STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

1. PLOTS: GRAPHICAL METHODS OF DATA EXPLORATION

Producing high quality graphs is one of the main reasons for doing statistical computing. There are two fundamentally different kinds of explanatory variables, continuous and categorical, and each of these leads to a completely different sort of graph. In cases where the explanatory variable is continuous, like length or weight or altitude, the appropriate plot is a **scatterplot**. In cases where the explanatory variable is categorical, like genotype or colour or gender, then the appropriate plot is a **boxplot**.

Plotting with Continuous Explanatory Variables: Scatterplots

The first thing to learn is that **allocation** in SPlus is done using "gets" rather than "equals". "Gets" is a composite symbol <- made up from a 'less than symbol' < and a 'minus symbol' - . Thus, to make a vector called x that contains the numbers 1 through 10, we just type:

```
x<- 1:10
```

which is read as saying "x gets the values 1 to 10 in series". The colon is the series-generating operator. Now if you type **x** you will see the contents of the vector called x

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

The [1] at the beginning of the row just means that this is the first (and only) list within the object called x.

Some general points are worth making here:

- variable names are case-sensitive so x is not the same as X
- variable names should not begin with numbers (e.g. 1x) or symbols (e.g. %x)
- variable names should not contain breaks: use back.pay not back_pay or back pay
- for help with commands just type ? followed by the name you want help with

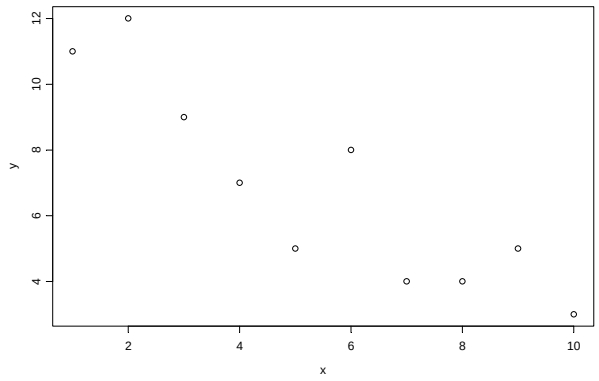
Now we type in the values for the response variable, y. The simplest way to type numbers into a vector is to use the **concatenate** directive "**c**". Concatenation is simply the process of joining lists together, and we shall meet it in many different circumstances.

```
y<-c(11,12,9,7,5,8,4,4,5,3)
```

y is called the **response variable** and is plotted on the vertical axis of a graph; x is called the **explanatory variable** and is plotted on the horizontal axis of a graph. It is extremely simple to obtain a scatter plot of y against x in SPlus.

The **plot** directive needs only 2 arguments: first the name of the explanatory variable (x in this case), and second the name of the response variable (y in this case)

```
plot(x,y)
```



For the purposes of data exploration, this may be all you need. But for publication it is useful to be able to change the appearance of the plot. It is often a good idea to have a longer, more explicit label for the axes than is provided by the variable names that are used as default options (x and y in this case). Suppose we want to change the label "x" into the longer label "Explanatory variable". To do this we use the **xlab** directive with the text of the label enclosed within double quotes. Use the Up Arrow to get back the last command line, and insert a comma after y, then type xlab= then the new label in double quotes, like this:

```
plot(x,y,xlab="Explanatory variable")
```

You might want to alter the label on the y axis as well. The directive you need for that is **ylab**. Use the Up Arrow key again, and insert the y label, like this:

```
plot(x,y,ylab="Response variable",xlab="Explanatory variable")
```

It is easy to change the plotting symbols. At the moment, you are using the default plotting character (**pch**=1) which is an open circle. If you want + (plus signs), use plotting character 3 . Use Up Arrow, then insert a comma after y, then type pch=3

```
plot(x,y,pch=3,ylab="Response variable",xlab="Explanatory variable")
```

Open triangles are plotting character pch=2, and so on.

```
plot(x,y,pch=2,ylab="Response variable",xlab="Explanatory variable")
```

Adding lines to a scatterplot

There are two kinds of lines that you might want to add to a scatterplot: (1) lines that the computer estimates for you (e.g. regression lines); or (2) lines that you specify yourself (e.g. lines indicating some theoretical values for y and x).

Regression lines

The simplest line to fit to some data is the linear regression of y on x.

$$y = a + bx$$

This has 2 variables (y and x) and 2 parameters (a and b, the intercept and the slope respectively). The model is interpreted as saying that y is a function of x, so that changing x might be expected to cause changes in y. The opposite is not true, and changing y by some other means would not be expected to bring about a change in x.

The basic statistical function to derive the two parameters of the linear regression (the intercept, *a*, and the slope, *b*) is the linear model **lm**. This uses the standard model form as its argument

y~x

This is read as "y tilde x" and means "y is to be estimated as a linear function of x". We learn all about this in Practical 4. To draw the regression line through the data, we employ the straight line drawing directive **abline** (intercept and slope, geddit?). This has two arguments: the intercept and the slope, separated by a comma. We can combine the regression analysis and the line drawing into a single directive like this:

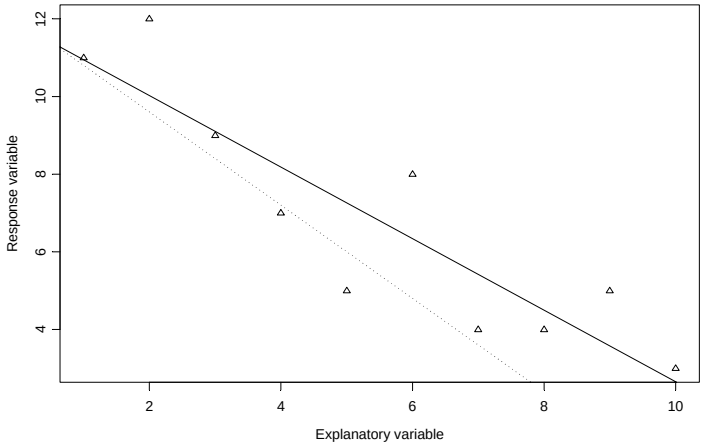
```
abline(lm(y~x))
```

One of the nice features of **abline** is that it automatically draws the line exactly within the limits set by the frame of our graph.

User-specified lines

Sometimes, you want to draw your own line on a scatter plot. A common case is where you want to draw a line of slope = 1 that goes through the origin. Or you might what to draw a horizontal line or a vertical line for some purpose. Suppose that in this case, we want to draw a line that has y = 12 for x = 0 and y = 0 when x = 10. To do this, we use concatenate to make 2 lists: a list of x points c(0,10), and a list of y points c(12,0). User-specified lines are then drawn using the **lines** directive. The first 2 arguments contain the x points of the lines and the y points of the lines, and we shall use a third argument to change the line type to number 2 (lty=2) in order to contrast with the solid regression line that we have just added

```
lines(c(0,10),c(12,0),lty=2)
```



Adding more points to a graph

The important point to appreciate is that material is overlaid on your graph until another **plot** directive is issued. You have seen lines added to the scatterplot using **abline** and **lines**. You can add new points to the scatterplot using **points**.

Sometimes we want several data sets on the same graph, and we would want to use different plotting symbols for each data set in order to distinguish between them. Suppose that we want to add the following points to our current graph. We have 5 new values of the explanatory variable: let's call them *v*, and give them the values 2, 4, 6, 8 and 10:

```
v<-c(2,4,6,8,10)
```

And 5 new values of the response variable, w:

```
w<-c(8,5,6,6,2)
```

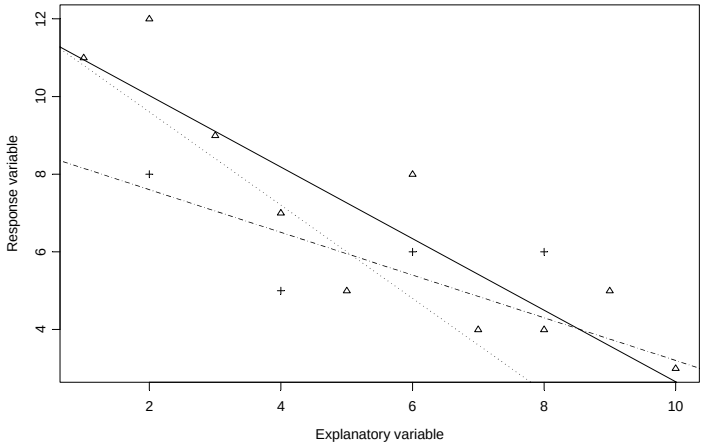
To add these points to the graph, we simply type **points** (not **plot** as this would draw fresh axes and we would lose what we already had done):

```
points(v,w,pch=3)
```

The points are added to the scatterplot using plotting character pch=3 (plus sign +). In SPlus (but **not in R**), a warning message is printed, because one of our points (10,2) is outside the plotting region. It is important to realise that the plotting region is defined when the plot directive is executed at the start of the process, and is not subsequently re-scaled as **lines** or **points** are added. Later on, we shall see how to deal with this in the context of writing general plotting routines that will always accommodate the largest and smallest values of x and y.

If we want to fit a separate regression line for these new data, we just type:

```
abline(lm(w~v),lty=3)
```



This is about as complicated as you would want to make any figure. Adding more information would begin to detract from the message.

Plotting with Categorical Explanatory Variables: Box Plots

Categorical variables are **factors** with 2 or more **levels**. For most kinds of animals, sex is a factor with 2 levels; male and female, and we could make a variable called sex (for example only) like this:

```
sex<-c("male","female")
```

The categorical variable is the factor called sex and the two levels are “male” and “female” respectively. In principle, factor levels can be names or numbers. We shall always use names, because they are so much easier to interpret, but some older software (like GLIM) can only handle numbers as factor levels.

The next example uses the factor called month to investigate weather patterns at Silwood Park. We begin by reading the data from a file like this:

```
weather<-read.table("c:\\temp\\SilwoodWeather.txt",header=T)
```

We set up a data frame called weather, which gets the contents of the file called SilwoodWeather.txt. Note the use of double slash \\ in the file path, and the option “header=T” . This means it is true (“T”) that the first row of the data file contains the names of the variables. We can see what these names are, using **names**

```
names(weather)
```

```
[1] "upper" "lower" "rain"  "month" "yr"
```

The variables in the data frame are upper and lower daily temperatures, rainfall that day, and the names of the month and year to which the data belong. To use these data we need to attach the data frame like this:

```
attach(weather)
```

There is one more bit of housekeeping we need to do before we can plot the data. We need to declare month to be a factor. At the moment, SPlus just thinks it is a number (1 – 12).

```
month<-factor(month)
```

You can always check on the status of a variable by asking “is it a factor ?” using the **is.factor** directive like this

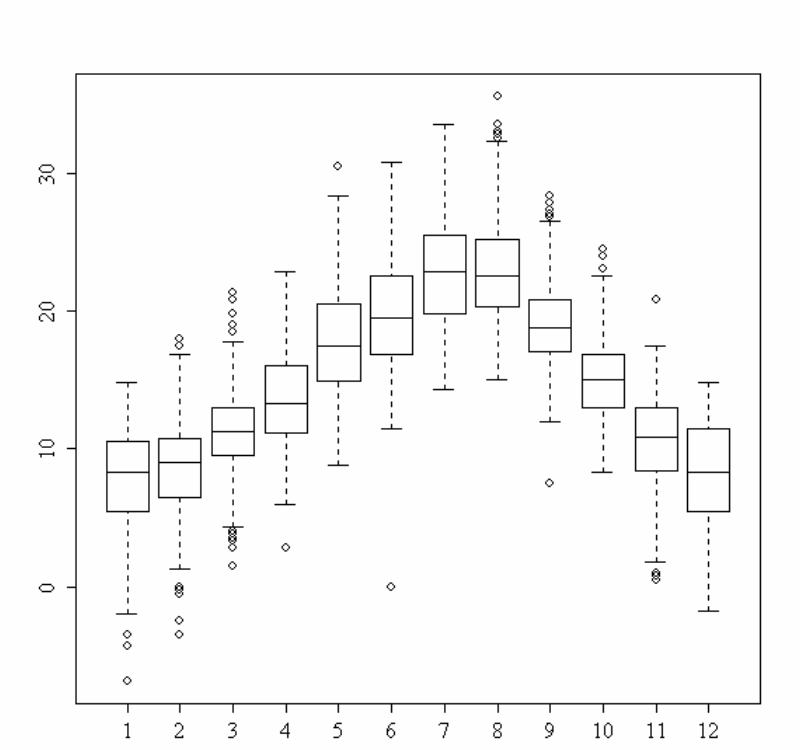
```
is.factor(month)
```

```
[1] TRUE
```

Yes, it is a factor. Now we can **plot** using a categorical explanatory variable (month)

```
plot(month,upper)
```

The syntax is exactly the same, but because the first variable is a factor we get a box plot rather than a scattergraph.



The boxplot summarises a great deal of information very clearly. The horizontal line shows the **median** response for each month. The bottom and top of the box show the 25 and 75 **percentiles** respectively (i.e. the location of the middle 50% of the data). The horizontal line joined to the box by the dashed line (sometimes called the

“whisker”) shows either the maximum or 1.5 times the **interquartile range** of the data (whichever is the smaller). Points beyond the whiskers (outliers) are drawn individually. Boxplots not only show the location and spread of data but also indicate skewness (asymmetry in the sizes of the upper and lower parts of the box). For example, in February the range of lower temperatures was much greater than the range of higher temperatures.

Summary

It is worth re-stating the really important things about plotting:

Plot: plot(xaxis,yaxis) gives a scatterplot if x is continuous, boxplot if x is a factor

Type of plot: options include lines type=”l” or null (axes only) type=”n”

Lines: lines(x,y) plots a smooth function of y against x using the x and y values provided

Line types: useful with multiple line plots, **lty**=2 (an option in plot or lines)

Points: points(x,y) adds another set of data points to a plot

Plotting characters for different data sets: **pch**=2 or **pch**=”*” (an option in points or plot)

Setting limits to x and y axis scales **xlim**=c(0,25), **ylim**=c(0,1) (an option in plot)

Colour in R (SPlus is different)

Plotting in black and white is fine for most purposes (and most printers), but colour is available, and you may want to use it (e.g. for fancy PowerPoint presentations). Try typing this:

```
pie(rep(1, 30), col = rainbow(30), radius = 0.9)
```

It produces a 30-sector colour wheel. The colour function **rainbow** is divided into 30 shades (you can change this) and a pie chart is produced with 30 equally sized sectors (each is width 1, repeated 30 times). The radius directive affects the size of the pie on the screen. See if you can produce a 10-sector colour wheel of radius 0.5.

```
pie(rep(1, 10), col = rainbow(10), radius = 0.5)
```

Colour with graphs

You might want to highlight the points on a graph using colour, draw different lines in different colours, and/or change the colour of the background. Suppose we want to draw lines of these two graphs on the same axes over the range $0 < x < 10$, but using different colours:

$$y_1 = 2 + 3x - 0.25x^2$$

$$y_2 = 3 + 3.3x - 0.3x^2$$

First we calculate values for x

```
x<-seq(0,10,0.1)
```

then the values for the two vectors y1 and y2:

```
y1 <- 2 + 3 * x - 0.25 * x ^ 2
```

```
y2 <- 3 + 3.3 * x - 0.3 * x ^ 2
```

You don't need brackets because of the "hierarchy of calculation": powers first, then times and divide, and finally plus or minus. Now we choose the paper colour to be ghostwhite (**bg** means 'background')

```
par(bg="ghostwhite")
```

The trick with multiple graphs is to draw a set of blank axes to begin with (using type = "n"), making sure that the scale is big enough to accommodate the largest values we want to plot. Since y_2 has larger values, we plot it first:

```
plot(x,y2,type="n",ylab="")
```

Now we draw the two lines, y_2 in red and y_1 in blue

```
lines(x,y2,col="red")
lines(x,y1,col="blue")
```

Coloured scatterplots

Read the following data from a file. They show the daily maximum and minimum temperatures at Silwood in January.

```
jantemps<-read.table("c:\\temp\\jantemp.txt",header=T)
attach(jantemps)
names(jantemps)
```

```
[1] "tmax" "tmin" "day"
```

To scale the axis properly we need to know the maximum and minimum values of y

```
max(tmax)
[1] 10.8


min(tmin)
[1] -11.5
```

Start by plotting the blank axes. We know that the scale on the y-axis needs to go from –12 to +12 degrees, and we want the label on the y-axis to be Temperature. We don't want to plot the data yet, so we use type = "n"

```
plot(day,tmax,ylim=c(-12,12),type="n",ylab="Temperature")
```

Now we can add the **points** to the graph, starting with the minima. We shall plot these as solid symbols (pch = 16, this stands for 'plotting character') in blue:

```
points(day,tmin,col="blue",pch=16)
```

Now for the maxima, in red:

```
points(day,tmax,col="red",pch=16)
```

Finally, we want to join together the maximum and minimum temperatures for the same day with a straight green line. For day 1 this involves plotting from the point (1, tmin[1]) to the point (1,tmax[1]). To automate this procedure for all 31 days in January we put the **lines** directive into a loop, like this

```
for (i in 1:31) lines(c(i ,i ), c( tmin[i], tmax[i] ), col="green")
```

The strong serial correlation in the weather data is clear from this plot. This is what gives rise to the weatherman's dictum when asked what tomorrow will be like: "tomorrow's weather will be like today's".

Colour with histograms

Let's produce a histogram based on 1000 random numbers from a normal distribution with mean 0 and standard deviation 1.

```
x <- rnorm(1000)
```

We shall draw the histogram on cornsilk coloured paper:

```
par(bg = "cornsilk")
```

with the bars of the histogram in a subtle shade of lavender, like this

```
hist(x, col = "lavender",  main = "")
```

The purpose of main = "" is to suppress the graph title. See what happens if you leave this out.

Colour with pie charts

Released insects landed on 6 plant species in the following proportions

```
fate <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
```

We can provide labels for the segments of the pie by naming the plant species like this:

```
names(fate)<-c("Ragwort", "Thistle", "Willowherb", "Rush", "Orchid",
        "Knapweed")
```

The pie chart can now be drawn, specifying different colours for each of the segments in sequence, like this

```
pie(fate, col = c("purple", "violetred1",  "green3", "cornsilk", "cyan",
        "white"))
```

Here **col** is specified as a vector with as many levels as there are sectors in the pie (6 in this case).

Multivariate Plots

Data sets often consist of many continuous variables, and it is important to find out if, and how, the variables are inter-related. Read the data file called pollute.txt

```
pollution<-read.table("c:\\temp\\pollute.txt",header=T)
attach(pollution)
```

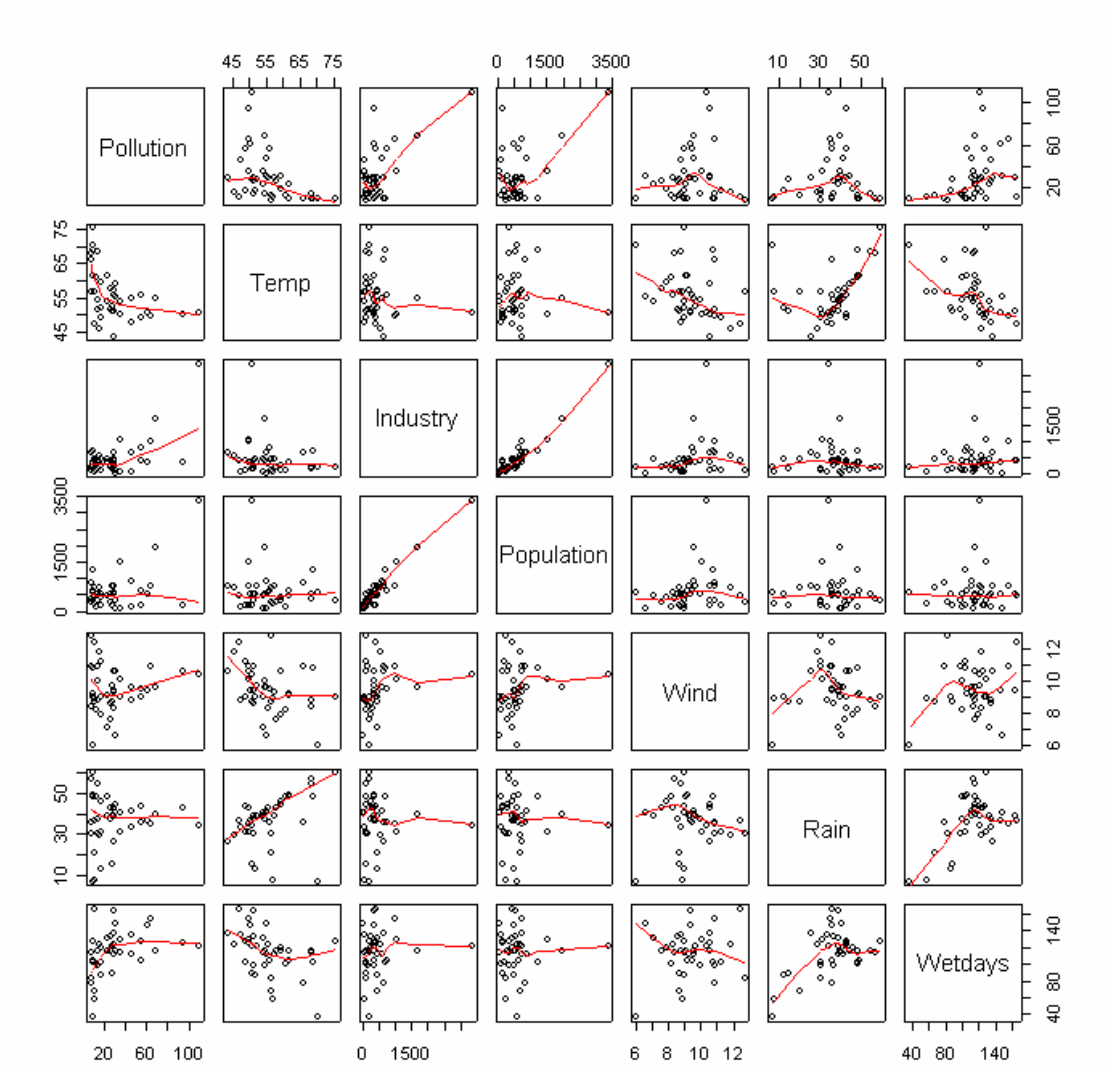
To see which variables are included in the data frame called pollution we use **names**

```
names(pollution)
```

```
[1] "Pollution"  "Temp"       "Industry"   "Population"
"Wind"       "Rain"  "Wetdays"
```

The directive **pairs** produces a matrix of plots of y against x and x against y for all of the variables in the data frame (all 7 in this case).

```
pairs(pollution,panel=panel.smooth)
```



Interpretation of the matrix takes some getting used to. The rows of the matrix have the response variable (the y axis) as labelled by the variable name that appears in that row. For example, all the graphs on the middle row (the 4th row) have Population on the y axis. Likewise, the columns of the matrix all have the same explanatory variable (the x axis) as labelled by the variable name that appears in that column. For example, all the graphs in the right-most column have Wetdays on their x axis.

This kind of multiple scatterplot is good at showing some patterns but poor at showing others. Where the data are well spaced, then relationships can show up clearly. But when the data are clumped at one end of the axis it is much more difficult. Likewise, the plots are poor when variables are categorical (integer). Not unexpectedly, there is a very close correlation between Industry and Population, but the relationship between Pollution and Wind is far from clear. We shall carry out the statistical analysis of these data later on (see Multiple Regression).

Tree-based models

The most difficult problems in interpreting multivariate data are that

1. the explanatory variables may be correlated with one another
2. there may be interactions between explanatory variables
3. relationships between the response and explanatory variables may be non-linear

An excellent way of investigating interactions between multiple explanatory variables involves the use of tree-based models. We need to get the code from the library:

```
library(tree)
```

Tree models are constructed on a simple, stepwise principle. The computer works out which of the variables explains most of the variance in the response variable, then determines the threshold value of the explanatory variable that best partitions the variance in the response. Then the process is repeated for the y values associated with *large values* of the first explanatory variable, asking which explanatory variable explains most of this variation. The process is then repeated for the y values associated with *low values* of the first explanatory variable. The process is repeated until there is no residual explanatory power. The procedure is very simple, involving the directive **tree**. For the pollution data set, we type:

```
regtree<-tree(Pollution ~ . , data=pollution)
```

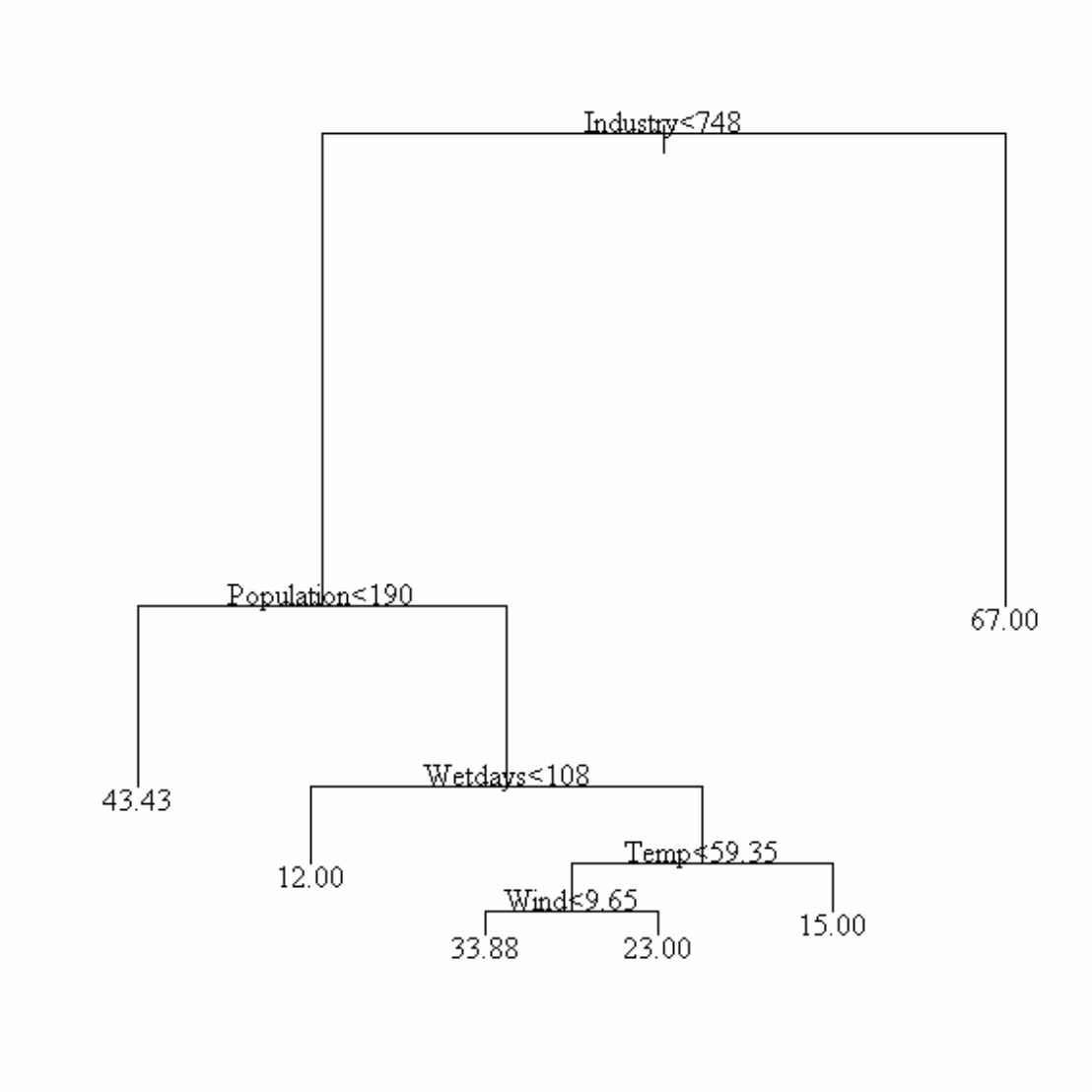
where the model formula reads "Pollution (our response variable) is a function of *all* of the explanatory variables in the data frame called pollution". The power comes from the 'dot option'. The tilde ~ means "is a function of". The dot . means "everything". So make sure you get the syntax right:

the punctuation is "**tilde dot comma**"

Now the object called **regtree** contains the regression tree and we want to see it, and to label it:

```
plot(regtree)
text(regtree)
```

This is interpreted as follows:



The most important explanatory variable is Industry, and the threshold value separating low and high values of Industry is 748. The right hand branch of the tree indicates the mean value of air pollution for high levels of industry (67.00). The fact that this limb is unbranched means that no other variables explain a significant amount of the variation in pollution levels for high values of Industry. The left-hand limb does not show the mean values of pollution for low values of industry, because there are other significant explanatory variables. Mean values of pollution are only shown at the extreme ends of branches. For low values of Industry, the tree shows us that Population has a significant impact on air pollution. At low values of Population (<190) the mean level of air pollution was 43.43. For high values of Population, the number of Wetdays is significant. Low numbers of wet days (< 108) have mean pollution levels of 12.00 while Temperature has a significant impact on pollution for places where the number of wet days is large. At high temperatures (> 59.35 ‘F) the mean pollution level was 15.00 while at lower temperatures the run of Wind is important. For still air (Wind < 9.65) pollution was higher (33.88) than for higher wind speeds (23.00). The statistical analyses of regression trees, including the steps involved in model simplification, are dealt with later (see Practical 10).

The virtues of tree-based models are numerous:

1) they are easy to appreciate and to describe to other people
2) the most important variables stand out
3) interactions are clearly displayed
4) non-linear effects are captured effectively (e.g. as step functions)
5) the complexity (or lack of it) in the behaviour of the explanatory variables is plain to see

Conditioning plots

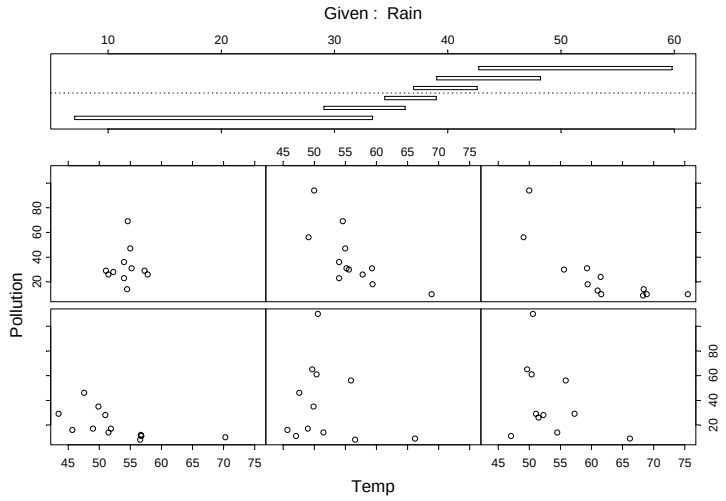
A real difficulty with multivariate data is that the relationship between two variables may be obscured by the effects of other processes.

```
attach(pollution)
```

When you carry out a 2-dimensional plot of y against x, then all of the effects of the other explanatory variables are squashed flat onto the plane of the paper. The function **coplot** produces a conditioning plot automatically in cases like the pollution example here, where the explanatory variables are continuous. It is extremely easy to use. Suppose we want to look at Pollution as a function of temperature but we want to condition this on different levels of rainfall. This is all we do:

```
coplot(Pollution~Temp | Rain)
```

The plot is described by a model formula: Pollution on the y axis, Temp on the x axis, with 6 separate plots conditioned on the value of Rain.



The panels are ordered from lower left, row-wise, to upper right, from lowest rainfall to highest. The upper panel shows the range of values for Rain chosen by **coplot** to form the 6 panels. Note that the range of rainfall values varies from panel to panel. The largest range is in the bottom left plot (range 8 to 33) and the smallest for the

bottom right plot (35 to 39). Note that the ranges of Rain overlap one another at both ends; this is called a **shingle**.

Graphical Parameters (par)

Without doubt, the graphical parameter you will change most often just happens to be the least intuitive to use. This is the number of graphs per screen, called somewhat unhelpfully, **mfrow**. The idea is simple, it is just the syntax that is hard to remember. You need to specify the number of rows of plots you want, and number of plots per row, in a vector of 2 numbers. The first number is the number of rows and the second number is the number of graphs per row. The vector is made using **c** in the normal way (**c** means concatenate). The default single plot screen is

```
par(mfrow=c(1,1))
```

Two plots side by side is

```
par(mfrow=c(1,2))
```

Four plots in a 2 x 2 square is

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

To move from one plot to the next, you need to execute a new **plot** directive. Control stays within the same plot frame while you execute directives like **points**, **lines** or **text**. Remember to return to the default single plot when you have finished your multiple plot by executing **par(mfrow=c(1,1))**.

If you have more than 2 graphs per row or per column, the character expansion **cex** is set to 0.5 and you get half-size characters and labels.

Logarithmic axes

You will often want to have one or both of your axes on a logarithmic scale.

log="xy", log="x" or log="y" controls logarithmic axes, producing log-log, log-x or log-y axes respectively. Here are the same data plotted 4 ways:

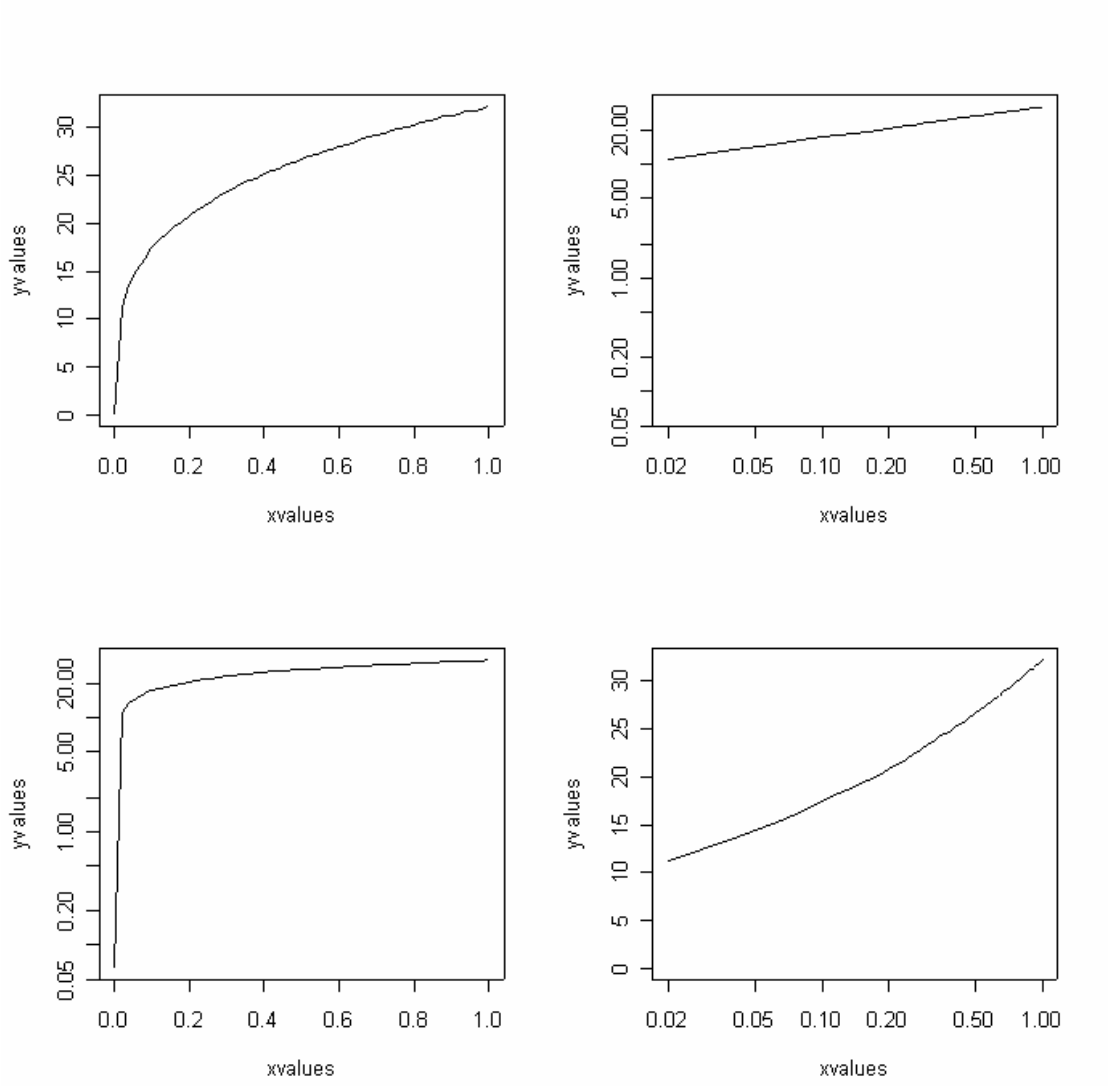
```
plotdata<-read.table("c:\\temp\\plotdata.txt", header =T)

attach(plotdata)
names(plotdata)

[1] "xvalues" "yvalues"
```

Make a frame for 4 adjacent plots is a 2 x 2 array:

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

From top left to bottom right plot the data with untransformed axes, as a log-log plot, as log(y) against x and as y against log(x)

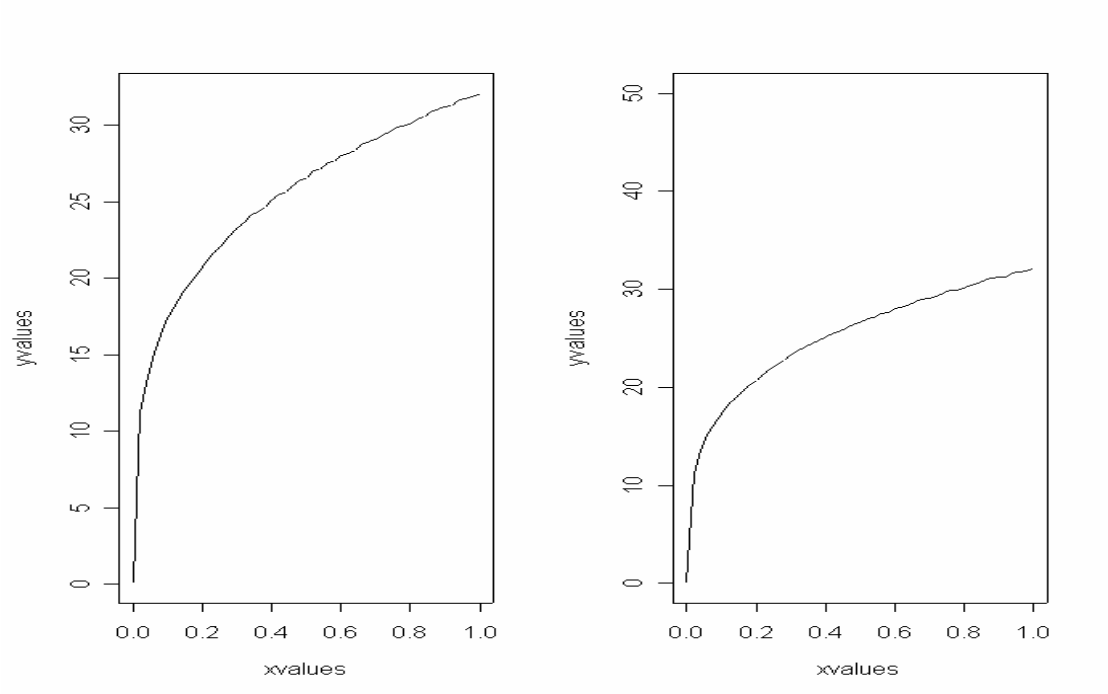
```
plot(xvalues,yvalues,type=”l”)
plot(xvalues,yvalues,log="xy",type=”l”)
plot(xvalues,yvalues,log="y", type=”l”)
plot(xvalues,yvalues,log="x", type=”l”)
```



Scaling the axes

To change the upper and lower values on the axes use **xlim** and **ylim** like this

```
par(mfrow=c(1,2))
plot(xvalues,yvalues,type=”l”)
plot(xvalues,yvalues,ylim=c(0,50),type=”l”)
```



You concatenate **c** a list of length 2, containing the approximate minimum and maximum values to be put on the axis. These values are automatically rounded to make them "pretty" for axis labelling. You will want to control the scaling of the axis

- when you want 2 comparable graphs side by side
- when you want to overlay several lines or sets of points on the same axes
- remember that the initial **plot** directive sets the axes scales
- this can be a problem if subsequent **lines** or **points** are off-scale

Text in graphs

It is very easy to add text to graphics. Just work out the x and y coordinates where you want the centre of the text to appear on an existing plot, using locator(1) to move the cursor to the appropriate position then left click the mouse to see the coordinates x and y. Suppose we want to add “(b)” at the point (0.8,45), then we just type:

```
text(0.8,45,”(b)”)
```

If you have factor level names with spaces in them (e.g. multiple words), then the best format for reading files is comma delimited (“.csv” rather than the standard tab delimited, ”.txt” files). You read them into a dataframe in R using read.csv in place of read.table. The next example is a bit more complicated. The task is to produce a map of place names, where the names required for plotting are in one file called map.places.csv, but their coordinates are in another, much longer file called bowens.csv, containing all place names.

```
map.places <-read.csv("c:\\temp\\map.places.csv",header=T)
attach(map.places)
names(map.places)
```

```
[1] "wanted"
```

```
map.data<-read.csv("c:\\temp\\bowens.csv",header=T)
attach(map.data)
names(map.data)

[1] "place" "east"  "north"
```

There is a slight complication to do with the coordinates. The northernmost places are in a different 100 km square so, for instance, a northing of 3 needs to be altered to 103. It is convenient that all of the values that need to be changed have northings < 60 in the dataframe:

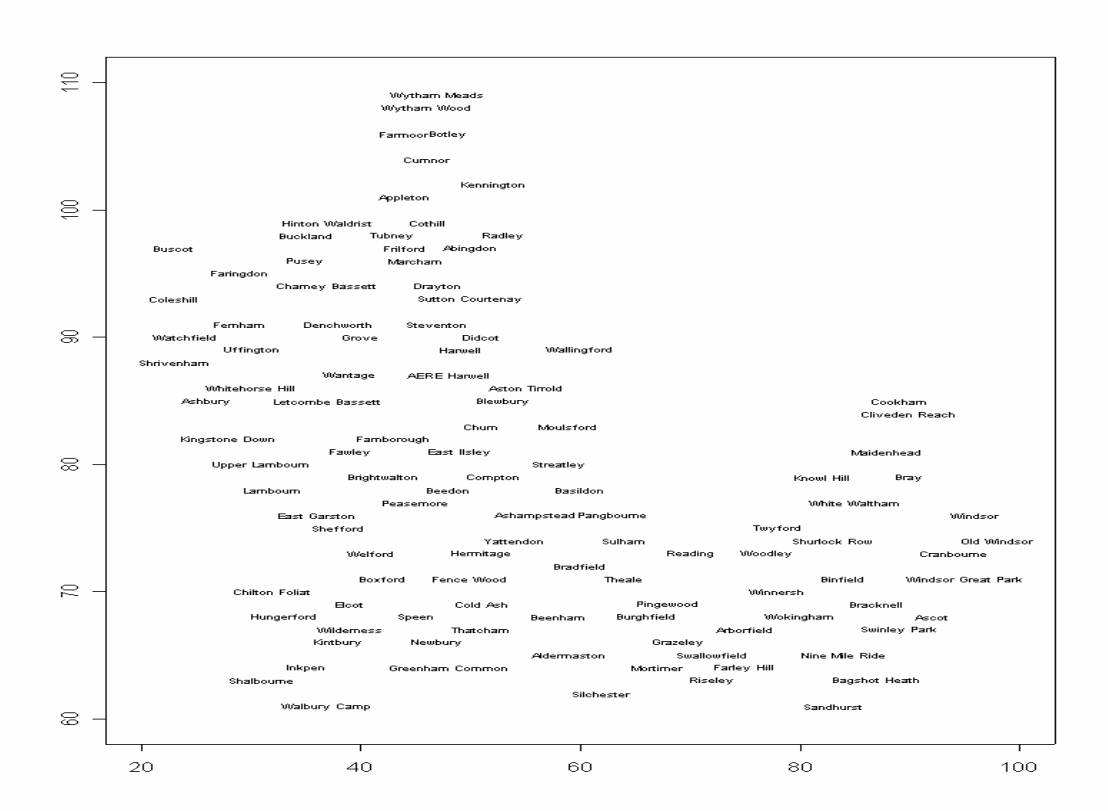
```
nn<-ifelse(north<60,north+100,north)
```

This means change all of the northings for which north < 60 is TRUE to nn<-north+100, and leave unaltered all the others (FALSE) as nn<-north. We begin by plotting a blank space (type=”n”) of the right size (eastings from 20 to 100 and northings from 60 to 110) with blank axis labels “”.

```
plot(c(20,100),c(60,110),type="n",xlab="",ylab="")
```

The trick is to select the appropriate places in the vector called place and use text to plot each name in the correct position (east[i],nn[i]). For each place name in wanted we find the correct subscript for that name within place using the which function

```
for (i in 1:length(wanted)){
ii <- which(place == as.character(wanted[i]))
text(east[ii], nn[ii], as.character(place[ii]), cex = 0.6) }
```



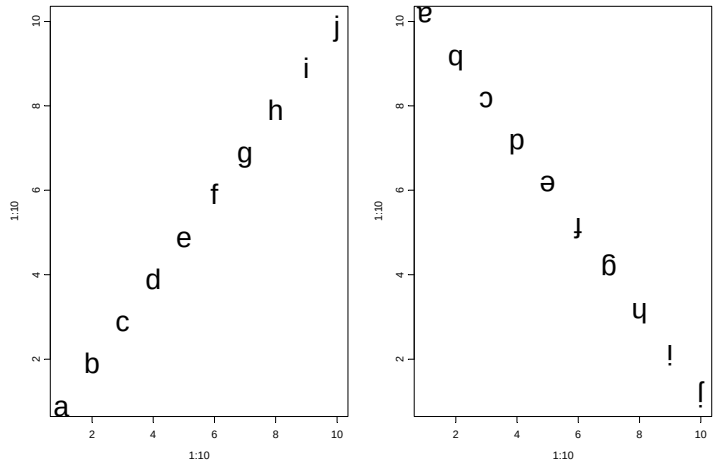
Character alignment

```
labels<-letters[1:10]
labels
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

Plot type = “n” is useful for scaling the axes but suppressing the creation of any filler for the space within the axes:

```
plot(1:10,1:10,type="n")
text(1:10,1:10,labels,cex=2)
plot(1:10,1:10,type="n")
text(1:10,10:1,labels,cex=2,srt=180)
```



Note the use of string (character) rotation **srt** (degrees counter clockwise from the horizontal) to turn the letters upside down (**srt** = 180) in the right hand plot. Note also the reversal of the sequence argument, so that the y values of the character locations in the right hand plot go down from 10 to 1 as x goes up from 1 to 10. These letters are twice normal size (**cex** = 2 stands for “character expansion 2-fold”).

STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

2. VECTORS AND LOGICAL ARITHMETIC

The screen prompt is an excellent calculator. All of the usual calculations can be done directly, and the standard order of precedence applies: powers and roots first, then multiplication and division, then finally addition and subtraction. Although there is a named function for square roots, **sqrt**, roots are generally calculated as fractional powers. So the cube root of 8 is

```
8^(1/3)
```

```
[1] 2
```

Use brackets as necessary to over-ride the precedence. For a t-test, where the value required is $\frac{|y_A - y_B|}{SE_d}$ and the numbers are 5.7, 6.8 and 0.38 you type

```
abs(5.7-6.8)/0.38
```

```
[1] 2.894737
```

All the mathematical functions you could ever want are here (see Table 1).

Integer quotients and remainders are obtained using the rather clumsy notation %/% (percent, divide, percent) and %% (percent, percent) respectively. Suppose we want to know the integer part of a division: say, how many 13's are there in 119

```
119 %/% 13
```

```
[1] 9
```

Now suppose we wanted to know the remainder (what is left over when 119 is divided by 13): in maths this is known as **modulo**

```
119 %% 13
```

```
[1] 2
```

Various sorts of rounding (rounding up, rounding down, rounding to the nearest integer) can be done easily. Take 5.7 as an example. The ‘greatest integer’ function is **floor**

```
floor(5.7)
```

```
[1] 5
```

and the ‘next integer’ function is **ceiling**

```
ceiling(5.7)
```

```
[1] 6
```

You can round to the nearest integer by adding 0.5 to the number then using **floor**. There is a built in function for this, but we can easily write one of our own to introduce the notion of function-writing. Call it *rounded*, then define it as a function like this

```
rounded<-function(x)  floor(x+0.5)
```

Now we can use the new function

```
rounded(5.7)
```

```
[1] 6
```

Table 1. Mathematical functions.

Function	Meaning
log(x)	log to base e of x
exp(x)	antilog of x (2.7818^x)
log(x,n)	log to base n of x
log10(x)	log to base 10 of x
sqrt(x)	square root of x
factorial(x)	x!
choose(n,x)	binomial coefficients n! / (x! (n-x)!)
gamma(x)	Γ x (x-1)! for integer x
lgamma(x)	natural log of gamma x
floor(x)	greatest integer < x
ceiling(x)	smallest integer > x
trunc(x)	closest integer to x between x and 0 trunc(1.5) = 1, trunc(-1.5) = -1 trunc is like floor for positive values and like ceiling for negative values
round(x, digits=0)	round the value of x to an integer
signif(x, digits=6)	give x to 6 digits in scientific notation
runif(n)	generates n random numbers between 0 and 1 from a uniform distribution
cos(x)	cosine of x in radians
sin(x)	sine of x in radians
tan(x)	tangent of x in radians
acos(x), asin(x), atan(x)	inverse trigonometric transformations of real or complex numbers.
acosh(x), asinh(x), atanh(x)	inverse hyperbolic trigonometric transformations on real or complex numbers
abs(x)	the absolute value of x, ignoring the minus sign if there is one

Generating sequences

The simplest way to generate a regularly spaced sequence of numbers is to use the colon operator like this:

```
1:7
```

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

For more complicated sequences, use the **seq** function in which the 3rd argument can be the step length (the increment or decrement you want to use): going up

```
seq(0,1,0.1)
```

```
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

or coming down

```
seq(5,-5,-1)
```

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

The **along** option allows you to map a sequence onto an existing vector (to ensure equal lengths) or if you know how many numbers you want but you can't be bothered to work out the final value of a series, you can do this. Suppose we want a series of 20 numbers, starting at 5 and going down in steps of –1:

```
seq(from=5,by= -1, along=1:20)
```

```
[1]   5   4   3   2   1   0  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10 -11
-12 -13 -14
```

Generating repeats

The **rep** function replicates the input either a certain number of times or to a certain length. In the unlikely event that you wanted to get 5.3 repeated 17 times you would type

```
rep(5.3,17)
```

```
 [1] 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3
```

The first parameter is the number and the second parameter is the repeat. More useful applications involve repeating sets of numbers, variable numbers of times. The general syntax is like this:

rep(x, times)

If times consists of a single integer, the result consists of the values in x repeated this many times. If times is a vector of the same length as x, the result consists of x[1] repeated times[1] times, x[2] repeated times[2] times and so on. The thing to remember is that the *length* of the second argument (the number of each repeat) has to match the length of the first.

rep(1:6,2)

```
 [1] 1 2 3 4 5 6 1 2 3 4 5 6
```

This says repeat the whole series 1 to 6, twice. That's fine, but what if we wanted each of the numbers 1 to 6 repeated, say, 3 times. We need to write a list of six 3's explicitly like this:

rep(1:6,rep(3,6))

```
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6
```

This emulates the old 'generate levels' function for those of you who grew up with GLIM. In R there is a direct copy of generate levels called **gl**. The third argument is the length of the whole vector (18 in this case)

gl(6,3,18)

```
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6

Levels:  1 2 3 4 5 6
```

The most complex case arises when we want to repeat each element of the first list a different number of times. One very symmetric case might be if we wanted one 1, two 2's, three 3's and so on. This is

rep(1:6,1:6)

```
 [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6
```

but more generally, we would like to be able to specify each repeat separately. We do that by using concatenate, **c**, to list each of the individual repeats in sequence. The list of numbers inside c(…) must match the length of the initial sequence:

rep(1:6,c(1,2,3,3,2,1))

This says that the 1 is repeated once, the 2 twice, the 3 and 4 thrice, the 5 twice and the 6 once:

```
 [1] 1 2 2 3 3 3 4 4 4 5 5 6
```

Finally, we might want to customise both the sequence and the repeat. This is especially useful in generating repeated factor levels that are text:

```
rep(c("monoecious","dioecious","hermaphrodite","agamic"),c(3,2,7,3))
```

```
[1] "monoecious"    "monoecious"    "monoecious"    "dioecious"     "dioecious"      "hermaphrodite"
[7] "hermaphrodite" "hermaphrodite" "hermaphrodite" "hermaphrodite" " hermaphrodite" "hermaphrodite"
[13] "agamic"       "agamic"        "agamic"
```

Table 2. Logical operations

Symbol	Meaning
!=	not equal
%%	the remainder of a division, modulo
%/%	the integer part
*	multiplication
+	addition
-	subtraction
^	to the power
/	division
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	logical equals (double =)

Vectors

The real power of the calculator is manifest in its abilities for vector calculations. Suppose that y contains the following 10 values:

```
y<-c(7,5,7,2,4,6,1,6,2,3)
```

where **c** is the concatenation function. This is a long name for a simple idea; **c** combines objects from left to right into a vector (as here) or a list, tagging the next item onto the bottom of the list so far. This is useful if you want a vector to be twice as long, with the contents repeated. For example, to string together 2 copies of the vector called y, end to end, we just write:

```
y<-c(y,y)
```

```
y
```

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

The most commonly used vector functions are **sum** and **mean**. Although we don't need to do this, because there is a built in function **var** to calculate the variance of a vector of numbers, it is useful as a demonstration of sum and mean in action. The formula we want to apply to our vector is the familiar definition of sample variance

$$s^2 = \frac{\sum (y - \bar{y})^2}{n-1}$$

which involves both sum $\sum$ and mean $\bar{y}$. There is an extra vector function that is very useful here. In this case we know that there are 20 numbers in y so we could divide by 19 to get the variance. But this is no good in general. The sample size *n* is found as the **length** of any vector, so we can write the variance like this:

```
sum((y-mean(y))^2)/(length(y)-1)
```

```
[1] 4.642105
```

In this case, we can use the built in function to check whether we got it right:

```
var(y)
```

```
[1] 4.642105
```

Sorting, Ranking and Ordering

These three related concepts are important and one of them (order) is difficult to understand on first acquaintance. Let's take a simple example

```
data<-c(7,4,6,8,9,1,0,3,2,5,0)
```

Now we apply the 3 different functions to the vector called data

```
ranks<-rank(data)
sorted<-sort(data)
ordered<-order(data)
```

We make a data frame out of the 4 vectors like this

```
view<-data.frame(data,ranks,sorted,ordered)
view
```

	data	ranks	sorted	ordered
1	7	9.0	0	7
2	4	6.0	0	11
3	6	8.0	1	6
4	8	10.0	2	9
5	9	11.0	3	8
6	1	3.0	4	2
7	0	1.5	5	10
8	3	5.0	6	3
9	2	4.0	7	1
10	5	7.0	8	4
11	0	1.5	9	5

Rank

The data themselves are in no particular sequence. Ranks contains the value that is the rank, out of **length**(data), of the particular data point. So the first element data = 7 is the 9th highest value in data. You should check that there are 8 values smaller than 7 in the vector called data. Fractional ranks indicate ties. There are 2 zero's in the data and their ranks are 1 and 2. Because they are tied, each gets the average of the sum of their ranks (1+2)/2 = 1.5.

