Draft of Changes and Additions in a Projected 3rd Edition of *Data Analysis and Graphics Using R*

J H Maindonald and W J Braun

NB: Section 13.2, on propensity scores, which has been substantially rewritten, is not included here. The draft will be available separately.

February 26, 2009

Languages shape the way we think, and determine what we can think about.
[Benjamin Whorf.]

S has forever altered the way people analyze, visualize, and manipulate data... S is an elegant, widely accepted, and enduring software system, with conceptual integrity, thanks to the insight, taste, and effort of John Chambers.
[From the citation for the 1998 Association for Computing Machinery Software award.]

Contents

1 Graphical User Interfaces to R **5**

1.1 The R Commander Graphical User Interface . . . 5

1.1.1 Installing the R Commander . . . 5

1.1.2 Basics of the R Commander's interface . . . 5

1.2 The rattle GUI . . . 6

1.3 Creating GUIs . . . 7

2 A Straight Line Regression Example **9**

2.1 World record times for track and field events . . . 9

2.1.1 Summary information from model objects . . . 10

2.1.2 The model object . . . 11

3 Multiple Regression **13**

3.1 Northern Irish hill race data – the regression coefficients . . . 13

3.1.1 Times for Northern Irish hill races . . . 13

3.2 Multiple Regression Models – Additional Points . . . 16

3.2.1 Errors in x . . . 16

3.2.2 Implications for variable selection . . . 19

4 Time Series Models **21**

4.1 Time Series – Some Basic Ideas . . . 21

4.1.1 Preliminary graphical explorations . . . 21

4.1.2 The autocorrelation and partial autocorrelation function . . . 22

4.1.3 Autoregressive (AR) models . . . 23

4.1.4 * Autoregressive moving average (ARMA) models – theory . . . 24

4.1.5 Automatic model selection? . . . 26

4.1.6 A time series forecast . . . 26

4.2 Regression modeling with moving average errors . . . 28

5 The R System – Additional Topics **37**

5.1 Data Input and Data Manipulation . . . 37

5.1.1 Accessing Data from the Internet . . . 37

5.1.2 Changing the shape of data frames (or matrices) . . . 37

5.2 Creation of R Packages . . . 38

5.3 Lattice Graphics, and the grid Package . . . 39

5.3.1 Lattice Graphics vs Base Graphics . . . 39

5.3.2 Groups within data, and/or columns in parallel . . . 40

5.3.3 Lattice Parameters and Graphics Features . . . 41

5.3.4 A further example . . . 43

5.3.5 Keys – `auto.key`, `key` & `legend` 43
5.3.6 Panel Functions and Interaction with Plots 44
5.3.7 Interaction with lattice plots – focus, interact, unfocus 46
5.3.8 Multiple lattice graphs on a graphics page 47
5.4 An Implementation of Wilkinson's Grammar of Graphics 48
5.4.1 Australian rain data 48
5.4.2 Physical measurements of Australian athletes 48
5.5 Dynamic Graphics – the rgl and `rggobi` packages 49
5.6 Further Reading 50

1

Graphical User Interfaces to R

The R Commander (*Rcmdr*) will be the main focus of discussion, with brief reference to other GUIs. The final subsection discusses the use of the of the function `gui`, from the *fgui* package, to create simple GUIs.

The R Commander is strongly recommended for use for data input. Especially for novices or infrequent users of R, there can be similar advantages for creating many types of graphs, for statistical testing, for simple tabulation and summarization, and for fitting standard types of model.

Some tasks are best done from the command line, and some from a graphical user interface (GUI), with the balance likely to change in favor of the command line as familiarity with R increases. The two modes of use can be mixed. All the GUIs discussed here make available the commands used by R, for inspection and/or modification and/or for audit trail purposes. The user can examine the help page for the relevant function(s), modify the code as required, and re-execute it.

In addition to the R Commander, note:

- The *rattle* GUI interfaces to a range of multivariate graphics, regression and classification routines.
- The function (*latticist()*) (*latticist* package) should be invoked with a data frame or table as argument. It then opens a GUI interface to the *lattice* package and *vcd* packages, allowing rapid creation of plots that may be useful in their own right, or may be a first step in the creation of more carefully honed plots.
- The *playwith* package has extensive features for interacting with graphs, including the addition of annotation. It includes an interface to *latticist*. The *playwith* package is further described in described further in Subsection 5.3.6.

For *rattle*, both the Gtk+ system and Glade must be installed. Further brief details are in Subsection 1.2. The *playwith* package requires Gtk+.

1.1 The R Commander Graphical User Interface

1.1.1 Installing the R Commander

To install the R Commander from the comamnd line, enter:

```
install.packages("Rcmdr", dependencies=TRUE)
```

Among the dependencies are the graphics packages *rgl* (3D dynamic graphics), *scatterplot3d*, *vcd* (visualization of categorical data) and *colorspace* (for generation of color palettes, etc).

1.1.2 Basics of the R Commander's interface

To start the R commander, start up R and enter:[1]

```
library(Rcmdr)
```

[1] At startup, the R Commander checks whether all the suggested packages, needed to use all its features, are available. If some are missing, then upon starting up, the R commander offers to install them. For installing such packages, there must be a live internet connection.

This opens an R Commander script window, with the output window underneath. This window can be closed by clicking on the × in the top left corner. If thus closed, enter `Commander()` to reopen it again later in the session.

Once the points that will now be noted are understood, use of the R Commander should for the most part be straightforward.

From GUI to writing code: The R commander displays the code that it generates. Users can take this code, modify it, and re-run it. The code can be run either from the R Commander script window or from the R console window (if open).

The active data set: The R commander has the notion of an active data set. Here are alternative ways to make a data set active. Start by clicking on the Data drop-down menu. Then

- Click on Active data set, and pick from among data sets, if any, in the workspace.
- Click on Import data, and follow instructions, to read in data from a file. The data set is read into the workspace, at the same time becoming the active data set.
- Click on New data set . . . , then entering data via a spreadsheet-like interface.
- Click on Data in packages, click on Read Data from Package, then identify one of the attached packages and choose a data set from among those that are included with the package.
- A further possibility is to load data from an R image (**.RData**) file; click on Load data set . . .

Creating graphs: To draw graphs, click on the Graphs drop-down menu. Then

- Click on Scatterplot . . . to obtain a scatterplot. This uses the function `scatterplot()` from the *car* package, which is an option rich interface to functions that are in base graphics.
- Click on X Y conditioning plot . . . for lattice scatterplots and panels of scatterplots.
- Click on 3D graph to obtain a 3D scatterplot, using the R Commander function `scatter3d()` that is an interface to functions in the *rgl* package.

Statistics (& fitting models): Click on the Statistics drop down menu to get submenus that give summary statistics and/or carry out various statistical tests. This includes (under Contingency tables) tables of counts and (under Means) One-way ANOVA. Also, click here to get access to the Fit models submenu.

***Models:** Click here to extract information from model objects once they have been fitted. (NB: To fit a model, go to the Statistics drop down menu, and click on Fit models).

1.2 The *rattle* GUI

A Windows binary that installs both Gtk+ and Glade, as required for *rattle*, is available from `http://downloads.sourceforge.net/gladewin32/`. Download `gtk-dev-2.12.9-win32-1.exe` and click on the executable to start the installation.

To get started, type:

```
library(rattle}
rattle()
```

First click on the radio button that identifies the source of the data – CSV or ARFF (used by some data mining packages); Library (from an R package); an RData file (a **.RData** image file); or ODBC. Click on the relevant button and click on Execute. Selecting CSV and clicking on the Execute tab leads to an offer to load the `audit` sample dataset, which can be used for experimenting with *rattle*'s abilities.

Once loaded, information will be displayed on the data columns, with a tentative specification as Input or Target or Risk or Ident or Ignore. This specification can be changed as required.

Immediately below Execute is a row of tabs: Data (to load data), Explore (for data exploration), ... Model (fit a model), Evaluate (create a new column) and Log (examine code). Thus, to explore the data, click on Explore, click on the type of information required (start maybe with Summary), and click on Execute. To fit a model, click on Model, click maybe on the Forest button, and click on Execute.

1.3 Creating GUIs

The *fgui* package allows the rapid creation of a GUI interface to a user function. Here is a simple example that uses the function `gui()` in this package:

```
library(fgui)
library(DAAGxtras)
## Create function that does the smoothing
fitsmooth <-
  function(x=bomregions$mdbRain, FUN=function(x)x^(1/3), span=2/3){
    if(!is.null(FUN))x <- FUN(x)
    scatter.smooth(x, span=span)
  }
## Create function that sets up a GUI interface to fitsmooth()
callsmooth <- function()gui(fitsmooth,
    argSlider=list(span=c(0.1,1.5,0.025)),  ## start,stop,stepsize
    output=NULL, callback=guiExec, exec="Show graph")
callsmooth()
## Click on "Show graph" to start the graphical display.
## Move the slider to change the width of the smoothing window
```

Note that:

- The first argument to `gui()` is the function `fitsmooth` that is to be executed.
- The argument `argSlider` is used to specify a slider that will control the value of the argument `span` to the function `fitsmooth`.
- The argument `output=NULL` suppresses the text box that would otherwise be created to hold function output. (Here, there is no output to hold.)
- The argument `callback=guiExec` ensures that the function `fitsmooth()` is re-executed whenever one of its arguments is changed, e.g., by moving the slider.
- The argument `exec="Show graph"` is used to label the button that the user should press in order to execute the function.
- Text boxes are automatically created that can be used for entry of the `x` and `FUN` arguments. If these are left blank, the defaults `x=bomregions$mdRain` and `FUN=function(x)x^(1/3)` will be used

2

A Straight Line Regression Example

This worked example is parked here for convenience. It is not intended for inclusion in the 3^{rd} edition.

2.1 World record times for track and field events

Data are world track and road record times, as at 9th August 2006. Data are from the web page `http://www.gbrathletics.com/wrec.htm`. Data are in the dataset `worldRecords` in the `DAAGxtras` package.

Data exploration

First, check what data are in the data frame. The function `str()` is useful for this purpose:

```
> str(worldRecords)
'data.frame':        40 obs. of  5 variables:
 $ Distance   : num  0.1 0.15 0.2 0.3 0.4 0.5 0.6 0.8 1 1.5 ...
 $ roadORtrack: Factor w/ 2 levels "road","track": 2 2 2 2 2 2 2 2 2 2 ...
 $ Place      : chr  "Athens" "Cassino" "Atlanta" "Pretoria" ...
 $ Time       : num  0.163 0.247 0.322 0.514 0.720 ...
 $ Date       :Class 'Date'  num [1:40] 12948  4889  9709 11040 10829 ...
```

A simple form of plot can be obtained thus:

```
library(DAAGxtras)   # Version 0.6-6 or later of DAAGxtras
## Plot with untransformed scales
plot(Time ~ Distance, data=worldRecords)
## Now use log scales
plot(Time ~ Distance, data=worldRecords, log="xy")
```

Distinguishing points for track events from those for road events is easiest if we use lattice graphics. Figure 2.1 is an example: The labeling on the axes is the only difference between the two plots. On the left the labeling is in the original units, but shown as a power of 10. On the right the logarithms of the values are shown. Notice the use of `auto.key` to obtain a key. On a colour device, different colours, rather than different symbols, are used to distinguish the groups.

The code is:

```
## Attach lattice package
library(lattice)
## Left panel
xyplot(Time ~ Distance, groups=roadORtrack, data=worldRecords,
       scales=list(log=10), auto.key=list(columns=2))
## Right panel
xyplot(log(Time) ~ log(Distance), groups=roadORtrack,
       data=worldRecords, auto.key=list(columns=2))
```

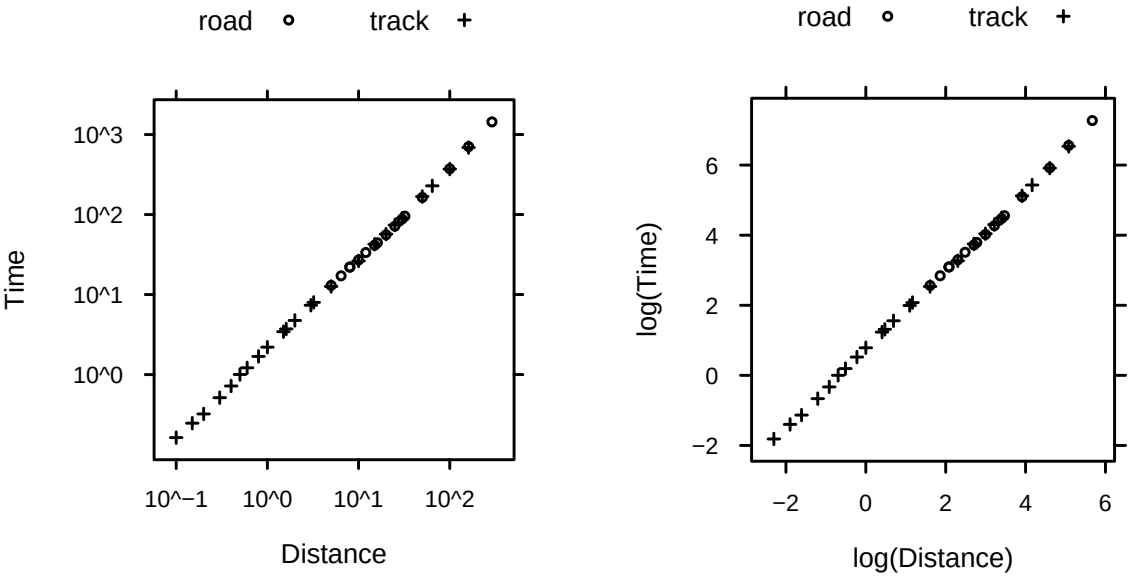

Figure 2.1: World record times versus distance, for field and road events.

Fitting a regression line

The plots made it clear that a line is a good fit, albeit with a small but noticeable lack of fit that for present purposes will be ignored.

The regression will use the function `lm()`. The name is a mnemonic for linear model. The following fits a straight line fit to the logged data:

```
> lm(log(Time) ~ log(Distance), data=worldRecords)

Call:
lm(formula = log(Time) ~ log(Distance), data = worldRecords)

Coefficients:
  (Intercept)  log(Distance)
       0.7316         1.1248
```

There is no difference that can be detected visually between the track races and the road races.

The predicted times from the equation are

$$\begin{aligned}\text{Time} &= e^{0.7316} \times \text{Distance}^{1.1248} \\ &= 2.08 \times \text{Distance}^{1.1248}\end{aligned}$$

This implies, as would be expected, that kilometers per minute increase with increasing distance.

Fitting a line to points that are represented on a log scale thus allows an immediate interpretation, which is a good reason for starting with a line.

2.1.1 Summary information from model objects

In order to avoid recalculation of the model information each time that some different information is required, we store the result from the `lm()` calculation in the model object `worldrec.lm`. The choice of name for the model object is arbitrary; any convenient name will do.

```
## Store the result in worldrec.lm
worldrec.lm <- lm(log(Time) ~ log(Distance), worldRecords)
```

Three functions that are commonly used to get information about model objects are: `print()`, `summary()` and `plot()`. These are all *generic* functions. The effect of the function depends on the class of object that is printed (ie, by default, displayed on the screen) or or plotted, or summarized.

Additionally, the function `abline()` can be used with the model object as argument to add a line to the plot of `log(Time)` against `log(Distance)`, thus:

```
plot(log(Time) ~ log(Distance), data = worldRecords)
abline(worldrec.lm)
```

The function `print()` typically displays relatively terse output, while `summary()` may display more extensive output. This varies from one type of model object to another.

When used with `lm` objects, `print()` calls `print.lm()`, while `summary()` calls `summary.lm()`.[1]

Compare the outputs from the following:

```
print(worldrec.lm)     # Equivalent to typing wtvol.lm at the command line
summary(worldrec.lm)
```

Diagnostic plots

Insight into the adequacy of the line can be obtained by examining the "diagnostic" plots, obtained by "plotting" the model object.

```
par(mfrow=c(1,2))  # Subsequent plots appear in a 1 x 2 layout
plot(worldrec.lm, which=1:2)
par(mfrow=c(1,1))  # Reset to 1 plot per page, for any later plots
```

Use of a 1 by 2 layout is convenient because the two plots can then be studied side by side.

By default, there are four "diagnostic" plots. Examination of the first two are adequate for present purposes. The first plot can be used to get an indication of whether the relationship really is linear, or whether there is some further systematic component that should perhaps be modeled. The second plot allows an assessment of whether residuals follow an approximate normal distribution. If the distribution is normal, points should lie, approximately, on a line.

