8
Regression with Linear Terms and Factors

Linear Models, in the style of `lm()`:

Linear model	Any model that `lm()` will fit is a "linear" model. `lm()` can fit highly non-linear forms of response!
Diagnostic plots	Use `plot()` with the model object as argument, to get a basic set of diagnostic plots.
`termplot()`	If there are no interaction terms, use `termplot()` to visualize the contributions of the different terms.
Factors	In model formulae, factors model qualitative effects.
Model matrices	The model matrix shows how coefficients should be interpreted. (This is an especial issue for factors.)
GLMs	Generalized Linear Models are an extension of linear models, commonly used for analyzing counts.
Modern regession	This can use smoothers – spline and other functions of explanatory variables that adapt to suit the data.

[NB: `lm()` assumes independently & identically distributed (iid) errors, perhaps after applying a weighting function.]

In this chapter, the chief focus will be on the `lm()` (*linear model*) function, discussed earlier in Section 3.2. The `lm()` function is the most widely used of a huge range of model fitting abilities, available in the various R packages.

Linear models are linear in the model parameters, not necessarily in the variables. A linear model can perfectly well fit a combination of *basis* curves.

Thus spline fits are formed as a linear combination from a kitset of curves.

8.1 Linear Models in R – Basic Ideas

Here, we fit a straight line, which is very obviously a linear model! This simple starting point gives little hint of the range of models that can be fitted using R's linear model `lm()` function.

The `lm()` function returns, as well as estimates, standard errors for parameters and for predictions. The standard error and p-value information provided by the `lm()` function assumes that the random term is i.i.d. (independently and identically distributed) normal. The independence assumption can be crucial.

The standard errors assume, also, a single model that is known from the start and on which the analysis is based.[1] If this assumption is incorrect, it can be important to resort to the use of empirical methods for assessing model performance – perhaps some variation of training/test methodology, or the bootstrap.

The symbolic notation[2] that is available in R for describing linear models makes it straightforward to set up quite elaborate and intricate models.

[1] The standard errors become increasingly unrealistic as the number of possible choices of model terms (variables, factors and interactions) increases.

[2] Wilkinson, GN and Rogers, CE, 1973. Symbolic description of models in analysis of variance, *Applied Statistics* 22: 392-399.

Scatterplot with fitted line – an example

The following plots data from the data frame `roller` (as in Figure 8.1) from the *DAAG* package.

```
library(DAAG)
plot(depression ~ weight, data=roller, fg="gray")
```

The formula `depression ~ weight` can be used either as a graphics formula or as a model formula. The following fits a straight line, then adding it to the above plot:

```
plot(depression ~ weight, data=roller, fg="gray")
roller.lm <- lm(depression ~ weight, data=roller)
# For a line through the origin, specify
# depression ~ 0 + weight
abline(roller.lm)
```

Figure 8.2 repeats the plot, now with a fitted line added.

The different explanatory variables in the model are called `terms`. In the above, there is one explicit term only on the right, i.e., `weight`. This is in addition to the intercept, which is included by default.

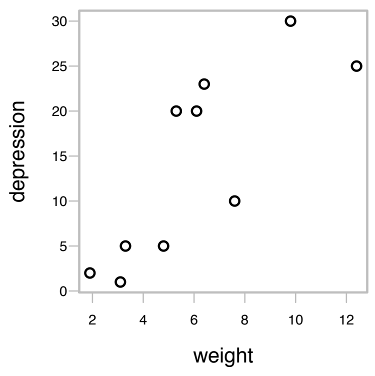


Figure 8.1: Plot of `depression` versus `weight`, using data from the data frame `roller` in the *DAAG* package.

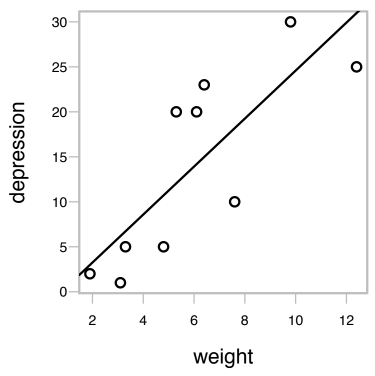


Figure 8.2: This repeats Figure 8.1, now adding a fitted line.

8.1.1 Straight line regression – algebraic details

The standard form of simple straight line model can be written

$$\text{depression} = \alpha + \beta \times \text{ weight } + \text{ noise.}$$

Now write y in place of `depression` and x in place of `weight`, and add subscripts, so that the observations are: (x_1, y_1), (x_2, y_2), ..., (x_n, y_n). Then the model can be written:

$$y_i = \alpha + \beta x_i + \varepsilon_i.$$

The $\alpha + \beta x_i$ term is the "fixed" component of the model, and ε_i is the random noise.

The line is chosen so that the sum of squares of residuals is as small as possible, i.e., the intercept α and the slope β chosen to minimize

$$\sum_{i=1}^{n}(y_i - \alpha - \beta_i x_i)^2$$

The R function `lm()` will provide estimates a of α and b of β. The straight line

$$\widehat{y} = a + bx$$

can then be added to the scatterplot.

Fitted or predicted values, calculated so that they lie on the estimated line, are obtained using the formula:

$$\widehat{y}_1 = a + bx_1,\ \widehat{y}_2 = a + bx_2, \ldots .$$

The residuals, which are the differences between the observed and fitted values, give information about the noise.

$$e_1 = y_1 - \widehat{y}_1,\ e_2 = y_2 - \widehat{y}_2, \ldots . \tag{8.1}$$

8.1.2 Syntax – model, graphics and table formulae:

The syntax for `lm()` models that will be demonstrated here is used, with modification, throughout the modeling functions in R. A very similar syntax can be used for specifying graphs and for certain types of tables.

Model objects

The following code returns a model object to the command line.

```
lm(depression ~ weight, data=roller)
```

```

Call:
lm(formula = depression ~ weight, data = roller)

Coefficients:
(Intercept)       weight
      -2.09         2.67
```

Components of model objects can be accessed directly, as list objects. But it is usually better to use an extractor function. Note in particular `residuals()` (can be abbreviated to `resid()`), `coefficients()` (`coef()`), and `fitted.values()` (`fitted()`). For example:
  `coef(roller.lm)`

When returned to the command line in this way, a printed summary is returned.

Alternatively, the result can be saved as a named object, which is a form of list.

```
roller.lm <- lm(depression ~ weight, data=roller)
```

The names of the list elements are:

```
names(roller.lm)
```

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

8.1.3 Matrix algebra – straight line regression example

In order to write the quantity

$$\sum_{i=1}^{10}(y_i - a - bx_i)^2$$

that is to be minimized in matrix form, set:

$$\mathbf{X} = \begin{pmatrix} 1 & 1.9 \\ 1 & 3.1 \\ 1 & 3.3 \\ 1 & 4.8 \\ 1 & 5.3 \\ 1 & 6.1 \\ 1 & 6.4 \\ 1 & 7.6 \\ 1 & 9.8 \\ 1 & 12.4 \end{pmatrix}; \quad \mathbf{y} = \begin{pmatrix} 2 \\ 1 \\ 5 \\ 5 \\ 20 \\ 20 \\ 23 \\ 10 \\ 30 \\ 25 \end{pmatrix}; \quad \mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{b} = \begin{pmatrix} 2 - (a + 1.9b) \\ 1 - (a + 3.1b) \\ 5 - (a + 3.3b) \\ 5 - (a + 4.8b) \\ 20 - (a + 5.3b) \\ 20 - (a + 6.1b) \\ 23 - (a + 6.4b) \\ 10 - (a + 7.6b) \\ 30 - (a + 9.8b) \\ 25 - (a + 12.4b) \end{pmatrix} \quad \text{where} \quad \mathbf{b} = \begin{pmatrix} a \\ b \end{pmatrix}$$

Here a and b are chosen to minimize the sum of squares of elements of $\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{b}$, i.e., to minimize

$$\mathbf{e}'\mathbf{e} = (\mathbf{y} - \mathbf{X}\mathbf{b})'(\mathbf{y} - \mathbf{X}\mathbf{b})$$

The least squares equations can be solved using matrix arithmetic.

Recap, and Next Steps in Linear Modeling

For this very simple model, the model matrix had two columns only. Omission of the intercept term will give an even simpler model matrix, with just one column.

Regression calculations in which there are several explanatory variables are handled in the obvious way, by adding further columns as necessary to the model matrix. This is however just the start to the rich range of possibilities that model matrices open up.

8.1.4 A note on the least squares methodology

More fundamental than least squares is the maximum likelihood principle. If the "error" terms are independently and identically normally distributed, then least squares and maximum likelihood are equivalent.

The assumptions of independence and identical distribution (iid) are crucial. The role of normality is commonly over-stated.

Least squares will not in general yield maximum likelihood estimates, and the SEs returned by `lm()` or by `predict()` from an `lm` model will be problematic or wrong if:

- Variances are not homogeneous[3];
- Observations are not independent;
- The sampling distributions of parameter estimates are noticeably non-normal.

[3] Weighted least squares is however justified by maximum likelihood if it is known how the variances change with x_i, or if the pattern of change can be inferred with some reasonable confidence.

- Model terms (variables, factors and/or interactions) have been chosen from some wider set of possibilities (the theory assumes a specfic known model).

Simplifying the model, in ways that do not much affect coefficients that remain in the model, may be acceptable.

Normality of the model 'errors' is more than is in practice required. Outliers, and skewness in the distribution, do often mean that the theory cannot be satisfactorily used as a good approxation.

8.2 *Checks — Before and After Fitting a Line*

Consider here a female versus male comparison of record times for Northern Island hill races.

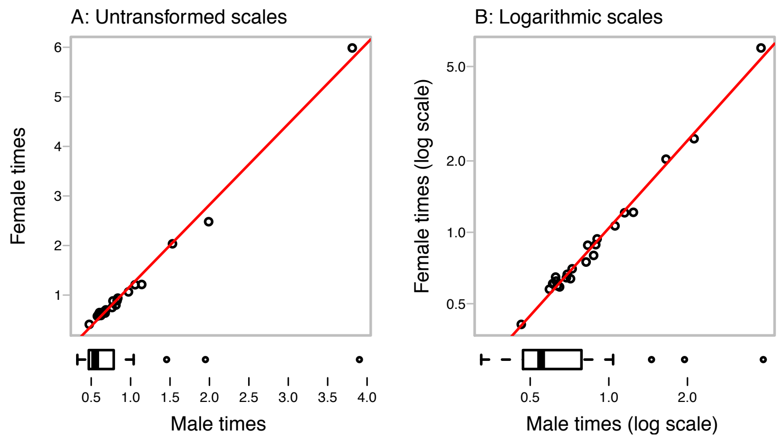


Figure 8.3: Graphs compare female with male record times, for Northern Ireland hill races. Least squares lines are added, and marginal boxplots are shown on the horizontal axis. Panel A has untransformed scales, while Panel B has log transformed scales. For the code, see the script file for this chapter.

Untransformed vs transformed scales: Figure 8.3 shows two alternative views of the data. Least squares line have in each case been added.

In Panel A, a single data point at the top right lies well away from the main body of data. In Panel B, points are more evenly spread out, though still with a tail out to long times.

The following fits a regression line on the untransformed scale:

```
mftime.lm <- lm(timef ~ time, data=nihills)
```

The line appears to fit the data quite reasonably well. Is this an effective way to represent the relationship? An obvious problem is that data values become increasingly sparse as values increase, with one point widely separated from other data. That one data point, widely separated from other points, stands to have a disproportionate effect in determining the fitted line.

The coefficients for the line that is fitted on a logarithmic scale is:

```
mflogtime.lm <- lm(log(timef) ~ log(time),
                   data=nihills)
round(coef(mflogtime.lm), 3)
```

```
(Intercept)   log(time)
      0.267       1.042
```

The coefficient of 1.042 for `log(time)` implies that the relative rate of increase of female times is 4.2% greater than the relative rate of increase of male times.

The use of residuals for checking the fitted line:

In Figure 8.3, departures from the line do not stand out well relative to the line. To make residuals stand out, Figures 8.4A and 8.4B rotate the lines, for the untransformed and transformed data respectively, ~45° clockwise about its mid-point, to the horizontal.

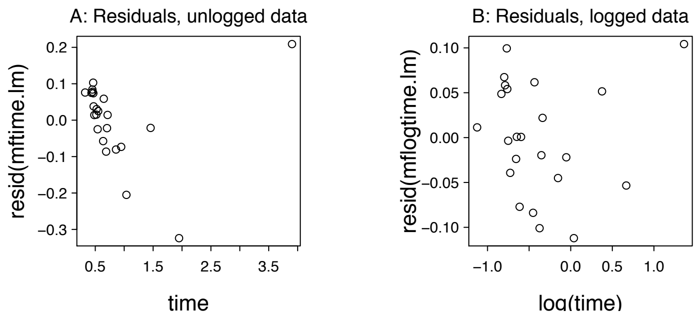


Figure 8.4: In Panel A, residuals from the line for the unlogged data have been plotted against male times. Panel B repeats the same type of plot, now for the regression for the logged data.

Notice that, in Figure 8.4A, all residuals except that for the largest time lie very nearly on a line. The point with the largest fitted value, with the largest male (and female) time, is pulling the line out of whack. There is a mismatch between the data and the model. The picture in Figure 8.4B is much improved, though there still is an issue with the point for the longest time.

Residuals on the vertical scale of Figure 8.4B are on a scale of natural logarithms (logarithms to base *e*). As the range is small (roughly between -0.1 and 0.1), the values can be interpreted as relative differences on the scale of times.

A residual of -0.1 denotes a time that is about 10% (more accurately 9.5%) less than the fitted value on the line. A residual of 0.1 denotes a time that is about 10% (more accurately 10.5%) more than the fitted value.

The common benefits of a logarithmic transformation: Where measurement data have a long tail out to the right, it commonly makes sense to work with logarithms of data values, as in Figure 8.3B. Often, working with data on a logarithmic scale has several useful consequences:

- The skewness is reduced
- The variation at the high end of the range of values is reduced, relative to variation at the low end of the range of values.
- Working on a logarithmic scale is equivalent to working with relative, rather than absolute, change. Thus a change from 10 to

20 is equivalent to a change from 20 to 40, or from 40 to 80. On a logarithmic scale, these are all changes by an amount of $\log(2)$.

- By default, the function `log()`) returns natural logarithms, i.e., logarithms to base e. On this scale, a change of 0.05 is very close to a change of 5%. A change of 0.15 is very roughly a change of 15%. [A decrease of 0.15 is a decrease of $\simeq$ 13.9%, while an increase of 0.15 is a increase of $\simeq$ 16.2%]

Once the model is fitted, checks can and should be made on the extent and manner of differences between observations and fitted model values. Graphical checks are the most effective,[4] at least as a starting point. Mostly, such checks are designed to highlight common types of departure from the model.

[4] Statistics that try to provide an overall evaluation focus too much on a specific form of departure, and do a poor job at indicating whether the departure from assumptions matters.

Figure 8.4B provided a simple form of diagnostic check. This is one of several checks that are desirable when models have been fitted.

8.2.1 *Diagnostics – checks on the fitted model

Section 8.3 demonstrates the use of simulation to help in judging between genuine indications of model departures and features of the plots that may well reflect statistical variation.

For `lm` models, the R system has a standard set of diagnostic plots that users are encouraged to examine. These are a starting point for investigation. Are apparent departures real, or may they be a result of statistical variation? For the intended use of the model output, do apparent departures from model assumptions matter.

For drawing attention to differences between the data and what might be expected given the model, plots that show residuals are in general much more effective than plots that show outcome (y) variable values. Additionally, plots are needed that focus on common specific types of departure from the model.