Sort

The sorted vector is very straightforward. It contains the values of data sorted into ascending order. If you want to sort into descending order, use the reverse order directive **rev** like this y<-rev(sort(x)). Note that ***sort is potentially very dangerous***, because it uncouples values that might need to be in the same row of the data frame (e.g. because they are the explanatory variables associated with a particular value of the response variable).

Order

This returns an integer vector containing the permutation that will sort the input into ascending order. You will need to think about this one. Look at the data frame and ask yourself "What is the subscript in the original vector called data of the value in the 1st element of the sorted vector. The 1st zero is in location 7 and this is order[1]. Where is the 2nd value in the sorted vector found within the original data? It is in position 11, so this is order[2]. The only 1 is found in position 6 within data, so this is order[3]. And so on.

Dataframes

A dataframe is an object with rows and columns (a bit like a 2-dimensional matrix). The rows contain different observations from your study, or measurements from your experiment. The columns contain different variables. The values in the body of the dataframe can be numbers (as they would be in as matrix), but they could also be text (e.g. the names of factor levels for categorical variables, like "male" or "female" in a variable called "gender"), they could be calendar dates (like 23/5/04), or they could be logical variables (like "TRUE" or "FALSE"). Here is a dataframe with 7 variables, the left-most of which comprises the row names, and other variables are numeric (Area, Slope, Soil pH and Worm density), categorical (Field Name and Vegetation) or logical (Damp is either true = T or false = F).

Field Name	Area	Slope	Vegetation	Soil pH	Damp	Worm density
Nash's Field	3.6	11	Grassland	4.1	F	4
Silwood Bottom	5.1	2	Arable	5.2	F	7
Nursery Field	2.8	3	Grassland	4.3	F	2
Rush Meadow	2.4	5	Meadow	4.9	T	5
Gunness' Thicket	3.8	0	Scrub	4.2	F	6
Oak Mead	3.1	2	Grassland	3.9	F	2
Church Field	3.5	3	Grassland	4.2	F	3
Ashurst	2.1	0	Arable	4.8	F	4
The Orchard	1.9	0	Orchard	5.7	F	9
Rookery Slope	1.5	4	Grassland	5	T	7
Garden Wood	2.9	10	Scrub	5.2	F	8
North Gravel	3.3	1	Grassland	4.1	F	1
South Gravel	3.7	2	Grassland	4	F	2
Observatory Ridge	1.8	6	Grassland	3.8	F	0
Pond Field	4.1	0	Meadow	5	T	6
Water Meadow	3.9	0	Meadow	4.9	T	8
Cheapside	2.2	8	Scrub	4.7	T	4
Pound Hill	4.4	2	Arable	4.5	F	5
Gravel Pit	2.9	1	Grassland	3.5	F	1
Farm Wood	0.8	10	Scrub	5.1	T	3

Perhaps the most important thing about analysing your own data properly is getting your dataframe absolutely right. The expectation is that you will have used a spreadsheet like Excel to enter and edit the data, and that you will have used plots to check for errors. The thing that takes some practice is learning exactly how to put your numbers into the spreadsheet. There are countless ways of doing it wrong, but only one way of doing it right. And this way is not the way that most people find intuitively to be the most obvious.

The key thing is this: *all values of the same variable must go in the same column.* It does not sound like much, but this is what people tend not to do. If you had an experiment with a control, pre-heated and pre-chilled as three treatments, and 4 measurements per treatment, it might seem like a good idea to create the spreadsheet like this:

control	preheated	prechilled			
6.1	6.3	7.1			
5.9	6.2	8.2			
5.8	5.8	7.3			
5.4	6.3	6.9			

This is not a dataframe, because values of the response variable appear in 3 different columns. The correct way to enter these data is to have two columns: one for the response and one for the levels of the experimental factor. Here are the same data, entered correctly as a dataframe:

response	treatment			
6.1	control			
5.9	control			
5.8	control			
5.4	control			
6.3	preheated			
6.2	preheated			
5.8	preheated			
6.3	preheated			
7.1	prechilled			
8.2	prechilled			
7.3	prechilled			
6.9	prechilled			

A good way to practice this layout is to use the excellent Excel function called Pivot Table (found under the Data tab on the main menu bar) on your own data: it requires your spreadsheet to be in the form of a dataframe, with each of the explanatory variables in its own column.

Once you have made your dataframe in Excel and corrected all the inevitable data-entry and spelling errors, then you need to save the dataframe in a file format that can be read by R. Much the simplest way is to save all your dataframes from Excel as tab-delimited text files: File / Save As / … then from the "Save as type" options choose "Text (Tab delimited)". There is no need to add a suffix, because Excel will automatically add ".txt" to your file name. This file can then be read into R directly as a dataframe, using the read.table function.

It is important to note that read.table would fail if there were any spaces in any of the variable names in row 1 of the dataframe (the header row) like Field Name, Soil pH or Worm Density, or between any of the words within the same factor level (as in many of

the Field Names). We should replace all these spaces by dots "." before saving the dataframe in Excel (use Edit /Replace with " " replaced by "."). Now the dataframe can be read into R. There are 3 things to remember:

- the whole path and file name needs to be enclosed in double quotes: "c:\\abc.txt"
- header =T says that the first row contains the variable names
- always use double backslash \\ rather than \ in the file path definition

Think of a name for the data frame (say "worms" in this case). Now use the **gets arrow** <- which is made up of the two characters < (less than) and – (minus) like this

```
worms<-read.table("c:\\temp\\worms.txt",header=T,row.names=1)
```

Once the file has been imported to R we want to do two things:

- use **attach** to make the variables accessible by name within the R session
- use **names** to get a list of the variable names

Typically, the 2 commands are issued in sequence, whenever a new data frame is imported from file:

```
attach(worms)
names(worms)
```

```
[1] "Area"         "Slope"        "Vegetation"
[4] "Soil.pH"      "Damp"         "Worm.density"
```

To see the contents of the dataframe, just type its name:

```
worms
```

```
                 Area Slope Vegetation Soil.pH  Damp Worm.density
Nash's.Field      3.6    11  Grassland     4.1 FALSE            4
Silwood.Bottom    5.1     2     Arable     5.2 FALSE            7
Nursery.Field     2.8     3  Grassland     4.3 FALSE            2
Rush.Meadow       2.4     5     Meadow     4.9  TRUE            5
Gunness'.Thicket  3.8     0      Scrub     4.2 FALSE            6
Oak.Mead          3.1     2  Grassland     3.9 FALSE            2
Church.Field      3.5     3  Grassland     4.2 FALSE            3
Ashurst           2.1     0     Arable     4.8 FALSE            4
The.Orchard       1.9     0    Orchard     5.7 FALSE            9
Rookery.Slope     1.5     4  Grassland     5.0  TRUE            7
Garden.Wood       2.9    10      Scrub     5.2 FALSE            8
North.Gravel      3.3     1  Grassland     4.1 FALSE            1
South.Gravel      3.7     2  Grassland     4.0 FALSE            2
```

```
Observatory.Ridge  1.8     6  Grassland     3.8 FALSE           0
Pond.Field         4.1     0     Meadow     5.0  TRUE           6
Water.Meadow       3.9     0     Meadow     4.9  TRUE           8
Cheapside          2.2     8      Scrub     4.7  TRUE           4
Pound.Hill         4.4     2     Arable     4.5 FALSE           5
Gravel.Pit         2.9     1  Grassland     3.5 FALSE           1
Farm.Wood          0.8    10      Scrub     5.1  TRUE           3
```

If, as here, the rows have unique names as part of the dataframe, we suppress R's natural inclination to produce its own row numbers, by telling R the number of the column containing the row names (1 in this case). Notice, also, that R has expanded our abbreviated T and F into TRUE and FALSE. The object called worms now has all the attributes of a dataframe. For example, you can summarize it, using summary:

```
summary(worms)
```

```
      Area           Slope          Vegetation    Soil.pH
 Min.   :0.800   Min.   : 0.00   Arable   :3    Min.   :3.500
 1st Qu.:2.175   1st Qu.: 0.75   Grassland:9    1st Qu.:4.100
 Median :3.000   Median : 2.00   Meadow   :3    Median :4.600
 Mean   :2.990   Mean   : 3.50   Orchard  :1    Mean   :4.555
 3rd Qu.:3.725   3rd Qu.: 5.25   Scrub    :4    3rd Qu.:5.000
 Max.   :5.100   Max.   :11.00                  Max.   :5.700
    Damp          Worm.density
 Length:20        Min.   :0.00
 Mode  :logical   1st Qu.:2.00
                  Median :4.00
                  Mean   :4.35
                  3rd Qu.:6.25
                  Max.   :9.00
```

Note that the summary does not count the numbers of cases of TRUE and FALSE in the logical variable called Damp, but it does count the number of cases of each Vegetation type. Note that the Field Names are not summarized, because they have been declared to be the row names (and hence all the names have to be unique).

Selecting parts of a dataframe: Subscripts

To select just the first 3 columns of the data frame, use subscript [,1:3] . The "blank then comma" means "all the rows", just as the "comma then blank" means all the columns. So to select all the rows of the first 3 columns of worms, we write:

worms[,1:3]

```
                  Area Slope Vegetation
Nash's.Field       3.6    11  Grassland
Silwood.Bottom     5.1     2     Arable
Nursery.Field      2.8     3  Grassland
Rush.Meadow        2.4     5     Meadow
Gunness'.Thicket   3.8     0      Scrub
Oak.Mead           3.1     2  Grassland
Church.Field       3.5     3  Grassland
Ashurst            2.1     0     Arable
The.Orchard        1.9     0    Orchard
Rookery.Slope      1.5     4  Grassland
Garden.Wood        2.9    10      Scrub
North.Gravel       3.3     1  Grassland
South.Gravel       3.7     2  Grassland
Observatory.Ridge  1.8     6  Grassland
Pond.Field         4.1     0     Meadow
Water.Meadow       3.9     0     Meadow
Cheapside          2.2     8      Scrub
Pound.Hill         4.4     2     Arable
Gravel.Pit         2.9     1  Grassland
Farm.Wood          0.8    10      Scrub
```

To select just the middle 10 rows of the data frame, use subscript [5:15,]

worms[5:15,]

```
                  Area Slope Vegetation Soil.pH  Damp Worm.density
Gunness'.Thicket   3.8     0      Scrub     4.2 FALSE            6
Oak.Mead           3.1     2  Grassland     3.9 FALSE            2
Church.Field       3.5     3  Grassland     4.2 FALSE            3
Ashurst            2.1     0     Arable     4.8 FALSE            4
The.Orchard        1.9     0    Orchard     5.7 FALSE            9
Rookery.Slope      1.5     4  Grassland     5.0  TRUE            7
Garden.Wood        2.9    10      Scrub     5.2 FALSE            8
North.Gravel       3.3     1  Grassland     4.1 FALSE            1
South.Gravel       3.7     2  Grassland     4.0 FALSE            2
Observatory.Ridge  1.8     6  Grassland     3.8 FALSE            0
Pond.Field         4.1     0     Meadow     5.0  TRUE            6
```

It is often useful to select certain rows, based on the logical tests on the values of one or more variables. Here is the code to select only those rows with Area > 3 and Slope < 3:

worms[Area>3 & Slope <3,]

```
                 Area Slope Vegetation Soil.pH  Damp Worm.density
Silwood.Bottom    5.1     2     Arable     5.2 FALSE            7
Gunness'.Thicket  3.8     0      Scrub     4.2 FALSE            6
Oak.Mead          3.1     2  Grassland     3.9 FALSE            2
North.Gravel      3.3     1  Grassland     4.1 FALSE            1
South.Gravel      3.7     2  Grassland     4.0 FALSE            2
Pond.Field        4.1     0     Meadow     5.0  TRUE            6
Water.Meadow      3.9     0     Meadow     4.9  TRUE            8
Pound.Hill        4.4     2     Arable     4.5 FALSE            5
```

Sorting

You can sort the rows or the columns of the data frame in any way you choose, but you need to state which columns you want to be sorted (typically you want them all sorted; i.e. columns 1:6 in the present case). Suppose we want the rows of the whole data frame sorted by Area (this is the first column [,1]):

```
worms[order(worms[,1]),1:6]
```

```
                  Area Slope Vegetation Soil.pH  Damp Worm.density
Farm.Wood          0.8    10      Scrub     5.1  TRUE            3
Rookery.Slope      1.5     4  Grassland     5.0  TRUE            7
Observatory.Ridge  1.8     6  Grassland     3.8 FALSE            0
The.Orchard        1.9     0    Orchard     5.7 FALSE            9
Ashurst            2.1     0     Arable     4.8 FALSE            4
Cheapside          2.2     8      Scrub     4.7  TRUE            4
Rush.Meadow        2.4     5     Meadow     4.9  TRUE            5
Nursery.Field      2.8     3  Grassland     4.3 FALSE            2
Garden.Wood        2.9    10      Scrub     5.2 FALSE            8
Gravel.Pit         2.9     1  Grassland     3.5 FALSE            1
Oak.Mead           3.1     2  Grassland     3.9 FALSE            2
North.Gravel       3.3     1  Grassland     4.1 FALSE            1
Church.Field       3.5     3  Grassland     4.2 FALSE            3
Nash's.Field       3.6    11  Grassland     4.1 FALSE            4
South.Gravel       3.7     2  Grassland     4.0 FALSE            2
Gunness'.Thicket   3.8     0      Scrub     4.2 FALSE            6
Water.Meadow       3.9     0     Meadow     4.9  TRUE            8
Pond.Field         4.1     0     Meadow     5.0  TRUE            6
Pound.Hill         4.4     2     Arable     4.5 FALSE            5
Silwood.Bottom     5.1     2     Arable     5.2 FALSE            7
```

or in descending order by Soil pH, with only Field Name and Worm.density as output

```
worms[rev(order(worms[,4])),c(4,6)]
```

```
                  Soil.pH Worm.density
The.Orchard           5.7            9
Garden.Wood           5.2            8
Silwood.Bottom        5.2            7
Farm.Wood             5.1            3
Pond.Field            5.0            6
Rookery.Slope         5.0            7
Water.Meadow          4.9            8
Rush.Meadow           4.9            5
Ashurst               4.8            4
Cheapside             4.7            4
Pound.Hill            4.5            5
Nursery.Field         4.3            2
Church.Field          4.2            3
Gunness'.Thicket      4.2            6
North.Gravel          4.1            1
```

```
Nash's.Field          4.1           4
South.Gravel          4.0           2
Oak.Mead              3.9           2
Observatory.Ridge     3.8           0
Gravel.Pit            3.5           1
```

Here is a simple example to demonstrate the beauty of the order function

```
houses <- read.table("c:\\temp\\houses.txt",header=T)
names(houses)
attach(houses)
houses
```

```
      Location Price
 1       Ascot   325
 2 Sunninghill   201
 3   Bracknell   157
 4   Camberley   162
 5     Bagshot   164
 6     Staines   101
 7     Windsor   211
 8  Maidenhead   188
 9     Reading    95
10   Winkfield   117
11    Warfield   188
12     Newbury   121
```

To see how this works, let's inspect the order of the house prices:

```
order(Price)
```

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

This says that the lowest price is in subscript 9 of (price = 95) where Location = Reading. Next cheapest is in location 6 (price = 101) where Location = Staines, and so on. The beauty of it is that we can use order(Price) as a subscript for Location to obtain the price-ranked list of locations:

```
Location[order(Price)]
```

```
 [1] Reading     Staines     Winkfield   Newbury
 [5] Bracknell   Camberley   Bagshot     Maidenhead
 [9] Warfield    Sunninghill Windsor     Ascot
```

When you see it used like this, you can see exactly why the function is called 'order'. If you want to reverse the order, just use the **rev** function like this:

```
Location[rev(order(Price))]
```

```
 [1] Ascot       Windsor     Sunninghill Warfield
 [5] Maidenhead  Bagshot     Camberley   Bracknell
 [9] Newbury     Winkfield   Staines     Reading
```

Subscripts with vectors

The use of subscripts in S Plus is superb. It is so simple, yet so powerful. There are basically two things you would want to do with subscripts:

- select values from a vector according to some logical criterion (i.e. questions involving the **contents** of the vector)

- discover which elements of a vector (i.e. which subscripts) contain certain values (i.e. questions involving the **addresses** of items within the vector)

Operation	Meaning
max(x)	maximum value in x
min(x)	minimum value in x
sum(x)	total of all the values in x
mean(x)	arithmetic average of the values in x
median(x)	median value in x
range(x)	vector of min(x) and max(x)
var(x)	sample variance of x, with degrees of freedom = length(x)-1
cor(x,y)	correlation between vectors x and y
sort(x)	a sorted version of x
rank(x)	vector of the ranks of the values in x
order(x)	an integer vector containing the permutation to sort x into ascending order
quantile(x)	vector containing the minimum, lower quartile, median, upper quartile, and maximum of x
cumsum(x)	vector containing the sum of all of the elements to that point.
cumprod(x)	vector containing the product of all of the elements to that point.
cummax(x)	vector of non-decreasing numbers which are the cumulative maxima of the values in x to that point.
cummin(x)	vector of non-increasing numbers which are the cumulative minima of the values in x to that point.
pmax(x,y,z)	vector, of length equal to the longest of x,

	y, or z containing the maximum of x, y or z for the ith position in each
pmin(x,y,z)	vector, of length equal to the longest of x, y, or z containing the minimum of x, y or z for the ith position in each
colMeans(x)	column means of data frame x
ColSums	column totals of data frame x
ColVars	column variances of data frame x
RowMean	row means of data frame x
RowSums	row totals of data frame x
RowVars	row variances of data frame x
peaks(x)	logical vector of length(x) with T for peak values bigger than both their neighbours

In working with vectors, it is important to understand the distinction between

- the logical status of a number (True or False)
- the value of a number (0.64 or 2541)

Take the example of a vector containing the 11 numbers 0 to 10

```
x<-0:10
```

There are two quite different things we might want to do with this. We might want to *add up* the values of the elements:

```
sum(x)
```

```
[1] 55
```

Alternatively we might want to *count* the elements that passed some logical criterion. Suppose we wanted to know how many of the values were less than 5:

```
sum(x<5)
```

```
[1] 5
```

You see the distinction. We use the vector function **sum** in both cases. But sum(x) adds up the values of the *x*'s and sum(x<5) counts up the number of cases that pass the logical condition "*x* is less than 5".

That is all well and good, but how do you add up the values of just some of the elements of x? We specify a logical condition, but we don't want to count the number of cases that pass the condition, we want to add up all the values of the cases that pass. This is the final

piece of the jigsaw, and involves the use of *logical subscripts*. Note that when we counted the number of cases, the counting was applied to the entire vector, using sum(x<5). To find the sum of the x values that are less than 5, we write:

```
sum(x[x<5])
```

```
[1] 10
```

The logical condition x<5 is either true or false.

```
x<5
```

```
 [1] T T T T T F F F F F F
```

You can imagine false as being numeric zero and true as being numeric one. Then the vector of subscripts [x<5] is 5 1's followed by 6 0's.

```
1*(x<5)
```

```
[1] 1 1 1 1 1 0 0 0 0 0 0
```

Now imagine multiplying the values of x by the values of the logical vector

```
x*(x<5)
```

```
 [1] 0 1 2 3 4 0 0 0 0 0 0
```

When the function **sum** is applied, it gives us the answer we want: the sum of the values of the numbers 0 + 1 + 2 + 3 + 4 = 10.

```
sum(x*(x<5))
```

```
[1] 10
```

The same answer as sum(x[x<5]), but rather less elegant.

Suppose we want to work out the sum of the 3 largest values in a vector. There are two steps: first sort the vector into descending order. Then add up the values of the first 3 elements of the sorted array. Let's do this in stages. First, the values of y:

```
y<-c(8,3,5,7,6,6,8,9,2,3,9,4,10,4,11)
```

Now if you apply **sort** to this, the numbers will be in ascending sequence, and this makes life a bit harder for the present problem:

```
sort(y)

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

We can use the **rev**erse function like this (use the Up Arrow key to save typing):

```
rev(sort(y))

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

So the answer to our problem is 11 + 10 + 9 = 30. But how to compute this? We can use specific subscripts to discover the contents of any element of a vector. We can see that 10 is the second element of the sorted array. To compute this we just specify the subscript [2]

```
rev(sort(y))[2]

[1] 10
```

A range of subscripts is simply a series generated using the colon operator. We want the subscripts 1 to 3, so this is:

```
rev(sort(y))[1:3]

[1] 11 10  9
```

So the answer to the exercise is just

```
sum(rev(sort(y))[1:3])

[1] 30
```

Note that we have not changed the vector y in any way, nor have we created any new space-consuming vectors during intermediate computational steps.

Addresses

There are two important functions for finding addresses within arrays. The function **which** is very easy to understand. The vector y looks like this:

```
y<-c ( 8,  3,  5,  7,  6,  6,  8,  9,  2,  3,  9,  4, 10,  4, 11 )
```

Suppose we wanted to know which elements of y contained values bigger than 5 we type

```
which(y>5)
```

```
[1]  1  4  5  6  7  8 11 13 15
```

Notice that the answer to this enquiry is *a set of subscripts*. We *don't* use subscripts in the **which** directive itself. The function is applied to the whole array. To see the *values of y* that are larger than 5 we just type

```
y[y>5]
```

```
[1]  8  7  6  6  8  9  9 10 11
```

If we want to know whether or not each element of y is bigger than 5 we just do this:

```
y>5
```

```
 [1] TRUE  FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE
[15] TRUE
```

and the answer is a logical vector of TRUE's and FALSE's.

Sample

This function *shuffles the contents of a vector* into a random sequence while maintaining all the numerical values intact. It is extremely useful in bootstrapping, simulation and Monte Carlo techniques of computationally intensive hypothesis testing. Here is the original y again:

```
y
 [1]  8  3  5  7  6  6  8  9  2  3  9  4 10  4 11
```

and here are 2 samples of y

```
sample(y)
```

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

```
sample(y)
```

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

This is "sampling without replacement": all the values appear once and only once in the shuffled list. The option **replace=T** allows for sampling *with replacement*, which is the basis of bootstrapping (see Practical 12). The vector produced by the **sample** function is the same length as the vector sampled, but some values are left out at random and other values, again at random, appear 2 or more times. The values selected will be different each time that sample is invoked.

```
sample(y,replace=T)
```

```
 [1]  9  6 11  2  9  4  6  8  8  4  4  4  3  9  3
```

In this sample, 10 has been left out, and are now three 9's.

```
sample(y,replace=T)
```

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

while in this one, there are two 10's and only one 9.

Logical arithmetic

A really useful computing skill involves the use of logical statements in arithmetic calculations. This takes advantage of the fact that logical "true" evaluates to 1.0 and logical "false" evaluates to 0.0. When we make a command only involving logic we get back a logical statement, T or F

Note that 'equals' is handled rather oddly in logical statements. We need to use "==" (double equals) to mean 'exactly equal to'. Suppose we want to add 1 to the value of y if y = 3 but otherwise leave y unaltered:

```
y
```

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

```
y+(y==3)
```

```
 [1]  8  4  5  7  6  6  8  9  2  4  9  4 10  4 11
```

It is often important to make one or more shorter versions of an array, and logical subscripting is a superb way of doing this. Let's make an array *ys* containing the small values of *y*, and *yb* containing the big values.

```
ys<-y[y<5]
ys

[1] 3 2 3 4 4

yb<-y[y>=5]
yb

 [1]  8  5  7  6  6  8  9  9 10 11
```

We can count the number of elements in an array that have various (potentially quite complicated) properties. For example we can count the number of extreme values in y, defining extreme values as those more than 2 greater than the mean and those less that 2 below the mean (there are 9 of them in this case)

```
sum(y> mean(y)+2 | y < mean(y)-2 )

 [1] 9
```

Note the use of the 'or operator', | , to separate the two parts of the expression. If we want the total of the y values that are extreme in one direction or the other we type

```
sum(y[y> mean(y)+2 | y < mean(y)-2] )

[1] 55
```

Do you see the distinction between these two operations? The first one evaluates as TRUE or FALSE whereas the second evaluates as numerical values of y.

If

Suppose that you wanted to replace all of the negative values in an array by zeros. In the old days, you might have written something like this;

```
for (i in 1:length(y))  { if(y[i] < 0 ) y[i] <- 0 }
```

Now, however, you would use logical subscripts like this

```
y[ y< 0 ] <- 0
```

Ifelse

Sometimes you want to do one thing if a condition is true and a different thing if the condition is false (rather than do nothing, as in the last example). The **ifelse** function allows you to do this for entire vectors without using **for** loops. We might want to replace any negative numbers by –1 and any positive values and zero by +1:

```
y <- ifelse( y < 0 , -1, 1 )
```

Trimming vectors using [minus]

Individual subscripts are referred to in square brackets. So if x is like this:

```
x<- c(5,8,6,7,1,5,3)
```

we can find the 4^{th} element of the vector just by typing

```
x[4]
```

```
[1] 7
```

An extremely useful facility is to use negative subscripts. These drop terms from a vector. Suppose we wanted a new vector, y, to contain everything but the first element of x

```
y<-x[-1]
y
[1] 8 6 7 1 5 3
```

This facility allows some extremely useful things to be done. Suppose we want to calculate a trimmed mean of x which ignores both the smallest and largest values (i.e. we want to leave out the 1 and the 8). There are two steps to this. First we **sort** the vector x. Then we remove the first element using x[-1] and the last using x[-length(x)] . We can do both drops at the same time by concatenating both instructions like this: -c(1,length(x)). Then we use the built-in function mean.

```
trim.mean <- function (x) mean(sort(x)[-c(1,length(x))])
```

Now try it out. The answer should be **mean**(5,6,7,5,3) = 26/5 = 5.2

```
trim.mean(x)
```

```
[1] 5.2
```

Summary of vectors

The rule is this. If you *can* use vectorized functions then *do*. Loops should be a last resort. You need to use them when you do something different to each element of an object, but when you do the same thing to each element, use subscripts or built in vector functions. It is a good idea to go through Table 2 and make sure that you understand when you might want to use each of the vector functions listed there.

Plotting mathematical functions

It is easy to plot mathematical functions in order to see how their shape depends on the values of their parameters. You need to know the equation relating y to x and you need to generate a set of values for the x variable. Suppose we want to draw the following asymptotic exponential function:

$$y = a(1 - e^{-bx})$$

for the particular parameter values $a = 3$ and $b = 0.1$. First we need to generate a vector of x values (between 30 and 50 values is all you need for producing smooth-looking curves). You need to know the range of x values. In this case we want the x axis to go from 0 to 50. To generate the x values all we do is this:

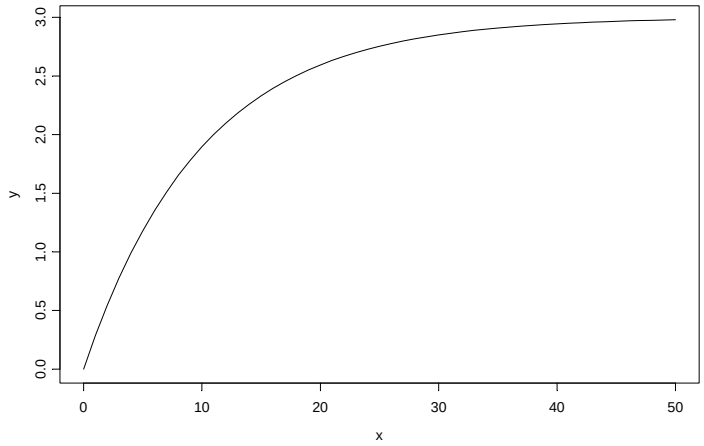
```
x<-0:50
```

This says x gets the values 0 to 50 in steps of 1. Now the values of y are obtained by evaluating the equation for every value of x

```
y<-3*(1-exp(-0.1*x))
```

If we were to say **plot**(x,y) we would get a scatterplot, rather than a smooth function. To get the desired result we just change the plot **type** to "l" which stands for "line" (note that the symbol is a lower case L and not a numeric 1):

```
plot(x,y,type="l")
```



Another thing we might want to do is to plot a family of curves showing their behaviour with different parameter values. Let us plot a family of 2-parameter logistic curves

$$y = \frac{e^{a+bx}}{1+e^{a+bx}}$$

for different values of b. It is worth working out the extreme values of y if we are going to produce multiple plots, because we do not want subsequent plots to have y values that are out of range when compared to the first plot. Try putting $x = -\infty$. The numerator is $\exp(-\infty)$ which is zero. The denominator is 1+0 = 1. So y is 0 when x is minus infinity. What about x is plus infinity. Now both the numerator and the denominator are plus infinity, so y should be 1. We shall scale the y axis to go from 0 to 1 in our first plot directive. We shall use the same range of x values for each plot (-5 to 5) so we do not need to make any adjustments to **xlim**. Because we want to have an increment of 0.1 in the x values (rather than 1 as in the last example), we use **seq** to generate our sequence of x values:

```
x<-seq(-5,5,.1)
```

The first plot we want to see has a = 0.1 and b = 0.4. So we calculate the y values

```
y<-exp(.1+.4*x)/(1+exp(.1+.4*x))
```

Now we make the first plot, bearing in mind 2 things: we need to make the plot **type** a line, and we need to scale the y axis from 0 to 1 using **ylim**

```
plot(x,y,type="l",ylim=c(0,1))
```

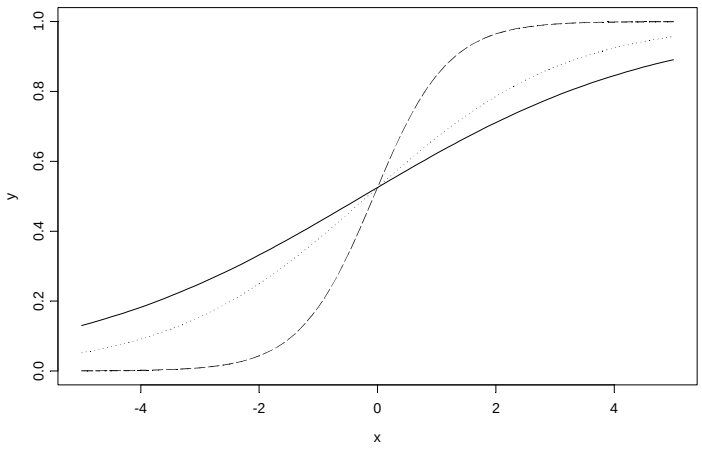
On these same axes we now want to overlay another plot (y2) with a = 0.1 and b = 0.6 Use Up Arrow to edit the new values into the original equation:

```
y2<-exp(.1+.6*x)/(1+exp(.1+.6*x))
```

Overlaying new functions is done with the lines directive; we use a different line type **lty**

```
lines(x,y2,lty=2)
```

The final curve (y3) has $a = 0.1$ and $b = 1.6$




```
y3<-exp(.1+1.6*x)/(1+exp(.1+1.6*x))
lines(x,y3,lty=4)
```

Matrices

Matrix arithmetic is handled in an intuitively obvious way. A matrix is defined with the **matrix** directive. Suppose we want to turn a new vector *y* of length 15

```
y<-c( 8,  3,  5,  7,  6,  6,  8,  9,  2,  3,  9,  4, 10,  4, 11)
```

```
y
```

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

into a matrix called *m* consisting of 5 rows and 3 columns

```
m<-matrix(y,nrow=5)
```

```
m
```

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

Another way to make matrices is to **bind** vectors together into matrices: row by row using **rbind**, or column by column using **cbind**. We'll meet this is a later practical.

Matrix arithmetic

It is important to understand that the * operator does **not** carry out matrix multiplication. This requires the %*% operator. Consider this example where a Leslie matrix, L, is to be multiplied by a column matrix of population sizes, n

```
L<-c(0,0.7,0,0,6,0,0.5,0,3,0,0,0.3,1,0,0,0)
```

```
L<-matrix(L,nrow=4)
```

Note that the elements of the matrix are entered in column-wise, not row-wise sequence. We make sure that the Leslie matrix is properly conformed:

```
L
```

```
     [,1] [,2] [,3] [,4]
[1,]  0.0  6.0  3.0    1
[2,]  0.7  0.0  0.0    0
[3,]  0.0  0.5  0.0    0
[4,]  0.0  0.0  0.3    0
```

The top row contains the age-specific fecundities, and the sub-diagonal contains the survivorships. Now the population sizes at each age go in a column vector, n.

```
n<-c(45,20,17,3)
n<-matrix(n,ncol=1)
n
```

```
     [,1]
[1,]   45
[2,]   20
[3,]   17
[4,]    3
```

Population sizes next year in each of the 4 age classes are obtained by matrix multiplication, %*%

```
L %*% n
```

```
      [,1]
[1,] 174.0
[2,]  31.5
[3,]  10.0
[4,]   5.1
```

We can check this out longhand. The number of juveniles next year (the 1st element of n) is the sum of all the babies born last year:

```
45*0+20*6+17*3+3*1
```

```
[1] 174
```

So that's OK. If you try to do ordinary multiplication with L and n you will fail because their dimensions do not match:

```
L*n
```

```
Error in L * n : non-conformable arrays
```

Matrix multiplication %*% on vectors of the same length returns the *vector dot product* (the sum of the pairwise products) as a 1x1 matrix.

Solving systems of linear equations

Suppose we have 2 equations containing 2 unknown variables:

$$3x + 4y = 12$$

$$x + 2y = 8$$

We can use the function **solve** to find the values of the variables if we provide it with 2 matrices:

- a square matrix **A** containing the coefficients
- a column vector **kv** contaiaing the known values

We set the two matrices up like this (column-wise)

```
A<-matrix(c(3,1,4,2),nrow=2)
```

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

```
kv<-matrix(c(12,8),nrow=2)
```

```
kv
      [,1]
[1,]   12
[2,]    8
```

Now we can solve the simultaneous equations

```
solve(A,kv)
```

```
      [,1]
[1,]   -4
[2,]    6
```

so x = -4 and y = 6 (as you can easily verify by hand). The function comes into its own when there are many simultaneous equations to be solved.

Table 3. Matrix operations

Operation	Meaning
solve(x)	inverse of matrix x
solve(x,y)	solution of simultaneous linear equations with coefficients x and known values y
backsolve(x,y)	solves a system of linear equations when the square matrix x is upper triangular ,y is the matrix containing the right-hand sides to equations (the known values)
eigen(x)	eigenvalues and eigenvectors of a square matrix x
diag(x)	diagonal of the matrix x
sum(diag(x))	trace of square matrix x
prod(eigen(x)$values)	determinant of real-valued matrix x
svd(x)	a list containing the singular value decomposition of x
qr(x)	a representation of an orthogonal (or unitary) matrix and a triangular matrix whose product is the input x
chol(x)	an upper-triangular matrix which is the Choleski decomposition of a (hermitian) symmetric, positive definite (or positive semi-definite) matrix
kronecker(x,y)	A matrix with dimension the product of the dimensions of the input matrices x & y; each element of x is replaced by that element times the entire matrix y
t(x)	transpose of the matrix x

STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

3. STATISTICS OF ONE AND TWO SAMPLES

The hardest part of any statistical work is knowing how to get started. One of the hardest things about getting started in choosing the right kind of statistical analysis for your data and your particular scientific question. The truth is that there is no substitute for experience. The way to know what to do is to have done it properly lots of times before. But here are some useful guidelines. It is essential, for example to know:

1) Which of your variables is the response variable?

2) Which are the explanatory variables?

3) Are the explanatory variables continuous or categorical, or a mixture of both?

4) What kind of response variable have you got: is it a continuous measurement, a count, a proportion, a time-at-death or a category ?

The answers to these questions should lead you quickly to the appropriate choice if you use the following dichotomous key. It begins by asking questions about the nature of your explanatory variables, and ends by asking about what kind of response variable you have got.

1. Explanatory variables all categorical		**2**
At least one explanatory variable a continuous measurement		**4**
2. Response variable a count	**Contingency table**	
Response variable not a count		**3**
3. Response variable a continuous measurement	**Analysis of variance**	
Response variable other than this	**Analysis of deviance**	
4. All explanatory variables continuous	**Regression**	**5**
Explanatory variables both continuous and categorical	**Analysis of covariance**	**5**
5. Response variable continuous	**Regression** or **Ancova**	
Response variable a count	**Log-linear** models (Poisson errors)	
Response variable a proportion	**Logistic** model (binomial errors)	
Response variable a time-at-death	**Survival analysis**	
Response variable binary	**Binary** logistic analysis	
Explanatory variable is time	**Time series** analysis	

Estimating parameters from data

Data have a number of important properties, and it is useful to be able to quantify the following attributes:

- sample size
- central tendency
- variation
- skew
- kurtosis

The sample size is easy: the number of measurements of the response variable is known universally as *n*. In the words of the old statistical joke: "It's the *n*'s that justify the means". Determining what sample size you *need* in order to address a particular question in specific circumstances is an important question that we deal with later. In R, you find the size of a vector using length(name).

Central tendency

Most data show some propensity to cluster around a central value. Averages from repeated bouts of sampling show a remarkable tendency to cluster around the arithmetic mean (this is the Central Limit Theorem; see below). There are several ways of estimating the central tendency of a set of data and they often differ from one another. These differences can be highly informative about the nature of the data set.

Mode

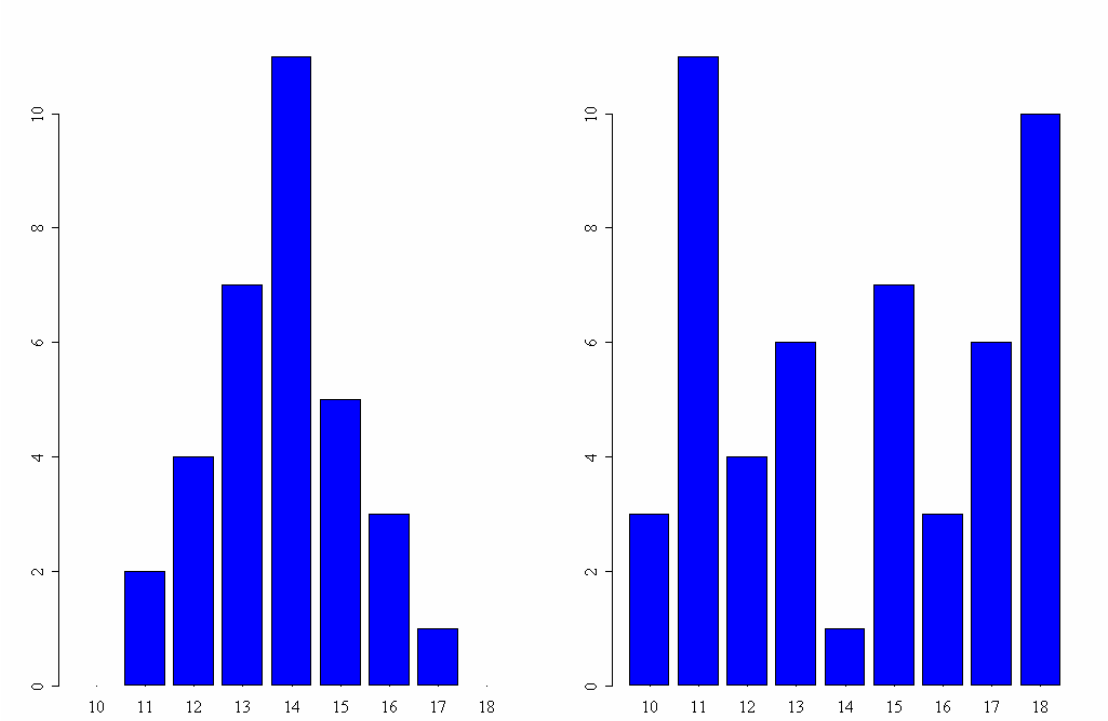
Perhaps the simplest measure of central tendency is the mode: the most frequently represented class of data. Some distributions have several modes, and these can give a poor estimate of central tendency.

```
rm(x)    # this deletes any existing vector called x
distribution<-read.table("c:\\temp\\mode.txt",header=T)
attach(distribution)
names(distribution)
```

```
[1] "fx" "fy"  "fz" "x"
```

To draw the two histograms side by side we set up 2 plotting panels:

```
par(mfrow=c(1,2))
barplot(fx,names=as.character(x))
barplot(fy,names=as.character(x))
```



The mode is an excellent descriptor of the central tendency of the data set on the left, but the two largest modes of the data set on the right are both very poor descriptors of its central tendency.

Median

The median is an extremely appealing measure of central tendency. It is the value of the response variable that lies in the middle of the ranked set of y values. That is to say 50% of the y values are smaller than the median, and 50% are larger. It has the great advantage of not being sensitive to outliers. Suppose we have a vector, y, that contains all of the x values in our left hand histogram. That is to say each *x* value is **rep**eated *fx* times:

```
y<-rep(x,fx)
```

Now to find the median from first principles we **sort** the y values (in this case they are already in ascending order, but we overlook this for the sake of generality).

```
y<-sort(y)
```

Now we need to know how many y values (n) there are; use the **length** directive for this

```
length(y)
```

```
[1] 33
```

Because this is an odd number it means that the median is uniquely determined as the y value in position 17 of the ranked values of y.

```
ceiling(length(y)/2)
```

```
[1] 17
```

We find this value of y using subcripts [].

```
y[17]
```

or, more generally, using the Up Arrow to edit in y[ ] to the **ceiling** directive

```
y[ceiling(length(y)/2)]
```

```
[1] 14
```

Had the vector been of even-numbered length, we could have taken the y values on either side of the mid point (y[length/2] and y[length/2 + 1]) and averaged them. A function to do this is described later). You will not be surprised to learn that there is a built in **median** function, used directly like this:

```
median(y)
```

```
[1] 14
```

Arithmetic mean

The most familiar measure of central tendency is the arithmetic mean. It is the sum of the *y* values divided by the number of *y* values (*n*). Throughout the book we use $\sum$, capital Greek sigma, to mean "add up all the values of". So the mean of *y*, called "y bar", can be written as

$$\bar{y} = \frac{\sum y}{n}$$

So we can compute this using the **sum** and **length** functions

```
sum(y)/length(y)
```

```
[1] 13.78788
```

but you will be not be surprised to learn that there is a built-in function called **mean**

```
mean(y)
```

```
[1] 13.78788
```

Notice that in this case, the arithmetic mean is very close to the median (14.0).

Geometric mean

This is much less familiar than the arithmetic mean, but it has important uses in the calculation of average rates of change in economics and average population sizes in ecology. Its definition appears rather curious at first. It is the n^{th} root of the product of the y values. Just as sigma means "add up", so capital Greek pi, $\prod$, means multiply together. So the geometric mean "y curl", is

$$\tilde{y} = \sqrt[n]{\prod y}$$

Recall that roots are fractional powers, so that the square root of x is $x^{1/2} = x^{0.5}$. So to find the geometric mean of y we need to take the product of all the y values, prod(y), to the power of 1/(length of y), like this:

```
prod(y)^(1/length(y))
```

```
[1] 13.71531
```

As you see, in this case the geometric mean is close to both the arithmetic mean (13.788) and the median (14.0). A different way of calculating the geometric mean is to think about the problem in terms of logarithms. The "log of times is plus", so the log of product of the y values is the sum of the logs of the y values. All the logs used in this book are natural logs (base e = 2.781828) unless stated otherwise (see below). So we could work out the mean of log(y) then calculate its antilog:

```
meanlogy<-sum(log(y))/length(y)
meanlogy
```

```
[1] 2.618513
```

Now you need to remember (or learn!) that the natural antilog function is **exp.** So the antilog of meanlogy is

```
exp(meanlogy)
```

```
[1] 13.71531
```

as obtained earlier by a different route. There is no built-in function for geometric mean, but we can easily make one like this:

```
geometric<-function(x)  exp(sum(log(x))/length(x))
```

log means "log to the base e". Then try the function out using our vector called y:

```
geometric(y)
```

```
[1] 13.71531
```

The reason that the geometric mean is so important can be seen by using a more extreme data set. Suppose we had made the following counts of aphids on different plants:

```
aphid<-c(10,1,1,10,1000)
```

Now the arithmetic mean is hopeless as a descriptor of central tendency:

```
mean(aphid)
```

```
[1] 204.4
```

This measure isn't even close to *any* of the 5 data points. The reason for this, of course, is that the arithmetic mean is so sensitive to outliers, and the single count of 1000 has an enormous influence on the mean.

We can break the rules, and just once use logs to the base 10 to look at this example. The $\log_{10}$ values of the 5 counts are 1, 0, 0, 1 and 3. So the sum of the logs is 5 and the average of the logs is 5/5 = 1. Antilog base 10 of 1 is $10^1 = 10$. This is the geometric mean of these data, and is a much better measure of central tendency (2 of the 5 values are exactly like this) We can check using our own function:

```
geometric(aphid)
```

```
[1] 10
```

Needless to say, we get the same answer, whatever base of logarithms we use. We always use geometric mean to average variables that change multiplicatively, like ecological populations or bank accounts accruing compound interest.

Harmonic mean

Suppose you are working on elephant behaviour. Your focal animal has a square home range with sides of length 2 km. He walks the edge of the home range as follows. The first leg of the journey is carried out at a stately 1 km/hour. He accelerates over the second side to 2 km/hour. He is really into his stride by the 3rd leg of the journey, walking at a dizzying 4 km/hour. Unfortunately, this takes so much out of him that he has to return home at a sluggish 1 km/hour.

The question is this. What is his average speed over the ground. He ended up where he started, so he has no net displacement. But our concern is with his mean velocity. What happens if we work out the average of his 4 speeds?

```
mean(c(1,2,4,1))
```

```
[1] 2
```

This is wrong, as we can see if we work out the average speed from first principles.

$$\text{velocity} = \frac{\text{distance travelled}}{\text{time taken}}$$

The total distance travelled is 4 sides of the square, each of length 2 km, a total of 8 km. Total time taken is a bit more tricky. The first leg took 2 hours (2km at 1 km/hour), the second look 1 hour, the third took half an hour and the last leg took another 2 hours. Total time taken was 2 + 1 + 0.5 + 2 = 5.5 hours. So the correct average velocity is

$$v = \frac{8}{5.5} = 1.455$$

What is the trick? The trick is that this question calls for a different sort of mean. The harmonic mean has a very long definition in words. It is *the reciprocal of the average of the reciprocals.* The reciprocal of x is $\frac{1}{x}$ which can also be written as x^{-1}. So the formula for harmonic mean, "y hat", looks like this:

$$\hat{y} = \frac{1}{\frac{\sum \frac{1}{y}}{n}} = \frac{n}{\sum \frac{1}{y}}$$

Lets try this out with the elephant data:

```
4/sum(1/c(1,2,4,1))
```

```
[1] 1.454545
```

So that works OK. Let's write a general function to compute the harmonic mean of any vector of numbers.

```
harmonic<-function(x) 1/mean(1/x)
```

We can try it out on our vector y:

```
harmonic(y)
```

```
[1] 13.64209
```

To summarise, our measures of central tendency all gave different values, but because the y values were well clustered and there were no serious outliers, all the different values were quite close to one another.

Mode	Median	Arithmetic mean	Geometric mean	Harmonic mean
14	14	13.7879	13.7153	13.6421

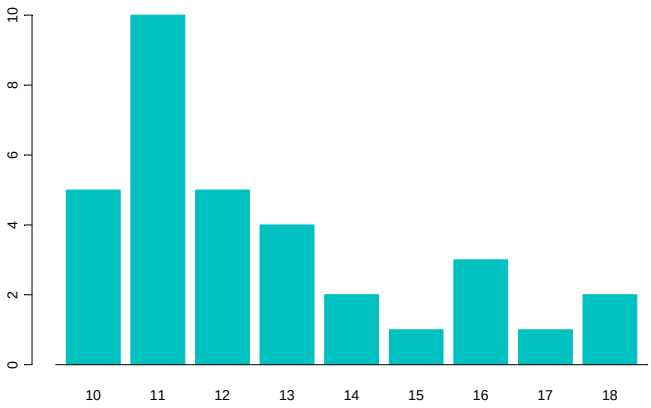
Notice that if you try typing mode(y) you don't get 14, you get the message

```
mode(y)

[1] "numeric"
```

because **mode** tells you **type** of an S-PLUS object, and y is of type numeric. A different mode is “character”.

The distribution of y was quite symmetric, but many data sets are skew to one side or the other. A long tail to the right is called positive skew, and this is much commoner in practice that data sets which have a long tail to the left (negative skew). The frequency distribution fz is an example of positive skew.

```
par(mfrow=c(1,1))
barplot(fz,names=as.character(x))
```



It is useful to know how the different measures of central tendency compare when the distribution is skew like this. First, we expand the frequency distribution into a vector of measurements, *w*, using **rep** to repeat each of the *x* values *fz* times

```
w<-rep(x,fz)
```

We want to produce a summary of all of the information we have gathered on central tendency for the data in the vector called *w*. For instance,

```
Arithmetic mean  =  12.606
Geometric mean   =  12.404
Harmonic mean    =  12.22
Median           =  12
Modes at 11 and 16 with the largest mode at x = 11
```

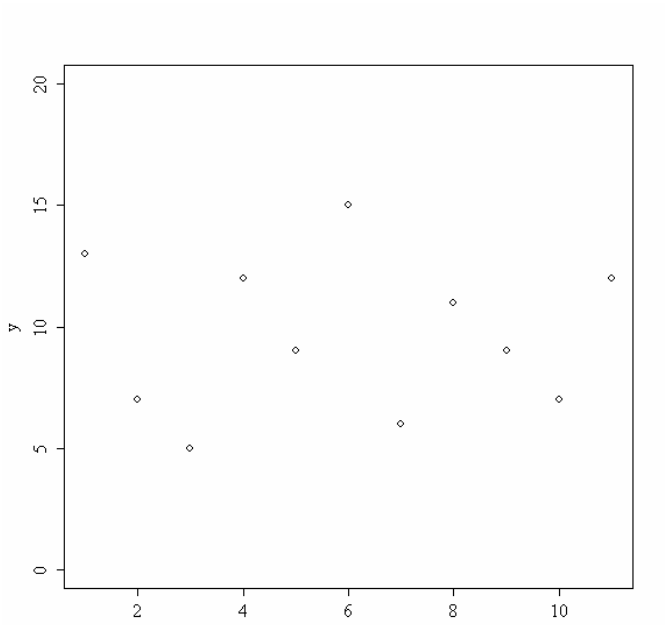
With positive skew, the *mode is the lowest* estimate of central tendency (11.0) and *arithmetic mean is the largest* (12.606). The others are intermediate, with geometric mean > harmonic mean > median in this case.