The first plot does suggest a small systematic difference from a line. There are mechanisms for using a smooth curve to account for this small difference from a line, if it is thought important enough to model.

2.1.2 The model object

The model object is actually a list. Here are the names of the list elements:

```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

These different list elements hold all very different classes and dimensions (or lengths) of object. Hence the use of a list; any collection of different R objects can be brought together into a list.

The following is a check on the model call:

```
> worldrec.lm$call
lm(formula = log(Time) ~ log(Distance), data = worldRecords)
```

Commonly required information can be accessed using generic extractor functions. Above, attention was drawn to `print()`, `summary()` and `plot()`. Other commonly used extractor functions are residuals(), `coefficients()`, and `fitted.values()`. These can be abbreviated to `resid()`, `coef()`, and

[1] More precisely, `summary(wtvol.lm)` calls `UseMethod("summary")`, which notes that `wtvol.lm` is an lm object and therefore calls `summary.lm()`.

`fitted()`. Where one of these functions gives the information that is required, this is preferable to accessing list elements directly.

The following demonstrates use of the extractor function `coef()`:

```
> coef(worldrec.lm)
  (Intercept) log(Distance)
    0.7316019     1.1247534
```

3

Multiple Regression

3.1 Northern Irish hill race data – the regression coefficients

If an aim is a scientific understanding that involves interpretation of model coefficients, then it is important to fit a model whose coefficients are open to the relevant interpretations. Different formulations of the regression model, or different models, may serve different explanatory purposes. Predictive accuracy is in any case a consideration, and is often the main interest.

Three examples will demonstrate issues for the interpretation of model coefficients. The first is a data set on record times, distances and amonts of climb for Northern Irish hill races. The second has data on mouse brain weight, litter size and body weight. The third has data on book dimensions and weight, from a highly biased sample of books.

3.1.1 Times for Northern Irish hill races

The data set `nihills`, from which Table 3.1 has selected observations, gives distances (`dist`), heights climbed (`climb`), male record times (`time`) and female record times (`timef`), for 23 Scottish hill races.

We begin with scatterplot matrices, both for the untransformed data (Figure 3.1A), and for for the log transformed data (Figure 3.1B).) Attention is limited to the male results.[1]The diagonal panels give the x-variable names for all plots in the column above or below, and the y-variable names for all plots in the row to the left or right. Note that the vertical axis labels alternate between the axis on the extreme left and the axis on the extreme right, and similarly for the horizontal axis labels. This avoids a proliferation of axis labels on the extreme left and lower axes.

Apart from a possible outlier, the relationship between `dist` and `climb` seems approximately linear. The same is true when logarithmic scales are used, and the outlier is now much less evident. Linear relationships between explanatory are helpful, in part, because they make the interpretation of diagnostic and other plots

[1]
```
## Scatterplot matrix: data frame nihills (DAAGxtras)
## Panel A: untransformed data
library(lattice)
splom(~ nihills[, c("dist","climb","time")], cex.labels=1.2,
      varnames=c("dist\n(miles)","climb\n(feet)", "time\n(hours)"))
## Panel B: log transformed data
splom(~ log(nihills[, c("dist","climb","time")]), cex.labels=1.2,
      varnames=c("dist\n(log miles)", "climb\n(log feet)", "time\n(log hours)"))
```

Table 3.1: *Distance* (`dist`)*, height climbed* (`climb`)*, and record times* (`time`)*, for four of the 23 Northern Irish hill races.*

	Name	`dist` (mi)	`climb` (ft)	`time` (h)	`timef` (h)
1	Binevenagh	7.5	1740	0.86	1.06
2	Slieve Gullion	4.2	1110	0.47	0.62
3	Glenariff Mountain	5.9	1210	0.70	0.89
...	...	...	...	...	...
23	BARF Turkey Trot	5.7	1430	0.71	0.94

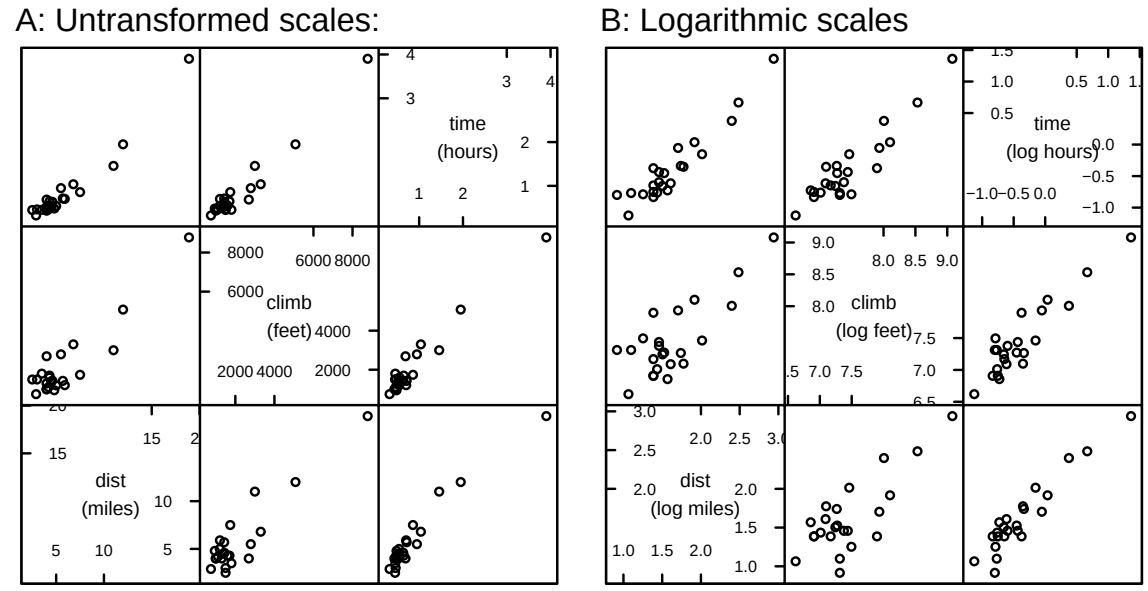

Figure 3.1: Scatterplot matrix for the `nihills` data (Table 3.1). Panel A uses the untransformed scales, while panel B uses logarithmic scales.

straightforward.

The following are reasons for investigating the taking of logarithms:

- The range of values of `time` is large (3.9:0.32, i.e., >10:1), and similarly for `dist` and `climb`. The times are bunched up towards zero, with a long tail. In such instances, use of a logarithmic transformation is likely to lead to a more symmetric distribution.
- One point in particular has a time that is more than twice that of the next largest `time`, as is evident in Figure 3.1A. The values of `dist` and `climb` similarly stand out as much larger than for other points. In a regression that uses the untransformed variables, this point will have a much greater say in determining the regression equation than any other point. It has large *leverage*. Even after taking logarithms (Figure 3.1B), its leverage remains large, but not quite so dominating.
- It can be expected that `time` will increase more than linearly at very long times, and similarly for `climb`, as physiological demands on the human athlete move closer to limits of human endurance.
- The relationship between the explanatory variables (*dist* and *climb*) is more nearly linear on the logarithmic scale of Figure 3.1B.

Additionally, use of a logarithmic scale may help stabilize the variance.

These considerations suggest fitting the equation

$$\log(\texttt{time}) = a + b_1 \log(\texttt{dist}) + b_2 \log(\texttt{climb}),$$

This is equivalent to the power relationship

$$\texttt{time} = A(\texttt{dist})^{b_1}(\texttt{climb})^{b_2},$$

where $a = \log(A)$.

Now fit the model and examine the diagnostic plots (shown in 3.2):

```
nihills.lm <- lm(log(time) ~ log(dist) + log(climb), data = nihills)
plot(nihills.lm)
```

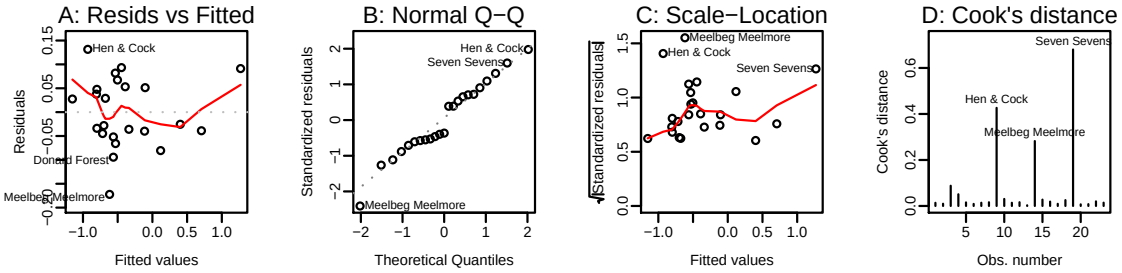

Figure 3.2: Diagnostic plots for the regression of `log(time)` on `log(dist)` and `log(climb)`.

The diagnostic plots do not indicate serious problems, apart from the moderately large residual associated with the Meelbeg Meelmore race.

The estimates of the coefficients (a, b_1 and b_2) are:

```
> summary(nihills.lm)$coef
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept) -4.9611313 0.27387193 -18.11479 7.085048e-14
log(dist)    0.6813596 0.05517831  12.34832 8.186381e-11
log(climb)   0.4657575 0.04530181  10.28121 1.980592e-09
```

Interpreting the coefficients

The estimated equation is

$$\texttt{log(time)} = \underset{\text{[SE=0.27]}}{-4.96} + \underset{\text{[SE=0.055]}}{0.68} \times \texttt{log(dist)} + \underset{\text{[SE=0.045]}}{0.47} \times \texttt{log(climb)}$$

Exponentiating both sides of this equation, and noting $\exp(-4.96) = 0.0070$, gives

$$\texttt{time} = 0.00070 \times \texttt{dist}^{0.68} \times \texttt{climb}^{0.0.47}.$$

This equation implies that for a given height of climb, the time taken is smaller for the second three miles than for the first three miles. (This follows because $0.68 < 1$.) Is this plausible? An answer requires an examination of the implications of holding `climb` constant. For a given value of `climb`, short races will be steep while for long races the slope will be relatively gentle. Thus a coefficient that is less than 1.0 is unsurprising.

A meaningful coefficient for `logdist`

The coefficient for `logdist` will be more meaningful if we regress on `logdist` and `log(climb/dist)`. Then we find:

```
> lognihills$logGrad <- with(nihills, log(climb/dist))
> nihillsG.lm <- lm(logtime ~ logdist + logGrad, data=lognihills)
> summary(nihillsG.lm)$coef
            Estimate Std. Error t value Pr(>|t|)
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -4.961     0.2739   -18.1 7.09e-14
logdist        1.147     0.0346    33.2 5.90e-19
logGrad        0.466     0.0453    10.3 1.98e-09
```

The coefficient of `logdist` is now, reassuringly, greater than 1. The coefficient of `logdist` depends, critically, on what the other explanatory variables are used!

There is another, related, benefit. The correlation between `logdist` and `logGradient`, is -0.065, negligible relative to the correlation of 0.78 between `logdist` `logclimb`.[2]Because the correlation between `logdist` and `logGradient` is so small, the coefficient of `logdist` (=1.124) in the regression on `logdist` alone is almost identical to the coefficient of `logdist` (=1.147) in the regression on `logdist` and `logGradient`.

The standard error of the coefficient of `logdist` is smaller – 0.035 as against 0.055 when the second explanatory variable is `logGradient` rather than `logclimb`. Note that the predicted values do not change. The models `nihills.lm` `nihillsG.lm` are different mathematical formulations of the same underlying model.

3.2 Multiple Regression Models – Additional Points

The following notes should dispel any notion that this chapter's account of multiple regression models has covered everything of importance. They draw attention to issues of large practical importance. These have received extensive attention in a literature where, without good guideposts, it is not easy to tease out practical implications.

3.2.1 Errors in x

In the discussion so far, it has been assumed, either that the explanatory variables are measured with negligible error or that the interest is in the regression relationship given the observed values of explanatory variables.

This subsection is designed to draw attention to the effect that errors in the explanatory variables can have on regression slope. Discussion will be limited to a relatively simple "classical" errors in x model. For this model the error in x, if large, reduces the chances that the estimated slope will appear statistically significant. Additionally, it reduces the expected magnitude of the slope, i.e., the slope is attenuated. Even with just one explanatory variable x, it is not possible to estimate the magnitude of the error or consequent attenuation from the information shown in a scatterplot of y versus x. For estimating the magnitude of the error, there must be a direct comparison with values that are measured with negligible error.

The discussion will now turn to as study on the measurement of dietary intake. The error in the explanatory variable, as commonly measured, turned out to be larger and of greater consequence than most researchers had been willing to contemplate.

Measurement of dietary intake

The 36-page Diet History Questionnaire is a Food Frequency Questionnaire (FFQ) that was developed and evaluated at the U.S. National Cancer Institute. In large-scale trials that look for dietary effects on cancer and on other diseases, it has been important to have an instrument for measuring food intake that is relatively cheap and convenient. (There have been trials costing US$100,000,000 or more.)

The FFQ asks for details of food intake over the previous year for 124 food items. It queries frequency of intake and, for most items, portion sizes. Supplementary questions query such matters as seasonal intake and food type. More detailed food records may be collected at specific times, which can then be used to calibrate the FFQ results. One such instrument is a 24-hour dietary recall, which questions participants on their dietary intake in the previous 24 hours.

The study that is reported in Schatzkin et al. (2003) compared results from the FFQ with those from Doubly Labeled Water, used as an accurate but highly expensive biomarker. Schatzkin et al. (2003) conclude that the FFQ is too inaccurate for its intended purpose. The accuracy of the 24-hour dietary recall was a further concern.

Schatzkin et al. (2003) reported measurement errors where the standard deviation for estimated energy intake was seven times the standard deviation, between different individuals, of the reference. Additionally, Schatzkin

[2]
```
with(lognihills, cor(logdist, logGrad))
 with(lognihills, cor(logdist, logclimb))
```

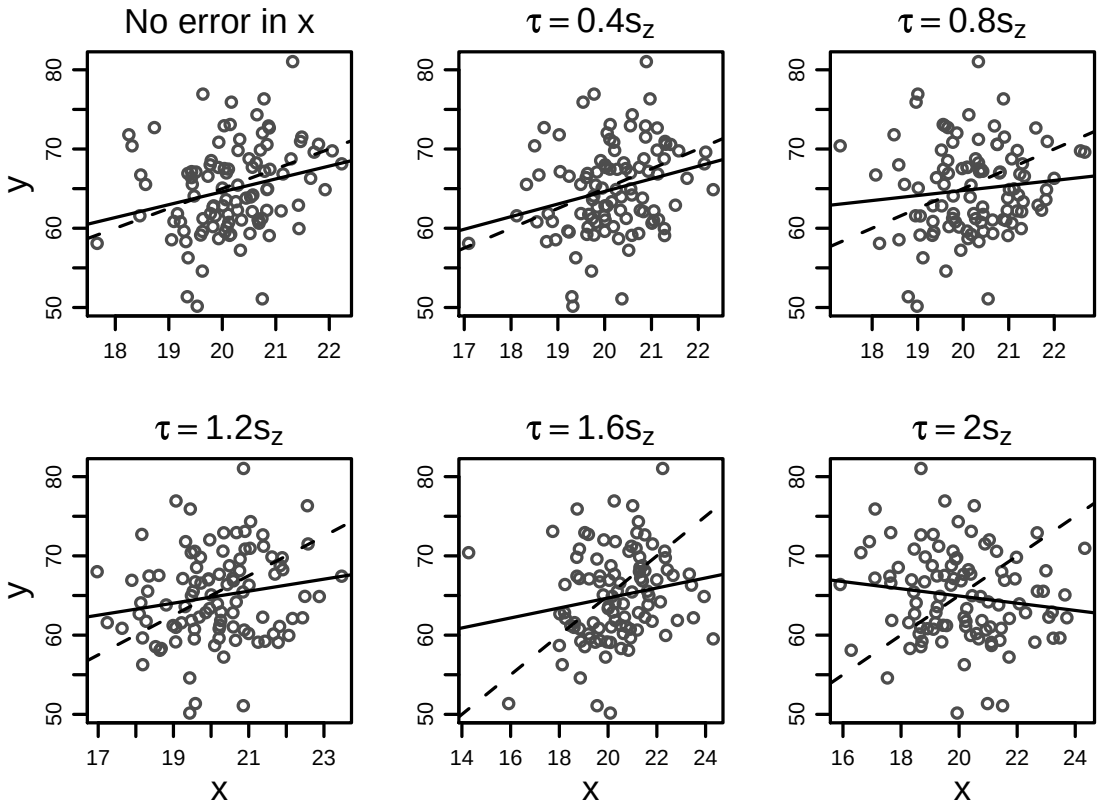

Figure 3.3: The fitted solid lines show how the regression line for y on x change as the error in x changes. The underlying relationship, shown with the dashed line, is in each instance $y = 15 + 2.5\ z$. Note that $s_z^2 = \sum_{i=1}^{n}(z_i - \bar{z})^2$, and that τ is the standard deviation of the independent errors that are added to the z_i.

et al. (2003) found a bias in the relationship between FFQ and reference that further reduced the attenuation factor, to 0.04 for women and to 0.08 for men. For the relationship between the 24 hour recalls and the reference, the attenuation factors were 0.1 for women and 0.18 for men, though these can be improved by use of repeated 24-hour recalls. These results raise serious questions about what such studies can achieve, using presently available instruments that are sufficiently cheap and convenient that they can be used in large studies. Carroll (2004) gives an accessible summary of the issues.

A simulation of the effect of measurement error

Suppose that the underlying regression relationship that is of interest is

$$y_i = \alpha + \beta z_i + \epsilon_i \ (i = 1, \ldots, n)$$

and that the measured values are

$$x_i = z_i + \eta_i$$

where

$$\mathrm{var}[\epsilon_\mathrm{i}] = \sigma^2;\ \mathrm{var}[\eta_\mathrm{i}] = \tau^2$$

Figure 3.3 shows the effect. If τ is 40% of the standard deviation in the x direction, i.e., $\tau = 0.4s_z$, the effect on the slope is modest. If $\tau = 2s_z$, the attenuation is severe.

The expected value of the attenuation in the slope is, to a close approximation

$$\lambda = \frac{1}{1 + \tau^2/s_z^2}$$

where $s_z = \sqrt{\sum_{i=1}^{n}(z_i - \bar{z})^2}$. If $\tau = 0.4 s_z$, then $\lambda \simeq 0.86$.

Whether a reduction in slope by a factor of 0.86 is of consequence will depend on the nature of the application. Often there will be more important concerns. Very small attenuation factors (large attenuations), e.g. less than 0.1 such as were found in the Schatzkin et al. (2003) study, are likely to have serious consequences for the use of analysis results.

Points to note are:

- From the data used in the panels of Figure 3.3, it is impossible to estimate τ, or to know the underlying z_i values. This can be determined only from an investigation that compares the x_i with an accurate, i.e., for all practical purposes error-free, determination of the z_i. The Schatzkin et al. (2003) study that will be discussed below made use of a highly expensive reference instrument, too expensive for standard use, to assess and calibrate the widely used cheaper measuring instruments.
- A test for $\beta = 0$ can be undertaken in the usual way, but with reduced power to detect an effect that may be of interest.
- The t-statistic for testing $\beta = 0$ is affected in two ways; the numerator is reduced by an expected factor of λ, while the standard error that appears in the numerator increases. Thus if λ = 0.1, the sample size required to detect a non-zero slope is inflated by more than the factor of 100 that the effect on the slope alone would suggest.

In social science, the ratio τ^2/s_z^2 has the name *reliability*. As Fuller (1987) points out, a better name is reliability ratio.

*Errors in variables – multiple regression**

Attention will be limited to the case where one predictor is measured with error, and others without error. Where more than one predictor is measured with error, the possibilities for spurious effects can only widen.

The coefficient of the variable that is measured with error is attenuated, as in the single variable case. The coefficients of other variables may be reversed in sign, or show an effect when there is none. See Carroll (2004, pp. 52-55) for summary comment.

Suppose that

$$y = \beta_x \mathbf{x} + \beta_z z + \epsilon$$

If w is unbiased for x and the measurement error u is independent of x and z, then least squares regression yields a consistent estimate of $\lambda\beta_x$

$$\lambda = \frac{\sigma^2_{x|z}}{\sigma^2_{x|z} + \sigma^2_u}$$

The σ_x^2 that appears in the single predictor case is replaced by $\sigma^2_{x|z}$.

A new feature is the bias in the least squares estimate of β_z. The naive least squares estimator estimates

$$\beta_z + \beta_x(1 - \lambda)\gamma_{x|z} \tag{3.1}$$

where $\gamma_{x|z}$ is the coefficient of z in the least squares regression of x on z. The estimate of β_z may be non-zero, even though $\beta_z = 0$. Where $\beta_z \neq 0$, the least squares estimate can be reversed in sign from β_z. The bias depends on the relative values of β_z, β_x and λ.

Where there are multiple explanatory variables that are measured without error, equation 3.1 can be applied to each of them in turn.

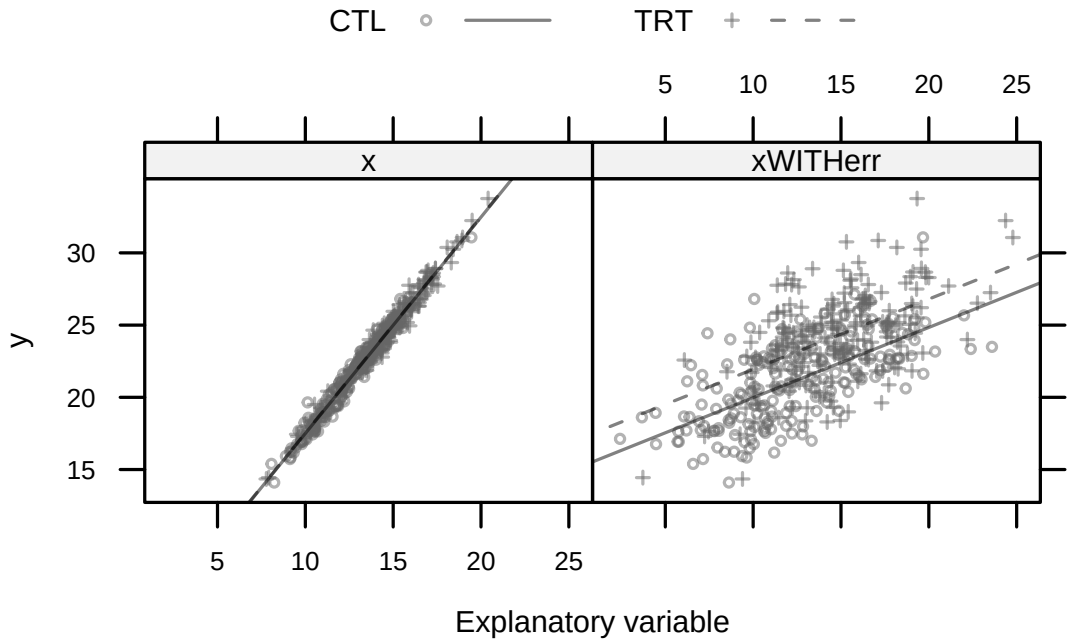

Figure 3.4: Here, y is a linear function of x. The mean value of x is 2.5 for the first level of a, and 4.25 for the second level of a. In the panel on the left, the values of x are measured without error. In the panel on the right, independent errors have been added to x from a distribution with an SD that is 1.5 times that of the residual error SD in the the regression in the left panel. The lines for the two levels of a now separate.

One predictor measured without error – a simulation

The function `errorsINxGP()`, available from `http://www.maths.anu.edu.au/~johnm/r/functions/`, simulates the effect when the variable that is measured without error code for a categorical effect. Two lines appear, where there should be one line only.

3.2.2 Implications for variable selection

The implications are clearly damaging. If one predictor only is measured with substantial error has a non-zero effect, then any variable that

- has a non-zero correlation with that predictor, and
- has no effect on the response

will have a non-zero expected least squares coefficient.

Where two or more variables are measured with substantial error, effects on other least squares coefficients may in fortuitous circumstances cancel. More important, in most practical circumstances, is a widening of the range of possibilities for obtaining least squares coefficients that are spuriously non-zero.

4

Time Series Models

Sections 9.1 and 9.2 of *Data Analysis and Graphics Using R* have been substantuallly rewritten. Drafts of the rewrite are reproduced here.

4.1 Time Series – Some Basic Ideas

The time series object `LakeHuron` (*datasets*) has annual depth measurements at a specific site on Lake Huron. The discussion of sequential dependence, and of the use of ARMA type models of this dependence, will use these data for illustrative purposes.

4.1.1 Preliminary graphical explorations

Figure 4.1 is a trace plot, i.e. an unsmoothed plot of the data against time. The code is:

```
## Plot depth measurements: ts object LakeHuron (datasets)
plot(LakeHuron, ylab="depth (in feet)", xlab = "Time (in years)")
```

There is a slight downward trend for the first half of the series. Observe also that, with some notable exceptions, depth measurements that are close together in time are often close in value. This suggests that these are not independent observations. Time series typically exhibit some form of sequential dependence, in which observations that are close in time are more similar than those that are widely separated. A key challenge for time series analysis is to find good ways to model this dependence.

Lag plots may give a useful visual impression of the dependence. Suppose that our observations are $x_1, x_2, \ldots, x_n$. Then the lag 1 plot plots x_i against x_{i-1} $(i = 2, \ldots, n)$, thus:

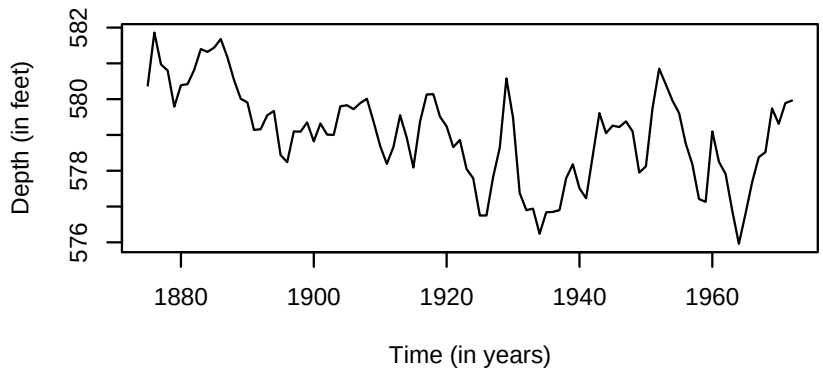

Figure 4.1: A trace plot of annual depth measurements of Lake Huron versus time.

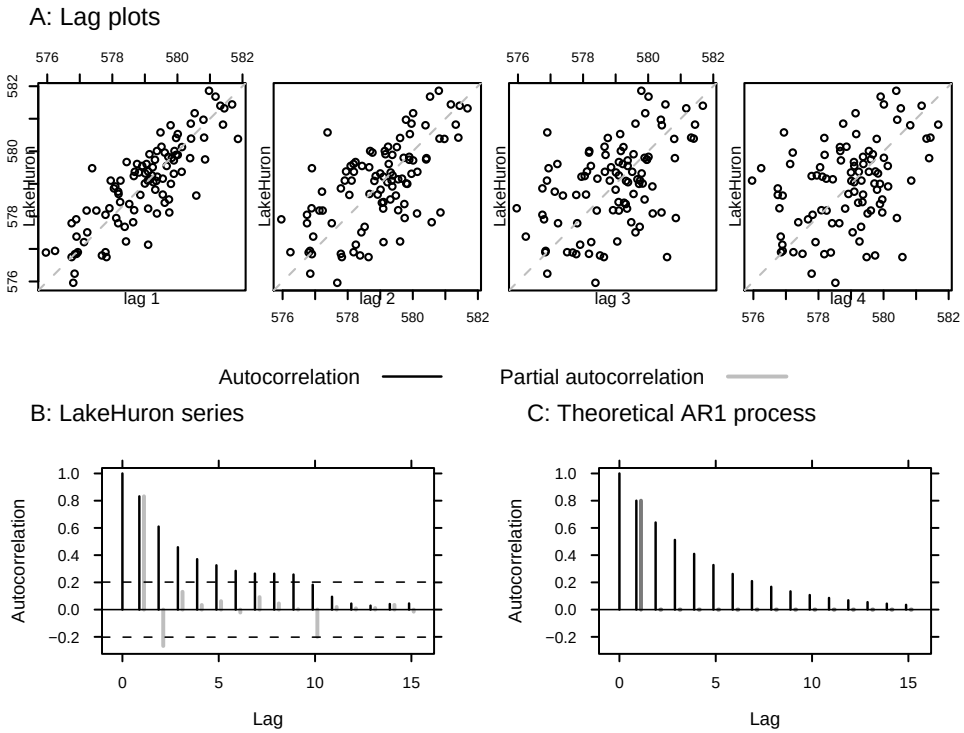

Figure 4.2: The top four panels (A) show the first four lag plots of the annual depth measurements of Lake Huron. Panel B shows the estimated autocorrelations, with the partial autocorrelations shown in gray. The dashed horizontal lines are approximate pointwise 95% confidence bounds for the autocorrelations for a pure noise process, i.e., independent normal data with mean 0. Panel C shows the theoretical autocorrelations for an AR1 process with `ar1 = 0.8`. Again the partial autocorrelation, with a single spike at lag 1, is shown in gray.

y-value	x_2	x_3	...	x_n
lag 1 (x-axis)	x_1	x_2	...	x_{n-1}

For a lag 2 plot, x_i is plotted against x_{i-2} ($i = 3, \ldots, n$), and so on for higher lags. Notice that the number of points that are plotted is reduced by the number of lags.

Figure 4.2A shows the first four lag plots for the Lake Huron data. The code is:

```
lag.plot(LakeHuron, lags=3, do.lines=FALSE)
```

The scatter of points about a straight line in the first lag plot suggests that the dependence between consecutive observations is linear. The increasing scatter observed in the second, third and fourth lag plots indicates that points separated by two or three lags are successively less dependent. Note that the slopes, and therefore the correlations, are all positive.

4.1.2 The autocorrelation and partial autocorrelation function

For each of the lag plots in Figure 4.2A, there is a correlation. Informally, such correlations are *autocorrelations* (literally, self-correlations). The autocorrelation function (ACF), which gives the autocorrelations at all possible lags, often gives useful insight. Figure 4.2B plots the first 19 lag autocorrelations. The code is:

```
acf(LakeHuron)
## pacf(LakeHuron) gives the plot of partial autocorelations
```

The autocorrelation at lag 0 is included by default; this always takes the value 1, since it represents the correlation between the data and themselves. Of more interest are the autocorrelations at other lags. In particular, as inferred from the lag plots, the autocorrelation at lag 1 (sometimes called the serial correlation) is fairly large. Later autocorrelations (lags 2, 3, ...) are successively smaller. There is no obvious linear association among observations separated by lags of more than about 10.

The *partial autocorrelation* at a particular lag measures the strength of linear correlation between observations separated by that lag, after adjusting for correlations between observations separated by fewer lags. Figure 4.2B indicates partial autocorrelations that may be greater than statistical error at lags 1, 2 and 10. Other autocorrelations are mostly a flow-on effect from the large autocorrelations at lag 1. Figure 4.2C shows the autocorrelation function and partial autocorrelation function for a very simple theoretical model that accounts for most of the correlation structure in Figure 4.2B. Notice however the very clear differences, especially evident in the comparison between the partial autocorrelation plots in Figures 4.2B and 4.2C.

As discussed in the next subsection, autocorrelation in a time series complicates the estimation of such quantities as the standard error of the mean. There must be appropriate modeling of the dependence structure. If there are extensive data, it helps to group the data into sets of k successive values, and take averages. The serial correlation then reduces to ρ_1/k approximately. See Miller (1986) for further comment.

4.1.3 Autoregressive (AR) models

We have indicated earlier that, whenever possible, the rich regression framework provides additional insights, beyond those available from the study of correlation structure. This is true also for time series. Autoregressive models move beyond attention to correlation structure to the modeling of regressions relating successive observations.

The AR(1) *model*

Figure 4.2C showed the theoretical autocorrelation and partial autocorrelation plots for an AR1 process. The autoregressive model of order 1 (AR(1)) for a time series $X_1, X_2, \ldots$, has the recursive formula

$$X_t = \mu + \alpha(X_{t-1} - \mu) + \varepsilon_t, \quad t = 1, 2, \ldots,$$

where μ and α are parameters. Figure 4.2C had $\alpha = 0.8$.

Usually, α takes values in the interval $(-1, 1)$; this is the so-called *stationary* case. Non-stationarity, in the sense used here, implies that the series is evolving with time. The mean may be changing with time, and/or the variances and covariances may depend on the time lag. Any such time dependence must be removed or modeled.

For series of positive values, a logarithmic transformation will sometimes bring the series closer to stationarity and/or make it simpler to model any trend. Discussion of standard ways to handle trends will be deferred to Subsection 4.1.4.

The error term ε_t is the familiar independent noise term with constant variance σ^2. The distribution of X_0 is assumed fixed and will not be of immediate concern.

For the AR(1) model, the ACF at lag i is α^i, for $i = 1, 2, \ldots$. If $\alpha = .8$, then the observed autocorrelations should be .8, .64, .512, .410, .328, ..., a geometrically decaying pattern, as in Figure 4.2C and not too unlike that in Figure 4.2B.

To gain some appreciation for the importance of models like the AR(1) model, we consider estimation of the standard error for the estimate of the mean μ. Under the AR(1) model, a large sample approximation to the standard error for the mean is

$$\frac{\sigma}{\sqrt{n}} \frac{1}{(1-\alpha)}.$$

For a sample of size 100 from an AR(1) model with $\sigma = 1$ and $\alpha = 0.5$, the standard error of the mean is 0.2. This is exactly twice the value that results from the use of the usual $\sigma/\sqrt{n}$ formula. Use of the usual standard

error formula will result in misleading and biased confidence intervals for time series where there is substantial autocorrelation.

There are several alternative methods for estimating for the parameter α. The *method of moments* estimator uses the autocorrelation at lag 1, here equal to .8319. The maximum likelihood estimator, equal to .8376, is an alternative.[1]

The general AR(p) *model*

It is possible to include regression terms for X_t against observations at greater lags than one. The autoregressive model of order p (the AR(p) model regresses X_t against $X_{t-1}, X_{t-2}, \ldots, X_{t-p}$:

$$X_t = \mu + \alpha_1(X_{t-1} - \mu) + \cdots + \alpha_p(X_{t-p} - \mu) + \varepsilon_t, \quad t = 1, 2, \ldots,$$

where $\alpha_1, \alpha_2, \ldots, \alpha_p$ are additional parameters that would need to be estimated. The parameter α_i is the partial autocorrelation at lag i.

Assuming an AR process, how large p should be, i.e., how many AR parameters are required? The function `ar()`, in the *stats* package, can be used to estimate the AR order. This uses the Akaike Information Criterion (AIC), which was introduced in connection with regression. Use of this criterion, with models fitted using maximum likelihood.