All four diagnostic plots

Figure 8.5 shows the default diagnostic plots for the regression with the untransformed data:

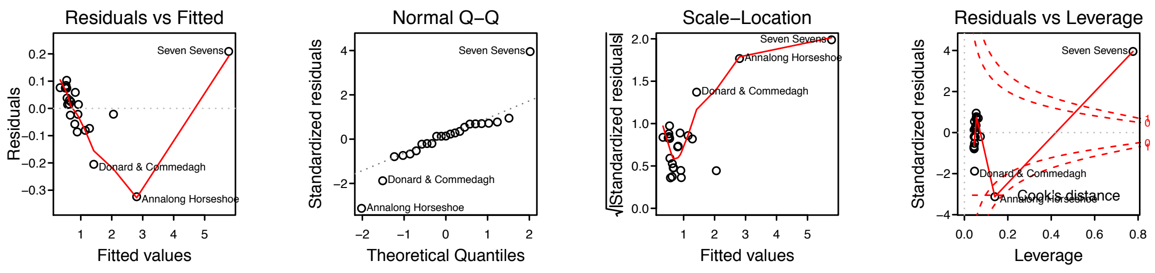


Figure 8.5: Diagnostic plots from the regession of `timef` on `time`.

Simplified code is:

```
mftime.lm <- lm(timef ~ time, data=nihills)
plot(mftime.lm, cex.caption=0.8)
```

The first of these plots has similar information to Figure 8.4A above. A difference is that residuals are now plotted against fitted values. It is immaterial, where there is just one explanatory variable, whether residuals are plotted against fitted values or against x-values – the difference between plotting against $a + bx$ and plotting against x amounts only to a change of labeling on the x-axis.

Figure 8.6 shows the default diagnostic plots for the transformed data. Simplified code is:

```
plot(mflogtime.lm, cex.caption=0.8)
par(opar)
```

Two further plots are available; specify `which=4` or `which=6`, e.g. `plot(mftime.lm, which=4`. These give a different slant on what is shown in the fourth default plot (`which=5`).

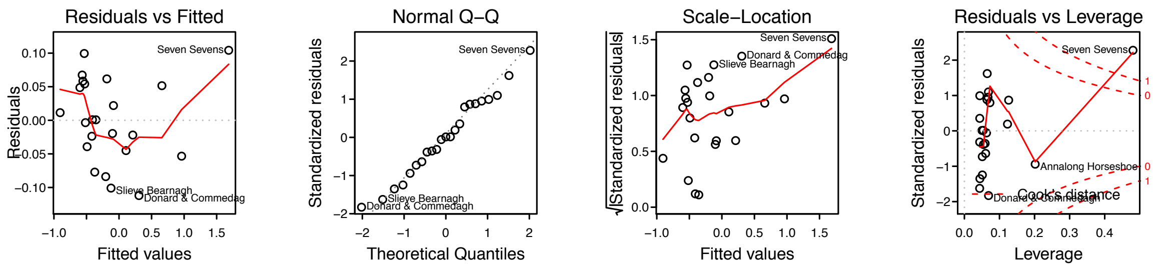


Figure 8.6: Diagnostic plots from the regression of `log(timef)` on `log(time)`.

The point for the largest time still has a very large leverage and, as indicated by its position relative to the Cook's distance contours, a large influence on the fitted regression line. This may, in part or entirely, be a result of a variance that, as suggested by the scale-location plot in Panel 3, tends to increase with increasing fitted value. The normal Q-Q plot (Panel 2) suggests an overall distribution of residuals that is acceptably normal. The large residual associated with the largest time in Panel 1 would look much less out of place if there was an adjustment that allowed for a variance that increases with increasing fitted value.

These differences from the assumed model are small enough that, for many purposes, the line serves as a good summary of the data. The equation is:

```
mflogtime.lm <- lm(log(timef) ~ log(time),
                   data=nihills)
round(coef(mflogtime.lm), 3)
```

```
(Intercept)   log(time)
      0.267       1.042
```

The coefficient that equals 1.042 can be interpreted as a relative rate of increase, for female time relative to male time. Consider a race for which the male record time is 100 minutes. The predicted female time is:

$$\exp(0.267 + 1.042 \log(100/60)) = 2.223\text{h} = 133.4\text{m}.$$

An increase of one minute, or 1%, in the male time, is predicted to lead to an increase of close to $1.042 \times 1\%$ in the female time. The predicted increase is 133.4×1.042m.

Panel 2 — A check for normality: The second panel in Figure 8.5 identifies two large negative residuals and one large positive residual. This seems inconsistent with the assumption that residuals have a normal dsitribution. Again, Figure 8.6 shows an improvement.

Modest departures from normality are not a problem per se. Heterogeneity of variance, and outliers in the data, are likely to be the more serious issues.

To obtain this second plot only, without the others, type:

```
plot(mftime.lm, which=2)
```

Panel 3 — Is the variance constant?: The third panel is designed to check whether variation about the fitted line, as measured by the variance, is constant. For this, there should be no trend up or down in the points. The large upward trend in the third panel of 8.5 has largely disappeared in the third panel of Figure 8.6.

To obtain this third plot only, without the others, type:

```
plot(mftime.lm, which=3)
```

Panel 4 — a check for high leverage points: The fourth panel is designed to check on points with large leverage and/or large influence. In straight line regression, the most extreme leverage points are points that are separated from the main body of points, and are at the high or (less commonly) low end of the range of x-values.

To obtain this fourth plot only, without the others, type:

```
plot(mftime.lm, which=5)
```

The combined effect of leverage and magnitude of residual determines what *influence* a point has. Large leverage translates into large influence, as shown by a large Cook's distance, when the residual is also large. Points that lie within the region marked out by the 0.5 or (especially) the 1.0 contour for Cook's distance have a noticeable influence on the fitted regression equation. Even with the logged data, the point for the largest time ('Seven Sevens') is skewing the regression line noticeably.

For working within the main range of the data values, we might prefer to use the regression line that is obtained when 'Seven Sevens' is omitted. If 'Seven Sevens' is omitted in estimating the regression line, this should be made clear in any report, and the reason explained. The large residual for this point does hint that extrapolation much beyond the upper range of data values is hazardous.

Note that there is no reason to suspect any error in this value. Possibly the point is taking us into a part of the range where the relationship is no longer quite linear. Or this race may be untypical for more reasons than that it is an unusually long race.

The following shows the change when 'Seven Sevens' is omitted:

```
round(coef(mflogtime.lm), 4)
```

```
(Intercept)   log(time)
     0.2667      1.0417
```

```
omitrow <- rownames(nihills)!="Seven Sevens"
update(mflogtime.lm, data=subset(nihills, omitrow))
```

```

Call:
lm(formula = log(timef) ~ log(time), data = subset(nihills, omitrow))

Coefficients:
(Intercept)    log(time)
      0.239        0.991
```

8.2.2 The independence assumption is crucial

A key assumption is that observations are independent. The independence assumption is an assumption about the process that generated the data, about the way that it should be modeled. There is no one standard check that is relevant in all circumstances. Rather the question should be: "Are there aspects of the way that the data were generated that might lead to some form of dependence?" Thus, when data are collected over time, there may be a time series correlation between points that are close together in time.

Another possibility is some kind of clustering in the data, where observations in the same cluster are correlated. In medical applications, it is common to have multiple observations on the one individual. Where clusters may be present, but there is no way to identify them, dependence is hard or impossible to detect.

Issues of dependence can arise in an engineering maintenance context. If the same mechanic services two aircraft engines at the same time using replacement parts from the same batch, this greatly increases the chances that the same mistake will be made on the two engines, or the same faulty part used. Maintenance faults are then not independent. Independence is not the harmless assumption that it is often made out to be!

There may be further checks and tests that should be applied. These may be specific to the particular model.

8.3 *Simulation Based Checks

A good way to check whether indications of departures from the model may be a result or random variation is to compare the plot with similar plots for several sets of simulated data values, as a means of verifying that the mismatch is, if residuals from the line are independent and normally distributed, real. This is the motivation for Figure 8.7.

If the assumption of independent random errors is wrong, patterns in the diagnostic plots that call for an explanation may be more common than suggested by the simulations.

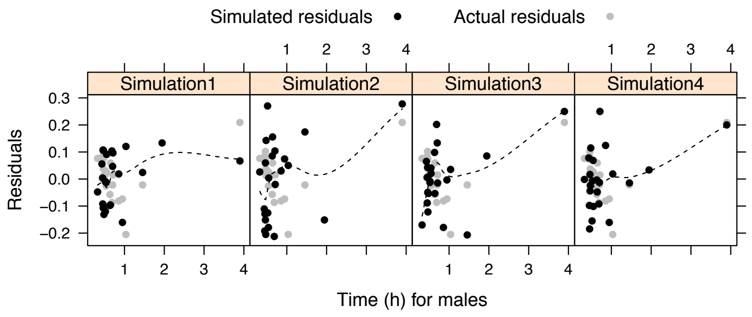


Figure 8.7: The plots are four simulations of residuals, from the model that is fitted to the unlogged data. The coefficients used, and the standard deviation, are from the fitted least squares line.

```
## Code
gph <- plotSimScat(obj=mftime.lm, show="residuals",
                   type=c("p","smooth"),
                   layout=c(4,1))
```

```
update(gph, xlab="Time (h) for males",
      ylab="Residuals")
```

The simulations indicate that there can be a pattern in the smooth curve that is largely due to the one point that is widely separated from other data. On the other hand, the very large residual seen in the actual data is not matched in any of the simulations.

This type of check can be repeated for the other diagnostic plots:

- A check for normality (Panel 2: `which=2`. Type:

  ```
  plotSimDiags(obj=mftime.lm, which=2, layout=c(4,1))
  ```

- Is the variance constant? (Panel 3: `which=3`. Type:

  ```
  plotSimDiags(obj=mftime.lm, which=3, layout=c(4,1))
  ```

- Are there issues of leverage and influence? (Panel 4: `which=5`. Type:

  ```
  plotSimDiags(obj=mftime.lm, which=5, layout=c(4,1))
  ```

Scatterplots that are derived from the simulation process

To see scatterplots that are derived from the simulation process, use the function `plotsimscat()`, from the *DAAG* package. For example, try, working with the untransformed data:

```
gph <- plotSimScat(mftime.lm, layout=c(4,1))
update(gph, xlab="Male record times (h)",
       ylab="Female record times (h)")
```

Observe that the largest simulated value lies consistently above the data value. Other simulated values, for male times of more than around one hour, tend to lie below the actual data. This is much easier to see in the plots of residuals. A scatterplot that shows the actual data values is not a good tool for making these difference visually obvious.

8.4 Key questions for the use of models

Key questions are:

- Modeling and analysis
  - Which model?
  - Do we want to make predictions? Or is the interest in getting parameter estimates that are interpretable?
  - How will model performance be measured?
  - How close can we get to measuring the performance that matters?
- Interpretation

- The task is easier if the aim is prediction, rather than interpretation of model parameters.
- Can model parameters be interpreted in scientifically meaningful ways?
  [This is a minefield, with huge scope for getting it wrong.]

More detailed comments will now follow on some of the issues raised above.

The choice of method: Note the use of the word "method", not algorithm. Algorithms specify a sequence of computational steps. Something more than an algorithm is needed, if results are to have some use that generalizes beyond the specific data used.

There are many different methods. How should the analyst choose between them? What are good ways to assess the performance of one or other algorithm? A credible measure of model performance is needed, evaluated on test data that closely reflects the context in which the model will be applied.

Which are the important variables? Often, the analyst would like to know which data columns (variables, or features) were important for, e.g., a classification. Could some of them be omitted without loss?

The analyst may wish to attach an interpretation to one or more coefficients? Does the risk of heart attack increase with the amount that a person smokes? For a meaningful interpretation of model parameters, it is necessary to be sure that:

- All major variables or factors that affect the outcome have been accounted for.
- Those variables and factors operate, at least to a first order of approximation, independently.

In some cases, a different but equivalent choice of parameters will be more meaningful. For working with the Northern Ireland hillrace data in Subsection 8.6, the parameters `dist` and `climb` clearly do not exercise their effects independently, making their coefficients difficult to interpret. It is better to work with `log(dist)` and `log(dist/climb)`, which are very nearly independent.

See Rosenbaum's *Observational Studies*[5] for comments on approaches that are often useful in the attempt to give meaningful interpretations to coefficients that are derived from observational data.

[5] Rosenbaum, P.R, 2002. *Observational Studies*, 2nd edn. Springer-Verlag.

8.5 Factor Terms – Contrasts

Here, we show how regression models can be adapted to fit terms involving factors.

Another special type of term is one that allows smooth functions of explanatory variables. Again, linear models can be adapted to handle such terms.

Water	A	B	C
(Water only)	(Additive 1)	(Additive 2)	(Additive 3)
1.50	1.50	1.90	1.00
1.90	1.20	1.60	1.20
1.30	1.20	0.80	1.30
1.50	2.10	1.15	0.90
2.40	2.90	0.90	0.70
1.50	1.60	1.60	0.80
Mean = 1.683	0.983	1.75	0.983

Table 8.1: Root weights (`weight`) (g) of tomato plants, grown with water only and grown wth three different treatments. Data are in the data frame `tomato` (DAAG 1.17 or later).

Additive A is `conc nutrient`, B is `3x conc nutrient`, and C is `2-4-D + conc nutrient`. For convenience, we label the factor levels `Water`, `A`, `B`, and `C`, in that order.

```
lev <- c("Water", "A", "B", "C")
tomato[, "trt"] <- factor(rep(lev, rep(6,4)),
                          levels=lev)
```

Taking `Water` as the initial level the effect, in the first analysis that is given below, that it is treated as a reference level.

8.5.1 *Example – tomato root weight*

The model can be fitted either using the function `lm()` or using the function `aov()`. The two functions give different default output. The main part of the calculations is the same whether `lm()` or `aov()` is used.

For model terms that involve factor(s), there are several different ways to set up the relevant columns of the model matrix. The default, for R and for many other computer programs, is to take one of the treatment levels as a baseline or reference, with the effects of other treatment levels then measured from the baseline. Here it makes sense to set `Water` as the baseline.

Table 8.2 shows the model matrix when `Water` is taken as the baseline. Values of the response (`tomato$weight`) have been added in the final column. Also included, in the column headers, is information from the least squares fit.

The following uses `aov()` for the calculations:

```
## Analysis of variance: tomato data (from DAAG)
tomato.aov <- aov(weight ~ trt, data=tomato)
```

Figure 8.8A is a useful summary of what the analysis has achieved. The values are called *partial*s because the overall mean has been subtracted off. Figure 8.8B that is shown alongside shows the effect of working with the logarithms of weights. The scatter about the mean for the treatment still appears much larger for the controls than for other treatments.
Code for Figures 8.8A and 8.8B is:

```
## Panel A: Use weight as outcome variable
```

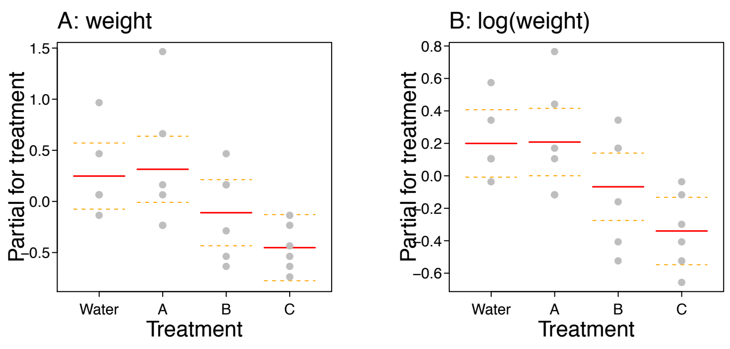


Figure 8.8: Termplot summary of the one-way analysis of variance result — A: for the analysis that uses weights as the outcome variable, and B: for the analysis that works with `log(weight)`

```
tomato.aov <- aov(weight ~ trt, data=tomato)
termplot(tomato.aov, xlab="Treatment",
         ylab="Partial for treatment",
         partial.resid=TRUE, se=TRUE, pch=16)
mtext(side=3, line=0.5, "A: weight", adj=0, cex=1.2)
## Panel B: Use log(weight) as outcome variable
logtomato.aov <- aov(log(weight) ~ trt, data=tomato)
termplot(logtomato.aov, xlab="Treatment",
         ylab="Partial for treatment",
         partial.resid=TRUE, se=TRUE, pch=16)
mtext(side=3, line=0.5, "B: log(weight)", adj=0,
      cex=1.2)
```

Residuals, if required, can be obtained by subtracting the fitted values in Table 8.2 from the observed values (y) in Table 8.1.