Measuring variation

A measure of variability is perhaps the most important quantity in statistical analysis. The greater the variability in the data, the greater will be our uncertainty in the values of parameters estimated from the data, and the lower will be our ability to distinguish between competing hypotheses about the data.

Consider the following data, *y*, which are simply plotted in the order in which they were measured:

```
y<-c(13,7,5,12,9,15,6,11,9,7,12)
plot(y,ylim=c(0,20))
```



Visual inspection indicates substantial variation in y. But how to measure it? One way would be to specify the range of *y* values.

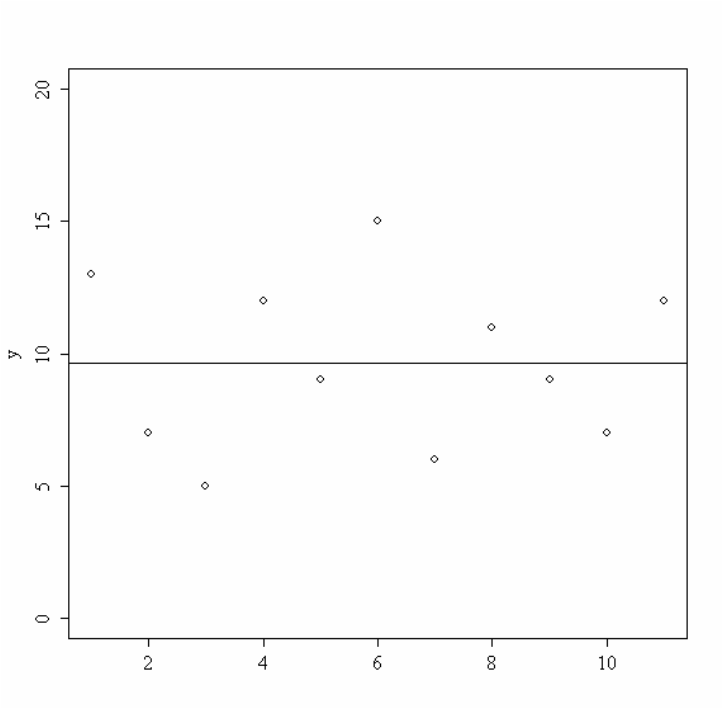
```
range(y)
```

```
[1]  5    15
```

The minimum value of *y* is 5 and the maximum is 15. The variability is contained within the range, and to that extent it is a very useful measure. But it is not ideal for general purposes. For one thing, it is totally determined by outliers, and gives us no indication of more typical levels of variation. Nor is it obvious how to use range in other kinds of calculations (e.g. in uncertainty measures). Finally, the range increases with the sample size, because if you add more numbers then eventually you will add one larger than the current maximum, and if you keep going you will find one smaller than the current minimum. This is a fact, but it is not a property of our unreliability estimate that we particularly want, We would like uncertainty to go down as the sample size went up.

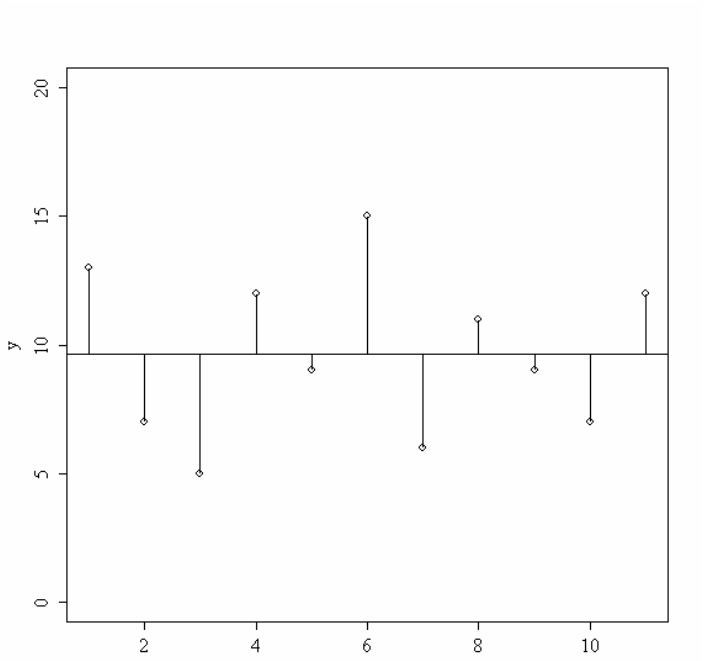
How about fitting the average value of *y* though the data and measuring how far each individual *y* value departs from the mean? The second parameter says the slope of the fitted line is zero:

```
abline(mean(y),0)
```



This divides the data into 5 points that are larger than the mean and 7 points that are smaller than the mean. The distance, *d*, of any point *y* from the mean $\bar{y}$ is

$$d = y - \bar{y}$$



Now a variety of possibilities emerge for measuring variability. How about adding together the different values of *d* ? This turns out to be completely hopeless, because the sum of the *d* values is always zero, no matter what the level of variation in the data !

$$\sum d = \sum (y - \bar{y})$$

Now $\sum \bar{y}$ is the same as $n.\bar{y}$ so

$$\sum d = \sum y - n\bar{y}$$

and we know that $\bar{y} = \sum y / n$ so

$$\sum d = \sum y - \frac{n\sum y}{n}$$

The n's cancel, leaving

$$\sum d \quad = \quad \sum y - \sum y \quad = \quad 0$$

So that's no good then. But wasn't this just a typical mathematician's trick, because the plus and minus values simply cancelled out? Why not ignore the signs and look at the sum of the absolute values of d ?

$$\sum |d| = \sum \left| y - \bar{y} \right|$$

This is actually a very appealing measure of variability because it does not give undue weight to outliers. It was spurned in the past because it made the sums much more difficult, and nobody wants that. It has had a new lease of life since computationally

intensive statistics have become much more popular. The other simple way to get rid of the minus signs is to square each of the *d*'s before adding them up.

$$\sum d^2 = \sum (y - \bar{y})^2$$

This quantity has a fantastically important role in statistics. Given its importance, you might have thought they would have given it a really classy name. Not so. It is called, with appalling literalness the "sum of squares". The sum of squares is the basis of all the measures of variability used in linear statistical analysis.

Is this all we need in our variability measure? Well not quite, because every time we add a new data point to our graph the sum of squares will get bigger. Sometimes by only a small amount when the new value is close to $\bar{y}$ but sometimes by a lot, when it is much bigger or smaller than $\bar{y}$. The solution is straightforward. We calculate the average of the squared deviations (also imaginatively known as the "mean square"). But there is a small hitch. We could not calculate $\sum d^2$ before we knew the value of $\bar{y}$. And how did we know the value of $\bar{y}$. Well we didn't know the value. We estimated it from the data. This leads us into a very important, but simple, concept.

Degrees of freedom

Suppose we had a sample of 5 numbers and their average was 4. What was the sum of the 5 numbers? It must have been 20, otherwise the mean would not have been 4. So now we think about each of the 5 numbers in turn.

We are going to put numbers in each of the 5 boxes. If we allow that the numbers could be positive or negative real numbers we ask how many values could the first number take. Once you see what I'm doing, you will realise it could take any value. Suppose it was a 2.

2				

How many values could the next number take ? It could be anything. Say it was a 7

2	7			

And the 3rd number. Anything. Suppose it was a 4.

2	7	4		

The 4th number could be anything at all. Say it was 0.

2	7	4	0	

Now then. How many values could the last number take? Just 1. It has to be another 7 because the numbers have to add up to 20.

2	7	4	0	7

To recap. We have total freedom in selecting the first number. And the second, third and fourth numbers. But no choice at all in selecting the fifth number. We have 4 degrees of freedom when we have 5 numbers. In general we have n-1 degrees of freedom if we estimated the mean from a sample of size n.

More generally still, we can propose a formal definition of degrees of freedom

Degrees of freedom is the sample size, n, minus the number of parameters, p, estimated from the data.

You should memorise this. In the example we just went through we had $n = 5$ and we had estimated just one parameter from the data: the sample mean, $\bar{y}$. So we had n-1 = 4 d.f.

In a linear regression we estimate two parameters from the data when we fit the model

$$y = a + bx$$

the intercept, *a*, and the slope *b*. Because we have estimated 2 parameters, we have n-2 d.f.. In a one way analysis of variance with 5 genotypes, we estimate 5 means from the data (one for each genotype) so we have n-5 d.f. And so on. The ability to work out degrees of freedom is an incredibly useful skill. It enables you to spot mistakes in experimental designs and in statistical analyses. It helps you to spot pseudoreplication in the work of others, and to avoid it in work of your own.

Variance

We are now in a position to define the most important measure of variability in all of statistics: a variance, s^2, is always

$$\text{variance} = \frac{\text{sum of squares}}{\text{degrees of freedom}}$$

In our case, working out the variance of a single sample, the variance is this:

$$s^2 = \frac{\sum (y - \bar{y})^2}{n - 1}$$

This is so important, we need to take stock at this stage. The data in the following table come from 3 market gardens. The data show the ozone concentrations in parts per hundred million (pphm) on ten summer days.

Garden A	Garden B	Garden C

3	5	3
4	5	3
4	6	2
3	7	1
2	4	10
3	4	4
1	3	3
3	5	11
5	6	3
2	5	10

We want to calculate the variance in ozone concentration for each garden. There are 4 steps to this

- determine the sample mean for each garden
- subtract the sample mean from each value
- square the differences and add them up to get the sum of squares
- divide the sum of squares by degrees of freedom to obtain the variance

We begin by typing the data for each garden into vectors called A, B and C:

```
A<-c(3,4,4,3,2,3,1,3,5,2)

B<-c(5,5,6,7,4,4,3,5,6,5)

C<-c(3,3,2,1,10,4,3,11,3,10)
```

and calculating the 3 sample means

```
mean(A)
[1] 3
mean(B)
[1] 5
mean(C)
[1] 5
```

Now we calculate vectors of length 10 containing the differences $y - \bar{y}$

```
dA <- A-3
dB <- B-5
dC <- C-5
```

then determine the **sum** of the squares of these differences

```
SSA<-sum(dA^2)
SSB<-sum(dB^2)
```

```
SSC<-sum(dC^2)
```

We find that SSA = 12, SSB = 12 and SSC = 128. The 3 variances are obtained by dividing the sum of squares by the degrees of freedom

```
s2A<-SSA/9
s2B<-SSB/9
s2C<-SSC/9
```

To see the values of the 3 variances we type their names separated by semicolons

```
s2A;s2B;s2C
```

```
[1] 1.333333
[1] 1.333333
[1] 14.22222
```

Of course there is a short-cut formula to obtain the sample variance directly

```
s2A<-var(A)
```

There are three important points to be made from this example:

- two populations can have different means but the same variance (Gardens A & B)
- two populations can have the same mean but different variances (Gardens B & C)
- comparing means when the variances are different is an extremely bad idea.

The first two points are straightforward. The third point is profoundly important. Let's look again at the data. Ozone is only damaging to lettuce crops at concentrations in excess of a threshold of 8 pphm. Now looking at the average ozone concentrations we would conclude that both gardens are the same in terms of pollution damage. Wrong ! Look at the data, and count the number of days of damaging air pollution in Garden B. None at all. Now count Garden C. Completely different, with damaging air pollution on 3 days out of 10.

The moral is clear. *If you compare means from populations with different variances you run the risk of making fundamental scientific mistakes.* In this case, concluding there was no pollution damage, when in fact there were damaging levels of pollution 30% of the time.

This begs the question of how you know whether or not 2 variances are significantly different. There is a very simple rule of thumb (the details are explained in Practical 5): if the larger variance is more than 4 times the smaller variance, then the 2 variances are significantly different. In our example here, the variance ratio is

```
14.2222 / 1.3333
[1] 10.66692
```

which is much greater than 4 which means that the variance in Garden C is significantly higher than in the other 2 gardens. Note in passing, that because the variance is the same in Gardens A and B, it is legitimate to carry out a significance test on the difference between their 2 means. This is Student's t-test, explained in full below.

Shortcut formulas for calculating variance

This really was a fairly roundabout sort of process. The main problem with the formula defining variance is that it involves all those subtractions, $y - \bar{y}$. It would be good to find a way of calculating the sum of squares that didn't involve all these subtractions. Let's expand the bracketed term $(y - \bar{y})^2$ to see if we can make any progress towards a subtraction-free solution.

$$(y - \bar{y})^2 \quad = \quad (y - \bar{y})(y - \bar{y}) \quad = \quad y^2 - 2y\bar{y} + \bar{y}^2$$

So far, so good. Now we put the summation through each of the 3 terms separately:

$$\sum y^2 - 2\bar{y}\sum y + n\bar{y}^2 \quad = \quad \sum y^2 - 2\frac{\sum y}{n}\sum y + n\left[\frac{\sum y}{n}\right]^2$$

where only the *y*'s take the summation sign, because we can replace $\sum \bar{y}$ by $n\bar{y}$. We replace $\bar{y}$ with $\sum y / n$ on the r.h.s. Now we cancel the n's and collect the terms

$$\sum y^2 - 2\frac{\left[\sum y\right]^2}{n} + n\frac{\left[\sum y\right]^2}{n^2} \quad = \quad \sum y^2 - \frac{\left[\sum y\right]^2}{n}$$

This gives us a formula for computing the sum of squares that avoids all the tedious subtractions. Actually, it is better computing practice to use the longhand method because it is less subject to rounding errors. Our shortcut formula could be wildly inaccurate if we had to subtract one very large number from another. All we need is the sum of the y's and the sum of the squares of the y's. It is very important to understand the difference between $\sum y^2$ and $\left[\sum y\right]^2$. It is worth doing a numerical example to make plain the distinction between these important quantities.
Let y be {1, 2, 3}. This means that $\sum y^2$ is {1 + 4 + 9} = 14.
The sum of the y's $\sum y$ is {1 + 2 + 3} = 6, so $\left[\sum y\right]^2 = 6^2 = 36$.

The square of the sum is always much larger than the sum of the squares.

Using variance

Variance is used in two main ways

- for establishing measures of unreliability
- for testing hypotheses

Consider the properties that you would like a measure of unreliability to possess. As the variance of the data increases what would happen to unreliability of estimated parameters ? Would it go up or down ? Unreliability would go up as variance increased, so we would want to have the variance on the top of any divisions in our formula for unreliability (i.e. in the numerator).

$$\text{unreliability} \quad \propto \quad s^2$$

What about sample size? Would you want your estimate of unreliability to go up or down as sample size, n, increased ? You would want unreliability to go down as sample size went up, so you would put sample size on the bottom of the formula for unreliability (i.e. in the denominator).

$$\text{unreliability} \quad \propto \quad \frac{s^2}{n}$$

Now consider the units in which unreliability is measured. What are the units in which our current measure are expressed. Sample size is dimensionless, but variance is based on the sum of squared differences, so it has dimensions of mean squared. So if the mean was a length in cm the variance would be an area in cm^2. This is an unfortunate state of affairs. It would make good sense to have the dimensions of the unreliability measure and the parameter whose unreliability it is measuring to be the same. That is why all unreliability measures are enclosed inside a big square root term. Unreliability measures are called *standard errors*. What we have just calculated is the standard error of the mean

$$SE_{\bar{y}} = \sqrt{\frac{s^2}{n}}$$

This is a very important equation and should be memorised. Let's calculate the standard errors of each of our market garden means:

```
sqrt(s2A/10)

[1] 0.3651484

sqrt(s2B/10)

[1] 0.3651484

sqrt(s2C/10)

[1] 1.19257
```

In written work one shows the unreliability of any estimated parameter in a formal, structured way like this.

"The mean ozone concentration in Garden A was 3.0 ± 0.365 (1 s.e., n = 10)"

You write plus or minus, then the unreliability measure then, in brackets, tell the reader what the unreliability measure is (in this case one standard error) and the size of the sample on which the parameter estimate was based (in this case, 10)). This may seem rather stilted, unnecessary even. But the problem is that unless you do this, the reader will not know what kind of unreliability measure you have used. For example, you might have used a 95% confidence interval or a 99% confidence interval instead of 1 s.e..

A confidence interval shows the likely range in which the mean would fall if the sampling exercise were to be repeated. It is a very important concept that people always find difficult to grasp at first. It is pretty clear that the confidence interval will get wider as the unreliability goes up, so

$$\text{confidence interval} \propto \text{unreliability measure} \propto \sqrt{\frac{s^2}{n}}$$

But what do we mean by "confidence" ? This is the hard thing to grasp. Ask yourself this question. Would the interval be wider or narrower if we wanted to be *more* confident that out repeat sample mean falls inside the interval ? It may take some thought, but you should be able to convince yourself that the more confident you want to be, the *wider* the interval will need to be. You can see this clearly by considering the limiting case of complete and absolute certainty. Nothing is certain in statistical science, so the interval would have to be infinitely wide. We can produce confidence intervals of different widths by specifying different levels of confidence. The higher the confidence, the wider the interval.

How exactly does this work. How do we turn the proportionality in the equation above into equality? The answer is by resorting to an appropriate theoretical distribution (see below). Suppose our sample size is too small to use the normal distribution (n < 30, as here), then we traditionally use Student's t distribution. The values of Student's t associated with different levels of confidence are tabulated but also available in the function **qt**, which gives the quantiles of the t distribution. Confidence intervals are always 2-tailed; the parameter may be larger or smaller than our estimate of it. Thus, if we want to establish a 95% confidence interval we need to calculate (or look up) Student's t associated with $\alpha = 0.025$ (i.e. with 0.01*(100 %– 95%)/2). The value is found like this for the left (0.025) and right (0.975) hand tails:

```
qt(.025,9)

[1] -2.262157

qt(.975,9)

[1] 2.262157
```

The first argument is the probability and the second is the degrees of freedom. This says that values as small as -2.262 standard errors below the mean are to be expected in 2.5% of cases (p = 0.025), and values as large as +2.262 standard errors above the mean with similar probability (p = 0.975). *Values of Student's t are* ***numbers of standard errors*** *to be expected with specified probability and for a given number of degrees of freedom*. The values of t for 99% are bigger than these (0.005 in each tail):

```
qt(.995,9)

[1] 3.249836
```

and the value for 99.5% bigger still (0.0025 in each tail):

```
qt(.9975,9)

[1] 3.689662
```

Values of Student's t like these appear in the formula for calculating the width of the confidence interval, and their inclusion is the reason why the width of the confidence interval goes up as our degree of confidence is increased. The other component of the formula, the standard error, is not affected by our choice of confidence level. So, finally, we can write down the formula for the confidence interval of a mean based on a small sample (n < 30):

$$CI_{95\%} = t_{(\alpha=0.025,\ d.f.=9)} \sqrt{\frac{s^2}{n}}$$

For Garden B, therefore, we could write

```
qt(.975,9)*sqrt(1.33333/10)

[1] 0.826022
```

"The mean ozone concentration in Garden B was 5.0 ± 0.826 (95% C.I., n = 10)."

Quantiles

Quantiles are important summary statistics. They show the values of y that are associated with specified percentage points of the distribution of the y values. For instance, the 50% quantile is the same thing as the median. The most commonly used quantiles are aimed at specifying the middle of a data set and the tails of a data set. By the middle of a data set we mean the values of y between which the middle 50% of the numbers lie. That is to say, the values of y that lie between the 25% and 75% quantiles. By the tails of a distribution we mean the extreme values of y: for example, we might define the tails of a distribution as the values that are smaller than the 2.5% quantile or larger than the 97.5% quantile. To see this, we can generate a vector called *z* containing 1000 random numbers drawn from a normal distribution using the

function rnorm, with a mean of 0 and a standard deviation of 1 (the ‘standard normal distribution’ as it is known). This is very straightforward:

```
z<-rnorm(1000)
```

We can see how close the mean really is to 0.0000

```
mean(z)
```

```
[1] -0.01325934
```

Not bad. It is out by just over 1.3%. But what about the tails of the distribution. We know that for an infinitely large sample, the standard normal should have 2.5% of its z values less than –1.96, and 97.5% of its values less than +1.96 (see below). So what is this sample of 1000 points like ? We concatenate the two fractions 0.025 and 0.975 to make the second argument of quantile

```
quantile(z,c(.025,.975))
```

```
      2.5%    97.5%
-1.913038 2.013036
```

Hm. Out by more than 2.5%. Close, but no coconut. It could be just a sample size thing. What if we try with 10,000 numbers ?

```
z<-rnorm(10000)
```

```
quantile(z,c(.025,.975))
```

```
      2.5%    97.5%
-1.985679 1.956595
```

Much better. Clearly the random number generator is a good one, but equally clearly, we should not expect samples of order 1000 to be able to produce exact estimates of the tails of a distribution. This is an important lesson to learn if you intend to use simulation (Monte Carlo methods) in testing statistical models. It says that 10,000 tries is much better than 1,000. Computing time is cheap, so go for 10,000 tries in each run.

Robust estimators

One of the big problems with out standard estimators of central tendency and variability, the mean and the variance, is that they are extremely sensitive to the presence of outliers. We already know how to obtain a robust estimate of central tendency that is not affected by outliers: the median. There are several modern methods of obtaining robust estimators of standard deviation that are less sensitive to outliers. The first is called **mad**: great name, the Median Absolute Deviation. Its value is scaled to be a consistent estimator of the standard deviation from the normal distribution.

```
y<- c(3,4,6,4,5,2,4,5,1,5,4,6)
```

For our existing data set, it looks like this:

```
mad(y)
```

```
[1] 1.4826
```

which is close to, but different from, the standard deviation of y

```
sd(y)
```

```
[1] 1.505042
```

Let's see how the different measures perform in the presence of outliers. We can add an extreme outlier (say y = 100) to our existing data set, using concatenation, and call the new vector y1:

```
y1<-c(y,100)
```

How has the outlier affected the mean (it used to be 4.08333) ?

```
mean(y1)
```

```
[1] 11.46154
```

It has nearly tripled it ! And the standard deviation (it used to be 1.505042) ?

```
sqrt(var(y1))
```

```
[1] 26.64149
```

A more than 17-fold increase. Lets see how the outlier has affected mad's estimate of the standard deviation:

```
mad(y1)
```

```
[1] 1.4826
```

Hardly at all (it was 1.505 before the outlier was added).

These robust estimators suggest a simple function to test for the presence of outliers in a data set. We need to make a decision about the size of the difference between the regular standard deviation and the **mad** in order for us to be circumspect about the influence of outliers. In our extreme example comparing the standard deviations of *y* and *y1* the ratio was 26.64 compared to 1.486 (a nearly 18-fold difference). What if we pick a 4-fold difference as our threshold; it will highlight extreme cases like *y* vs *y1* but it will allow a certain leeway for ecological kinds of data. Let's call the function outlier, and write it like this:

```
outlier<-function(x) {
   if(sqrt(var(x))>4*mad(x))   print("Outliers present")
   else   print("Deviation reasonable") }
```

We can test it by seeing what it makes of the original data set, y:

```
outlier(y)

[1] "Deviation reasonable"
```

What about the data set including y = 100 ?

```
outlier(y1)

[1] "Outliers present"
```

It is good practice to compare robust and standard estimators. I am not suggesting that we do away with means and variances; just that we be aware as to how sensitive they are to extreme values.

Single-sample estimation

Suppose we have a single sample. The questions we might want to answer are these:

1) what is the mean value ?

2) is the mean value significantly different from current expectation or theory ?

3) what is the level of uncertainty associated with our estimate of the mean value ?

In order to be reasonably confident that our inferences are correct, we need to establish some facts about the distribution of the data.

1) are the values normally distributed or not ?

2) are there outliers in the data ?

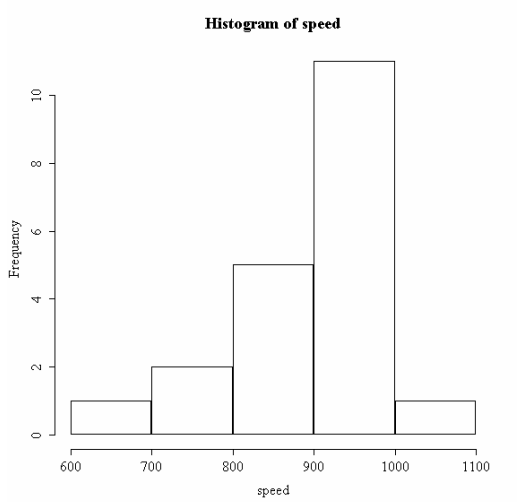
3) if data were collected over a period of time, is there evidence for serial correlation?

Non-normality, outliers and serial correlation can all invalidate inferences made by standard parametric tests like Student's t-test. Much better in such cases to use a robust non-parametric technique like Wilcoxon's signed-rank test.

We can investigate the issues involved with Michelson's (1879) famous data on estimating the speed of light. The actual speed is 299,000 km sec^{-1} plus the values in our data frame called light:

```
light<-read.table("c:\\temp\\light.txt",header=T)
attach(light)
```

```
names(light)
[1] "speed"
hist(speed)
```



We get a **summary** of the non-parametric descriptors of the sample like this:

```
summary(speed)
```

```
 Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
  650     850       940       909       980      1070
```

From this, you see at once that the median (940) is substantially bigger than the mean (909), as a consequence of the strong negative skew in the data seen in the histogram. The **interquartile range** is the difference between the 1st and 3rd quartiles: 980 – 850 = 130. This is useful in the detection of outliers: a good rule of thumb is this

*an **outlier** is a value more than 1.5 times the interquartile range above the 3rd quartile, or below the 1st quartile.*

In this case, outliers would be measurements of speed that were less than 850 – 195 = 655 or greater than 980 + 195= 1175. You will see that there are no large outliers in this data set, but one or more small outliers (the minimum is 650).

Inference in the 1-sample case

We want to test the hypothesis that Michelson's estimate of the speed of light is significantly different from the value of 299,990 thought to prevail at the time. The data have all had 299,000 subtracted from them, so the test value is 990. Because of the non-normality, the use of Student's t-test in this case is ill advised. The correct test is Wilcoxon's signed rank test. The code for this is in a library (note the 'dot')

```
library(ctest)

wilcox.test(speed,mu=990)

Warning: Cannot compute exact p-value with ties
```

```
        Wilcoxon signed rank test with continuity
correction

data:  speed
V = 22.5, p-value = 0.00213
alternative hypothesis: true mu is not equal to 990
```

We accept the alternative hypothesis because p = 0.00213 (i.e. much less than 0.05).

For the purpose of demonstration only, we demonstrate the use of Student's t-test

```
t.test(speed,mu=990)

     One-sample t-Test

data:  speed
t = -3.4524, df = 19, p-value = 0.0027
alternative hypothesis: true mean is not equal to 990
95 percent confidence interval:
 859.8931 958.1069
```

This result says that the sample mean is highly significantly different from the null hypothesis value of 990 (p = 0.0027). The 95% confidence interval (859.9 to 958.1) does not include the hypothesised value.

Comparing two means

There are two simple tests for comparing two sample means:

- **Student's t-test** when the means are independent, the variances constant, and the errors are normally distributed

- **Wilcoxon rank sum test** when the means are independent but errors are *not* normally distributed

What to do when these assumptions are violated (e.g. when the variances are different) is discussed later on .

Student was the pseudonym of W.S. Gosset who published his influential paper in *Biometrika* in 1908. He was prevented from publishing under his own name by dint of the archaic employment laws in place at the time which allowed his employer, the Guiness Brewing Company, to prevent him publishing independent work. Student's t distribution, later perfected by R. A. Fisher, revolutionised the study of small sample statistics where inferences need to be made on the basis of the sample variance s^2 with the population variance σ^2 unknown (indeed, usually unknowable). The test statistic is the number of standard errors by which the 2 sample means are separated:

$$t = \frac{\text{difference between the 2 means}}{\text{SE of the difference}} = \frac{\bar{y}_A - \bar{y}_B}{SE_{\text{diff}}}$$

Now we know the standard error of the mean (see p 65) but we have not yet met the standard error of the difference between two means. *The variance of a difference is the sum of the separate variances*. To see this, think about the sum of squares of a difference:

$$\sum\left[(y_A - y_B) - (\mu_A - \mu_B)\right]^2$$

If we average this, we get the variance of the difference (forget about degrees of freedom for a minute). Let's call this average $\sigma^2_{\bar{y}_A - \bar{y}_B}$ and rewrite the sum of squares

$$\sigma^2_{\bar{y}_A - \bar{y}_B} \;=\; \text{average of } [\,(y_A - \mu_A) - (y_B - \mu_B)\,]^2$$

Now square, then expand the brackets, to give

$$(y_A - \mu_A)^2 + (y_B - \mu_B)^2 - 2(y_A - \mu_A)(y_B - \mu_B)$$

We already know that the average of $(y_A - \mu_A)^2$ is the variance of population A and the average of $(y_B - \mu_B)^2$ is the variance of population B. So the variance of the *difference* between the two sample means is the *sum* of the variances of the two samples, plus this term $2(y_A - \mu_A)(y_B - \mu_B)$. But we also know that $\sum d = 0$ (see above) so, because the samples from A and B are independently drawn they are uncorrelated , which means that

$$\text{average of } (y_A - \mu_A)(y_B - \mu_B) \;=\; (y_A - \mu_A)\{\text{average of } (y_B - \mu_B)\} \;=\; 0$$

This important result needs to be stated separately

$$\sigma^2_{\bar{y}_A - \bar{y}_B} \;=\; \sigma^2_A \;+\; \sigma^2_B$$

so if both samples are drawn from populations with the same variance, then *the variance of the difference is twice the variance of an individual mean*.

This allows us to write down the formula for the standard error of the difference between two sample means

$$SE_{\text{difference}} \;=\; \sqrt{\frac{s_A^2}{n_A} + \frac{s_B^2}{n_B}}$$

At this stage we have everything we need to carry out students t-test. Our null hypothesis is that the two sample means are the same, and we shall accept this unless the value of Student's t is so large that it is unlikely that such a difference could have arisen by chance alone. Each sample has 9 degrees of freedom, so we have 18 d.f. in total. Another way of thinking of this is to reason that the complete sample size as 20, and we have estimated 2 parameters from the data, $\bar{y}_A$ and $\bar{y}_B$, so we have 20 – 2 = 18 d.f. We typically use 5% as the chance of rejecting the null hypothesis when it is true

(this is called the Type I error rate). Because we didn't know in advance which of the two gardens was going to have the higher mean ozone concentration (we usually don't), this is a 2-tailed test, so the *critical value* of Student's t is:

```
qt(.975,18)
```

```
[1] 2.100922
```

Thus our test statistic needs to be bigger than 2.1 in order to reject the null hypothesis, and to conclude that the two means are significantly different at $\alpha = 0.05$.

We can write the test like this, using the variances s2A and s2B that we calculated earlier

```
(mean(A)-mean(B))/sqrt(s2A/10+s2B/10)
```

which gives the value of Student's t as

```
[1] -3.872983
```

You won't be at all surprised to learn that there is a built-in function to do all the work for us. It is called, helpfully, **t.test** and is used simply by providing the names of the two vectors containing the samples on which the test is to be carried out (A and B in our case).

```
t.test(A,B)
```

There is rather a lot of output . You often find this. The simpler the statistical test, the more voluminous the output.

```
     Standard Two-Sample t-Test

data:  A and B
t = -3.873, df = 18, p-value = 0.0011
alternative hypothesis: true difference in means is not
equal to 0
95 percent confidence interval:
 -3.0849115 -0.9150885
sample estimates:
 mean of x mean of y
         3         5
```

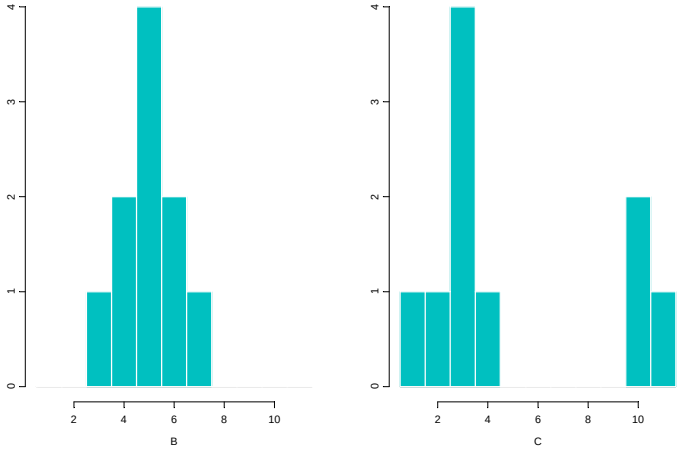
The result is exactly the same as we obtained long hand. The value of t is –3.873 and since *the sign is irrelevant in a t test* we reject the null hypothesis because the test statistic is larger than the critical value of 2.1. The mean ozone concentration is significantly higher in Garden B than in Garden A. The computer print out also gives a p value and a confidence interval. Note that, because the means are significantly different, *the confidence interval on the difference does not include zero* (in fact, it goes from –3.085 up to –0.915). The p value is useful in written work, like this:

“Ozone concentration was significantly higher in Garden B (mean = 5.0 pphm) than in Garden A (mean = 3.0; t = 3.873, p = 0.001, d.f. = 18).”

Wilcoxon rank sum test

A non-parametric alternative to Student’s t test which we could use if the errors looked to be non-normal. Let’s look at the errors in Gardens B and C

```
par(mfrow=c(1,2))
hist(B,breaks=c(0.5:11.5))
hist(C,breaks=c(0.5:11.5))
```



The errors in B look to be reasonably normal, but the errors in C are definitely not normal. The Wilcoxon rank sum test statistic, W, is defined like this. Both samples are put into a single array with their sample names (B and C in this case) clearly attached. Then the aggregate list is sorted, taking care to keep the sample labels with their respective values. Then a rank is assigned to each value, with ties getting appropriate average rank (2-way ties get (rank i + (rank i + 1))/2 , 3-way ties get (rank i + (rank i + 1) + (rank i +2))/3, and so on). Finally the ranks are added up for each of the two samples, and significance is assessed on size of the smaller sum of ranks.

This involves some interesting computing. First we make a combined vector of the samples

```
combined<-c(B,C)
combined
 [1]  5  5  6  7  4  4  3  5  6  5  3  3  2  1 10  4  3
11  3 10
```

then make a list of the sample names, “B” and “C”

```
sample<-c(rep("B",10),rep("C",10))
sample
```

```
 [1] "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "C" "C" "C"
"C" "C" "C" "C" "C" "C" "C"
```

Now the trick is to use the built-in function **rank** to get a vector containing the ranks, smallest to largest, within the combined vector:

```
rank.combi<-rank(combined)
rank.combi
```

```
[1] 12.5 12.5 15.5 17.0  9.0  9.0  5.0 12.5 15.5 12.5  5.0  5.0  2.0  1.0 18.5  9.0  5.0 20.0  5.0 18.5
```

Notice that the ties have been dealt with by averaging the appropriate ranks. Now all we need to do is calculate the sum of the ranks for each garden. We could use **sum** with conditional subscripts for this:

```
sum(rank.combi[sample=="B"])
```

```
[1] 121
```

```
sum(rank.combi[sample=="C"])
```

```
[1] 89
```

Alternatively, we could use **tapply** with **sum** as the required operation

```
tapply(rank.combi,sample,sum)
```

```
   B         C
 121        89
```

In either case, we compare the smaller of the two values (89) with values in Tables (e.g. Snedecor & Cochran, p. 555) and reject the null hypothesis if our value of 89 is *smaller* than the value in tables. For samples of size 10 and 10, the value in tables is 78. Our value is much bigger than this, so we accept the null hypothesis. The two sample means are not significantly different (they are identical, in fact, as we already know).

We can carry out the whole procedure automatically, and avoid the need to use tables of critical values of Wilcoxon's rank sum test, by using the built-in function **wilcox.test**:

```
wilcox.test(B,C)
```

which produces the following output:

```
     Wilcoxon rank-sum test

data:  B and C
rank-sum normal statistic with correction Z = 1.1879, p-
value = 0.2349
alternative hypothesis: true mu is not equal to 0
```

```
Warning messages:
  cannot compute exact p-value with ties in:
wil.rank.sum(x, y, alternative, exact, correct)
```

This is interpreted as follows. The function uses a normal approximation algorithm to work out a z value and from this a p value for the assumption that the means are the same. This p value is much bigger than 0.05, so we accept the null hypothesis. Unhelpfully, it then prints the alternative hypothesis in full, which a careless reader could take as meaning that "true mu is not equal to 0" ("mu" is the difference between the 2 means). We have just demonstrated, rather convincingly, that the true mu *is* equal to 0. The idea of putting the message in the output is to show that we were doing (the default) 2-tailed test, but this is a silly way of saying it, with a serious risk of misinterpretation. The warning message at the end draws attention to the fact that there are ties in the data, and hence the p value can not be calculated exactly (this is seldom a real worry).

Overview

The non-parametric test is much more appropriate than the t-test when the errors are not normal, and the non-parametric is about 95% as powerful with normal errors, and can be *more* powerful than the t-test if the distribution is strongly skewed by the presence of outliers. But the Wilcoxon test does not make the really important point for this case, because like the t-test, it says that ozone pollution in the 2 gardens is not significantly different. Yet we know, because we have looked at the data, that Gardens B and C are definitely different in terms of their ozone pollution, but it is the *variance* that differs, not the mean. Neither the t-test nor the sum rank test can cope properly with situations where the variances are different, but the means are the same.

This draws attention to a very general point:

scientific importance and statistical significance are not the same thing

Lots of results can be highly significant but of no scientific importance at all (e.g. because the effects are cancelled out by later events). Likewise, lost of scientifically important processes may not be statistically significant (e.g. density dependence in population growth rate may not be statistically significant, but leaving it out of a population model because it is not significant will have disastrous effects on the predictions of population dynamics made by the model).

Tests on paired samples

Sometimes, 2-sample data come from paired observations. In this case, we might expect a correlation between the two measurements, either because they were made on the same individual, or were taken from the same location. You might recall that earlier we found that the variance of a difference was the average of

$$(y_A - \mu_A)^2 + (y_B - \mu_B)^2 - 2(y_A - \mu_A)(y_B - \mu_B)$$

which is the variance of sample A plus the variance of sample B minus 2 times the covariance of A and B (see above). When the covariance of A and B is *positive*, this is a great help because it reduces the variance of the difference, and should make it easier to detect significant differences between the means. Pairing is not always effective, because the correlation between y_A and y_B may be weak. It would be disastrous if the correlation were to turn out to be negative!

One way to proceed is to reduce the estimate of the standard error of the difference by taking account of the measured correlation between the two variables. A simpler alternative is to calculate the difference between the two samples from each pair, then do a 1-sample test comparing the mean of the differences to zero. This halves the number of degrees of freedom, but it reduces the error variance substantially if there is a strong positive correlation between y_A and y_B.

The data are a composite biodiversity score based on a kick sample of aquatic invertebrates.

```
x<-c(20,15,10,5,20,15,10,5,20,15,10,5,20,15,10,5)

y<-c(23,16,10,4,22,15,12,7,21,16,11,5,22,14,10,6)
```

The elements of x and y are paired because the 2 samples were taken on the same river, upstream (*y*) or downstream (*x*) of a sewage outfall. If we ignore the fact that the samples are paired, it appears that the sewage outfall has no impact on biodiversity score ($p = 0.6856$):

```
t.test(x,y)

     Standard Two-Sample t-Test

data:  x and y
t = -0.4088, df = 30, p-value = 0.6856
alternative hypothesis: true difference in means is not
equal to 0
95 percent confidence interval:
 -5.246747  3.496747
sample estimates:
 mean of x mean of y
      12.5    13.375
```

However, if we allow that the samples are paired (simply by specifying the option **paired=T**), the picture is completely different.

```
t.test(x,y,paired=T)

     Paired t-Test

data:  x and y
t = -3.0502, df = 15, p-value = 0.0081
```

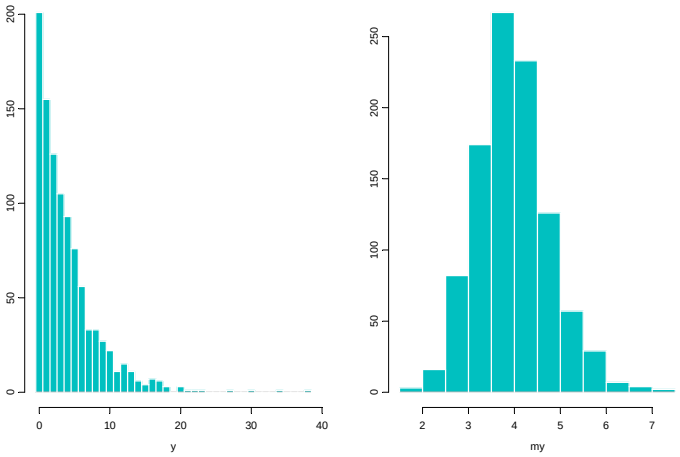
```
alternative hypothesis: true mean of differences is not
equal to 0
95 percent confidence interval:
 -1.4864388 -0.2635612
sample estimates:
 mean of x - y
       -0.875
```

The difference between the means is highly significant (p = 0.0081). The moral is clear. If you have information on blocking (the fact that the two samples came from the same river in this case), then use it in the analysis. It can never do any harm, and sometimes (as here) it can do a huge amount of good.

Central Limit Theorem

The central limit theorem states that for any distribution with a *finite variance*, the mean of a *random sample* from that distribution tends to be normally distributed.

The central limit theorem works remarkably well, even for really badly behaved data. Take this negative binomial distribution that has a mean of 3.88, a variance mean ratio of 4.87, and k = 1.08 (on the left): you see that the means of repeated samples of size n = 30 taken from this highly skew distribution are close to normal in their



distributions. The panels were produced like this. The left hand histogram is a frequency distribution of 1000 negative binomial random numbers (with parameters size = 1, probability = 0.2). The frequency distribution is viewed using **table**. There were 201 zero's in this particular realisation, and 1 value of 38. Note the use of the sequence to produce the required **break** points in the histogram.

```
par(mfrow=c(1,2))
y<-rnbinom(1000,1,.2)
```

This says generate 1000 random numbers from a negative binomial distribution whose 2 parameters are 1.0 and 0.2 respectively.

```
mean(y)

[1] 3.879
var(y)

[1] 18.90326

table(y)

  0   1   2   3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 20 21 22 23 27 30 34 38
201 155 126 105 93 76 56 33 33 27 22 11 15 11  6  4  7  6  3  3  1  1  1  1  1  1  1

hist(y,breaks=-0.5:38.5)
```

The right hand panel shows the distributions of means of samples of n = 30 from this same negative binomial distribution. One thousand different samples were taken and their average recorded in the vector called *my* from which the histogram is produced. When you type the for loop, hit “hard return” after the opening curly bracket and at the end of each line inside the multi-line function (you will see the line continuation prompt “+” at the beginning of each line until the closing curly bracket is entered).

```
my <- numeric(1000)
for (i in 1:1000) {
                y <- rnbinom(30, 1, 0.2)
                my[i] <- mean(y) }
hist(my)
```

The 1000 calculations are carried out very swiftly, and you see a normal distribution of the mean values of y. In fact, the central limit theorem works reasonably well for sample sizes much smaller than 30 as we shall see in a later practical.

```
par(mfrow=c(1,1))
```

STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

4. REGRESSION

Regression is the statistical model we use when the explanatory variable is continuous. If the explanatory variables were categorical we would use Analysis of Variance (Exercises 5). If you are in any doubt about whether to use regression or analysis of variance, ask yourself whether your graphical investigation of the data involved producing scatter plots or bar charts. If you produced scatter plots, then the statistics you need to use are regression. If you produced bar charts, then you need Analysis of Variance.

To understand how the statistical part of R works, we shall work through a series of simple examples in sufficient detail that the calculations carried out within R become apparent. It is most important when learning how to use a new statistical package to understand exactly what the output means. What the numbers are, and where they come from. More experienced readers can skip this introductory material.

In all the examples that follow, we make the following assumptions:

- errors are normally distributed
- variances are constant
- the explanatory variable is measured without error
- all of the unexplained variation is confined to the response variable.

Later on, we shall deal with cases that have non-normal errors and unequal variances.

The Model

We begin with the simplest possible linear model; the straight line

$$y = a + bx$$

where a **response variable** y is hypothesised as being a linear function of the **explanatory variable** x, and the two **parameters** a and b. In the case of simple linear regression, the parameter a is called the *intercept* (the value of y when x = 0), and b is the *slope* of the line (or the gradient, measured as the change in y in response to unit change in x). The aims of the analysis are as follows:

- to estimate the values of the parameters a and b
- to estimate their standard errors
- to use the standard errors to assess which terms are necessary within the model (i.e. whether the parameter values are significantly different from zero)

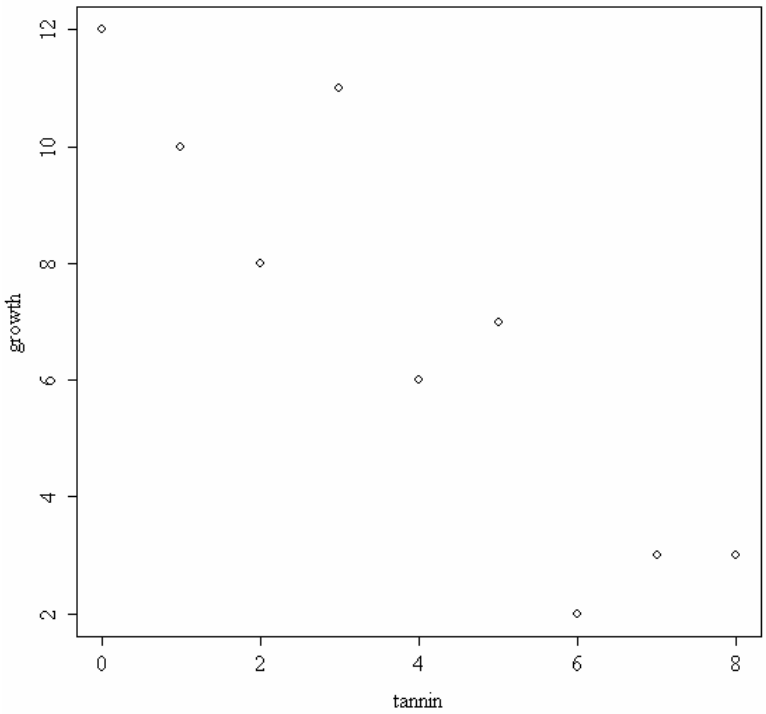
- to determine what fraction of the variation in *y* is explained by the model and how much remains unexplained

Data inspection.

The first step is to look carefully at the data. Is there an upward or downward trend, or could a horizontal straight line be fit through the data? If there is a trend, does it look linear or curvilinear? Is the scatter of the data around the line more or less uniform, or does the scatter change systematically as x changes?

Consider the data in the data frame called tannin. They show how weight gain (mg) of individual caterpillars declines as the tannin content of their diet (%) increases.

```
regression<-read.table("c:\\temp\\regression.txt",header=T)
attach(regression)
names(regression)
[1] "growth" "tannin"
plot(tannin,growth)
```



It looks as if there is a downward trend in *y* as *x* increases, and that the trend is roughly linear. There is no evidence of any systematic change in the scatter as *x* changes. This cursory inspection leads to several expectations:

- the intercept *a* is greater than zero
- the slope *b* is negative
- the variance in *y* is constant

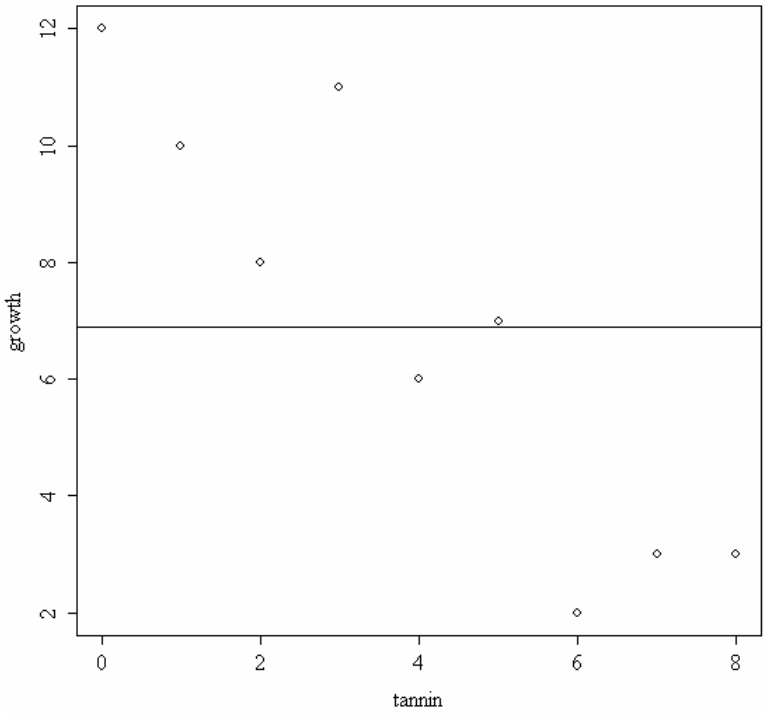
- the scatter about the straight line is relatively slight

It now remains to carry out a thorough statistical analysis to substantiate or refute these initial impressions, and to provide accurate estimates of the slope and intercept, their standard errors and the degree of fit.

Least squares

The technique of least squares linear regression defines the *best fit* straight line as the line which minimises *the sum of the squares of the departures of the y values from the line*. We can see what this means in graphical terms. The first step is to fit a horizontal line though the data, using **abline**, showing the average value of *y* (the first parameter is the intercept, the second is the slope):

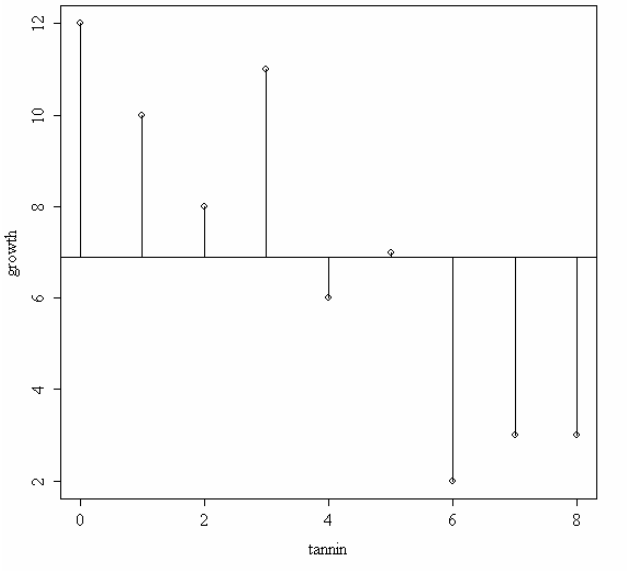
```
mean(growth)
[1] 6.888889
abline(6.889,0)
```



The scatter around the line defined by $\bar{y}$ is said to be the total variation in y. Each point on the graph lies a vertical distance $d = y - \bar{y}$ from the horizontal line, and we define the total variation in y as being the sum of the squares of these departures:

$$SST = \sum \left(y - \bar{y} \right)^2$$

where SST stands for *total sum of squares*. It is the sum of the squares of the vertical distances shown here:

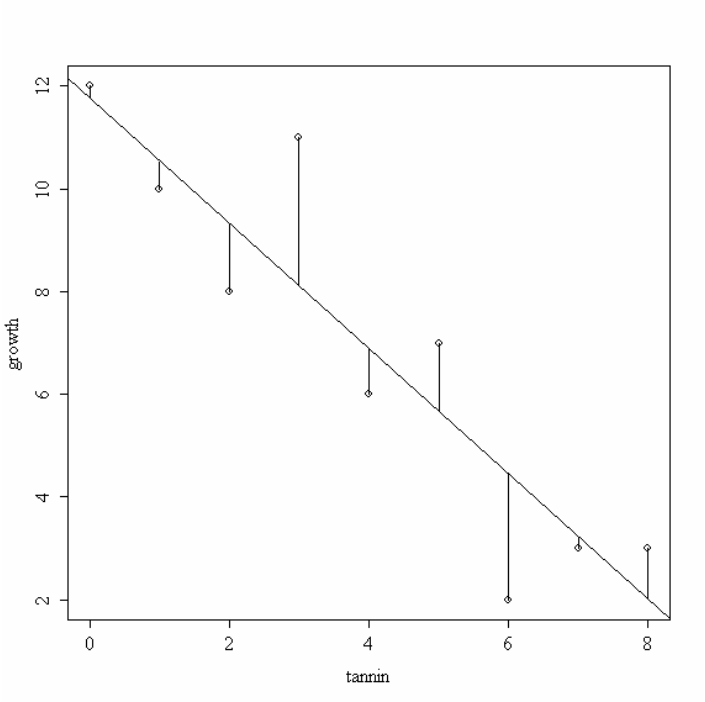


Now we want to fit a straight line through the data. There are two decisions to be made about such a best fit line:

- where should the line be located?
- what slope should it have?