```
> ar(LakeHuron, method="mle")

Call:
ar(x = LakeHuron, method = "mle")

Coefficients:
      1        2
 1.0437  -0.2496

Order selected 2  sigma^2 estimated as  0.4788
ar(LakeHuron, method="mle")     # AIC is used by default if
                                # order.max is not specified
```

This suggests an AR(2) model, with the parameter estimates given above. However it also has α_1 (= 1.0437) greater than one, i.e., a nonstationary process.

The search for a stationary AR process that will adequately model these data has failed. Indeed, while Figure 4.2B suggested that an AR process might be a good first approximation, it also had features that suggested that an AR process was unlikely to account adequately for the whole of the correlation structure. The lag 2 spike in the plot of partial autocorrelations is not consistent with a pure low order AR process. Moving average models, which will be discussed next, give the needed additional flexibility.

4.1.4 Autoregressive moving average (ARMA) models – theory*

In a moving average (MA) process of order q, the *error* term is the sum of an *innovation* ϵ_t that is specific to that term, and a linear combination of earlier *innovations* ϵ_{t-1}, $\epsilon_{t-2}, \ldots,$

[1]
```
LH.yw <- ar(x = LakeHuron, order.max = 1, method = "yw")
                          # autocorrelation estimate
                          # order.max = 1 for the AR(1) model
LH.yw$ar                  # autocorrelation estimate of alpha
LH.mle <- ar(x = LakeHuron, order.max = 1, method = "mle")
                          # maximum likelihood estimate
LH.mle$ar                 # maximum likelihood estimate of alpha
## Estimates of the series mean and of sigma^2 can be obtained by typing
LH.mle$x.mean             # estimated series mean
LH.mle$var.pred           # estimated innovation variance
```

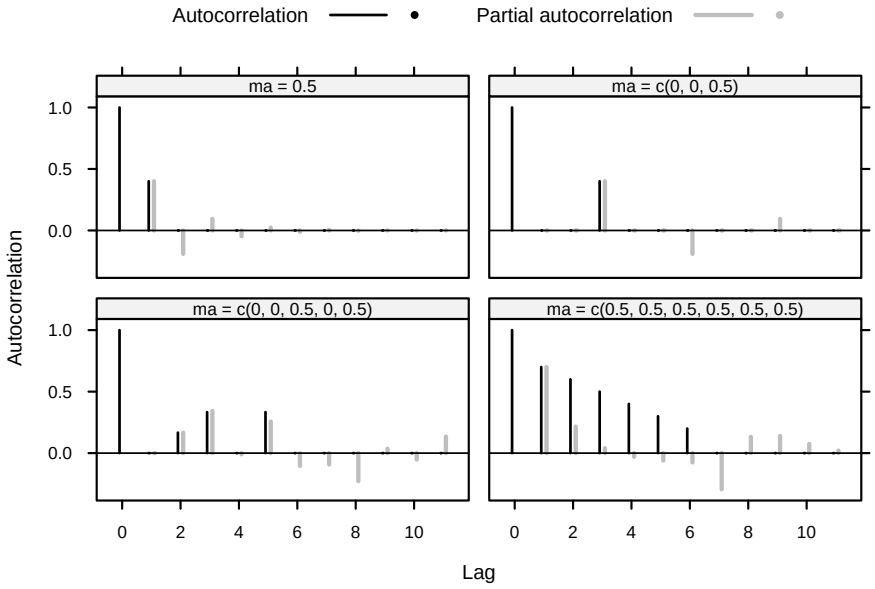

Figure 4.3: Theoretical autocorrelations for a moving average process, with the parameters shown. The thin gray lines are the partial autocorrelations.

ϵ_{t-q}. The equation is

$$X_t = \mu_t + \epsilon_t + b_1\epsilon_{t-1} + \cdots + b_q\epsilon_{t-q} \tag{4.1}$$

where $\epsilon_1, \epsilon_2, \ldots, \epsilon_q$ are independent random normal variables, all with mean 0. The autocorrelation between terms that are more than q time units apart is zero. Moving average terms can be helpful for modeling autocorrelation functions that are spiky, or that have a sharp cutoff.

An autoregressive moving average (ARMA) model is an extension of an autoregressive model to include "moving average" terms. Autoregressive integrated moving average (ARIMA) models allow even greater flexibility. (The model (AR or MA or ARMA) is applied, not to the series itself, but to a *differenced* series, where the differencing process may be repeated more than once.) There are three parameters: the order p of the autoregressive component, the order d of differencing (in our example 0) and the order q of the moving average component.

Differencing of order 1 removes a linear trend. Differencing of order 2 removes a quadratic trend. The downside of differencing is that it complicates the correlation structure. Thus, it turns an uncorrelated series into an MA series of order 1. Differencing can be done explicitly prior to analysis.[2]However it is easiest to let the `arima()` function do any differencing that is required, then reversing the process following the fitting of an ARMA model to the differenced series. For details, see `help(arima)`.

Figure 4.3 shows the autocorrelation functions for simulations of several different moving average models. Notice that

- with $b_1 = 0.5$ ($q = 1$) there is a spike at lag 1,
- with $b_1 = b_2 = 0$, $b_3 = 0.5$ ($q = 3$) there is a spike at lag 3,
- with $b_1 = b_2 = 0$, $b_3 = 0.5$, $b_4 = 0, b_5 = 0.5$ ($q = 5$) there are spikes at lags 2, 3 and 5,
- with $b_i = 0.5$ ($i = 1, \ldots, 5$), there are spikes at the first five lags.

[2]
```
## Calculate differenced series
## series.diff <- diff(series)
```

4.1.5 Automatic model selection?

The function `auto.arima()`, from the *forecast* package uses the AIC criterion in a model selection process that can proceed pretty much automatically. This takes some of the inevitable arbitrariness from the selection process. Fortunately, in applications such as forecasting or the regression example in the next section, what is important is to account for the major part of the correlation structure. A search for finesse in the detail may be pointless, with scant practical consequence.

The algorithm looks for the optimum AR order p, the optimum order of differencing and the optimum MA order q. Additionally, there is provision for seasonal terms and for for "drift". Drift implies that there is a constant term in a model that has $d > 0$. Seasonal terms, which are important in some applications, will not be discussed further here.

An exhaustive (non-stepwise) search can be very slow. The `auto.arima()` default is a stepwise search. At each iteration, the search is limited to a parameter space that is "close" to the parameter space of the current model. Values of AR and MA parameters are allowed to change by at most one from their current values. There are similar restrictions on the search space for seasonal and other parameters.

A good strategy may to be check the autocorrelation plot for "spikes" that may indicate high order MA terms, before proceeding to a stepwise search. Note however that, in a plot that shows lags 1 to 20 of a pure noise process, one can expect, on average, one spike that extends beyond the 95% pointwise limits,

In the call to `auto.arima()`, the parameter `start.q` should be set to the highest order of MA parameter that seems likely to be present. The spike in the plot of partial autocorrelations in Figure 4.2B might suggest trying `start.q=10`. Application of `auto.arima()` to the `LakeHuron` data gives the following:

```
> library(forecast)
## Use auto.arima() to select model
## NB: Use of the argument start.q=10 does not change the result.
> auto.arima(LakeHuron)
Series: LakeHuron
ARIMA(1,1,2)

Coefficients:
         ar1      ma1      ma2
      0.6475  -0.5837  -0.3279
s.e.  0.1292   0.1393   0.1082

sigma^2 estimated as 0.4816:  log likelihood = -102.56
AIC = 213.12   AICc = 213.55   BIC = 223.46
```

Now try

```
## Check that model removes most of the correlation structure
acf(resid(arima(LakeHuron, order=c(p=1, d=1, q=2))))
## The following achieves the same effect, for these data
acf(resid(auto.arima(LakeHuron)))
```

Observe that `auto.arima()` chose an ARIMA(1,1,2) model; i.e., the order of the autoregressive terms is $p = 1$, the order of the differencing is $d = 1$, and the order of the moving average terms is $q = 2$.

4.1.6 A time series forecast

The function `forecast()` in the *forecast* package makes it easy to obtain forecasts,.

```
LH.arima <- auto.arima(LakeHuron)
fcast <- forecast(LH.arima)
plot(fcast)
```

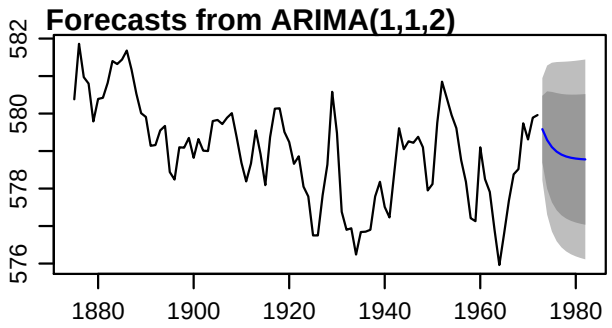

Figure 4.4: Forecast levels for Lake Huron, based on the `LakeHuron` data. The intervals shown are 80% and 95% prediction intervals.

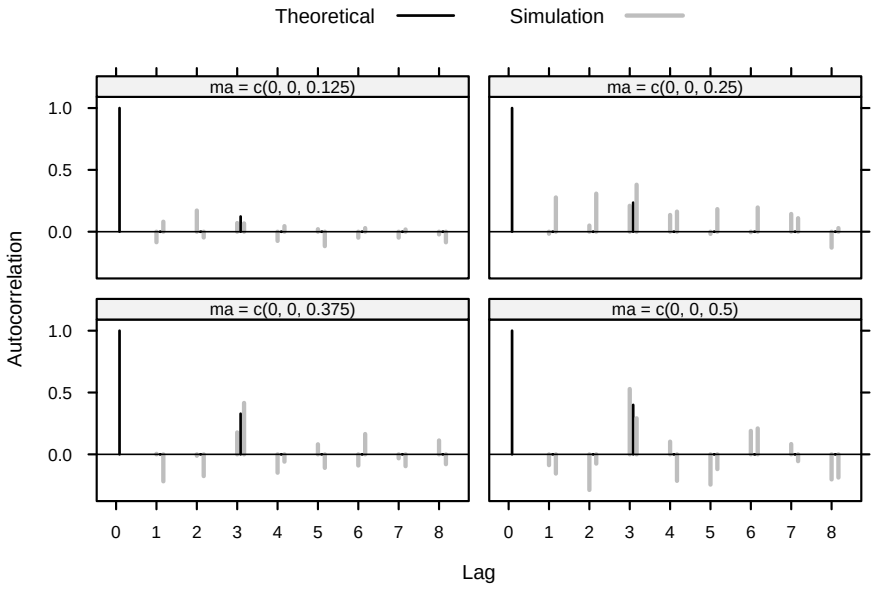

Figure 4.5: Results of two simulation runs (in gray), for an MA process of order 3, with `ma3` set in turn to 0.125, 0.25, 0.375 and 0.5. In each case, coefficients other than `ma3` were set to zero. The theoretical autocorrelation is shown, in each case, in black.

Figure 4.4 shows the results.

Use of simulation as a check

Simulation can be used to provide a check on the magnitude of MA or other coefficients that may be detectable. To simplify the discussion, we limit attention to a moving average process of order 3, with `ma1=0` and `ma2=0`, and `ma3` taking one of the values 0.125, 0.25, 0.374 and 0.5. Below we will tabulate, for each of these values, the level of MA process detected by `auto.arima()` over 20 simulation runs.

As an indication of the variation between different simulation runs, Figure 4.5 shows the autocorrelations and partial autocorrelations from two simulation runs. Code that plots results from a single set of simulation runs is:

```
oldpar <- par(mfrow=c(3,2), mar=c(3.1,4.6,2.6, 1.1))
for(i in 1:4){
  ma3 <- 0.125*i
  simts <- arima.sim(model=list(order=c(0,0,3), ma=c(0,0,ma3)),
```

```
                    n=98)
  acf(simts, main="", xlab="")
  mtext(side=3, line=0.5, paste("ma3 =", ma3), adj=0)
}
par(oldpar)
```

Now do 20 simulation runs for each of the four values of `ma5`, recording in each case the order of MA process that is detected:

```
set.seed(29)          # Ensure that results are reproducible
estMAord <- matrix(0, nrow=4, ncol=20)
for(i in 1:4){
  ma3 <- 0.125*i
  for(j in 1:20){
    simts <- arima.sim(n=98, model=list(ma=c(0,0,ma3)))
    estMAord[i,j] <- auto.arima(simts, start.q=3)$arma[2]
}
}
detectedAs <- table(row(estMAord), estMAord)
dimnames(detectedAs) <- list(ma3=paste(0.125*(1:4)),
                             Order=paste(0:(dim(detectedAs)[2]-1)))
```

The following table summarizes the result of this calculation:

```
> print(detectedAs)
       Order
ma3      0  1  2  3  4
  0.125 12  4  3  1  0
  0.25   7  3  2  8  0
  0.375  3  1  2 11  3
  0.5    1  1  0 15  3
```

Even with `ma3=0.375`, the chances are only around 50% that an MA component of order 3 will be detected.

4.2* Regression modeling with arima errors

The Southern Oscillation Index (SOI) is the difference in barometric pressure at sea level between the Pacific island of Tahiti and Darwin close to the northernmost tip of Australia. Annual SOI and rainfall data for various parts of Australia, for the years 1900–2005, are in the data frame `bomregions` (*DAAGxtras* package). (See Nicholls et al. (1996) for background.) To what extent is the SOI useful for predicting rainfall in one or other part of Australia?

This section will examine the relationship between rainfall in the Murray-Darling basin (`mdRain`) and `SOI`, with a brief look also at the relationship in northern Australia (`northRain`). The Murray-Darling Basin takes its name from its two major rivers, the Murray and the Darling. Over 70% of Australia's irrigation resources are concentrated there. It is Australia's most significant agricultural area.[3]

Figure 4.6 plots the two series. The code is

```
library(DAAGxtras)
## Plot time series mdRain and SOI: ts object bomregions (DAAGxtras)
plot(ts(bomregions[, c("mdRain","SOI")], start=1900),
     panel=function(y,...)panel.smooth(bomregions$Year, y,...))
```

Trends that are evident in the separate curves are small relative to the variation of the data about the trend curves.

[3]For a map, go to `http://www.bom.gov.au/silo/products/cli_chg/rain_timeseries.shtml`

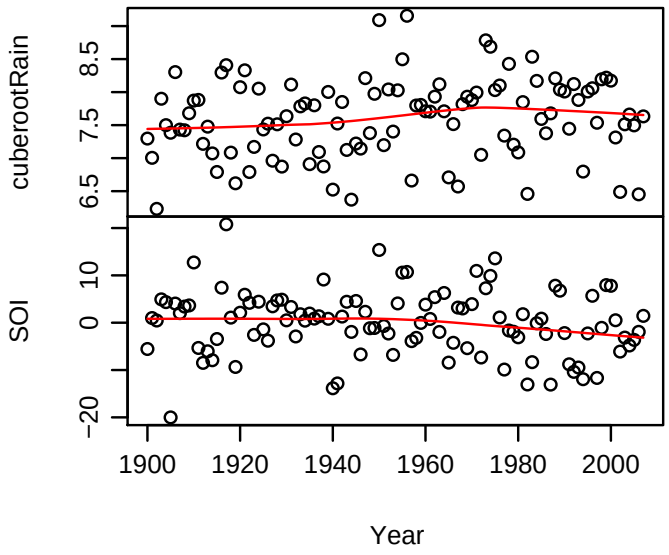

Figure 4.6: Plots of `mdRain` (Australia's Murray-Darling basin) and `SOI` (Southern Oscillation Index), against year.

Note also that a cube root transformation has been used to reduce or remove the skewness in the data. Use of a square root or cube root transformation (Stidd, 1953) is common for such data. The following code creates a new data frame `xbomsoi` that has the cube root transformed rainfall data, together with trend estimates (over time) for `SOI` and `md3rootRain`.

