations, taken during the years 1970-79, are on October snow cover for Eurasia. (Snow cover is in millions of square kilometers):

```
year snow.cover
1970 6.5
1971 12.0
1972 14.9
1973 10.0
1974 10.7
1975 7.9
1976 21.9
1977 12.5
1978 14.5
1979 9.2
```

i. Enter the data into R. [Section 1.3.1 showed one way to do this. To save keystrokes, enter the successive years as **1970:1979**]

ii. Plot **snow.cover** versus **year**.

iii Use the **hist()** command to plot a histogram of the snow cover values.

iv. Repeat ii and iii after taking logarithms of snow cover.

3. Input the following data, on damage that had occurred in space shuttle launches prior to the disastrous launch of Jan 28 1986. These are the data, for 6 launches out of 24, that were included in the pre-launch charts that were used in deciding whether to proceed with the launch. (Data for the 23 launches where information is available is in the data set **orings** that accompanies these notes.)

Temperature	Erosion	Blowby	Total

2. An Overview of R

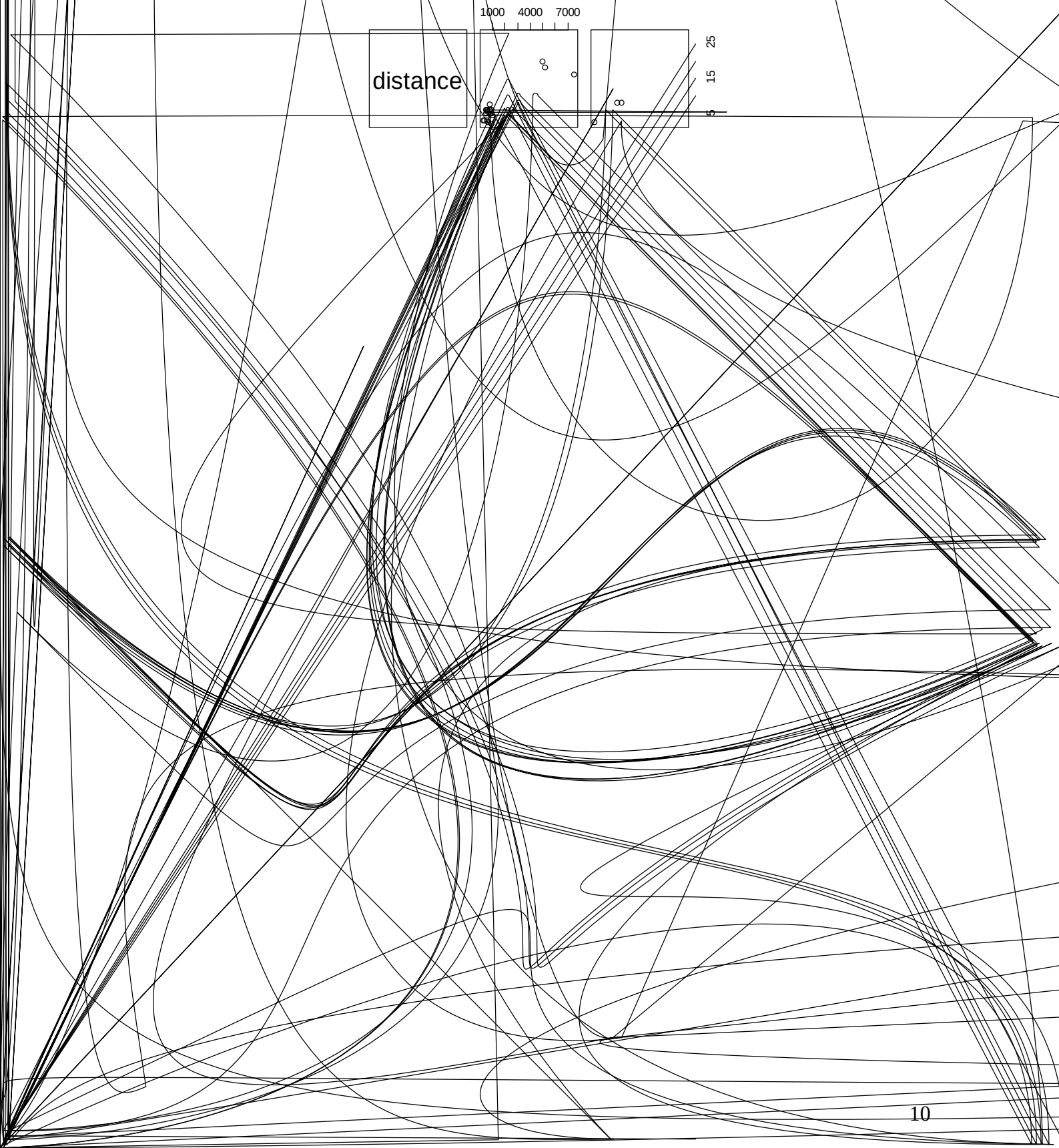
2.1 The Uses of R

2.1.3 R has extensive graphical abilities

The main R graphics function is **`plot()`**. In addition to **`plot()`** there are functions for adding points and lines to existing graphs, for placing text at specified positions, for specifying tick marks and tick labels, for labelling axes, and so on.

There are various other alternative helpful forms of graphical summary. A helpful graphical summary for the **`hills`** data frame is the scatterplot matrix, shown in Fig. 5. For this, type:

```
> pairs(hills)
```



2.3 R Objects

All R entities, including functions and data structures, exist as objects. They can all be operated on as data. Type in **`ls()`** to see the names of all objects in your workspace. An alternative to **`ls()`** is **`objects()`**. In both cases there is provision to specify a particular pattern, e.g. starting with the letter `p'[8].

Typing the name of an object causes the printing of its contents. Try typing **`q`**, **`mean`**, etc.

Important: On quitting, R offers the option of saving the workspace image. This allows the retention, for use in the next session in the same workspace, any objects that were created in the current session. Careful housekeeping may be needed to distinguish between objects that are to be kept and objects that will not be used again. Before typing **`q()`** to quit, use **`rm()`** to remove objects that are no longer required. Saving the

There is a much easier (and better) way to do this calculation:

```
> sum(c(31,51,91))
[1] 173
```

Skilled R users have limited recourse to loops. There are often, as in the example above, better alternatives.

2.5 R Functions

We give two simple examples of R functions.

2.5.1 An Approximate Miles to Kilometers Conversion

```
miles.to.km <- function(miles)miles*8/5
```

The return value is the value of the final (and in this instance only) expression that appears in the function body[11]. Use the function thus

```
> miles.to.km(175)  # Approximate distance to Sydney, in miles
[1] 280
```

The function will do the conversion for several distances all at once. To convert a vector of the three distances 100, 200 and 300 miles to distances in kilometers, specify:

```
> miles.to.km(c(100,200,300))
[1] 160 320 480
```

2.5.2 A Plotting function

The data set `florida` has the votes in the 2000 election for the various US Presidential candidates, county by county in the state of Florida. The following plots the vote for Buchanan against the vote for Bush.

```
attach(florida)
plot(BUSH, BUCHANAN, xlab="Bush", ylab="Buchanan")
detach(florida)   # In S-PLUS, specify detach("florida")
```

Here is a function that makes it possible to plot the figures for any pair of candidates.

```
plot.florida <- function(xvar="BUSH", yvar="BUCHANAN"){
x <- florida[,xvar]
y<- florida[,yvar]
plot(x, y, xlab=xvar,ylab=yvar)
m0.0267 TD-0.TD-n-3, line=1.75, .0267 TD(plot(x, y, xlTD-)Tj-(n-3, line=1.75, .026)Tj-0.0267 0
```

The concatenate function **c()** may also be used to join lists.

2.6.2 Subsets of Vectors

There are two common ways to extract subsets of vectors[13].

1. Specify the numbers of the elements that are to be extracted, e.g.

```
> x <- c(3,11,8,15,12)  # Assign to x the values 3, 11, 8, 15, 12
> x[c(2,4)]   # Extract elements (rows) 2 and 4
[1] 11 15
```

One can use negative numbers to omit elements:

```
> x <- c(3,11,8,15,12)
> x[-c(2,3)]
[1]  3 15 12
```

2. Specify a vector of logical values. The elements that are extracted are those for which the logical value is **T**. Thus suppose we want to extract values of **x** that are greater than 10.

```
> x>10  # This generates a vector of logical (T or F)
```

```
gender <- c(rep("female",691), rep("male",692))
```

(The usage is that **rep("female", 691)** creates 691 copies of the character string "female", and similarly for the creation of 692 copies of "male".)

We can change the vector to a factor, by entering:

```
gender <- factor(gender)
```

Internally the factor **gender** is stored as 691 1's, followed by 692 2's. It has stored with it a table that looks like this:

1	female
2	male

Once stored as a factor, the space required for storage is reduced.

Whenever the context seems to demand a character string, the 1 is translated into "female" and the 2 into "male". The values "female" and "male" are the *levels* of the factor. By default, the levels are in alphanumeric order, so that "female" precedes "male". Hence:

```
> levels(gender)  # Assumes gender is a factor, created as above
[1] "female" "male"
```

The order of the levels in a factor determines the order in which the levels appear in graphs that use this information, and in tables. To cause "male" to come before "female", use

```
gender <- relevel(gender, ref="male")
```

An alternative is

```
gender <- factor(gender, levels=c("male", "female"))
```

This last syntax is available both when the factor is first created, or later when one wishes to change the order of levels in an existing factor. Incorrect spelling of the level names will generate an error message. Try

```
gender <- factor(c(rep("female",691), rep("male",692)))
table(gender)
gender <- factor(gender, levels=c("male", "female"))
table(gender)
gender <- factor(gender, levels=c("Male", "female"))
                # Erroneous
```

of passengers for cars in this category), **Max.passengers, No.of.cars**., and **abbrev**. The first three columns have mode numeric, and the fourth has mode character. Columns can be vectors of any mode. The column **abbrev** could equally well be stored as a factor.

Any of the following[15] will pick out the fourth column of the data frame **Cars93.summary**, then storing it in the vector **type**.

```
type <- Cars93.summary$abbrev
type <- Cars93.summary[,4]
type <- Cars93.summary[,"abbrev"]
type <- Cars93.summary[[4]]    # Take6.2e objr type# Take6.2e objr   Car# Take6.2e objr
```

2.8 Common Useful Functions

```
print()    # Prints a single R object
cat()      # Prints multiple objects, one after the other
length()   # Number of elements in a vector or of a list
mean()
median()
range()
unique()   # Gives the vector of distinct values
diff()     # Replace a vector by the vector of first differences
           # N. B. diff(x) has one less element than x
sort()     # Sort elements into order, but omitting NAs
order()    # x[order(x)] orders elements of x, with NAs last
cumsum()
cumprod()
rev()      # reverse the order of vector elements
```

The functions `mean()`, `median()`, `range()`, and a number of other functions, take the argument `na.rm=T;` i.e. remove NAs, then proceed with the calculation.

By default, `sort()` omits any NAs. The function `order()` places NAs last. Hence:

```
> x <- c(1, 20,  2, NA, 22)
> order(x)
[1] 1 3 2 5 4
> x[order(x)]
[1]  1  2 20 22 NA
> sort(x)
[1]  1  2 20 22
```

2.8.1 Applying a function to all columns of a data frame

The function `sapply()` does this. It takes as arguments the name of the data frame, and the function that is to be applied. Here are examples, using the supplied data set `rainforest` [18].

```
> sapply(rainforest, is.factor)
    dbh    wood    bark    root  rootsk  branch species
  FALSE   FALSE   FALSE   FALSE   FALSE   FALSE    TRUE
> sapply(rainforest[,-7], range)   # The final column (7) is a factor
     dbh wood bark root rootsk branch
[1,]   4   NA   NA   NA     NA     NA
[2,]  56   NA   NA   NA     NA     NA
```

The functions mean and range, and several of the other functions noted above, have parameters `na.rm`. For

```
[1] ".GlobalEnv"   "Autoloads"    "package:base"
```

3. Plotting

3.2 Fine control – Parameter settings

The default settings of parameters, such as character size, are often adequate. When it is necessary to change parameter settings for a subsequent plot, the **par()** function does this. For example,

```
par(cex=1.25, mex=1.25)   # character (cex) & margin (mex) expansion
```

increases the text and plot symbol size 25% above the default. The addition of **mex=1.25** makes room in the margin to accommodate the increased text size.