Location of the line is straightforward, because a best fit line should clearly pass through the point defined by the average values of x and y . The line can then be pivoted at the point $(\bar{x}, \bar{y})$, and rotated until the best fit is achieved. The process is formalised as follows. Each point on the graph lies a distance $e = y - \hat{y}$ from the fitted line, where the predicted value $\hat{y}$ is found by evaluating the equation of the straight line at the appropriate value of x:

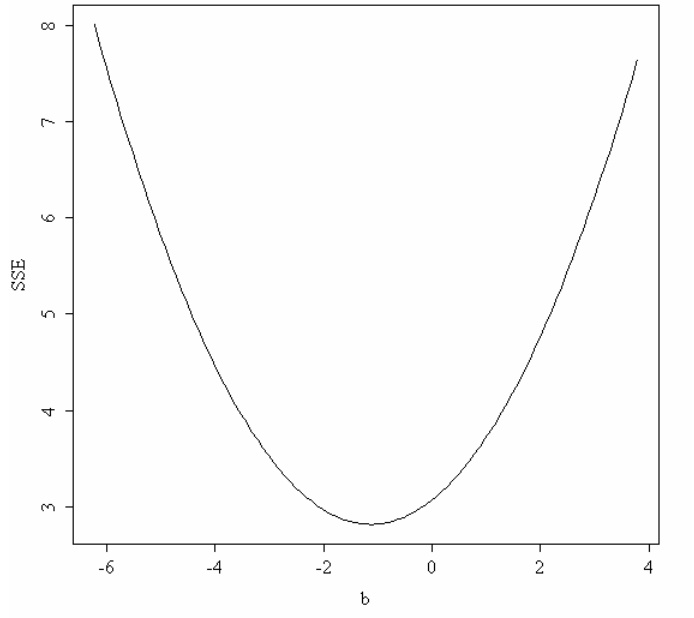
$$\hat{y} = a + bx$$



Let us define the error sum of squares as the sum of the squares of the e's:

$$SSE = \sum (y - \hat{y})^2$$

Now imagine rotating the line around the point $(\bar{x}, \bar{y})$. SSE will be large when the line is too steep. It will decline as the slope gets closer to the best-fit line, then it will increase again as the line becomes too shallow. We can draw a graph of SSE against the slope b, like this:



We define *the best fit line as the one that minimises SSE*. The maximum likelihood estimate of the slope is obviously somewhere around –1.2. To find this value analytically, we find the derivative of SSE with respect to *b*, set it to zero, and solve for b.

Maximum likelihood estimate of the slope

The location of the line is fixed by assuming that it passes through the point $(\bar{x}.\bar{y})$, so that we can rearrange the equation to obtain an estimate of the intercept *a* in terms of the best fit slope, *b*.

$$a = \bar{y} - b\bar{x}$$

We begin by replacing the average values of y and x by $\sum y / n$ and $\sum x / n$ so

$$a = \frac{\sum y}{n} - b\frac{\sum x}{n}$$

The *best fit* slope is found by rotating the line until the *error sum of squares*, SSE, is minimised. The error sum of squares is the sum of squares of the individual departures, $e = y - \hat{y}$, shown above:

$$SSE = \text{ minimum } \sum (y - a - bx)^2$$

noting the change in sign of *bx*. This is how the best fit is defined.

Next, we find the derivative of SSE with respect to *b*

$$\frac{dSSE}{db} = -2\sum x(y - a - bx)$$

because the derivative with respect to b of the bracketed term is -*x*, and the derivative of the squared term is 2 times the squared term. The constant -2 can be taken outside the summation. Now, multiplying through the bracketed term by *x* gives

$$\frac{dSSE}{db} = -2\sum xy - ax - bx^2$$

Now take the summation of each term separately, set the derivative to zero, and divide both sides by -2 to remove the unnecessary constant:

$$\sum xy - \sum ax - \sum bx^2 = 0$$

We can not solve the equation as it stands because there are two unknowns, *a* and *b*. However, we already know the value of a in terms of b. Also, note that $\sum ax$ can be

written as $a\sum x$, so, replacing *a* and taking both *a* and *b* outside their summations gives:

$$\sum xy - \left[\frac{\sum y}{n} - b\frac{\sum x}{n}\right]\sum x - b\sum x^2 = 0$$

Now multiply out the central bracketed term by $\sum x$ to get

$$\sum xy - \frac{\sum x \sum y}{n} + b\frac{(\sum x)^2}{n} - b\sum x^2 = 0$$

Finally, take the 2 terms containing *b* to the other side, and note their change of sign:

$$\sum xy - \frac{\sum x \sum y}{n} = b\sum x^2 - b\frac{(\sum x)^2}{n}$$

and then divide both sides by $\sum x^2 - (\sum x)^2 / n$ to obtain the required estimate b:

$$b = \frac{\sum xy - \frac{\sum x \sum y}{n}}{\sum x^2 - \frac{(\sum x)^2}{n}}$$

Thus, the value of b that minimises the sum of squares of the departures is given simply by

$$b = \frac{SSXY}{SSX}$$

where SSXY stands for the corrected sum of products (*x* times *y*; the measure of how *x* and y co-vary), and SSX is the corrected sum of squares for x, calculated in exactly the same manner as the total sum of squares SST, which we met earlier.
For comparison, these 3 important formulas are presented together:

$$SST = \sum y^2 - \frac{(\sum y)^2}{n}$$

$$SSX = \sum x^2 - \frac{(\sum x)^2}{n}$$

$$SSXY = \sum xy - \frac{\sum x \sum y}{n}$$

Note the similarity of their structures. SST is calculated by adding up *y* times *y* then subtracting the total of the *y*'s times the total of the *y*'s divided by *n* (the number of points on the graph). Likewise, SSX is calculated by adding up *x* times *x* then subtracting the total of the *x*'s times the total of the *x*'s divided by *n*. Finally, SSXY is calculated by adding up *x* times *y* then subtracting the total of the *x*'s times the total of the *y*'s divided by *n*.

It is worth reiterating what these three quantities represent. SST measures the total variation in the y values about their mean (it is the sum of the squares of the *d*'s in the earlier plot). SSX represents the total variation in x (expressed as the sum of squares of the departures from the mean value of x), and is a measure of the range of x values over which the graph has been constructed. SSXY measures the correlation between *y* and *x* in terms of the corrected sum of products. Note that SSXY is negative when *y* declines with increasing *x*, positive when *y* increases with *x*, and zero when *y* and *x* are uncorrelated.

Analysis of variance in regression

The idea is to partition the total variation in the response, SST, into 2 components: the component that is explained by the regression line (which we shall call SSR, the regression sum of squares) and an unexplained component (the residual variation which we shall call SSE, the error sum of squares).

Look at the relative sizes of the departures d and e in the figures we drew earlier. Now, ask yourself what would be the relative size of $\sum d^2$ and $\sum e^2$ if the slope of the fitted line were *not significantly different from zero*? A moment's thought should convince you that if the slope of the best fit line were zero, then the two sums of squares would be exactly the same. If the slope were zero, then the two lines would lie in exactly the same place, and the sums of squares would be identical. Thus, when the slope of the line is not significantly different from zero, we would find that SST = SSE. Similarly, if the slope was significantly different from zero (i.e. significantly positive or negative), then SSE would be substantially *less* than SST. In the limit, if all the points fell exactly on the fitted line, then SSE would be zero.

Now we calculate a third quantity, SSR, called the *regression sum of squares*:

$$SSR = SST - SSE$$

This definition means that SSR will be large when the fitted line accounts for much of the variation in *y*, and small when there is little or no linear trend in the data. In the limit, SSR would be equal to SST if the fit was perfect (because SSE would then equal zero), and SSR would equal zero if y was independent of x (because, in this case, SSE would equal SST).

These three quantities form the basis for drawing up the ANOVA table:

Source	SS	df	MS	F	F tables (5%)
Regression	SSR	1	SSR	$F = \frac{SSR}{s^2}$	qf(0.95,1,n-2)
Error	SSE	n-2	$s^2 = \frac{SSE}{n-2}$		
Total	SST	n-1			

The sums of squares are entered in the first column. SST is the corrected sum of squares of *y*, as given above. We have defined SSE already, but this formula is inconvenient for calculation, since it involves n different estimates of $\hat{y}$ and *n* subtractions. Instead, it is easier to calculate SSR and then to estimate SSE by difference. The regression sum of squares is simply:

$$SSR = b.SSXY$$

so that

$$SSE = SST - SSR$$

The second column of the anova table contains the degrees of freedom. The estimation of the total sum of squares required that one parameter $\bar{y}$, the mean value of *y*, be estimated from the data prior to calculation. Thus, the total sum of squares has *n*-1 degrees of freedom when there are *n* points on the graph. The error sum of squares could not be estimated until the regression line had been drawn through the data. This required that two parameters, the mean value of *y* and the slope of the line were estimated from the data. Thus, the error sum of squares has *n*-2 degrees of freedom. The regression degrees of freedom represents *the number of extra parameters* involved in going from the null model $y = \bar{y}$ to the full model $y = a + bx$. (i.e. 2 –1 = 1 degree of freedom) because we have added the slope, *b* to the model. As always, the component d.f. add up to the total d.f.

At this stage it is worth recalling that variance is ***always*** calculated as

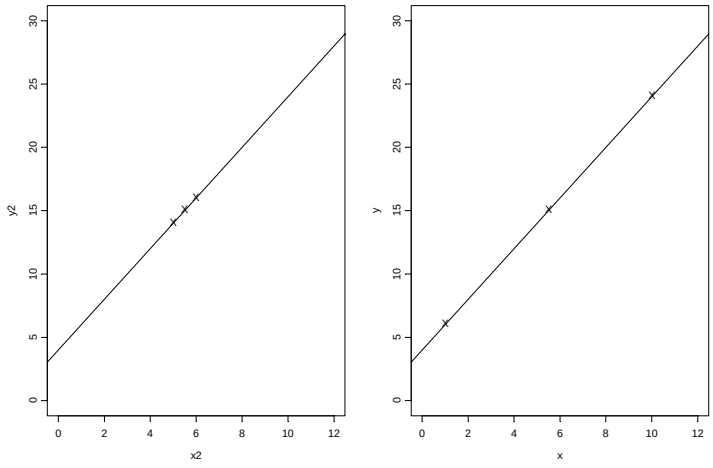
$$\text{variance} = \frac{\text{sum of squares}}{\text{degrees of freedom}}$$

The anova table is structured to make the calculation of variances as simple and clear as possible. For each row in an anova table, you simply divide the sum of squares by the adjacent degrees of freedom. The third column therefore contains two variances; the first row shows the regression variance and the second row shows the all-important error variance (s^2). One of the oddities of analysis of variance is that the variances are referred to as *mean squares* (this is because a variance is defined as a mean squared deviation). The error variance (also known as the *error mean square*, MSE) is the quantity used in calculating standard errors and confidence intervals for the parameters, and in carrying out hypothesis testing.

The **standard error of the regression slope**, *b*, is given by:

$$SE_b = \sqrt{\frac{s^2}{SSX}}$$

Recall that standard errors are *unreliability estimates*. Unreliability increases with the error variance so it makes sense to have s^2 in the numerator (on top of the division). It is less obvious why unreliability should depend on the range of x values. Look at these two graphs that have exactly the same slopes and intercepts. The difference is that the left hand graph has all of its x values close to the mean value of x while the graph on the right has a broad span of x values. Which of these would you think would give the most reliable estimate of the slope?



It is pretty clear that it is the graph on the right, with the wider range of x values. Increasing the spread of the x values reduces unreliability and hence appears in the denominator (on the bottom of the equation).

What is the purpose of the big square root term? This is there to make sure that the units of the unreliability estimate are the same as the units of the parameter whose unreliability is being assessed. The error variance is in units of *y squared*, but the slope is in units of *y* per unit change in x.

A 95% confidence interval for *b* is now given in the usual way:

$$CI_b = \text{t (from tables, } \alpha = 0.025, d.f. = n - 2).SE_b$$

where t is obtained using **qt**, the quantile of Student's t distribution (2-tailed, α = 0.025) with *n*-2 degrees of freedom.

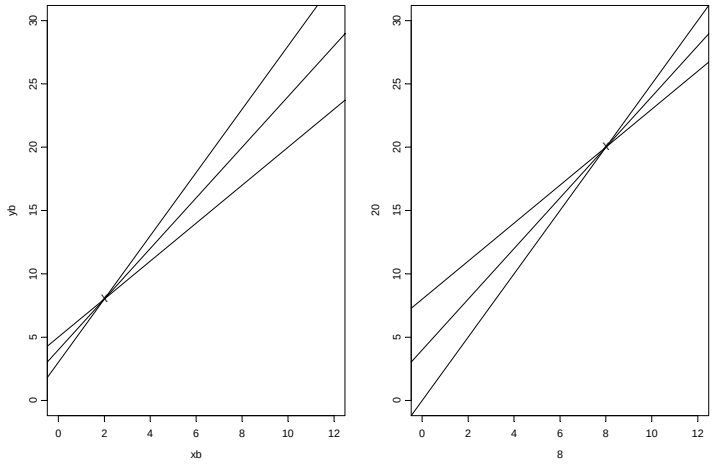
```
qt(.975,7)
```

```
[1] 2.364624
```

a little larger than the rule of thumb $(t \approx 2)$ because the degrees of freedom is so small. The **standard error of the intercept**, *a*, is given by:

$$SE_a = \sqrt{\frac{s^2 \sum x^2}{n.SSX}}$$

which is like the formula for the standard error of the slope, but with two additional terms. Uncertainly declines with increasing sample size n. It is less clear why uncertainty should increase with $\sum x^2$. The reason for this is that uncertainly in the estimate of the intercept increases, the further away from the intercept it is that the mean value of *x* lies. You can see this from the following graphs. On the left is a graph with a low value of $\bar{x}$ and on the right an identical graph (same slope and intercept) but estimated from a data set with a higher value of $\bar{x}$. In both cases there is a 25% variation in the slope. Compare the difference in the prediction of the intercept in the two cases.



Confidence in predictions made with linear regression declines with the square of the distance between the mean value of x and the value at which the prediction is to be made (i.e. with $(x - \bar{x})^2$). Thus, when the origin of the graph is a long way from the mean value of x, the standard error of the intercept will be large, and vice versa. A 95% confidence interval for the intercept, therefore, is

$$CI_a = \text{t (fromtables, } \alpha = 0.025, d.f. = n - 2).SE_a$$

with the same 2-tailed t-value as before. In general, the **standard error for a predicted value** $\hat{y}$ is given by:

$$SE_{\hat{y}} = \sqrt{s^2 \left[\frac{1}{n} + \frac{(x - \bar{x})^2}{SSX} \right]}$$

Note that the formula for the standard error of the intercept is just the special case of this for x = 0 (you should check the algebra of this result as an exercise). An important significance test concerns the question of whether or not the slope of the best fit line is significantly different from zero. If it is not, then the principal of

parsimony suggests that the line should be assumed to be horizontal. If this were the case, then y would be independent of x (it would be a constant, y = a).

Calculations involved in linear regression

The data are in a data frame called regression: the data frame contains the response variable, growth, and the continuous explanatory variable, tannin. You see the 9 values of each by typing the name of the data frame:

```
regression
```

```
  growth tannin
1     12      0
2     10      1
3      8      2
4     11      3
5      6      4
6      7      5
7      2      6
8      3      7
9      3      8
```

The numbers are mean weight gain (mg) of individual caterpillars and tannin content of their diet (%) . You will find it useful to work through the calculations long-hand as we go. The first step is to compute the "famous 5": $\sum x = 36$, $\sum x^2 = 204$, $\sum y = 62$, $\sum y^2 = 536$ and $\sum xy = 175$. Note that $\sum xy$ is 0x12 + 1x10 + 2x8 ... +8x3 = 175. Note the use of the semicolon **;** to separate two directives

```
sum(tannin);sum(tannin^2)
```

```
[1] 36
[1] 204
```

```
sum(growth);sum(growth^2)
```

```
[1] 62
[1] 536
```

```
sum(tannin*growth)
```

```
[1] 175
```

Now calculate the three corrected sums of squares, using the formulas on p.92:

$$SST = 536 - \frac{62^2}{9} = 108.889$$

$$SSX = 204 - \frac{36^2}{9} = 60$$

$$SSXY = 175 - \frac{36 \times 62}{9} = -73$$

then the slope of the best fit line is simply

$$b = \frac{SSXY}{SSX} = \frac{-73}{60} = -1.21666$$

and the intercept is

$$a = \bar{y} - b\bar{x} = \frac{62}{9} + 1.21666\frac{36}{9} = 11.755$$

The regression sum of squares is

$$SSR = b.SSXY = -1.21666 \times -73 = 88.82$$

so that the error sum of squares can be found by subtraction

$$SSE = SST - SSR = 108.89 - 88.82 = 20.07$$

Now we can complete the ANOVA table:

Source	SS	df	MS	F	Probability
Regression	88.82	1	88.82	30.98	0.0008
Error	20.07	7	2.867		
Total	108.89	8			

The calculated F ratio of 30.98 is much larger than the 5% value in tables with 1 degree of freedom in the numerator and 7 degrees of freedom in the denominator. To find the expected F value from tables we use **qf** (quantiles of the F distribution)

```
qf(0.95,1,7)
```

```
[1] 5.591448
```

We can ask the question the other way round. What is the probability of getting an F value of 30.98 if the null hypothesis of b = 0 is true ? We use 1 – **pf** for this

```
1-pf(30.98,1,7)
```

```
[1] 0.0008455934
```

so we can unequivocally reject the null hypothesis that the slope of the line is zero. Increasing dietary tannin leads to significantly reduced weight gain for these caterpillars. We can now calculate the standard errors of the slope and intercept

$$SE_b = \sqrt{\frac{2.867}{60}} = 0.2186$$

$$SE_a = \sqrt{\frac{2.867 \times 204}{9 \times 60}} = 1.041$$

To compute the 95% confidence intervals, we need the 2-tailed value of Student's t with 7 degrees of freedom

```
qt(.975,7)

[1] 2.364624
```

so the 95% confidence intervals for the slope and intercept are given by

$$a = 11.756 \pm 2.463$$

$$b = -1.21666 \pm 0.5169$$

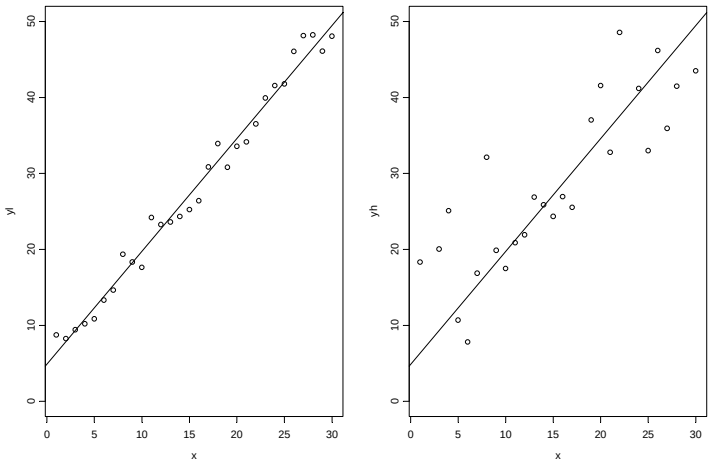
In written work we would put something like this:

"The intercept was 11.76 ± 2.46 (95% CI, n = 9)"

The reader needs to know that the interval is a 95% CI (and not say a 99% interval or a single standard error) and that the sample size was 9. Then they can form their own judgement about the parameter estimate and its unreliability.

Degree of scatter

Knowing the slope and the intercept tells only part of the story. The two graphs below have exactly the same slope and intercept, but they are completely different from one another in their degree of scatter.



It is obvious that we need to be able to quantify the degree of scatter, so that we can distinguish cases like this. In the limit, the fit could perfect, in which case all of the points fall exactly on the regression line. SSE would be zero and SSR = SST. In the opposite case, there is absolutely no relationship between y and x, so SSR = 0, SSE = SST, and the slope of the regression b = 0. Formulated in this way, it becomes clear that we could use some of the quantities already calculated to derive an estimate of scatter. We want our measure to vary from 1.0 when the fit is perfect, to zero when there is no fit at all. Now recall that the total variation in *y* is measured by SST. We can rephrase our question to ask: what fraction of SST is explained by the regression line? A measure of scatter with the properties we want is given by the ratio of SSR to SST. This ratio is called the *coefficient of determination* and is denoted by r^2:

$$r^2 = \frac{SSR}{SST}$$

Its square root, r, is the familiar *correlation coefficient*. Note that because r^2 is always positive, it is better to calculate the correlation coefficient from

$$r = \frac{SSXY}{\sqrt{SSX.SST}}$$

You should check that these definitions of r and r^2 are consistent. In statistical text books you will see the correlation coefficient defined in terms of covariance like this:

$$r = \frac{\mathrm{cov}(x, y)}{\sqrt{\mathrm{var}(x) \times \mathrm{var}(y)}}$$

The two definitions are identical; our version using the sums of squares is simpler because the degrees of freedom in the numerator and denominator have cancelled out.

Using R for regression

The procedure so far should have been familiar. Let us now use R to carry out the same regression analysis. At this first introduction to the use of R each step will be

explained in detail. As we progress, we shall take progressively more of the procedures for granted. There are 6 steps involved in the analysis of a linear model:

- get to know the data;
- suggest a suitable model;
- fit the model to the data;
- subject the model to criticism;
- analyse for influential points;
- simplify the model to its bare essentials.

With regression data, the first step involves the visual inspection of plots of the response variable against the explanatory variable. We need answers to the following questions:

- is there a trend in the data?
- what is the slope of the trend (positive or negative)?
- is the trend linear or curved?
- is there any pattern to the scatter around the trend?

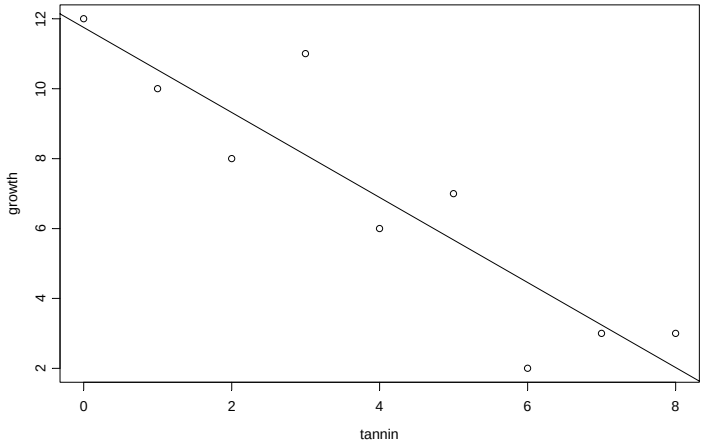
It appears that a linear model would be a sensible first approximation. There are no massive outliers in the data, and there is no obvious trend in the variance of *y* with increasing *x*. We have already attached the data frame called regression (p. 87). The scatterplot is obtained like this:

```
plot(tannin,growth)
```

Note that in the plot directive the arguments are in the order *x* then *y*. The fitted line added by **abline** like this:

```
abline(lm(growth~tannin))
```

where **lm** means linear model, **~** is pronounced tilde, and the order of the variables is *y* ~ *x* (in contrast to the plot directive, above).



Statistical modelling of regression proceeds as follows. We pick a name, say model, for the regression object, then choose a fitting procedure. We could use any one of several but let's keep it simple and use **lm**, the linear model that we have already used in **abline**, earlier. All we do it this:

```
model<-lm(growth~tannin)
```

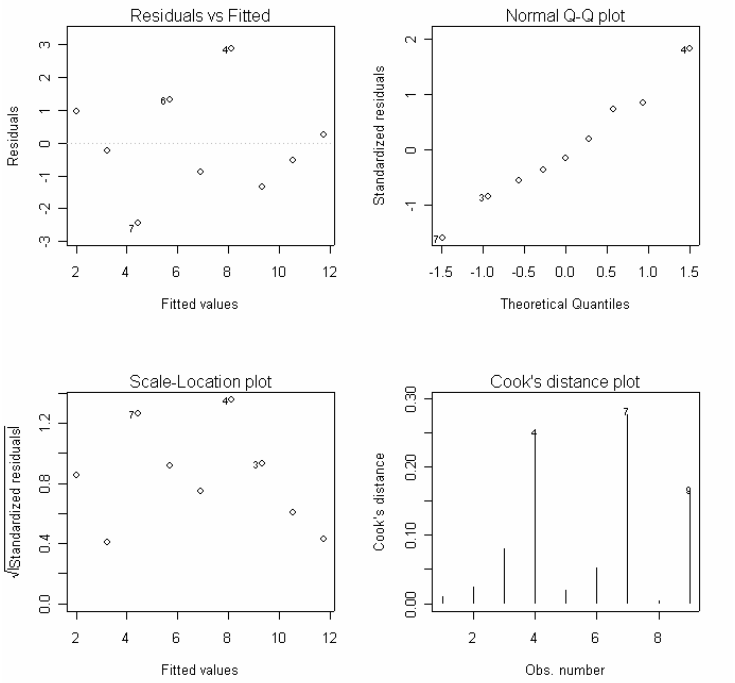
which is read: "model gets a linear model in which growth is modelled as a function of tannin". Now we can do lots of different things with the object called model. The first thing we might do is summarise it:

```
summary(model)
```

```
Call:
lm(formula = growth ~ tannin)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4556 -0.8889 -0.2389  0.9778  2.8944

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  11.7556     1.0408  11.295 9.54e-06 ***
tannin       -1.2167     0.2186  -5.565 0.000846 ***

Residual standard error: 1.693 on 7 degrees of freedom
Multiple R-Squared: 0.8157,     Adjusted R-squared: 0.7893
F-statistic: 30.97 on 1 and 7 DF,  p-value: 0.000846
```

The first part of the output shows the call (the model you fitted). This is useful as a label for the output when you come back to it weeks or months later. Next is a summary of the residuals, showing the largest and smallest as well as the 25% (1Q) and 75% (3Q) quantiles in which the central 50% of the residuals lie. Next the parameter estimates (labelled Coefficients) are printed, along with their standard errors just as we calculated them by hand. The t-values and *p*-values are for tests of the null hypotheses that the intercept and slope are equal to zero (against 2-tailed alternatives that they are not equal to zero). Residual standard error: 1.693 on 7 degrees of freedom is the square root of the error variance from our anova table. Multiple R-Squared: 0.8157 is the ratio SSR/SST we calculated earlier. F-statistic: 30.97 on 1 and 7 degrees of freedom, the *p*-value is 0.0008461 are the last two columns of out anova table. Next, we might plot the object called model. This produces a useful series of diagnostics.

```
par(mfrow=c(2,2))
plot(model)
```



Top left shows the residuals against the fitted values. It is good news because it shows no evidence if variance increasing for larger values of $\hat{y}$, and shows no evidence of curvature. Top right shows the ordered residuals plotted against the quantiles of the standard normal distribution. If, as we hope, our errors really are normal, then this plot should be linear. The data are very well behaved with only point number 4 having a larger residual than expected. The scale location plot is very like the first plot but shows the square root of the standardised residuals against the fitted values (useful for detecting non-constant variance). The last plot shows Cooks distance. This is an influence measure that shows which of the outlying values might be expected to have the largest effects on the estimated parameter values. It is often a good idea to repeat the modelling exercise with the most influential points omitted in order to assess the magnitude of their impact on the structure of the model.

Suppose we want to know the predicted value $\hat{y}$ at tannin = 5.5%. We could write out the equation in calculator mode, using the parameter estimates from the summary table, like this

```
11.7556-1.2167*5.5
```

```
[1] 5.06375
```

Alternatively, we could use the **predict** directive with the object called model. We provide the value for tannin = 5.5 in a **list** in order to protect the vector of x values (called tannin, of course) from being adulterated. The name of the x values to be used for prediction (tannin in this case) must be *exactly* the same as the name of the explanatory variable in the model.

```
predict(model,list(tannin=5.5))
```

```
 5.063889
```

For linear plots we use **abline** for superimposing the model on a scatterplot of the data points. For curved responses, it is sensible to use the predict function to generate the lines, by supplying it with a suitably fine-scale vector of x values (60 or more x points produce a smooth line in most cases: we shall do this later).

Summary

The steps in the regression analysis were:

- data inspection
- model specification
- model fitting
- model criticism.

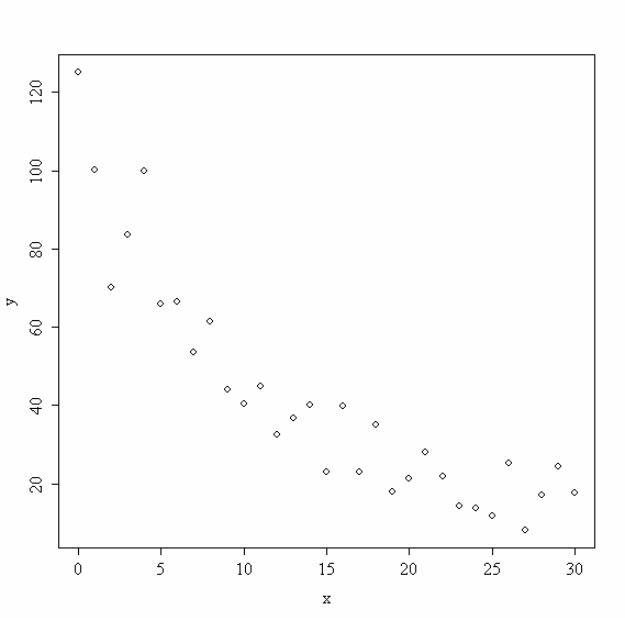
We conclude that the present example is well described by a linear model with normally distributed errors. There are some large residuals, but no obvious patterns in the residuals that might suggest any systematic inadequacy of the model. The slope of the regression line is highly significantly different from zero, and we can be confident that, for the caterpillars in question, increasing dietary tannin reduces weight gain and that, over the range of tannin concentrations considered, the relationship is reasonably linear. Whether the model could be used accurately for predicting what would happen with much higher concentrations of tannin, would need to be tested by further experimentation. It is extremely unlikely to remain linear, however, as this would soon begin to predict substantial weight losses (negative gains), and these could not be sustained in the medium term. We end by removing **rm** the current *x* and *y* vectors

```
par(mfrow=c(1,1))
rm(x,y)
```

Transformation of non-linear responses

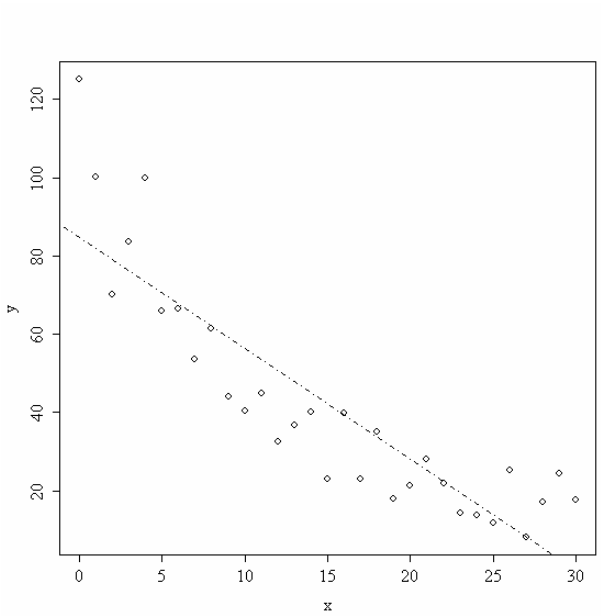
The next example involves data that can not be represented by a straight line. The response variable is dry mass of organic matter remaining after different periods of time in a decomposition experiment. The data are in a file called decay.txt

```
decay<-read.table("c:\\temp\\decay.txt",header=T)
attach(decay)
names(decay)
[1] "x" "y"
plot(x,y)
```



This does not look particularly like a straight-line relationship. Indeed, it makes no scientific sense to fit a linear relationship to these data, because it would begin to predict negative dry matter once time (x) got bigger than 30 or so. Let's put a straight line through the data using **abline** to get a better impression of the curvature.

abline(lm(y~x))



What we see are "groups of residuals". Below about x = 5 most of the residuals are positive, as is the case for the residuals for x bigger than about 25. In between, most of the residuals are negative. This is what we mean be "evidence of curvature". Or more forthrightly "systematic inadequacy of the model". Let's do a linear regression anyway, then look at what the model checking tells us. We call the model object "result" and fit the model as before:

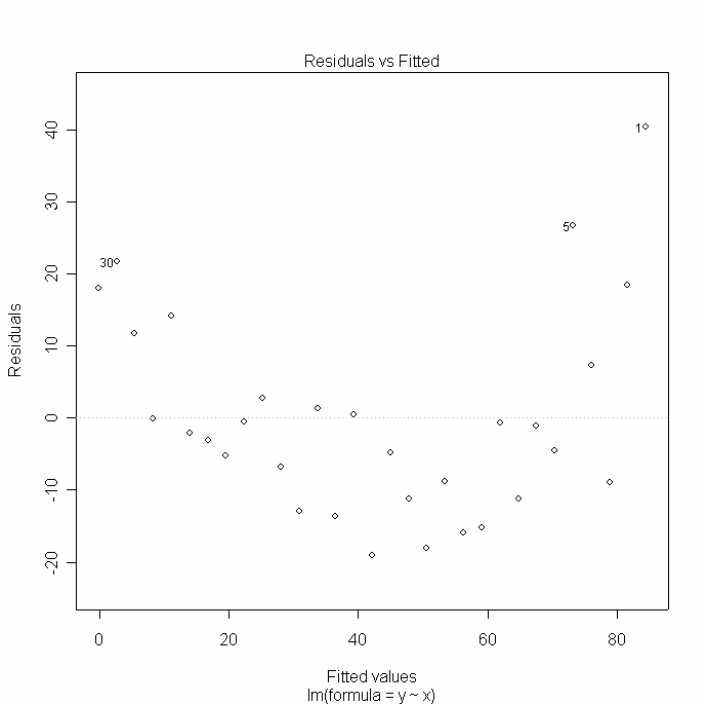
result<-lm(y~x)

summary(result)

```
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-19.065 -10.029  -2.058   5.107  40.447

Coefficients:
            Estimate Std. Error t value  Pr(>|t|)
(Intercept)  84.5534     5.0277   16.82 2.22e-016 ***
x            -2.8272     0.2879   -9.82 9.94e-011 ***

Residual standard error: 14.34 on 29 degrees of freedom
Multiple R-Squared: 0.7688,     Adjusted R-squared: 0.7608
F-statistic:96.44 on1 and 29 degrees of freedom, p-value: 9.939e-011
```

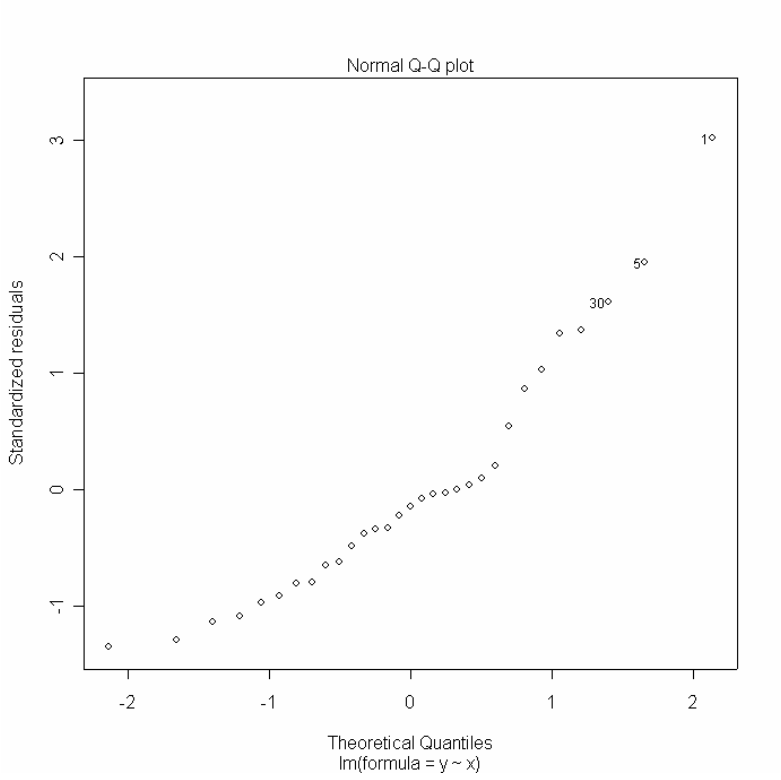
Everything is highly significant. But is the model any good? To test this we carry out the **diagnostic plots**.

plot(result)



The first plot (residuals against fitted values) shows the problem at once. This should look like the sky at night (i.e. no pattern) but in fact, it looks like a letter U. The positive residuals occur only for small and large fitted values of y (as we suspected from our earlier inspection of the abline plot). The message here is simple. A U-shaped plot of residuals against fitted values means that the linear model is inadequate. We need to account for curvature in the relationship between y and x.

The second diagnostic plot looks like this:



This is banana shaped, not straight as it should be. It gets much steeper above theoretical quantiles of about +0.5, further evidence that something is wrong with the model.

The first response to curvature is often transformation, and that is what we shall do here. When we have learned **glm** we shall have lots more options of things to do.

Looking back at the plot and thinking about the fact that it is a decay curve leads us first to try a model in which log(y) is a linear function of x.

You will recall that the equation for exponential decay looks like this:

$$y = ae^{-bx}$$

Taking logs of this, from left to right, we say "log y is ln(y), log a is ln(a), log of 'times' is 'plus', and log of exp(-bx) is –bx ". Remember that exp is the antilog function, so "log of antilog" cancels out (leaving –bx in this case).

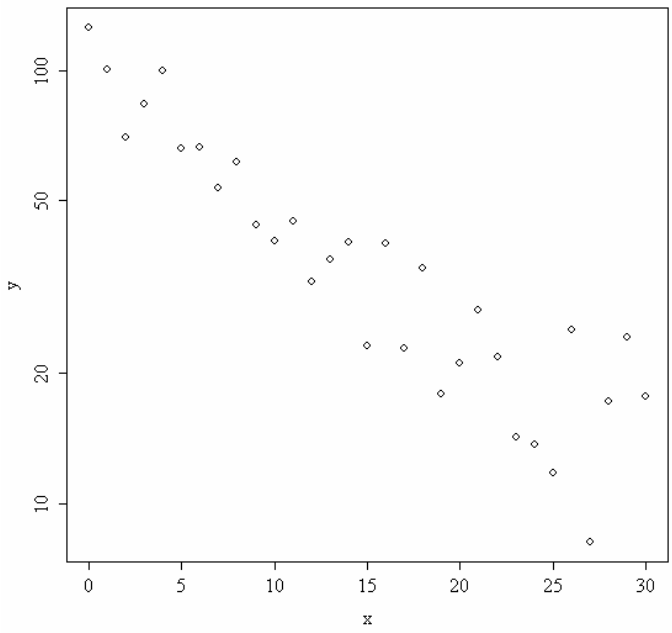
$$\ln(y) = \ln(a) - bx$$

Note that "+-b" becomes "-b" (when plus and minus occur together minus always wins). Now if you replace ln(y) with Y and ln(a) with A you get

$$Y = A - bx$$

which is a straight line: the intercept is A and the slope is –b. This suggests that if we plot of graph of ln(y) against x it should look much straighter:

```
plot(x,y,log="y")
```



That is, indeed, much straighter, but now a new problem has arisen: the variance in y increases as y gets smaller (the scatter of the data around the line is fan-shaped). We shall deal with that later.
For the moment, we fit a different linear model. Instead of

$$y \sim x$$

we shall fit

$$\log(y) \sim x$$

This is fantastically easy to do by replacing “y” in the model formula by “log(y)”. Let’s call the new model object “transformed” like this

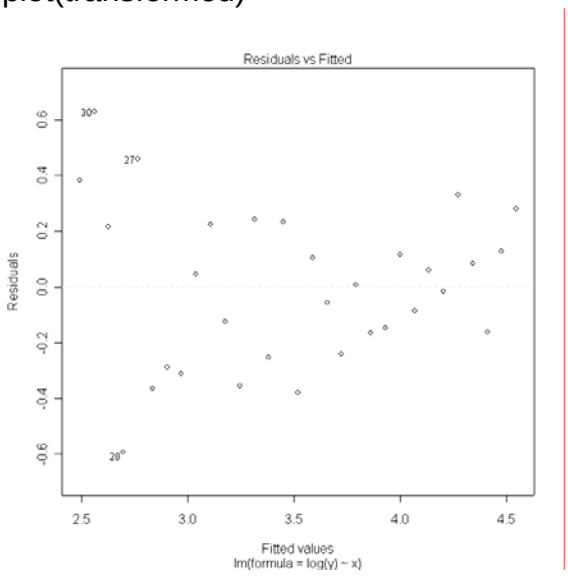
```
transformed <- lm(log(y)~x)
```

```
summary(transformed)
```

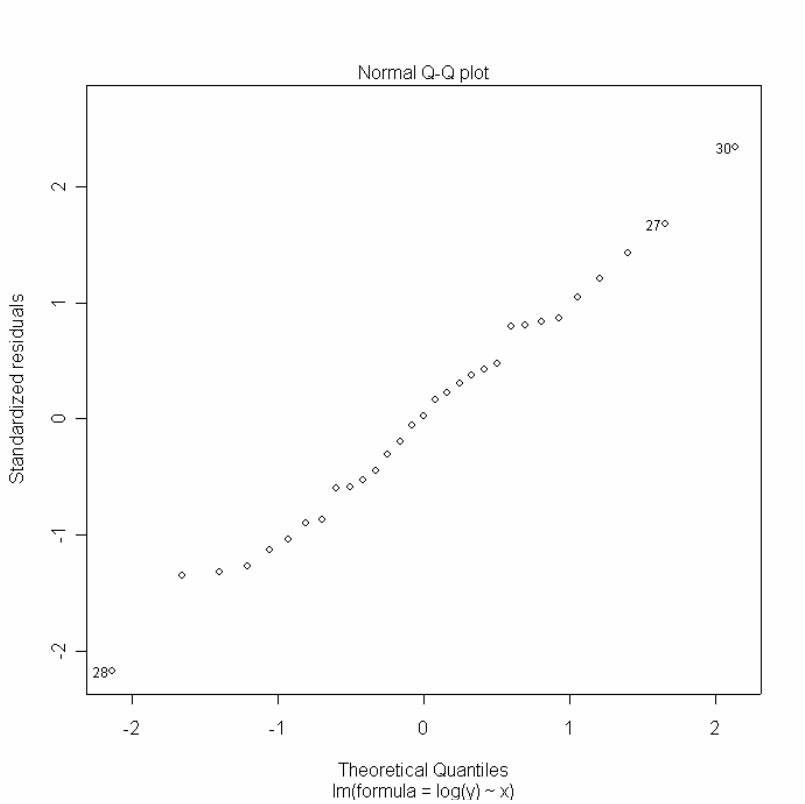
```
Call:
lm(formula = log(y) ~ x)

Residuals:
      Min        1Q    Median        3Q       Max
-0.593515 -0.204324  0.006701  0.219835  0.629730

Coefficients:
             Estimate Std. Error t value  Pr(>|t|)
(Intercept)  4.547386   0.100295   45.34  < 2e-016 ***
x           -0.068528   0.005743  -11.93 1.04e-012 ***
---
Signif. codes:  0  `***'  0.001  `**'  0.01  `*'  0.05  `.'  0.1  ` '

Residual standard error: 0.286 on 29 degrees of freedom
Multiple R-Squared: 0.8308,     Adjusted R-squared: 0.825
F-statistic: 142.4 on 1 and 29 degrees of freedom,      p-value:
1.038e-012
```

As before, everything is highly significant. But does the model behave any better? Lets look at the diagnostic plots.

```
plot(transformed)
```



We have cured the U-shaped residuals, but at the price of introducing non-constancy of variance (in the jargon, this is the dreaded heteroscedasticity). What about the normality of the errors ? This is on the second plot (press the return key):



Not perfect, but very much better than before.

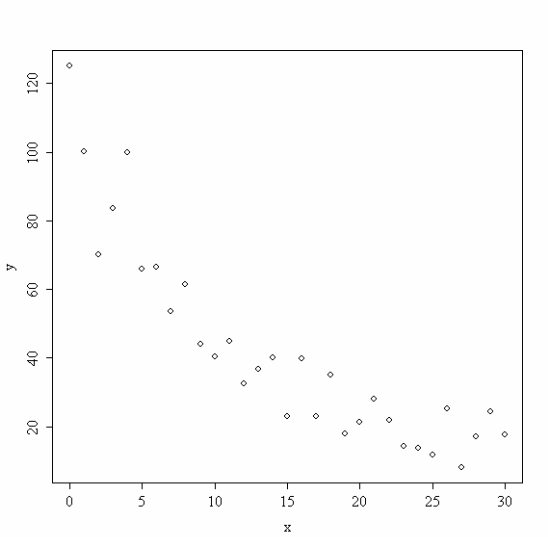
We finish at this point, leaving the problem of non-constant variance until we know about using **glm**s.

Generally, you would want to plot your model as a smooth line through the data. You use the **predict** function for this. There are 3 steps:

- use **seq** to generate a series of values for the x axis (they should have the same range as the data and about 100 increments to give the curve a nice smooth look; if you have too few increments, the curve will look as if it is made up from straight sections (which, of course, it is !))
- put these values into the **predict** function to get the matching y values of the fitted curve (this step can be tricky to understand at first)
- **back transform** the predicted values to get the smooth values for y
- use **lines** to overlay the smooth fitted curve on the data

Let's remind ourselves what the raw data look like

```
plot(x,y)
```



So we want our generated *x* values to go from 0 to 30 and ***they must be called x*** because that is the name of the explanatory variable in the model called 'transformed'. Now there's a slight problem here because *x* is the name of the vector containing our data points, and we don't want to mess with this. The solution is this. Generate the values for drawing the fitted curve under a ***different*** name, *smoothx*, like this:

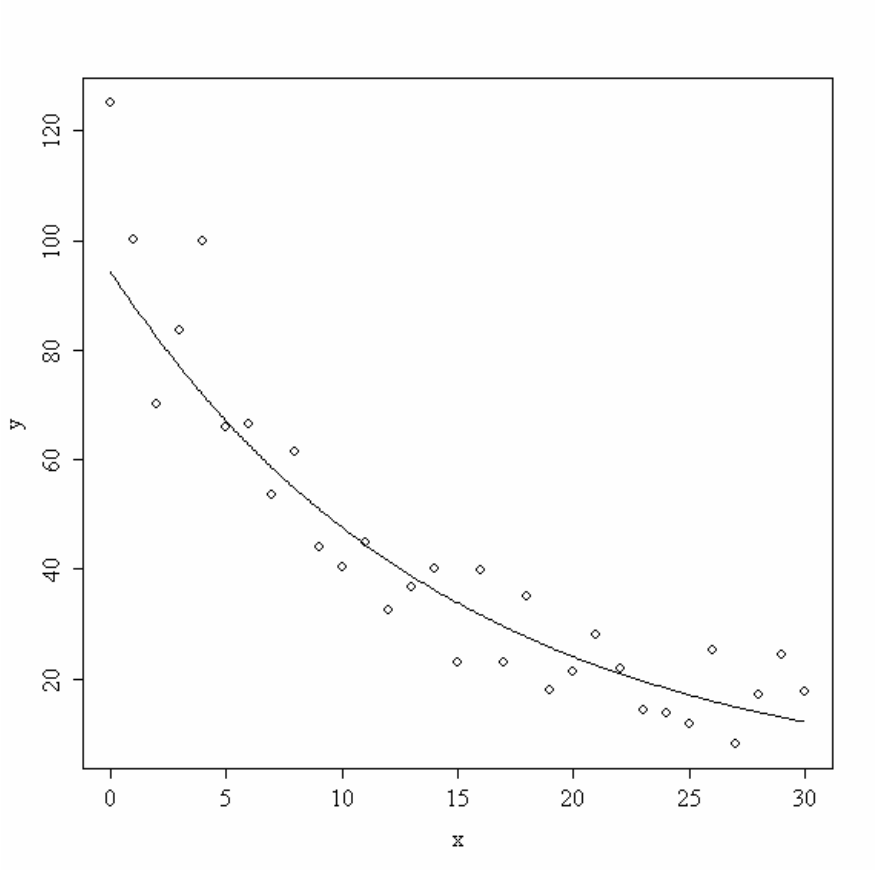
```
smoothx<-seq(0,30,0.1)
```

Fool the model into thinking that the data are called *x* rather than *smoothx* by using a **list** like this **list(x=smoothx)** inside the **predict** function. At the same time, we need to back transform the predicted values (which are in logs) using **exp** (the antilog function):

```
smoothy<-exp(predict(transformed,list(x=smoothx)))
```

It is important that you understand this. You will want to use predict a lot. We **predict** using the current model, which is called 'transformed'. This takes values of *x* because that was the name of the explanatory variable we used in fitting the model log(y)~x We use **list** to coerce the *smoothx* values into a vector that will be known internally as *x*, but which will not alter the values of our data called *x*. Finally, we back transform the predicted values to get our *smoothy* values on the untransformed scale. Since the model was log(y)~x, the appropriate back transformation is exp(predicted). Now (finally !) we can draw the fitted model through the data, using **lines**:

lines(smoothx,smoothy)



Summary

- regression involves estimating parameter values from data
- there are always lots of different possible models to describe a given data set
- model choice is a big deal (here we chose exponential in preference to linear)
- always use diagnostic plots to check the adequacy of your model after fitting
- the two big problems are **non-constant variance** and **non-normal errors**
- one solution for curvature in the relationship between y and x is transformation
- transformation can be specified simply in the model formula log(y) ~ x
- use predict to generate smooth curves for plotting through the scatter of data
- generate the values of x for calculating fitted values of y under a *different* name
- use list to coerce the different name into the original name: e.g. **list(x=smoothx)**
- don't forget to back transform before you draw the curve using **lines**

STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

5. ANALYSIS OF VARIANCE

Instead of fitting continuous, measured variables to data (as in regression), many experiments involve exposing experimental material to a range of discrete *levels* of one or more categorical variables known as *factors*. Thus, a factor might be drug treatment for a particular cancer, with 5 levels corresponding to a placebo plus 4 new pharmaceuticals. Alternatively, a factor might be mineral fertilizer, where the 4 levels represented 4 different mixtures of nitrogen, phosphorus and potassium. Factors are often used in experimental designs to represent statistical *blocks*; these are internally homogeneous units in which each of the experimental treatments is repeated. Blocks may be different fields in an agricultural trial, different genotypes in a plant physiology experiment, or different growth chambers in a study of insect photoperiodism.