```
xbomsoi <-
  with(bomregions, data.frame(SOI=SOI, mdRain=mdRain,
                              md3rootRain=mdRain^0.33))
xbomsoi$trendSOI <- lowess(xbomsoi$SOI, f=0.1)$y
xbomsoi$trendRain <- lowess(xbomsoi$md3rootRain, f=0.1)$y
```

For understanding the relationship between `md3rootRain` and `SOI`, it can be helpful to distinguish effects that seem independent of time from effects that result from a steady pattern of change over time. Detrended series, for `md3rootRain` and for `SOI`, can be obtained thus:

```
xbomsoi$detrendRain <-
  with(xbomsoi, md3rootRain - trendRain + mean(trendRain))
xbomsoi$detrendSOI <-
  with(xbomsoi, SOI - trendSOI + mean(trendSOI))
```

Figure 4.7A plots `md3rootRain` against `SOI`, while Figure 4.7B gives the equivalent plot for the detrended

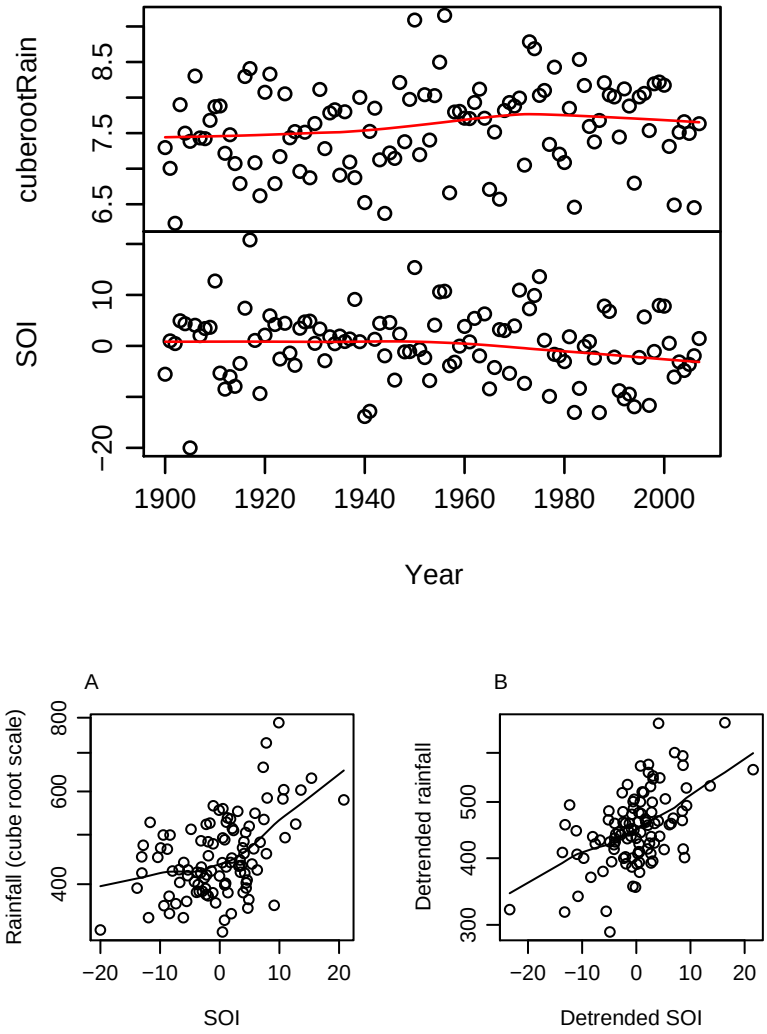

Figure 4.7: Plot of (A) rainfall (cube root scale) against SOI, and (B) detrended rainfall against detrended SOI.

series.[4]

Both relationships seem acceptably close to linear. Here, we model the relationship without the detrending. A linear relationship will be assumed. Figures 4.6 and 4.7 might suggest that the relationship for the detrended series will not be much different. We can check this directly. A further matter to check is the suggestion, in

[4]
```
oldpar <- par(mfrow=c(1,2), pty="s")
plot(md3rootRain ~ SOI, data = xbomsoi,
     ylab = "Rainfall (cube root scale)", yaxt="n")
rainpos <- pretty(xbomsoi$mdRain, 6)
axis(2, at = rainpos^0.33, labels=paste(rainpos))
with(xbomsoi, lines(lowess(md3rootRain ~ SOI, f=0.1)))
plot(detrendRain ~ detrendSOI, data = xbomsoi,
  xlab="Detrended SOI", ylab = "Detrended rainfall", yaxt="n")
axis(2, at = rainpos^0.33, labels=paste(rainpos))
with(xbomsoi, lines(lowess(detrendRain ~ detrendSOI, f=0.1)))
par(oldpar)
```

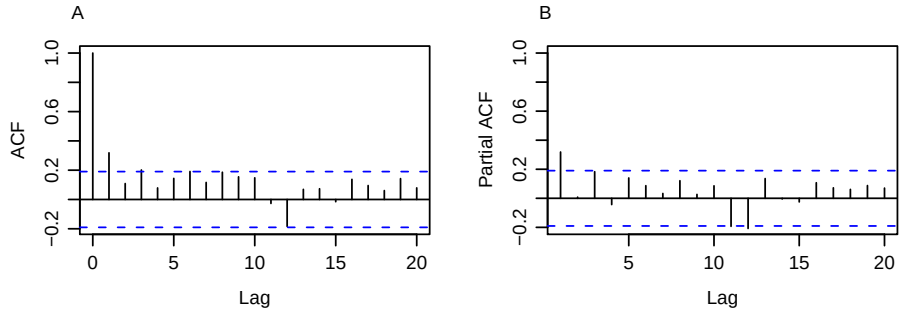

Figure 4.8: Panel A shows autocorrelation function for the residuals from the regression of `md3rootRain` (`mdRain^(1/3)`) on `SOI`. Panel B shows the partial autocorrelation function.

Figure 4.7A, of a quadratic relationship.

The time series structure of the data will almost inevitably lead to correlated errors that should be modeled or at least investigated, in order to obtain realistic standard errors for model parameters and to make accurate inferences.

A first step may be to fit a line as for uncorrelated errors; then using the residuals to get an initial estimate of the error structure. Figure 4.8 shows the autocorrelation structure and partial autocorrelation structure of the residuals. The code is:

```
par(mfrow=c(1,2))
acf(resid(lm(md3rootRain ~ SOI, data = xbomsoi)), main="")
pacf(resid(lm(md3rootRain ~ SOI, data = xbomsoi)), main="")
par(mfrow=c(1,1))
```

These plots have a spike at a lag of 5; is this large enough to try including a moving average term of order 5? We run the analysis both with the default starting value of 2 for the moving average parameter, and with `start.q=5`.

```
> ## Fit model with default start.q=2
> with(xbomsoi, auto.arima(md3rootRain, xreg=SOI))
Series: md3rootRain
ARIMA(0,1,1)

Call: auto.arima(x = md3rootRain, xreg = SOI)

Coefficients:
          ma1    xreg
      -0.9375  0.0427
s.e.   0.0398  0.0073

sigma^2 estimated as 0.2735:  log likelihood = -83.52
AIC = 173.03   AICc = 173.27   BIC = 181.05
> ## Not fit model setting start.q=5
> with(xbomsoi, auto.arima(md3rootRain, xreg=SOI, start.q=5))
Series: md3rootRain
ARIMA(3,1,4)

Call: auto.arima(x = md3rootRain, start.q = 5, xreg = SOI)

Coefficients:
          ar1     ar2      ar3      ma1      ma2     ma3      ma4     xreg
```

```
      -0.0602  0.1032  -0.3938  -0.8360  -0.3284  0.6521  -0.3886  0.0471
s.e.   0.1698  0.1596   0.2059   0.1715   0.2267  0.2195   0.2135  0.0081

sigma^2 estimated as 0.2501:  log likelihood = -77.68
AIC = 174.52   AICc = 176.38   BIC = 198.58
Warning message:
In auto.arima(md3rootRain, xreg = SOI, start.q = 5) :
  Unable to fit final model using maximum likelihood. AIC value approximated
```

Starting with `q=5` leads to the choice of an ARIMA(3,1,4) model with AIC=174.52, while starting with the default `q=2` leads to an ARIMA(0,1,1) model with AIC=173.03. The default search strategy tests the result of changing `p`, `d` and `q` in turn by 1 at each step. It tests also the inclusion of a drift term. For the calculations above, there is no sequence of available steps, each improving on the current value of the AIC, that lead from the starting model that had `q=5` to the ARIMA(0,1,1) model.

Note that the model selection process had other restrictions also; see `help(auto.arima)`. Note also that there were problems in fitting the more complex model.

The coefficient and standard error of the regression variable `SOI` is very similar in the two models. Here again, then, is the output from the simpler model:

```
> with(xbomsoi, arima(md3rootRain, xreg=SOI, order=c(0,1,1)))
Series: md3rootRain
ARIMA(0,1,1)

Call: arima(x = md3rootRain, order = c(0, 1, 1), xreg = SOI)

Coefficients:
          ma1     SOI
      -0.9375  0.0427
s.e.   0.0398  0.0073

sigma^2 estimated as 0.2735:  log likelihood = -83.52
AIC = 173.03   AICc = 173.27   BIC = 181.05
```

There is clear regression dependence of rainfall on SOI, with rainfall increasing, by about 0.047 in units of $\texttt{mdRain}^{0.33}$, for each unit increase in `SOI`. At the low end of the scale (a rainfall of about 256mm), a unit increase in `SOI` increases the rainfall by about 5mm. At the high end of the scale (821mm), the estimated increase is about 11.3mm.[5]

Figure 4.9 plots the residuals from this model against `SOI`, and shows also the autocorrelation plot of the residuals. These plots seem unexceptional.

The variable `SOI` explains around 24% of the variation in `md3rootRain`. This is found by comparing the estimate of σ^2 from the model that was fitted above with that from the model:

```
> ## Fit a constant mean model with the same correlation structure
> with(xbomsoi, arima(x = md3rootRain, order = c(0, 1, 1)))
. . . .
sigma^2 estimated as 0.358:  log likelihood = -98
. . . .
## Now calculate proportion of variance explained
> (.358-.273)/.358
[1] 0.2374
```

[5]
```
> range(xbomsoi$mdRain)
[1] 255.8 821.0
> (256^(1/3) + 0.0427)^3 - 256
[1] 5.2
> (821^(1/3) + 0.0427)^3 - 821
[1] 11.3
```

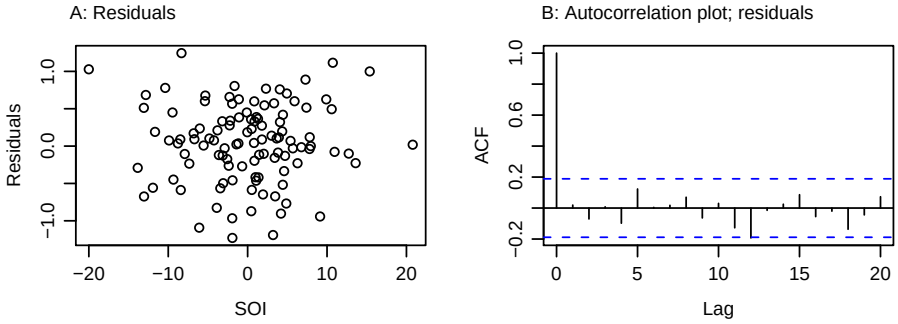

Figure 4.9: Panel A plots residuals from the arima model for `mdRain` against `SOI`. Panel B shows the autocorrelation plot for the residuals.

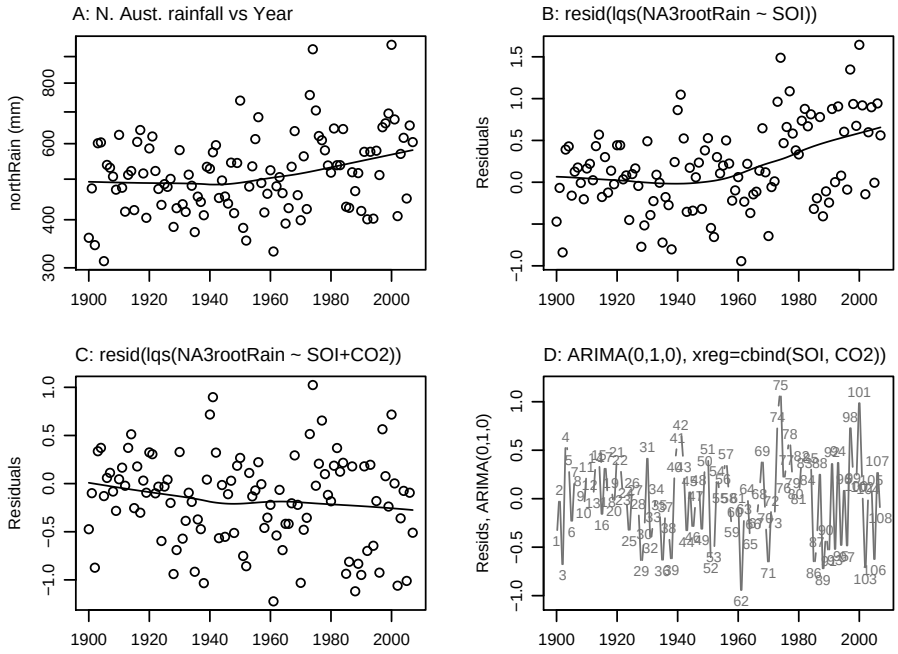

Figure 4.10: The x-ordinate in all four panels is `Year`. Panel A plots Northern Australian rainfall on a cube root scale. Panel B plots residuals from the regression of `NA3rootRain` (= `northRain^(1/3)`) on `SOI`. In panel C the residuals are from the regression on `SOI` and `CO2`. Panel D plots residuals, from an ARIMA(0,1,0) model for `NA3rootRain`, with `SOI` and `CO2` as regressors.

*The `northRain` series

The plot of `northRain^(1/3)` against `Year`, shown in Figure 4.10A, is flat until about 1950, after which it trends upwards. The plot against time of residuals from the regression of `northRain^(1/3)` against `SOI`, shown in Figure 4.10B shows a similar trend. It is tempting to use atmospheric carbon dioxide levels as a second covariate. This is indeed able, as Figure 4.10C shows, to account for most of the trend. Note that any variable that remained pretty much constant until 1950, thereafter trending upwards, would do just as well.

Note that the residuals in Figures 4.10B and C are from resistant regression. Code for Figures A, B and C is:

```
par(mfrow=C(2,2))
## Panel A
```

```
NA3rootRain <- bomregions$northRain^(1/3)
with(bomregions, plot(NA3rootRain ~ Year, yaxt="n",
                      xlab="", ylab="northRain (mm)"))
with(bomregions, lines(lowess(NA3rootRain ~ Year, f=0.75)))
yloc <- pretty(bomregions$northRain)
axis(2, at=yloc^(1/3), labels=paste(yloc))
## Panel B
north.lqs <- lqs(NA3rootRain ~ SOI, data=bomregions)
plot(resid(north.lqs) ~ Year, type="p", data=bomregions,
     xlab="", ylab="Residuals")
with(bomregions, lines(lowess(resid(north.lqs) ~ Year, f=0.5)))
## Panel C
north2.lqs <- lqs(NA3rootRain ~ SOI+CO2,
                  data= bomregions)
plot(resid(north2.lqs) ~ Year, type="p", data= bomregions,
     xlab="", ylab="Residuals")
with(bomregions, lines(lowess(resid(north2.lqs) ~ Year, f=0.75)))
```

Code for fitting the arima model, with output is:

```
> north.arima <- with(bomregions, auto.arima(NA3rootRain,
+                                            xreg=cbind(SOI, CO2)))
> print(north.arima)
Series: NA3rootRain
ARIMA(0,0,1) with non-zero mean

Call: auto.arima(x = NA3rootRain, xreg = cbind(SOI, CO2))

Coefficients:
        ma1  intercept    SOI    CO2
      0.194      5.148  0.036  0.009
s.e.  0.099      0.711  0.007  0.002

sigma^2 estimated as 0.204:  log likelihood = -67.36
AIC = 144.7   AICc = 145.3   BIC = 158.1
> ## Now plot the residuals, as in Panel D
> plot(resid(north.arima) ~ Year, type="p", data= bomregions)
```

Other checks and computations

The calculations below use as a starting point the model that was derived above for the rainfall in the Murray-Darling basin:

```
> (md.arima <- with(xbomsoi, auto.arima(md3rootRain, xreg=SOI)))
Series: md3rootRain
ARIMA(0,1,1)
. . . .
AIC = 173.0   AICc = 173.3   BIC = 181.1
```

Checking the autocorrelations among the residuals is recommended. The `Box.test()` function provides a so-called *portmanteau* test of whiteness (i.e. lack of autocorrelation) of the residuals. The Box-Ljung test statistic essentially adds up the squares of the first m sample autocorrelation estimates. Properly normalized, this statistic has an approximate chi-squared distribution on m degrees of freedom, provided there really is no autocorrelation among the residuals.