Coefficient estimates for the model that uses `weight` as the dependent variable, taken from the output summary from R, are:

```
round(coef(summary.lm(tomato.aov)),3)
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    1.683      0.187   9.019    0.000
trtA           0.067      0.264   0.253    0.803
trtB          -0.358      0.264  -1.358    0.190
trtC          -0.700      0.264  -2.652    0.015
```

The row labeled `(Intercept)` gives the estimate (= 1.683) for the baseline, i.e., `Water`. The remaining coefficients (differences from the baseline) are:

A: weight differs by 0.067.

B: weight differs by −0.358.

C: weight differs by −0.700.

Regression calculations have given us a relatively complicated way to calculate the treatment means! The methodology shows its power to better effect in more complex forms of model, where there is no such simple alternative.

Examination of the model matrix can settle any doubt about how to interpret the coefficient estimates, The first four columns of Table 8.2 comprise the model matrix, given by:

```
model.matrix(tomato.aov)
```

The multipliers determined by least squares calculations are shown above each column. Also shown is the fitted value, which can be calculated either as `fitted(tomato.aov)` or as `predict(tomato.aov)`.

Water: 1.683	A: +0.067	B: −0.358	C: −0.700	Fitted value
1	0	0	0	1.683
1	0	0	0	1.683
. . . .				
1	0	0	0	1.683
1	1	0	0	1.750
1	1	0	0	1.750
. . . .				
1	1	0	0	1.750
1	0	1	0	1.325
1	0	1	0	1.325
. . . .				
1	0	1	0	1.325
1	0	0	1	0.983
1	0	0	1	0.983
. . . .				
1	0	0	1	0.983

Table 8.2: The model matrix for the analysis of variance calculation for the data in Table 8.1 is shown in gray. A fourth column has been added that shows the fitted values. At the head of each column is the multiple, as determined by least squares, that is taken in forming the fitted values.

8.5.2 *Factor terms – different choices of model matrix*

In the language used in the R help pages, different choices of *contrasts* are available, with each different choice leading to a different model matrix and to different regression parameters. The fitted values remain the same, the termplot in Figure fig:tomatotermA is unchanged, and the analysis of variance table is unchanged.

Where there is just one factor, the constant term can be omitted, i.e., it is effectively forced to equal zero. The parameters are then the estimated treatment means. Specify:

```
## Omit constant term from fit;
## force parameters to estimate treatment means
tomatoM.aov <- aov(weight ~ 0 + trt, data=tomato)
```

The first nine rows of the model matrix are:

```
mmat <- model.matrix(tomatoM.aov)
mmat[1:9, ]
```

```
  trtWater trtA trtB trtC
1        1    0    0    0
2        1    0    0    0
3        1    0    0    0
4        1    0    0    0
5        1    0    0    0
```

```
6          1    0    0    0
7          0    1    0    0
8          0    1    0    0
9          0    1    0    0
```

```
## ...    ...     ...     ...
```

Observe that there is now not an initial column of ones. This is fine when there is just one factor, but does not generalize to handle more than one factor and/or factor interaction.

The default (*treatment*) choice of *contrasts* uses the initial factor level as baseline, as we have noted. Different choices of the baseline or reference level lead to different versions of the model matrix. The other common choice, i.e., *sum* contrasts, uses the average of treatment effects as the baseline.

Be sure to choose the contrasts that give the output that will be most helpful for the problem in hand. Or, more than one run of the analysis may be necessary, in order to gain information on all effects that are of interest.

The sum contrasts

Here is the output when the baseline is the average of the treatment effects, i.e., from using the *sum* contrasts:

```
oldoptions <- options(contrasts=c("contr.sum",
                                  "contr.poly"))
tomatoS.aov <- aov(weight ~ trt, data=tomato)
round(coef(summary.lm(tomatoS.aov)),3)
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    1.435      0.093  15.381    0.000
trt1           0.248      0.162   1.534    0.141
trt2           0.315      0.162   1.946    0.066
trt3          -0.110      0.162  -0.683    0.502
```

```
options(oldoptions)  # Restore default contrasts
```

The baseline, labeled `(Intercept)`, is now the treatment mean. This equals 1.435. Remaining coefficients are differences, for Water and for treatment levels A and B, from this mean. The sum of the differences for all three treatments is zero. Thus the difference for C is (rounding up)

$$-(0.2479 + 0.3146 - 0.1104) = -0.4521.$$

Yet other choices of contrasts are possible; see `help(contrasts)`.

The estimates (means) are:

Water: $1.435 + 0.248 = 1.683$.

A: $1.435 + 0.315 = 1.750$.

B: $1.435 - 0.110 = 1.325$.

C: $1.435 - 0.452 = 0.983$.

Interaction terms

The data frame `cuckoos` has the lengths and breadths of cuckoo eggs that were laid in the nexts of one of six different bird species. The following compares a model where the regression line of breadth against length is the same for all species, with a model that fits a different line for each different cuckoo species:

```
cuckoos.lm <- lm(breadth ~ species + length, data=cuckoos)
cuckoosI.lm <- lm(breadth ~ species + length + species:length, data=cuckoos)
print(anova(cuckoos.lm, cuckoosI.lm), digits=3)
```

```
Analysis of Variance Table

Model 1: breadth ~ species + length
Model 2: breadth ~ species + length + species:length
  Res.Df  RSS Df Sum of Sq    F Pr(>F)
1    113 18.4
2    108 17.2  5      1.24 1.56   0.18
```

Here, the model `cuckoos.lm`, where the regression lines are parallel (the same slope for each species), appears adequate.

An alternative way to compare the two models is:

```
anova(cuckoos.lm, cuckoosI.lm, test="Cp")
```

The `Cp` statistic (smaller is better) compares models on the basis of an assessment of their predictive power. Note the use of the argument `test="cp"`, even though this is not a comparison that is based on a significance text.

8.6 *Regression with two explanatory variables*

Data exploration

The dataset `nihills` in the *DAAG* package has record times for Northern Ireland mountain races. First, get a few details of the data:

```
str(nihills)
```

```
'data.frame':   23 obs. of  4 variables:
 $ dist : num  7.5 4.2 5.9 6.8 5 4.8 4.3 3 2.5 12 ...
 $ climb: int  1740 1110 1210 3300 1200 950 1600 1500 1500 5080 ...
 $ time : num  0.858 0.467 0.703 1.039 0.541 ...
 $ timef: num  1.064 0.623 0.887 1.214 0.637 ...
```

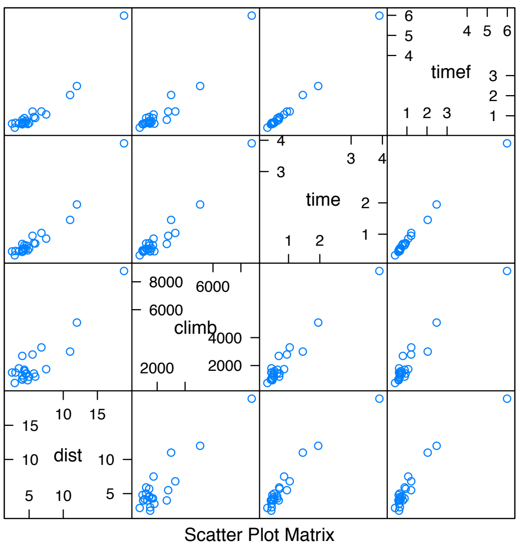
The following Figure 8.9 repeats Figure 3.6 from Chapter 3.3. The left panel shows the unlogged data, while the right panel shows the logged data:

The relationships between explanatory variables, and between the dependent variable and explanatory variables, are closer to linear when logarithmic scales are used. Just as importantly, the point with the largest unlogged values (the same for all variables) will have a leverage, influencing the fitted regression, that is enormously larger than that of other points.

The log transformed data are consistent with a form of parsimony that is well-designed to lead to a simple form of model. We will see that this also leads to more readily interpretable results. Also the distributions for individual variables are more symmetric.

Here again is the code:

```
## Unlogged data
library(lattice)
## Scatterplot matrix; unlogged data
splom(~nihills)
```



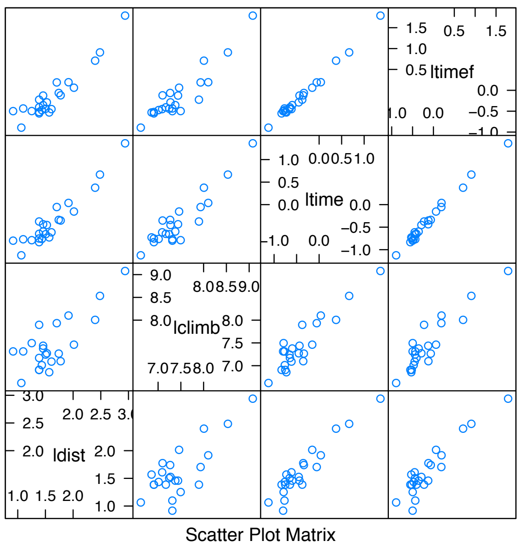


Figure 8.9: Scatterplot matrices for the Northern Ireland mountain racing data. In the right panel, code has been added that shows the correlations. This repeats Figure 3.6 from Chapter 3.3.

The right panel requires a data frame that has the logged data

```
## Logged data
lognihills <- log(nihills)
names(lognihills) <- paste0("l", names(nihills))
## Scatterplot matrix; log scales
splom(~ lognihills)
```

8.6.1 The regression fit

The following regression fit uses logarithmic scales for all variables:

```
lognihills <- log(nihills)
lognam <- paste0("l", names(nihills))
names(lognihills) <- lognam
lognihills.lm <- lm(ltime ~ ldist + lclimb,
                    data=lognihills)
round(coef(lognihills.lm),3)
```

```
(Intercept)         ldist       lclimb
     -4.961         0.681        0.466
```

Thus for constant `climb`, the prediction is that time per mile will decrease with increasing distance. Shorter races with the same climb will involve steeper ascents and descents.

The fitted equation gives predicted times:

$$\begin{aligned} & e^{3.205} \times \text{dist}^{0.686} \times \text{climb}^{0.502} \\ = \; & 24.7e^{3.205} \times \text{dist}^{0.686} \times \text{climb}^{0.502} \end{aligned}$$

A result that is easier to interpret can be obtained by regressing `log(time)` on `log(dist)` and `log(gradient)`, where `gradient` is `dist/climb`.

```
nihills$gradient <- with(nihills, climb/dist)
lognihills <- log(nihills)
lognam <- paste0("l", names(nihills))
names(lognihills) <- lognam
lognigrad.lm <- lm(ltime ~ ldist + lgradient,
                       data=lognihills)
round(coef(lognigrad.lm),3)
```

```
(Intercept)        ldist    lgradient
     -4.961        1.147        0.466
```

Thus, with `gradient` held constant, the prediction is that `time` will increase at the rate of dist$^{1.147}$. This makes good intuitive sense.

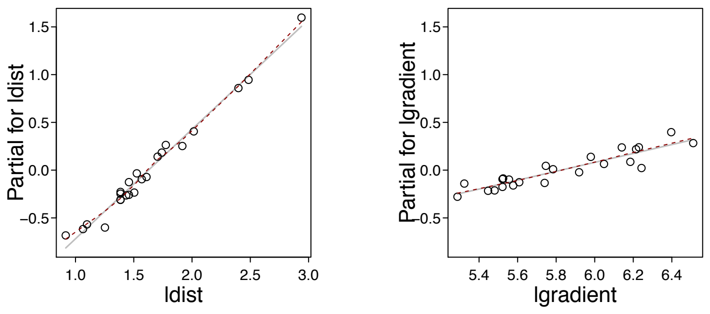


Figure 8.10: The vertical scales in both "term plot" panel show log(time), centered to a mean of zero. The partial residuals in the left panel are for `ldist`, while those in the right panel are for `lgradient`, i.e., `log(climb/dist)`. Smooth curves (dashes) have been passed through the points.

We pause to look more closely at the model that has been fitted. Does `log(time)` really depend linearly on the terms `ldist` and `log(lclimb)`? The function `termplot()` gives a good graphical indication (Figure 8.10).

```
## Plot the terms in the model
termplot(lognigrad.lm, col.term="gray", partial=TRUE,
         col.res="black", smooth=panel.smooth)
```

The vertical scales show changes in `ltime`, about the mean of `ltime`. The lines show the estimated effect of each explanatory variable when the other variable is held at its mean value. The lines, which are the contributions of the individual linear terms ("effects") in this model, are shown in gray so that they do not obtrude unduly. The dashed curves, which are smooth curves that are passed through the residuals, are the primary features of interest.

Notice that, in the plot for `ldist`, the smooth dashed line does not quite track the fitted line; there is a small but noticeable indication of curvature that can be very adequately modeled with a quadratic curve. Note also that until we have modeled effectively the clear trend that seems evident in this plot, there is not much point in worrying about possible outliers.

8.7 Variable Selection – Stepwise and Other

Common variable selection methods include various versions of forward and backward selection, and exhaustive (best subset) selection. These or other variable selection methods invalidate standard model assumptions, which assume a single known model.

There are (at least) three inter-related issues for the use of results from a variable selection process:

The points made here can have highly damaging implications for analyses where it is important to obtain interpretable regression coefficients. In such analyses, changes to the initial model should be limited to simplifications that do not modify the model in any substantial manner. Following the selection process, check coefficients against those from the full model. Any large changes should ring a warning bell.

The implications for prediction are, relatively, much more manageable.

(i) Use of standard theoretically based model fitting procedures, applied to the model that results from the model selection process, will lead to a spuriously small error variance, and to a spuriously large model F-statistic. Coefficient estimates will be inflated, have spuriously small standard errors, and spuriously large t-statistics. (Or to put the point another way, it is inappropriate to refer such statistics to a standard t-distribution.)

(ii) Commonly used stepwise and other model selection processes are likely to over-fit, i.e., the model will not be optimal for prediction on test data that are distinct from the data used to train the model. The selected model may in some instances be inferior, judged by this standard, to a model that uses all candidate explanatory variables. (There are alternative ways to use all variables. Should low order interactions be included? Should some variables be transformed?)

(iii) Coefficients may change, even to changing in sign, depending on what else is included in the model. With a different total set of coefficients, one has a different model, and the coefficients that are common across the two models may be accordingly different. (They will be exactly the same only in the unusual case where "variables" are uncorrelated.) There is a risk that variable selection will remove variables on whose values (individually, or in total effect) other coefficient estimates should be conditioned. This adds uncertainty beyond what arises from sampling variation.

Note that these points apply to pretty much any type of regression modelling, including generalized linear models and classification (discriminant) models.

Where observations are independent, items (i) and (ii) can be addressed, for any given selection process, by splitting the data into training, validation and test sets. Training data select the model, with the validation data used to tune the selection process. Model performance is then checked against the test data.

Somewhat casual approaches to the use of backward (or other) stepwise selection may be a holdover from hand calculator days, or from times when computers grunted somewhat to handle even modest sized calculations. This may be one of the murky dark alleys of statistical practice, where magic incantations and hope too often prevail over hard evidence.

Appropriate forms of variable selection process can however be effective in cases where a few only of the coefficients have predictive power, and the relevant t-statistics are large – too large to be substantially inflated by selection effects.