On the first use of **par()** to make changes to the current device, it is often useful to store existing settings, so that they can be restored later. For this, specify

```
oldpar <- par(cex=1.25, mex=1.25)
```

This **oldpar**, then changes parameters (here **cex** and **mex**) as requested. To restore the original parameter settings at some later time, enter **par(oldpar)**. Here is an example:

```
attach(elasticband)
oldpar <- par(cex=1.5, mex=1.5)
plot(distance ~ stretch)
par(oldpar)         # Restores the earlier settings
detach(elasticband)
```

Inside a function specify, e.g.

```
oldpar <- par(cex=1.25, mex=1.25)
on.exit(par(oldpar))
```

Type in **help(par)** to get details of all the parameter settings that are available with **par()**.

3.2.1 Multiple plots on the one page

The parameter **mfrow** can be used to configure the graphics sheet so that subsequent plots appear row by row, one after the other in a rectangular layout, on the one page. For a column by column layout, use **mfcol instead**. In the example below we present four different transformations of the primates data, in a two by two layout:

```
par(mfrow=c(2,2), pch=16)
data(Animals)     # Needed if Animals (MASS library) is not already loaded
attach(Animals)
plot(body, brain)
plot(sqrt(body), sqrt(brain))
plot((body)^0.1, (brain)^0.1)
plot(log(body),log(brain))
detach(Animals)
par(mfrow=c(1,1), pch=1)      # Restore to 1 figure per page
```

3.2.2 The shape of the graph sheet

Often it is desirable to exercise contr1 0over the shape of the graph page, e.g. so that the individual plots are rectangular rather than square. The R for Windows functions **win.graph()** or **x11()** that set up the Windows screen take the parameters **width** (in inches), **height** (in inches) and **pointsize** (in 1/72 of an inch). The setting of **pointsize** (default =12) determines character heights. It is the relative sizes of these parameters that matter for screen display or for incorporation into Word and similar programs. Graphs can be enlarged orj0runk by pointing at one corner, holding down the left mouse button, and pulling.

3.3 Adding points, lines and text

Here is a simple example that shows how to use the function **`text()`** to add text labels to the points on a plot.

```
> primates
               Bodywt Brainwt
 Potar monkey    10.0     115
      Gorilla   207.0     406
        Human    62.0    1320
Rhesus monkey     6.8     179
        Chimp    52.2     440
```

Observe that the row names store labels for each row[20].

```
> attach(primates)  # Needed if primates is not already attached.
> plot(Bodywt, Brainwt, xlim=c(5, 250))
> # Specify xlim so that there is room for the labels
> text(x=Bodywt, y=Brainwt, labels=row.names(primates), adj=0)
   # adj=0 implies left adjusted text
> detach(primates)
```

Fig. 7 shows the result.

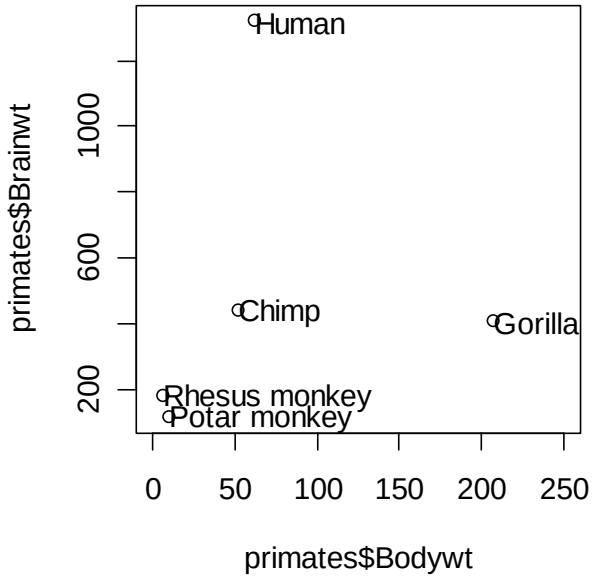


Figure 7: Plot of the primate data, with labels on points

Fig. 7 would be adequate for identifying points, but is not a presentation quality graph. We now show how to improve it.

---

[20] Row names can be created in several different ways. They can be assigned directly, e.g.

```
row.names(primates) <- c("Potar monkey","Gorilla","Human","Rhesus monkey","Chimp")
```

When using **`read.table()`** to input data, the parameter **`row.names`** is available to specify, by number or name, a column that holds the row names.

In Fig. 8 we use the **`xlab`** (x-axis) and **`ylab`** (y-axis) parameters to specify meaningful axis titles. Weecifveethe labelling to one side of the points by including appropriate horizontal and vertical offsets. We use **`chw <- par()$cxy[1]`** to get a 1-character space horizontal offset, and

```
points(1:7,rep(2,7), pch=(0:6)+7)           # Plot symbols 7 to 13
text((1:7)+0.25, rep(2,7), paste((0:6)+7))  # Label with symbol number
points(1:7,rep(1,7), pch=(0:6)+14)          # Plot symbols 14 to 20
text((1:7)+0.25, rep(1,7), paste((0:6)+14)) # Labels with symbol number
```

Here is the plot:

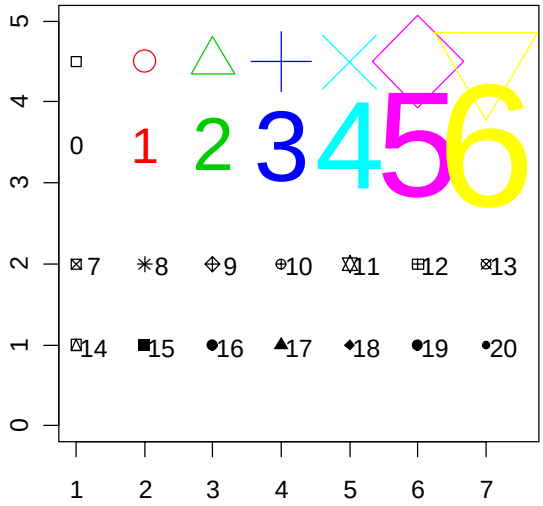


Figure 9: Different plot symbols, colours and sizes

A variety of color palettes are available. Iere is a function that displays some of the possibilities:

```
view.colours <- function(){
   plot(1, 1, xlim=c(0,14), ylim=c(0,3), type="n", axes=F, xlab="",ylab="")
   text(1:6, rep(2.5,6), paste(1:6), col=palette()[1:6], cex=2.5)
   text(10, 2.5, "Default palette", adj=0)
   rainchars <- c("R","O","Y","G","B","I","V")
   text(1:7, rep(1.5,7), rainchars, col=rainbow(7), cex=2.5)
   text(10, 1.5, "rainbow(7)", adj=0)
   cmtxt <- substring("cm.colors", 1:9,1:9)
      # Split “cm.colors” into its 9 characters
   text(1:9, rep(0.5,9), cmtxt, col=cm.colors(9), cex=3)
   text(10, 0.5, "cm.colors(9)", adj=0)
}
```

To run the function, enter

```
view.colours()
```

3.3.2 Adding Text in the Margin

`mtext(side, line, text, ..)` adds text in the margin of the current plot. The sides are numbered 1(x-axis), 2(y-axis), 3(top) and 4.

3.4 Identification and Location on the Figure Region

Two functions are available foigure Region

- locator() prints out the co-ordinates of points. One positions the cursor at the location for which coordinates are required, and clicks the left mouse button.

A click with the right mouse button signifies that the identification or location task is complete, unless the setting of the parameter **n** is reached first. For `identify()`

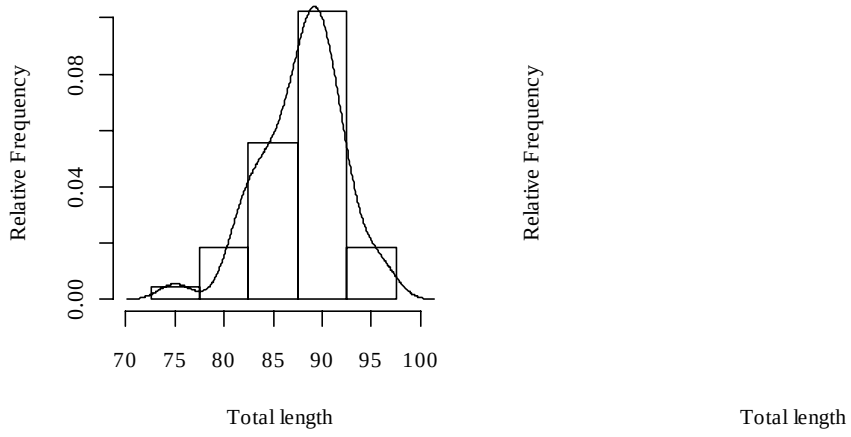
Figure 10: The two graphs show the same data, but with a different choice of breakpoints.

Here is the code used to plot the histograms:

```
par(mfrow = c(1, 2))
attach(possum)
here <- sex == "f"
hist(totlngth[here], breaks = 72.5 + (0:5) * 5, ylim = c(0, 22),
          xlab="Total length", main ="A: Breaks at 72.5, 77.5, ...")
hist(totlngth[here], breaks = 75 + (0:5) * 5, ylim = c(0, 22),
          xlab="Total length", main="B: Breaks at 75, 80, ...")
par(mfrow=c(1,1))
detach(possum)
```

3.5.2 Density Plots

Density plots, now that they are available, are often a preferred alternative to a histogram. In Fig. 11 the histograms from Figure 10 are overlaid with a density plot.



```
lines(dens)
detach(possum)
```

3.5.3 Boxplots

We now make a boxplot of the above data:

```
attach(possum)
```

3.6.2 Adding lines to plots

Use the function `abline()T4.8193ine()-1.1ts`

```
data(islands)     # if not already loaded
dotplot(islands)  # vector of named numeric values
```

Unfortunately there are many names, and there is substantial overlap. The following is better, but shrinks the sizes of the points so that they almost disappear:

```
dotplot(islands, cex=0.2)
```

3.7 Plotting Mathematical Symbols

Both **`text()`** and **`mtext()`** will take an expression rather than a text string. **`In plot()`**, either or both of **`xlab`** and **`ylab`** can be an expression. Fig. 15 was produced with

```
plot(x, y, xlab="Radius", ylab=expression(Area == pi*r^2))
```

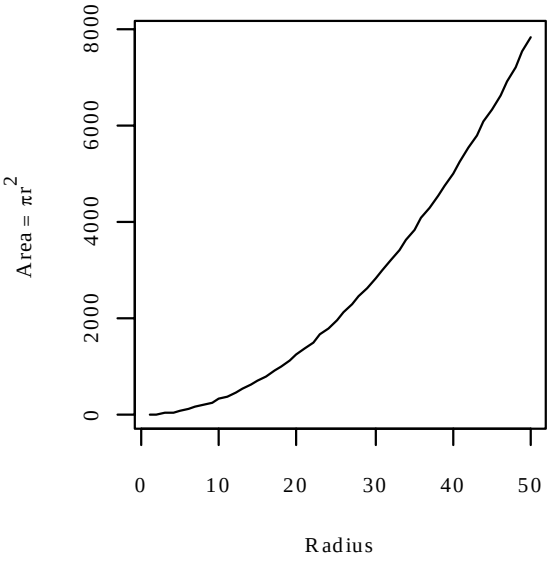


Figure 15: The y-axis label is a mathematical expression.