```
> Box.test(resid(md.arima), lag=20)

        Box-Pierce test

data:  resid(md.arima)
X-squared = 14.19, df = 20, p-value = 0.8208
```

Figure 4.7 suggested that a straight line regression model was adequate. We may however wish to carry out a formal check on whether a squared (`SOI^2`) term is justified. The code is

```
> (md2.arima <- with(xbomsoi, auto.arima(md3rootRain,
+                     xreg=poly(SOI,2))))
Series: md3rootRain
ARIMA(0,1,3)

Call: auto.arima(x = md3rootRain, xreg = poly(SOI, 2))

Coefficients:
         ma1     ma2    ma3      1     2
      -0.972  -0.035  0.074  2.991  0.96
s.e.   0.109   0.168  0.115  0.518  0.55

sigma^2 estimated as 0.264:  log likelihood = -81.63
AIC = 175.3   AICc = 176.1   BIC = 191.3
```

The quadratic orthogonal polynomial term falls short of the conventional 5% level of statistical significance. THe AIC statistic is slightly larger than before. Additionally, a different model has been chosen.

As well as checking that most of the sequential correlation has been removed from the residuals, a check on normality may be desirable:

```
## Examine normality of estimates of "residuals" ("innovations")
qqnorm(resid(md.arima, type="normalized"))
```

5

The R System – Additional Topics

The following add to or modify the discussion in Chapter 14 of the 3rd edition of *Data Analysis and Graphics Using R*.

5.1 Data Input and Data Manipulation

5.1.1 Accessing Data from the Internet

Data can be input directly from text files that are on the web. The following function accesses historical climate data from the Australian Bureau of Meteorology's web page:

```
`getbom` <-
  function(suffix=c("AVt","Rain"), loc="eaus"){
    webroot <- "http://www.bom.gov.au/web01/ncc/www/cli_chg/timeseries/"
    midfix <- switch(suffix[1], AVt="tmean/0112/", Rain="rain/0112/")
    webpage <- paste(webroot, midfix, loc, "/latest.txt", sep="")
    print(webpage)
    nam <- switch(loc, seaus="se", saus="south", eaus="east",
                  naus="north", swaus="sw",
                  qld="QLD", nsw="NSW", nt="NT",sa="SA",
                  tas="TAS", vic="VIC", wa="WA")
    x <- read.table(url(webpage))$V2
    if(suffix[1]%in%c("avt","AVt")){
      off <- switch(loc, seaus=14.72, saus=18.58, eaus=20.51, qld=23.23,
                    naus=24.71,swaus=16.3,nsw=17.33,nt=25.17,sa=19.45,
                    tas=10.35,vic=14.10,wa=22.47)
      x <- c(rep(NA,10), x+off)
    }
    x
  }
##
## Example of use
seRain <- getbom(suffix="Rain", loc="seaus")
```

The function `htmlParse()` in the *XML* package can be used to parse the XML/HTML content of web pages. Use the function `xpathSApply()` to find the nodes that are of interest. The help page for `htmlParse()` has extensive examples.

5.1.2 Changing the shape of data frames (or matrices)

The manipulations that will be described use the functions `melt()` and `cast()` in the *reshape* package. There are methods for both data frames and matrices

Melting and casting

The `melt()` function is a counterpart of `stack()` (discussed in Subsection 1.4.2) that has the advantage of carrying along other columns of the data frame.

```
> library(reshape)
> Jobs <- melt(jobs, measure.vars=names(jobs)[1:6],
               variable_name="Region", id.vars="Date")
> head(Jobs, 3)
      Date Region value
1 95.00000     BC  1752
2 95.08333     BC  1737
3 95.16667     BC  1765
```

The explicit use of the argument `id.vars="Date"` was unnecessary. If the argument `measure.vars` is given, the default is to carry along all other columns in the data frame.

The following returns the data frame `Jobs` that was created above, to the wide format:

```
jobsBack <- cast(Jobs, Date ~ Region)
> head(jobsBack,3)
      Date   BC Alberta Prairies Ontario Quebec Atlantic
1 95.00000 1752    1366      982    5239   3196      947
2 95.08333 1737    1369      981    5233   3205      946
3 95.16667 1765    1380      984    5212   3191      954
```

In the casting formula `Date  Region`, `Date` and `Region` uniquely identified a row of the data frame `Jobs`. In the data frame (`jobsBack`) that results, values of `Date` identify rows, and elements of `Region` identify columns. Where the casting formula does not uniquely specify rows, the argument `fun.aggregate` must be supplied.

5.2 Creation of R Packages

Much of the functionality of R, for many important tasks, comes from the packages that are built on top of base R. Users who make extenive use of R may soon find a need to document and organize both their own functions and associated data. Packages are the preferred vehicle for making functions and/or data available to others, or for use by posterity.

Organisation of data and functions into a package may have the following benefits:

- Where the package relates to a project, it should be straightforward to return to the project at some later time, and/or to pass the project across to someone else.
- Attaching the packages give immediate access to functions, data and associated documentation.
- Where a package is submitted to CRAN (Comprehensive R Archive Network) and used by others, this extends opportunities for testing and/or getting contributions from other workers. Checks that are required by CRAN ensure that the package (code and documentation) meets certain formal standards, including some modest level of consistency between code and documentation.

It is relatively straightforward to create packages that make no calls to externally compiled C or Fortran code. The following description applies to Unix, Linux and MacOS X systems. For Windows, tools that are needed for package construction must first be installed. See the manual "Writing R Extensions". See also `http://www.murdoch-sutherland.com/Rtools`.

1. Ensure that the workspace has only the functions and data objects that are to be included in the package.

   Thus, to create a package from the functions and data set from the image file `viewtemps.RData` that holds objects functions `plotD`, `plotT`, `plotTofD` (all functions) and `housetemps` (a data frame), start with an empty workspace and load the image file `viewtemps.RData`. There should the

be four objects in the workspace – the functions `plotD`, `plotT` and `plotTofD`, and the data frame `housetemps`.

2. Type, e.g.
   `package.skeleton(name = "viewtemps")`
   This creates all the needed directories and files or file skeletons, in a subdirectory **viewtemps** in the working directory. Instructions for proceeding further are placed in a file **Read-and-delete-me** in this directory; the following is a synopsis.
3. Consider setting aside a folder for use when creating new packages, e.g.
   `package.skeleton(name = "viewtemps", path="~/packages")`
4. Edit the DESCRIPTION file in the directory **viewtemps**, as required. (This is not absolutely necessary, if the file is for your own use.)
5. Edit the help file skeletons in **viewtemps/man**. (Optionally, combine help files for two or more functions.)
6. Go the parent directory of textbfviewtemps, and type

   ```
   R CMD check viewtemps
   ```

   Once this runs without error, go to the next step.
7. To builds the package tarball, type

   ```
   R CMD build viewtemps
   ```

To compile the package into a zip file on a Windows system, go to a DOS prompt in the parent directory and type:

```
Rcmd build --binary viewtemps
```

This assumes that the necessary tools have been installed.

Namespaces

Packages can have their own namespaces, with private functions and classes that are not ordinarily visible from the command line, or from other packages. For example, the function `intervals.lme()` that is part of the *nlme* package must be called via the generic function `intervals()`.

5.3 Lattice Graphics, and the *grid* Package

Details of lattice plots that will now be discussed include: customization of arguments, creation of keys or legends, interaction with plots, the use of grid functions to supplement plots, and the ”printing” of multiple lattice plots on the one graphics page.

5.3.1 Lattice Graphics vs Base Graphics

Note an important difference between the effects of base and lattice graphics functions:

- In base graphics, plots go to a graphics page (possibly the first of a sequence of pages), on whatever device is open to receive them. For a screen device, plots go to the screen. For a hardcopy device plots go, in the first place, to a file.
- Lattice functions create trellis objects. Objects can be created even if no device is open. Such objects can be updated. Objects are plotted (by this time, a device must be open), either when output from a lattice function goes to the command line (thus implicitly invoking the `print()` command), or by the explicit use of `print()`.

The updating feature makes it possible to build up and/or modify the graphics object in steps. While it is possible to add new features to a "printed" page, this is in the nature of an afterthought in the design.

5.3.2 Groups within data, and/or columns in parallel

Here are selected lines from the data set `grog` (*DAAGxtras* package):

	Beer	Wine	Spirit	Country	Year
1	5.24	2.86	1.81	Australia	1998
2	5.15	2.87	1.77	Australia	1999
....					
9	4.57	3.11	2.15	Australia	2006
10	4.50	2.59	1.77	NewZealand	1998
11	4.28	2.65	1.64	NewZealand	1999
....					
18	3.96	3.09	2.20	NewZealand	2006

There are three liquor products (drinks), in different columns, and two countries, occupying different rows that are indexed by the factor `Country`. The function `xyplot()` can accommodate any of the following:

- Different symbols and/or colors are used for different drinks, within the one panel. Different panels must then be used for different countries, as in Figure 5.1.[1] Or if different countries are in the same panel, then drinks must be separated across different panels.
- Use a 3 drinks $\times$ 2 countries, or 2 countries $\times$ 3 drinks layout of panels.

Where plots are superposed in the one panel and, e.g., regression lines or smooth curves are fitted, this is done separately for each different set of points. Different colors, and/or by different symbols and/or line styles, can be used to make the necessary distinctions.

Code for Figure 5.1 is:

```
## Simple version of plot
grogplot <- xyplot(Beer+Spirit+Wine ~ Year | Country, data=grog,
                   outer=FALSE, auto.key=list(columns=3))
## Enhance, and print enhanced code
update(grogplot, ylim=c(0,5.5),
       xlab="", ylab="Amount consumed (per person)",
       par.settings=simpleTheme(pch=c(1,3,4)))
```

The footnote[2]has alternative code that updates the object, then uses an explicit `print()`.

Observe that:

- Use of `Beer+Spirit+Wine` gives plots for each of `Beer`, `Spirit` and `Wine`. The effect of `outer=FALSE` is that these appear in the same panel.
- Conditioning by country (`| Country`) gives separate panels for separate countries.
- The function `simpleTheme()` is convenient for setting or changing point and line settings.

For separate panels for the three liquor products (different levels of `Country` can now use the same panel), specify `outer=TRUE`:

[1]The data (dataset `grog`, from *DAAGxtras*) are 1998 – 2006 Australian and New Zealand apparent per person annual consumption (in liters) of the pure alcohol content of `Beer`, `Wine` and `Spirit`. Data, based on Australian Bureau of Statistics and Statistics New Zealand figures, are obtained by dividing estimates of total available alcohol by number of persons aged 15 or more.

[2]
```
## Update trellis object, then print
frillyplot <-
  update(grogplot, ylim=c(0,5.5), xlab="", ylab="Amount consumed (per person)",
         par.settings=simpleTheme(pch=c(1,3,4)))
print(frillyplot)
```

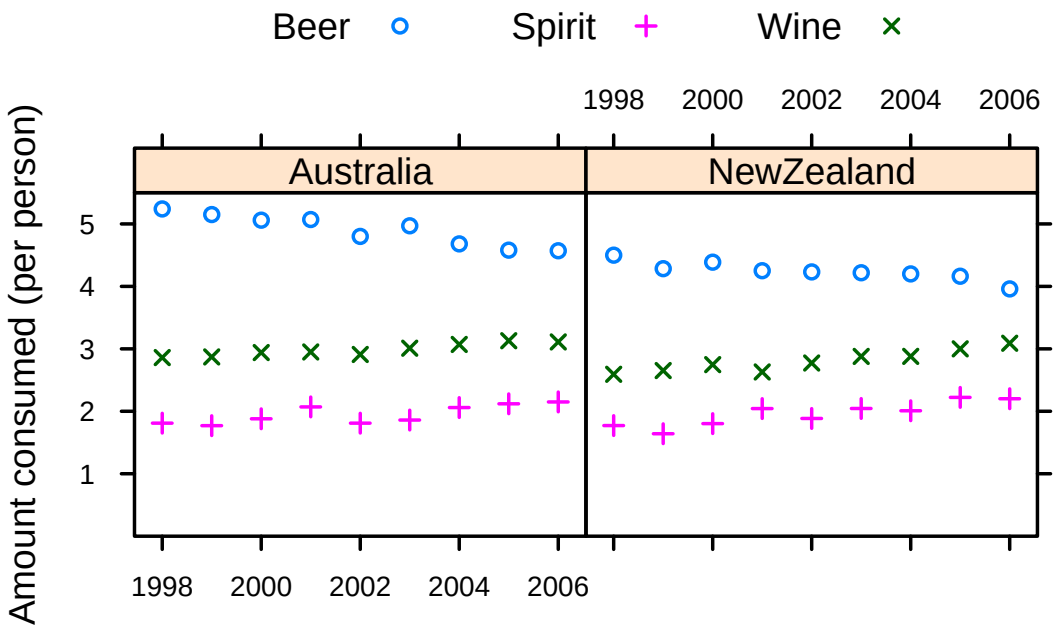

Figure 5.1: Australian and New Zealand apparent per person annual consumption (in liters) of the pure alcohol content of liquor products, for 1998 to 2006.

```
xyplot(Beer+Spirit+Wine ~ Year, groups=Country, outer=TRUE,
       data=grog, auto.key=list(columns=2) )
```

For breaking data down according to levels of a factor, options are:

Overplot (a single panel)	Separate panels (conditioning)
`Beer ~ Year, groups=Country`	`Beer ~ Year | Country`

For plotting columns in parallel, as in `Beer+Wine+Spirit ~ Year`, options are:

Overplot (a single panel)	Separate panels (conditioning)
`outer=FALSE`	`outer=TRUE`

5.3.3 Lattice Parameters and Graphics Features

Point, line and fill color settings, using `simpleTheme()`

The function `simpleTheme()` creates a "theme", i.e., a list of parameter settings, in a form that can be supplied: (i) in the argument `par.settings` in the graphics function call; or (ii) in the argument `theme` in a call to `trellis.par.set()`, prior to calling the graphics function. A further possibility is to include an argument `theme` when using `trellis.device()` to start a new device. This has the default `retain=FALSE`, with the result that unless otherwise specified, parameters are reset to their defaults for the relevant device.

The following creates two "themes":

```
settings1 <- simpleTheme(pch = c(1,3,4), cex=1.25)
settings2 <- simpleTheme(pch = c(1,3,4), alpha=0.5)
```

Use of `settings1` may be appropriate when the number of points is small, while `settings2` may be appropriate when there are many points and there is extensive over-plotting. Here, `alpha` controls the back-

ground transparency (c.f., also, `alpha.points` and `alpha.line`). Use of a value less than 1 helps in showing the density of points in regions where there is extensive overlapping.

The following gives the symbols and size of symbol used in Figure 5.1:

```
grogplot0 <- xyplot(Beer+Spirit+Wine ~ Year | Country, outer=FALSE,
                    data=grog, ylim=c(0,5.5))
grogplot <- update(grogplot0, par.settings=settings1)
print(grogplot)
```

The settings are stored as part of the graphics object `grogplot`.

Consider now the use of `trellis.par.set()` to change the settings globally, os that they remain in place until there is a further change or a new device is opened.