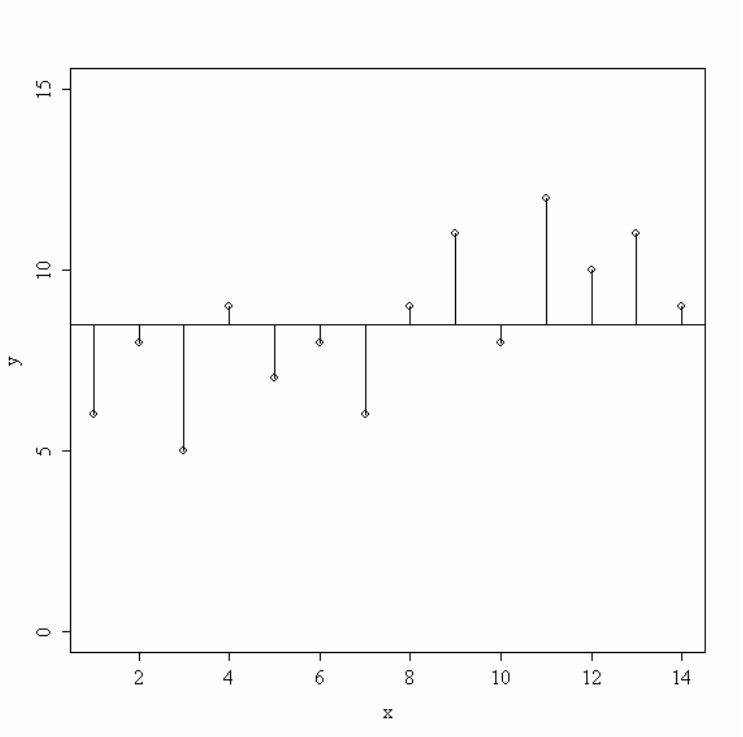
It is important to understand that regression and anova are identical approaches except for the nature of the explanatory variables. For example, it is a small step from having three levels of a shade factor (say light, medium and heavy shade cloths) then carrying out a one-way analysis of variance, to measuring the light intensity in the three treatments and carrying out a regression with light intensity as the explanatory variable. As we shall see later on, some experiments combine regression and analysis of variance by fitting a series of regression lines, one in each of several levels of a given factor (this is called analysis of covariance; see Exercises 6).

Statistical Background

The emphasis in anova has traditionally been on hypothesis testing. The aim of an analysis of variance in R is to estimate means and standard errors of differences between means. Comparing two means by a t-test involved calculating the difference between the two means, dividing by the standard error of the difference, and then comparing the resulting statistic with the value of Student's t from tables (or better still, using **qt** to calculate the critical value; Exercies 3). The means are said to be significantly different when the calculated value of t is larger than the critical value. For large samples (n > 30) a useful rule of thumb is that a t-value greater than 2 is significant. In Analysis of Variance, we are concerned with cases where we want to compare 3 or more means. For the 2-sample case, the t-test and the anova are identical, and the t-test is to be preferred because it is simpler.

It is not at all obvious how you can analyse differences between mean by looking at variances. But this is what analysis of variance is all about. An example should make clear how this works. To keep things simple, suppose we have just two levels of a

single factor. We plot the data in the order in which they were measured: first for the first level of the factor and then for the second level.



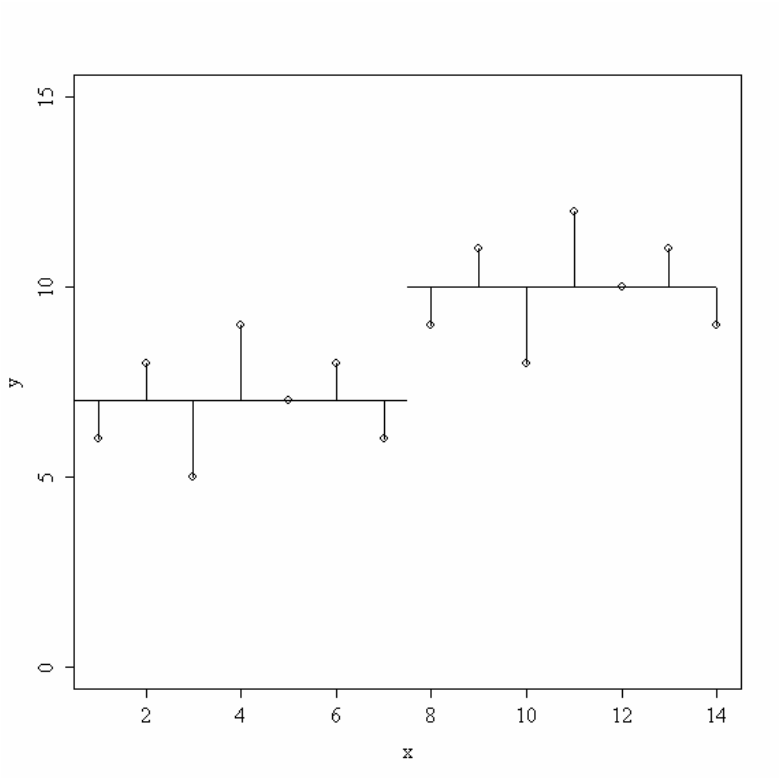
This shows the variation in the data set as a whole. SST, the total variation about the overall mean value of y, is the sum of the squares of these differences:

$$SST = \sum (y - \bar{\bar{y}})^2$$

Next we can fit each of the separate means, $\bar{y}_A$ and $\bar{y}_B$, and consider the sum of squares of the differences between each y value and its own treatment mean. We call this SSE, the error sum of squares, and calculate it like this:

$$SSE = \sum (y_A - \bar{y}_A)^2 + \sum (y_B - \bar{y}_B)^2$$

On the graph, the differences from which SSE is calculated look like this:



Now ask yourself this question. If the two means were the same, what would be the relationship between SSE and SST ? After a moments though you should have been able to convince yourself that if the means are the same, then SSE is the same as SST, because the two horizontal lines in the last plot would be in the same position as the single line in the earlier plot. Now what if the means were significantly different from one another? What would be the relationship between SSE and SST in this case? Which would be the larger? Again, it should not take long for you to see that if the means *are* different, then SSE will be less than SST. Indeed, in the limit, SSE could be zero if the replicates from each treatment fell exactly on their respective means. This is how analysis of variance works. It's amazing but true. *You can make inferences about differences between means by looking at variances* (well, at sums of squares actually, but more of that later).

We can calculate the difference between SST and SSE, and use this as a measure of the difference between the treatment means; this is traditionally called the *treatment sum of squares*, and is denoted by SSA:

$$SSA = SST - SSE$$

The technique we are interested in, however, is analysis of variance, not analysis of sums of squares. We convert the sums of squares into variances by dividing by their degrees of freedom. In our example, there are two levels of the factor and so there is 2-1=1 degree of freedom for SSA. In general, we might have k levels of any factor and hence k-1 d.f. for treatments. If each factor level were replicated n times, then there would be n-1 d.f. for error within each level (we lose one degree of freedom for each individual treatment mean estimated from the data). Since there are k levels, there would be $k(n-1)$ d.f. for error in the whole experiment. The total number of

numbers in the whole experiment is *kn*, so total d.f. is *kn*-1 (the single degree is lost for our estimating the overall mean, $\bar{\bar{y}}$). As a check in more complicated designs, it is useful to make sure that the individual component degrees of freedom add up to the correct total.

$$kn-1 = k-1+k(n-1) = k-1+kn-k = kn-1$$

The divisions for turning the sums of squares into variances are conveniently carried out in an anova table:

Source	SS	d.f.	MS	F	Critical F
Treatment	*SSA*	*k-1*	$MSA = \frac{SSA}{k-1}$	$F = \frac{MSA}{s^2}$	qf(0.95, k-1,k(n-1))
Error	*SSE*	*k(n-1)*	$s^2 \frac{SSE}{k(n-1)}$		
Total	*SST*	*kn-1*			

Each element in the sums of squares column is divided by the number in the adjacent degrees of freedom column to give the variances in the mean square column (headed MS for mean square). The significance of the difference between the means is then assessed using an F test (a variance ratio test). The treatment variance MSA is divided by the error variance, s^2, and the value of this test statistic is compared with the critical value of F using **qf** (the quantiles of the F distribution, with p = 0.95, *k*-1 degrees of freedom in the numerator, and *k*(*n*-1) degrees of freedom in the denominator). If you need to look up the critical value of F in tables, remember that you look up the numerator degrees of freedom (on top of the division) across the *top* of the table, and the denominator degrees of freedom down the rows. If the test statistic is larger than the critical value we *reject* the null hypothesis

$$H_0 : \textit{all the means are the same}$$

and accept the alternative

$$H_1 : \textit{at least one of the means is significantly different from the others}$$

If the test statistic is less than the critical value, then it could have arisen due to chance alone, and so we accept the null hypothesis.

Another way of visualising the process of anova is to think of the relative amounts of sampling variation between replicates receiving the same treatment (i.e. between individual samples in the same level), and between different treatments (i.e. between-level variation). When the variation between replicates within a treatment is large compared to the variation between treatments, we are likely to conclude that the difference between the treatment means is not significant. Only if the variation between replicates within treatments is relatively small compared to the differences between treatments, will we be justified in concluding that the treatment means are significantly different.

Calculations in anova

The definitions of the various sums of squares can now be formalised, and ways found of calculating their values from samples. The total sum of squares, SST is defined as:

$$SST = \sum y^2 - \frac{(\sum y)^2}{kn}$$

just as in regression (see Exercises 4). Note that we divide by the total number of numbers we added together to get $\sum y$ (the grand total of all the y's) which is *kn*. It turns out that the formula that we used to define SSE is rather difficult to calculate (see above), so we calculate the treatment sums of squares SSA, and obtain SSE by difference.

The treatment sum of squares, SSA, is defined as:

$$SSA = \frac{\sum C^2}{n} - \frac{(\sum y)^2}{kn}$$

where the new term is C, the *treatment total*. This is the sum of all the *n* replicates within a given level. Each of the *k* different treatment totals is squared, added up, and then divided by *n* (the number of numbers added together to get the treatment total). The formula is slightly different if there is unequal replication in different treatments, as we shall see below. The meaning of C will become clear when we work through the example later on. Notice the symmetry of the equation. The second term on the right hand side is also divided by the number of numbers that were added together (*kn*) to get the total ($\sum y$) which is squared in the numerator. Finally,

$$SSE = SST - SSA$$

to give all the elements required for completion of the anova table.

Assumptions of anova

You should be aware of the assumptions underlying the analysis of variance. They are all important, but some are more important than others:

- random sampling
- equal variances
- independence of errors
- normal distribution of errors
- additivity of treatment effects

These assumptions are well and good, but what happens if they do not apply to the data you propose to analyse? We consider each case in turn.

Random sampling

If samples are not collected at random, then the experiment is seriously flawed from the outset. The process of randomisation is sometimes immensely tedious, but none the less important for that. Failure to randomise properly often spoils what would otherwise be well-designed ecological experiments (see Hairston 1989 for examples). Unlike the other assumptions of anova, there is nothing we can do to rectify non-random sampling after the event. If the data are not collected randomly they will probably be biased. If they are biased, their interpretation is bound to be equivocal.

Equal variances

It seems odd, at first, that a technique which works by comparing variances should be based on the assumption that variances are equal. What anova actually assumes, however, is that the *sampling errors* do not differ significantly from one treatment to another. The comparative part of anova works by comparing the variances that are due to differences between the treatment means with the variation between samples within treatments. The contributions towards SSA are allowed to vary between treatments, but the contributions towards SSE should not be significantly different.

Recall the folly of attempting to compare treatment means when the variances of the samples are different in the example of three commercial gardens producing lettuce (see Practical 3). There were 3 important points to be made from these calculations:

- two treatments can have different means and the same variance
- two treatments can have the same mean but different variances
- when the variances are different, the fact that the means are identical does *not* mean that the treatments have identical effects.

We often encounter data with non-constant variance, and R is ideally suited to deal with this. With Poisson distributed data, for example, the variance is equal to the mean, and with binomial data the variance (npq) increases to a maximum and then declines with the mean (np). Many data on time to death (or time to failure of a component) exhibit the property that the coefficient of variation is roughly constant, which means that a plot of log(variance) against log(mean) increases with a slope of approximately 2 (this is known as Taylor's Power Law); gamma errors can deal with this. Alternatively, the response variable can be transformed (see Practical 4) to stabilise the variance, and the analysis carried out on the new variable.

Independence of Errors and Pseudoreplication

It is very common to find that the errors are not independent from one experimental unit to another. We know, for example, that the response of an organism is likely to be influenced by its sex, age, body size, neighbours and their sizes, history of development, genotype, and many other things. Differences between our samples in these other attributes would lead to non-independence of errors, and hence to bias in our estimation of the difference that was due to our experimental factor. For instance, if more of the irrigated plots happened to be on good soil, then increases in shoot weight might be attributed to the additional water when, in fact, they were due to the

higher level of mineral nutrient availability on those plots. The ways to minimise the problems associated with non-independence of errors are:

- use block designs, with each treatment combination applied in every block (i.e. repeat the whole experiment in several different places)
- divide the experimental material up into homogeneous groups at the outset (e.g. large, medium and small individuals), then make these groups into statistical blocks, and apply each treatment within each of them
- have high replication
- insist on thorough randomisation.

No matter how careful the design, however, there is always a serious risk of non-independence of errors. A great deal of the skill in designing good experiments is in anticipating where such problems are likely to arise, and in finding ways of blocking the experimental material to maximise the power of the analysis of variance (see Hurlbert, 1984). This topic is considered in more detail later.

Normal distribution of errors

Data often have non-normal error distributions. The commonest form of non-normality is skewness, in which the distribution has a long 'tail' at one side or the other. A great many collections of data are skewed to the right, because most individuals in a population are small, but a few individuals are very large indeed (well out towards the right hand end of the size axis). The second kind of non-normality that is encountered is kurtosis. This may arise because there are long tails on both sides of the mean so that the distribution is more 'pointed' than the bell-shaped normal distribution (so called *leptokurtosis*), or conversely because the distribution is more 'flat-topped' than a normal (so called *platykurtosis*). Finally, data may be bimodal or multi-modal, and look nothing at all like the normal curve. This kind of non-normality is most serious, because skewness and kurtosis are often cured by the same kinds of transformations that can be used to improve homogeneity of variance. If a set of data are strongly bimodal, then it is clear that any estimate of central tendency (mean or median) is likely to be *unrepresentative of most of the individuals* in the population.

Additivity of treatment effects in multi-factor experiments

In 2-way analysis there are two kinds of treatments (say drugs and radiation therapy) each with 2 or more levels. Anova is based on the assumption that the effects of the different treatments are additive. This means that if one of the drug treatments produced a response of 0.4 at low radiation, then the drug will produce the same response at high radiation. Where this is *not* the case, and the response to one factor depends upon the level of another factor (i.e. where there is *statistical interaction*), we must use factorial experiments. Even here, however, the model is still assumed to be additive. When treatments effects are multiplicative (e.g. temperature and dose are multiplicative in some toxicity experiments), then it may be appropriate to transform the data by taking logarithms in order to make the treatment effects additive. In R, we could specify a log link function in order to achieve additivity in a case like this.

A worked example of One-way Analysis of Variance

To draw this background material together, we shall work through an example by hand. In so doing, it will become clear what R is doing during its analysis of the same data. The data come from a simple growth room experiment, in which the response variable is growth (mm) and the categorical explanatory variable is a factor called Photoperiod with 4 levels: Very short, Short, Long and Very long daily exposure to light. There were 6 replicates of each treatment:

V short	Short	Long	V. long
2	3	3	4
3	4	5	6
1	2	1	2
1	1	2	2
2	2	2	2
1	1	2	3

To carry out a 1-way ANOVA we aim to partition the total sum of squares SST into just two parts: variation attributable to differences between the treatment means, SSA, and the unexplained variation which we call SSE, the error sum of squares.

$$SST = 175 - \frac{57^2}{24} = 39.625$$

$$SSA = \frac{10^2 + 13^2 + 15^2 + 19^2}{6} - \frac{57^2}{24} = 7.125$$

$$SSE = SST - SSA = 39.625 - 7.125 = 32.5$$

Source	SS	d.f.	MS	F
Photoperiod	7.125	3	2.375	1.462
Error	32.5	20	$S^2 = 1.625$	
Total	39.625	23		

```
oneway<-read.table("c:\\temp\\oneway.txt",header=T)
attach(oneway)
names(oneway)
```

```
[1] "Growth"      "Photoperiod"
```

We begin by calculating the mean growth at each photoperiod, using **tapply** in the normal way

```
tapply(Growth,Photoperiod,mean)
```

```
 Long              Short      Very.long      Very.short
  2.5           2.166667       3.166667        1.666667
```

The values look fine, with mean growth increasing with the duration of illumination. But the order in which the factor levels are printed is in alphabetical order. This usually does not matter, but here it is inconvenient because the factor Photoperiod is ordered. We can fix this very simply by declaring photoperiod to be an ordered factor, and providing an ordered list of the factor levels to reflect the fact that Very short < Short < Long < Very long.

```
Photoperiod<-ordered(Photoperiod,levels=c("Very.short","Short","Long","Very.long"))
```

Now when we use **tapply**, the means are printed in the desired sequence:

```
tapply(Growth,Photoperiod,mean)
```

```
 Very.short        Short      Long           Very.long
   1.666667    2.166667       2.5            3.166667
```

The one-way analysis of variance is carried out using the **aov** directive. The model formula is written like this:

Response.variable ~ Explanatory.variable

where it is understood that the explanatory variable is a factor. Although in this case we know that Photoperiod must be a factor because it has text as the factor levels, it is useful to know how to check that a variable is a factor (especially for variables with numbers as factor levels). We use the **is.factor** directive to find this out

```
is.factor(Photoperiod)
```

```
[1] TRUE
```

so Photoperiod *is* a factor. All we need to do now is to give a name to the object that will contain the results of the analysis (one.way), and carry out the **aov**

```
one.way<-aov(Growth~Photoperiod)
```

Nothing happens until we ask for the results. All of the usual generic functions can be used for ANOVA including **summary** and **plot**.

```
summary(one.way)
```

```
                Df  Sum of Sq Mean Sq       F Value      Pr(F)
Photoperiod      3     7.125     2.375      1.461538  0.2550719
  Residuals     20    32.500    1.625
```

It is abundantly clear that there is no significant difference between the mean growth rates with different periods of illumination. An F value of 1.46 will arise due to chance when the means are all the same with a probability greater than 1 in 4 (for significance, you will recall, we want this probability to be less than 0.05). The differences we noted at the beginning are not significant because the variance is so large ($s^2 = 1.625$) and the error degrees of freedom (d.f. = 20) rather small.

Model checking involves **plot**(one.way). You will see that while there is slight evidence of heteroscedasticity (i.e. there is some tendency for the variance to increase as the fitted values increase), but there are strong signs of non-normality in the residuals (J-shape not linear). This is perhaps not surprising given that the data are small integers. We return to these issues later. For the time being, we continue with the analysis of the same data set, but taking further information into account.

Two-way Analysis of Variance

Continuing the analysis from where we left off, we had fitted a 4-level factor for Photoperiod, but the unexplained variation, SSE, was very large (32.5). So large in fact, that the differences between mean growth in the different photoperiods was not significant. Fortunately, the experiment was well designed. Material from 6 plant Genotypes was cloned and Photoperiod treatments were allocated at random to each of 4 clones of the same Genotype. Each Photoperiod was allocated once to each Genotype. There was no replication of Genotypes within Photoperiods, so we can not carry out a Factorial ANOVA. But we can carry out a 2-way ANOVA. The sum of squares attributable to Photoperiod is unchanged, and we can add this directly to our new, expanded ANOVA table. The total sum of squares will also be unchanged:

Source	SS	d.f.	MS	F
Photoperiod	7.125	3		
Genotype		5		
Error		15		
Total	39.625	23		

We know that the sums of squares for Genotype and Error must add up to 32.5, but we need to work out one of them, before we can obtain the other by subtraction. Before we do that, it is worth noting that we can fill in the degrees of freedom column without doing any calculations. There a 6 levels of Genotype so there must be 6-1 = 5 d.f. for Genotype. We know that d.f. must always add up to the total, so

$$\text{error d.f.} = 23 - 5 - 3 = 15$$

There is an analytical way to calculate error d.f.. Technically, in an analysis like this without any replication, the error term is the interaction between the constituent factors. You will recall that interaction degrees of freedom, are the product of the component degrees of freedom. So, in this case with (c-1) = 3 d.f. for Photoperiod and (r-1) = 5 d.f. for Genotype, we have

$$\text{error d.f.} = (r-1)(c-1) = 5x3 = 15$$

The only extra calculation we need to do is for the Genotype sum of squares. The procedure is exactly analogous to computing the Photoperiod sum of squares. Photperiod differences were reflected in the column totals. Likewise Genotype differences are reflected in the row totals:

Genotype	V short	Short	Long	V. long	Row totals
A	2	3	3	4	**12**
B	3	4	5	6	**18**
C	1	2	1	2	**6**
D	1	1	2	2	**6**
E	2	2	2	2	**8**
F	1	1	2	3	**7**

So we use the squares of the row totals in computing SSB. Note that we divide by the number of numbers in a row (4 in this case; 1 for each photoperiod). For SSA we divided by the number of numbers in a column (6 in that case). In general, in ANOVA, you add up the squares of sub totals and *divide by the number of numbers that were added together to get that sub total* . The Genotype sum of squares is therefore calculated as

$$SSB = \frac{12^2 + 18^2 + 6^2 + 6^2 + 8^2 + 7^2}{4} - \frac{57^2}{24} = 27.875$$

and the new, much smaller SSE is obtained by difference

$$SSE = SST - SSA - SSB = 39.625 - 7.125 - 27.875 = 4.625$$

The 2-way ANOVA table can now be completed

Source	SS	d.f.	MS	F
Photoperiod	7.125	3	2.375	7.703
Genotype	27.875	5	5.575	18.08
Error	4.625	15	$s^2 = 0.308$	
Total	39.625	23		

The interpretation of the experiment is completely different. Now we conclude that Photoperiod has a highly significant effect on mean growth ($F = 7.703$, d.f. =3,15, $p<0.01$). The moral is that good experimental design, aimed at reducing variation between individuals (blocking by Genotype in this case), can pay huge dividends in terms of the likelihood of detecting biologically important differences.

We tidy up by removing (**rm**) the variable names used in the previous one-way analysis, and detaching the old dataframe called oneway

```
rm(Growth, Photoperiod)
detach(oneway)
```

Two-way ANOVA in R

```
twoway<-read.table("c:\\temp\\twoway.txt",header=T)
attach(twoway)
names(twoway)
```

```
[1] "Growth"      "Photoperiod"    "Genotype"
```

We begin by comparing the mean growth rates of the 6 genotypes, using **tapply**:

```
tapply(Growth,Genotype,mean)
```

```
 A      B        C     D    E        F
 3    4.5      1.5   1.5    2     1.75
```

Evidently, there are substantial differences between the mean growth rates of the different genotypes. This variation was formerly included in the error variance, so by removing the variation attributable to genotype, we will make the error variance smaller, and hence the significance of the difference between the photoperiod means greater. The 2-way ANOVA is carried out simply by specifying the names of two explanatory variables (both must be factors) in the model formula, linked by a plus sign (+). We call the object containing the results of the analysis two.way, and proceed as follows:

```
two.way<-aov(Growth~Genotype+Photoperiod)
```

Nothing happens until we specify the form of output we would like.

```
summary(two.way)
```

```
             Df    Sum of Sq   Mean Sq     F Value          Pr(F)
   Genotype 5     27.875      5.575000    18.08108    0.000007092
Photoperiod 3      7.125      2.375000     7.70270    0.002404262
  Residuals 15     4.625      0.308333
```

The 2-way ANOVA table shows the dramatic effect of including Genotype in the model. Far from being insignificant (as it was in the 1-way ANOVA) the difference between the Photoperiod means is now highly significant ($p < 0.0025$). This example

highlights the enormous potential benefits of introducing blocking into the experimental design. If the photoperiod treatments had been allocated to different genotypes at random, we would not have obtained a significant result. By ensuring that every photoperiod was applied to every genotype, the experimental design was greatly improved. Randomisation was carried out, but it involved deciding at random which of the 4 clonal plants from each genotype was allocated to a given photoperiod treatment.

Model criticism involves using plot(two.way). You will see that the non-normality problems that looked so severe after the 1-way analysis have been completely cured; evidently it was the differences between the genotype means that were the principal cause of the difficulty. If we had not known the identity of the genotypes, and hence been unable to carry out the 2-way analysis, we would have been left with this problem. We return to the issue of unexplained variation under the topic of **overdispersion**, later on.

Interpretation of the parameters of 2-way Anova is described later.

Two-way Factorial Analysis of Variance

The new concept here is the calculation of the *interaction sum of squares*. Like all components of ANOVA it is based on a sum of squared totals, divided by the number of numbers that were added together to get each total. For interactions, the sub totals we need are often not directly available (like the row totals and column totals of the standard 2-way analysis). To make calculation as straightforward as possible, it is a good idea to draw up a table of interaction totals; the levels of Factor A make up the rows and the levels of Factor B make up the columns. Each cell of the table contains the sum of the replicates in each factor combination. We call these totals Q (don't ask me why).

Now the interaction sum of squares, SSAB, is calculated like this:

$$SSAB = \frac{\sum Q^2}{n} - SSA - SSB - CF$$

because the interaction sub totals contain the main effects of Factor A and Factor B as well as the interaction effect we are after. That is why we subtract SSA and SSB in calculating SSAB.

Source	SS	d.f.	MS	F
Factor A	SSA	a-1	$\frac{SSA}{(a-1)}$	$\frac{MSA}{s^2}$
Factor B	SSB	b-1	$\frac{SSB}{(b-1)}$	$\frac{MSB}{s^2}$
Interaction A:B	SSAB	(a-1)(b-1)	$\frac{SSAB}{(a-1)(b-1)}$	$\frac{MSAB}{s^2}$
Error	SSE	ab(n-1)	$s^2 = \frac{SSE}{ab(n-1)}$	
Total	SST	abn-1		

```
factorial<-read.table("c:\\temp\\factorial.txt",header=T)
factorial
```

```
   growth diet  coat
1     6.6    A light
2     7.2    A light
3     6.9    B light
4     8.3    B light
5     7.9    C light
6     9.2    C light
7     8.3    A  dark
8     8.7    A  dark
9     8.1    B  dark
10    8.5    B  dark
11    9.1    C  dark
12    9.0    C  dark
```

There are 3 levels of diet (A, B and C), 2 levels of coat colour (light and dark) and 2 replicates. The analysis proceeds very simply by using the * operator. This means fit all the main effects and their interactions. Thus the **model formula**

growth ~ diet * coat

is shorthand for

growth ~ diet + coat + diet : coat

where the colon operator means “the interaction between”.

```
attach(factorial)

model<-aov(growth~diet*coat)

summary(model)
```

```
            Df  Sum Sq Mean Sq F value  Pr(>F)
diet         2 2.66000 1.33000  3.6774 0.09069 .
coat         1 2.61333 2.61333  7.2258 0.03614 *
diet:coat    2 0.68667 0.34333  0.9493 0.43833
Residuals    6 2.17000 0.36167
```

The interaction sum of squares is 0.68667, and we should look at how it was obtained. The **interaction totals** are the sums of the 2 replicates in each of the 6 combinations of factor levels:

tapply(growth,list(coat,diet),sum)

```
          A    B    C
dark  17.0 16.6 18.1
light 13.8 15.2 17.1
```

We need the sum of the squares of these subtotals, then we divide by 2 (because each of the subtotals is the sum of 2 numbers).

SSAB<-sum(as.vector(tapply(growth,list(coat,diet),sum))^2)/2

Note the use of the **as.vector** directive to turn the table produced by **tapply** into a set of numbers that we can use in calculations. Now we need to compute the correction factor

CF<-sum(growth)^2/length(growth)

Now, as we saw earlier, the interaction sum of squares is SSAB–SSA-SSB-CF. We can steal the values of SSA and SSB from the anova table above (2.66 and 2.6133)

SSAB-CF-2.66-2.61333

[1] 0.68667

which, to our considerable relief, is the interaction sum of squares as computed by R.

The output of the anova table is interpreted like this. We always start by looking at the interaction rather than the main effects. The F value of 0.9493 falls well short of significance, so there is no evidence at all for an interaction between diet and coat colour. Now for the main effects. There is a significant effect of coat colour ($p < 0.04$) but not of diet ($p > 0.05$). Now we use **plot(model)** to carry out model criticism. The variance is constant and the errors are normal, so that is good.

It is a good idea at this stage to do model simplification to remove any non-significant parameters. We use a new directive **update** to remove the interaction term. It works like this:

model2<-update(model , ~ . - diet:coat)

which is read like this. The new model (called “model2” here) gets an **update** of the earlier model (called “model” in the first argument in update). The syntax is now very

important: it is “comma tilde dot minus”. This says take the whole of “model” (that is the “tilde dot” bit, where dot means “all of”), and remove from it (hence the minus sign) the interaction diet : coat (which is read “diet by coat”). Now we have 2 models: a complicated one (called model) and a simpler one (called model2). R is **brilliant** at comparing models. We use the **anova** directive to compare 2 or more models, like this (make sure you understand the difference between **aov** and **anova**)

```
anova(model,model2)
```

```
Analysis of Variance Table

Model 1: growth ~ diet * coat
Model 2: growth ~ diet + coat

  Res.Df Res.Sum Sq Df   Sum Sq F value Pr(>F)
1      6    2.17000
2      8    2.85667 -2 -0.68667  0.9493 0.4383
```

This says that the simpler *model2* is not significantly different in its explanatory power than the more complicated *model*. In other words, the model simplification is justified. Let’s look at the output of *model2*:

```
summary(model2)
```

```
             Df  Sum Sq Mean Sq F value  Pr(>F)
diet          2 2.66000 1.33000  3.7246 0.07190 .
coat          1 2.61333 2.61333  7.3186 0.02685 *
Residuals     8 2.85667 0.35708
```

There is a hint of an effect of diet (p = 0.0719) but we are ruthless in the elimination of non-significant terms, so we shall try leaving that out as well:

```
model3<-update(model2, ~. -diet)
```

```
anova(model2,model3)
```

```
Analysis of Variance Table

Model 1: growth ~ diet + coat
Model 2: growth ~ coat

  Res.Df Res.Sum Sq Df  Sum Sq F value Pr(>F)
1      8     2.8567
2     10     5.5167 -2 -2.6600  3.7246 0.0719 .
```

This gives us exactly the same p value as in model 2, but when we come onto more complicated statistical models this will not always be the case, and we prefer to compare models by deletion rather than by tests on their parameter values. The simplified model (known as the **minimal adequate model**, because it only contains parameters that are significantly different from zero) looks like this:

```
summary(model3)
```

```
            Df Sum Sq Mean Sq F value  Pr(>F)
coat         1 2.6133  2.6133  4.7372 0.05457 .
Residuals   10 5.5167  0.5517
```

This is an interesting example of what can happen in model simplification. The effect of coat was highly significant in model 2 but just fails to reach significance when diet is left out of the model. It looks as if either both factors are important, or neither factor is unequivocally important. Only more detailed analysis will tell.

Let's look at the means for the three diets:

```
tapply(growth,diet,mean)
```

```
   A    B    C
7.70 7.95 8.80
```

It looks as if diets A and B produce a rather similar response, but perhaps diet C produces faster growth than the other 2 ? We calculate a new factor, diet2, which is 1 for diets A and B and 2 for diet C:

```
diet2<-factor(1+(diet=="C"))
diet2
```

```
 [1] 1 1 1 1 2 2 1 1 1 1 2 2
Levels:  1 2
```

Make sure you understand what we've done here. All of the diets are represented by a number 1, except for diet C which is represented by a 2 (1+0 = 1, 1+1 = 2 only when it is TRUE that diet equals "C"). Now we try adding this revised dietary factor to the minimal model:

```
model4<-update(model3, ~. +diet2)
```

and see whether it makes a significant difference using **anova**:

```
anova(model3,model4)
```

```
Analysis of Variance Table

Model 1: growth ~ coat
Model 2: growth ~ coat + diet2
  Res.Df Res.Sum Sq Df Sum Sq F value  Pr(>F)
1     10     5.5167
2      9     2.9817  1 2.5350  7.6518 0.02189 *
```

Yes, it does. Saving that extra degree of freedom, by reducing the levels of diet from 3 to 2 has turned a non-significant effect into a significant one. Let's see if there is an interaction with this new, simpler factor:

```
model5<-update(model4, ~. +diet2:coat)
anova(model4,model5)
```

```
Analysis of Variance Table

Model 1: growth ~ coat + diet2
Model 2: growth ~ coat + diet2 + coat:diet2

  Res.Df Res.Sum Sq Df  Sum Sq F value Pr(>F)
1      9    2.98167
2      8    2.70000  1 0.28167  0.8346 0.3877
```

No, there isn't. This means that model4 is the minimal adequate model.

summary(model4)

```
             Df Sum Sq Mean Sq F value  Pr(>F)
coat          1 2.6133  2.6133  7.8882 0.02042 *
diet2         1 2.5350  2.5350  7.6518 0.02189 *
Residuals     9 2.9817  0.3313
```

This example shows the benefits of model simplification. Our interpretation is now much simpler and much clearer than before. In brief, there is a significant effect of coat colour phenotype on mean growth, and there is a significant difference between diet C and the other two diets in mean growth rate. There is no evidence, however, that different coat colours respond to diet C in different ways (i.e. there is no interaction effect). Diagnostic plots on model4 show that the assumption of the anova in regard to normality of errors seems reasonable, but there is a hint of variance declining with the fitted values. Given the low replication, this is not serious.

Three-way Factorial Analysis of Variance

The only new element in a 3-way analysis is the extra weight of calculation involved. The principles are exactly the same as in 2-way factorial ANOVA. We have *a* levels of factor A, *b* levels of factor B and *c* levels of factor C. There are *n* replicates of each treatment combination (unequal replication can be handled, but it makes the formulas unnecessarily complicated, so we consider only the equal replication case in the hand calculations). The main effect totals are denoted by A, B and C respectively. The main effect sums of squares are calculated as in 1-way ANOVA, subtracting CF (the correction factor) from the sums of squares of the treatments totals divided by the relevant replication:

$$CF = \frac{\left[\sum y\right]^2}{abcn}$$

$$SSA = \frac{\sum A^2}{bcn} - CF$$

$$SSB = \frac{\sum B^2}{acn} - CF$$

$$SSC = \frac{\sum C^2}{abn} - CF$$

The three 2-way interactions are calculated in exactly the same way as they were in 2-way ANOVA, based on 2-way interaction tables of subtotals (see above). Only the 3-way formula is given here: this is based on a new set of tables of subtotals: there is a separate AB table for every level of C. The sums of squares of each of the elements, T, of these tables are added together and divided by *n*, the number of numbers added together to obtain each of these subtotals. The 3-way interaction sum of squares is then calculated like this:

$$SSABC = \frac{\sum T^2}{n} - SSA - SSB - SSC - SSAB - SSAC - SSBC - CF$$

The term $\sum T^2 / n$ contains all three 2-way interaction sums of squares as well as all 3 main effects sums of squares, so we need to subtract these in order to obtain the 3-way interaction sum of squares that we are after. Now the error sum of squares is obtained by subtracting all the other component sums of squares from the total sum of squares in the usual way:

$$SSE = SST - SSA - SSB - SSC - SSAB - SAC - SSBC - SSABC$$

and the ANOVA table can be completed.

Source	SS	d.f.	MS	F
Factor A	SSA	a-1	$\frac{SSA}{(a-1)}$	$\frac{MSA}{s^2}$
Factor B	SSB	b-1	$\frac{SSB}{(b-1)}$	$\frac{MSB}{s^2}$
Factor C	SSC	c-1	$\frac{SSC}{(c-1)}$	$\frac{MSC}{s^2}$
Interaction A:B	SSAB	(a-1)(b-1)	$\frac{SSAB}{(a-1)(b-1)}$	$\frac{MSAB}{s^2}$
Interaction A:C	SSAC	(a-1)(c-1)	$\frac{SSAC}{(a-1)(c-1)}$	$\frac{MSAC}{s^2}$
Interaction B:C	SSBC	(b-1)(c-1)	$\frac{SSBC}{(b-1)(c-1)}$	$\frac{MSBC}{s^2}$
Interaction A:B:C	SSABC	(a-1)(b-1)(c-1)	$\frac{SSABC}{(a-1)(b-1)(c-1)}$	$\frac{MSABC}{s^2}$
Error	SSE	abc(n-1)	$s^2 = \frac{SSE}{abc(n-1)}$	
Total	SST	$abcn$-1		

Now, bracing ourselves, we start the calculations by hand. The response variable is the population growth rate of *Daphnia*. There are 3 categorical response variables: Water (with 2 levels from the rivers Tyne and Wear), Detergent (with 4 levels, imaginatively named BrandA, BrandB, BrandC and BrandD) and *Daphnia* clone (with 3 levels, Clone1, Clone2 and Clone3). Each treatment combination was replicated 3 times. So our factor level codings from the equations above are $a = 2$, $b = 4$, $c = 3$ and $n = 3$.

	Growth.rate	Water	Detergent	*Daphnia*
1	2.919086	Tyne	BrandA	Clone1
2	2.492904	Tyne	BrandA	Clone1
3	3.021804	Tyne	BrandA	Clone1
4	2.350874	Tyne	BrandA	Clone2
5	3.148174	Tyne	BrandA	Clone2
6	4.423853	Tyne	BrandA	Clone2
7	4.870959	Tyne	BrandA	Clone3
8	3.897731	Tyne	BrandA	Clone3
9	5.830882	Tyne	BrandA	Clone3
10	2.302717	Tyne	BrandB	Clone1
11	3.195970	Tyne	BrandB	Clone1
12	2.829021	Tyne	BrandB	Clone1
13	3.027500	Tyne	BrandB	Clone2
14	5.285108	Tyne	BrandB	Clone2
15	4.260955	Tyne	BrandB	Clone2
16	4.049528	Tyne	BrandB	Clone3
17	4.623400	Tyne	BrandB	Clone3
18	5.625845	Tyne	BrandB	Clone3
19	2.634182	Tyne	BrandC	Clone1
20	3.513746	Tyne	BrandC	Clone1
21	3.714657	Tyne	BrandC	Clone1
22	3.862106	Tyne	BrandC	Clone2
23	3.955181	Tyne	BrandC	Clone2
24	3.044308	Tyne	BrandC	Clone2
25	3.358051	Tyne	BrandC	Clone3
26	5.633479	Tyne	BrandC	Clone3
27	4.613175	Tyne	BrandC	Clone3
28	2.200593	Tyne	BrandD	Clone1
29	2.233800	Tyne	BrandD	Clone1
30	3.357184	Tyne	BrandD	Clone1
31	3.111764	Tyne	BrandD	Clone2
32	4.842598	Tyne	BrandD	Clone2
33	4.362592	Tyne	BrandD	Clone2
34	2.230210	Tyne	BrandD	Clone3
35	3.104323	Tyne	BrandD	Clone3
36	4.762764	Tyne	BrandD	Clone3

	Growth.rate	Water	Detergent	*Daphnia*
37	2.319406	Wear	BrandA	Clone1
38	2.098191	Wear	BrandA	Clone1
39	3.541969	Wear	BrandA	Clone1
40	3.888784	Wear	BrandA	Clone2
41	3.960038	Wear	BrandA	Clone2
42	5.742290	Wear	BrandA	Clone2
43	5.244230	Wear	BrandA	Clone3
44	4.795048	Wear	BrandA	Clone3
45	5.380755	Wear	BrandA	Clone3
46	3.610532	Wear	BrandB	Clone1
47	2.247651	Wear	BrandB	Clone1
48	3.388947	Wear	BrandB	Clone1
49	4.061630	Wear	BrandB	Clone2
50	5.478983	Wear	BrandB	Clone2
51	4.303407	Wear	BrandB	Clone2
52	3.973060	Wear	BrandB	Clone3
53	4.632224	Wear	BrandB	Clone3
54	5.284316	Wear	BrandB	Clone3
55	2.480984	Wear	BrandC	Clone1
56	3.950710	Wear	BrandC	Clone1
57	2.133731	Wear	BrandC	Clone1
58	5.586468	Wear	BrandC	Clone2
59	6.918344	Wear	BrandC	Clone2
60	5.270421	Wear	BrandC	Clone2
61	2.015707	Wear	BrandC	Clone3
62	3.467042	Wear	BrandC	Clone3
63	5.028927	Wear	BrandC	Clone3
64	2.307174	Wear	BrandD	Clone1
65	2.962274	Wear	BrandD	Clone1
66	2.699762	Wear	BrandD	Clone1
67	6.569412	Wear	BrandD	Clone2
68	6.405130	Wear	BrandD	Clone2
69	5.990973	Wear	BrandD	Clone2
70	2.553241	Wear	BrandD	Clone3
71	2.592766	Wear	BrandD	Clone3
72	1.761603	Wear	BrandD	Clone3

We begin by calculating SST because it is easy and this will make us feel better.

$$CF = \frac{277.3372^2}{2 \times 4 \times 3 \times 3} = \frac{76915.9}{72} = 1068.276$$

$$SST = \sum y^2 - CF = 1185.434 - 1068.276 = 117.1572$$

Now we can work out the 3 main effect sums of squares, using the equations above:

$$SSA = \frac{132.691^2 + 144.6461^2}{4 \times 3 \times 3} - CF = 1.98506$$

$$SSB = \frac{69.927^2 + 72.1808^2 + 71.1812^2 + 64.0482^2}{2 \times 3 \times 3} - CF = 2.21157$$

$$SSC = \frac{68.157^2 + 109.8509^2 + 99.3293^2}{2 \times 4 \times 3} - CF = 39.1777$$

Calculation of the three 2-way interactions requires that we draw up **3 tables of subtotals**: an AB table summing over *Daphnia* Clones and replicates, an AC table summing over Detergents and replicates, and a BC table summing over Water and replicates. The sub totals in each of the 3 tables are squared, added up and divided by the number of numbers that were added together to get each sub total (3x3 = 9, 4x3 = 12 and 2x3 = 6 respectively). Here are the three tables, and the sums of their squares:

	BrandA	BrandB	BrandC	BrandD
Tyne	32.95627	35.20004	34.32889	30.20583
Wear	36.97071	36.98075	36.85233	33.84233

$$\sum T^2 = 9653.83$$

	Clone1	Clone2	Clone3
Tyne	34.41566	45.67501	52.60035
Wear	33.74133	64.17588	46.72892

$$\sum T^2 = 13478.05$$

	Clone1	Clone2	Clone3
BrandA	16.39336	23.51401	30.01961
BrandB	17.57484	26.41758	28.18837
BrandC	18.42801	28.63683	24.11638
BrandD	15.76078	31.28247	17.00491

$$\sum T^2 = 6781.597$$

The 2-way interaction sums of squares now look like this:

$$SSAB = \frac{9653.83}{3 \times 3} - SSA - SSB - CF = 0.17486$$

$$SSAC = \frac{13478.05}{4 \times 3} - SSA - SSC - CF = 13.732$$

$$SSBC = \frac{6781.597}{2 \times 3} - SSB - SSC - CF = 20.6006$$

Calculation of the 3-way interaction sum of squares involves working out all of the individual subtotals for the 2x4x3 = 24 treatment combinations, each of which is the sum of $n = 3$ replicates.

```
8.433794  7.959566  8.327708  9.247130  9.862586
8.565425  7.791576  7.969209  9.922901  13.591112
12.573563 13.844020 10.861595 17.775234 12.316954
18.965514 14.599572 15.420034 14.298773 13.889600
13.604706 10.511676 10.097297 6.907610
```

Now we need to square each of these, add them up and divide by 3

$$SSABC = \frac{34.56.017}{3} - SSA - SSB - SSC - SSAB - SSAC - SSBC - CF = 5.8476$$

The error sum of squares is the last thing we need to calculate:

$$SSE = 117.152 - 1.98506 - 2.21157 - 39.1777 - 0.17486 - 13.732 - 20.6006 - 5.8476 = 33.42776$$

Here is the complete ANOVA table.

Source	SS	d.f.	MS	F
Water	1.985	1	1.985	2.852
Detergent	2.116	3	0.705	1.013
Daphnia	39.178	2	19.589	28.145
Water:Detergent	0.1749	3	0.058	0.083
Water:*Daphnia*	13.732	2	6.866	9.865
Detergent:*Daphnia*	20.6006	6	3.433	4.932
3-way Interaction	5.8476	6	0.975	1.401
Error	33.4278	48	$s^2 = 0.696$	
Total	117.1572	71		

The 3-way interaction is not significant, nor is the Water:Detergent interaction, but both other 2-way interactions Water:*Daphnia* and Detergent :*Daphnia* are highly significant. All three factors therefore appear in at least one significant interactions, so model simplification ends here. The apparently insignificant main effects of Water and Detergent should *not* be interpreted as meaning that these factors have no significant effect on population growth rate. Always check the highest order

interactions first, and stop model simplification as soon as all of the terms have appeared in at least one significant interaction.

We can compare our answers with those produced by the computer.

```
Daphnia.data<-read.table("c:\\temp\\Daphnia.txt",header=T)
attach(Daphnia.data)
names(Daphnia.data)
```

```
[1] "Growth.rate"   "Water"        "Detergent"   "Daphnia"
```

The model formula uses the * notation to specify the fit of all interaction terms. In this case, the model calls for the estimation of 23 parameters, using up 6 degrees of freedom for the 3-way interaction, 11 for the three 2-way interactions and 6 for the 3 main effects.

```
factorial<-aov(Growth.rate~Water*Detergent*Daphnia)
```

```
summary(factorial)
```

```
                          Df     Sum of Sq   Mean Sq    F Value    Pr(F)
                Water     1      1.98506      1.98506  2.85042   0.0978380
             Detergent    3      2.21157      0.73719  1.05855   0.3754783
              Daphnia     2      39.17773    19.58887 28.12828   0.0000000
        Water:Detergent   3      0.17486      0.05829  0.08370   0.9686075
          Water:Daphnia   2      13.73204     6.86602  9.85914   0.0002587
      Detergent:Daphnia   6      20.60056     3.43343  4.93017   0.0005323
Water:Detergent:Daphnia   6      5.84762      0.97460  1.39946   0.2343235
             Residuals    48     33.42776     0.69641
```

As always, the interpretation begins with the highest order interactions. The present data provide no support for the hypothesis that the interaction between Water and *Daphnia* depends upon the kind of Detergent (p=0.23). There is no significant interaction between Water and Detergent, but highly significant two way interactions between Water and *Daphnia* and Detergent and *Daphnia*. All 3 factors appear in one or more significant interactions, so all factors are important. The non-significant main effect for Detergent should ***not***, therefore, be interpreted as meaning that Detergent has no significant effect on *Daphnia* growth rate. The correct interpretation is that the effect of Detergent on *Daphnia* growth rate varies from one *Daphnia* clone to another. Model criticism is carried out using **plot**(factorial).

In order to see how the interaction works, we can use **tapply** to calculate a 2-dimensional table of mean growth rates for each combination of Detergent and *Daphnia* clone (note the use of **list** to get the 2-dimensions we want):

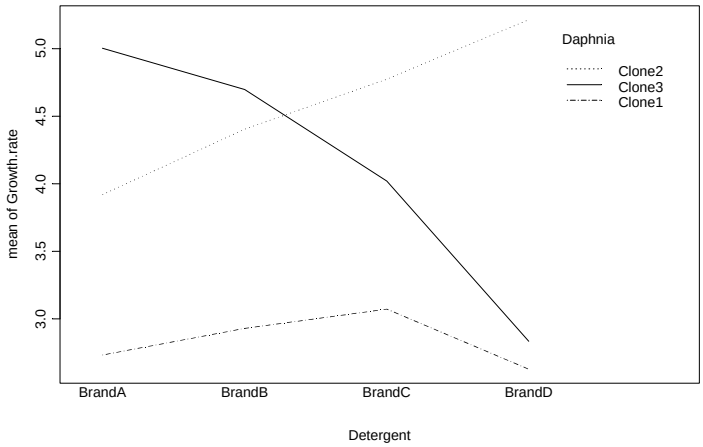
```
tapply(Growth.rate,list(Detergent,Daphnia),mean)
```

	Clone1	Clone2	Clone3
BrandA	2.732227	3.919002	5.003268
BrandB	2.929140	4.402931	4.698062
BrandC	3.071335	4.772805	4.019397
BrandD	2.626797	5.213745	2.834151

The interaction between Detergent and Clone is interpreted as follows. Consider Brand A. Mean growth rate rises from Clone 1 to Clone 2 To Clone 3. Brands B and C show a different pattern, with growth rate not increasing from Clone 2 to Clone 3. Brand D is different again, with growth rate declining steeply from Clone 2 to Clone 3. In general, it is a good idea to produce plots to show these interaction effects.

This is achieved by using the **interaction.plot** directive, like this (note that the response variable is **last** in the list, and that the factor to make the x axis is first in the list):

```
interaction.plot(Detergent,Daphnia,Growth.rate)
```



In general, interactions show up as non-parallelness in the lines of the interaction plot.

- How would you simplify this model ?
- What is the minimal adequate model for these data

STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

6. ANALYSIS OF COVARIANCE

Analysis of covariance combines elements from regression and analysis of variance. There is at least one continuous explanatory variable and at least one categorical explanatory variable. The procedure works like this:

- fit two or more linear regressions of *y* against *x* (one for each level of the factor)
- estimate different slopes and intercepts for each level
- use model simplification (deletion tests) to eliminate unnecessary parameters

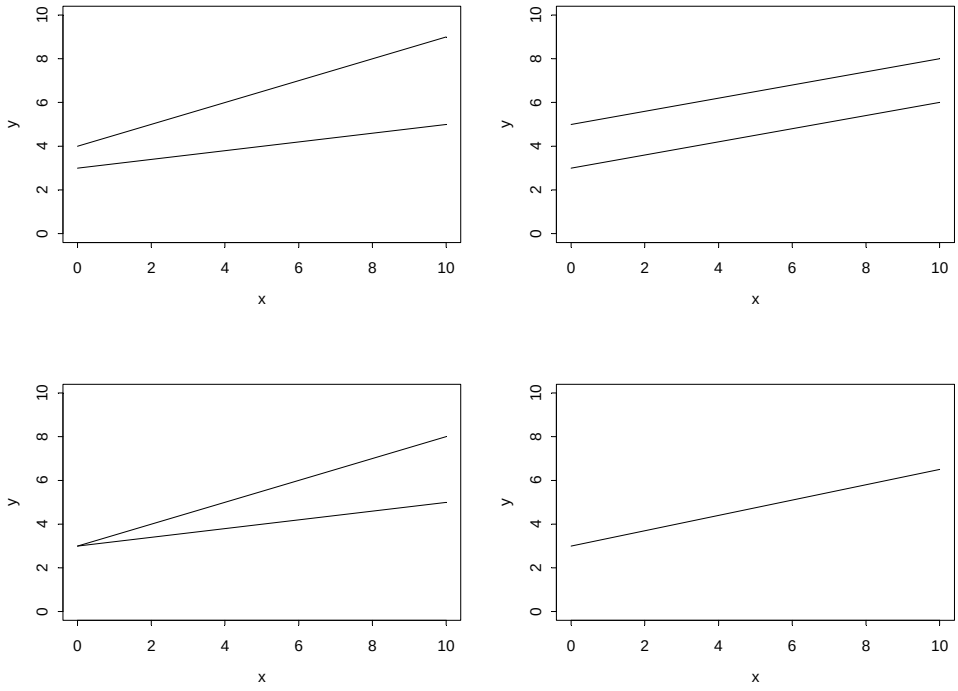
For example, we could use Ancova in a medical experiment where the response variable was 'days to recovery' and the explanatory variables were 'smoker or not' (categorical) and 'blood cell count' (continuous). In economics, local unemployment rate might be modelled as a function of country (categorical) and local population size (continuous).

Suppose we are modelling weight (the response variable) as a function of sex and age. Sex is a factor with 2 levels (male and female) and age is a continuous variable. The maximal model therefore has 4 parameters: two slopes (a slope for males and a slope for females) and two intercepts (one for males and one for females) like this:

$$weight_{male} = a_{male} + b_{male} \times age$$

$$weight_{female} = a_{female} + b_{female} \times age$$

This maximal model is shown in the top left panel:



Model simplification is an essential part of analysis of covariance, because the principle of parsimony requires that we keep as few parameters in the model as possible. In this case, the process of model simplification begins by asking whether we need all 4 parameters. Perhaps we could make do with 2 intercepts and a common slope (top right). Or a common intercept and two different slopes (bottom left). Alternatively, there may be no effect of sex at all, in which case we only need 2 parameters (one slope and one intercept) to describe the effect of age on weight (bottom right). There again, the continuous variable may have no significant effect on the response, so we only need 2 parameters to describe the main effects of sex on weight. This would show up as two separated, horizontal lines in the plot (one mean weight for each sex). In the limit, neither the continuous nor the categorical explanatory variables might have any significant effect on the response, in which case, model simplification will lead to the 1-parameter null model $\hat{y} = \overline{y}$ (a single, horizontal line).