Notice that in **`expression(Area == pi*r^2)`**, there is a double equals sign ("**`==`**"), although what will appear on the plot is Area = pi*r^2, with a single equals sign. The reason for this is that **`Area == pi*r^2`** is a valid mathematical expression, while **`Area = pi*r^2`** is not.

See help(plotmath) for detailed information on the plotting of mathematical expressions. There is a further example in chapter 12.

The final plot from

```
demo(graphics)
```

shows some of the possibilities for plotting mathematical symbols.

3.8 Guidelines for Graphs

Design graphs to make their point tersely and clearly, with a minimum waste of ink. Label as necessary to identify important features. In scatterplots the graph should attract the eye's attention to the points that are plotted, and to important grouping in the data. Use solid points, large enough to stand out relative to other features, when there is little or no overlap.

When there is extensive overlap of plotting symbols, use open plotting symbols. Where points are dense, overlapping points will give a high ink density, which is exactly what one wants.

Use scatterplots in preference to bar or related graphs whenever the horizontal axis represents a quantitative effect.

Use graphs from which information can be read directly and easily in preference to those that rely on visual impression and perspective. Thus in scientific papers contour plots are much preferable to surface plots or two-dimensional bar graphs.

Draw graphs so that reduction and reproduction will not interfere with visual clarity.

Explain clearly how error bars should be interpreted — ± SE limits, ± 95% confidence interval, ± SD limits, or whatever. Explain what source of `error(s)' is represented. It is pointless to present information on a source of error that is of little or no interest, for example analytical error when the relevant source of `error' for comparison of treatments is between fruit.

Use colour or different plotting symbols to distinguish different groups. Take care to use colours that contrast.

The list of references at the end of this chapter has further comments on graphical and other presentation issues.

3.9 Exercises

1. Plot the graph of brain weight (**`brain`**) versus body weight (**`body`**) for the data set **`Animals`** `from the` *`MASS`* `library`. Label the axes appropriately.
[To access this data frame, specify **`library(mass); data(Animals)`**]

2. Repeat the plot 1, but this time plotting log(brain weight) versus log(body weight). Use the row labels to label

6. Try **`x <- rnorm(10)`**. Print out the numbers that you get. Look up the help for **`rnorm`**. Now generate a sample of size 10 from a normal distribution with mean 170 and standard deviation 4.

7. Use **`mfrow()`** to set up the layout for a 3 by 4 array of plots. In the top 4 rows, show normal probability plots (section 3.4.2) for four separate `random' samples of size 10, all from a normal distribution. In the middle 4 rows, display plots for samples of size 100. In the bottom four rows, display plots for samples of size 100e 10 Comment on how the appearance of the plots changes as the sample size changes.

8. The function **`runif()`** can be used to generate a sample from a uniform distribution, by default on the

4. Lattice graphics, and coplot()

Lattice plots allow the use of the layout on the page to reflect meaningful aspects of data structure. They offer abilities similar to those in the S-PLUS *trellis* library0

At the time of writing the implementation of the lattice library was incomplete, though already with impressive functionality. The *lattice* library sits on top of the *grid* library. To use lattice graphics, both these libraries must be installed. Providing it is installed, the grid library will be loaded automatically when lattice is loaded.

The older **`coplot()`** function that is in the *base* library has some of same abilities as **`xyplot( )`**, but with a limitation to two conditioning factors or variables only.

4.1 Examples that Present Panels of Scatterplots – Using `xyplot()`

The basic function for drawing panels of scatterplots is **`xyplot()`**. We will use the data frame **`tinting`** (supplied) to demonstrate the use of **`xyplot()`**. These data are from an experiment that investigated the effects of tinting of car windows on visual performance[24]

```
xyplot(csoa ~ it | sex * agegp, data=tinting) # Simple use of xyplot()
```

Here is the statement used to get Fig. 16. The two different symbols distinguish between low contrast and high contrast targets.

In each instance, one can add conditioning variables.

4.2 Using `coplot()`

The *lattice* library makes **`coplot()`** largely redundant. It may still be useful if the lattice library is not available, or if one wants its particular layout and labelling.

We again use data from the data frame **`tinting`**

5. Linear (Multiple Regression) Models and Analysis of Variance

5.1 The Model Formula in Straight Line Regression

We begin with the straight line regression example that appeared earlier, in section 2.1.4. First we plot the data:

```
plot(distance ~ stretch, data=elasticband)
```

The code for the regression calculation is:

```
elastic.lm <- lm(distance ~ stretch, data=elasticband)
```

Here `distance ~ stretch` is a model formula. Other model formulae will appear in the course of this chapter. Fig. 18 shows the plot:

```
Residual standard error: 16.3 on 5 degrees of freedom
Multiple R-Squared: 0.635,      Adjusted R-squared: 0.562
F-statistic: 8.71 on 1 and 5 degrees of freedom,        p-value: 0.0319
```

5.2 Regression Objects

An **lm** object is a list of named elements. Above, we created the object **elastic.lm** . Here are the names of its elements:

```
> names(elastic.lm)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

Various functions are available for extracting information that you might want from the list. This is better than manipulating the list directly. Examples are:

```
> coef(elastic.lm)
(Intercept)     stretch
    -63.571       4.554
> resid(elastic.lm)
       1        2        3        4        5        6        7
  2.1071  -0.3214  18.0000   1.8929 -27.7857  13.3214  -7.2143
```

The function most often used to inspect regression output is `summary()`

```
par(mfrow = c(2, 2))
plot(elastic.lm)
```

The following are the fitted values and residuals that we get with the estimates of a (= -63.6) and b (= 4.55) that result from least squares regression:

X		$\hat{y}$	y	$y - \hat{y}$
	Stretch (mm)	*(Fitted)*	*(Observed)*	*(Residual)*
× -63.6	× 4.55	1 × -63.6 + 4.55 × Stretch	Distance (mm)	Observed - Fitted
1	46	-63.6 + 4.55 × 46 = 145.7	148	148-145.7 = 2.3
1	54	-63.6 + 4.55 × 54 = 182.1	182	182-182.1 = -0.1
1	48	-63.6 + 4.55 × 48 = 154.8	173	173-154.8 = 18.2
1	50	-63.6 + 4.55 × 50 = 163.9	166	166-163.9 = 2.1
1	44	-63.6 + 4.55 × 44 = 136.6	109	109-136.6 = -27.6
1	42	-63.6 + 4.55 × 42 = 127.5	141	141-127.5 = 13.5
1	52	-63.6 + 4.55 × 52 = 173.0	166	166-173.0 = -7.0

Note that we use $\hat{y}$ [pronounced y-hat] as the symbol for predicted values.

We might alternatively fit the simpler (no intercept) model. For this we have

$$y = x \times b + e$$

where e is a random variable with mean 0. The X matrix then consists of a single column, the x's.

5.3.1 Model Formulae in General

Model formulae take a form such as:

`y~x+z` : lm, glm,, etc.

`y~x + fac + fac:x` : lm, glm, aov, etc. (If **`fac`** is a factor and **`x`** is a variable, **`fac:x`** allows a different slope for each different level of **`fac`**.)

Model formulaeare widely used to set up most of the model calculations in R.

Notice the similarity between model formulaeand the formulaethat are used for specifying coplots. Thus, recall that the graph formula for a coplot that gives a plot of **`y`** against **`x`** for each different combination of levels of **`fac1`** (across the page) and **`fac2`** (up the page) is:

```
y ~ x | fac1+fac2
```

*5.3.2 Manipulating Model Formulae

Model formulae can be assigned, e.g.

```
formyxz <- formula(y~x+z)
```

or

```
formyxz <- formula("y~x+z")
```

The argument to **`formula()`**

```
> formds <- formula(paste(nam[1],"~",nam[2]))
> lm(formds, data=elasticband)

Call:
lm(formula = formds, data = elasticband)

Coefficients:
(Intercept)     distance
    26.3780       0.1395
```

Note that graphics formulae can be manipulated in exactly the same way as model formulae.

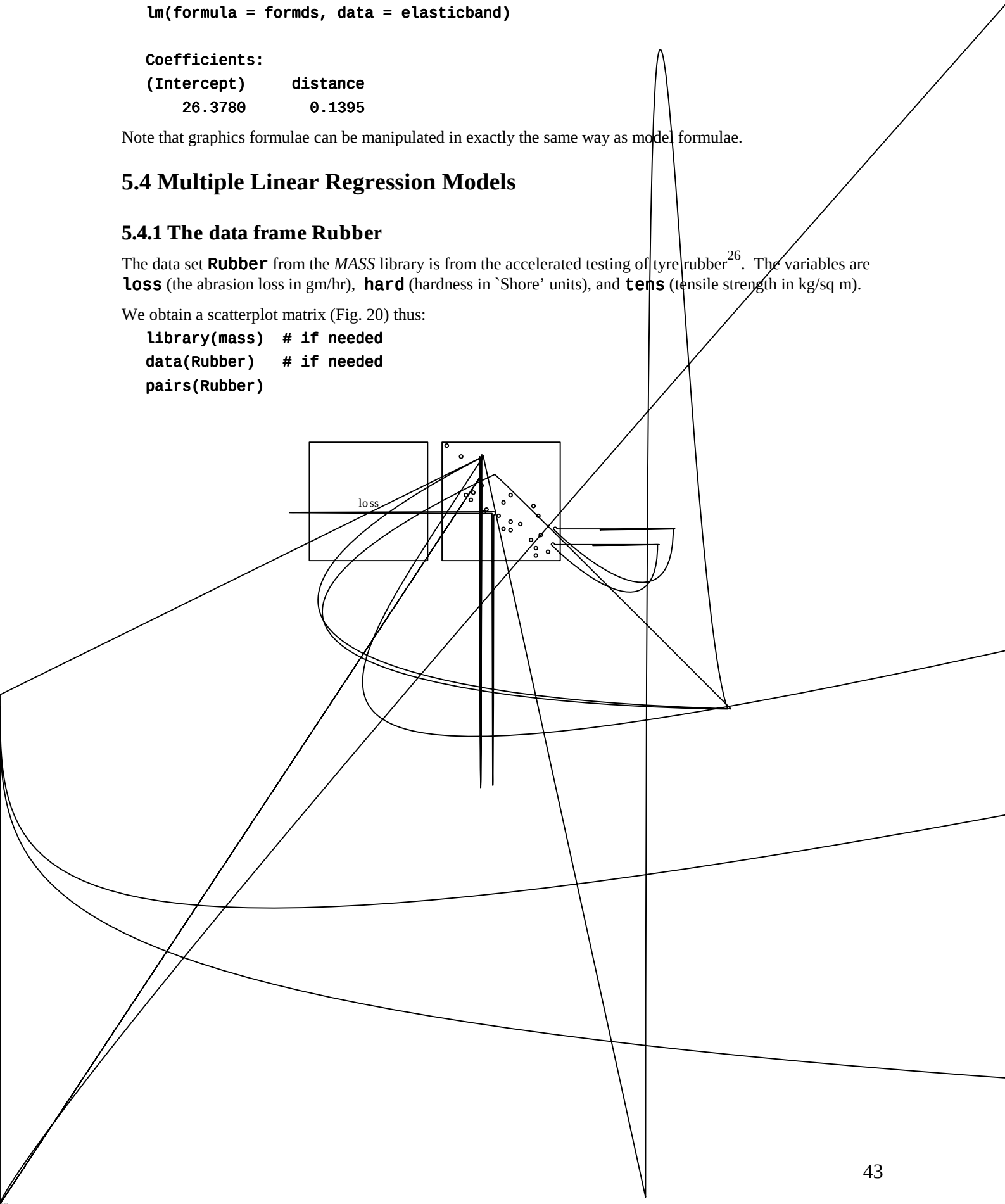
5.4 Multiple Linear Regression Models

5.4.1 The data frame Rubber

The data set **Rubber** from the *MASS* library is from the accelerated testing of tyre rubber[26]. The variables are **loss** (the abrasion loss in gm/hr), **hard** (hardness in `Shore' units), and **tens** (tensile strength in kg/sq m).

We obtain a scatterplot matrix (Fig. 20) thus:

```
library(mass)  # if needed
data(Rubber)   # if needed
pairs(Rubber)
```



```
Rubber.lm <- lm(loss~hard+tens, data=Rubber)
> options(digits=3)
> summary(Rubber.lm)

Call:
lm(formula = loss ~ hard + tens, data = Rubber)
```

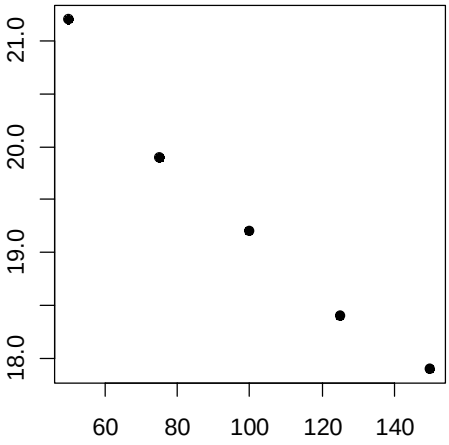
```
> summary(logbooks.lm2)$coef
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -1.263      3.552  -0.356   0.7303
thick          0.313      0.472   0.662   0.5243
height         2.114      0.678   3.117   0.0124

> logbooks.lm3<-lm(weight~thick+height+width,data=logbooks)
> summary(logbooks.lm3)$coef
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -0.719      3.216  -0.224    0.829
thick          0.465      0.434   1.070    0.316
height         0.154      1.273   0.121    0.907
width          1.877      1.070   1.755    0.117
```

So is **weight** proportional to **thick * height * width**?

The correlations between **thick**, **height** and **width** are so strong that if one tries to use more than one of them as a explanatory variables, the coefficients are ill-determined. They contain very similar information, as is evident from the scatterplot matrix. The regressions on **height** and **width** give plausible results, while the coefficient of the regression on **thick** is entirely an artefact of the way that the books were selected.

The design of the data collection really is important for the interpretation of coefficients from a regression


21.0
20.0
19.0
18.0
60
80
100
120
140

```
4           1  125     15625
5           1  150     22500
attr(,"assign")
[1] 0 1 2
```

This example demonstrates a way to extend linear models to handle specific types of non-linear relationships. We can use any transformation we wish to form columns of the model matrix. We could, if we wished, add an x^3 column.

Once the model matrix has been formed, we are limited to taking linear combinations of columns.

5.5.2 What order of polynomial?

A polynomial of degree 2, i.e. a quadratic curve, looked about right for the above data. How does one check?

One way is to fit polynomials, e.g. of each of degrees 1 and 2, and compare them thus:

```
> seedrates.lm1<-lm(grain~rate,data=seedrates)
> seedrates.lm2<-lm(grain~rate+I(rate^2),data=seedrates)
> anova(seedrates.lm2,seedrates.lm1)
Analysis of Variance Table

Model 1: grain ~ rate + I(rate^2)
Model 2: grain ~ rate
  Res.Df Res.Sum Sq Df    Sum Sq F value  Pr(>F)
1      2   0.026286
2      3   0.187000 -1 -0.160714  12.228 0.07294
```

The F-value is large, but on this evidence there are too few degrees of freedom to make a totally convincing case for preferring a quadratic to a line. However the paper from which these data come gives an independent estimate of the error mean square (0.17 on 35 d.f.) based on 8 replicate results that were averaged to give each value for number of grains per head. If we compare the change in the sum of squares (0.1607, on 1 df) with a mean square of 0.17^2 (35 df), the F-value is now 5.4 on 1 and 35 degrees of freedom, and we have p=0.024 . The increase in the number of degrees of freedom more than compensates for the reduction in the F-statistic.

```
> # However we have an independent estimate of the error mean
> # square. The estimate is 0.17^2, on 35 df.
> 1-pf(0.16/0.17^2, 1, 35)
[1] 0.0244
```

Finally note that R^2 was 0.972 for the straight line model. This may seem good, but given the accuracy of these data it was not good enough! The statistic is an inadequate guide to whether a model is adequate. Even for any one context, R^2 will in general increase as the range of the values of the dependent variable increases. (R^2 is larger when there is more variation to be explained.) A predictive model is adequate when the standard errors of predicted values are acceptably small, not when R^2 achieves some magic threshold.

5.5.3 Pointwise confidence bounds for the fitted curve

Here is code that will give pointwise 95% confidence bounds. Note that these do not combine to give a confidence region for the total curve! The construction of such a region is a much more complicated task!

```
plot(grain ~ rate, data = seedrates, pch = 16, xlim = c(50, 175), ylim
                = c(15.5, 22),xlab="Seeding rate",ylab="Grains per head")
new.df <- data.frame(rate = c((4:14) * 12.5))
seedrates.lm2 <- lm(grain ~ rate + I(rate^2), data = seedrates)
pred2 <- predict(seedrates.lm2, newdata = new.df, interval="confidence")
hat2 <- data.frame(fit=pred2[,"fit"],lower=pred2[,"lwr"],
  upper=pred2[,"upr"])
attach(new.df)
```

```
lines(rate, hat2$fit)
lines(rate,hat2$lower,lty=2)
lines(rate, hat2$upper,lty=2)
detach(new.df)
```

The extrapolation has deliberately been taken beyond the range of the data, in order to show how the confidence bounds spread out. Confidence bounds for a fitted line spread out more slowly, but are even less believable!

5.5.4 Spline Terms in Linear Models

By now, readers of this document will be used to the idea that it is possible to use linear models to fit terms that may be highly nonlinear functions of one or more of the variables. The fitting of polynomial functions was a

```
> insulation <-      - factor(c(rep("without", 8), rep("with", 7)))
```

*5.6.2 Other Choices of Contrasts

There are other ways to set up the X matrix. In technical jargon, there are other choices of *contrasts*. One obvious alternative is to make **without** the first factor level, so that it becomes the baseline. For this, specify:

```
insulation <- relevel(insulation, baseline="without")
  # Make `without' the baseline
```

Anotherpossibility is to use what are called the "sum" contrasts. With the "sum" contrasts the bae is the mean over all factor levels. The effect for the first level is omitted; the user has to calculate it as minus the sum of the remaining effects. Here is the output from use of the `sum' contrasts[30]:

```
> options(contrasts = c("contr.sum", "contr.poly"), digits = 2)
   #  Try the `sum' contrasts
> insulation <- factor(insulation, levels=c("without", "with"))
   # Make `without' the baseline
> insulation.lm <- lm(kWh ~ insulation)
> summary(insulation.lm, corr=F)

Call:
lm(formula = kWh ~ insulation)

Residuals:
   Min     1Q Median     3Q    Max
 -4409   -979    132   1575   3690

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)     9442        598   15.78  7.4e-10
insulation      1551        598    2.59    0.022

Residual standard error: 2310 on 13 degrees of freedom
Multiple R-Squared: 0.341,      Adjusted R-squared: 0.29
F-statistic: 6.73 on 1 and 13 degrees of freedom,   p-value: 0.0223
```

Here is the interpretation:

average of (mean for "without", "mean for with") = 9442

To get the estimate for uninsulated houses (the first level), take 9442 + 1551 = 10993

The `effects' sum to one. So the effect for the second level (`with') is -1551. Thus

to get the estimate for insulated houses (the first level), take 9442 - 1551 = 7980.

The sum contrasts are sometimes called "analysis of variance" contrasts.

You can set the choice of contrasts for each factor separately, with a statement such as:

```
insulation <- C(insulation, contr=treatment)
```

Also available are the Helmert contrasts. These are not at all intuitive and rarely helpful, even though S-PLUS uses them as the default. Novices should avoid them[31].

---

[30] The second string element, i.e. **"contr.poly"**, is the default setting for factors with ordered levels. [One uses the function ordered() to create ordered factors.]

[31] The interpretation of the helmert contrasts is simple enough when there are just two levels. With >2 levels,

5.7 Multiple Lines – Different Regression Lines for Different Species

The terms that appear on the right of the model formula may be variables or factors, or interactions between variables and factors, or interactions between factors. Here we take advantage of this to fit different lines to different subsets of the data.

In the example that follows, we had weights for a porpoise species (*Stellena styx*) and for a dolphin species (*Delphinus delphis*). We take x_1 to be a variable that has the value 0 for *Delphinus delphis*, and 1 for *Stellena styx*. We take x_2 to be body weight. Then possibilities we may want to consider are:

A: A single line: $y = a + b\,x_{2}$ x; 2 Tm (2)Tj 504.638 0 Td [(+)-1015(e)] TJ 0 -15 Td (1.37965 6 (a) 4.4889 nse)- 67458n alui

```
> model.matrix(cet.lm2)
   (Intercept) factor(species) logweight
1            1               1     3.555
2            1               1     3.738
. . . .
8            1               0     3.989
. . . .
16           1               0     3.951
attr(,"assign")
[1] 0 1 2
attr(,"contrasts")
[1] "contr.treatment"
```

Enter **`summary(cet.lm2)`** to get an output summary, and **`plot(cet.lm2)`** to plot diagnostic information for this model.

For model C, the statement is:

```
> cet.lm3 <- lm(logheart ~ factor(species) + logweight +
    factor(species):logweight, data=dolphins)
```

Check what the model matrix looks like:

```
> model.matrix(cet.lm3)
   (Intercept) factor(species) logweight factor(species).logweight
1            1               1     3.555                     3.555
. . . .
8            1               0     3.989                     0.000
. . . .
16           1               0     3.951                     0.000
attr(,"assign")
[1] 0 1 2 3
attr(,"contrasts")
attr(,"contrasts")$`factor(species)`
[1] "contr.treatment"
```

5.8.1 Plant Growth Example

Here is a simple randomised block design:

```
> data(PlantGrowth)              # From the MASS library
> attach(PlantGrowth)
> boxplot(split(weight,group))    # Looks OK
> data()
> PlantGrowth.aov <- aov(weight~group)
> summary(PlantGrowth.aov)
            Df  Sum Sq Mean Sq F value  Pr(>F)
group        2  3.7663  1.8832  4.8461 0.01591
Residuals   27 10.4921  0.3886
> summary.lm(PlantGrowth.aov)

Call:
aov(formula = weight ~ group)

Residuals:
    Min      1Q  Median      3Q     Max
-1.0710 -0.4180 -0.0060  0.2627  1.3690

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   5.0320     0.1971  25.527   <2e-16
grouptrt1    -0.3710     0.2788  -1.331   0.1944
grouptrt2     0.4940     0.2788   1.772   0.0877

Residual standard error: 0.6234 on 27 degrees of freedom
Multiple R-Squared: 0.2641,     Adjusted R-squared: 0.2096
F-statistic: 4.846 on 2 and 27 degrees of freedom,      p-value: 0.01591

> help(cabbages)
> data(cabbages)                  # From the MASS library
> names(cabbages)
[1] "Cult"   "Date"   "HeadWt" "VitC"
> coplot(HeadWt~VitC|Cult+Date,data=cabbages)
```

Examination of the plot suggests that cultivars differ greatly in the variability in head weight. Variation in the vitamin C levels seems relatively consistent between cultivars.