```
trellis.par.set(settings2)
```

Then `print(grogplot)` will use `settings1` which are stored as part of the object, while `print(grogplot0)` will use the global `settings2`.

Beyond `simpleTheme()`

For changes that go beyond what `simpleTheme()` allows, it is necessary to know the names under which settings are stored. To inspect these, type:

```
> names(trellis.par.get())
 [1] "fontsize"         "background"       "clip"
. . .
[28] "par.sub.text"
```

For a visual display that shows default settings for points, lines and fill colour, try the following:

```
trellis.device(color=FALSE)
show.settings()
trellis.device(color=TRUE)
show.settings()
```

The following sets the fontsize. Notice the separate settings for text and symbols:

```
trellis.par.set(list(fontsize = list(text = 7, points = 4)))
```

Parameters that affect axes, tick marks, and axis labels

These are readily manipulated by use of the `scales` argument to the lattice function. Consider data (in `jobs`, from *DAAG*) on quarterly labor force numbers, in six regions of Canada, over 1995-1996. The code is:

```
## Create a simplified version of the graphics object
jobs.xyplot <-
  xyplot(Ontario+Quebec+BC+Alberta+Prairies+Atlantic ~ Date,
         data=jobs, type="b", layout=c(3,2), ylab="Number of jobs",
         scales=list(y=list(relation="sliced", log=TRUE)),
         outer=TRUE)
## Update jobs.xyplot, with various enhancements
ylabpos <- exp(pretty(log(unlist(jobs[,-7])), 100))
ylabels <- paste(round(ylabpos),"\n(", log(ylabpos), ")", sep="")
## Create a date object 'startofmonth'; use this instead of 'Date'
atdates <- seq(from=95, by=0.5, length=5)
datelabs <- format(seq(from=as.Date("1Jan1995", format="%d%b%Y"),
                   by="6 month", length=5), "%b%y")
update(jobs.xyplot, xlab="", between=list(x=0.5, y=0.5),
```

```
        scales=list(x=list(at=atdates, labels=datelabs),
                    y=list(at=ylabpos, labels=ylabels), tck=0.6) )
```

The enhancements are:

- The y-axis labels show number of jobs, with $\log(\text{number})$ in parentheses underneath.
- Dates of the form `Jan95` label the x-axis.
- Tick marks are reduced in length (`tck=0.6`, i.e., 60% of the default).

Notice also the use of `between=list(x=0.5, y=0.5)` to add horizontal and vertical space between the panels, ensuring that the tick labels do not overlap.

5.3.4 A further example

Data in the dataset `ais` (*DAAG*) were collected with a view to studying possible differences in blood characteristics, between athletes in endurance-related events and those in power-related events. The help page for `ais` for details of the measurements, including a variety of blood cell counts. Here is a breakdown, by sex and sport, of numbers:

```
> with(ais, table(sex,sport))
   sport
sex B_Ball Field Gym Netball Row Swim T_400m T_Sprnt Tennis W_Polo
  f     13     7   4      23  22    9     11       4      7      0
  m     12    12   0       0  15   13     18      11      4     17
```

There are 202 athletes in total, all from the Australian Institute of Sport.

Figure 5.2 plots blood cell to plasma ratio (%) against red cell count, for two sports only. The two sports appear in the one panel, distinguished by different symbols and/or colours. Females and males are in separate panels. Figure 5.2 shows a suitable plot, adding also regression lines. Again, an initial basic plot is updated to give the desired result.

Code is:

```
basic1 <-  xyplot(hc ~ rcc | sex, groups=sport[drop=TRUE],
                  data=subset(ais, sport %in% c("B_Ball", "Swim")),
                  ylab="Blood cell to plasma ratio (%)")
parSet <- simpleTheme(pch = c(1,3), lty=1:2, lwd=1.5)
xaxlab <- expression("Red cell count (10"^{12}*"."*L^{-1}*")")
basic2 <- update(basic1, par.settings=parSet,
                 auto.key=list(columns=2, lines=TRUE),
                 scales=list(tck=0.5), xlab=xaxlab)
print(update(basic2, type=c("p", "r")))
```

Again, there are details that require explanation:

- Omission of `drop=TRUE` from `groups=sport[drop=TRUE]`, will result in one legend item for each of the 10 sports, usually not what is required. Subsetting a factor leaves the levels attribute unchanged, even if some levels are no longer present in the data.
- As in base graphics, graphical annotation (tick labels, axis labels, labels on points, etc.) can be given using the function `expression()`. In this context,"expression" is broadly defined. Thus expressions can have terms that are character strings. See `help(plotmath)`.
- The argument `type=c("p","r")` gives both points and fitted regression lines.

5.3.5 Keys – `auto.key`, `key` & `legend`

The argument `auto.key=TRUE` gives a basic key that identifies colors, plotting symbols and names for the groups. For greater flexibility, `auto.key` can be a list. Settings that are often useful are:

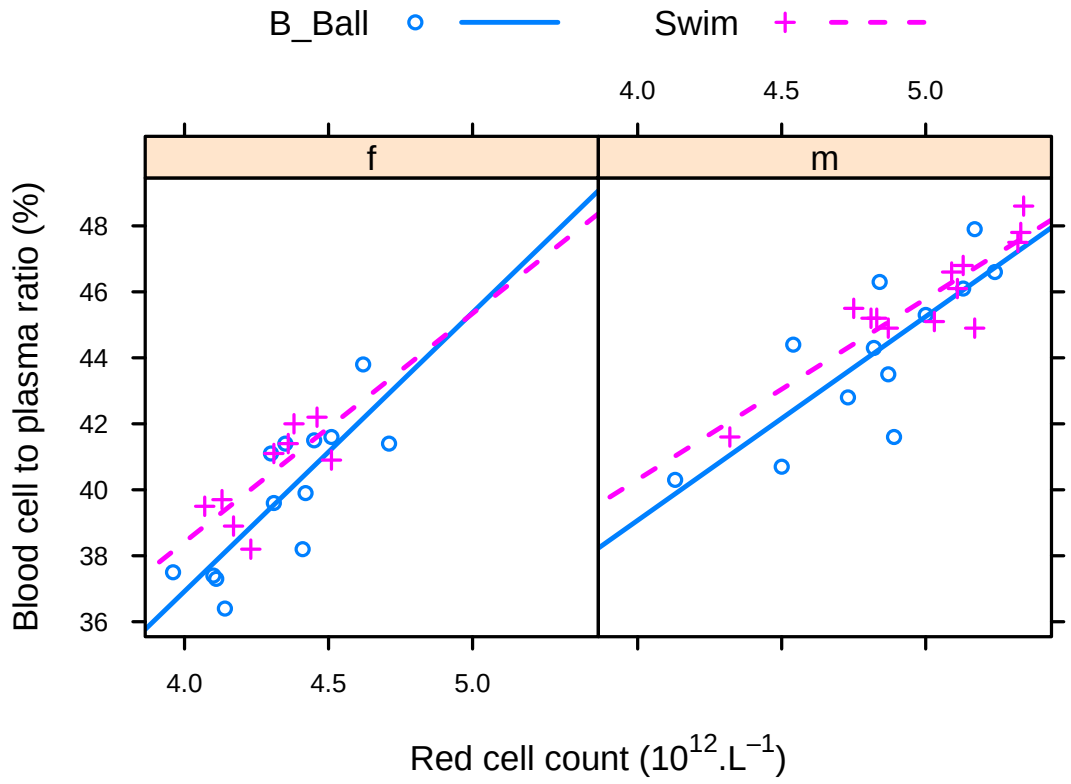

Figure 5.2: Blood cell to plasma ratio (`hc`) versus red cell count (`rcc`), by `sex` (different panels) and `sport` (distinguished within each panel). The argument `type=c("p", "r")` displays both points (`"p"`) and regression lines (`"r"`).

- `points`, `lines`: in each case set to `TRUE` or `FALSE`.
- `columns`: number of columns of keys.
- `x` and `y`, which are coordinates with respect to the whole display area. Use these with `corner`, which is one of `c(0,0)` (bottom left corner of legend), `c(1,0)`, `c(1,1)` and `c(0,1)`.
- `space`: one of `"top"`, `"bottom"`, `"left"`, `"right"`.

Use of `auto.key` sets up the call `key=simpleKey()`. If not otherwise specified, colors, plotting symbols, and line type use the current trellis settings for the device. Unless `text` is supplied as a parameter, `levels(groups)` provides the legends. If necessary, use `legend=NULL` when updating, to remove an existing key as a preliminary to adding a different key.

5.3.6 Panel Functions and Interaction with Plots

Each lattice command that creates a graph has its own panel function. Thus `xyplot()` has the panel function `panel.xyplot()`. The following are equivalent:

```
xyplot(ACT ~ year, data=austpop)
xyplot(ACT ~ year, data=austpop, panel=panel.xyplot)
```

The user's own function can be substituted for `panel.xyplot()`. Panel functions that may be used, either in combination with functions such as `panel.xyplot()` or separately, include:

- `panel.points`, `panel.lines()` and a number of other such functions that are documented on the same help page as `panel.points`);

- `panel.abline()`, `panel.curve()`, `panel.rug()`, `panel.average()` and a number of other functions that are documented on the same help page as `panel.abline()`.

In the following update of `basic2`, used for Figure 5.2, the lines for the two sports are parallel:

```
update(basic2,
    strip=strip.custom(factor.levels=c("Female","Male")),
    # In place of level names c("f", "m"), use c("Female", "Male")
    panel=function(x, y, groups, subscripts, ...){
         panel.superpose(x,y, groups=groups,
                         subscripts=subscripts, ...)
         b <- coef(lm(y ~ groups[subscripts] + x))
         lcol <- trellis.par.get()$superpose.line$col
         lty <- trellis.par.get()$superpose.line$lty
         panel.abline(b[1], b[3], col=lcol[1], lty=lty[1])
         panel.abline(b[1]+b[2], b[3], col=lcol[2], lty=lty[2])
         })
```

When there are groups within panels, `panel.xyplot()` calls `panel.superpose()`. The customized panel function has the structure:

```
  panel=function(x, y, groups, subscripts, ...){
       panel.superpose(x,y, groups=groups,
                       subscripts=subscripts, ...)
       . . . .
       panel.abline(b[1], b[3], col=lcol[1], lty=lty[1])
       panel.abline(b[1]+b[2], b[3], col=lcol[2],
                       lty=lty[2])
       })
```

As the function `panel.superpose()` lacks an option to fit and display multiple parallel lines, the user must supply the needed code. The following calculates the regression slope estimates:

```
    b <- coef(lm(y ~ groups[subscripts] + x))
      # NB: groups (but not x and y) must be subscripted
```

The lines are then drawn one at a time, taking care that the line settings agree with those that will appear in the key.

Interaction with lattice plots – the playwith package

For using *playwith*, the GTK+ toolkit must be installed. For details, go to the website `http://playwith.googlecode.com/`.

For installing the *playwith* package type, from the command line:

```
install.packages("playwith", dependencies=TRUE)
```

Now type, for example

```
library(DAAGxtras)
library(playwith)
playwith(xyplot(age ~ distance, data=hotspots),
                labels=hotspots$name)
```

An alternative is

```
gph <- xyplot(age ~ distance, data=hotspots)
playwith(update(gph), labels=hotspots$name)
```

The menu that appears to the left of the graph can be used to initiate single click identification, to add annotation or arrows, or to mark out a rectangle on the graph for zooming in or out. If labels are not specified, row names are used.

Note also the function `latticist()` in the *latticist* package. When called with a data frame as argument, this opens a window that has graphical summary information on the columns of the data frame. Additionally, the window gives a graphical user interface to the creation of further lattice plots from the data frame. As when `playwith()` is used as a wrapper for a call to a lattice function, various annotation features are available.

5.3.7 Interaction with lattice plots – focus, interact, unfocus

As described here, interaction starts with the use of `trellis.focus()` to focus down to the relevant "viewport", by default a panel. It may be called without arguments. If there is only one panel, it is then selected immediately. If there is more than one panel, the user chooses a panel by clicking on it.

Other choices of `name` include `"panel"`, `"strip"`, `name="legend"` and `"toplevel"`. For `name="legend"`; `side` should be indicated. See below (Subsection 5.3.8) for an example.

Use of `panel.identify()` to label points interactively

Here is an example of interactive labeling.

```
## Use of xyplot(): data frame tinting (DAAG)
library(lattice)
xyplot(it ~ csoa | sex, data=tinting)
trellis.focus("panel", column=1, row=1)
panel.identify(labels=as.character(tinting$target))
## Now click near points as required.
## Terminate by right clicking inside the panel.
## Now interact with panel 2
trellis.focus("panel", row=1, column=2)
panel.identify(labels=as.character(tinting$target))
## Click, ..., and right click
```

By default, the `x` and `y` arguments to the function `panel.identify()` are taken to be those that were supplied to the lattice function, here `xyplot()`.

Use of `panel.text()` to label points

Following a call to `trellis.focus()`, grid functions can be used to supplement plots. The following adds text labels:

```
xyplot(Brainwt ~ Bodywt, data=primates)
trellis.focus("panel", row=1, column=1, clip.off=TRUE)
xyetc <- trellis.panelArgs()
panel.text( x=xyetc$x, y=xyetc$y, labels=row.names(primates), pos=3)
trellis.unfocus()
```

Use the call `trellis.panelArgs()` to extract arguments that are available to panel functions following a call to `trellis.focus()`.

Functions that may be called include `panel.lines()` and related functions, and `panel.abline()` and related functions, as described earlier.

Non-interactive use of `trellis.focus()`

For non-interactive use, turn off highlighting, i.e., the call becomes `trellis.focus(highlight=FALSE)`.

5.3.8 Multiple lattice graphs on a graphics page

The function `print.trellis()` takes an argument `position` that controls the positioning of a graph on the graphics page. The following demonstrates its use. Note also the placing of text on the `"toplevel"` viewport.

```
## Two lattice graphs on one page: data frame cuckoos (DAAG)
trellis.par.set(layout.heights=list(key.top=0.25, axis.top=0.5,
                                    bottom.padding=0.15))
cuckoos.strip <- stripplot(species ~ length, xlab="", data=cuckoos)
print(cuckoos.strip, position=c(0,.475,1,1))
  # Left bottom corner is (0,.475); right upper corner is (1,1)
trellis.focus(name="toplevel")
panel.text("A", x=0.05, y=0.95)
  # Here, x=0.05 translates to x=unit(0.05,"npc"); the range is (0,1)
trellis.unfocus()
cuckoos.bw <- bwplot(species~length, xlab="Egg length (mm)",
                     data=cuckoos)
print(cuckoos.bw, newpage=FALSE, position=c(0,0,1,.525))
trellis.focus(name="toplevel")
panel.text("B", x=0.05, y=0.95)
trellis.unfocus()
```

Note that conventions for positioning a graph differ between base graphics and lattice graphics.

Use of `textGrob()` *in the* `legend` *argument*

The following shows an alternative way to position the label, using the function `textGrob()`, from the *grid* package, to control the detailed content and placing of a "legend".