8.7.1 Use of simulation to check out selection effects:

The function `bestsetNoise()` (*DAAG*) can be used to experiment with the behaviour of various variable selection techniques with data that is purely noise. For example, try:[6]

```
bestsetNoise(m=100, n=40, nvmax=3)
bestsetNoise(m=100, n=40, method="backward",
             nvmax=3)
```

The analyses will typically yield a model that appears to have highly (but spuriously) statistically significant explanatory power, with one or more coefficients that appear (again spuriously) significant at a level of around p=0.01 or less.

[6] See also Section 6.5, pp. 197-198, in: Maindonald, JH and Braun, WJ, 2010. *Data Analysis and Graphics Using R – An Example-Based Approach*, 3rd edition. Cambridge University Press.

The extent of selection effects – a detailed simulation: As above, datasets of random normal data were created, always with 100 observations and with the number of variables varying between 3 and 50. For three variables, there was no selection, while in other cases the "best" three variables were selected, by exhaustive search.

Figure 8.11 plots the p-values for the 3 variables that were selected against the total number of variables. The fitted line estimates the median p-value. Code is:

```
library(DAAG)
library(quantreg)
library(splines)
set.seed(37)   # Use to reproduce graph shown
bsnVaryNvar(m=100, nvar=3:50, nvmax=3, fg="gray")
```

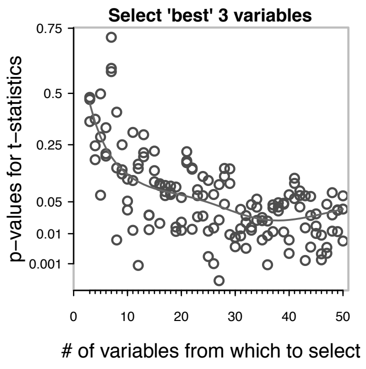


Figure 8.11: p-values, versus number of variables available for selection, when the "best" 3 variables were selected by exhaustive search. The fitted line estimates the median p-value.

When all 3 variables are taken, the p-values are expected to average 0.5. Notice that, for selection of the best 3 variables out of 10, the median p-value has reduced to about 0.1.

Examples from the literature The paper cited in the sidenote[7] gives several examples of published spurious results, all for the use of discriminant methods with microarray data. The same effects can arise from model tuning.

[7] Ambroise, C and McLachlan, GJ, 2001. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences USA*, **99**: 6562-6566.

8.7.2 Variable and model selection – strategies

Several alternative mechanisms are available that can yield reasonable standard errors and other accuracy measures. These include:

a) Fit the model to test data that have played no part in the model selection and tuning process;

b) use cross-validation. The model selection and fitting process must be repeated at each cross-validation fold;

c) repeat the whole analysis, selection and all, with repeated bootstrap samples, using variation between the different sample results to assess the accuracy of one or other statistic;

d) simulate, including all selection and tuning steps, from the fitted model.

For b) and c), there will be somewhat different selections for each different cross-validation fold or bootstrap sample. This is itself instructive.

One possibility, following stepwise or other selection, is that the p-values of one or more coefficients may be so small that they are very unlikely to be an artefact of the selection process. In general, a simulation will be required, in order to be sure.

If however the coefficients are not themselves very meaningful, what is the point?

Model selection more generally:

More generally, the model may be chosen from a wide class of models. Again, model selection biases standard errors to be smaller than indicated by the theory, and coefficients and t-statistics larger. The resulting anti-conservative estimates of standard errors and other statistics should be regarded sceptically.

Use of test data that are separate from data used to develop the model deals with this issue.

A further issue, which use of separate test data does not address, is that none of the models on offer is likely to be strictly correct. Mis-specification of the fixed effects will bias model estimates, at the same time inflating the error variance or variances. Thus it will to an extent work in the opposite direction to selection effects.

8.8 1970 cost for US electricity producers

There is a wide range of possible choices of model terms. Figure 8.12 shows the scatterplot matrices of the variables. Code is:

```
library(car)
library(Ecdat)
data(Electricity)
spm(Electricity, smooth=TRUE, reg.line=NA,
    col=adjustcolor(rep("black",3), alpha.f=0.3))
```

8.8.1 Model fitting strategy

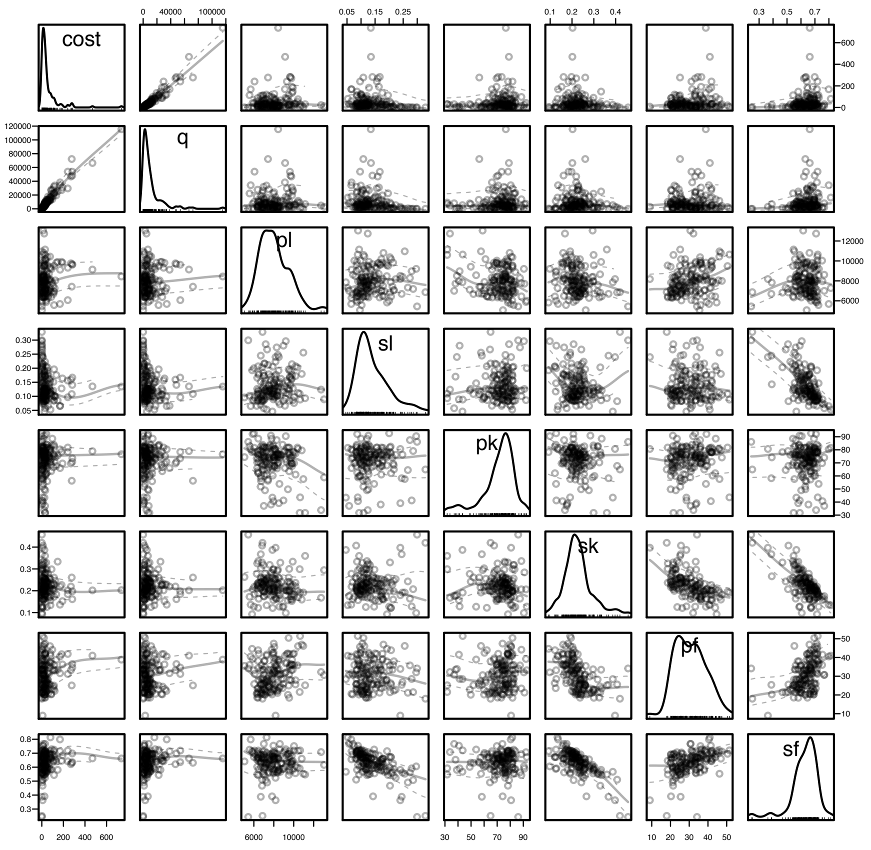
The analysis will start by checking for clearly desirable transformations to variables. Then, for obtaining a model whose whose parameters are as far as possible interpretable, a strategy is:

(i) Start with a model that includes all plausible main effects (variables and factors). Ensure that the model is parameterised in a way that makes parameters of interest as far as possible interpretable (e.g., in Subsection 3.3 above, work with `distance` and `gradient`, not `distance` and `climb`)

(ii) [21pt] Model simplification may be acceptable, if it does not change the parameters of interest to an extent that affects interpretation. The common $p > 0.05$ is too severe; try instead $p = 0.15$ (remove terms with $p > 0.15$) or $p = 0.20$.

Removal of terms with $p > 0.15$ or $p > 0.2$ rather than $p > 0.05$ greatly reduces the risk that estimates of other parameters, and their standard errors, will change in ways that affect the interpretation of model results.

(iii) Variables and/or factors that have no detectable main effect are in general unlikely to show up in interactions. Limiting attention to the main effects that were identified in (ii) above, we then compare a model which has only main effects with a model that inludes all



`cost:`	total cost
`q:`	total output
`pl:`	wage rate
`sl:`	cost share, labor
`pk:`	capital price index
`sk:`	cost share, capital
`pf:`	fuel price
`sf:`	cost share, fuel

Figure 8.12: Scatterplot matrix, for the variables in the data set `Electricity`, in the *Ecdat* package. Density plots are shown in the diagonal.

2-way interactions. Then, using $p \simeq 0.15$ or $p \simeq 0.2$ as the cutoff, remove interaction terms that seem unimportant, and check that there are no changes of consequence in terms that remain.

(iv) In principle, the process may be repeated for order 3 interactions.

(v) Use the function `add1()` to check for individual highly significant terms that should be included. For this purpose, we might set $p = 0.01$ or perhaps $p = 0.001$.

The strategy is to be cautious (hence the cutoff of $p = 0.2$) in removing terms, whether main effects or first order interactions. In a final check whether there is a case for adding in terms that had been omitted, we include a term only if it is highly statistically significant. This limits the scope for selection effects.

Distributions of variables

The distributions of `cost` and `q` are highly skew. The relationship between these two variables is also very close to linear. We might try taking logarithms of both these variables.

Figure 8.13 examines the scatterplot matrix for the logarithms of the variables `cost` and `q`. Code is:

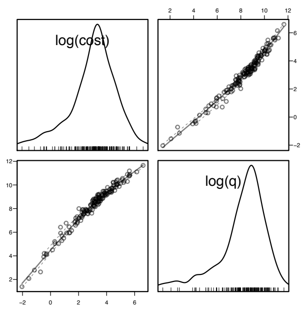


Figure 8.13: Scatterplot matrix for the logarithms of the variables `cost` and `q`. Density plots are shown in the diagonal.

```
varlabs <- c("log(cost)", "log(q)")
spm(log(Electricity[,1:2]), var.labels=varlabs,
    smooth=TRUE, reg.line=NA,
    col=adjustcolor(rep("black",3), alpha.f=0.5))
```

We start with a model that has main effects only:

```
elec.lm <- lm(log(cost) ~ log(q)+pl+sl+pk+sk+pf+sf,
              data=Electricity)
```

Now examine the termplot (Figure 8.14): Code is:

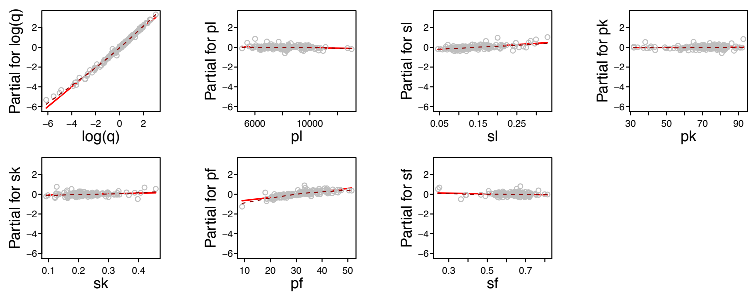


Figure 8.14: Termplot summary for the model that has been fitted to the `Electricity` dataset.

```
termplot(elec.lm, partial=T, smooth=panel.smooth,
         transform.x=TRUE)
```

Notice that in the partial plot for `q`, the dashed curve that is fitted to the residuals closely tracks the fitted effect (linear on a scale of `log(q)`. This confirms the use of `log(q)`, rather than `q`, as explanatory variable.

Now examine the model output:

```
round(coef(summary(elec.lm)),5)
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.41328    0.70720 -7.6545  0.00000
log(q)       0.89250    0.00994 89.8326  0.00000
pl          -0.00002    0.00001 -1.9341  0.05499
sl           2.48020    0.74898  3.3114  0.00116
pk           0.00083    0.00127  0.6562  0.51272
sk           0.62272    0.70837  0.8791  0.38076
pf           0.03042    0.00228 13.3338  0.00000
sf          -0.30965    0.69091 -0.4482  0.65467
```

The *p*-values suggest that `pk`, `sk`, and `sf` can be dropped from the model. Omission of these terms makes only minor differences to the coefficients of terms that remain.

```
elec2.lm <- lm(log(cost) ~ log(q)+pl+sl+pf,
               data=Electricity)
round(coef(summary(elec2.lm)),5)
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.28641    0.13701 -38.585  0.00000
log(q)       0.88901    0.00986  90.167  0.00000
pl          -0.00002    0.00001  -2.072  0.03994
sl           2.69722    0.32464   8.308  0.00000
pf           0.02659    0.00191  13.934  0.00000
```

Now check whether interaction terms should be included:

```
elec2x.lm <- lm(log(cost) ~ (log(q)+pl+sl+pf)^2,
                data=Electricity)
anova(elec2.lm, elec2x.lm)
```

```
Analysis of Variance Table

Model 1: log(cost) ~ log(q) + pl + sl + pf
Model 2: log(cost) ~ (log(q) + pl + sl + pf)^2
  Res.Df  RSS Df Sum of Sq    F  Pr(>F)
1    153 5.00
2    147 2.81  6      2.19 19.1 2.3e-16
```

The case for including first order interactions seems strong. The coefficients and SEs are:

```
round(coef(summary(elec2x.lm)),5)
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.63931    0.67803 -5.3675  0.00000
log(q)       0.71481    0.05455 13.1039  0.00000
pl          -0.00031    0.00007 -4.2508  0.00004
sl           6.06389    1.64592  3.6842  0.00032
pf           0.01592    0.01499  1.0623  0.28985
log(q):pl    0.00003    0.00001  6.2902  0.00000
log(q):sl   -0.67829    0.10229 -6.6308  0.00000
log(q):pf    0.00080    0.00113  0.7133  0.47680
pl:sl        0.00007    0.00018  0.4144  0.67916
pl:pf        0.00000    0.00000  0.1421  0.88722
sl:pf        0.01680    0.03092  0.5432  0.58780
```

This suggests omitting the terms `pf`, and all interactions except `log(q):pl` and `log(q):sl`. We check that omission of these terms makes little difference to the terms that remain:

```
elec2xx.lm <- lm(log(cost) ~ log(q)+pl+sl+pf+
                 log(q):pl+log(q):sl,
                 data=Electricity)
round(coef(summary(elec2xx.lm)),5)
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.12003    0.33312 -12.368        0
log(q)       0.74902    0.03852  19.445        0
pl          -0.00029    0.00004  -7.755        0
sl           7.29642    0.70758  10.312        0
pf           0.02611    0.00145  18.017        0
log(q):pl    0.00003    0.00000   7.657        0
log(q):sl   -0.68969    0.09435  -7.310        0
```

Now check whether there is a strong case for adding in any further individual terms:

```
add1(elec2xx.lm, scope=~(log(q)+pl+sl+pk+sk+pf+sf)^2, test="F")
```

```
Single term additions

Model:
```

```
log(cost) ~ log(q) + pl + sl + pf + log(q):pl + log(q):sl
            Df Sum of Sq  RSS  AIC F value Pr(>F)
<none>                   2.83 -622
pk           1    0.0041 2.82 -620    0.22   0.64
sk           1    0.0329 2.79 -621    1.76   0.19
sf           1    0.0294 2.80 -621    1.58   0.21
log(q):pf    1    0.0040 2.82 -620    0.21   0.64
pl:sl        1    0.0060 2.82 -620    0.32   0.57
pl:pf        1    0.0004 2.83 -620    0.02   0.88
sl:pf        1    0.0016 2.83 -620    0.09   0.77
```

8.9 *An introduction to logistic regression*

The data that will be used for illustration are from the data frame `bronchit` in the *DAAGviz* package. The following loads packages that will be needed:

The dataset `bronchit` may alternatively be found in the *SMIR* package.

```
library(DAAGviz, quietly=TRUE)
library(KernSmooth, quietly=TRUE)
```

Figure 8.15 shows two plots – one of `poll` (pollution level) against `cig` (number of cigarettes per day), and the other of `poll` against `log(poll)`. In each case, points are identified as with or without bronchitis.