```
> VitC.aov<-aov(VitC~Cult+Date,data=cabbages)
> summary(VitC.aov)
            Df  Sum Sq Mean Sq F value    Pr(>F)
Cult         1 2496.15 2496.15 53.0411 1.179e-09
Date         2  909.30  454.65  9.6609 0.0002486
Residuals   56 2635.40   47.06
```

*5.8.2 Shading of Kiwifruit Vines

These data (yields in kilograms) are in the data frame **kiwishade** that accompanies these notes. They are from [32] where there were four treatments - no shading, shading from August to December, shading from December to February, and shading from February to May. Each treatment appeared once in each of the three blocks. The northernmost plots were grouped in one block because they were similarly affected by shading from the sun. For the remaining two blocks shelter effects, in one case from the east and in the other case from the west, were thought more important. Results are given for each of the four vines in each plot. In experimental design parlance, the four vines within a plot constitute subplots.

The **block:shade** mean square (sum of squares divided by degrees of freedom) provides the error term. (If this is not specified, one still gets a correct analysis of variance breakdown. But the F-statistics and p-values will be wrong.)

```
> kiwishade$shade <- relevel(kiwishade$shade, ref="none")
> ## Make sure that the level "none" (no shade) is used as reference
> kiwishade.aov<-aov(yield~block+shade+Error(block:shade),data=kiwishade)
> summary(kiwishade.aov)

Error: block:shade
          Df  Sum Sq Mean Sq F value   Pr(>F)
block      2  172.35   86.17  4.1176 0.074879
shade      3 1394.51  464.84 22.2112 0.001194
Residuals  6  125.57   20.93

Error: Within
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 36 438.58   12.18
> coef(kiwishade.aov)
(Intercept) :
(Intercept)
    96.5327

block:shade :
  blocknorth    blockwest shadeAug2Dec shadeDec2Feb shadeFeb2May
    0.993125    -3.430000     3.030833   -10.281667    -7.428333

Within :
numeric(0)
```

5.9 Exercises

1. Here are two sets of data that were obtained the same apparatus, including the same rubber band, as the data frame elasticband. For the data set **elastic1**, the values are:

stretch (mm): 46, 54, 48, 50, 44, 42, 52
distance (cm): 183, 217, 189, 208, 178, 150, 249.

For the data se6(e elas)5.4(ticba6.4337 0 TD-0.0001 Tc0 Tw(elastic2)Tj-0.0241 -0.0241 TD(elastic2)Tj0.0241 0 TD(elastic2)Tj-0.

Using a different symbol and/or a different colour, plot the data from the two data frames **elastic1** and **elastic2** on the same graph. Do the two sets of results appear consistent.

2. For each of the data sets **elastic1** and **elastic2**, determine the regression of stretch on distance. In each case determine (i) fitted values and standard errors of fitted values and (ii) the R^2 statistic. Compare the two sets of results. What is the key difference between the two sets of data?

3. Use the method of section 5.7 to determine, formally, whether one needs different regression lines for the two data frames **elastic1** and **elastic2**.

4. Using the data frame **cars** (in the base library), plot **distance** (i.e. stopping distance) versus **speed**. Fit -2-1.18072/ to this relatio.18072ship, and plot the2-1.18072/. The.18072 try fitt1.18072g18072 and plott1.18072g18072 a qu1807

6. Multivariate and Tree-Based Methods

6.1 Multivariate EDA, and Principal Components Analysis

Principal components analysis is often a useful exploratory tool for multivariate data. The idea is to replace the initial set of variables by a small number of "principal components" that together may explain most of the variation in the data. The first principal component is the component (linear combination of the initial variables) that explains the greatest part of the variation. The second principal component is the component that, among linear combinations of the variables that are uncorrelated with the first principal component, explains the greatest part of the remaining variation, and so on.

The measure of variation used is the sum of the variances of variables, perhaps after scaling the variables so that they each have variance one. An analysis that works with the unscaled variables, and hence with the variance-covariance matrix, gives a greater weight to variables that have a large variance. The common alternative – scaling variables so that they each have variance equal to one – is equivalent to working with the correlation matrix.

With biological measurement data, it is usually desirable to begin by taking logarithms. The standard deviations then measure the logarithm of relative change. Because all variables measure much the same quantity (i.e. relative variability), and because the standard deviations are typically fairly comparable, scaling to give equal variances is unnecessary.

The data set **possum** that accompanies these notes has nine morphometric measurements on each of 102 mountain brushtail possums, trapped at seven sites from southern Victoria to central Queensland[33]. It is good practice to begin by examining relevant scatterplot matrices. This may draw atte.9(a)-1ion to gross errors in the data.

are then indicative of different patterns of allometric growth. The reader may wish to repeat the above analysis, but working with the logarithms of measurements.

*7. R Data Structures

7.1 Vectors

Recall that vectors may have mode logical, numeric or character[36].

7.1.1 Subsets of Vectors

Recall (section 2.6.2) two common ways to extract subsets of vectors:

1. Specify the numbers of the elements that are to be extracted. One can use negative numbers to omit elements.
2. Specify a vector of logical values. The elements that are extracted are those for which the logical value is **T**. Thus suppose we want to extract values of **x** that are greater than 10.

The following demonstrates a third possibility, for vectors that have named elements:

```
> c(Andreas=178, John=185, Jeff=183)[c("John","Jeff")]
 John Jeff
  185  183
```

A vector of names has been used to extract the elements.

7.1.2 Patterned Data

Use 5:15 to generate the numbers 5, 6, …, 15. Entering 15:5 will generate the sequence in the reverse order.

To repeat the sequence (2, 3, 5) four times over, enter **rep(c(2,3,5), 4)** thus:

```
> rep(c(2,3,5),4)
 [1] 2 3 5 2 3 5 2 3 5 2 3 5
>
```

If instead one wants four 2s, then four 3s, then four 5s, enter **rep(c(2,3,5), c(4,4,4))**.

```
> rep(c(2,3,5),c(4,4,4))    # An alternative is rep(c(2,3,5), each=4)
 [1] 2 2 2 2 3 3 3 3 5 5 5 5
```

Note further that, in place of **c(4,4,4)** we could write **rep(4,3)**. So a further possibility is that in place of **rep(c(2,3,5), c(4,4,4))** we could enter **rep(c(2,3,5), rep(4,3)).**

In addition to the above, note that the function **rep()** has an argument **length.out**, meaning "keep on repeating the sequence until the length is **length.out**."

7.2 Missing Values

In R, the missing value symbol is **NA**. Any arithmetic operation or relation that involves **NA** generates an **NA**.

This applies also to the relations **<**, **<=**, **>**, **>=**, **==** and **!=**The first four compare magnitudes, **==** tests for equality, and **!=**

```
> is.na(x)  # TRUE for when NA appears, and otherwise FALSE
[1] FALSE FALSE FALSE  TRUE
> x==NA     # All elements are set to NA
[1] NA NA NA NA
> NA==NA
[1] NA
```

WARNING:

7.3.1 Extraction of Component Parts of Data frames

Consider the data frame **`Barley`**. A version is available with the data sets that are supplied to complement these notes. The data set **`immer`** that is bundled with the Venables and Ripley *MASS*

```
primates <- read.table("a:/primates.txt")
```

will create the data frame **primates**, from a file on the **a:** drive. The text strings in the first column will become the first column in the data frame.

Suppose that primates is a data frame with three columns – species name, body weight, and brain weight. You can give the columns names by typing in:

```
names(primates)<-c("Species","Bodywt","Brainwt")
```

Here then are the contents of the data frame.

```
> primates
         Species Bodywt Brainwt
1  Potar monkey   10.0     115
2       Gorilla  207.0     406
3         Human   62.0    1320
4 Rhesus monkey    6.8     179
5         Chimp   52.2     440
```

Specify **header=TRUE** if there is an initial how of header information. If the number of headers is one less than the number of columns of data, then the first column will be used, providing entries are unique, for row labels.

7.4.1 Idiosyncrasies

The function **read.table()** is straightforward for reading in rectangular arrays of data that are entirely numeric. When, as in the above example, one of the columns contains text strings, the column is by default stored as a factor with as many different levels as there are unique text strings[37].

Problems may arise when small mistakes in the data cause R to interpret a column of supposedly numeric data as character strings, which are automatically turned into factors. For example there may be an O (oh) somewhere where there should be a 0 (zero), or an el (**l**) where there should be a one (**1**). If you use any missing value symbols other than the default (**NA**), you need to make this explicit see section 7.3.2 below. Otherwise any appearance of such symbols as *****, period(.) and blank (in a case where the separator is something other than a space) will cause to whole column to be treated as character data.

Users who find this default behaviour of **read.table()** confusing may wish to use the parameter setting **as.is = TRUE.**[38] If the column is later required for use as a factor in a model or graphics formula, it may be necessary to make it into a factor at that time. Some functions do this conversion automatically.

7.4.2 Missing values when using `read.table()`

7.5 Factors and Ordered Factors

We discussed factors in section 2.6.4. They provide an economical way to store vectors of character strings in which there are many multiple occurrences of the same strings. More crucially, they have a central role in the incorporation of qualitative effects into model and graphics formulae.

Factors have a dual identity. They are stored as integer vectors, with each of the values interpreted according to the information that is in the table of levels[40].

The data frame **islandcities** that accompanies these notes holds the populations of the 19 island nation cities with a 1995 urban centre population of 1.4 million or.4 mre. The row na4 ms are the city na4 ms, the first column (**country**) has the na4 me of the country, and the second colu4 mn (

```
Australia Cuba Indonesia Japan Philippines Taiwan United Kingdom
        3    1         4     6           2      1              2
[There are 19 cities in all.]
```

Printing the contents of the column with the name **country** gives the names, not the codes. As in most operations with factors, R does the translation invisibly. There are though annoying exceptions that can make the use of factors tricky. To be sure of getting the country nae ci4.5(30ru a)so betre

7.6 Ordered Factors

Actuallyc their levels that are ordered. To create an ordered factor, or to turn a factor into an ordered factor, use the function **`ordered()`**. The levels of an ordered factor are assumed to specify positions on an ordinal

The second list element is a vector of length 7

```
> options(digits=3)
> elastic.lm$residuals
      1       2       3       4       5       6       7
  2.107  -0.321  18.000   1.893 -27.786  13.321  -7.214
```

The tenth list element documents the function call:

```
> elastic.lm$call
lm(formula = distance ~ stretch, data = elasticband)
> mode(elastic.lm$call)
[1] "call"
```

*7.8 Matrices and Arrays

In these notes the use of matrices and arrays will be quite limited. For almost everything we do here, data frames have more general relevance, and achieve what we require. Matrices are likely to be important for those users who wish to implement new regression and multivariate methods.

All the elements of a matrix have the same mode, i.e. all numeric, or all character. Thus a matrix is a more restricted strsctsre than a data frame. One reason for numeric matrices is that they allow a variety of mathematical operations that are not available for data frames. (A)132(n)7.9(oth)7.9(e)0(r reas)5.5(on)7.9(is)5.5(th)7.9(at)]TJ/T

7.8.1 Arrays

The generalisation from a matrix (2 dimensions) to allow > 2 dimensions gives an array. A matrix is a 2-dimensional array.

Consider a numeric vector of length 24. So that we can easily keep track of the elements, we will make them 1, 2, .., 24. Thus

```
x <- 1:24
```

Then

```
dim(x) <- c(4,6)
```

turns this into a 4 x 6 matrix.

```
> x
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
```

Now try

```
> dim(x) <-c(3,4,2)
> x

, , 1
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

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

7.8.2 Conversion of Numeric Data frames into Matrices

There are various manipulations that are available for matrices, but not for data frames. Use **`as.matrix()`** to handle any conversion that may be necessary.

7.9 Different Types of Attachments

When R starts up, it has a list of directories where it looks, in order, for objeects. You can inspect the curr4(h)6.6t list by typing in **`search()`**. The working directory comes first on the search list.