Calculations involved in Ancova

We start with the simple example that we analysed as an Anova on p. 122, in which 6 different plant genotypes were exposed to 4 different light regimes. When we analysed the data as a 1-way Anova we found no significant effect of photoperiod, because the variation in growth rate between the different genotypes was so great. When we took differences between the genotype means into account by doing a 2-way Anova, we found a highly significant effect of photoperiod. Here we change the explanatory variable from

a categorical variable (daylength with 4 levels) into a continuous variable (the number of hours of illumination). There are 2 benefits in doing this

- is saves us 2 degrees of freedom in describing the effect of photoperiod (1 slope and 1 intercept instead of 4 photoperiod means)
- it allows us to test for an interaction between genotype and photoperiod on growth response (recall that we couldn't do this using Anova because there was no replication, so the interaction term had to be used as the error term)
- other things being equal, a regression approach will generally be more powerful than an Anova-based approach, because Anova takes no account of the *ordering* of the factor levels, only of the differences between their means.

The calculations are as follows. We need to do 6 separate regressions of growth against photoperiod (one for each genotype), keeping account of the corrected sums of products, SSXY, for each one separately. Here are the data again, this time showing the number of hours of illumination as the continuous explanatory variable.

Genotype	8hr	12hr	16hr	24hr
A	2	3	3	4
B	3	4	5	6
C	1	2	1	2
D	1	1	2	2
E	2	2	2	2
F	1	1	2	3

Here is the Anova table that we calculated in Chapter 15:

Source	SS	d.f.	MS	F
Photoperiod	7.125	3	2.375	7.703
Genotype	27.875	5	5.575	18.08
Error	4.625	15	$s^2 = 0.308$	
Total	39.625	23		

The Genotype and Total sums of squares will remain as they were. What we plan to do is to reduce the Error sum of squares and increase the explained Photoperiod sum of squares by taking account of the fact that the different daylengths can be ordered.

We shall do an overall regression to begin with, just estimating a single overall slope for the relationship between growth and photoperiod. The Anova table will therefore look like this:

Source	SS	d.f.	MS	F
Photoperiod		1		
Genotype	27.875	5		
Error		17	s^2	
Total	39.625	23		

We have reduced the degrees of freedom for photoperiod from 3 to 1 (a useful model simplification in itself) and increased the error degrees of freedom accordingly (this is good for reducing the error variance). So we start by calculating an overall regression sum of squares using all 24 data points.

First we need the famous five $\sum x$, $\sum x^2$, $\sum y$, $\sum y^2$, $\sum xy$

```
photoperiod<-read.table("c:\\temp\\photoperiod.txt",header=T)
attach(photoperiod)
names(photoperiod)
```

```
[1] "Genotype"    "Growth"      "Photoperiod"
```

So *x* is Photoperiod and *y* is Growth. We get the famous 5 like this:

```
sum(Photoperiod);sum(Photoperiod^2)
```

```
[1] 360
[1] 6240
```

```
sum(Growth);sum(Growth^2)
```

```
[1] 57
[1] 175
```

```
sum(Growth*Photoperiod)
```

```
[1] 932
```

So the corrected sums of squares (see Chapter 14) are calculated like this

$$SST = 175 - \frac{57^2}{24} = 39.625$$

$$SSXY = 932 - \frac{57 \times 360}{24} = 77$$

$$SSX = 6240 - \frac{360^2}{24} = 840$$

Now we have everything we need to calculate the slope of the overall straight line relating growth to photoperiod (remember, $b = SSXY/SSX$)

$$b = \frac{77}{840} = 0.0917$$

The next step is to work out the sums of squares for the Anova table. First the explained sum of squares (the regression sum of squares, SSR):

$$SSR = b \times SSXY = 0.0917 \times 77 = 7.0583$$

We know SST = 39.625 already from our Anova calculations (and see above), so we obtain SSE by subtracting SSR from SST.

$$SSE = SST - SSR = 39.625 - 7.0583 = 32.567$$

The Anova table can now be completed:

Source	SS	d.f.	MS	F
Photoperiod	7.0583	1	7.0583	25.575
Genotype	27.875	5	5.575	20.199
Error	4.4917	17	$s^2 = 0.276$	
Total	39.625	23		

This is already a considerable improvement over the 2-way Anova: the F ratio for the effect of photoperiod has gone up from 7.703 to 25.575, and the error variance has gone down from 0.308 to 0.276 (see Chapter 15). But we can do better than this. We can ask whether there is any evidence that *different genotypes showed different responses* to photoperiod. This is an interaction term, which involves fitting 6 different slopes to the model instead of 1 common slope (as we just did here).

The calculations are not hard, just tedious. We need to find SSR separately for each of the 6 genotypes. For this, we need 6 values of *b* and 6 values of SSXY.

First, we need the sums and sums of squares

tapply(Growth,Genotype,sum)

```
 A  B  C  D  E  F
12 18  6  6  8  7
```

tapply(Growth^2,Genotype,sum)

```
 A  B  C  D  E  F
38 86 10 10 16 15
```

tapply(Photoperiod,Genotype,sum)

```
 A  B  C  D  E  F
60 60 60 60 60 60
```

tapply(Photoperiod^2,Genotype,sum)

```
  A    B    C    D    E    F
1040 1040 1040 1040 1040 1040
```

Finally, we calculate the sums of products

tapply(Photoperiod*Growth,Genotype,sum)

```
  A   B   C   D   E   F
196 296  96 100 120 124
```

All the SSX's are the same:

$$SSX = 1040 - \frac{60^2}{4} = 140$$

Now we calculate the 6 separate SSXY's

$$SSXY_1 = 196 - \frac{12 \times 60}{4} = 16.0$$

$$SSXY_2 = 296 - \frac{18 \times 60}{4} = 26.0$$

$$SSXY_3 = 96 - \frac{6 \times 60}{4} = 6.0$$

$$SSXY_4 = 100 - \frac{6 \times 60}{4} = 10.0$$

$$SSXY_5 = 120 - \frac{8 \times 60}{4} = 0.0$$

$$SSXY_6 = 124 - \frac{7 \times 60}{4} = 19.0$$

So the 6 slopes are 16/140 = 0.11429, 26/140 = 0.1857, 6/140 = 0.04286, 0 and 19/140 = 0.1357.

Finally we can work out the 6 SSR's

Clone	SSXY	SSX	b	SSR
A	$196 - \frac{60 \times 12}{4} = 16$	$1040 - \frac{60^2}{4} = 140$	0.114	1.829
B	$296 - \frac{60 \times 18}{4} = 26$	$1040 - \frac{60^2}{4} = 140$	0.186	4.829
C	$96 - \frac{60 \times 6}{4} = 6$	$1040 - \frac{60^2}{4} = 140$	0.043	0.257
D	$100 - \frac{60 \times 6}{4} = 10$	$1040 - \frac{60^2}{4} = 140$	0.071	0.714
E	$120 - \frac{60 \times 8}{4} = 0$	$1040 - \frac{60^2}{4} = 140$	0	0
F	$124 - \frac{60 \times 7}{4} = 19$	$1040 - \frac{60^2}{4} = 140$	0.136	2.579
			Total	10.208

The 6 component regression sums of squares add up to 10.208. Note that there is substantial variation between the slopes for the different genotypes (*b* ranges from 0 to 0.186).

The next step is to work out the sum of squares attributable to differences between the slopes. The logic is simple. A single regression slope explained an SSR of 7.058 (above). Different slopes, as we have just seen, explained an SSR of 10.208. So the difference, attributable to having 6 slopes instead of 1, is 10.208 – 7.058 = 3.149. Because we have estimated 6 slopes now, where we had 1 before, this sum of squares is based on 6 – 1 = 5 d.f..

Now we can draw all the information together into a single Anova table:

Source	SS	d.f.	MS	F
Genotype	27.87	5	5.574	43.21
Regression	7.059	1	7.059	54.72
Differences in slope	3.149	5	0.63	4.88
Error	1.55	12	$s^2 = 0.129$	
Total	39.63	23		

The Ancova has produced a much better model than either the simple regression or the 1-way Anova, and a substantially better model than the 2-way Anova. Generally, if the nature of your data allows it, it is a good idea to use analysis of covariance whenever you can.

Analysis of Covariance in R

We could use either **lm** or **aov**; the choice affects only the format of the summary table. We shall use both and compare their output. The model with 6 different slopes is specified using the asterisk operator

```
Growth ~ Genotype * Photoperiod
```

Note than in S-Plus the output of summary.lm would be different because it uses Helmert contrasts, rather than the Treatment contrasts used by R. To get the same output in S-Plus you would need to type the following (**but this is unnecessary in R**):

```
options(contrasts=c("contr.treatment","contr.poly"))
```

We call the output *model* and work like this, starting with **lm**

```
model<-lm(Growth~Genotype*Photoperiod)

summary(model)
```

```
Coefficients:
                     Estimate Std. Error t value  Pr(>|t|)
(Intercept)           1.28571    0.48865   2.631  0.02193 *
GenotypeB             0.42857    0.69105   0.620  0.54674
GenotypeC            -0.42857    0.69105  -0.620  0.54674
GenotypeD            -0.85714    0.69105  -1.240  0.23855
GenotypeE             0.71429    0.69105   1.034  0.32169
GenotypeF            -1.57143    0.69105  -2.274  0.04213 *
Photoperiod           0.11429    0.03030   3.771  0.00267**
GenotypeB.Photoperiod 0.07143    0.04286   1.667  0.12145
GenotypeC.Photoperiod -0.07143    0.04286  -1.667  0.12145
GenotypeD.Photoperiod -0.04286    0.04286  -1.000  0.33705
GenotypeE.Photoperiod -0.11429    0.04286  -2.667  0.02054 *
GenotypeF.Photoperiod  0.02143    0.04286   0.500  0.62612
```

The output from Ancova takes a lot of getting used to. The most important thing to remember is that there are 12 rows in the summary table because the model has estimated 12 parameters from the data: 6 intercepts and 6 slopes.

With Treatment Contrasts (as here), the rows of the table are interpreted as follows:

- the Intercept in row 1 is the *intercept* for Genotype A
- the next 5 rows are *differences between intercepts*; for instance, row 2 (labelled Genotype B) is the difference in intercept between Genotypes B and A
- the *slope* is in row 6 (labelled Photoperiod); this is the slope of the graph of growth against photoperiod for Genotype A
- the last 5 rows are *differences between slopes*; for instance, row 7 (labelled GenotypeB : Photoperiod) is the difference between the slopes of the graphs for Genotypes B and A

So, for example, the parameterised equation for genotype D is as follows. The intercept is 1.28571 – 0.85714 = 0.42857, and the slope is 0.11429 – 0.04286 = 0.07143. The rightmost column indicates the parameter values that are significantly different from zero (when compared with Genotype A). The table shows that there is 1 significant interaction term (Genotype E has a significantly different response to Photoperiod compared with the other genotypes: in fact, it didn't respond at all to increasing light exposure. It also shows that Genotype F has a significantly lower intercept than has Genotype A.

This output generated by **lm** is good when you want to look at the individual parameter values. Let's see what happens when we fit the same model using **aov**.

```
model<-aov(Growth~Genotype*Photoperiod)

summary(model)
```

```
                     Df  Sum Sq Mean Sq F value     Pr(>F)
Genotype              5 27.8750  5.5750 43.3611 2.848e-007 ***
Photoperiod           1  7.0583  7.0583 54.8981 8.171e-006 ***
Genotype:Photoperiod  5  3.1488  0.6298  4.8981     0.0113 *
Residuals            12  1.5429  0.1286
```

Here we get a tidy Anova table to compare with the one we calculated earlier by hand, but there is no information on *which* parameters contribute to the significant differences. In practice, there is no need to fit two separate models as we did here using **lm** and **aov**. You only need to use one of them, then use **summary.aov** to produce an Anova table or **summary.lm** to produce a list of parameter estimates and standard errors. Try this:

```
summary.aov(model)
summary.lm(model)
```

Ancova with different values of the covariates

The calculations are more complicated if the different factor levels are associated with different values of the covariate. In the last example, all the genotypes were exposed to exactly the same photoperiods, which made the calculations much more straightforward. The next worked example concerns an experiment on the impact of grazing on the seed production of a biennial plant. Forty plants were allocated to two treatments, grazed and ungrazed, and the grazed plants were exposed to rabbits during the first two weeks of stem elongation. They were then protected from subsequent grazing by the erection of a fence and allowed to regrow. Because initial plant size was thought likely to influence fruit production, the diameter of the top of the rootstock was measured before each plant was potted up. At the end of the growing season, the fruit production (dry wt, mg) was recorded on each of the 40 plants, and this forms the response variable in the following analysis.

Ungrazed plants:

Fruit	59.77	60.98	14.73	19.28	34.25	35.53	87.73	63.21	24.25
Roots	6.225	6.487	4.919	5.130	5.417	5.359	7.614	6.352	4.975

Fruit	64.34	52.92	32.35	53.61	54.86	64.81	73.24	80.64	18.89
Roots	6.930	6.248	5.451	6.013	5.928	6.264	7.181	7.001	4.426

Fruit	75.49	46.73
Roots	7.302	5.836

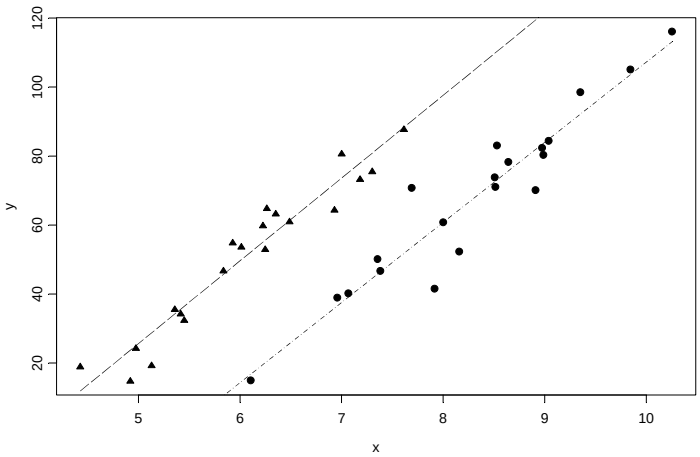
Grazed plants:

Fruit	80.31	82.35	105.1	73.79	50.08	78.28	41.48	98.47	40.15
Roots	8.988	8.975	9.844	8.508	7.354	8.643	7.916	9.351	7.066

Fruit	116.1	38.94	60.77	84.37	70.11	14.95	70.70	71.01	83.03
Roots	10.25	6.958	8.001	9.039	8.910	6.106	7.691	8.515	8.530

Fruit	52.26	46.64
Roots	8.158	7.382

The object of the exercise is to estimate the parameters of the minimal adequate model for these data: here is a plot of the final result so that you can see where we are heading: the triangles are ungrazed plants and the circles are grazed plants.



We start by working out the sums, sums of squares and sums of products for the whole data set combined (40 pairs of numbers), and then for each treatment separately (20 pairs of numbers).The data frame is called ipomopsis.txt

```
ipomopsis<-read.table("c:\\temp\\ipomopsis.txt",header=T)
attach(ipomopsis)
names(ipomopsis)
```

```
[1] "Root"    "Fruit"   "Grazing"
```

First, we'll work out the overall totals based on all 40 data points:

```
sum(Root);sum(Root^2)
```

```
[1] 287.246
[1] 2148.172
```

Start to fill in the table of totals (it helps to be really well organised for these calculations). Check to see where (and why) the sum (287.246) and the sum of squares (2148.172) of the root diameters (the *x* values) have gone in the table:

```
sum(Fruit);sum(Fruit^2)
```

```
[1] 2376.42
[1] 164928.1
```

```
sum(Root*Fruit)
```

```
[1] 18263.16
```

That completes the overall data summary. Now we select only the "Grazed" plant data:

```
sum(Root[Grazing=="Grazed"]);sum(Root[Grazing=="Grazed"]^2)
```

```
[1] 166.188
[1] 1400.834
```

Make sure you see where (and why) these totals go where they go. Now the “Ungrazed” subtotals

```
sum(Root[Grazing=="Ungrazed"]);sum(Root[Grazing=="Ungrazed"]^2)
```

```
[1] 121.058
[1] 747.3387
```

Now for the data on Fruits for the “Grazed” plants:

```
sum(Fruit[Grazing=="Grazed"]);sum(Fruit[Grazing=="Grazed"]^2)
```

```
[1] 1358.81
[1] 104156.0
```

And the Fruit data from the “Ungrazed” plants:

```
sum(Fruit[Grazing=="Ungrazed"]);sum(Fruit[Grazing=="Ungrazed"]^2)
```

```
[1] 1017.61
[1] 60772.11
```

Finally we want the data for the sums of products: first for the “Grazed” plants:

```
sum(Root[Grazing=="Grazed"]*Fruit[Grazing=="Grazed"])
```

```
[1] 11753.64
```

and for the “Ungrazed” plants:

```
sum(Root[Grazing=="Ungrazed"]*Fruit[Grazing=="Ungrazed"])
```

```
[1] 6509.522
```

	$\sum$ sums	$\sum^2$ products	$\sum\sum$ totals
x Ungrazed	121.058	747.3387	
y Ungrazed	1017.61	60772.11	
xy Ungrazed		6509.522	
$\sum x1 \sum y1$			123189.8
x Grazed	166.188	1400.834	
y Grazed	1358.81	104156.0	
xy Grazed		11753.64	
$\sum x2 \sum y2$			225817.9
x overall	287.246	2148.172	
y overall	2376.42	164928.1	
xy overall		18263.16	
$\sum x \sum y$			682617.14

Now we have all of the information necessary to carry out the calculations of the corrected sums of squares and products, SSY, SSX and SSXY for the whole data set (n = 40) and for the two separate treatments (20 reps in each).

To get the right answer you will need to be extremely methodical, but there is nothing mysterious or difficult about the process. First, calculate the regression statistics for the whole experiment, ignoring the grazing treatment.

The famous 5 are

sum(Root)	sum(Root^2)	sum(Fruit)	sum(Fruit^2)	sum(Root*Fruit)
287.246	2148.172	2376.42	164928.1	18263.16

so

$$SST = 164928.1 - \frac{2376.42^2}{40} = 23743.84$$

$$SSX = 2148.172 - \frac{287.246^2}{40} = 85.4158$$

$$SSXY = 18263.16 - \frac{287.246 \times 2376.42}{40} = 1197.731$$

$$SSR = \frac{(1197.731)^2}{85.4158} = 16795$$

$$SSE = 23743.84 - 16795 = 6948.835$$

The effect of differences between the two grazing treatments, SSA, is

$$SSA = \frac{1358.81^2 + 1017.61^2}{20} - \frac{2376.42^2}{40} = 2910.436$$

Next calculate the regression statistics for each of the grazing treatments separately. First, for the grazed plants:

$$SST_g = 104156 - \frac{1358.81^2}{20} = 11837.79$$

$$SSX_g = 1400.834 - \frac{166.188^2}{20} = 19.9111$$

$$SSXY_g = 11753.64 - \frac{1358.81 \times 166.188}{20} = 462.7415$$

$$SSR_g = \frac{(462.7415)^2}{19.9111} = 10754.29$$

$$SSE_g = 11837.79 - 10754.29 = 1083.509$$

so the slope of the graph of Fruit against Root for the grazed plants is given by

$$b_g = \frac{SSXY_g}{SSX_g} = \frac{462.7415}{19.9111} = 23.240$$

Now for the ungrazed plants:

$$SST_u = 60772.11 - \frac{1017.61^2}{20} = 8995.606$$

$$SSX_u = 747.3387 - \frac{121.058^2}{20} = 14.58677$$

$$SSXY_u = 6509.522 - \frac{121.058 \times 1017.61}{20} = 350.0302$$

$$SSR_u = \frac{(350.0302^2}{14.58677} = 8399.466$$

$$SSE_u = 8995.606 - 8399.466 = 596.1403$$

so the slope of the graph of Fruit against Root for the ungrazed plants is given by

$$b_u = \frac{SSXY_u}{SSX_u} = \frac{350.0302}{14.58677} = 23.996$$

Now add up the regression statistics across the factor levels (grazed and ungrazed):

$$SST_{g+u} = 11837.79 + 8995.606 = 20833.4$$

$$SSX_{g+u} = 19.9111 + 14.58677 = 34.49788$$

$$SSXY_{g+u} = 462.7415 + 350.0302 = 812.7717$$

$$SSR_{g+u} = 10754.29 + 8399.436 = 19153.75$$

$$SSE_{g+u} = 1083.509 + 596.1403 = 1684.461$$

The SSR for a model with a single common slope is given by

$$SSR = \frac{(SSXY_{g+u})^2}{SSX_{g+u}} = \frac{812.7717^2}{34.49788} = 19148.94$$

and the value of the single common slope is

$$b = \frac{SSXY_{g+u}}{SSX_{g+u}} = \frac{812.7717}{34.49788} = 23.560$$

The difference between the two estimates of SSR (SSR_{diff} = 19153.75 – 19148.94 = 4.81) is a measure of the significance of the difference between the two slopes estimated separately for each factor level. Finally, SSE is calculated by difference:

$$SSE = SST - SSA - SSR - SSR_{diff}$$

$$SSE = 23743.84 - 2910.44 - 19148.94 - 4.81 = 1679.65$$

Now we can complete the Anova table for the full model:

Source	SS	d.f.	MS	F
Grazing	2910.44	1		
Root	19148.94	1		
Different slopes	4.81	1	4.81	n.s.
Error	1679.65	36	46.66	
Total	23743.84	39		

Degrees of freedom for error are 40 – 4 = 36 because we have estimated 4 parameters from the data: 2 slopes and 2 intercepts. So the error variance is 46.66 (= SSE/36). The difference between the slopes is clearly not significant (F = 4.81/46.66 = 0.10) so we can fit a simpler model with a common slope of 23.56. The sum of squares for differences between the slopes (4.81) now becomes part of the error sum of squares:

Source	SS	d.f.	MS	F
Grazing	2910.44	1	2910.44	63.9291
Root	19148.94	1	19148.94	420.6156
Error	1684.46	37	45.526	
Total	23743.84	39		

This is the minimal adequate model. Both of the terms are highly significant and there are no redundant factor levels. The next step is to calculate the intercepts for the two parallel regression lines. This is done exactly as before, by rearranging the equation of the straight line to obtain $a = y - bx$. For each line we can use the mean values of x and y, with the common slope in each case. Thus:

$$a_1 = \overline{Y_1} - b\overline{X_1} = 50.88 - 23.56 \times 6.0529 = -91.7261$$
$$a_2 = \overline{Y_2} - b\overline{X_2} = 67.94 - 23.56 \times 8.309 = -127.8294$$

This demonstrates that the grazed plants produce, on average, 36.1 fruits *fewer* than the ungrazed plants (127.83 - 91.73). Finally, we need to calculate the standard errors for the common regression slope and for the difference in mean fecundity between the treatments, based on the error variance in the minimal adequate model:

$$s^2 = \frac{1684.46}{37} = 45.526$$

The standard errors are obtained as follows. The standard error of the common slope is found like this:

$$SE_b = \sqrt{\frac{s^2}{SSX}} = \sqrt{\frac{45.526}{19.9111 + 14.45667}} = 1.149$$

The standard error of the intercept of the regression for the grazed treatment (mean root size = 8.3094) is found as follows:

$$SE_a = \sqrt{s^2\left[\frac{1}{n} + \frac{(0-\bar{x})^2}{SSX}\right]} = \sqrt{45.526\left[\frac{1}{20} + \frac{8.3094^2}{34.498}\right]} = 9.664$$

It is clear that the intercept of –127.829 is very significantly less than zero (t = 127.829/9.664 = 13.2), suggesting that there is a threshold rootstock size before reproduction can begin. Finally, the standard error of the difference between the elevations of the two lines (the grazing effect) is given by

$$SE_{\hat{y}_1 - \hat{y}_2} = \sqrt{s^2\left[\frac{2}{n} + \frac{(\bar{x_1} - \bar{x_2})^2}{SSX}\right]}$$

which, substituting the values for the error variance and the mean rootstock sizes of the plants in the two treatments, becomes:

$$SE_{\hat{y}_1 - \hat{y}_2} = \sqrt{45.526\left[\frac{2}{20} + \frac{(6.0529 - 8.3094)^2}{34.498}\right]} = 3.357$$

This suggests that any lines differing in elevation by more than about 2 × 3.357 = 6.66 mg dry weight would be regarded as significantly different. Thus, the present difference of 36.09 clearly represents a highly significant reduction in fecundity caused by grazing (t = 10.83).

Ancova in R using lm

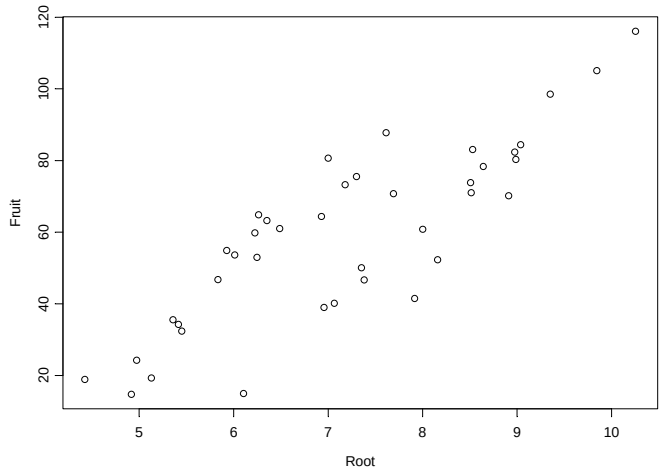
We now repeat the analysis using **lm**. The response variable is fecundity, and there is one experimental factor (grazing) with two levels (ungrazed and grazed) and one covariate (initial root stock diameter). There are 40 values for each of these variables. The odd thing about these data is that grazing seems to *increase* fruit production, a highly counter intuitive result:

```
tapply(Fruit,Grazing, mean)
```

```
 Grazed Ungrazed
 67.9405  50.8805
```

How could this have come about ? We begin by inspecting the data:

```
plot(Root,Fruit)
```



This demonstrates clearly that size matters: the plants that had larger rootstocks at the beginning produced more fruit when they matured. This plot does not help with the real question, however, which is “how did grazing affect fruit production” ? What we need to do is to plot the data separately for the grazed and ungrazed plants. First we plot blank axes:

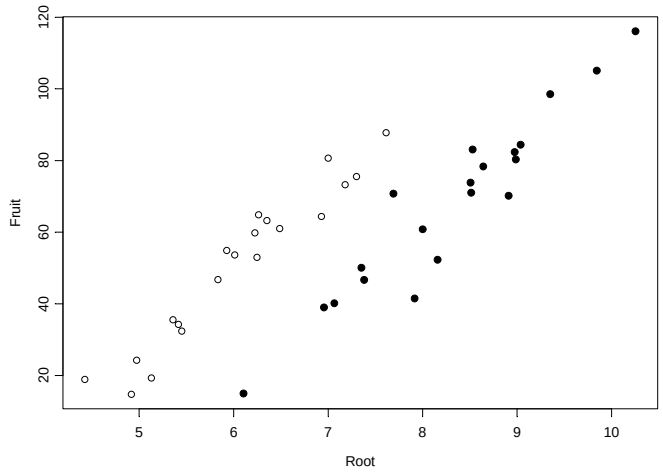
```
plot(Root,Fruit,type="n")
```

Next we select only those points that refer to Ungrazed plants, and add them using **points**

```
points(Root[Grazing=="Ungrazed"],Fruit[Grazing=="Ungrazed"])
```

Now, using a different plotting symbol (the filled circle, pch=16), we add the points that relate to the Grazed plants (use the Up Arrow to edit the last line):

```
points(Root[Grazing=="Grazed"],Fruit[Grazing=="Grazed"],pch=16)
```



This shows a striking pattern which suggests that the largest plants were allocated to the grazed treatments (the filled symbols). But the plot also indicates that for a *given* rootstock diameter (say 7 mm) the grazed plants produced *fewer* fruits than the ungrazed plants (not more, as a simple comparison of the means suggested). This is an excellent example of where analysis of covariance comes into its own. Here, the correct analysis using Ancova completely *reverses* our interpretation of the data.

The analysis proceeds in the following way. We fit the most complicated model first, then simplify it by removing non-significant terms until we are left with a minimal adequate model, in which all the parameters are significantly different from zero. For Ancova, the most complicated model has different slopes and intercepts for each level of the factor. Here we have a 2-level factor (Grazed and Ungrazed) and we are fitting a linear model with 2 parameters ($y = a + bx$) so the most complicated mode has 4 parameters (2 slopes and 2 intercepts). To fit different slopes and intercepts we use the asterisk * notation:

```
ancova<-lm(Fruit~Grazing*Root)
```

You should realise that *order matters*: we would get a different output if the model had been written `Fruit ~ Root * Grazing` (more of this later).

```
summary(ancova)
```

```
Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)         -125.173     12.811  -9.771 1.15e-11 ***
GrazingUngrazed       30.806     16.842   1.829   0.0757 .
Root                  23.240      1.531  15.182  < 2e-16 ***
GrazingUngrazed:Root   0.756      2.354   0.321   0.7500

Residual standard error: 6.831 on 36 degrees of freedom
Multiple R-Squared: 0.9293,     Adjusted R-squared: 0.9234
F-statistic: 157.6 on 3 and 36 DF,  p-value: < 2.2e-16
```

This shows that initial root size has a massive effect on fruit production (t = 15.182), but there is no indication of any difference in the slope of this relationship between the two grazing treatments (this is the Grazing by Root interaction with t = 0.321, $p >> 0.05$). The Anova table for the maximal model looks like this (the same as **summary.aov(ancova)**):

anova(ancova)

```
              Df Sum of Sq  Mean Sq  F Value     Pr(F)
     Grazing   1   2910.44  2910.44  62.3795 0.0000000
        Root   1  19148.94 19148.94 410.4201 0.0000000
Grazing:Root   1      4.81     4.81   0.1031 0.7499503
   Residuals  36   1679.65    46.66
```

The next step is to delete the non-significant interaction term from the model. We can do this manually or automatically: here we'll do both for the purposes of demonstration. The directive for manual model simplification is **update**. We update the current model (here called ancova) by deleting terms from it. The syntax is important: the punctuation reads "comma tilde dot minus" . We define a new name for the simplified model like ancova2

ancova2<-update(ancova**, ~ . -** Grazing:Root)

Now we compare the simplified model with just 3 parameters (1 slope and 2 intercepts) with the maximal model using **anova** like this:

anova(ancova,ancova2)

```
Analysis of Variance Table

Model 1: Fruit ~ Grazing * Root
Model 2: Fruit ~ Grazing + Root

  Res.Df     RSS Df Sum of Sq      F Pr(>F)
1     36 1679.65
2     37 1684.46 -1     -4.81 0.1031   0.75
```

This says that model simplification was justified because it caused a negligible reduction in the explanatory power of the model (p = 0.75; to retain the interaction term in the

model we would need $p < 0.05$). The next step in model simplification involves testing whether or not grazing had a significant effect on fruit production once we control for initial root size. The procedure is similar: we make a new model name, say ancova3, and use **update** to remove Grazing from ancova2 like this:

```
ancova3<-update(ancova2, ~ . - Grazing)
```

Now we compare the two models using **anova**:

```
anova(ancova2,ancova3)
```

```
Analysis of Variance Table

Model 1: Fruit ~ Grazing + Root
Model 2: Fruit ~ Root

  Res.Df Res.Sum Sq Df  Sum Sq F value     Pr(>F)
1     37     1684.5
2     38     6948.8 -1 -5264.4  115.63 6.107e-013 ***
```

This model simplification is a step too far. Removing the Grazing term causes a massive reduction in the explanatory power of the model, with an F value of 115.63 and a vanishingly small *p* value. The effect of grazing in reducing fruit production is highly significant and needs to be retained in the model. Thus ancova2 is our minimal adequate model, and we should look at its summary table to compare with our earlier calculations carried out by hand:

```
summary(ancova2)
```

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     -127.829      9.664  -13.23 1.35e-15 ***
GrazingUngrazed   36.103      3.357   10.75 6.11e-13 ***
Root              23.560      1.149   20.51  < 2e-16 ***

Residual standard error: 6.747 on 37 degrees of freedom
Multiple R-Squared: 0.9291,     Adjusted R-squared: 0.9252
F-statistic: 242.3 on 2 and 37 DF,  p-value: < 2.2e-16
```

You know when you have got the minimal adequate model, because every row of the coefficients table has one or more significance stars (there are 3 stars in this case, because the effects are all so strong). Unfortunately, the interpretation is not crystal clear from this table because *the variable name, not the level name* appears in the first column, and you might misread this as saying that "Grazing is associated with + 36.103 mg of Fruit production". This is wrong. It is *the second level of Grazing* (which is "Ungrazed" in the

present case) that is associated with this positive difference in intercepts. For a given root size, the Grazed plants (factor level 1) produce 36.103 mg of fruit *less* than the Ungrazed plants (factor level 2). R arranges the factor levels in alphabetical order unless you tell it to do otherwise.

anova(ancova2)

```
          Df Sum of Sq  Mean Sq  F Value         Pr(F)
  Grazing  1   2910.44  2910.44  63.9291 1.397196e-009
     Root  1  19148.94 19148.94 420.6156 0.000000e+000
Residuals 37   1684.46    45.53
```

These are the values we obtained longhand on p. 152. Now we repeat the model simplification using the automatic model-simplification directive called **step**. It couldn't be easier to use. The full model is called ancova:

step(ancova)

This directive causes all the terms to be tested to see whether they are needed in the minimal adequate model. The criterion used is AIC, the Akaike Information Criterion (more of which later). In the jargon, this is a "penalised log likelihood". What this means in simple terms is that it weighs up the inevitable trade off between degrees of freedom and fit of the model. You can have a perfect fit if you have a parameter for every data point, but this model has zero explanatory power. Thus ***deviance goes down as degrees of freedom in the model goes up***. The AIC adds 2 times the number of parameters in the model to the deviance (to penalise it). Deviance, you will recall, is twice the log likelihood of the current model.

Anyway, AIC is a measure of lack of fit; big AIC is bad, small AIC is good. The full model (4 parameters, 2 slopes and 2 intercepts) is fitted first, and AIC calculated as 157.5

```
Start:  AIC= 157.5
 Fruit ~ Grazing + Root + Grazing:Root
```

Then **step** tries removing the most complicated term (the Grazing by Root interaction)

```
               Df Sum of Sq     RSS     AIC
- Grazing:Root  1      4.81 1684.46  155.61
<none>                     1679.65  157.50

Step:  AIC= 155.61
 Fruit ~ Grazing + Root
```

This has reduced AIC to 155.61 (an improvement, so the simplification is justified)

```
          Df Sum of Sq     RSS     AIC
<none>                  1684.5   155.6
```

```
- Grazing  1    5264.4  6948.8   210.3
- Root     1   19148.9 20833.4   254.2

Call:
lm(formula = Fruit ~ Grazing + Root)

Coefficients:
(Intercept)      Grazing         Root
    -127.83        36.10        23.56
```

No further simplification is possible (as we saw when we used **update** to remove the Grazing term from the model) because AIC goes up to 210.3 when Grazing is removed and up to 254.2 if Root size is removed. Thus, **step** has found the minimal adequate model (it doesn't always, as we shall see later; it is "good but not perfect"). Again, with this form of output it is possible to misinterpret the result of Grazing; it would be more informative to have the intercepts labelled by the factor levels Grazed and Ungrazed, as when we used **tapply** at the beginning of the exercise.

Drawing the regression lines through the scatterplot following Ancova

You should already have the scatterplot on the graphics window. All you need to do now is to use abline to draw the two lines (one for the grazed plants and one for the ungrazed). The first argument to abline is the intercept (-127.829 for the grazed plants) and the second is the slope (23.56). The parameter values are in the model called ancova2:

```
summary(ancova2)
```

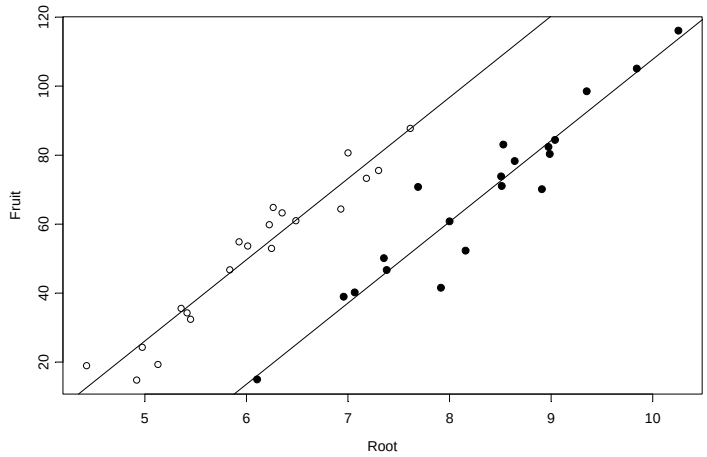
```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     -127.829      9.664  -13.23 1.35e-15 ***
GrazingUngrazed   36.103      3.357   10.75 6.11e-13 ***
Root              23.560      1.149   20.51  < 2e-16 ***
```

So, to get the first line, we write

```
abline(-127.829 , 23.56)
```

The second line is obtained simply by adding the difference between the intercepts (the ungrazed plants have 36.1032 mg more fruit than the grazed plants): use the Up arrow to edit the last command

```
abline(-127.829+36.1032 , 23.56)
```



Ancova and experimental design

There is an extremely important general message in this example for experimental design. No matter how carefully we randomise at the outset, our experimental groups are likely to be heterogeneous. Sometimes, as in this case, we may have made initial measurements that we can use as covariates later on, but this will not always be the case. There are bound to be important factors that we did not measure. If we had not measured initial root size in this example, we would have come to entirely the wrong conclusion about the impact of grazing on plant performance.

A far better design for this experiment would have been to measure the root stock diameters of all the plants at the beginning of the experiment (as was done here), but then to place the plants in matched pairs with similar sized rootstocks. Then, one of the plants is picked at random and allocated to one of the two grazing treatments (e.g. by tossing a coin); the other plant of the pair then receives the unallocated gazing treatment. Under this scheme, the size ranges of the two treatments would overlap, and the analysis of covariance would be unnecessary.

A more complex Ancova: 2 factors and 1 continuous covariate

This experiment with Weight as the response variable involved Genotype and Sex as two categorical explanatory variables and Age as a continuous covariate. There are 6 levels of Genotype and 2 levels of Sex.

```
gain<-read.table("c:\\temp\\gain.txt",header=T)
attach(gain)
names(gain)
```

```
[1] "Weight"   "Sex"      "Age"      "Genotype" "Score"
```

We begin by fitting the maximal model with its 24 parameters; different slopes and intercepts for every one of the 12 combinations of Sex (2) and Genotype (6).

```
m1<-lm(Weight~Sex*Age*Genotype)

summary(m1)

Coefficients:
                      Estimate Std. Error t value  Pr(>|t|)
(Intercept)            7.80053    0.24941  31.276  < 2e-016 ***
Sex                   -0.51966    0.35272  -1.473   0.14936
Age                    0.34950    0.07520   4.648 4.39e-005 ***
GenotypeCloneB         1.19870    0.35272   3.398   0.00167 **
GenotypeCloneC        -0.41751    0.35272  -1.184   0.24429
GenotypeCloneD         0.95600    0.35272   2.710   0.01023 *
GenotypeCloneE        -0.81604    0.35272  -2.314   0.02651 *
GenotypeCloneF         1.66851    0.35272   4.730 3.41e-005 ***
Sex.Age               -0.11283    0.10635  -1.061   0.29579
Sex.GenotypeCloneB    -0.31716    0.49882  -0.636   0.52891
Sex.GenotypeCloneC    -1.06234    0.49882  -2.130   0.04010 *
Sex.GenotypeCloneD    -0.73547    0.49882  -1.474   0.14906
Sex.GenotypeCloneE    -0.28533    0.49882  -0.572   0.57087
Sex.GenotypeCloneF    -0.19839    0.49882  -0.398   0.69319
Age.GenotypeCloneB    -0.10146    0.10635  -0.954   0.34643
Age.GenotypeCloneC    -0.20825    0.10635  -1.958   0.05799 .
Age.GenotypeCloneD    -0.01757    0.10635  -0.165   0.86970
Age.GenotypeCloneE    -0.03825    0.10635  -0.360   0.72123
Age.GenotypeCloneF    -0.05512    0.10635  -0.518   0.60743
Sex.Age.GenotypeCloneB  0.15469    0.15040   1.029   0.31055
Sex.Age.GenotypeCloneC  0.35322    0.15040   2.349   0.02446 *
Sex.Age.GenotypeCloneD  0.19227    0.15040   1.278   0.20929
Sex.Age.GenotypeCloneE  0.13203    0.15040   0.878   0.38585
Sex.Age.GenotypeCloneF  0.08709    0.15040   0.579   0.56616

Residual standard error: 0.2378 on 36 degrees of freedom
Multiple R-Squared: 0.9742,     Adjusted R-squared: 0.9577
F-statistic: 59.06 on 23 and 36 degrees of freedom,
p-value:     0
```

Model simplification

There are one or two significant parameters, but it is not at all clear that the 3-way or 2-way interactions need to be retained in the model. As a first pass, let's use **step** to see how far it gets with model simplification:

```
step(m1)

Start:  AIC= -155.01
 Weight ~ Sex + Age + Genotype + Sex:Age + Sex:Genotype +
Age:Genotype +  Sex:Age:Genotype
```

```
                    Df Sum of Sq      RSS      AIC
- Sex:Age:Genotype  5     0.349    2.385 -155.511
<none>                            2.036 -155.007
```

It definitely doesn't need the 3-way interaction, despite the effect of Sex.Age.GenotypeCloneC which gave a significant t-test on its own. How about the 3 2-way interactions ?

```
Step:  AIC= -155.51
 Weight ~ Sex + Age + Genotype + Sex:Age + Sex:Genotype +
Age:Genotype


                Df Sum of Sq      RSS      AIC
- Sex:Genotype  5     0.147    2.532 -161.924
- Age:Genotype  5     0.168    2.553 -161.423
- Sex:Age       1     0.049    2.434 -156.292
<none>                         2.385 -155.511
```

It has left out Sex by Genotype and now assesses the other two:

```
Step:  AIC= -161.92
 Weight ~ Sex + Age + Genotype + Sex:Age + Age:Genotype

                Df Sum of Sq      RSS      AIC
- Age:Genotype  5     0.168    2.700 -168.066
- Sex:Age       1     0.049    2.581 -162.776
<none>                         2.532 -161.924
```

No need for Age by Genotype. Try removing Sex by Age:

```
Step:  AIC= -168.07
 Weight ~ Sex + Age + Genotype + Sex:Age

           Df Sum of Sq      RSS      AIC
- Sex:Age   1     0.049    2.749 -168.989
<none>                    2.700 -168.066
- Genotype  5    54.958   57.658    5.612
```

Nothing. What about the main effects?

```
Step:  AIC= -168.99
 Weight ~ Sex + Age + Genotype

           Df Sum of Sq      RSS      AIC
<none>                    2.749 -168.989
```

```
- Sex       1    10.374   13.122  -77.201
- Age       1    10.770   13.519  -75.415
- Genotype  5    54.958   57.707    3.662
```

They are all highly significant. This is R's idea of the minimal adequate model. Three main effects but no interactions. That is to say, that the slope of the graph of weight gain against age does not vary with sex or genotype, but the intercepts *do* vary.

It would be a good idea to look at the Anova table for this model:

m2<-aov(Weight~Sex+Age+Genotype)

summary(m2)

```
Coefficients:

            Df Sum Sq Mean Sq F value     Pr(>F)
Sex          1 10.374  10.374  196.23 < 2.2e-016 ***
Age          1 10.770  10.770  203.73 < 2.2e-016 ***
Genotype     5 54.958  10.992  207.93 < 2.2e-016 ***
Residuals   52  2.749   0.053
```

That certainly looks pretty convincing. What does **lm** produce ?

summary.lm(m2)

```
Coefficients:

               Estimate Std. Error t value  Pr(>|t|)
(Intercept)     7.93701    0.10066  78.851  < 2e-016 ***
Sex            -0.83161    0.05937 -14.008  < 2e-016 ***
Age             0.29958    0.02099  14.273  < 2e-016 ***
GenotypeCloneB  0.96778    0.10282   9.412 8.07e-013 ***
GenotypeCloneC -1.04361    0.10282 -10.149 6.22e-014 ***
GenotypeCloneD  0.82396    0.10282   8.013 1.21e-010 ***
GenotypeCloneE -0.87540    0.10282  -8.514 1.98e-011 ***
GenotypeCloneF  1.53460    0.10282  14.925  < 2e-016 ***

Residual standard error: 0.2299 on 52 degrees of freedom
Multiple R-Squared: 0.9651,     Adjusted R-squared: 0.9604
F-statistic: 205.7 on 7 and 52 degrees of freedom,
p-value:     0
```

This is where Helmert contrasts would come in handy. Everything is 3-star significantly different from Genotype[1] Sex[1], but it is not obvious that the intercepts for Genotypes B and D need different values (+0.96 and +0.82 above Genotype A with s.e. difference = 0.1028), nor is it obvious that C and E have different intercepts (-1.043 and –0.875).

Perhaps we could reduce the number of factor levels of Genotype from the present 6 to 4 without any loss of explanatory power ?

Factor-level reduction

We create a new categorical variable called newgen with separate levels for clones A and F, and for B & D combined and C & E combined.

```
newgen<-factor(1+(Genotype=="CloneB")+(Genotype=="CloneD")+
   2*(Genotype=="CloneC")+2*(Genotype=="CloneE")+3*(Genotype=="CloneF"))
```

Then we re-do the modelling with newgen (4 levels) instead of Genotype (6 levels)

```
m3<-lm(Weight~Sex+Age+newgen)
```

and check that the simplification was justified

```
anova(m2,m3)
```

Remember:

In ancova,

The intercept belongs to the factor level that comes first in the alphabet

The slope belongs to the first factor level in the alphabet

Categorical effects are differences between intercepts

Interaction terms are differences between slopes

STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

7. NESTED ANALYSIS & SPLIT PLOT DESIGNS

Up to this point, we have treated all categorical explanatory variables as if they were the same. This is certainly what R.A. Fisher had in mind when he invented the analysis of variance in the 1920's and 30's. It was Eisenhart (1947) who realised that there were actually 2 fundamentally different sorts of categorical explanatory variables: somewhat confusingly he called these **fixed effects** and **random effects**. The distinction is best seen by an example. In most mammal species the categorical variable 'sex' has two levels: "male" and "female". For any individual that you find, the knowledge that it is, say, female conveys a great deal of information about the individual, and this information draws on experience gleaned from many other individuals that were female. A female will have a whole set of attributes (associated with her being female) no matter what population that individual was drawn from. Take a different categorical variable like genotype. If we have two genotypes in a population we might label them A and B. If we take two more genotypes from a *different* population we might label them A and B as well. In a case like this, the label A does not convey any information at all about the genotype, other than that it is different from Genotype B. In the case of sex, the factor level (male or female) is informative: sex is a fixed effect. In the case of genotype, the factor level (A or B) is uninformative: genotype is a random effect. Random effects have factor levels that are drawn from a large (potentially very large) population in which the individuals differ in many ways, but *we do not know exactly how or why they differ*. In the case of sex we know that males and females are likely to differ in characteristic and predictable ways. To get a feel for the difference between fixed effects and random effects here are some more examples:

Fixed Effects	Random effects
Drug administered or not	Genotype
Insecticide sprayed or not	Brood
Nutrient added or not	Block within a field
One country versus another	Split plot within a plot
Male or female	History of development
Wet versus dry	Untreated individuals
Light versus shade	Family
One age versus another	Parent

There are three main parts to this practical:

- Nested Designs
- Designed Split-Plot Experiments
- Mixed Effects Models

They are linked by two facts: (1) they involve categorical variables of two kinds (fixed effects and random effects); and (2) because their data frames all involve pseudoreplication, they offer great scope for getting the analysis wrong.

Model I and Model II anova

The distinction between the two models of anova was made by Eisenhart in 1947. In Model I, the differences between the means are ascribed entirely to the fixed treatment effects, so any data point can be decomposed like this:

$$y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$$

It is the overall mean μ plus a treatment effect due to the *i*th level of the fixed effect α_i plus a deviation ε_{ij} that is unique to that individual and is drawn from a Normal distribution with mean 0 and variance σ^2.

Model II is subtly different. Here, the model is written

$$y_{ij} = \mu + U_i + \varepsilon_{ij}$$

which looks as if it is *exactly* the same as Model I, just replacing α_i by U_i. The distinction is this. In Model I the experimenter either *made* the treatments like they were (e.g. by the random allocation of treatments to individuals), or selected the categories because they were fixed and clearly distinctive (e.g. a comparison of individual fossils from the Jurassic and Carboniferous periods). In Model II the factor levels are different from one another, but the experimenter did not *make* them different. They were selected (perhaps *because* they were different, perhaps not), but they came from a much larger pool of factor levels that exhibits variation beyond the control of the experimenter and beyond their immediate interest (e.g. 6 genotypes were selected to work with, out of a pool of, who knows, perhaps many 1000's of genotypes). We call it random variation, and call such factors random effects.

The important point is that because the random effects U_i come from a large population, there is not much point in concentrating on estimating means of our small subset, *a*, of factor levels, and no point at all in comparing individual pairs of means for different

factor levels. Much better to recognise them for what they are, random samples from a much larger population, and to concentrate on their variance σ_U^2. This is the *added* variation caused by differences between the levels of the random effects. Model II anova is all about estimating the size of this variance, and working out its percentage contribution to the overall variation.

The issues involved in Model II work fall into 2 broad categories

- questions about experimental design and the management of experimental error (e.g. where does most of the variation occur, and where would increased replication be most profitable?)

- questions about hierarchical structure, and the relative magnitude of variation at different levels within the hierarchy (e.g. studies on the genetics of individuals within families, families within parishes, and parishes with counties, to discover the relative importance of genetic and phenotypic variation).

Nested Analysis

Most anova models are based on the assumption that there is a single error term. But in nested experiments like split-plot designs, where the data are gathered at two or more different spatial scales, there is *a different error variance for each different plot size*.

We take an example from Sokal & Rohlf (1981). The experiment involved a simple one-factor anova with 3 treatments given to 6 rats. The analysis was complicated by the fact that three preparations were taken from the liver of each rat, and two readings of glycogen content were taken from each preparation. This generated 6 pseudoreplicates per rat to give a total of 36 readings in all. Clearly, it would be a mistake to analyse these data as if they were a straightforward one-way anova, because that would give us 33 degrees of freedom for error. In fact, since there are only two rats in each treatment, we have only one degree of freedom per treatment, giving a total of 3 d.f. for error.

The variance is likely to be different at each level of this nested analysis because:

- the readings differ because of variation in the glycogen detection method within each liver sample (measurement error)
- the pieces of liver may differ because of heterogeneity in the distribution of glycogen within the liver of a single rat
- the rats will differ from one another in their glycogen levels because of sex, age, size, genotype, etc.
- rats allocated different experimental treatments may differ as a result of the fixed effects of treatment.

If all we want to test is whether the experimental treatments have affected the glycogen levels, then we are not interested in liver bits within rat's livers, or in preparations within

liver bits. We could add all the pseudoreplicates together, and analyse the 6 averages. This would have the virtue of showing what a tiny experiment this really was (we do this later; see below). But to analyse the full data set, we must proceed as follows.

The only trick is to ensure that the factor levels are set up properly. There were 3 treatments, so we make a treatment factor *T* with 3 levels. While there were 6 rats in total, there were only 2 in each treatment, so we declare rats as a factor *R* with 2 levels (not 6). There were 18 bits of liver in all, but only 3 per rat, so we declare liver-bits as a factor *L* with 3 levels (not 18).

```
rats<-read.table("c:\\temp\\rats.txt",header=T)
attach(rats)
names(rats)
```

```
[1] "Glycogen"  "Treatment" "Rat"       "Liver"
```

```
tapply(Glycogen,Treatment,mean)
```

```
       1        2        3
140.5000 151.0000 135.1667
```

There are substantial differences between the treatment means, and our job is to say whether these differences are statistically significant or not. Because the 3 factors Treatment, Rat and Liver have numeric factor levels, we must declare them to be factors before beginning the modelling.