```
library(grid)
stripplot(species ~ length, xlab="", data=cuckoos,
          legend=list(top=list(fun=textGrob,
                               args=list(label="A", x=0))))
  # Here, x=0 is equivalent to x=unit(0,"npc"); the range is (0,1)
```

Overlaid plots with different scales

The dataset `edcT` holds estimates of temperature anomalies `dT` (i.e., differences from the average of approximately -54.5degC over the past 1000 years), from the EPICA (European Project for Ice Coring in Antarctica) Dome C ice cores that cover 0 to 800,000 years before the present. The dataset `edcCO2` (*DAAGxtras*) holds estimates of carbon dioxide levels, over the same time period. Figure 5.3 overlays mildly smoothed versions of the two data series.

The plot uses the function `doubleYScale()` from the *latticeExtra* package. Here is the code:

```
library(latticeExtra)
library(DAAGxtras)
CO2smu <- with(edcCO2, supsmu(age, co2, span=.005))
tempsmu <- with(edcT, supsmu(Age, dT, span=.001))
co2_layer <- xyplot(y ~ I(-x), data=CO2smu, type="l",
```

```
                    xlab="Years Before Present",
                    ylab = expression(CO[2]*" (ppm)"))
temp_layer <- xyplot(I(y+54.5) ~ I(-x), data=tempsmu, type="l",
                     ylab=expression("Temperature ("*degree*C*")"))
doubleYScale(co2_layer, temp_layer, add.ylab2 = TRUE)
```

Two separate graphics objects, `co2_layer` and `temp_layer`, are created. The function `doubleYScale()` superposes the two graphs. The argument `add.ylab2=TRUE` ensures that the y-label for the second object will appear in the right margin.

Base and trellis plots on the same graphics page

For example, base graphics commands such as `mtext()` can be used to label a lattice plot:

```
plot(0:1, 0:1, type="n", bty="n", axes=FALSE, xlab="", ylab="")
mtext(side=3, line=3, "Lattice bwplot (i.e., boxplot)")
cuckoos.bw <- bwplot(species~length, data=cuckoos)
print(cuckoos.bw, newpage=FALSE)
```

5.4 An Implementation of Wilkinson's *Grammar of Graphics*

The *ggplot2* syntax implements the ideas set out in Wilkinson (2005). It is highly consistent, but less stylized than *lattice*.

Examples given here will use the wrapper function `quickplot()`. This remarkably versatile function can be used to create a wide variety of ggplot graphics objects.

5.4.1 Australian rain data

Figure 5.4 plots annual rainfall for South-East Australia.

Here is the code:

```
library(DAAG)
library(ggplot2)
## Default loess smooth, with SE bands added.
quickplot(Year, seRain, data=bomsoi, geom=c("point","smooth"),
          span=0.1, xlab="", ylab="Av. rainfall, M-D basin")
```

Try also the following:

```
library(splines)
quickplot(Year, seRain, data=bomsoi, geom=c("point", "quantile"),
          formula = y ~ ns(x,5), quantiles=c(0.2,0.5,0.8) )
```

The normal spline basis `ns(x,5)` is supplied to the function that fits the quantiles, so that 5 d.f. spline curves are fitted at the 20%, 50% and 80% quantiles.

5.4.2 Physical measurements of Australian athletes

Figure 5.5 plots height against weight, for the `ais` data. Boxplots that show the distributions of heights, and two-dimensional density contour estimates have been added. Figure 5.5 shows boxplots (`geom="boxplot"`), by `sport` (given as the x-variable) and `sex` (separate panels): The code is:

```
## Overlay scatterplots with boxplots and with density contours
quickplot(wt, ht, xlab="Weight (kg)", ylab="Height (cm)", data=ais,
          geom=c("boxplot", "point", "density2d"),
          facets = . ~ sex)
```

The facets argument takes the form row.var col.var, where row.var indexes the rows of panels, col.var indexes columns. and . is used as a placeholder when there is one row or one column only.

Try also:

```
## Two panels (columns): sexes have different colors and symbols
quickplot(wt, ht, data=aisBS, type="point",
          colour=sex, shape=sex,
          facets = . ~ sport)
## Single panel: distinguish sexes by colors; sports by symbols
quickplot(wt, ht, data=aisBS, type="point",
          colour=sex, shape=sport)
```

To get different colours for different levels of sport, specify colour=sport, For different plotting symbols, specify shape=sport. For different sizes of symbol, specify size=sport. Two of these may appear together. For example:

```
aisBS <- subset(ais, sport %in% c("Row","Swim"))
aisBS$sport <- factor(aisBS$sport)
quickplot(wt, ht, data=aisBS, type="point",
          colour=sex, shape=sex, size=I(2.5),
          facets = . ~ sport)
## Single panel: distinguish sexes by colors; sports by symbols
quickplot(wt, ht, data=aisBS, type="point",
          color=sex, shape=sport, size=I(2.5))
## Single panel: distinguish sexes by colors; sports by symbol size
quickplot(wt, ht, data=aisBS, type="point",
          color=sex, size=sport)
```

Possible choices of geom, additional to those already demonstrated, are "path" (join points), "line" (join points), "histogram", and "density".

Note the difference between:

> size=I(2.5),used to make points somewhat larger than otherwise.
> size=2.5, which specifies a mapping from the vector with the single element 2.5 to all points in the data. This does change the point size, but it adds adds an extraneous key. (This is analagous to the use of size=sport to specify a mapping from sport to size.)

Thus also, to make all points red, specify color=I("red"), not color="red"

Note also the following:

```
## Change the base pointsize for text to 8
theme_set(theme_gray(base_size=8))  # Gray theme
theme_set(theme_bw(base_size=8))    # Black and white theme
## Modify the pointsize
update_geom_defaults("point", aes(cex=1.25))
```

The function aes() generates aesthetic mappings. These map variables in the data to visual properties (aesthetics) of geoms.

Consult the web page http://had.co.nz/ggplot/ for up to date information on *ggplot2*.

5.5 Dynamic Graphics – the *rgl* and rggobi packages

Both these packages provide three-dimensional dynamic graphics. The following code may be used to give the left panel in Figure 5.6 It uses functions in the *Rcmdr* package – scatter3d() and identify3d(), which may be more convenient for novices than the *rgl* functions that are called by scatter3d() and identify3d().

```
## The Rcmdr and rgl packages must be installed
library(Rcmdr)       # This makes scatter3d() available
## The call to open3d() is optional, but see below
open3d(usermatrix= matrix(c(
       1, 0.0000000,  0.0000000,    0,
       0, 0.9659258, -0.2588190,    0,
       0, 0.2588190,  0.9659258,    0,
       0, 0.0000000,  0.0000000,    1), byrow=TRUE))
par3d(cex=0.6)       # Optional.  Requires an rgl device to be open
with(nihills, scatter3d(x=log(dist), y=log(climb), z=log(time),
                        grid=FALSE, surface.col="gray",
                        point.col="black", axis.scales=FALSE))
with(nihills, identify3d(x=log(dist), y=log(climb), z=log(time),
                         labels=row.names(nihills), col="gray40"))
```

The optional initial call to `open3d()` serves two purposes:

- The `usermatrix` argument is used to ensure that the initial view is that shown in Figure 5.6.
- Once the *rgl* graphics device is open, parameter settings can be modified, prior to displaying the plot.

Following the call to `identify3d()`, use the middle (or maybe right) mouse button to drag a rectangle around any point that is to be labeled. To cease identifying points, make a middle (or right) click on an empty region of the plot. Use `rgl.snapshot()` to save the current plot into a file.

The *rggobi* package offers a wider range of features, via an interface to the GGobi system (Cook and Swayne, 2007). For installation details go to `http://www.ggobi.org/`.

5.6 Further Reading

The definitive document is the relevant up to date version of the R Language Definition document (R Core Development Team). Recent texts that give comprehensive accounts of R syntax and semantics are Chambers (2007); Gentleman (2008). See also Muenchen (2008).

For citing R in a publication, use R Development Core Team (2004). For historical background, see Ihaka and Gentleman (1996). Note that the R function `citation()` can be used to obtain citation information, both for R itself and for any installed package, thus:

```
citation()
citation("DAAG")
```

References

Carroll, R., 2004. Measuring diet. Texas A & M Distinguished Lecturer series.
URL `http://stat.tamu.edu/~carroll/talks.php`

Chambers, J. M., 2007. *Software for Data Analysis: Programming with R*. Springer.

Cook, D. and Swayne, D. F., 2007. *Interactive and Dynamic Graphics for Data Analysis*. Springer.

Fuller, W. A., 1987. *Measurement Error Models*. Wiley.

Gentleman, R., 2008. *Bioinformatics with R*. Chapman & Hall/CRC.

Ihaka, R. and Gentleman, R., 1996. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314.

Miller, R. G., 1986. *Beyond ANOVA, Basics of Applied Statistics*. Wiley.

Muenchen, R. A., 2008. *R for SAS and SPSS Users*. Springer.

Murrell, P., 2005. *R Graphics*. Chapman and Hall/CRC.
URL `http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html`

Nicholls, N.; Lavery, B.; Frederiksen, C.; and Drosdowsky, W., 1996. Recent apparent changes in relationships

between the El Niño –, southern oscillation and Australian rainfall and temperature. *Geophysical Research Letters*, 23:3357–3360.

Paradis, E., 2006. *Analysis of Phylogenetics and Evolution with R*. Springer, New York.

R Development Core Team, 2004. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria.
URL `http://www.R-project.org`

Sarkar, D., 2007. *Lattice: Multivariate Data Visualization with R*. Springer, New York.

Schatzkin, A.; Kipnis, V.; Carroll, R.; Midthune, D.; Subar, A.; Bingham, S.; Schoeller, D.; Troiano, R.; and Freedman, L., 2003. A comparison of a food frequency questionnaire with a 24-hour recall for use in an epidemiological cohort study: results from the biomarker-based observing protein and energy nutrition (open) study. *International Journal of Epidemiology*, 32:1054–1062.

Stidd, C. K., 1953. Cube-root-normal precipitation distributions. *Transactions of the American Geophysical Union*, 34:31–35.

Wilkinson, L., 2005. *The Grammar of Graphics*. Springer.

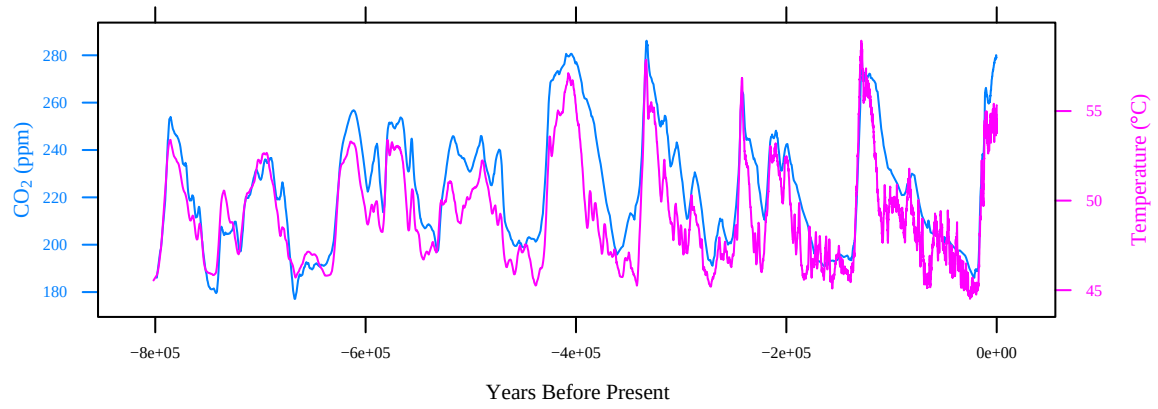

Figure 5.3: This graph overlays EPICA Dome C ice core 800KYr temperature and CO_2 estimates, showing the strong correlation. Both sets of data have been locally smoothed.

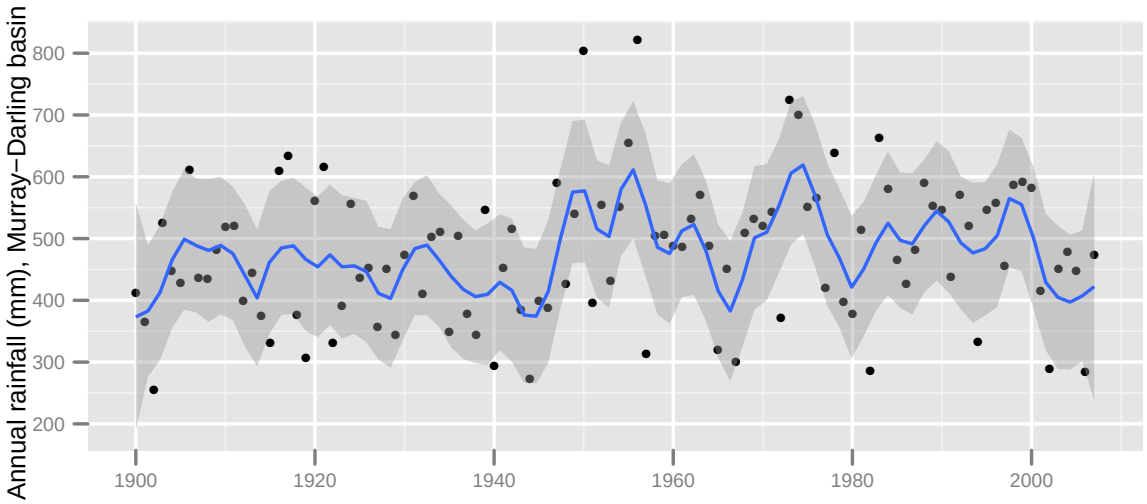

Figure 5.4: Annual rainfall, from 1901 to 2007, for the Murray-Darling basin region of Australia. The curve is fitted using the default loess smoother. The pointwise standard error bands assume that errors about the curve are independent; this is unlikely to be strictly true. To suppress the bands, specify `se=FALSE`.

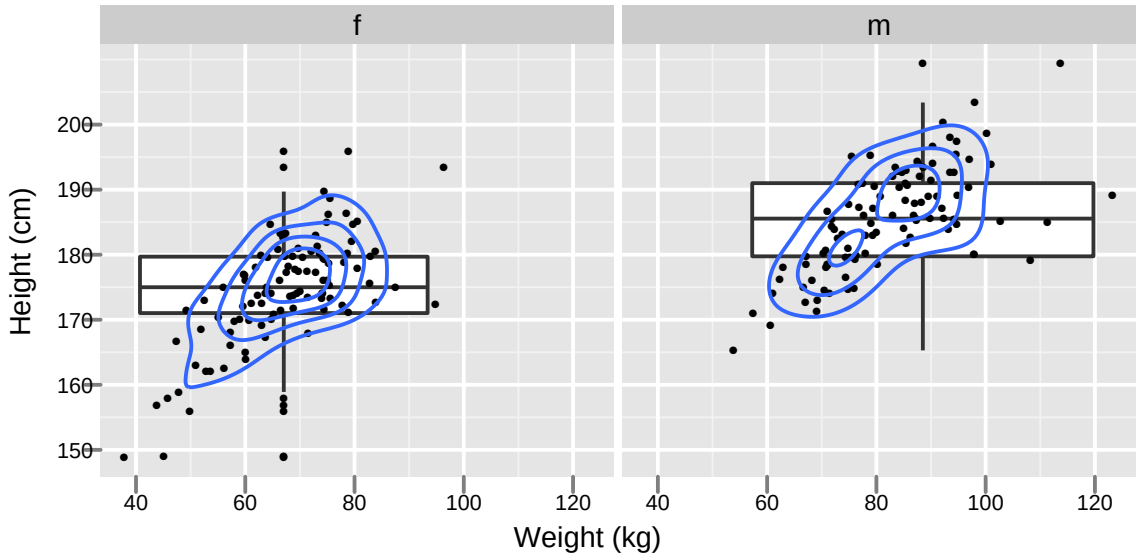

Figure 5.5: Height versus weight, by sex, for Australian athletes in the `ais` data set. Boxplots that show the distributions of heights, and two-dimensional density contours have been added.

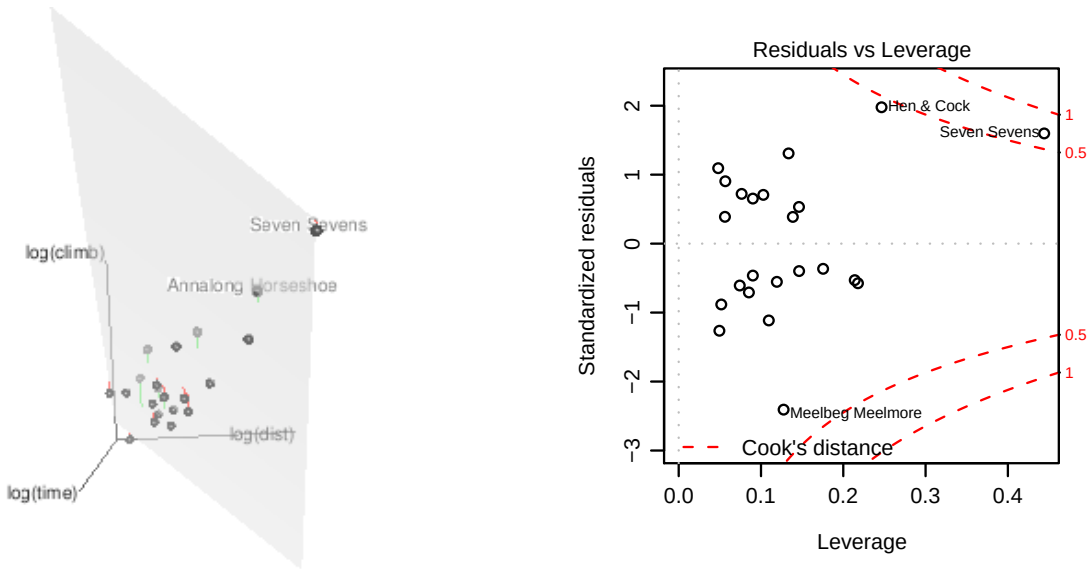

Figure 5.6: The left panel is a snapshot of a three-dimensional dynamic graphic plot. The two points that have the largest *leverage* in the regression of `logtime` on `logdist` and `logclimb` have been labeled. In the right panel, standardized residuals are plotted against leverages. Contours are shown for Cook's distances of 0.5 and 1.0.