You can extend the search list in two ways. The **`library()`** command adds libraries. Alternatively, or in addition, the **`attach()`** command places a data frame on the search list. A data frame is in fact a specialised list, with its columns as the objects. Recall the syntax

```
> attach(primates)     # NB: No quotes
> detach(primates)     # NB: S-PLUS requires detach("primates")
```

7.10 Exercises

1. Generate the numbers 101, 102, …, 112, and store the result in the vector **`x`**

3. Generate the sequence consisting of eight 4s, then seven 6s, and finally nine 3s.

4. Create a vector consisting of one 1, then two 2's, three 3's, etc., and ending with nine 9's.

5. Determine, for each of the columns of the data frame **`airquality`** (base library), the median, mean, upper and lower quartiles, and range.
[Specify **`data(airquality)`** to bring the data frame **`airquality`** into the working directory.]

6. For each of the following calculations, decide w0(th)at you would expect, and then check to see if you were right!

a)

```
answer <- c(2, 7, 1, 5, 12, 3, 4)
for (j in 2:length(answer)){ answer[j] <- max(answer[j],answer[j-1])}
```

b)

```
answer <- c(2, 7, 1, 5, 12, 3, 4)
for (j in 2:length(answer)){ answer[j] <- sum(answer[j],answer[j-1])}
```

7. In the built-in data frame **`airquality`** (a) extract the row or rows for which **`Ozone`** has its maximum value; and (b) extract the vector of values of **`Wind`** for values of **`Ozone`** that are above the upper quartile.

8. Refer to the Eurasian snow data that is given in Exercise 1.6 . Find the mean of the snow cover (a) for the odd-numbered years and (b) for the even-numbered years.

9. Determine which columns of the data frame **`Cars93`** (MASS library) are factors. For each of these factor columns, print out the levels vector. Which of these are ordered factors?

10. Use **`summary()`** to get information about data in the data frames **`airquality`**, **`attitude`** (both in the base library), and **`cpus`** (MASS library). Write brief notes, for each of these data sets, on what you have been able to learn.

11. From the data frame **`mtcars`** (MASS library) extract a data frame **`mtcars6`** that holds only the information for cars with 6 cylinders.

12. From the data frame **`Cars93`** (MASS library) extract a data frame which holds only information for small and sporty cars.

13. Store the numbers obtained in exercise 2, in order, in the columns of a 3 x 4 matrix.

14. Store the numbers obtained in exercise 3, in order, in the columns of a 6 by 4 matrix. Extract the matrix consisting of rows 3 to 6 and columns 3 and 4, of this matrix.

81 Useful Functions

811 Confidence Intervals and Tests

Use the help to get complete information. Below, I note two of the simpler functions.

811.1 The t-test and associated confidence interval

Use **`t.test()`**. This allows both a one-sample and a two-sample test.

811.2 Chi-Square tests for two-way tables

Use **`chisq.test()`** for a test for no association between rows and columns in the output from **`table()`**. Alternatively, the argument may be a matrix.

This test that counts enter independently into the cells of a table. For example, the test is invalid if there is c0.0 mtoe ar isd

```
car.brandnames <- substring(Cars93$Make, 1, nblank-1)
> car.brandnames[1:5]
[1] "Acura" "Acura" "Audi"  "Audi"  "BMW"
```

To find the position at w[(-1)2hich the first space appears, w[(-1)2e might do the follow[(-1)2ing:

```
nblank <- sapply(Cars93$Make, function(x){n <- nchar(x);
      a <- substring(x, 1:n, 1:n); m <- match(" ", a,nomatch=1); m})
```

8.4 Application of a Function to the Columns of an Array or Data Frame

```
apply(<array>, <dimension>, <function>)
lapply(<list>, <function>)
               ## N. B. A dataframe is a list.  Output is a list.
sapply(<list>, <function>)
               ## As lapply(), but simplify (e.g. to a vector
               ## or matrix), if possible.
```

8.4.1 apply()

The function apply() can be used on data frames as well as matrices. Here is an example:

```
> apply(airquality,2,mean)   # All elements must be numeric!
  Ozone Solar.R    Wind    Temp   Month     Day
     NA      NA    9.96   77.88    6.99   15.80
> apply(airquality,2,mean,na.rm=T)
  Ozone Solar.R    Wind    Temp   Month     Day
  42.13  185.93    9.96   77.88    6.99   15.80
```

The use of `apply(airquality,1,mean)` will give means for each row. These are not, for these data, useful information!

8.4.2 sapply()

The function `sapply()` can be useful for getting information about the columns of a data frame. Here we use it to count that number of missing values in each column of the built-in data frame `airquality`.

```
> sapply(airquality, function(x)sum(is.na(x)))
  Ozone Solar.R    Wind    Temp   Month     Day
     37       7       0       0       0       0
```

Here are several further examples that use the data frame `moths` that accompanies these notes:

```
> sapply(moths,is.factor)            # Determine which columns are factors
 meters       A       P habitat
  FALSE   FALSE   FALSE    TRUE
> # How many levels does each factor have?
> sapply(moths, function(x)if(!is.factor(x))return(0) else length(levels(x)))
 meters       A       P habitat
      0       0       0       8
```

*8.5 tapply()

The arguments are a variable, a list of factors, and a function that operates on a vector to return a single value. For each combination of factor levels, the function is applied to corresponding values of the variable. The output is an array with as many dimensions as there are factors. Where there are no data values for a particular combination of factor levels, **NA** is returned.

Often one wishes to get back, not an array, but a data frame with one row for each combination of factor levels. For example, we may have a data frame with two factors and a numeric variable, and want to create a new data

frame with all possible combinations of the factors, and the cell means as the response. Here is an example of how to do it.

First, use **tapply()** to produce an array of cell means. The function **dimnames()**, applied to this array, returns a list whose first element holds the row names (i.e. for the level names for the first factor), and whose second element holds the column names. [Further dimensions are possible.] We pass this list (row names, column names) to **expand.grid()**, which returns a data frame with all possible combinations of the factor levels. Finally, stretch the array of means out into a vector, and append this to the data frame. Here is an example using the data set **cabbages** from the MASS library.

```
> data(cabbages)
> names(cabbages)
[1] "Cult"   "Date"   "HeadWt" "VitC"
> sapply(cabbages, levels)
$Cult
[1] "c39" "c52"

$Date
[1] "d16" "d20" "d21"

$HeadWt
NULL
```

If there are no data for some combinations of factor levels, one might want to omit the corresponding rows.

8.6 Splitting Vectors and Data Frames Down into Lists – split()

As an example,

```
split(cabbages$HeadWt, cabbages$Date)
```

returns a list with three elements, the first named "d16" and containing values of **He267** where **Date** has the level **d16**, and similarly for the remaining lists with names "d20" and "d21". You need to use **split()** in this way in order to do side by side boxplots. The function **boxplot()** takes as its first element a list in which the first list element is the vector of values for the first boxplot, the second list element is the vector of values for the second boxplot, and so on.

You can use split to split up a data frame into a list of data frames. For example

```
split(cabbages[,-1], cabbages$Date)  # Split remaining columns
                                     # by levels of Date
```

*8.7 Merging Data Frames

The data frame **Cars93** (*mass* library) holds extensive information on data from 93 cars on sale in the USA in 1993. One of the variables, stored as a factor, is **Type**. I have created a data frame **Cars93.summary**, in which the row names are the distinct values of Type, while a later column holds two character abbreviations of each of the car types, suitable for use in plotting.

```
> Cars93.summary
        Min.passengers Max.passengers No.of.cars abbrev
Compact              4              6         16      C
Large                6              6         11      L
Midsize              4              6         22      M
Small                4              5         21     Sm
Sporty               2              4         14     Sp
Van                  7              8          9      V
```

WeTje5 8 d1(e]TJu]TJs)4.J-to adder colu]TJmn

```
> as.date("31/12/60","dmy")-as.date("1/1/60","dmy")
[1] 365
> as.integer(as.date("1/1/60","dmy"))
[1] 0

> as.integer(as.date("1/1/2000","dmy"))
[1] 14610
> as.integer(as.date("29/2/2000","dmy"))
[1] 14669
> as.integer(as.date("1/3/2000","dmy"))
[1] 14670
```

A wide variety of different formats are possible. Among the legal formats are 8-31-2000 (or 31-8-2000 if you specify **`order="dmy"`**), 8/31/2000 (cf 31/8/2000), or August 31 2000.

Observe that one can subtract two dates and get the time between them in days. There are several functions (including **`date.ddmmmyy()`**) for printing out dates in various different formats.

8.9 Exercises

1. For the data frame **`Cars93`**, get the information provided by **`summary()`** for each level of **`Type`**. (Use **`split()`**.)
2. Determine the number of cars, in the data frame **`Cars93`**

9. Writing Functions and other Code

We have already met several functions. Here is a function to convert Fahrenheit to Celsius:

```
> fahrenheit2celsius <- function(fahrenheit=32:40)(fahrenheit-32)*5/9
> # Now invoke the function
> fahrenheit2celsius(c(40,50,60))
[1]  4.444444 10.000000 15.555556
```

The function returns the value `(fahrenheit-32)*5/9` generally, a function returns the value of the last statement of the function. Unless the result from the function is assigned to a name, th

Here is a function that prints out the mean and standard deviation of a set of numbers:

```
> mean.and.sd <- function(x=1:10){
+ av <- mean(x)
+ sd <- sqrt(var(x))
+ c(mean=av, SD=sd)
+ }
>
> # Now invoke the function
> mean.and.sd()
    mean       SD
5.500000 3.027650

> mean.and.sd(hills$climb)
    mean       SD
1815.314 1619.151
```

9.1 Syntax and Semantics

A function is created using an assignment. On the right hand side, the parameters appear within round brackets. You can, if you wish, give a default. In the example above the default was x = 1:10, so that users can run the function without specifying a parameter, just to see what it does.

Following the closing ")" the function body appears. Except where the function body consists of just one statement, this is enclosed between curly braces ({ }). The return value usually appears on the final line of the function body. In the example above, this was the vector consisting of the two named elements mean and sd.

9.1.1 A Function that gives Data Frame Details

First we will define a function that accepts a vector **x** as its only argument. It will allow us to determine whether

Finally, we may want to do similar calculations on a number of different data frames. So we create a function **check.df()** that encapsulates the calculations. Here is the definition of **check.df()**.

```
check.df <- function(df=moths)
            sapply(df, function(x)if(!is.factor(x))return(0) else
                                                  length(levels(x)))
```

9.1.2 Compare Working Directory Data Sets with a Reference Set

At the beginning of a new session, we might store the names of the objects in the working directory in the vector **dsetnames**, thus:

```
dsetnames <- objects()
```

Now suppose that we have a function **additions()**, defined thus:

```
additions <- function(objnames = dsetnames)
{
        newnames <- objects(pos=1)
        existing <- as.logical(match(newnames, objnames, nomatch = 0))
        newnames[!existing]
}
```

At some later point in the session, we can enter

```
additions(dsetnames)
```

to get the names of objects that have been added since the start of the session.

9.2 Issues for the Writing and Use of Functions

There can be many functions. Choose their names carefully, so that they are meaningful.

Choose meaningful names for arguments, even if this means that they are longer than one would like. Remember that they can be abbreviated in actual use.

As far as possible, make code self-documenting. Use meaningful names for R objects. Ensure that the names used reflect the hierarchies of files, data structures and code.

R allows the use of names for elements of vectors and lists, and for rows and columns of arrays and dataframes. Consider the use of names rather than numbers when you pull out individual elements, columns etc. Thus **dead.tot[,"dead"]** is more meaningful and safer than **dead.tot[,2]**.

Settings that may need to change in later use of the function should appear as default settings for parameters. Use lists, where this seems appropriate, to group together parameters that belong together conceptually.

Where appropriate, provide a demonstration mode for functions. Such a mode will print out summary information on the data and/or on the results of manipulations prior to analysis, with appropriate labelling. The code needed to implement this feature has the side-effect of showing by example what the function does, and may be useful for debugging.