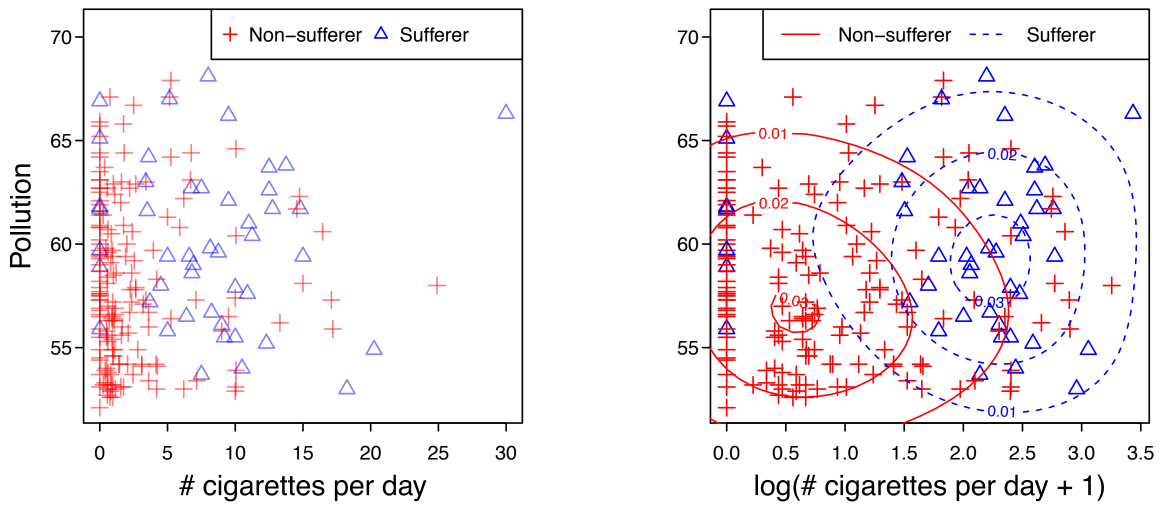


Figure 8.15: Panel A plots `poll` (pollution level) against `cig` (number of cigarettes per day). In panel B, the *x*-scale shows the logarithm of the number of cigarettes per day.

```
## Panel A
colr <- adjustcolor(c("red","blue"), alpha=0.5)
plot(poll ~ cig,
     xlab="# cigarettes per day", ylab="Pollution",
     col=colr[r+1], pch=(3:2)[r+1], data=bronchit,
     ylim=ylim)
legend(x="topright",
       legend=c("Non-sufferer","Sufferer"),
       ncol=2, pch=c(3,2), col=c(2,4), cex=0.8)
```

```
## Panel B
plot(poll ~ log(cig+1), col=c(2,4)[r+1], pch=(3:2)[r+1],
     xlab="log(# cigarettes per day + 1)", ylab="", data=bronchit, ylim=ylim)
xy1 <- with(subset(bronchit, r==0), cbind(x=log(cig+1), y=poll))
xy2 <- with(subset(bronchit, r==1), cbind(x=log(cig+1), y=poll))
est1 <- bkde2D(xy1, bandwidth=c(0.7, 3))
est2 <- bkde2D(xy2, bandwidth=c(0.7, 3))
lev <- pretty(c(est1$fhat, est2$fhat),4)
contour(est1$x1, est1$x2, est1$fhat, levels=lev, add=TRUE, col=2)
contour(est2$x1, est2$x2, est2$fhat, levels=lev, add=TRUE, col=4, lty=2)
legend(x="topright", legend=c("Non-sufferer","Sufferer"), ncol=2, lty=1:2,
```

The logarithmic transformation spreads the points out in the x-direction, in a manner that is much more helpful for prediction than the untransformed values in panel A. The contours for non-sufferer and sufferer in panel B have a similar shape. The separation between non-sufferer and sufferer is stronger in the x-direction than in the y-direction. As one indication of this, the contours at a density of 0.02 overlap slightly in the x-direction, but strongly in the y-direction.

Logistic regression calculations

Figure 8.15 made it clear that the distribution of number of cigarettes had a strong positive skew. Thus, we might fit the model:

```
cig2.glm <- glm(r ~ log(cig+1) + poll, family=binomial, data=bronchit)
summary(cig2.glm)
```

```

Call:
glm(formula = r ~ log(cig + 1) + poll, family = binomial, data = bronchit)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.611  -0.586  -0.362  -0.239   2.653

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -10.7877     2.9885   -3.61  0.00031
log(cig + 1)   1.2882     0.2208    5.83  5.4e-09
poll           0.1306     0.0494    2.64  0.00817

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 221.78  on 211  degrees of freedom
Residual deviance: 168.76  on 209  degrees of freedom
AIC: 174.8

Number of Fisher Scoring iterations: 5
```

Termplots (Figure 8.16) provide a useful summary of the contributions of the covariates. For binary (0/1) data such as here, including the data values provides no visually useful information. Code is:

```
termplot(cig2.glm)
```

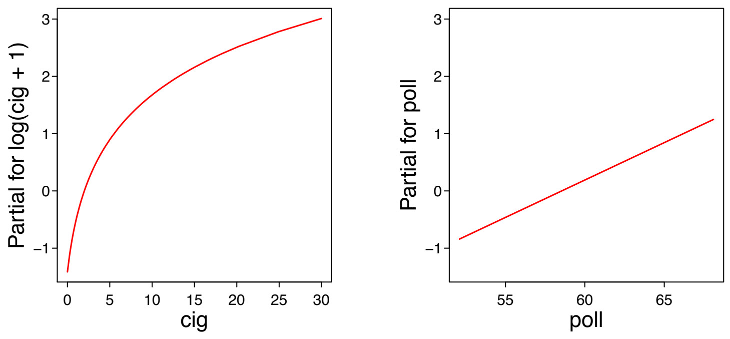


Figure 8.16: The panels show the contributions that the respective terms make to the fitted values (logit of probability of bronchitis), when the other term is held constant.

8.10 Exercises

1. Exercise 3 in Section 2.6.2 involved reading data into a data frame `molclock1`. Plot `AvRate` against `Myr`. Fit a regression line (with intercept, or without intercept?), and add the regression line to the plot. What interpretation can be placed upon the regression slope?

2. Attach the *DAAG* package. Type `help(elasticband)` to see the help page for the data frame `elasticband`. Plot `distance` against `stretch`. Regress `distance` against `stretch` and explain how to interpret the coefficient.

3. Repeat the calculations in Section 8.6, now examining the regression of `time` on `dist` and `climb`. Does this regression adequately model the data. Comment on the results.

4(a) Investigate the pairwise relationships between variables in the data frame `oddbooks` (*DAAG*).

(b) Fit the models

```
volume <- apply(oddbooks[, 1:3], 1, prod)
area <- apply(oddbooks[, 2:3], 1, prod)
lob1.lm <- lm(log(weight) ~ log(volume), data=oddbooks)
lob2.lm <- lm(log(weight) ~ log(thick)+log(area), data=oddbooks)
lob3.lm <- lm(log(weight) ~ log(thick)+log(breadth)+log(height),
              data=oddbooks)
```

Comment on what you find, i.e., comment both on the estimates and on the standard errors.

(c) Can `weight` be completely explained as a function of `volume`? Is there another relevant variable?

5. Repeat the calculations in Section 3.3, now with the dataset `hills2000` (*DAAG*). Do any of the points stand out as outliers? Use `predict()`, with `newdata = hills200`, to obtain predictions from the `hills2000` model for the `nihills` data. Compare with the predictions from the `nihills` model.

9
*A Miscellany of Models & Methods

This chapter is a tour through models and methods that are straightforward to fit using R. Some of these lend themselves to relatively automated use. There is some limited attention to the traps that can catch users who are unwary, or who have ventured too easily into areas that call for some greater level of statistical sophistication than their training and experience has given them.

In each case, comments with be introductory and brief. Firstly, there are brief comments on the fitting of smooth curves. The second and third topics highlight specific types of departure from the iid (independently and identically distributed) assumption.

9.1 Regression with Fitted Smooth Curves

Load the *DAAG* package:

```
library(DAAG)
```

Commentary on Smoothing Methods

Two types of methods will be described – those where the user controls the choice of smoothing parameter, and statistical learning type methods where the amount of smoothing is chosen automatically:

- The first class of methods rely on the user to make a suitable choice of a parameter that controls the smoothness. The default choice is often a good first approximation. Note here:
  - Smoothing using a "locally weighted regression smoother". Functions that use this approach include `lowess()`, `loess()`, `loess.smooth()`, and `scatter.smooth()`.
  - Use of a regression spline basis in a linear model. Here the smoothness is usually controlled by the choice of number of spline basis terms.

Smoothness is controlled by the width of the smoothing window. The default is `f=2/3` for `lowess()`, or `span=0.75` for `loess()`. For other functions that rely on this methodology, check the relevant help page.

- A second class of methods use a "statistical learning" approach in which the amount of smoothing is chosen automatically. The approach of the *mgcv* package extends and adapts the regression spline approach.[1] The methodology generalizes to handle more general types of outcome variables, including proportions and counts. These extensions will not be further discussed here.

[1] Strong assumptions are required, notably that observations are independent. Normality assumptions are, often, less critical.

9.1.1 Locally weighted scatterplot smoothers

Locally weighted scatterplot smoothers pass a window across the data, centering the window in turn at each of a number points that are equally spaced through the data. The smooth at an x-value where the window has been centred is the predicted value from a line (or sometimes a quadratic or other curve) that is fitted to values that lie within the window. A weighted fit is used, so that neighbouring points get greater weight than points out towards the edge of the window.

Figure 9.1 shows a smooth that has been fitted using the `lowess` (locally weighted scatterplot smoothing) methodology. The default choice of with of smoothing window (a fraction $f = \frac{2}{3}$ of the total range of x) gives a result that, for these data, looks about right. The curve does however trend slightly upwards at the upper end of its range. A monotonic response might seem more appropriate.

The code used to plot the graph is:

```
## Plot points
plot(ohms ~ juice, data=fruitohms, fg="gray")
## Add smooth curve, using default
## smoothing window
with(fruitohms,
     lines(lowess(ohms ~ juice), col="gray", lwd=2))
```

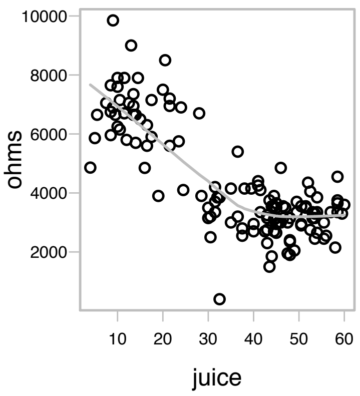


Figure 9.1: Resistance in ohms is plotted against apparent juice content. A smooth curve (in gray) has been added, using the `lowess` smoother. The width of the smoothing window was the default fraction $f = \frac{2}{3}$ of the range of values of the x-variable.

A more sophisticated approach uses the `gam()` function in the *mgcv* package. This allows automatic determination of the amount of smoothing, providing the assumption of independent residuals from the curve is reasonable. We now demonstrate the use of a GAM model for a two-dimensional smooth.

9.1.2 Contours from 2-dimensional Smooths

Data are the amplitudes of responses to a visual stimulus, for each of 20 individuals, at different regions of the left eye. We use the function `gam()` to create smooth surfaces, for males and females separately. Figure 9.2 then uses the function `vis.gam()` to plot heatmaps that show the contours:

The GAM fit will as far as possible use the smooth surface to account for the pattern of variation across the eye, with residuals from the surface treated as random normal noise.

```
## Code
library(DAAGviz)
library(mgcv)
eyeAmpM.gam <- gam(amp ~ s(x,y), data=subset(eyeAmp, Sex=="m"))
eyeAmpF.gam <- gam(amp ~ s(x,y), data=subset(eyeAmp, Sex=="f"))
```

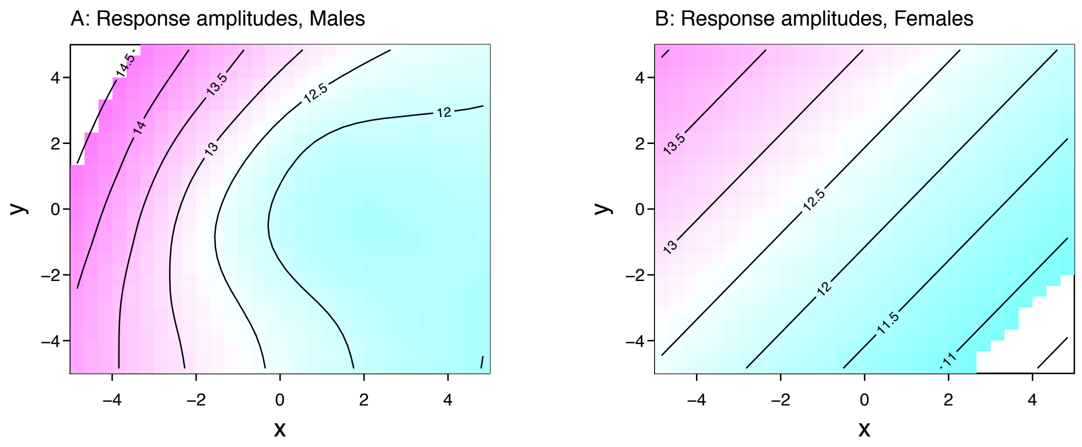


Figure 9.2: Estimated contours of left eye responses to visual stimulae,

```
lims <- range(c(predict(eyeAmpF.gam), predict(eyeAmpM.gam)))
vis.gam(eyeAmpM.gam, plot.type='contour', color="cm", zlim=lims, main="")
mtext(side=3, line=0.5, adj=0, "A: Response amplitudes, Males")
vis.gam(eyeAmpF.gam, plot.type='contour', color="cm", zlim=lims, main="")
mtext(side=3, line=0.5, adj=0, "B: Response amplitudes, Females")
```

9.2 *Hierarchical Multi-level Models*

Models with Non-iid Errors – Multi-level models:

Error Term	Errors do not have to be (and often are not) iid
Multi-level models	Multi-level models are a (relatively) simple type of non-iid model, implemented using `lme()` (*nlme*) or `lmer()` (*lme4* package). Such models allow different errors of prediction, depending on the intended prediction.

Figure 9.3 shows corn yield data from the Caribbean island of Antigua, as in the second column ("Yields") of Table 9.1. Each value is for one package of land. The code for the figure is:

```
# ant111b is in DAAG
Site <- with(ant111b, reorder(site, harvwt,
                              FUN=mean))
stripplot(Site ~ harvwt, data=ant111b, fg="gray",
          scales=list(tck=0.5),
          xlab="Harvest weight of corn")
```

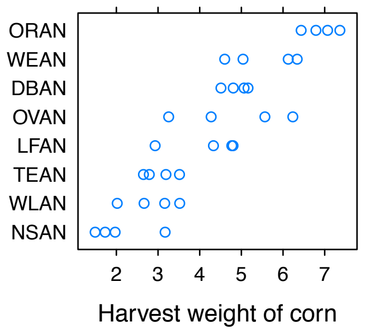


Figure 9.3: Yields from 4 packages of land on each of eight sites on the Caribbean island of Antigua. Data are a summarized version of a subset of data given in Andrews and Herzberg 1985, pp.̃339-353.

Depending on the use that will be made of the results, it may be essential to correctly model the structure of the random part of the model. In comparing yields from different packages of land, there are two sorts of comparison. Packages on the same location should be relatively similar, while packages on different locations should

Location	Yields	Location effect		Residuals from location mean
DBAN	5.16, 4.8, 5.07, 4.51	(4.29)	+0.59	0.28, −0.08, 0.18, −0.38
LFAN	2.93, 4.77, 4.33, 4.8		−0.08	−1.28, 0.56, 0.12, 0.59
NSAN	1.73, 3.17, 1.49, 1.97		−2.2	−0.36, 1.08, −0.6, −0.12
ORAN	6.79, 7.37, 6.44, 7.07		+2.62	−0.13, 0.45, −0.48, 0.15
OVAN	3.25, 4.28, 5.56, 6.24		+0.54	−1.58, −0.56, 0.73, 1.4
TEAN	2.65, 3.19, 2.79, 3.51		−1.26	−0.39, 0.15, −0.25, 0.48
WEAN	5.04, 4.6, 6.34, 6.12		+1.23	−0.49, −0.93, 0.81, 0.6
WLAN	2.02, 2.66, 3.16, 3.52		−1.45	−0.82, −0.18, 0.32, 0.68

Table 9.1: The leftmost column has harvest weights (`harvwt`), for the packages of land in each location, for the Antiguan corn data. Each of these harvest weights can be expressed as the sum of the overall mean (= 4.29), location effect (third column), and residual from the location effect (final column).

be relatively more different, as Figure 9.3 suggests. A prediction for a new package at one of the existing locations is likely to be more accurate than a prediction for a totally new location.