```
Treatment<-factor(Treatment)
Rat<-factor(Rat)
Liver<-factor(Liver)
```

Because the sums of squares in nested designs are so confusing on first acquaintance, it is important to work through a simple example like this one by hand. The first two steps are easy, because they relate to the **fixed effect**, which is **Treatment**. We calculate SST and SSA in the usual way (see Practical 5, ANOVA). We need the sum of squares and the grand total of the response variable Glycogen

```
sum(Glycogen);sum(Glycogen^2)
```

```
[1] 5120
[1] 731508
```

$$SST = \sum y^2 - \frac{\left[\sum y\right]^2}{36}$$

SST = 731508-5120^2/36 = 3330.222

For the treatment sum of squares, SSA, we need the 3 treatment totals

```
tapply(Glycogen,Treatment,sum)
```

```
  1    2    3
1686 1812 1622
```

Each of these was the sum of 12 numbers (2 preparations x 3 liver bits x 2 rats) so we divide the square of each subtotal by 12 before subtracting the correction factor

$$SSA = \frac{\sum T^2}{12} - \frac{\left[\sum y\right]^2}{36}$$

```
sum(tapply(Glycogen,Treatment,sum)^2)/12-5120^2/36
```

```
[1] 1557.556
```

So the Error sums of squares must be SST – SSA = 3330.222-1557.556 = 1772.666. If this is correct, then the ANOVA Table must be like this:

Source	SS	d.f.	MS	F	Critical F
Treatment	1557.556	2	778.778	14.497	3.28
Error	1772.66	33	53.717		
Total	3330.222	35			

```
qf(.95,2,33)
```

```
[1] 3.284918
```

The calculated value is much larger than the value in tables, so treatment has a highly significant effect on liver glycogen content. **Wrong !** We have made the classic mistake of pseudoreplication. We have counted all 36 data points as if they were replicates. The definition of replicates is that *replicates are independent of one another*. Two measurements from the same piece of rat's liver are clearly not independent. Nor are measures from three regions of the same rat liver. It is the rats that are the replicates in this experiment and there are only 6 of them in total ! So the correct total degrees of freedom is 5, not 35, and *there are 5-2 = 3 degrees of freedom for error, not 33*.

There are lots of ways of doing this analysis wrong, but only one way of doing it right. There are 3 spatial scales to the measurements (rats, liver bits from each rat and preparations from each liver bit) and hence there must be 3 different error variances in the analysis. Our first task is to compute the sum of squares for differences between the rats. It is easy to find the sums:

tapply(Glycogen,list(Treatment,Rat),sum)

```
    1   2
1 795 891
2 898 914
3 806 816
```

Note the use of Treatment and Rat to get the totals: it is **wrong** to do the following:

tapply(Glycogen,Rat,sum)

```
   1    2
2499 2621
```

because the rats are numbered 1 and 2 within each treatment. **This is very important**. There are 6 rats in the experiment, so we need 6 rat totals. Each rat total is the sum of 6 numbers (3 liver bits and 2 preparations per liver bit). So we square the 6 rat totals, add them up and divide by 6:

sum(tapply(Glycogen,list(Treatment,Rat),sum)^2)/6

```
[1] 730533
```

But what now? Our experience so far tells us simply to subtract the correction factor. But this is wrong. We use *the sum of squares from the spatial scale above* as the correction term at any given level. This may become clearer from the worked example.

What about the sum of squares for liver bits? There are 3 per rat, so there must be 18 liver-bit totals in all. We can inspect them like this:

tapply(Glycogen,list(Treatment,Rat,Liver),sum)

261	298	256	283	278	310
302	306	296	294	300	314
259	278	276	277	271	261

We need to square these, add them up and divide by 2. Why 2 ? Because each of these totals is the sum of the 2 measurements of each preparation from each liver bit.

sum(tapply(Glycogen,list(Treatment,Rat,Liver),sum)^2)/2

```
[1] 731127
```

So we have the following uncorrected sums of squares

Source	**Uncorrected SS**

Treatment	729735.3
Rats	730533
Liver bits	731127
Total	731508

The key to understanding nested designs is to understand the next step

Up to now, we have used the correction factor

$$CF = \frac{\left[\sum y\right]^2}{\sum n}$$

to determine the corrected sum of squares in all cases. We used it (correctly) to determine the treatment sum of squares, earlier in this example. With nested factors, however, we don't do this. **We use the uncorrected sum of squares from the next spatial scale larger than the scale in question**.

$$SS_{Rats} = \frac{\sum R^2}{6} - \frac{\sum T^2}{12}$$

Where R is a rat total and T is a treatment total. The sum of squares of rat totals is divided by 6 because each total was the sum of 6 numbers. The sum of squares of treatment totals is divided by 12 because each treatment total was the sum of 12 numbers (2 rats each generating 6 numbers). So, if L is the sum of the 2 liver-bit preparations, and y is an individual preparation (the "sum of 1 numbers" if you like), we get

$$SS_{Liver.Bits} = \frac{\sum L^2}{2} - \frac{\sum R^2}{6}$$

$$SS_{P.reparations} = \frac{\sum y^2}{1} - \frac{\sum L^2}{2}$$

We can now compute the numerical values of the nested sums of squares:

$$SS_{Rats} = 730533 - 729735.3 = 797.7$$

$$SS_{LiveBits} = 731127 - 730533 = 594.0$$

$$SS_{P.reparations} = 731508 - 731127 = 381.0$$

So now we can fill in the ANOVA table correctly, taking account of the nesting and the pseudoreplication.

Source	SS	d.f.	MS	F	Critical F
Treatment	1557.556	2	778.778	2.929	9.552094
Rats in Treatments	797.7	3	265.9	5.372	3.490295
Liver bits in Rats	594.0	12	49.5	2.339	2.342067
Readings in Liver bits	381.0	18	21.1666		
Total	3330.222	35			

There are several important things to see in this Anova table.

- First, you use the mean square from the spatial scale immediately below in testing significance. (You do **not** use 21.166 as we have done up to now). So the F ratio for treatment is 778.778/265.9 = 2.929 on 2 and 3 d.f. which is way short of significance (the critical value is 9.55; compare this with what we did wrong, earlier).
- Second, a different recipe is used in nested designs to compute the degrees of freedom. Treatment and Total degrees of freedom are the same as usual but the others are different
- There are 6 rats in total but there are not 5 d.f. of freedom for rats: there are two rats per treatment, so there is 1 d.f. for rats within each treatment. There are 3 treatments so there are 3 x 1 = 3 d.f. for rats within treatments.
- There are 18 liver bits in total but there are not 17 d.f. for liver bits: there are 3 bits in each liver, so there are 2 d.f. for liver bits within each liver. Since there are 6 livers in total (one for each rat) there are 6 x 2 = 12 d.f. for liver bits within rats.
- There are 36 preparations but there are not 35 d.f. for preparations; there are 2 preparations per liver bit, so there is 1 d.f. for preparation within each liver bit. There are 18 liver bits in all (3 from each of 6 rats) and so there are 1 x 18 = 18 d.f. for preparations within liver bits.
- Using the 3 different error variances to carry out the appropriate F tests we learn that there are no significant differences between treatments but there are significant differences between rats, and between parts of the liver within rats (i.e. small scale spatial heterogeneity in the distribution of glycogen within livers).

Now we carry out the analysis in SPlus. The new concept here is that we include an **Error** term in the model formula to show:

- How many error terms are required (answer: as many as there are plot sizes)
- What is the hierarchy of plot sizes (biggest on the left, smallest on the right)
- After the tilde ~ in the model formula we have Treatment as the only fixed effect
- The Error term follows a plus sign + and is enclosed in round brackets
- The plots, ranked by their relative sizes are separated by slash operators like this

```
model<-aov(Glycogen~Treatment+Error(Treatment/Rat/Liver))
```

```
summary(model)
```

```
Error: Treatment
          Df  Sum Sq Mean Sq
Treatment  2 1557.56  778.78

Error: Treatment:Rat
              Df Sum Sq Mean Sq
Treatment:Rat  3 797.67  265.89

Error: Treatment:Rat:Liver
                     Df Sum Sq Mean Sq
Treatment:Rat:Liver 12  594.0    49.5

Error: Within
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 18 381.00   21.17
```

These are the correct mean squares, as we can see from our long-hand calculations earlier. The F test for the effect of treatment is 778.78/265.89 = 2.93 (n.s.), for differences between rats within treatments it is 265.89/49.5 = 5.37 (p<0.05) and for liver bits within rats F = 49.5/21.17 = 2.24 (p = 0.05).

In general, we use the slash operator in **model formulas** where the variables are random effects (i.e. where the factor levels are uninformative), and the asterisk operator in model formulas where the variables are fixed effects (i.e. the factor levels **are** informative). Knowing that a rat is number 2 tells us nothing about that rat. Knowing a rat is male tells us a lot about that rat. In **error formulas** we **always use the slash operator** (never the asterisk operator) to indicate the order of 'plot-sizes': the largest plots are on the left of the list and the smallest on the right (see below).

The Wrong Analysis

Here is what ***not*** to do.

```
model2<-aov(Glycogen~Treatment*Rat*Liver)
```

The model has been specified as if it were a full factorial with no nesting and no pseudoreplication. Note that the structure of the data allows this mistake to be made. It is a very common problem with data frames that include pseudoreplication. A summary of the model fit looks like this:

```
summary(model2)
```

```
                    Df  Sum Sq Mean Sq F value     Pr(>F)
Treatment            2 1557.56  778.78 36.7927  4.375e-07***
Rat                  1  413.44  413.44 19.5328  0.0003308***
Liver                2  113.56   56.78  2.6824  0.0955848 .
```

```
Treatment:Rat         2  384.22  192.11  9.0761  0.0018803 **
Treatment:Liver       4  328.11   82.03  3.8753  0.0192714 *
Rat:Liver             2   50.89   25.44  1.2021  0.3235761
Treatment:Rat:Liver   4  101.44   25.36  1.1982  0.3455924
Residuals            18  381.00   21.17
```

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1 summary(nested1)

This says that there was an enormously significant difference between the treatment means, rats were significantly different from one another and there was a significant interaction between treatments and rats. ***Wrong !*** The analysis is flawed because it is based on the assumption that there is only one error variance and that its value is 21.17. This value is actually the measurement error; that is to say the variation between one reading and another from the *same* piece of liver. For testing whether the treatment has had any effect, it is the rats that are the replicates, and there were only 6 of them in the whole experiment.

Here is a way to avoid making the mistake of pseudoreplication

The idea is to get rid of the pseudoreplication by averaging over the liver bits and preparations for each rat. We need to create a new vector, *ym*, of length 6 containing the mean glycogen levels of each rat. You can see how this works as follows

```
tapply(Glycogen,list(Treatment,Rat),mean)
```

```
          1        2
1 132.5000 148.5000
2 149.6667 152.3333
3 134.3333 136.0000
```

We make this into a vector for use in the model like this:

```
ym<-as.vector(tapply(Glycogen,list(Treatment,Rat),mean))
```

We also need a new vector, *tm*, of length 6 to contain a factor for the 3 treatment levels:

```
tm<-factor(as.vector(tapply(as.numeric(Treatment),list(Treatment,Rat),mean)))
```

Now we can do the anova:

```
summary(aov(ym~tm))
```

```
            Df  Sum Sq Mean Sq F value Pr(>F)
tm           2 259.593 129.796   2.929 0.1971
Residuals    3 132.944  44.315
```

This gives us the same (correct) value of the F test as when we did the full nested analysis. The sums of squares are different, of course, because we are using 6 mean

values rather than 36 raw values of glycogen to carry out the analysis. We conclude (correctly) that treatment had no significant effect on liver glycogen content.

Analysis of split plot experiments

Split plot experiments are like nested designs in that they involve plots of different sizes and hence have multiple error terms (one error term for each plot size). They are also like nested designs in that they involve pseudoreplication: measurements made on the smaller plots are pseudoreplicates as far as the treatments applied to larger plots are concerned. This is spatial pseudoreplication, and arises because the smaller plots nested within the larger plots are not spatially independent of one another. The only real difference between nested analysis and split plot analysis is that other than blocks, all of the factors in a split plot experiment are typically fixed effects, whereas in most nested analyses most (or all) of the factors are random effects.

The only things to remember about split plot experiments are that

- we need to draw up as many anova tables as there are plot sizes
- the error term in each table is *the interaction between blocks and all factors applied at that plot size or larger*

This experiment involves the yield of cereals in a factorial experiment with 3 treatments, each applied to plots of different sizes. The largest plots (half of each block) were irrigated or not because of the practical difficulties of watering large numbers of small plots. Next, the irrigated plots were split into 3 smaller split-plots and seeds were sown at different densities. Again, because the seeds were machine sown, larger plots were preferred. Finally, each sowing density plot was split into 3 small split-split plots and fertilisers applied by hand (N alone, P alone and N+P together). The yield data look like this:

	Control			Irrigated		
	N	NP	P	N	NP	P
Block A	81	93	92	78	122	98
	90	107	95	80	100	87
	92	92	89	121	119	110
Block B	74	74	81	136	132	133
	83	95	80	102	105	109
	98	106	98	99	123	94
Block C	82	94	78	119	136	122
	85	88	88	60	114	104
	112	91	104	90	113	118
Block D	85	83	89	116	133	136
	86	89	78	73	114	114
	79	87	86	109	126	131

We begin by calculating SST in the usual way.

$$CF = \frac{\left[\sum y\right]^2}{abcd} = \frac{7180^2}{72} = 716005.6$$

$$SST = \sum y^2 - \frac{\left[\sum y\right]^2}{abcd} = 739762 - CF = 23756.44$$

The block sum of squares, SSB, is straightforward. All we need are the 4 block totals

$$SSB = \frac{\sum B^2}{acd} - CF = \frac{1746^2 + 1822^2 + 1798^2 + 1814^2}{18} - CF = 194.444$$

The irrigation main effect is calculated in a similar fashion

$$SSI = \frac{\sum I^2}{bcd} - CF = \frac{3204^2 + 3976^2}{4 \times 3 \times 3} - CF = 8277.556$$

This is where things get different. In a split plot experiment, *the error term is the interaction between blocks and all factors at that plot size or larger*. Because irrigation treatments are applied to the largest plots, the error term is just block:irrigation. We need to calculate an interaction sub-total table (see Practical 5 for an introduction to interaction sums of squares).

	control	irrigated
A	831	915
B	789	1033
C	822	976
D	762	1052

The large plot error sum of squares SSBI is therefore

$$SSBI = \frac{\sum Q^2}{cd} - SSB - SSI - CF$$

$$SSBI = \frac{831^2 + 915^2 + ... + 1052^2}{3 \times 3} - SSB - SSI - CF = 1411.778$$

At this point we draw up the anova table for the largest plots. There were only 8 large plots in the whole experiment, so there are just 7 degrees of freedom in total. Block has 3 d.f., irrigation has 1 d.f., so the error variance has only 7-3-1 = 3 d.f.

Source	SS	d.f.	MS	F	p
Block	194.44	3			
Irrigation	8277.556	1	8277.556	17.59	0.025
Error	1411.778	3	470.593		

Block is a random effect so we are not interested in testing hypotheses about differences between block means. We have not written in a row for the totals, because we want the totals and the degrees of freedom to add up correctly across the 3 different anova tables.

Now we move on to consider the sowing density effects. At this split-plot scale we are interested in the main effects of sowing density, and the interaction between sowing density and irrigation. The error term will be the block:irrigation:density interaction.

$$SSD = \frac{2467^2 + 2226^2 + 2487^2}{4 \times 2 \times 3} - CF = 1758.361$$

For the irrigation:density interaction we need the table of sub totals

	high	low	medium
control	1006	1064	1134
irrigated	1461	1162	1353

$$SSID = \frac{1006^2 + ... + 1353^2}{4 \times 3} - SSI - SSD - CF = 2747.028$$

The error term for the split-plots is the block:irrigation:density interaction, and we need the table of sub totals:

	high		low		medium	
	control	irrigated	control	irrigated	control	irrigated
A	266	298	292	267	273	350
B	229	401	258	316	302	316
C	254	377	261	278	307	321
D	257	385	253	301	252	36

$$SSBID = \frac{226^2 + 298^2 + \ldots + 252^2 + 36^2}{3} - SSBI - SSID - SSB - SSI - SSD - CF = 2787.944$$

At this point we draw up the second anova table for the split-plots:

Source	SS	d.f.	MS	F	p
Density	1758.361	2	879.181	3.784	0.053
Irrigation:Density	2747.028	2	1373.514	5.912	0.016
Error	2787.944	12	232.329		

There are 4 x 2 x 3 = 24 of these plots so there are 23 d.f. in total. We have used up 7 in the first anova table: there are 2 for density, 2 for irrigation:density and hence 23 – 7 – 2 – 2 = 12 d.f. for error. There is a significant interaction between irrigation and density, so we take no notice of the non significant main effect of density.

Finally, we move on to the smallest, split-split-plots. We first calculate the main effect of fertilizer in the familiar way:

$$SSF = \frac{\sum F^2}{abc} - CF = \frac{2230^2 + 2536^2 + 2414^2}{2 \times 4 \times 3} - CF = 1977.444$$

Now the irrigation:fertilizer interaction: the interaction sub totals are

	N	NP	P
control	1047	1099	1058
irrigated	1183	1437	1356

$$SSIF = \frac{1047^2 + 1099^2 + \ldots + 1356^2}{4 \times 3} - SSI - SSF - CF = 953.444$$

and the density:fertilizer interaction

	N	NP	P
high	771	867	829
low	659	812	755
medium	800	857	830

$$SSDF = \frac{771^2 + 867^2 + \ldots + 830^2}{4 \times 2} - SSD - SSF - CF = 304.889$$

The final 3-way interaction is calculated next: irrigation:density:fertilizer

```
             N                    NP                    P
          high  low medium  high low medium  high low medium
  control  322   344   381   344 379   376   340   341   377
irrigated  449   315   419   523 433   481   489   414   453
```

$$SSIDF = \frac{322^2 + 344^2 + \ldots + 414^2 + 453^2}{4} - SSID - SSIF - SSDF - SSI - SSD - SSF - CF = 234.722$$

The rest is easy. The error sum of squares is just the remainder when all the calculated sums of squares are subtracted from the total:

$$SSE = SST - SSB - SSI - SSD - SSF - SSIB - SSID - \ldots - SSIDF = 3108.833$$

Technically, this is the block:irrigation:density:fertilizer interaction. There are 72 plots at this scale and we have used up 7 + 16 = 23 degrees of freedom in the first two anova tables. In the last anova table fertilizer has 2 d.f., the fertilizer:irrigation interaction has 2 d.f., the fertilizer: density interaction has 4 d.f. and the irrigation:density:fertilizer interaction has a further 4 d.f.. This leaves 71- 23 – 2 – 2 – 4 – 4 = 36 d.f. for error. At this point we can draw up the final anova table.

Source	SS	d.f.	MS	F	p
Fertilizer	1977.444	2	988.722	11.449	0.00014
Irrigation:Fertilizer	953.444	2	476.722	5.52	0.0081
Density:Fertilizer	304.889	4	76.222	0.883	n.s.
Irrigtn:Density:Fertilr	234.722	4	58.681	0.68	n.s.
Error	3108.833	36	86.356		

The 3-way interaction was not significant, nor was the 2-way interaction between density and fertilizer. The 2-way interaction between irrigation and fertilizer, however, was highly significant and, not surprisingly, there was a highly significant main effect of fertilizer.

Obviously, you would not want to have to do calculations like this by hand every day. Fortunately, the computer eats analyses like this for breakfast.

```
splityield<-read.table("c:\\temp\\splityield.txt",header=T)
attach(splityield)
names(splityield)

[1] "yield"  "block"  "irrigation" "density"  "fertilizer"
```

```
model<-aov(yield~irrigation*density*fertilizer+Error(block/irrigation/density/fertilizer))
```

The model is long, but not particularly complicated. Note the two parts: the model formula (the factorial design: irrigation*density*fertilizer), and the Error structure (with plot sizes listed left to right from largest to smallest, separated by slash / operators). The main replicates are blocks, and these provide the estimate of the error variance for the largest treatment plots (irrigation). We use asterisks in the model formula because these are fixed effects (i.e. their factor levels *are* informative).

```
summary(model)
```

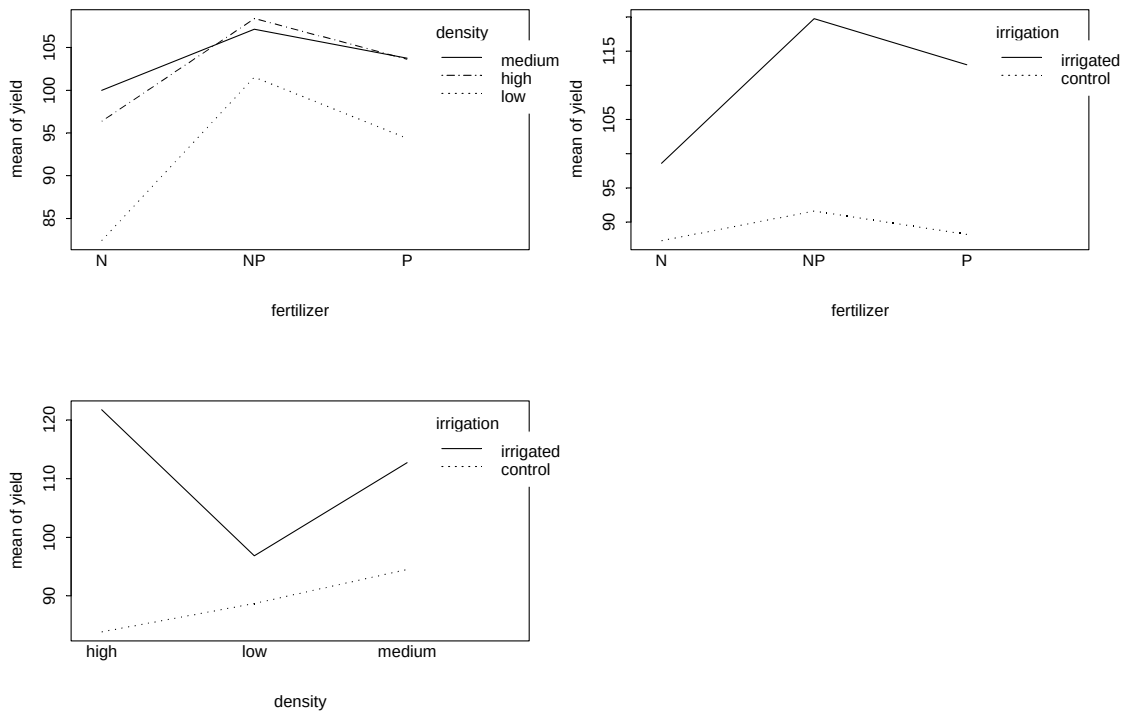
This produces a series of anova tables, one for each plot size, starting with the largest plots (block), then looking at irrigation within blocks, then density within irrigation within block, then finally fertilizer within density within irrigation within block. Notice that the error degrees of freedom are correct in each case (e.g. there are only 3 d.f. for error in assessing the irrigation main effect, but it is nevertheless significant; p = 0.025).

```
Error: block
      Df  Sum Sq Mean Sq
block  3 194.444  64.815

Error: block:irrigation
           Df Sum Sq Mean Sq F value  Pr(>F)
irrigation  1 8277.6  8277.6  17.590 0.02473 *
Residuals   3 1411.8   470.6

Error: block:irrigation:density

                   Df  Sum Sq Mean Sq F value  Pr(>F)
density             2 1758.36  879.18  3.7842 0.05318 .
irrigation:density  2 2747.03 1373.51  5.9119 0.01633 *
Residuals          12 2787.94  232.33

Error: block:irrigation:density:fertilizer

                             Df  Sum Sq Mean Sq F value    Pr(>F)
fertilizer                    2 1977.44  988.72 11.4493 0.0001418 ***
irrigation:fertilizer         2  953.44  476.72  5.5204 0.0081078 **
density:fertilizer            4  304.89   76.22  0.8826 0.4840526
irrigation:density:fertilizer 4  234.72   58.68  0.6795 0.6106672
Residuals                    36 3108.83   86.36
```

There are two significant interactions. The best way to understand these is to use the **interaction.plot** directive. The variables are listed in a non-obvious order: first the factor to go on the x axis, then the factor to go as different lines on the plot, then the response variable. There are 3 plots to look at so we make a 2 x 2 plotting area:

```
par(mfrow=c(2,2))
interaction.plot(fertilizer,density,yield)
interaction.plot(fertilizer,irrigation,yield)
interaction.plot(density,irrigation,yield)
```

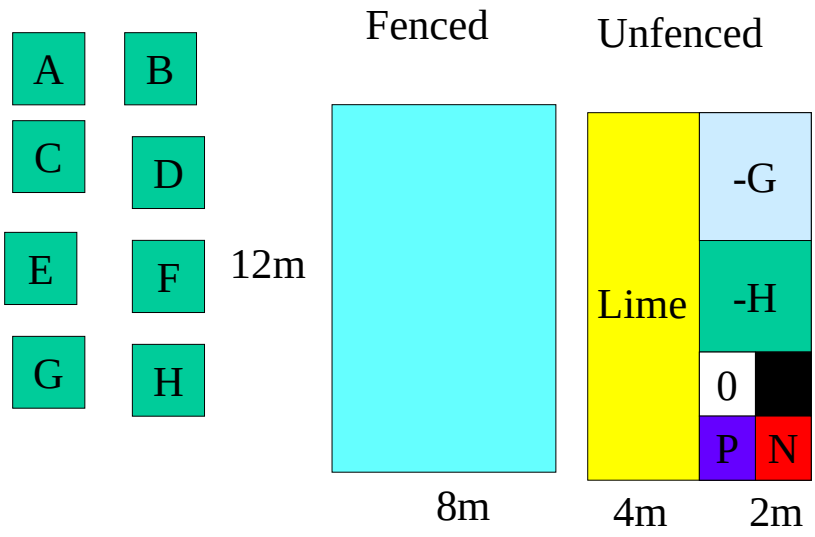
```
par(mfrow=c(1,1))
```



Obviously, the really pronounced interaction is that between irrigation and density, with a reversal of the high to low density difference on the irrigated and control plots. Interestingly, this is not the most significant interaction. That honour goes to the fertilizer:irrigation interaction (top right graph).

A complex split-split-split-split plot field experiment

This example explains the analysis of a field experiment on the impact of grazing and plant competition on the yield of forage biomass (measured as dry matter in tonnes per hectare). To understand what is going on you will need to study the experimental layout quite carefully.



There are 8 large plots (A to H), forming the Blocks of the experiment each measuring 12 x 16 m. Two treatments are applied at this scale: plus and minus Insecticide spray and plus and minus Mollusc pellets. Each of the 4 treatment combinations is replicated twice, and applied at random to one of the 8 Blocks. Each Block is split in half in a randomly selected direction (up and down, or left to right) and Rabbit fencing is allocated at random to half of the Block. Within each split plot the plot is split again, and 1 of 2 liming treatments is allocated at random. Within each split-split plot the area is divided into 3, and 1 of 3 plant competition treatments is allocated at random (control ,minus grass using a grass-specific herbicide, or minus herb using a herb-specific herbicide). Finally, within each split-split-split plot the area is divided into 4 and 1 of 4 nutrient treatments is applied at random (plus and minus nitrogen and plus and minus phosphorus). The whole design, therefore, contains

$$8 \times 2 \times 2 \times 3 \times 4 = 384$$

of the smallest (2m x 2m) plots. The data frame looks like this:

```
splitplot<-read.table("c:\\temp\\splitplot.txt",header=T)
attach(splitplot)
names(splitplot)
```

```
[1] "Block"   "Insect"   "Mollusc"  "Rabbit"  "Lime"   "Competition"
[7] "Nutrient"    "Biomass"
```

The **response variable** is Biomass, Block is a **random effect** and the other variables are all **fixed effects** applied as treatments (at random of course, which is a bit confusing). As before, analysing the data requires us to specify 2 things: the treatment structure and the error structure. The treatment structure is simply a full factorial

```
Insect*Mollusc*Rabbit*Lime*Competition*Nutrient
```

Specified using the * operator to indicate that all main effects and all interactions are to be fitted. The error term shows how the different plot sizes are related to the explanatory variables. We list, for left to right, the names of the variables relating to progressively smaller plots, with each name separated by the / (slash) operator:

```
Block/Rabbit/Lime/Competition/Nutrient
```

There are 5 variable names in the Error directive because there are 5 different sizes of plots (2 x 2m for nutrients, 4 x 4m for competition, 12 x 4m for lime, 12 x 8m for rabbit grazing and 12 x 16m for insecticide or molluscicide). Where treatments are applied to plots of the same size (e.g. insecticide and molluscicide in this example) we need only specify *one* of the names (it does not matter which: we used Insect but we could equally well have used Mollusc). The analysis is run by combining the treatment and error structure in a single **aov** directive (it may not run on your machine because of memory limitations: but here is the SPlus output anyway)

```
model<-aov(Biomass~Insect*Mollusc*Rabbit*Lime*Competition*Nutrient
                        +Error(Block/Rabbit/Lime/Competition/Nutrient))

summary(model)
```

```
Error: Block
               Df Sum of Sq  Mean Sq  F Value     Pr(F)
        Insect  1  414.6085 414.6085 34.27117 0.0042482
       Mollusc  1    8.7458   8.7458  0.72292 0.4430877
Insect:Mollusc  1   11.0567  11.0567  0.91394 0.3932091
     Residuals  4   48.3915  12.0979

Error: Rabbit %in% Block
                     Df Sum of Sq  Mean Sq  F Value     Pr(F)
              Rabbit  1  388.7935 388.7935 4563.592 0.0000003
        Insect:Rabbit  1    0.4003   0.4003    4.698 0.0960688
       Mollusc:Rabbit  1    0.0136   0.0136    0.160 0.7096319
Insect:Mollusc:Rabbit  1    0.2477   0.2477    2.908 0.1633515
           Residuals  4    0.3408   0.0852

Error: Lime %in% (Block/Rabbit)
                          Df Sum of Sq  Mean Sq  F Value     Pr(F)
                   Lime  1  86.63703 86.63703 1918.264 0.0000000
             Insect:Lime  1   0.03413  0.03413    0.756 0.4100144
            Mollusc:Lime  1   0.12197  0.12197    2.701 0.1389385
             Rabbit:Lime  1   0.14581  0.14581    3.228 0.1100955
       Insect:Mollusc:Lime  1   0.05160  0.05160    1.143 0.3163116
        Insect:Rabbit:Lime  1   0.00359  0.00359    0.079 0.7852903
       Mollusc:Rabbit:Lime  1   0.09052  0.09052    2.004 0.1945819
Insect:Mollusc:Rabbit:Lime  1   0.46679  0.46679   10.335 0.0123340
                Residuals  8   0.36131  0.04516
```

```
Error: Competition %in% (Block/Rabbit/Lime)

                                     Df Sum of Sq  Mean Sq  F Value     Pr(F)
                            Competition  2  214.4145 107.2073 1188.317 0.0000000
                      Insect:Competition  2    0.1502   0.0751    0.832 0.4442496
                     Mollusc:Competition  2    0.1563   0.0782    0.866 0.4300752
                      Rabbit:Competition  2    0.1981   0.0991    1.098 0.3457568
                        Lime:Competition  2    0.0226   0.0113    0.125 0.8825194
               Insect:Mollusc:Competition  2    0.4132   0.2066    2.290 0.1176343
                Insect:Rabbit:Competition  2    0.1221   0.0611    0.677 0.5153674
               Mollusc:Rabbit:Competition  2    0.0221   0.0111    0.123 0.8850922
                  Insect:Lime:Competition  2    0.0527   0.0263    0.292 0.7487901
                 Mollusc:Lime:Competition  2    0.0296   0.0148    0.164 0.8493921
                  Rabbit:Lime:Competition  2    0.0134   0.0067    0.074 0.9286778
       Insect:Mollusc:Rabbit:Competition  2    0.0307   0.0154    0.170 0.8442710
         Insect:Mollusc:Lime:Competition  2    0.0621   0.0311    0.344 0.7112350
          Insect:Rabbit:Lime:Competition  2    0.3755   0.1878    2.081 0.1413456
         Mollusc:Rabbit:Lime:Competition  2    0.5007   0.2504    2.775 0.0773730
  Insect:Mollusc:Rabbit:Lime:Competition  2    0.0115   0.0057    0.064 0.9385470
                             Residuals 32    2.8870   0.0902

Error: Nutrient %in% (Block/Rabbit/Lime/Competition)
                                          Df Sum of Sq  Mean Sq  F Value     Pr(F)
                                 Nutrient   3  1426.017 475.3389 6992.589 0.0000000
                           Insect:Nutrient   3     0.213   0.0711    1.046 0.3743566
                          Mollusc:Nutrient   3     0.120   0.0400    0.589 0.6231103
                           Rabbit:Nutrient   3     0.087   0.0291    0.428 0.7332115
                             Lime:Nutrient   3     0.035   0.0116    0.171 0.9156675
                      Competition:Nutrient   6     0.724   0.1207    1.775 0.1081838
                 Insect:Mollusc:Nutrient   3     0.112   0.0373    0.549 0.6497462
                  Insect:Rabbit:Nutrient   3     0.783   0.2611    3.840 0.0110859
                 Mollusc:Rabbit:Nutrient   3     0.929   0.3096    4.554 0.0044319
                    Insect:Lime:Nutrient   3     0.059   0.0196    0.289 0.8335956
                   Mollusc:Lime:Nutrient   3     0.476   0.1585    2.332 0.0766893
                    Rabbit:Lime:Nutrient   3     0.376   0.1252    1.842 0.1422169
             Insect:Competition:Nutrient   6     0.556   0.0927    1.363 0.2333048
            Mollusc:Competition:Nutrient   6     0.347   0.0578    0.851 0.5327535
             Rabbit:Competition:Nutrient   6     0.431   0.0718    1.057 0.3914693
               Lime:Competition:Nutrient   6     0.451   0.0752    1.106 0.3616015
          Insect:Mollusc:Rabbit:Nutrient   3     0.411   0.1370    2.016 0.1143309
            Insect:Mollusc:Lime:Nutrient   3     0.697   0.2324    3.418 0.0190776
             Insect:Rabbit:Lime:Nutrient   3     0.435   0.1450    2.132 0.0987222
            Mollusc:Rabbit:Lime:Nutrient   3     0.055   0.0184    0.270 0.8468838
       Insect:Mollusc:Competition:Nutrient   6     0.561   0.0936    1.376 0.2279809
        Insect:Rabbit:Competition:Nutrient   6     0.728   0.1213    1.784 0.1062479
       Mollusc:Rabbit:Competition:Nutrient   6     0.427   0.0712    1.047 0.3974232
          Insect:Lime:Competition:Nutrient   6     0.258   0.0430    0.632 0.7043702
         Mollusc:Lime:Competition:Nutrient   6     0.417   0.0695    1.023 0.4128486
          Rabbit:Lime:Competition:Nutrient   6     0.406   0.0676    0.994 0.4315034
       Insect:Mollusc:Rabbit:Lime:Nutrient   3     0.350   0.1166    1.715 0.1665522
 Insect:Mollusc:Rabbit:Competition:Nutrient   6     0.259   0.0431    0.635 0.7023702
   Insect:Mollusc:Lime:Competition:Nutrient   6     0.403   0.0672    0.988 0.4355882
    Insect:Rabbit:Lime:Competition:Nutrient   6     0.282   0.0470    0.692 0.6565638
   Mollusc:Rabbit:Lime:Competition:Nutrient   6     0.355   0.0592    0.870 0.5183944
Insect:Mollusc:Rabbit:Lime:Competition:Nutrient   6     0.989   0.1648    2.424 0.0291380
                                       Residuals 144     9.789   0.0680
```

Notice that you get 5 separate ANOVA tables, one for each different plot size. It is the number of plot sizes, not the number of treatments that determines the shape of the split-plot ANOVA table. The number of ANOVA tables would not have changed if we had specified the Nutrient treatment (4 levels) as a 2 by 2 factorial with Nitrogen and Phosphorus each as 2-level treatments. Because Insecticide and Molluscicide were both applied at the same plot size (whole Blocks) they appear in the same ANOVA table.

Interpretation of output tables like this requires a high level of serenity. The first thing to do is to check that the degrees of freedom have been handled properly. There were 8 bocks, with 2 replicates of the large-plot factorial experiment of plus and minus insects and plus and minus molluscs. This means that there are 7 d.f. in total, and so with 1 d.f. for Insect, 1 d.f. for Mollusc and 1 d.f. for the Insect:Mollusc interaction, there should be

7-1-1-1 = 4 d.f. for error. This checks out in the top ANOVA table where Error is labelled as Block. For the largest plots, therefore, the error variance = 12.1. We can now assess the significance of these treatments that were applied to the largest plots. As ever, we begin with the interaction. This is clearly not significant, so we can move on to interpreting the main effects. There is no effect of mollusc exclusion on biomass, but insect exclusion led to a significant increase in mean biomass ($p = 0.0042$).

The 2nd largest plots were those with or without fences to protect them from rabbit grazing. To check the degrees of freedom, we need to work out the total number of rabbit-grazed and fenced plots. There were 8 blocks, each split in half, so there are 16 plots. We have already used 7 d.f. for the insect by mollusc experiment (above) so there are 16 – 7 – 1 = 8 d.f. remaining. Rabbit grazing has 1 d.f. and there are 3 interaction terms, each with 1 d.f. (Rabbit:Insect, Rabbit:Mollusc and Rabbit:Insect:Mollusc). This means that these terms should be assessed by an error variance that has 8 – 1- 3 = 4 d.f. This also checks out. The error variance is 0.085, and shows that there are no significant interactions between rabbit grazing and invertebrate herbivores, but there is a highly significant main effect of rabbit grazing.

The 3rd largest plots were either limed or not limed. In this case there are 8 d.f. for error, and we discover the first significant interaction: Insect:Mollusc:Rabbit:Lime ($p = 0.012$). Like all high-order interactions, this is extremely complicated to interpret. It means that the 3-way interaction between Insect:Mollusc:Rabbit works differently on limed and unlimed plots. It would be unwise to over-interpret this result without further experimentation focussed on the way that this interaction might work. There is a highly significant main effect of lime.

The 4th largest plots received one of 3 plant competition treatments: control, minus grass or minus herb. There are 96 competition plots, and we have used 31 d.f. so far on the larger plots, so there should be 96 – 31 – 1 = 64 d.f. at this level. With 32 d.f. for main effects and interactions, that leaves 32 d.f. for error. This checks out, and the error variance is 0.09. There are no significant interaction terms, but competition had a significant main effect on biomass.

The 5th largest plots (the smallest at 2m x 2m) received one of 4 nutrient treatments: plus or minus nitrogen and plus or minus phosphorus. All the remaining degrees of freedom can be used at this scale, leaving 144 d.f. for error, and an error variance of 0.068. There are several significant interactions: the 6-way Insect:Mollusc:Rabbit:Lime: Competition:Nutrient ($p = 0.029$), a 4-way, Insect:Mollusc:Lime:Nutrient ($p = 0.019$), and two 3-way interactions, Insect:Rabbit:Nutrient ($p = 0.011$) and Mollusc:Rabbit:Nutrient ($p = 0.004$). At this point I can confide in you: I made up these results, so I know that all of these small-plot interactions are due to chance alone.

This raises an important general point. In big, complicated experiments like this it is sensible to use a very high level of alpha in assessing the significance of high order interactions. This compensates for the fact that you are doing a vast number of hypothesis tests, and has the added bonus of making the results much more straightforward to write

up. It is a trade-off, of course, because you do not want to be so harsh that you throw out the baby with the bathwater, and miss biologically important and potentially very interesting interactions.

We can inspect the interactions in 2 ways. Tables of interaction means can be produced using **tapply**:

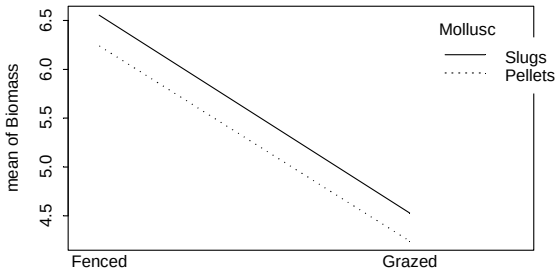
```
tapply(Biomass,list(Mollusc,Rabbit,Nutrient),mean)
```

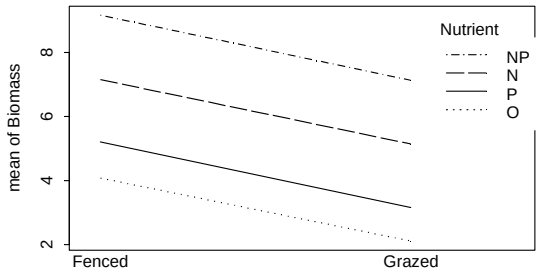
```
, , N
          Fenced   Grazed
Pellets 6.963431 4.984890
  Slugs 7.322273 5.298524

, , NP
          Fenced   Grazed
Pellets 9.056132 6.923177
  Slugs 9.257630 7.324827

, , O
          Fenced   Grazed
Pellets 3.873364 2.069350
  Slugs 4.282690 2.149054

, , P
          Fenced   Grazed
Pellets 5.066604 2.979994
  Slugs 5.351919 3.344672
```

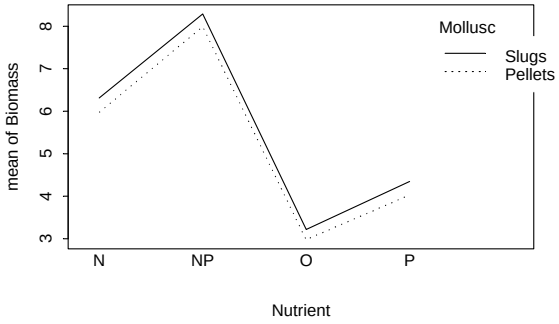
Better still, use **interaction.plot** to inspect the interaction terms, 2 at a time. Take the Mollusc:Rabbit:Nutrient interaction ($p = 0.004$) whose means we have just calculated. We expect that the interaction plot will show non-parallelness of some form or other. We divide the plotting space into 4, then make 3 separate interaction plots:

```
par(mfrow=c(2,2))
interaction.plot(Rabbit,Mollusc,Biomass)
interaction.plot(Rabbit,Nutrient,Biomass)
interaction.plot(Nutrient,Mollusc,Biomass)
```

It is quite clear from the plots that the interaction, though statistically significant, is not biologically substantial. The plots are virtually parallel in all 3 panels. The graphs demonstrate clearly the interaction between nitrogen and phosphorus (bottom left) but this is not materially altered by either mollusc or rabbit grazing.







Doing the wrong "Factorial" analysis of multi-split-plot data

Here's how **not** to do it. With data like this there are always lots of different ways of getting the model and/or the error structure wrong. The commonest mistake is to treat the data as a full factorial, like this:

```
model<-aov(Biomass~Insect*Mollusc*Rabbit*Lime*Competition*Nutrient)
```

It looks perfectly reasonable, and indeed, this is the experimental design intended. But it is rife with pseudoreplication. Ask yourself how many independent plots had insecticide applied to them or not. The answer is 8. The analysis carried out here assumes there were 384 ! And what about plant competition treatments ? The answer is 96, but this analysis assumes 384. And so on. Let's see the consequences of the pseudoreplication: It thinks the error variance is 0.3217 for testing all the interactions and main effects. The only significant interaction is Insecticide by Molluscicide and this appears to have an F ratio of 34.4 on d.f. = 1,192. All 6 main effects appear to be significant at $p<0.000001$.

summary(model)

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
Insect	1	414.609	414.6085	1288.743	0.0000000
Mollusc	1	8.746	8.7458	27.185	0.0000005
Rabbit	1	388.793	388.7935	1208.502	0.0000000
Lime	1	86.637	86.6370	269.297	0.0000000
Competition	2	214.415	107.2073	333.236	0.0000000
Nutrient	3	1426.017	475.3389	1477.514	0.0000000
Insect:Mollusc	1	11.057	11.0567	34.368	0.0000000
Insect:Rabbit	1	0.400	0.4003	1.244	0.2660546
Mollusc:Rabbit	1	0.014	0.0136	0.042	0.8371543
Insect:Lime	1	0.034	0.0341	0.106	0.7450036
Mollusc:Lime	1	0.122	0.1220	0.379	0.5388059
Rabbit:Lime	1	0.146	0.1458	0.453	0.5016203
Insect:Competition	2	0.150	0.0751	0.233	0.7920629
Mollusc:Competition	2	0.156	0.0782	0.243	0.7845329
Rabbit:Competition	2	0.198	0.0991	0.308	0.7353323
Lime:Competition	2	0.023	0.0113	0.035	0.9654345
Insect:Nutrient	3	0.213	0.0711	0.221	0.8817673
Mollusc:Nutrient	3	0.120	0.0400	0.124	0.9455545
Rabbit:Nutrient	3	0.087	0.0291	0.090	0.9652343
Lime:Nutrient	3	0.035	0.0116	0.036	0.9907599
Competition:Nutrient	6	0.724	0.1207	0.375	0.8942470
Insect:Mollusc:Rabbit	1	0.248	0.2477	0.770	0.3813061
Insect:Mollusc:Lime	1	0.052	0.0516	0.160	0.6892415
Insect:Rabbit:Lime	1	0.004	0.0036	0.011	0.9160383
Mollusc:Rabbit:Lime	1	0.091	0.0905	0.281	0.5964109
Insect:Mollusc:Competition	2	0.413	0.2066	0.642	0.5272844
Insect:Rabbit:Competition	2	0.122	0.0611	0.190	0.8272856
Mollusc:Rabbit:Competition	2	0.022	0.0111	0.034	0.9662288
Insect:Lime:Competition	2	0.053	0.0263	0.082	0.9214289
Mollusc:Lime:Competition	2	0.030	0.0148	0.046	0.9550433
Rabbit:Lime:Competition	2	0.013	0.0067	0.021	0.9794192
Insect:Mollusc:Nutrient	3	0.112	0.0373	0.116	0.9506702
Insect:Rabbit:Nutrient	3	0.783	0.2611	0.811	0.4889277
Mollusc:Rabbit:Nutrient	3	0.929	0.3096	0.962	0.4116812
Insect:Lime:Nutrient	3	0.059	0.0196	0.061	0.9802371
Mollusc:Lime:Nutrient	3	0.476	0.1585	0.493	0.6877598
Rabbit:Lime:Nutrient	3	0.376	0.1252	0.389	0.7609242
Insect:Competition:Nutrient	6	0.556	0.0927	0.288	0.9421006
Mollusc:Competition:Nutrient	6	0.347	0.0578	0.180	0.9820982
Rabbit:Competition:Nutrient	6	0.431	0.0718	0.223	0.9688921
Lime:Competition:Nutrient	6	0.451	0.0752	0.234	0.9651323
Insect:Mollusc:Rabbit:Lime	1	0.467	0.4668	1.451	0.2298613
Insect:Mollusc:Rabbit:Competition	2	0.031	0.0154	0.048	0.9534089
Insect:Mollusc:Lime:Competition	2	0.062	0.0311	0.097	0.9079804
Insect:Rabbit:Lime:Competition	2	0.376	0.1878	0.584	0.5588583
Mollusc:Rabbit:Lime:Competition	2	0.501	0.2504	0.778	0.4606637
Insect:Mollusc:Rabbit:Nutrient	3	0.411	0.1370	0.426	0.7346104
Insect:Mollusc:Lime:Nutrient	3	0.697	0.2324	0.722	0.5397982
Insect:Rabbit:Lime:Nutrient	3	0.435	0.1450	0.451	0.7171854
Mollusc:Rabbit:Lime:Nutrient	3	0.055	0.0184	0.057	0.9820420
Insect:Mollusc:Competition:Nutrient	6	0.561	0.0936	0.291	0.9407903
Insect:Rabbit:Competition:Nutrient	6	0.728	0.1213	0.377	0.8930313
Mollusc:Rabbit:Competition:Nutrient	6	0.427	0.0712	0.221	0.9695915
Insect:Lime:Competition:Nutrient	6	0.258	0.0430	0.134	0.9918574
Mollusc:Lime:Competition:Nutrient	6	0.417	0.0695	0.216	0.9713330
Rabbit:Lime:Competition:Nutrient	6	0.406	0.0676	0.210	0.9733124
Insect:Mollusc:Rabbit:Lime:Competition	2	0.011	0.0057	0.018	0.9823388
Insect:Mollusc:Rabbit:Lime:Nutrient	3	0.350	0.1166	0.362	0.7802436
Insect:Mollusc:Rabbit:Competition:Nutrient	6	0.259	0.0431	0.134	0.9917702
Insect:Mollusc:Lime:Competition:Nutrient	6	0.403	0.0672	0.209	0.9737285
Insect:Rabbit:Lime:Competition:Nutrient	6	0.282	0.0470	0.146	0.9896249
Mollusc:Rabbit:Lime:Competition:Nutrient	6	0.355	0.0592	0.184	0.9810233
Insect:Mollusc:Rabbit:Lime:Competition:Nutrient	6	0.989	0.1648	0.512	0.7987098
Residuals	192	61.769	0.3217		

The problems are of 2 kinds. For the largest plots, the error variance is underestimated because of the pseudoreplication, so things appear significant which actually are not. For example, the correct analysis shows that the Insect by Mollusc interaction is not significant ($p = 0.39$) but the wrong analysis suggests that it is highly significant ($p = 0.00000$). The other problem is of the opposite kind. Because this analysis does not factor out the large between plot variation at the larger scale, the error variance for testing small

plot effects is much too big. In the correct, split plot analysis, the small-plot error variance is very small (0.068) compared with the pseudoreplicated, small-plot error variance of 0.32 (above).

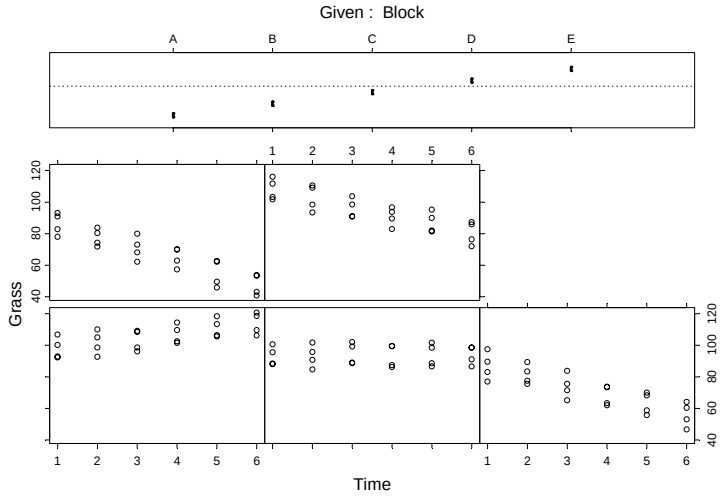
Mixed Effects Models (you can skip this section if you have had enough already)

This example is like the last one except now we have introduced temporal pseudoreplication (repeated measurements) as well as spatial pseudoreplication (different plot sizes) in a split plot experiment. The name “mixed effects” means that we have a mixture of fixed effects (experimental treatments) and random effects (blocks and time in this case). There are 5 replicates and 2 experimental treatments (insect exclusion and mollusc exclusion). Data were gathered from 6 successive harvests on each plot.