Break functions up into a small number of sub-functions or "primitives". Re-use existing functions wherever possible. Write any new "primitives" so that they can be re-used. This helps ensure that functions contain well-tested and well-understood components. Watch the r-help electronic mail list (section 13.3) for useful functions for routine tasks.

Wherever possible, give parameters sensible defaults. Often a good strategy is to use as defaults parameters that will serve for a demonstration run of the function.

NULL is a useful default where the parameter mostly is not required, but where the parameter if it appears may be any one of several types of data structure. The test **if(!is.null())** then determines whether one needs to investig.2(t)4..4(e n)8..2(t)4..4(e n)8..2(t)4..0241 lrest)1.6(v)98.3(f)8e tasks.

9.3 Functions as aids to Data Management

Where data, labelling etc must be pulled together from a number of sources, and especially where you may want to retrace your steps some months later, take the same care over structuring data as over structuring code. Thus if there is a factorial structure to the data files, choose file names that reflect it. You can then generate the file names automatically, using **`paste()`** to glue the separate portions of the name together.

Lists are a useful mechanism for grouping together all data and labelling information that one may wish to bring together in a single set of computations. Use as the name of the list a unique and meaningful identification code. Consider whether you should include objects as list items, or whether identification by name is preferable. Bear in mind, also, the use of **`switch()`**, with the identification code used to determine what **`switch()`** should pick out, to pull out specific information and data that is required for a particular run.

Concentrate in one function the task of pulling together data and labelling information, perhaps with some subsequent manipulation, from a number of separate files. This structures the code, and makes the function a source of documentation for the data.

Use user-defined data frame attributes to document your data. For example, given the data frame **`elastic`** containing the amount of stretch and resulting distance of movement of a rubber band, one might specify

```
attributes(elasticband)$title <-
```

We pursue the Poisson distribution in an exercise below.

15. Write a function that does an arbitrary number **n** of repeated simulations of the number of accidents in a year, plotting the result in a suitable way. Assume that the number of accidents in a year follows a Poisson distribution. Run the function assuming an average rate of 2.8 accidents per year.

16. Write a function that simulates the repeated calculation of the coefficient of variation (= the ratio of the mean to the standard deviation), for independent random samples from a normal distribution.

17. Write a function that, for any sample, calculates the median of the absolute values of the deviations from the sample median.

*18. Generate random samples from normal, exponential, t (2 d. f.), and t (1 d. f.), thus:

a) **xn<-rnorm(100)**

b) **xe<-rexp(100)**

c) **xt2<-rt(100, df=2)**

d) **xt2<-rt(100, df=1**)

Apply the function from exercise 17 to each sample. Compare with the standard deviation in each case.

*19. The vector **x** consists of the frequencies

5, 3, 1, 4, 6

The first element is the number of occurrences of level 1, the second is the number of occurrences of level 2, and so on. Write a function that takes any such vector x as its input, and outputs the vector of factor levels, here **1 1 1 1 1 2 2 2 3 . . .**

[You'll need the information tha5(t is)6.1provided by cum ism(x). Form a vector in which 1' ishenever the factor level is incremented, and is otherwise zero. . . .]

*20.rite a function tha5(calcu)6.9(lates th)6.9(e m)19.7(i)1.6(n)6.9(i)1.6(m)19.7(u)6.9(m)19.7(o)-5.1(f)8.6(a q)-5.1(u)6.9(ad)-5.1(

*21. A "between time is matrix, ha is calculated from data on heights of trees at time is

84 84 84 84 84 84

*10. GLM, and General Non-linear Models

GLM models are Generalized Linear Models. They extend the multiple regression model. The GAM (Generalized Additive Model) model is a further extension.

10.1 A Taxonomy of Extensions to the Linear Model

R allows a variety of extensions to the multiple linear regression model. In this chapter we describe the alternative functional forms.

The basic model formulation[41] is:

Observed value = Model Prediction + Statistical Error

Often it is assumed that the statistical error values (values of ε in the discussion below) are independently and identically distributed as Normal. Generalized Linear Models, and the other extensions we describe, allow a variety of non-normal distributions. In the discussion of this section, our focus is on the form of the model prediction, and we leave until later sections the discussion of different possibilities for the "error" distribution.

Multiple regression model

$y =$

Additive models are a generalization of **lm** models. In 1 dimension

$$y = \phi_1(x_1) + \varepsilon$$

Some of $z_1 = \phi_1(x_1), z_2 = \phi_2(x_2), \ldots, z_p = \phi_p(x_p)$ may be smoothing functions, while others may be the usual linear model terms. The constant term gets absorbed into one or more of the ϕ s.

Generalized Additive Model

$$y = g(\phi_1(x_1) + \phi_2(x_2) + \ldots. + \phi_p(x_p)) + \varepsilon$$

Generalized Additive Models are a generalisation of Generalized Linear Models. For example, *g*(.) may be the function that undoes the logit transformation, as in a logistic regression model.

Some of $_1 \quad _1(\ _1), \ _2 \quad _2(\ _2), \ldots, \ _p \quad _p(\ _p)$

Table 1: Patients moving (0) and not moving (1), for each of six different alveolar concentrations.

Fig. 25 then displays a plot of these proportions.

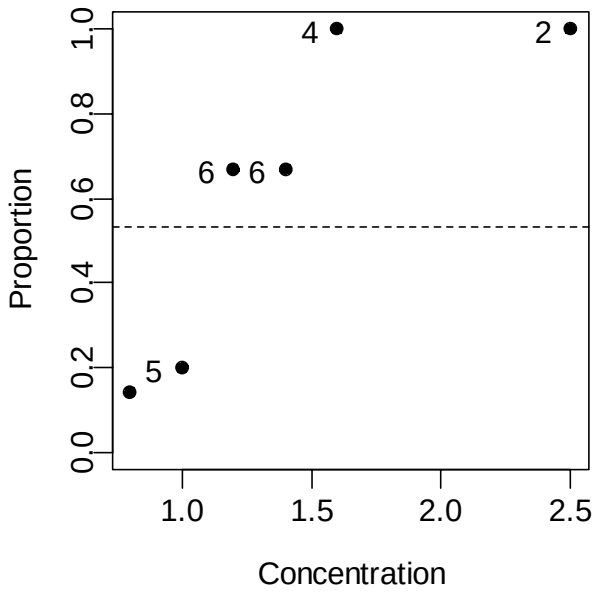


Figure 25: Plot, versus concentration, of proportion of patients not moving. The horizontal line is the estimate of the proportion of moves one would expect if the concentration had no effect.

We fit two models, the logit model and the complementary log-log model. We can fit the models either directly to the 0/1 data, or to the proportions in Table 1. To understand the output, you need to know about "deviances". A deviance has a role ve.9(r)-y simila.9(r)- to a sum of squa.9(r)-es in .9(r)-eg.9(r)-ession. Thus we have:

```
> summary(anaes.logit)

Call: glm(formula = nomove ~ conc, family = binomial(link = logit),
         data = anesthetic)
Deviance Residuals:
   Min     1Q Median    3Q  Max
 -1.77 -0.744 0.0341 0.687 2.07

Coefficients:
            Value Std. Error t value
(Intercept) -6.47       2.42   -2.68
       conc  5.57       2.04    2.72

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 41.5 on 29 degrees of freedom
Residual Deviance: 27.8 on 28 degrees of freedom
Number of Fisher Scoring Iterations: 5

Correlation of Coefficients:
     (Intercept)
conc -0.981
```

Fig. 26 is a graphical summary of the results:

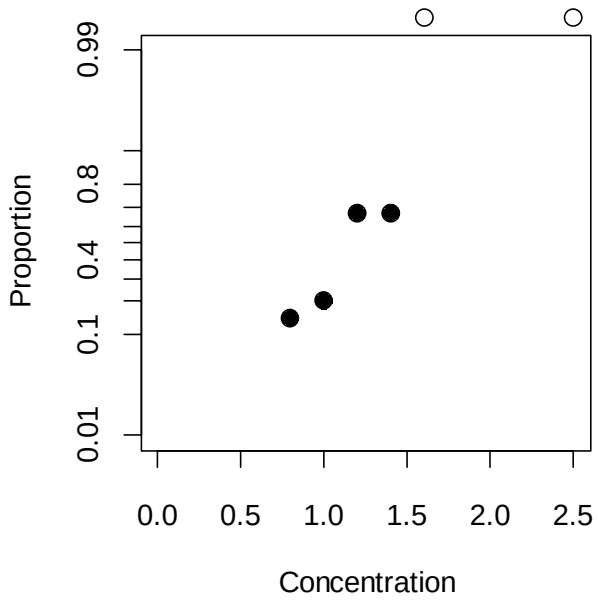


Figure 26: Plot, versus concentration, of log(odds) [= logit(proportion)] of patients not moving. The line is the estimate of the proportion of moves, based on the fitted logit model.

```
methods(summary)
```

to get a list of the summary methods that are available. You 9.8(m)19.8(a)-y wat to 9.8(m)1ix**summary, lm()**-tch, e.g. on an aov or glmect. The output mot be what yu mght expect. So be careful!

10.urther Elaborations

Geeed Lear Model we te 1970s. Teyni huge range o dvers mth odolog. They have no bm7(e)-1(a sto)-5.11ck-in-t o statistical analysts. Their pm7(p)-5.11lm7(en)6.9(tatio)-5.11n bilt o the powpation th, hodel calcu.

Praalysem6(a)-1.11nds frtheors i9.8(m6(p)-5.1me-5m2(2)3(p)-4am)5.3(0(n)57.2(tsu)13.2(eli)7m2(ex)7.2(o=)57.2(s)4.5(f)-1.8(a)7(l

1. Fit a Pisson regn 9.8(m21o)-5.6(d)-5.6(e)-1.51l to tha tham2**moths** thccompanies the notes.llow di ints fr di hitats. Use logmv

10.9 References

Dobson. J. 1983. Antical Modellin. Chapmann

Hastie, T. J. anihi. J. 1990. Genralized Additiv. Chapmann

McCllagh, P.d Nelder, J. nd eeear Modelhapma and Ha

Ypdii-Plpri Yrk

*11. Multi-level Models, Time Series and Survival Analysis

Repeated measures models are a special case of multi-level models.

11.1 Multi-Level Models, Including Repeated Measures Models

Models have both a fixed effects structure and an error structure. For example, in an inter-laboratory comparison there may be variation between laboratories, between observers within laboratories, and between multiple determinations made by the same observer on different samples. If we treat laboratories and observers as random, the only fixed effect is the mean.

The functions **lme()** and **nlme()**, from the Pinheiro and Bates library, handle models in which a repeated measures error structure is superimposed on a linear (**lme**) or non-linear (**nlme**) model. Version 3 of lme is broadly comparable to *Proc Mixed* in the widely used SAS statistical package. The function **lme** has associated with it highly useful abilities for diagnostic checking and for various insightful plots.

There is a strong link between a wide class of repeated measures models and time series models. In the time series context there is usually just one realisation of the series, which may however be observed at a large number of time points. In the repeated measures context there may be a large number of realisations of a series that is typically quite short.