Multi-level models are able to account for such differences in predictive accuracy. For the Antiguan corn yield data, it is necessary to account both for variation within sites and for variation between sites. The R packages `nlme` and `lme4` are both able to handle such data.

Because of the balance the corn yield data, an analysis of variance that specifies a formal `Error` term is an alternative to the fitting of a multi-level model.

9.3 *Regular Time Series in R*

Models with Non-iid Errors – Time Series:

Time sequential	Points that are close together in time commonly show a (usually, +ve) correlation. R's `acf()` and `arima()` functions are powerful tools for use with time series.

Any process that evolves in time is likely to have a sequential correlation structure. The value at the current time is likely to be correlated with the value at the previous time, and perhaps with values several time points back. The discussion that follows will explore implications for data analysis.

9.3.1 *Example – the Lake Erie data*

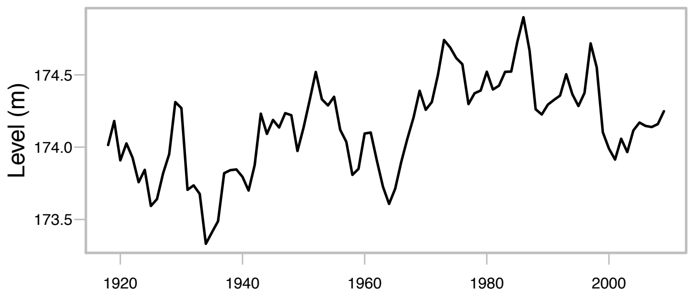


Figure 9.4: Lake Erie levels (m).

```
Erie <- greatLakes[,"Erie"]
plot(Erie, xlab="", fg="gray",
     ylab="Level (m)")
```

The data series `Erie`, giving levels of Lake Erie from 1918 to 2009, will be used as an example from which to start the discussion.[2] The series is available in the *DAAG* package, as the column `Erie` in the multivariate time series object `greatLakes`.

Figure 9.4 shows a plot of the series.

[2] Data are from `http://www.lre.usace.army.mil/greatlakes/hh/greatlakeswaterlevels/historicdata/greatlakeshydrographs/`

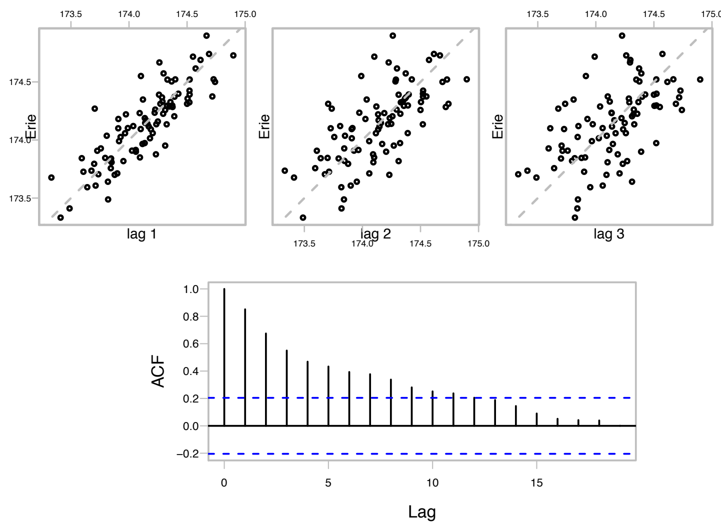


Figure 9.5: Panel A plots Lake Erie levels vs levels at lags 1, 2 and 3 respectively. Panel B shows a consistent pattern of decreasing autocorrelation at successive lags.

```
## Panel A
lag.plot(Erie, lags=3,
         do.lines=FALSE,
         layout=c(1,3), fg="gray",
```

```
## Panel B
acf(Erie, main="", fg="gray")
```

The plots in Figure 9.5 are a good starting point for investigation of the correlation structure. Panel A shows lag plots, up to a lag of 3. Panel B shows estimates of the successive correlations, in this context are called autocorrelations.

There is a strong correlation at lag 1, a strong but weaker correlation at lag 2, and a noticeable correlation at lag 3. Such a correlation pattern is typical of an autoregressive process where most of the sequential dependence can be explained as a flow-on effect from a dependence at lag 1.

If possible, the analyst will want to find covariates that largely or partly explain that dependence. At best, such covariates will commonly explain part only of the dependence, and there will remain dependence that requires to be modeled.

In an autoregressive time series, an independent error component, or "innovation" is associated with each time point. For an order p autoregressive time series, the error for any time point is obtained by taking the innovation for that time point, and adding a linear combination of the innovations at the p previous time points. (For the present time series, initial indications are that $p = 1$ might capture most of the correlation structure.)

An autoregressive model is a special case of an Autoregressive Moving Average (ARMA) model.

9.3.2 *Patterns that are repeatable*

What sorts of patterns may then be repeatable? Indications that a pattern may be repeatable include:

- A straight line trend is a good starting point for some limited extrapolation. But think: Is it plausible that the trend will continue more than a short distance into the future?
- There may be a clear pattern of seasonal change, e.g., with seasons of the year or (as happens with airborne pollution) with days of the week. If yearly seasonal changes persist over different years, or weekly day-of-the-week changes persist over different weeks, these effects can perhaps be extrapolated with some reasonable confidence.
- There is a regression relationship that seems likely to explain future as well as current data.

Smoothing terms can be fitted to the pattern apparent in serially correlated data, leaving *errors* that are pretty much uncorrelated. Such a pattern is in general, however, unrepeatable. It gives little clue of what may happen the future. A re-run of the process (a new *realization*) will produce a different series, albeit one that shows the same general tendency to move up and down.

An ideal would be to find a covariate or covariates than can largely explain the year to year changes. For this series, this does not seem a possibility. In the absence of identifiable direct cause for the year to year changes, a reasonable recourse is to look for a correlation structure that largely accounts for the pattern of the year to year change.

Smooth, with automatic choice of smoothing parameter

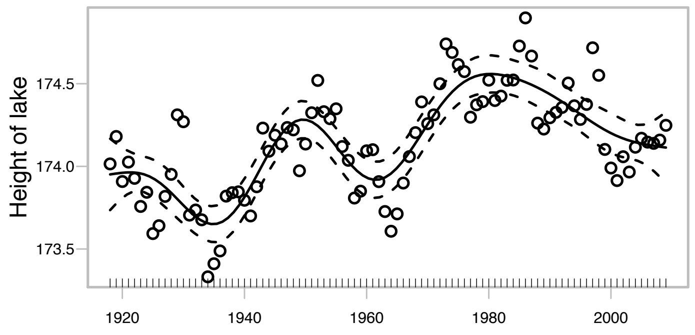


Figure 9.6: GAM smoothing term, fitted to the Lake Erie Data. Most of the autocorrelation structure has been removed, leaving residuals that are very nearly independent.

```
## Code
library(mgcv)
df <-  data.frame(
  height=as.vector(Erie),
  year=time(Erie))
obj <- gam(height ~ s(year),
           data=df)
plot(obj, fg="gray",
     shift=mean(df$height),
     residuals=TRUE, pch=1,
     xlab="",
     ylab="Height of lake")
```

While smoothing methods that asssume independent errors can be used, as in Figure 9.6, to fit a curve to such data, the curve will not be repeatable. Figure 9.6 does not separate systematic effects from effects due to processes that evolve in time. Figure 9.6 uses the abilities of the *mgcv* package, assuming independently and identically distributed data (hence, no serial correlation!) to make an automatic choice of the smoothing parameter. As the curve is conditional on a particular realization of the process that generated it, its usefulness is limited.

The pointwise confidence limits are similarly conditioned, relevant perhaps for interpolalation given this particular realization. All that is repeatable, given another realization, is the process that generated the curve, not the curve itself.

9.3.3 *Fitting and use of an autoregressive model*

There are several different types of time series models that may be used to model the correlatios structure, allowing realistic estimates of the lake level a short time ahead, with realistic confidence bounds around those estimates. For the Lake Erie data, an autoregressive correlation structure does a good job of accounting for the pattern of change around a mean that stays constant.

Figure 9.5 suggested that a correlation between each year and the previous year accounted for the main part of the autocorrelation structure in Figure 9.4. An AR1 model (autoregressive with a correlation at lag 1 only), which we now fit, formalizes this.

```
ar(Erie, order.max=1)
```

```

Call:
ar(x = Erie, order.max = 1)

Coefficients:
    1
0.851

Order selected 1  sigma^2 estimated as  0.0291
```

The one coefficient that is now given is the lag 1 correlation, equalling 0.851.

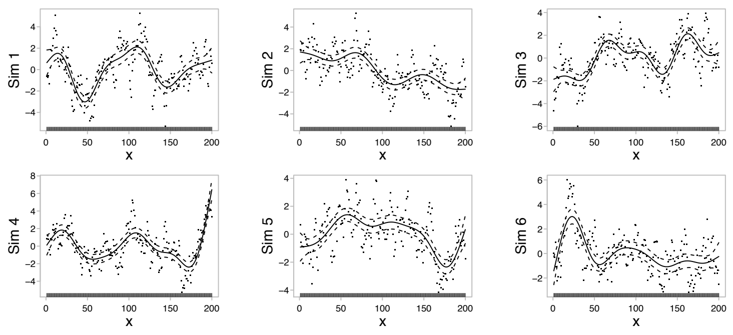


Figure 9.7: The plots are from repeated simulations of an AR1 process with a lag 1 correlation of 0.85. Smooth curves, assuming independent errors, have been fitted.

```
for (i in 1:6){
ysim <-
  arima.sim(list(ar=0.85),
            n=200)
df <- data.frame(x=1:200,
                 y=ysim)
df.gam <- gam(y ~ s(x),
              data=df)
plot(df.gam, fg="gray",
     ylab=paste("Sim", i),
     residuals=TRUE)
}
```

Figure 9.7 then investigates how repeated simulations of this process, with a lag 1 correlation of 0.0.85, compare with Figure 9.4. This illustrates the point that a GAM smooth will extract, from an autoregressive process with mean 0, a pattern that is not repeatable when the process is re-run.

The curves are different on each occasion. For generalization beyond the particular realization that generated them, they serve no useful purpose.

Once an autoregressive model has been fitted, the function `forecast()` in the *forecast* package can be used to predict future levels, albeit with very wide confidence bounds. For this, it is necessary to refit the model using the function `arima()`. An arima model with order (1,0,0) is an autoregressive model with order 1.

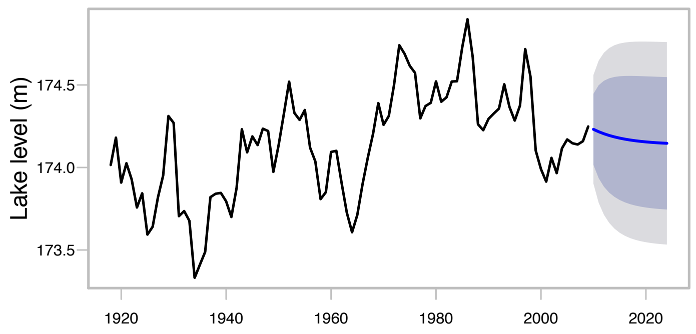


Figure 9.8: Predictions, 15 years into the future, of lake levels (m). The shaded areas give 80% and 95% confidence bounds.

```
erie.ar <- arima(Erie,
            order=c(1,0,0))
library(forecast)
fc <- forecast(erie.ar,
               h=15)
plot(fc, main="", fg="gray",
     ylab="Lake level (m)")
  # 15 time points ahead
```

This brief excursion into a simple form of time series model is intended only to indicate the limitations of automatic smooths. and to give a sense of the broad style of time series modeling. The list of references at the end of the chapter has details of several books on time series.

9.3.4 Regression with time series errors

Figure 9.9 fits annual rainfall, in the Murray-Darling basin of Australia, as a sum of smooth functions of `Year` and `SOI`. Figure 3.9 shows the estimated contributions of the two model terms.

```
## Code
mdbRain.gam <- gam(mdbRain ~ s(Year) + s(SOI),
                   data=bomregions)
plot(mdbRain.gam, residuals=TRUE, se=2, fg="gray",
     pch=1, select=1, cex=1.35, ylab="Partial, Year")
mtext(side=3, line=0.75, "A: Effect of Year", adj=0)
plot(mdbRain.gam, residuals=TRUE, se=2, fg="gray",
     pch=1, select=2, cex=1.35, ylab="Partial, SOI")
mtext(side=3, line=0.75, "B: Effect of SOI", adj=0)
```

The left panel indicates a consistent pattern of increase of rainfall with succeeding years, given an adjustment for the effect of SOI. Errors from the fitted model are consistent with the independent errors assumption. The model has then identified a pattern of increase of rainfall with time, given SOI, that does seem real. It is necessary to warn against reliance on extrapolation more than a few time points into the future. While the result is consistent with expected effects from global warming, those effects are known to play out very differently in different parts of the globe.

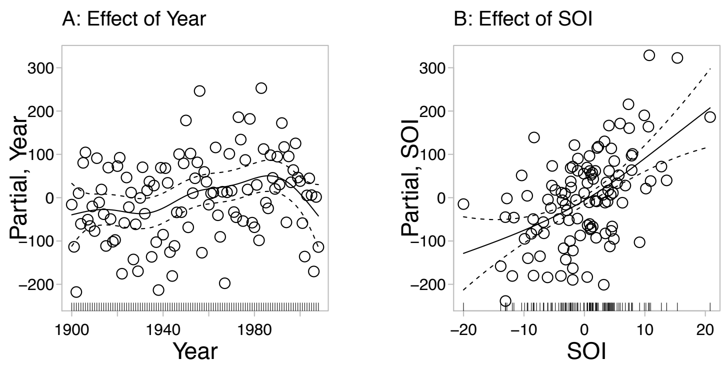


Figure 9.9: Estimated contributions of model terms to `mdbRain`, in a GAM model that adds smooth terms in `Year` and `Rain`. The dashed curves show pointwise 2-SE limits, for the fitted curve.

Investigation of the residual error structure

Sequential correlation structures are often effective, with data collected over time, for use in modeling departure from iid errors. Where there is such structure structure in the data, the methodology will if possible use a smooth curve to account for it.

The residuals can be checked to determine whether the fitted curve has removed most of the correlation structure in the data. Figure 9.10 shows the autocorrelation function of the residuals, followed by autocorrelation functions for several series of independent random normal numbers. Apart from the weakly attested correlation at a lag of 12 years, which is a commonplace of weather data, the pattern of sequential correlation is not much different from what can be expected in a sequence of independent random normal numbers.

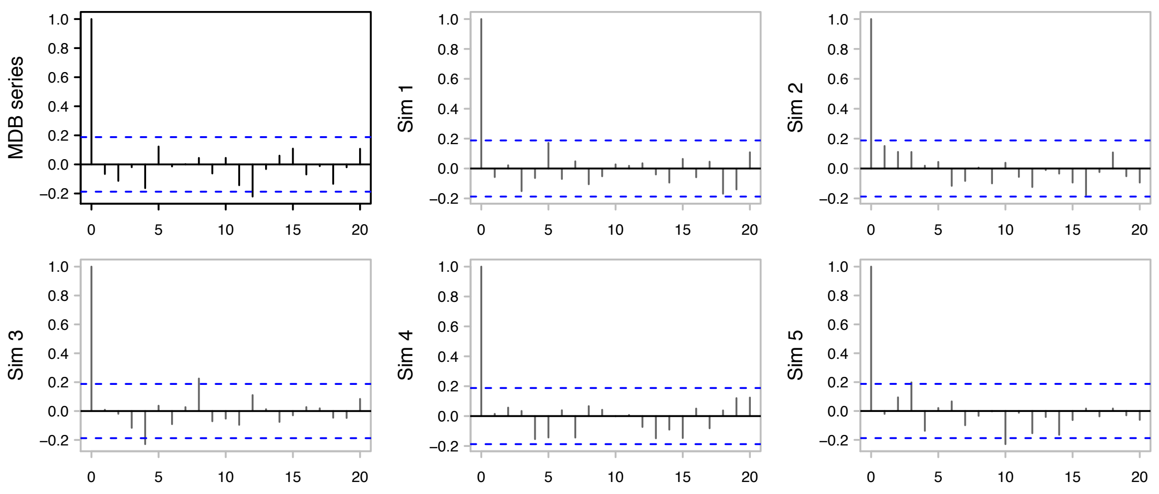


Figure 9.10: The top left panel shows the autocorrelations of the residuals from the model `mdbRain.gam`. The five remaining panels are the equivalent plots for sequences of independent random normal numbers.

Code is:

```
mdbRain.gam <- gam(mdbRain ~ s(Year) + s(SOI),
                   data=bomregions)
n <-  dim(bomregions)[1]
acf(resid(mdbRain.gam), ylab="MDB series")
```

```
for(i in 1:5)acf(rnorm(n), ylab=paste("Sim",i),
                 fg="gray", col="gray40")
```

9.3.5 *Box-Jenkins ARIMA Time Series Modeling

From the perspective of the Box-Jenkins ARIMA (Autoregressive Integrated Moving Average) approach to time series models, autoregressive models are a special case. Many standard types of time series can be modeled very satisfactorily as ARIMA processes.

Models that are closely analagous to ARIMA models had been used earlier in control theory. ARIMA models are feedback systems!

Exercise

The simulations in Figure 9.7 show a pattern of variation that seems not too different from that in the actual series. Modeling of the process as an ARMA or ARIMA process (i.e., allow for a moving average term) may do even better. Use the `auto.arima()` function in the *forecast* package to fit an ARIMA process:

9.3.6 Count Data with Poisson Errors

Data is for aircraft accidents, from the website `http://www.planecrashinfo.com/`. The 1920 file has accidents starting from 1908. The full data are in the dataset `gamclass::airAccs`. Such issues as there are with sequential correlation can be ameliorated by working with weekly, rather than daily, counts.

Data are a time series. Serious accidents are however sufficently uncommon that occasions where events occur together, or where one event changes the probability of the next event, seem likely to be uncommon.

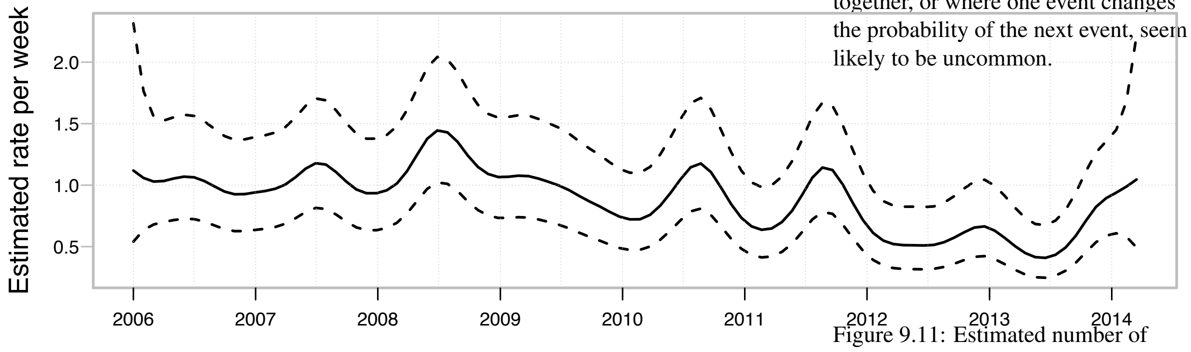


Figure 9.11: Estimated number of events (aircraft crashes) per week, versus time. The yearly tick marks are for January 1 of the stated year.

Figure 9.11 shows a fitted smooth curve, with pointwise confidence bounds, from a GAM smoothing model that was fitted to the weekly counts.

The function `gamclass::eventCounts()` was used to create weekly counts of accidents from January 1, 2006:

See Section 4.3.9 for further details on the function `eventCounts()`.

```
## Code
airAccs <- gamclass::airAccs
fromDate <- as.Date("2006-01-01")
dfWeek06 <- gamclass::eventCounts(airAccs, dateCol="Date",
                                  from=fromDate,
                                by="1 week", prefix="num")
dfWeek06$day <- julian(dfWeek06$Date, origin=fromDate)
```

Code for Figure 9.11 is then.

```
## Code
library(mgcv)
year <- seq(from=fromDate, to=max(dfWeek06$Date), by="1 year")
at6 <- julian(seq(from=fromDate, to=max(dfWeek06$Date), by="6 months"), origin=fromDate)
atyear <- julian(year, origin=fromDate)
dfWeek06.gam <- gam(num~s(day, k=200), data=dfWeek06, family=quasipoisson)
avWk <- mean(predict(dfWeek06.gam))
plot(dfWeek06.gam, xaxt="n", shift=avWk, trans=exp, rug=FALSE,
     xlab="", ylab="Estimated rate per week", fg="gray")
axis(1, at=atyear, labels=format(year, "%Y"), lwd=0, lwd.ticks=1)
abline(h=0.5+(1:4)*0.5, v=at6, col="gray", lty=3, lwd=0.5)
```

The argument '`k`' to the function `s()` that sets up the smooth controls the temporal resolution. A large `k` allows, if the data seem to justify it, for fine resolution. A penalty is applied that discriminates against curves that are overly "wiggly".

Not all count data is suitable for modeling assuming a Poisson type rare event distribution. For example, the dataset `http://maths-people.anu.edu.au/~johnm/stats-issues/data/hurric2014.csv` has details, for the years 1950-2012, of US deaths from Atlantic hurricanes. For any given hurricane, deaths are not at all independent rare events.

9.4 *Classification*

Classification models have the character of regression models where the outcome is categorical, one of g classes. The `fgl` (forensic glass) dataset that will be used as an example has measurements of each on nine physical properties, for 214 samples of glass that are classified into $g = 6$ different glass types.

For the special case $g = 2$, logistic regression models are an alternative.

This section will describe a very limited range of available approaches. For details on how and why these methods work, it will be necessary to look elsewhere.[3]

[3] Limited further details and references are provided in Maindonald and Braun: *Data Analysis and Graphics Using R*, Cambridge University Press, 3rd edn 2010.

Linear discriminant analysis (LDA), and quadratic discriminant analysis (QDA) which slightly generalizes LDA, both use linear functions of the explanatory variables in the modeling of the probabilities of group membership. These methods will be contrasted with the strongly non-parameteric approaches of tree-based classification and of random forests.

Linear and quadratic discriminant analysis

The functions that will be used are `lda()` and `qda()`, from the *MASS* package. The function `lda()` implements linear discriminant analysis, while `qda()` implements quadratic discriminant analysis.[4]

[4] Quadratic discriminant analysis is an adaptation of linear discriminant analysis to handle data where the variance-covariance matrices of the different classes are markedly different.

```
library(MASS, quietly=TRUE)
```

Results from use of `lda()` lead very naturally to useful and informative plots. Where `lda()` gives results that are a sub-optimal fit to the data, the plots may hint at what type of alternative method may be preferable. They may identify subgroups of the orginal g groups, and/or identify points that seem misclassified.

An attractive feature of `lda()` is that the discriminant rule that is obtained has a natural representation r-dimensional space. Providing that there is sufficient independent covariate data, $r = g - 1$. The analysis leads[5] to r sets of scores, where each set of scores explains a successively smaller (or at least, not larger) proportion of the sum of squares of differences of group means from the overall mean. The r sets of scores can be examined using a pairs plot. With larger numbers of groups, it will often happen that two or at most three dimensions will account for most of the variation.

[5] This is based on a *spectral* decomposition of the model matrix.

With three groups, two dimensions will account for all the variation. A scatterplot is then a geometrically complete representation of what the analysis has achieved.

Use of `lda()` to analyse the forensic glass data

As noted above, the data frame `fgl` has 10 measured physical characteristics for each of 214 glass fragments that are classified into 6 different types. First, fit a linear discriminant analysis, and use leave-one-out cross-validation to check the accuracy, thus:

```
fglCV.lda <- lda(type ~ ., data=fgl, CV=TRUE)
tab <- table(fgl$type, fglCV.lda$class)
## Confusion matrix
print(round(apply(tab, 1, function(x)x/sum(x)),
            digits=3))
```

```

         WinF WinNF   Veh   Con  Tabl  Head
  WinF  0.729 0.237 0.647 0.000 0.111 0.034
  WinNF 0.229 0.684 0.353 0.462 0.222 0.069
  Veh   0.043 0.000 0.000 0.000 0.000 0.000
  Con   0.000 0.039 0.000 0.462 0.000 0.034
  Tabl  0.000 0.026 0.000 0.000 0.556 0.000
  Head  0.000 0.013 0.000 0.077 0.111 0.862
```

The function `confusion()` (*DAAG*) makes it easy to get all the above output. Enter:

```
library(DAAG)
confusion(fgl$type, fglCV.lda$class)
```

Two-dimensional representation

Now fit the model with `CV=FALSE`, which is the default:

```
fgl.lda <- lda(type ~ ., data=fgl)
```

The final three lines of the output, obtained by entering `fgl.lda` at the command line, are:

```
Proportion of trace:
  LD1   LD2   LD3   LD4   LD5
0.815 0.117 0.041 0.016 0.011
```

The numbers show the successive proportions of a measure of the

Observe that most of the discriminatory power is in the first two dimensions.

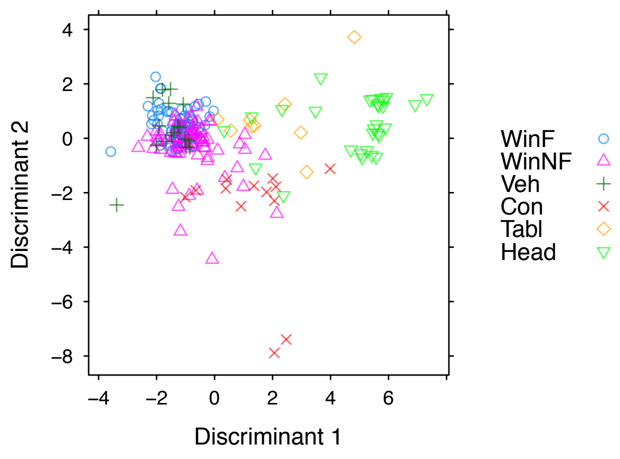


Figure 9.12: Visual representation of scores from a *linear discriminant analysis*, for the forensic glass data. A six-dimensional pattern of separation between the categories has been collapsed to two dimensions. Some categories may therefore be better distinguished than is evident from this figure.

variation that are accounted for by projections onto spaces with successively larger numbers of dimensions.

Figure 9.12 shows the two-dimensional representation.

```
library(lattice)
scores <- predict(fgl.lda)$x
xyplot(scores[,2] ~ scores[,1], groups=fgl$type,
       xlab="Discriminant 1",
       ylab="Discriminant 2",
       aspect=1, scales=list(tck=0.4),
       auto.key=list(space="right"),
```

Additionally, it may be useful to examine the plot of the third versus the second discriminant. Better still, use the abilites of the *rgl* package to examine a 3-dimensional dynamic representation. With most other methods, a low-dimensional representation does not arise so directly from the analysis.

See Figure 9.15 in Subsection 9.5.1, for an example of the type of low-dimensional representation that is possible for results from a randdom forest classification.

Two groups – comparison with logistic regression

The approach is to model the probability of class membership given the covariates, using the same logistic fixed part of the model as for linear and quadratic discriminant analysis. With π equal to the probability of membership in the second class, the model assumes that

$$\log(\pi/(1-\pi) = \beta'\mathbf{x}$$

where β is a vector of coefficients that are to be estimated, and $\mathbf{x}$ is a vector of covariate values.

A logistic regression model is a special case of a Generalized Linear Model (GLM), as implemented by R's function `glm()`. There is no provision to adjust predictions to take account of prior probabilities, though this can be done as an add-on to the analysis. Other points of difference from linear discriminant analysis are:

More technical points, as they apply to the use of R's function `glm()` for logistic regression, are:

- The fitting procedure minimizes the *deviance*. This equals 2 (*loglikelihood* for fitted model, minus the loglikelihood for the 'saturated' model). The 'saturated model has predicted values equal to observed values.
- Standard errors and Wald statistics (roughly comparable to *t*-statistics) are given for parameter estimates. These depend on approximations that may fail if predicted proportions are close to 0 or 1 and/or the sample size is small.

- Inference is conditional on the observed covariate values. A model for the probability of covariate values **x** given the class *c*, as for linear discriminant analysis, is not required. (Linear discriminant analysis assumes a multivariate normal distribution assumptions for **x**, given the class *c*. In practice, results seem relatively robust against failure of this assumption.)
- The logit model uses the *link* function $f(\pi) = \log(\pi/(1-\pi)$. Other choices of link function are available. Where there are sufficient data to check whether one of these other links may be more appropriate, this should be checked. Or there may be previous experience with comparable data that suggests use of a link other than the logit.
- Observations can be given prior weights.

9.5 Tree-based methods and random forests

On a scale in which highly parametric methods lie at one end and highly non-parametric methods at the other, linear discriminant methods lie at the parametric end, and tree-based methods and random forests at the non-parametric extreme. An attraction of tree-based methods and random forests is that model choice can be pretty much automated.

We begin by loading the *rpart* package:

```
library(rpart)
```

For the calculations that follow, data are columns in the data frame `bronchit`, in the *DAAGviz* package.

The dataset `bronchit` may alternatively be found in the *SMIR* package.

```
head(bronchit, 3)
```

```
  r  cig poll
1 0 5.15 67.1
2 0 6.75 64.4
3 0 0.00 65.9
```

Here `r=1` denotes bronchitis, while `r=0` indicates that bronchitis is absent.

In place of the variable `r` with values 0 and 1, we use a factor with levels `abs` and `pres`. Labels that appear in the output are then more meaningful.

```
## Now make the outcome variable a factor
bronchit <-
  within(bronchit,
         rfac <- factor(r, labels=c("abs","pres")))
```

The following fits a tree-based model:

```
set.seed(47)   # Reproduce tree shown
b.rpart <- rpart(rfac ~ cig+poll, data=bronchit,
                 method="class")
```

With a factor (`rfac`) as outcome, `method="class"` is the default. Setting `method="class"`, to make it quite clear that we are using a splitting rule that is appropriate to a categorical (rather than continuous) outcome, is good practice.

The “complexity” paremeter `cp`, by default set to 0.01, controls how far splitting continues. In practice, it is usual to set `cp` small enough that splitting continues further than is optimal, then pruning the tree back. Cross-validation based accuracies are calculated at each split, and can be used to determine the optimal depth of tree. Details will not be given at this point, as the interest is in trees as a lead-in to random forests. For random forests, the depth of the splits in individual trees is not of great consequence — it is not important for individual trees to be optimal.

Figure 9.13 is a visual summary of results from the tree-based classification, designed to predict the probability that a miner will have bronchitis. Where the condition at a node is satisfied, the left branch is taken. Thus, at the initial node, `cig<4.385` takes the branch to the left. In general (no random number seed), the tree may be different for each different run of the calculations.

Tree-based classification proceeds by constructing a sequence of decision steps. At each node, the split is used that best separates the data into two groups. Here (Figure 9.13) tree-based regression does unusually well (CV accuracy = 97.2%), perhaps because it is well designed to reproduce a simple form of sequential decision rule that has been used by the clinicians.

How is 'best' defined? Splits are chosen so that the Gini index of "impurity" is minimized. Other criteria are possible, but this is how `randomForest()` constructs its trees.