11.1.1 The Kiwifruit Shading Data, Again

Refer back to section 5.8.2 for details of these data. The fixed effects are **block** and treatment (**shade**). The random effects are **block** (though making block a random effect is optional), **plot** within **block**, and units within each block/plot combination. Here is the analysis:

```
> library(nlme)
Loading required package: nls
> kiwishade$plot<-factor(paste(kiwishade$block, kiwishade$shade,
    sep="."))
> kiwishade.lme<-
```

```
             (Intr) shdA2D shdD2F
shadeAug2Dec -0.53
shadeDec2Feb -0.53   0.50
shadeFeb2May -0.53   0.50   0.50

Standardized Within-Group Residuals:
       Min         Q1        Med         Q3        Max
-2.4153887 -0.5981415 -0.0689948  0.7804597  1.5890938

Number of Observations: 48
Number of Groups:
          block plot %in% block
              3              12
> anova(kiwishade.lme)
            numDF denDF  F-value p-value
(Intercept)     1    36 5190.552  <.0001
shade           3     6   22.211  0.0012
```

This was a balanced design, which is why section 5.8.2 could use **aov()** for an analysis. We can get an output summary that is helpful for showing how the error mean squares match up with standard deviation information given above thus:

```
> intervals(kiwishade.lme)
Approximate 95% confidence intervals

 Fixed effects:
                lower       est.      upper
(Intercept)   96.62977 100.202500 103.775232
shadeAug2Dec  -1.53909   3.030833   7.600757
shadeDec2Feb -14.85159 -10.281667  -5.711743
shadeFeb2May -11.99826  -7.428333  -2.858410

 Random Effects:
  Level: block
                   lower     est.   upper
sd((Intercept)) 0.5473014 2.019373 7.45086
  Level: plot
                   lower     est.    upper
sd((Intercept)) 0.3702555 1.478639 5.905037

 Within-group standard error:
   lower     est.    upper
2.770078 3.400378 4.174244
```

The above allows us to put together the information for an analysis of variance table. We have:

	Variance component	Mean square for anova table	d.f.
block	4.076	$12.180 + 4 \times 2.186 + 16 \times 4.076$ = 86.14	2 (3-1)
plot	2.186	$12.180 + 4 \times 2.186$ = 20.92	6 (3-1) ×(2-1)
residual (within group)	12.180	12.18	3×4×(4-1)

Now find see where these same pieces of information appeared in the analysis of variance table of section 5.8.2:

```
> kiwishade.aov<-aov(yield~block+shade+Error(block:shade),data=kiwishade)
> summary(kiwishade.aov)

Error: block:shade
          Df  Sum Sq Mean Sq F value   Pr(>F)
block      2  172.35   86.17  4.1176 0.074879
shade      3 1394.51  464.84 22.2112 0.001194
Residuals  6  125.57   20.93

Error: Within
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 36 438.58   12.18
```

11.1.2 The Tinting of Car Windows

In section 4.1 we encountered data from an experiment that aimed to model the effects of the tinting of car windows on visual performance[44]. The authors are mainly interested in effects on side window vision, and hence in visual recognition tasks that would be performed when looking through side windows.

Data are in the data frame **tinting**. In this data frame, **csoa** (critical stimulus onset asynchrony, i.e. the time in milliseconds required to recognise an alphanumeric target), **it** (inspection time, i.e. the time required for a simple discrimination task) and **age** are variables, while **tint** (3 levels) and **target** (2 levels) are ordered factors. The variable **sex** is coded 1 for males and 2 for females, while the variable **agegp** is coded 1 for young people (all in their early 20s) and 2 for older participants (all in the early 70s).

We have two levels of variation – within individuals (who were each tested on each combination of **tint** and **target**), and between individuals. So we need to specify **id** (identifying the individual) as a random effect.

Finally, there is the very simple model, allowing only for main effects:

```
it1.lme<-lme(log(it)~(tint+target+agegp+sex),
  random=~1|id, data=tinting,method="ML")
```

Note that we have fitted all these models by maximum likelihood. This is so that we can do the equivalent of an analysis of variance comparison. Here is what we get:

```
> anova(itstar.lme,it2.lme,it1.lme)
           Model df       AIC      BIC    logLik   Test  L.Ratio p-value
itstar.lme     1 26  8.146187 91.45036 21.926906
it2.lme        2 17 -3.742883 50.72523 18.871441 1 vs 2  6.11093  0.7288
it1.lme        3  8  1.138171 26.77022  7.430915 2 vs 3 22.88105  0.0065
```

The model that limits attention to first order interactions is adequate. We will need to examine the first order interactions individually. For this we re-fit the model used for **it2.lme**, but now with **method="REML"**.

```
it2.reml<-update(it2.lme,method="REML")
```

We now examine the estimated effects:

```
> options(digits=3)
> summary(it2.reml)$tTable
                     Value Std.Error  DF t-value  p-value
(Intercept)         6.05231    0.7589 145   7.975 4.17e-13
tint.L              0.22658    0.0890 145   2.547 1.19e-02
tint.Q              0.17126    0.0933 145   1.836 6.84e-02
targethicon        -0.24012    0.1010 145  -2.378 1.87e-02
agegp              -1.13449    0.5167  22  -2.196 3.90e-02
sex                -0.74542    0.5167  22  -1.443 1.63e-01
tint.L.targethicon -0.83193    0.0461 145  -1.996 4.78e-02
tint.Q.targethicon -0.80722    0.0482 145  -0.150 8.81e-01
tint.L.agegp       -0.13075    0.0492 145  -2.658 8.74e-03
tint.Q.agegp       -0.86972    0.0520 145  -1.341 1.82e-01
tint.L.sex          0.09794    0.0492 145   1.991 4.83e-02
tint.Q.sex         -0.80542    0.0520 145  -0.104 9.17e-01
targethicon.agegp   0.13887    0.0584 145   2.376 1.88e-02
targethicon.sex    -0.87785    0.0584 145  -1.332 1.85e-01
agegp.sex           0.33164    0.3261  22   1.017 3.20e-01
```

Because **tint** is an ordered factor, polynomial contrasts are used.

11.1.3 The Michelson Speed of Light Data

Here is an example, using the Michelson speed of light data from the Venables and Ripley *MASS* library. The model allows the determination to vary linearly with **Run** (from 1 to 20), with the slope varying randomly between the five experiments of 20 runs each. We assume an autoregressive dependence structure of order 1. We allow the variance to change from one experiment to another. Maximum likelihood tests suggest that one needs at least this complexity in the variance and dependence structure to reprtsent the data accurately. A model that has neither fixed nor random **Run** effects seems all that is justified statistically. To test this, one needs to fit models with and without these effects, setting **method="ML"** in each case, and compare the likelihoods. (I leave this as an exercise!) For purposes of doing this test, a first order autoregressive model would probably be adequate. A model that ignorts the sequential dependence entirely dots give misleading rtsults.

```
> library(mass)  # if needed
> data(michelson)  # if needed
```

There are (at least) two types of method – time domain methods and frequency domain methods. In the time domain models may be conventional “short memory” models where the autocorrelation function decays quite rapidly to zero, or the relatively recently developed “long memory” time series models where the autocorrelation function decays very slowoy as observations move apart in time. A characteristic of “long memory” models is that there is variation at all temporal scales. Thus in a study of wind speeds it may be possible to characterise windy days, windy weeks, windy months, windy years, windy decades, and perhaps even windy centuries. R does not yet have functions for fitting the more recently developed long memory models.

Tw(9stl))-j90.00241 0.00241 D-9stl))-j9

*12. Advanced Programming Topics

12.1. Methods

R is an object-oriented language. Objects may have a "class". For functions such as **`print()`**, **`summary()`**, etc., the class of the object determines what action will be taken. Thus in response to **`print(x)`**, R determines the class attribute of **`x`**, if one exists. If for example the class attribute is "factor", then the function which finally handles the printing is **`print.factor()`**. The function **`print.default()`** is used to print objects that have not been assigned a class.

More generally, the class attribute of an object may be a vector of strings. If there are "ancestor" classes – parent, grandparent, . . ., these are specified in order in subsequent elements of the class vector. For example, ordered factors have the class "ordered", which inherits from the class "factor". Thus:

```
> fac<-ordered(1:3)
> class(fac)
[1] "ordered" "factor"
```

Here **`fac`** has the class "ordered", which inherits from the parent class "factor".

The function **`print.ordered()`**, which is the function that is called when you invoke **`print()`** with an ordered factor, makes use of the fact that "ordered" inherits from "factor".

```
> print.ordered
function (x, quote = FALSE)
{
    if (length(x) <= 0)
        cat("ordered(0)\n")
    else print(levels(x)[x], quote = quote)
    cat("Levels: ", paste(levels(x), collapse = " < "), "\n")
    invisible(x)
}
```

Note that it is a convenience for **`print.ordered()`** to call **`print.factor()`**. The function **`print.glm()`** does not call **`print.lm()`**, even though glm objects inherit from lm objects.

12.2 Extracting Arguments to Functions

How, inside a function, can one extract the value assigned to a parameter when the function was called? Below there is a function **`extract.arg()`**. When it is called as **`extract.arg(a=xx)`**, we want it to return **`"xx"`**. When it is called as **`extract.arg(a=xy)`**, we want it to return **`"xy"`**. Here is how it is done.

```
extract.arg <-
function (a)
{
    s <- substitute(a)
    as.character(s)
}


> extract.arg(a=xy)
[1] "xy"
```

If the argument is a function, we may want to get at the arguments to the function. Here is how one can do it

```
deparse.args <-
function (a)
{
    s <- substitute (a)
```

```
if(mode(s) == "call"){
```

We store the expression in **my.exp2**, and then evaluate it

```
> my.exp2 <-
```

```
xname <- all.vars(expr)
```

13. R Resources

13.1 R Packages for Windows

To get information on R packages (libraries), go to:

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

The Australian link (accessible only to users in Australia) is:

`http://mirror.aarnet.edu.au/pub/CRAN/`

For Windows 95 etc binaries, look in

`http://mirror.aarnet.edu.au/pub/CRAN/windows/windows-9x/`

Look in the directory **contrib** for libraries.

Venables, W.N. and Ripley, B.D. 2000. S Programming. Springer 2000. This is a terse and careful introduction to the dialects of the S language, including R.

R Development Core Team 1999. An Introduction to R.
This document is available from the CRAN sites noted in section 13.1.

13.3 The R-help electronic mail discussion list

Details of the *r-help* list, and of other lists that serve the R community, are available from the web site:

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

13.4 Competing Systems – XLISP-STAT

XLISP-STAT is a lisp-based system that, like S-PLUS and R, allows a seamless extensibility. It is available from

`http://www.stat.umn.edu/~luke/xls/xlsinfo/xlsinfo.html`

See also the code designed to accompany Cook and Woisberg's book "Applied Regression Including Computing and Graphics" (Wiley 1999), available from

`http://www.stat.umn.edu/arc`

14. Appendix 1

14.1 Data Sets Referred to in these Notes

Data sets accompanying these notes

Barley	Cars93.summary	ais	anesthetic	austpop
dewpoint	dolphins	elasticband	florida	hills
huron	islandcities	kiwishade	leafshape	milk
moths	oddbooks	orings	possum	primates

```
sapply(tinting[, 4:6], levels)
sapply(tinting[, 4:6], is.ordered)
```

Section 3.7

1.
```
plot(Animals$body, Animals$brain, pch=1,
    xlab="Body weight (kg)",ylab="Brain weight (g)")
```

2.
```
plot(log(Animals$body),log(Animals$brain),pch=1,
 xlab="Body weight (kg)", ylab="Brain weight (g)", axes=F)
brainaxis <- 10^seq(-1,4)
bodyaxis <-10^seq(-2,4)
axis(1, at=log(bodyaxis), lab=bodyaxis)
axis(2, at=log(brainaxis), lab=brainaxis)
box()
identify(log(Animals$body), log(Animals$brain), labels=row.names(Animals))
```

(See problem 4.)

3. `par(mfrow = c(1,2)), etc.`

Section 7.9

1. `x <- seq(101,112)` or `x <- 101:112`
2. `rep(c(4,6,3),4)`
3. `c(rep(4,8),rep(6,7),rep(3,9))` or `rep(c(4,6,3),c(8,7,9))`
4. `rep(seq(1,9),seq(1,9))` or `rep(1:9, 1:9)` `ity); summary(attitude); summary(cpus)`

Comment on ranges of values, whether distributions seem skew, etc.

11. `mtcars6<x <`