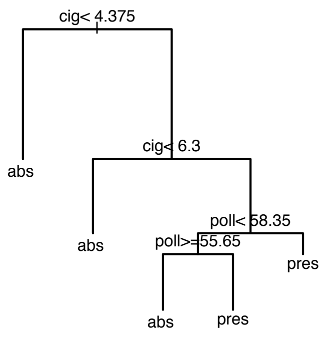


Figure 9.13: Decision tree for predicting whether a miner has bronchitis.

Code for Figure 9.13 is:

```
plot(b.rpart)
text(b.rpart, xpd=TRUE)
```

9.5.1 *Random forests*

```
library(randomForest, quietly=TRUE)
```

Figure 9.14 shows trees that have been fitted to different bootstrap samples of the bronchitis data. Typically 500 or more trees are fitted, without a stopping rule. Individual trees are likely to overfit. As each tree is for a different random sample of the data, there is no overfitting overall.

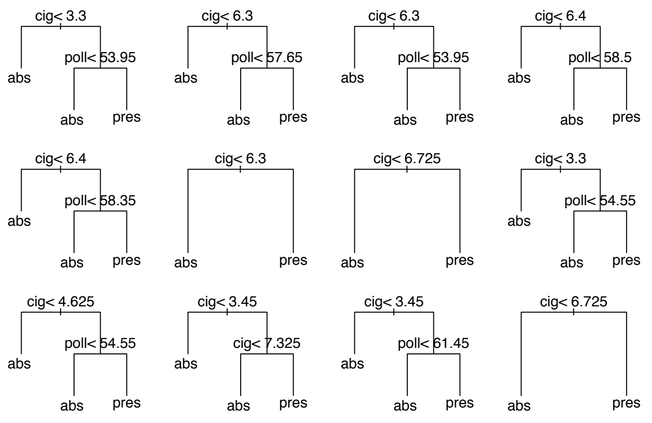


Figure 9.14: Each tree is for a different bootstrap sample of observations. The final classification is determined by a random vote over all trees. Where there are > 2 explanatory variables (but not here) a different random sample of variables is typically used for each different split. The final classification is determined by a random vote over all trees.

For each bootstrap sample, predictions are made for the observations that were not included – i.e., for the out-of-bag data. Comparison with the actual group assignments then provides an unbiased estimate of accuracy.

For the `bronchit` data, here is the `randomForest()` result.

```
(bronchit.rf <- randomForest(rfac ~ cig+poll,
                             data=bronchit))
```

```

Call:
 randomForest(formula = rfac ~ cig + poll, data = bronchit)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 1

        OOB estimate of  error rate: 22.64%
Confusion matrix:
     abs pres class.error
abs  146   20      0.1205
pres  28   18      0.6087
```

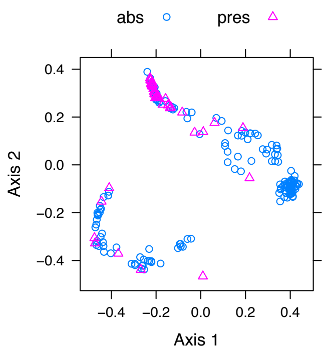


Figure 9.15: The plot is designed to represent, in two dimensions, the random forest result. It aims to reflect probabilities of group membership given by the analysis. It is not derived by a 'scaling' of the feature space.

The accuracy is much better than the `rpart()` accuracy. The random forest methodology will often improve, sometimes quite dramatically, on tree-based classification.

Figure 9.15 is a visual summary of the random forest classification result. The proportion of trees in which any pair of points appear together at the same node may be used as a measure of the "proximity" between that pair of points. Then, subtracting proximity from one to obtain a measure of distance, an ordination method is used to find an approximates representation of those points in a low-dimensional space.

There is a tuning parameter `mtry` which controls the number of randomly chosen variables considered for each tree. This is not too much of an issue for the present data, where there are only two explanatory variables.

Code for Figure 9.15 is:

```
parset <- simpleTheme(pch=1:2)
bronchit.rf <- randomForest(rfac ~ cig+poll,
                            proximity=TRUE,
                            data=bronchit)
points <- cmdscale(1-bronchit.rf$proximity)
xyplot(points[,2] ~ points[,1],
       groups=bronchit$rfac,
       xlab="Axis 1", ylab="Axis 2",
       par.settings=parset, aspect=1,
       auto.key=list(columns=2))
```

A random forest fit to the forensic glass data

The algorithm can be used in a highly automatic manner. Here then is the random forest analysis for the forensic glass data, leaving the tuning parameter (`mtry`) at its default[6]:

```
(fgl.rf <- randomForest(type ~ ., data=fgl))
```

[6] The default is to set `mtry` to the square root of the total number of variables, rounded up to an integral value.

```
Call:
 randomForest(formula = type ~ ., data = fgl)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 20.09%
Confusion matrix:
      WinF WinNF Veh Con Tabl Head class.error
WinF    63     6   1   0    0    0      0.1000
WinNF    9    59   1   4    2    1      0.2237
Veh      7     3   7   0    0    0      0.5882
Con      0     3   0   9    0    1      0.3077
Tabl     0     2   0   0    7    0      0.2222
Head     1     2   0   0    0   26      0.1034
```

This improves substantially on the linear discriminant result. This may happen because the explanatory variables have effects that are non-linear on a logit scale. The more likely reason is that there are interaction effects, perhaps of a relatively complicated kind, for which the `lda()` analysis has not accounted.

The predictive accuracy might be further improved by varying the tuning parameter `mtry` from its default. See `help(tuneRF)` for details of the function `tuneRF()` that is designed to assist in findind=g the optimum choice of `mtry`.

9.6 *Ordination

From Australian road travel distances between cities and larger towns, can we derive a plausible "map", or "ordination", showing the relative locations? The resulting "map" would give a better indication than a geographical map of the road travel effort involved in getting from one place to another.

An ordination might alternatively be based on road travel times, or on air travel times.

Genomic data provides another example. Various methods are available for calculating genomic "distances " between, e.g., different insect species. The distance measures are based on evolutionary models that aim to give distances between pairs of species that are a monotone function of the time since the two species separated.

The ordination methods described here are all versions of multi-dimensional scaling (MDS). If distances are not already given, a first tasl is to calculate 'distances' between points. Or if similarities are given, they must be first be transformed into 'distances'.

Ordination is a generic name for methods for providing a low-dimensional view of points in multi-dimensional space, such that "similar" objects are near each other and dissimilar objects are separated. The plot(s) from an ordination in 2 or 3 dimensions may provide useful visual clues on clusters in the data and/or on outliers.

One standard type of problem starts from a matrix $\mathbf{X}$ of n observations by p variables, then seeking a low-dimensional representation. A first step is then to calculate distances between observations.[7] The hope is that a major part of the information content in the p variables, as it relates to the separation between observations, can be pretty much summarized in a small number of constructed

[7] Principal components analysis circumvents the calculation of distances, for the commonly used Euclidean distance measure. See below.

variables.

There is typically no good model, equivalent to the evolutionary models used by molecular biologists, that can be used to motivate distance calculations. There is then a large element of arbritariness in the distance measure used. Results may depend strongly on the distance measure used. Unless measurements are comparable (e.g., relative growth, as measured perhaps on a logarithmic scale, for different body measurements), it is usually desirable to calculate distances from standardized variable values. This is done by subtracting the mean and dividing by the standard deviation.

If data can be separated into known classes that should be reflected in any ordination, then the scores from classification using `lda()` may be a good basis for an ordination. Plots in 2 or perhaps 3 dimensions may then reveal additional classes and/or identify points that may be misclassified and/or are in some sense outliers. They give an indication of the effectiveness of the discrimination method in choosing the boundaries between classes.

Figure 9.15 demonstrated the use of "proximities" that are available from `randomForest()` as measures of the closeness of any pair of points. These were then turned into rough distance measures that then formed the basis for an ordination. With Support Vector Machines, distance measures can be derived from the 'decision values' and used for ordination.

9.6.1 Distance measures

Euclidean distances

Treating the rows of $\mathbf{X}$ (n by p) as points in a p-dimensional space, the squared Euclidean distance d_{ij}^2 between points i and j is

$$d_{ij}^2 = \sum_{k=1}^{p} (x_{ik} - x_{jk})^2$$

The distances satisfy the triangle inequality[8]

$$d_{ij} \leq d_{ik} + d_{kj}$$

[8] This says that a straight line is the shortest distance between two points!

The columns may be weighted differently.[9] Use of an unweighted measure with all columns scaled to a standard deviation of one is equivalent to working with the unscaled columns and calculating d_{ij}^2 as

[9] More generally, they can be arbitrarily transformed before calculating the d_{ij}.

$$d_{ij}^2 = \sum_{k=1}^{p} w_{ij} (x_{ik} - x_{jk})^2$$

where $w_{ij} = (s_i s_j)^{-1}$ is the inverse of the product of the standard deviations for columns i and j.

Where all elements of a column are positive, use of the logarithmic transformation is common. A logarithmic scale makes sense for biological morphometric data, and for other data with similar characteristics. For morphometric data, the effect is to focus attention on relative changes in the various body proportions, rather than on absolute magnitudes.

Non-Euclidean distance measures

Euclidean distance is one of many possible choices of distance measures, still satisfying the triangle inequality. As an example of a non-Euclidean measure, consider the Manhattan distance. The Manhattan distance is the shortest distance for a journey that always proceeds along one of the co-ordinate axes. In Manhattan in New York, streets are laid out in a rectangular grid. This is then (with $k = 2$) the walking distance along one or other street. For other choices, see the help page for the function `dist()`.[10]

For the Manhattan distance:

$$d_{ij} = \sum_{k=1}^{p} \mid x_{ik} - x_{jk} \mid$$

[10] The function `daisy()` in the *cluster* package offers a wider choice, including distance measures for factor or ordinal data. Its argument `stand` causes prior standardization of data.

From distances to a representation in Euclidean space

Irrespective of the method of calculation of the distance measure, ordination methods yield a representation in Euclidean space. It is always possible to find a configuration X in Euclidean space in which the "distances" are approximated, perhaps rather poorly.[11] It will become apparent in the course of seeking the configuration whether an exact embedding (matrix X) is possible, and how accurate this embedding is. The representation is not unique. The matrices X and X**P**, where **P** is an orthonormal matrix, give exactly the same distances.

[11] This is true whether ot not the triangle inequality is satisfied.

The connection with principal components

Let **X** be an n by p matrix that is used for the calculation of Euclidean distances, after any transformations and/or weighting. Then metric p-dimensional ordination, applied to Euclidean distances between the rows of **X**, yields a representation in p-dimensional space that is formally equivalent to that derived from the use of principal components. The function `cmdscale()` yields, by a different set of matrix manipulations, what is essentially a principal components decomposition. Principal components circumvents the calculation of distances.

We assume that none of the columns can be written as a linear combination of other columns.

Semi-metric and non-metric scaling

Semi-metric and non-metric methods all start from "distances", but allow greater flexibility in their use to create an ordination. The aim is to represent the "distances" in some specified number of dimensions, typically two dimensions. As described here, a first step is to treat the distances as Euclidean, and determine a configuration in Euclidean space. These Euclidean distances are then used as a

The assumption of a Euclidean distance scale is a convenient starting point for calculations. An ordination that preserves relative rather than absolute distances can often be more appropriate. Additionally, small distances may be measured more accurately than large distances.

starting point for a representation in which the requirement that these are Euclidean distances, all determined with equal accuracy, is relaxed. The methods that will be noted here are Sammon scaling and Kruskal's non-metric multidimensional scaling.

Example – Australian road distances

The distance matrix that will be used is in the matrix `audists`, from the `DAAG` package. Figure 9.16 is from the use of classical multi-dimensional scaling, as implemented in the function `cmdscale()`: An alternative way to add names of cities or other labels is to use `identify()` to add labels interactively, thus:

```
identify(weight ~ volume, labels=description)
```

Code is:

```
aupts <- cmdscale(audists)
plot(aupts, axes=FALSE, ann=FALSE, fg="gray",
     frame.plot=TRUE)
city <- rownames(aupts)
pos <- rep(1,length(city))
pos[city=="Melbourne"]<- 3
pos[city=="Canberra"] <- 4
par(xpd=TRUE)
text(aupts, labels=city, pos=pos)
par(xpd=FALSE)
```

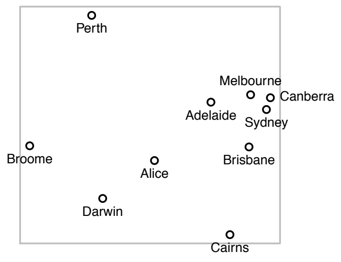


Figure 9.16: Relative locations of Australian cities, derived from road map distances, using metric scaling.

Classical multi-dimensional scaling, as implemented by `cmdscale()`, gives long distances the same weight as short distances. It is just as prepared to shift Canberra around relative to Melbourne and Sydney, as to move Perth. It makes more sense to give reduced weight to long distances, as is done by `sammon()` (*MASS*).

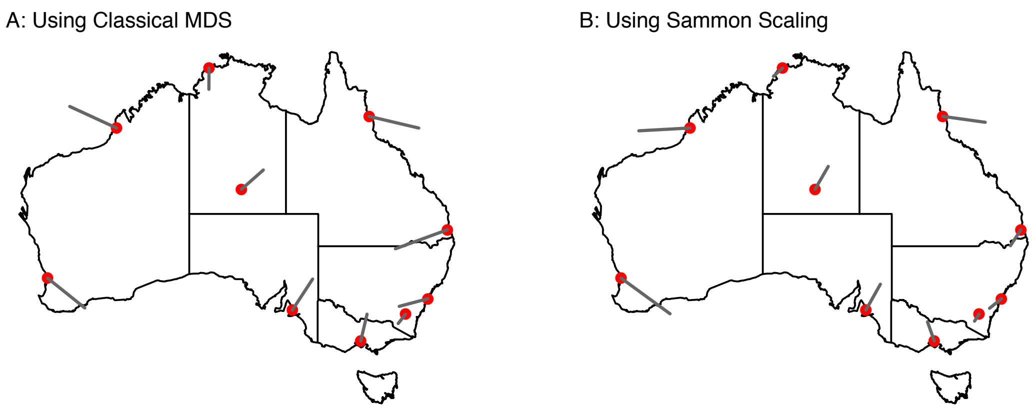


Figure 9.17: In Panel A, Figure 9.16 has been linearly transformed, to give a best fit to a map of Australia. Each city moves as shown by the line that radiates out from it. Panel B is the equivalent plot for Sammon scaling.

Figure 9.17 shows side by side the overlays of the "maps" that result from the different ordinations, onto a physical map of Australia. Panel A shows the result for classical multi-dimensional scaling.

Panel B does the same, now for the result from Sammon scaling. Notice how Brisbane, Sydney, Canberra and Melbourne now maintain their relative positions better.

To see the code for Figure 9.17, source the file that has the code for the figures of this chapter, and type:

```
fig12.15A
fig12.15B
```

The exercise can be repeated for multidimensional scaling (MDS). MDS preserves only, as far as possible, the relative distances. A starting configuration of points is required. This might come from the configuration given by `cmdscale()`. For the supplementary figure `supp12.1()` that shows the MDS configuration, however, we use the physical latitudes and longitudes.

To show this figure, source the file, if this has not been done previously, that has the code for the figures of this chapter. Then type:

```
supp12.1()
```

References

Cowpertwait P. S. P. and Metcalfe A. V. 2009. *Introductory Time Series with R.* Springer.

Hyndman, R. J.; Koehler, A. B.; Od, J. K.; and Snyder, R. D. 2008. *Forecasting with Exponential Smoothing: The State Space Approach*, 2nd edn, Springer.

Taleb, Naseem. 2004. *Fooled By Randomness: The Hidden Role Of Chance In Life And In The Markets.* Random House, 2ed.

[Has insightful comments about the over-interpretation of phenomena in which randomness is likely to have a large role.]