```
repeated<-read.table("c:\\temp\\repeated.txt",header=T)
attach(repeated)
names(repeated)
[1] "Block"   "Time"    "Insect"  "Mollusc" "Grass"
```

This is what the data look like (**coplot** draws Grass as a function of Time, *given* Block)

```
coplot(Grass ~ Time | Block)
```



Here is the most obvious way of doing the **wrong** analysis. Just fit Block and the 2 x 2 factorial (Insect*Mollusc) to the whole data set:

```
model<-aov(Grass~Insect*Mollusc+Error(Block))
```

summary(model)

```
Error: Block

          Df  Sum Sq Mean Sq F value Pr(>F)
Residuals  4 25905.9  6476.5

Error: Within
                Df Sum Sq Mean Sq F value     Pr(>F)
Insect           1 3826.1  3826.1 43.9838 1.218e-009 ***
Mollusc          1  314.6   314.6  3.6169    0.05976 .
Insect:Mollusc   1    0.1     0.1  0.0010    0.97429
Residuals      112 9742.6    87.0
```

The conclusion is clear: there is no interaction, but insect exclusion has a highly significant main effect on grass yield and mollusc exclusion has a close-to-significant effect. However, you can see at once that the analysis is wrong, because there are far too many degrees of freedom for error (d.f. = 112). There are only 5 replicates of the experiment, so this is clearly not right. Both effects are in the expected direction (herbivore exclusion increases grass yield).

tapply(Grass,list(Insect,Mollusc), mean)

```
               Absent     Present
  Sprayed      93.71961  90.42613
Unsprayed      82.37145  79.18800
```

A sensible way to see what is going on is to do a regression of Grass against Time separately for each treatment in each block, to check whether there are temporal trends, and if so, whether the temporal trends are the same in each block. This is an analysis of covariance (see Practical 6) with different slopes (Time) estimated for every combination of Block:Insect:Mollusc

model<-aov(Grass~Block*Insect*Mollusc*Time)

summary(model)

```
                          Df Sum of Sq  Mean Sq  F Value     Pr(F)
                Block      4  25905.93 6476.483 2288.045 0.0000000
               Insect      1   3826.05 3826.051 1351.687 0.0000000
              Mollusc      1    314.63  314.629  111.154 0.0000000
                 Time      1   3555.40 3555.400 1256.070 0.0000000
        Block:Insect      4      1.59    0.397    0.140 0.9667569
       Block:Mollusc      4     10.74    2.685    0.949 0.4404798
      Insect:Mollusc      1      0.09    0.091    0.032 0.8583185
          Block:Time      4   5898.01 1474.502  520.920 0.0000000
         Insect:Time      1      0.39    0.389    0.137 0.7118010
        Mollusc:Time      1     13.90   13.900    4.911 0.0295350
Block:Insect:Mollusc      4     11.21    2.802    0.990 0.4179119
   Block:Insect:Time      4      3.65    0.914    0.323 0.8619207
  Block:Mollusc:Time      4      2.33    0.583    0.206 0.9343920
 Insect:Mollusc:Time      1     16.38   16.381    5.787 0.0184497
Block:Insect:Mollusc:Time 4      2.58    0.644    0.227 0.9222823
           Residuals     80    226.45    2.831
```

This shows some interesting features of the data. Starting with the highest order interaction and working upwards, we see that there is a significant interaction between Insect, Mollusc and Time. That is to say, the slope of the graph of grass against time is significantly different for different combinations of insecticide and molluscicide. The effect is not massively significant (p = 0.018) but it *is* there (another component of this effect appears as a Mollusc by Time interaction). There is an extremely significant interaction between Block and Time, with different slopes in different Blocks (as we saw in the **coplot**, earlier). At this stage, we need to consider whether the suggestion of an Insect by Mollusc by Time interaction is worth following up. It is plausible that insects had more impact on grass yield at one time of year and molluscs at another. The problem is that this analysis does not take account of the fact that the measurements through time are correlated because they were taken from the same location (a plot within a block).

One simple way around this is to carry out separate analyses of variance for each time period. We use the **subset** directive to restrict the analysis, and put the whole thing in a loop for time i = 1 to 6:

```
for (i in 1:6 ) print(summary(aov(Grass~Insect*Mollusc+Error(Block),subset=(Time==i))))
```

```
Error: Block
          Df  Sum Sq Mean Sq F value Pr(>F)
Residuals  4 1321.30  330.32

Error: Within
               Df Sum Sq Mean Sq  F value     Pr(>F)
Insect          1 652.50  652.50 257.4725 1.794e-009 ***
Mollusc         1  77.76   77.76  30.6823  0.0001280 ***
Insect:Mollusc  1   8.47    8.47   3.3435  0.0924206 .
Residuals      12  30.41    2.53
```

```
Error: Block
          Df  Sum Sq Mean Sq F value Pr(>F)
Residuals  4 2119.29  529.82

Error: Within
               Df Sum Sq Mean Sq  F value     Pr(>F)
Insect          1 620.38  620.38 253.1254 1.978e-009 ***
Mollusc         1  94.88   94.88  38.7122 4.438e-005 ***
Insect:Mollusc  1   0.01    0.01   0.0034     0.9545
Residuals      12  29.41    2.45

Error: Block
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  4 3308.8   827.2

Error: Within
               Df Sum Sq Mean Sq  F value     Pr(>F)
Insect          1 628.26  628.26 183.8736 1.225e-008 ***
Mollusc         1  71.41   71.41  20.9002  0.0006416 ***
Insect:Mollusc  1   4.88    4.88   1.4273  0.2552897
Residuals      12  41.00    3.42

Error: Block
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  4 5063.6  1265.9

Error: Within
               Df Sum Sq Mean Sq  F value     Pr(>F)
Insect          1 550.43  550.43 194.9970 8.784e-009 ***
Mollusc         1  26.50   26.50   9.3878   0.009827 **
Insect:Mollusc  1   3.63    3.63   1.2876   0.278649
Residuals      12  33.87    2.82

Error: Block
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  4 8327.2  2081.8

Error: Within
               Df Sum Sq Mean Sq  F value     Pr(>F)
Insect          1 713.18  713.18 301.8366 7.168e-010 ***
Mollusc         1  30.45   30.45  12.8863   0.003715 **
Insect:Mollusc  1   1.09    1.09   0.4625   0.509353
Residuals      12  28.35    2.36
Error: Block
          Df  Sum Sq Mean Sq F value Pr(>F)
Residuals  4 11710.4  2927.6

Error: Within
               Df Sum Sq Mean Sq  F value     Pr(>F)
Insect          1 667.19  667.19 459.7610 6.178e-011 ***
Mollusc         1  33.34   33.34  22.9714  0.0004388 ***
Insect:Mollusc  1  13.04   13.04   8.9868  0.0111120 *
Residuals      12  17.41    1.45
```

Each one of these tables has the correct degrees of freedom for error (d.f. = 12) and they all tell a reasonably consistent story. All show significant main effects for Insect and Mollusc, Insect is generally more significant than Mollusc, and there was a significant interaction only at Time 6 (this is the Insect:Mollusc:Time interaction we discovered earlier).

The simplest way to get rid of the temporal pseudoreplication is just to average it away. That would produce a single number per treatment per block, or 20 numbers in all. We need to produce new, shorted vectors (length 20 instead of length 120) for each of the variables. The shorter vectors will all need new names: let's define shorter Block as b, Insect as i and Mollusc as m, like this

```
b<-Block[Time==1]
i<-Insect[Time==1]
m<-Mollusc[Time==1]
```

What values should go in the shorter vector g for Grass ? They should be the grass values for block, insect and mollusc, averaged over the 6 values of Time. We need to be very careful here, because the averages need to be in exactly the same order as the subscripts in the new explanatory variables we have created. Let's look at their values:

```
b
```

```
 [1] A A A A B B B B C C C C D D D D E E E E
```

```
i
```

```
[1] Sprayed   Sprayed   Unsprayed Unsprayed Sprayed   Sprayed   Unsprayed Unsprayed
Sprayed   Sprayed
[11] Unsprayed Unsprayed Sprayed   Sprayed   Unsprayed Unsprayed Sprayed   Sprayed
Unsprayed Unsprayed
```

```
m
```

```
[1] Present Absent  Present Absent  Present Absent  Present Absent  Present Absent
Present Absent  Present
[14] Absent  Present Absent  Present Absent  Present Absent
```

The first 4 numbers in our array of means need to come from Block A, the first 2 from insecticide Sprayed and the mollusc numbers to alternate Present, Absent, Present, Absent, etc. This means that the list in the **tapply** directive must be in the order Mollusc, Insect, Block. The most rapidly changing subscripts are first in the list, the most slowly changing last. So the shorted vector of Grass weights, *g*, is computed like this:

```
g<-as.vector(tapply(Grass,list(Mollusc,Insect,Block),mean))
```

The **as.vector** part of the expression is necessary to make *g* into a column vector of length 20 rather than a 3-D matrix 2 x 2 x 5.

Now we can carry out the ANOVA fitting block and the 2 x 2 factorial:

```
model<-aov(g~i*m+Error(b))
summary(model)
```

```
Error: b
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  4 4317.7  1079.4

Error: Within
          Df Sum Sq Mean Sq   F value     Pr(>F)
i          1 637.68  637.68 1950.6284 1.177e-014 ***
m          1  52.44   52.44  160.4069 2.645e-008 ***
i:m        1   0.02    0.02    0.0463     0.8333
Residuals 12   3.92    0.33
```

Now we have the right number of degrees of freedom for error (d.f. = 12). The analysis shows highly significant main effects for both treatments, but no interaction (recall that it was only significant at Time = 6). Note that the error variance $s^2 = 0.327$ is larger than s^2 was in 5 of the 6 separate analyses, because of the large Block:Time interaction.

An alternative is to detrend the data by regressing Grass against Time for each treatment combination (Code = 1 to 20), then use the slopes and the intercepts of these regressions in 2 separate ANOVA's in what is called a *derived variables analysis*.

Here is one way to extract the 20 regression intercepts, *a*, which are parameter [1] within the **coef** of the fitted model:

```
a<-1:20
Code<-gl(20,6)
for (k in 1:20) a[k]<-coef(lm(Grass~Time,subset=(Code==k)))[1]

a
```

```
[1]  97.27997 103.22584  87.71079  90.27373  95.41965
[6] 101.91840  87.22158  88.60817  94.41061 103.25222
[11]  85.07103  89.28192  96.10313 100.96869  86.18559
[16]  91.19490 116.83651 121.69893 106.40919 108.55959
```

This shortened vector now becomes the response variable in a simple analysis of variance, using the shortened factors we calculated earlier:

```
model<-aov(a~i*m+Error(b))
summary(model)
```

```
Error: b
          Df  Sum Sq Mean Sq F value Pr(>F)
Residuals  4 1253.30  313.33

Error: Within
          Df Sum Sq Mean Sq F value     Pr(>F)
i          1 611.59  611.59 594.933 1.359e-011 ***
m          1 107.34  107.34 104.420 2.834e-007 ***
i:m        1  12.32   12.32  11.980   0.004707 **
Residuals 12  12.34    1.03
```

Everything in the model has a significant effect on the intercept, including a highly significant interaction between Insects and Molluscs (p = 0.0047). What about the slopes of the Grass against Time graphs; **coef** [2] of the fitted object ?

```
aa<-1:20
for (k in 1:20) aa[k]<-coef(lm(Grass~Time,subset=(Code==k)))[2]

aa
 [1]  3.3885561  2.8508475  3.2423677  3.1764151
 [5]  0.6762409 -0.3742587 -0.1142285  0.1536275
 [9] -5.5199901 -6.7429909 -6.1245833 -6.1412692
[13] -6.9687946 -7.7624054 -7.6693108 -7.9243495
[17] -5.2672184 -5.8186351 -5.5222919 -5.2818614
```

As we saw from the initial graphs, there is great variation in slope from Block to Block, but is there any effect of treatments ?

```
model<-aov(aa~i*m+Error(b))
summary(model)


Error: b
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  4 337.03   84.26

Error: Within
          Df  Sum Sq Mean Sq F value    Pr(>F)
i          1 0.02223 0.02223  0.5453 0.4744357
m          1 0.79426 0.79426 19.4805 0.0008449 ***
i:m        1 0.93608 0.93608 22.9588 0.0004398 ***
Residuals 12 0.48927 0.04077
```

Yes there is. Perhaps surprisingly, there is a highly significant interaction between Insect and Mollusc in their effects on the slope of the graph of Grass biomass against Time. We need to see the values involved:

```
tapply(aa,list(m,i),mean)
```

```
                Sprayed  Unsprayed
 Absent        -3.569489  -3.203488
Present        -2.738241  -3.237609
```

All the average slopes are negative, but insect exclusion reduces the slope under one mollusc treatment but increases it under another. The mechanisms underlying this interaction would clearly repay further investigation.

The statistical lesson is that derived variable analysis provided many more insights than either the 6 separate analyses or the analysis of the time-averaged data.

The final technique we shall use is *mixed effects* modelling. This requires a clear understanding of which of our 4 factors are **fixed effects** and which are **random effects**. We applied the insecticide and the molluscicide to plots within Blocks at random, so Mollusc and Insect are *fixed* effects. Blocks are different, but we didn't make them different. They are assumed to come from a population of different locations, and hence are random effects. Things differ through time, but mainly because of the weather, and the passage of the seasons, rather than because of anything we do. So time, as well, is a *random* effect. One of our random effects is **spatial** (Block) and one **temporal** (Time). The factorial experiment (Mollusc*Insect) is nested within each block, and Time codes for repeated measures made within each treatment plot within each Block.

It is worth re-emphasising that if the smallest plot size are pseudoreplicates (rather than small-plot fixed effect treatments), then they should **not** be included as the last term in the Error directive. See the rat's liver example (above), where we did not include Preparations (the measurement error) in the error formula (and it would be a mistake to do so).

Mixed effects models

Linear mixed effects models and the groupedData directive

If you have not read the data file repeated.txt earlier, you should input it now:

```
library(nlme)

repeated<-read.table("c:\\temp\\repeated.txt",header=T)
attach(repeated)
names(repeated)
```

```
[1] "Block"   "Time"    "Insect"  "Mollusc" "Grass"
```

We need to turn this data frame into grouped data. The key thing to understand is where, exactly, each of the 5 variables (above) is placed in a mixed effects model. We create a

formula of the form resp ~ cov | group where resp is the response variable, cov is the primary covariate, and group is the grouping factor.

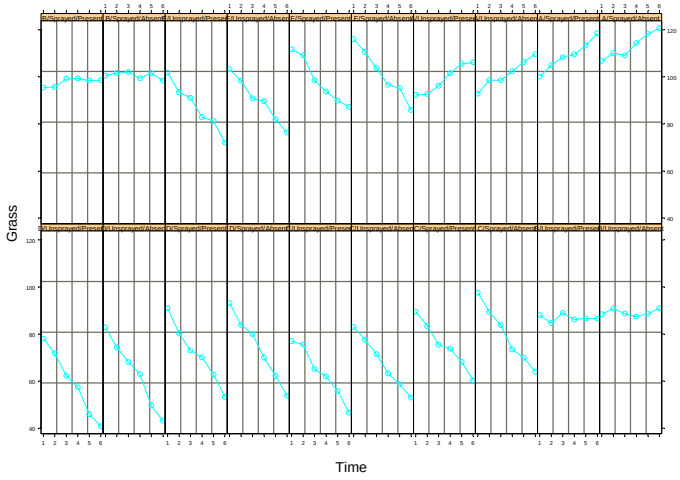
The response variable is easy: it is Grass. This goes on the left of the ~ in the model formula. The manipulated parts of the experiment (the fixed effects) are Insect and Mollusc; 2-level factors representing plus and minus each kind of herbivore. These factors that apply to the whole experiment are called the **outer** factors. They appear in a model formula (linked in this case by the factorial operator *) to the right of the ~ in the outer statement. But what about Block and Time? You can easily imagine a graph of Grass against Time. This is a time series plot, and each treatment in each Block could produce one such time series. Thus, Time goes on the right of the ~ in the model formula. Block is the spatial unit in which the pseudoreplication has occurred, so it is the factor that goes on the right of the conditioning operator | (vertical bar) after the model formula. In general, the conditioning factors can be nested, using the / (slash operator). In medical studies the conditioning factor is often the Subject from which repeated measures were taken.

The new thing here is the **groupedData** directive. It is constructed like this to create a new data frame called nash:

```
nash<-groupedData(Grass~Time|Block/Insect/Mollusc,data=repeated,outer=~Insect*Mollusc)
```

With grouped data, some previously complicated things become very simple. For instance, panel plots are produced automatically.

```
plot(nash)
```



It shows the data for the whole factorial experiment (2 levels of Insecticide and 2 levels of Molluscicide) in a single group of panels (one for each Block). What it shows very clearly, however, are the different time series exhibited in the 5 different Blocks. Note that **groupedData** has ordered the blocks from lowest to highest mean grass biomass (D, C, B, A & E) and that grass biomass increases through time on Block E, is roughly constant on Block B and declines on Blocks D, C and A. To see the time series for the 4 combinations of fixed effects, we use the outer=T option within the plot directive:

```
mixed<-lme(fixed=Grass~Insect*Mollusc,data=nash,random=~Time|Block)

summary(mixed)
```

```
Linear mixed-effects model fit by REML
 Data: nash
       AIC      BIC    logLik
  538.3928 560.4215 -261.1964

Random effects:
 Formula:  ~ Time | Block
 Structure: General positive-definite
              StdDev   Corr
(Intercept) 8.885701 (Inter
       Time 5.193374 -0.261
   Residual 1.644065

Fixed effects: Grass ~ Insect * Mollusc
                   Value Std.Error  DF   t-value p-value
   (Intercept)  96.14375  3.848244 112  24.98380  <.0001
        Insect  -5.64657  0.150082 112 -37.62326  <.0001
       Mollusc  -1.61923  0.150082 112 -10.78899  <.0001
Insect:Mollusc   0.02751  0.150082 112   0.18327  0.8549
 Correlation:
               (Intr) Insect Mollsc
        Insect 0
       Mollusc 0      0
Insect:Mollusc 0      0      0

Standardized Within-Group Residuals:
       Min         Q1         Med        Q3       Max
 -1.877155 -0.8299427 -0.01336901 0.7337527 2.253834

Number of Observations: 120
Number of Groups: 5
```

The interpretation is unequivocal: the mixed effects model gives no indication of an interaction between insect and mollusc exclusion ($p = 0.8549$). The effects we saw in the earlier analyses were confounded with block effects.

Contrasts: Single degree of freedom comparisons

Once the anova table has been completed and the F test carried out to establish that there are indeed significant differences between the means, it is reasonable to ask which factor levels are significantly different from which others. This subject is developed in more detail in the chapter on Multiple Comparisons (see Statistical computing p. 274); here we introduce the technique of contrasts (also known as single degree of freedom comparisons). There are two sorts of contrasts we might want to carry out

- contrasts we had planned to carry out at the experimental design stage (these are referred to as *a priori* contrasts)
- contrasts that look interesting after we have seen the results (these are referred to as *a posteriori* contrasts)

Some people are very snooty about *a posteriori* contrasts, but you can't change human nature. The key point is that you do contrasts *after* the anova has established that there really are significant differences to be investigated. It is not good practice to carry out tests to compare the largest mean with the smallest mean if the anova fails to reject the null hypothesis (tempting though this may be).

Contrasts are used to compare means or groups of means with other means or groups of means. In a drug trial classified by cities, for example, you might want to compare the response to the drugs for all mid west cities with all west coast cities. There are two important points to understand about contrasts:

- there are absolutely loads of possible contrasts
- there are only k-1 orthogonal contrasts

Lets take a simple example. Suppose we have one factor with 5 levels and the factor levels are called *a, b, c, d,* and *e*. Let's start writing down the possible contrasts. Obviously we could compare each mean singly with every other:

a vs. b, a vs. c, a vs. d, a vs. e, b vs. c, b vs. d, b vs. e, c vs. d, c vs. e, d vs. e

but we could also compare pairs of means:

{a,b} vs. {c,d}, {a,b} vs. {c,e}, {a,b} vs. {d, e}, {a,c} vs. {b, d}, {a,c} vs. {b,e}, etc.

or triplets of means:

{a,b,c} vs. d, {a,b,c} vs. e, {a,b,d} vs. c, {a,b,d} vs. e, {a,c,d} vs. b, and so on.

or groups of four means

{a,b,c,d} vs. e, {a,b,c,e} vs. d, {b,c,d,e} vs. a, {a,b,d,e} vs c, {a,b,c,e} vs d

I think you get the idea. There are absolutely loads of possible contrasts.

Orthogonal contrasts are different, and it is important that you understand how and why they are different. We refer to the number of factor levels as *k* (this was 5 in the last example). Out of all the many possible contrasts, only *k*-1 of them are orthogonal. In this context, orthogonal means "statistically independent" (it also means "at right angles" which in mathematical terms is another way of saying statistically independent). In practice we should only compare things once, either directly or implicitly. So the two contrasts:

a vs. b and *a vs. c*

implicitly contrasts *b vs. c*. This means that if we have carried out the two contrasts *a vs. b* and *a vs. c* then the third contrast *b vs. c* is **not** an orthogonal contrast. Which particular contrasts are orthogonal depends very much on your choice of the first contrast to make. Suppose there were good reasons for comparing *{a,b,c,e} vs. d*. For example, *d* might be the placebo and the other 4 might be different kinds of drug treatment, so we make this our first contrast. Because *k*-1 = 4 we only have 3 possible contrasts that are orthogonal to this. There may be *a priori* reasons to group *{a,b}* and *{c,e}* so we make this our second orthogonal contrast. This means that we have no degrees of freedom in choosing the last 2 orthogonal contrasts: they have to be *a vs. b* and *c vs. e*.

Just remember that with orthogonal contrasts you only compare things once.

Contrast coefficients

Rules for constructing contrast coefficients are straightforward:

- treatments to be lumped together get like sign (plus or minus)
- groups of means to be to be contrasted get opposite sign
- factor levels to be excluded get a contrast coefficient of 0
- the contrast coefficients, *c*, must add up to 0

Suppose that with our 5-level factor {*a,b,c,d,e*} we want to begin by comparing the 4 levels {*a,b,c,e*} with the single level *d*. All levels enter the contrast, so none of the coefficients is 0. The four terms {*a,b,c,e*} are grouped together so they all get the same sign (minus, for example, although it makes not matter which sign is chosen). They are to be compared to *d*, so it gets the opposite sign (plus, in this case). The choice of what numeric values to give the contrast coefficients is entirely up to you. Most people use whole numbers rather than fractions, but it really doesn't matter. All that matters is that the c's sum to 0. The positive and negative coefficients have to add up to the same value. In our example, comparing 4 means with one mean, a natural choice of coefficients would be -1 for each of {*a,b,c,e*} and +4 for *d*. Alternatively with could have selected +0.25 for each of {*a,b,c,e*} and -1 for *d*. It really doesn't matter.

factor level:	*a*	*b*	*c*	*d*	*e*
contrast 1 coefficients, c:	-1	-1	-1	4	-1

Suppose the second contrast is to compare {*a*,*b*} with {*c*,*e*}. Because this contrast excludes *d*, we set its contrast coefficient to 0. {*a*,*b*} get the same sign (say, plus) and {*c*,*e*} get the opposite sign. Because the number of levels on each side of the contrast is equal (2 in both cases) we can use the name numeric value for all the coefficients. The value 1 is the most obvious choice (but you could use 13.7 if you wanted to be perverse).

factor level:	*a*	*b*	*c*	*d*	*e*
contrast 2 coefficients, c:	1	1	-1	0	-1

There are only 2 possibilities for the remaining orthogonal contrasts: *a vs. b* and *c vs. e*:

factor level:	*a*	*b*	*c*	*d*	*e*
contrast 3 coefficients, c:	1	-1	0	0	0
contrast 4 coefficients, c:	0	0	1	0	-1

The key point to understand is that the treatment sum of squares SSA is the sum of the k-1 orthogonal sums of squares. It is useful to know which of the contrasts contributes most to SSA, and to work this out, we compute the contrast sum of squares SSC as follows:

$$SSC = \frac{\left(\sum \frac{c_i T_i}{n_i}\right)^2}{\sum \frac{c_i^2}{n_i}}$$

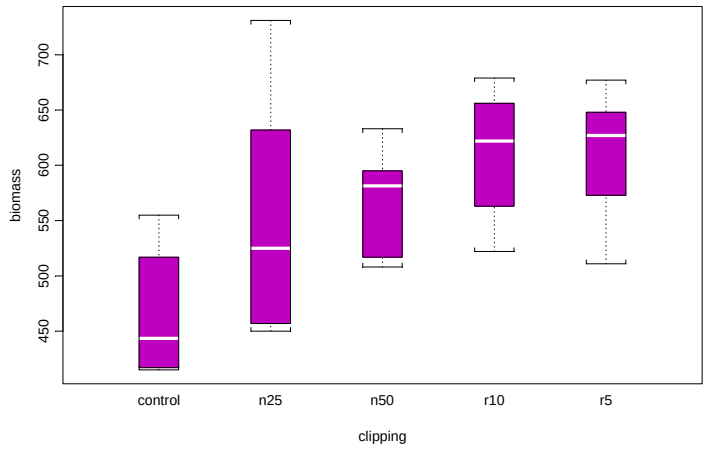
The significance of a contrast is judged in the usual way by carrying out an F test to compare the contrast variance with the error variance, s^2. Since all contrasts have a single degree of freedom, the contrast variance is equal to SSC, so the F test is just

$$F = \frac{SSC}{s^2}$$

The contrast is significant (i.e. the two contrasted groups have significantly different means) if the calculated value is larger than the value of F in tables with 1 and $k(n-1)$ degrees of freedom.

An example should make all this clearer. Suppose we have a plant ecology experiment with 5 treatments: a control, a shoot-clipping treatment where 25% of the neighbouring plants are clipped back to ground level, another where 50% on neighbours are clipped back, a 4th where a circle of roots around the target plant is pruned to a depth of 10cm and a 5th with root pruning to only 5cm depth. The data look like this:

```
compexpt<-read.table("c:\\temp\\compexpt.txt",header=T)
attach(compexpt)
names(compexpt)

[1] "biomass"  "clipping"

levels(clipping)

[1] "control" "n25"     "n50"     "r10"     "r5"

tapply(biomass,clipping,mean)

  control       n25       n50       r10     r5
 465.1667 553.3333 569.3333 610.6667 610.5

plot(clipping,biomass)
```



The treated plant means are all higher than the controls, suggesting that competition was important in this system. The root pruned plants were larger than the shoot pruned plants, suggesting that below ground competition might be more influential than above ground. The different intensities of shoot pruning differed more than the different intensities of root pruning. It remains to see whether these differences are significant by using contrasts. We can do this long-hand by recoding the factor levels, or make use of SPlus built in facilities for defining contrasts.

We begin by re-coding the factor levels to calculate the contrast sums of squares. First, the overall anova to see whether there are any significant differences at all

```
model1<-aov(biomass~clipping)

summary(model1)
```

```
            Df Sum of Sq  Mean Sq  F Value       Pr(F)
 clipping    4   85356.5 21339.12 4.301536 0.008751641
Residuals   25  124020.3  4960.81
```

Yes. There are highly significant differences ($p < 0.01$). We reject the null hypothesis that all the means are the same, and accept the alternative hypothesis that at least one of the means is significantly different from the others. This is not a particularly informative alternative hypothesis, however. We want to know what, exactly, the experiment has shown.

The treatment sum of squares, SSA = 85356.6 is made up by k-1 = 4 orthogonal contrasts. The most obvious contrast to try first is the control versus the rest. We compute a new factor, c1, to reflect this. It has value 1 for all the competition treatments and 2 for the controls:

```
c1<-factor(1+(clipping=="control"))
```

Now we fit this as the single explanatory variable in a one-way anova:

```
model2<-aov(biomass~c1)
summary(model2)
```

```
            Df Sum of Sq  Mean Sq  F Value        Pr(F)
         c1  1   70035.0 70035.01 14.07317 0.0008149398
Residuals 28  139341.8  4976.49
```

This contrast explained 70035.0 out of the total SSA = 85356.5, so it is clear that competition was highly significant. What about the difference between defoliation and root pruning (light versus below-ground competition). The computing is a little more complicated in this case because be need to **weight out** the control individuals from the analysis as well as calculating a new factor to compare “n25” and “n50” with “r10” and “r5”. Note the use of “!=” for “not equal to” in the weight directive:

```
c2<-factor(1+(clipping=="r10")+(clipping=="r5"))
model3<-aov(biomass~c2,weight=(clipping!="control"))
summary(model3)
```

```
            Df Sum of Sq  Mean Sq  F Value      Pr(F)
         c2  1   14553.4 14553.38 2.959217 0.09942696
Residuals 22  108195.6  4917.98
```

This contrast with SSC = 14553.4 is not significant. Between them, the first two contrasts explain 14553.4 + 70035 = 84588.4 out of the total explained variation of SSA = 85356.5. The other two untested (but obviously not significant) orthogonal contrasts are "n25" vs. "n50" and "r10" vs "r5".

There are sophisticated built-in functions in SPlus for defining and executing contrasts. The contrast coefficients are specified as attributes of the factor called clipping like this

```
contrasts(clipping)<-cbind(c(4,-1,-1,-1,-1),c(0,1,1,-1,-1),c(0,0,0,1,-1),c(0,-1,1,0,0))
```

where we bind the relevant contrast vectors together using **cbind**. To inspect the contrasts association with any factor we just type:

```
contrasts(clipping)
```

```
        [,1] [,2] [,3] [,4]
control    4    0    0    0
    n25   -1    1    0   -1
    n50   -1    1    0    1
    r10   -1   -1    1    0
     r5   -1   -1   -1    0
```

Notice that all of the column totals sum to zero as required ($\sum c_i = 0$, see above). You can see that the different contrasts are all **orthogonal** because *the products of their coefficients all sum to zero*. For example, comparing contrasts 1 and 2 we have

$$(4\times 0)+(-1\times 1)+(-1\times 1)+(-1\times -1)+(-1\times -1) \;=\; -1+-1+1+1 \;=\; 0$$

Now when we carry out a one way anova using the factor called clipping, the parameter estimates reflect the differences between the contrasted group means rather than differences between the individual treatment means.

```
model<-aov(biomass~clipping)
```

We use the function **summary.lm** to obtain a listing of the coefficients for the 4 contrasts we have specified:

```
summary.lm(model)
```

Coefficients:

```
                Value Std. Error   t value  Pr(>|t|)
(Intercept)  561.8000   12.8593    43.6884    0.0000
  clipping1  -24.1583    6.4296    -3.7573    0.0009
  clipping2  -24.6250   14.3771    -1.7128    0.0991
  clipping3    0.0833   20.3323     0.0041    0.9968
  clipping4    8.0000   20.3323     0.3935    0.6973
```

This gives all the information we need. The first row shows the overall mean biomass (561.8). The second row shows clipping contrast 1: the overall mean versus the 4 competition levels (585.958). The overall mean was 24.1583 g below the mean of the 4 competition treatments (561.8 – 585.958 = -24.1583), and this difference is highly significant (p = 0.0009). The 3rd row shows the effect of the 2nd contrast between clipped and root-pruned plants. This contrast is not significant (p = 0.0991) despite the fact that the clipped plants produced an average biomass of about 49.25 g less than the root pruned plants (the coefficient (-24.625) is *half* of the difference between the 2 groups of means (561.333 – 610.585) because all the contrasts are calculated relative to overall means (in this case the mean of {a,b,c,e} excluding the controls, 585.96). There is no hint of any difference between the different intensities of root pruning 610.67 – 610.585 = 0.0833 (p = 0.9968) or shoot clipping 569.333 – 561.333 = 8.0 (p = 0.6973), despite the fact that the means differ by 16.0 g (569.333 – 553.333 = 16).

You may have noticed that the standard errors are different in four of the 5 rows of the coefficients table. All the standard errors are based on the same pooled error variance of $s^2 = 4960.813$. For the grand mean, based on 30 samples, the standard error is $\sqrt{\frac{4960.813}{30}} = 12..859$. The first contrast (row 2) compares a group of 4 means with the overall mean (a comparison based notionally on 30 x 4 = 120 numbers) so the standard error is $\sqrt{\frac{4960.813}{120}} = 6.4296$. The second contrast (row 3) compares the two defoliation means with the two root pruning means (a total of 24 numbers) so the standard error is $\sqrt{\frac{4960.813}{24}} = 14.3771$. The third and fourth contrasts (rows 4 & 5) both involve the comparison of one mean with another (a comparison based on 6 + 6 = 12 numbers) and therefore they have the same standard error $\sqrt{\frac{4960.813}{12}} = 20.3323$.

The contrasts that are built into SPlus are Helmert, Sum and Treatment, and we should compare the output produced by each. First we need to remove our own contrasts from the factor called clipping:

```
contrasts(clipping)<-NULL
```

Helmert contrasts

These are the default option in SPlus (but not in R) and are preferred because they represent orthogonal contrasts (the columns of the contrast matrix sum to 0).

```
options(contrasts=c("contr.helmert","contr.poly"))
model2<-lm(biomass~clipping)
summary(model2)
```

Note that each of the standard errors is different, because one mean is being compared with an increasingly larger group of means (so replication is higher, and the standard error is correspondingly lower, as explained below).

```
Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept) 561.8000  12.8593    43.6884   0.0000
  clipping1  44.0833  20.3323     2.1681   0.0399
  clipping2  20.0278  11.7388     1.7061   0.1004
  clipping3  20.3472   8.3006     2.4513   0.0216
  clipping4  12.1750   6.4296     1.8936   0.0699
```

Now you are not going to like this, but we are going to understand exactly where each of the coefficients comes from. It will be useful to remind ourselves of the overall mean (561.8) and the 5 treatment means to begin with:

```
tapply(biomass,clipping,mean)

  control       n25       n50       r10     r5
 465.1667 553.3333 569.3333 610.6667 610.5
```

The first parameter (labelled Intercept) is the overall mean. That's the easy part. The second term (labelled clipping1) is the difference between the mean of clipping treatment 1 (465.1667) and the *average of the means* of treatments 1 and 2: (465.1667+553.333)/2 = 509.25.

```
509.25 - 465.1667
[1] 44.0833
```

The third term is the difference between the *average of the first 3* clipping treatments (465.1667 + 553.333 + 569.333)/3 = 529.2778) and the *average of the first two* means (already calculated as 509.25):

```
529.2778 - 509.25
[1] 20.0278
```

Are you getting this? The 4th term is the *average of the first 4* clipping treatments (465.1667 + 553.333 + 569.333 + 610.667) / 4 = 549.525) minus the *average of the first 3* (529.2778, above)

```
549.625 - 529.2778
[1] 20.3472
```

The last coefficient in table is the difference between the *overall mean* (561.8) and the *mean of the first 4* treatments (just calculated, above):

```
561.8 - 549.625
[1] 12.175
```

I hope you understood all that. Statisticians like Helmert contrasts because they are *proper* contrasts: the columns of the contrast matrix sum to zero. No one else likes them, because they are a pain in the neck.

Sum contrasts

```
options(contrasts=c("contr.sum","contr.poly"))
model3<-lm(biomass~clipping)
summary(model3)
```

```
Coefficients:
                Value Std. Error   t value  Pr(>|t|)
(Intercept)  561.8000   12.8593    43.6884    0.0000
  clipping1  -96.6333   25.7185    -3.7573    0.0009
  clipping2   -8.4667   25.7185    -0.3292    0.7447
  clipping3    7.5333   25.7185     0.2929    0.7720
  clipping4   48.8667   25.7185     1.9001    0.0690
```

As with Helmert contrasts, the intercept is the overall mean. The second row, clipping contrast 1, is the difference between the grand mean and the mean of the controls: 561.8 – 96.6333 = 465.1667. The third row is the difference between the grand mean and the mean of clipping treatment 2: 561.8 – 8.4667 = 553.333. And so on. The standard error of the intercept is the same as with Helmert contrasts: $\sqrt{s^2 / 30}$. All the other standard errors are the same $\sqrt{\frac{4960.813}{12}+\frac{4960.813}{20}}$ = 25.7185

Treatment contrasts

```
options(contrasts=c("contr.treatment","contr.poly"))
model<-lm(biomass~clipping)
summary(model)
```

These are the default contrasts in R and in GLIM. The first coefficient is *the mean of the factor level that comes first in alphabetical order* (the control mean in this example). The *remaining parameters are all differences between means* (the mean in question compared with the control mean). The standard error of the intercept is the standard error of one treatment mean. It is based on 6 replicates, so the standard error is $\sqrt{\frac{s^2}{6}}$ =28.7542 . The remaining parameters are differences between two means, so their standard errors are $\sqrt{2\frac{s^2}{6}}$ = 40.6645.

```
Coefficients:
              Value Std. Error  t value Pr(>|t|)
(Intercept) 465.1667  28.7542    16.1774   0.0000
```

```
clippingn25  88.1667  40.6645     2.1681   0.0399
clippingn50 104.1667  40.6645     2.5616   0.0168
clippingr10 145.5000  40.6645     3.5781   0.0015
 clippingr5 145.3333  40.6645     3.5740   0.0015
```

Obviously, you like what you are used to. But I find these treatment contrasts much more intuitive and much more useful than sum and Helmert contrasts. I can compare any means with any other directly. For example r10 and r5 differ by only 0.1666 (you do the subtraction in your head) and this has a standard error of a difference which is 40.66. Say no more. Neighbour clipping to 25 and 50 cm differ by 16.0 with the same standard error. Not significant. And so on. Easy and intuitive. The downside is that the probabilities in the coefficients table don't refer directly to the need to retain a particular parameter in the model (they refer to the significance of the comparison between that level and level 1, the intercept). To summarise, here is a table comparing the standard error terms used in our own contrast and in the 3 built-in contrasts:

Parameter	Ours	Helm.	Sum	Treat.	Standard error
Grand mean	*	*	*		$\sqrt{\frac{4960.813}{30}} = 12..859$
One treatment mean				*	$\sqrt{\frac{4960.813}{6}} = 28.754$
Difference between 2 means				*	$\sqrt{2 \times \frac{4960.813}{6}} = 40.665$
Averaged over 2 treatments, n = 6	*	*			$\sqrt{\frac{4960.813}{12}} = 20.3323$
Averaged over 4 treatments, n = 6	*				$\sqrt{\frac{4960.813}{24}} = 14.3771$
Averaged over 6 treatments, n = 6		*			$\sqrt{\frac{4960.813}{36}} = 11.7388$
Averaged over 12 treatments, n = 6		*			$\sqrt{\frac{4960.813}{72}} = 8.3006$
Averaged over 20 treatments, n = 6	*	*			$\sqrt{\frac{4960.813}{120}} = 6.4296$
Sums			*		$\sqrt{\frac{4960.813}{12} + \frac{4960.813}{20}}$

Contrasts in analysis of covariance

Remember that the minimal adequate model has *a common slope* and *two different intercepts* for the relationship between fruit production and initial rootstock size.

The important point to understand is that using different contrasts gives you

- different parameter values
- different standard errors

so you need to be extremely careful that you know what is going on. It is safest to use the same form of contrasts in all of your work, then you won't be misled. The default in S-Plus is Helmert contrasts and the default in R is Treatment contrasts. Old GLIM users will be familiar with Treatment contrasts.

1) Helmert contrasts

```
ipomopsis<-read.table("c:\\temp\\ipomopsis.txt",header=T)
attach(ipomopsis)

options(contrasts=c("contr.helmert","contr.poly"))
modelH<-lm(Fruit~Root+Grazing)
summary(modelH)
```

The Intercept is the *average of the two intercepts* (-127. 83 and –91.73). The effect of root is the slope of the graph of Fruit against Root (this is the same with all 3 contrasts). The effect of grazing is the difference (+18.0516) between the Grazed intercept (-127.83) and the average intercept (-109.777); i.e. *half the difference between the two different intercepts*.

```
Coefficients:
                Value Std. Error   t value  Pr(>|t|)
(Intercept) -109.7777    8.3182   -13.1973    0.0000
       Root   23.5600    1.1488    20.5089    0.0000
    Grazing   18.0516    1.6787    10.7533    0.0000
```

Sum contrasts

```
options(contrasts=c("contr.sum","contr.poly"))
modelS<-lm(Fruit~Root+Grazing)
summary(modelS)
```

The coefficients are exactly the same as for Helmert contrasts, except for the fact that the Grazing effect is sign-reversed: it is the difference between the Ungrazed intercept (-91.73) and the average intercept.

```
Coefficients:
                Value Std. Error   t value  Pr(>|t|)
(Intercept) -109.7777    8.3182   -13.1973    0.0000
       Root   23.5600    1.1488    20.5089    0.0000
    Grazing  -18.0516    1.6787   -10.7533    0.0000
```

Treatment contrasts

```
options(contrasts=c("contr.treatment","contr.poly"))
modelT<-lm(Fruit~Root+Grazing)
summary(modelT)
```

Coefficients:

```
                Value Std. Error   t value  Pr(>|t|)
(Intercept) -127.8294    9.6641   -13.2272    0.0000
       Root   23.5600    1.1488    20.5089    0.0000
    Grazing   36.1032    3.3574    10.7533    0.0000
```

Here the Intercept is the intercept for the factor level that comes first in the alphabet (Grazed in this case). The second parameter (Root) is the slope of the graph of Fruit against Root. The third parameter (labelled Grazing) is the difference between the 2 intercepts. The effect of grazing is to reduce fruit production, so Ungrazed plants have an intercept 36.1032 higher than Grazed plants. The standard error for Root is the *standard error of a slope* while the standard error for grazing is the *standard error of the difference between two intercepts* (see our hand calculation in Practical 6).

STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

8. MULTIPLE REGRESSION & MODEL SIMPLIFICATION

Model Simplification

The *principle of parsimony* (Occam's razor) requires that the model should be as simple as possible. This means that the model should not contain any redundant parameters or factor levels. We achieve this by fitting a maximal model then simplifying it by following one or more of these steps:

- remove non-significant interaction terms;
- remove non-significant quadratic or other non-linear terms;
- remove non-significant explanatory variables;
- group together factor levels that do not differ from one another;
- amalgamate explanatory variables that have similar parameter values;
- set non-significant slopes to zero within ANCOVA

subject, of course, to the caveats that the simplifications make good scientific sense, and do not lead to significant reductions in explanatory power.

Model	**Interpretation**
Saturated model	One parameter for every data point Fit: perfect Degrees of freedom: none Explanatory power of the model: none
Maximal model	Contains all (p) factors, interactions and covariates that might be of any interest. Many of the model's terms are likely to be insignificant Degrees of freedom: $n - p - 1$ Explanatory power of the model: it depends
Minimal adequate model	A simplified model with $0 \leq p' \leq p$ parameters Fit: less than the maximal model, but not significantly so Degrees of freedom: $n - p' - 1$ Explanatory power of the model: r^2 = SSR/SST
Null model	Just 1 parameter, the overall mean $\bar{y}$ Fit: none; SSE = SST Degrees of freedom: $n - 1$ Explanatory power of the model: none

The steps involved in model simplification

There are no hard and fast rules, but the procedure laid out below works well in practice. With large numbers of explanatory variables, and many interactions and non-linear terms, the process of model simplification can take a very long time. But this is time well spent because it reduces the risk of overlooking an important aspect of the data. It is important to realise that there is no guaranteed way of finding all the important structures in a complex data frame.

Step	Procedure	Explanation
1	Fit the maximal model	Fit all the factors, interactions and covariates of interest. Note the residual deviance. If you are using Poisson or binomial errors, check for overdispersion and rescale if necessary
2	Begin model simplification	Inspect the parameter estimates using **summary**. Remove the least significant terms first, using **update -**, starting with the highest order interactions
3	If the deletion causes an insignificant increase in deviance	Leave that term out of the model Inspect the parameter values again Remove the least significant term remaining
4	If the deletion causes a significant increase in deviance	Put the term back in the model using **update +** . These are the statistically significant terms as assessed by deletion from the maximal model
5	Keep removing terms from the model	Repeat steps 3 or 4 until the model contains nothing but significant terms This is the minimal adequate model If none of the parameters is significant, then the minimal adequate model is the null model

Deletion

Deletion uses the **update** directive to remove terms from the model (you can automate this procedure, using **step**). Perhaps the simplest deletion is to remove the intercept from a regression study using the **~ . -1** directive (you need to be careful with the punctuation: the update formula contains "tilde dot minus" which means "fit the model (the tilde) with the last set of explanatory variables (the dot) but remove (the minus sign) the following"). This fits a new regression line with a single parameter Note that it does *not* rotate the regression line about the point $(\bar{x}, \bar{y})$ until it passes through the origin (a common misconception). We can demonstrate this as follows. Take the following data

```
x<-c(0,1,2,3,4,5)
y<-c(2,1,1,5,6,8)
```

and fit a 2-parameter linear regression as model1.

```
model1<-lm(y~x)
model1

Coefficients:

 (Intercept)   x
   0.3333333 1.4
```

The intercept is 0.33333 and the slope is 1.4. This regression line is defined as passing through the point $(\bar{x},\bar{y})$ but we should check this:

```
mean(x); mean(y)

[1] 2.5
[1] 3.833333
```

We check that the model goes through the point (2.5,3.8333) using predict like this

```
predict(model1,list(x=2.5))

    1
3.833333
```

So that's all right then. Now we remove the intercept (~ . -1) using update to fit a single parameter model that is forced to pass through the origin:

```
model2<-update(model1,~. -1)

model2

Coefficients:

        x
1.490909
```

Note how the model formula has been altered by update. The slope of model2 is steeper (1.49 compared with 1.40), but does the line still pass through the point $(\bar{x},\bar{y})$? We can test this using predict:

```
predict(model2,list(x=2.5))

        1
3.727273
```

No, it doesn't. The single slope parameter is estimated from

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

in which the least squares estimate of β is b where

$$b = \frac{\sum xy}{\sum x^2}$$

instead of SSXY / SSX when 2 parameters are estimated from the data (see Practical 4). This graph does not pass through the point ($\bar{x}, \bar{y}$).

Because update(model1,~. –1) causes the regression line to be rotated away from its maximum likelihood position, this will inevitably cause an increase in the residual deviance. If the increase in deviance is significant, as judged by a likelihood ratio test, then the simplification is unwarranted, and the intercept should be added back to the model. Forcing the regression to pass through the origin may also cause problems with non-constancy of variance. Removing the intercept is generally not recommended unless we have confidence in the linearity of the relationship over the whole range of the explanatory variable(s) (i.e. all the way from the origin up the maximum value of *x*).

The practice of model simplification

In general we shall tend to begin the process of model simplification by removing high order interaction terms from a maximal model. Thoughts about forcing the line though the origin are most unlikely to arise. If removal of the high order interaction terms shows them to be non significant, we move on to test the low order interactions and then main effects. The justification for deletion is made with the current model *at the level in question*. Thus in considering whether or not to retain a given second order interaction, it is deleted with all other second order interactions (plus any significant higher-order terms that do not contain the variables of interest) included in the model. If deletion leads to a significant increase in deviance, the term must retained, and the interaction added back into the model. In considering a given first order interaction, all other first order interactions plus any significant higher-order interactions are included at each deletion. And so on.

Main effects which figure in significant interactions should not be deleted. If you do try to delete them there will be no change in deviance and no change in degrees of freedom, because the factor is aliased; the deviance and degrees of freedom will simply be transferred to a surviving interaction term. It is a moot point whether block effects should be removed during model simplification. In the unlikely event that block effects were *not* significant (bearing in mind that in science, everything varies, and so insignificant block effects are the exception rather than the rule), then you should compare your conclusions with and without the block terms in the model. If the conclusions differ, then you need to think very carefully about why, precisely, this has happened. The balance of opinion is that block effects are best left unaltered in the minimal adequate model.

Collapsing factor levels

It often happens that a categorical explanatory variable is significant, but not all of the factor levels are significantly different from one another. We may have a set of *a priori* contrasts that guide model selection. Often, however, we simply want to get rid of factor levels that are redundant. A frequent outcome during anova that, say, the 'low' and 'medium' levels of a treatment are not significantly different from one another, but both differ from the 'high' level treatment. Collapsing factor levels involves calculating a new factor that has the same value for 'low' and 'medium' levels, and another level for ‘high’. This new 2 level factor is then added to the model at the same time as the original 3 level factor is removed. The increase in deviance is noted. If the change is not significant, then the simplification is justified and the new 2-level factor is retained. If a significant increase in deviance occurs, then the original 3 level factor must be restored to the model.

```
yield.data<-read.table("c:\\temp\\levels.txt",header=T)
attach(yield.data)
names(yield.data)
```

```
[1] "yield" "level"
```

```
model<-aov(yield~level)
summary(model)
```

```
            Df  Sum Sq Mean Sq F value   Pr(>F)
level        2 28.1333 14.0667  13.188 0.000935 ***
Residuals   12 12.8000  1.0667
```

A highly significant effect of level is clear. We might leave it at this, but in the interests of parsimony, it would be sensible to ask whether we really need to retain all 3 levels of the factor. We should tabulate the treatment means:

```
tapply(yield,level,mean)
```

```
   A   B    C
 7.2 7.4 10.2
```

The means of levels A and B are very close to one another. Perhaps we can get almost the same explanatory power with a 2-level factor (level2) that has level 1 for A and for B and level 2 for C ?

```
level2<-factor(1+(level=="C"))
level2
```

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

Note the use of logical arithmetic (see Practical 2) to get the second level of *level2*; level==”C” evaluates to 1 when true and to zero when false. Now we can remove *level* from *model* and replace it with *level2* using update like this:

```
model2<-update(model , ~ . - level + level2 )
```

To see if the model simplification was justified, we compare the simpler *model2* with the original *model* using **anova**:

```
anova(model,model2)
```

```
Analysis of Variance Table

Model 1: yield ~ level
Model 2: yield ~ level2

  Res.Df  RSS Df Sum of Sq      F Pr(>F)
1     12 12.8
2     13 12.9 -1      -0.1 0.0937 0.7647
```

The simplification is vindicated by the very high *p* value. Had *p* been less than 0.05 we would return the 3-level factor to the model, but there is no justification for keeping the 3-level factor in this case.

In multiple regression, two explanatory variables may have similar parameter values, and it may make sense to create a single variable that is the sum of the original two variables. For example, if yield increases by 2 kg for every unit of Maxigrow and 2.5 kg for every unit of Yieldmore we might try calculating a single variable (Fertilizer = Maxigrow + Yieldmore) and fitting this to the model instead. If there is no significant increase in residual deviance then we have saved another degree of freedom, and Occam would be proud of us.

Offsets in model simplification

Sometimes there may be a clear theoretical prediction about one or more of the parameter values. In a study of density dependent mortality in plant populations, for example, we might wish to test whether the self-thinning rule applied to a particular set of data relating mean plant weight (*y*) to mean plant density (*x*). Yoda's rule states that the relationship is allometric with $y = ax^{-3/2}$. We might wish to test whether our estimated exponent is significantly different from this. One way would be to do a t-test, comparing our estimate to –1.5 using the standard error from the summary table. An alternative is so specify the exponent as -3/2 in an offset and compare the fit of this model with our initial model in which the maximum likelihood estimate of the exponent was used.

```
Yoda<-read.table("c:\\temp\\Yoda.txt",header=T)
attach(Yoda)
names(Yoda)
[1] "density"  "meansize"
```

The theory is a power law relationship, so the appropriate linear model is log(y) against log(x). We fit the model as a **glm** rather than an **lm** simply because **glm** allows us to use offsets and **lm** does not.

```
model<-glm(log(meansize)~log(density))

summary(model)

Coefficients:
                  Value Std. Error   t value
 (Intercept)  12.213579 0.26622433  45.87702
log(density)  -1.529393 0.06442428 -23.73938
```

The slope is close to the predicted value of-3/2 but is it close enough ? A t-test suggests that it is definitely not significantly different from –1.5

```
(1.529393-1.5)/0.06442482

[1] 0.4562372
```

To find the probability of t = 0.456 with 19 d.f. we use the cumulative probability of Student's t-distribution **pt** like this:

```
1-pt(0.4562372,19)

[1] 0.3266958
```

Here is the same test, but using offsets to specify the slope as exactly –3/2.

```
values<-  - 1.5*log(density)
```

The trick is that we specify the **offset** as part of the model formula like this (much as we specified the error term in a nested design). Now, the only parameter estimated by maximum likelihood is the intercept (coefficient ~1):

```
model2<-glm(log(meansize)~1+offset(values))
```

You get exactly the same model if you leave out the 1+ term; it is included just to emphasise that you are estimating only one parameter. Now we compare the two models using **anova**

```
anova(model,model2,test="F")

Analysis of Deviance Table
Response: log(meansize)

          Terms Resid. Df Resid. Dev Test Df    Deviance   F Value     Pr(F)
1       log(density)   19   8.781306
2 1 + offset(values)   20   8.877508      -1 -0.09620223 0.2081515 0.6533922
```

Our conclusion is the same in both cases, even though the p values differ somewhat (0.327 vs. 0.653). These data conform very precisely with the allometry predicted by Yoda's rule.

There may be some virtue in simplifying parameter values in order to make the numbers easier to communicate. This should never be done, of course, if changing the parameter values causes a significant reduction in the explanatory power of the

model. It is much more straightforward, for example, to say that yield increases by 2 kg per hectare for every extra unit of fertilizer, than to say that it increases by 1.947 kg. Similarly, it may be preferable to say that the odds of infection increase 10-fold under a given treatment, than to say that the logits increase by 2.321; without model simplification this is equivalent to saying that there is a 10.186-fold increase in the odds.

Caveats

Model simplification is an important process but it should not be taken to extremes. For example, the interpretation of deviances and standard errors produced with fixed parameters that have been estimated from the data, should be undertaken with caution. Again, the search for 'nice numbers' should not be pursued uncritically. Sometimes there are good scientific reasons for using a particular number (e.g. a power of 0.66 in an allometric relationship between respiration and body mass), but it would be absurd to fix on an estimate of 6 rather than 6.1 just because 6 is a whole number.

Summary

Remember that *order matters*. If your explanatory variables are correlated with each other, then the significance you attach to a given explanatory variable will depend upon whether you delete it from a maximal model or add it to the null model. Always test by model simplification and you won't fall into this trap.

The fact that we have laboured long and hard to include a particular experimental treatment does not justify the retention of that factor in the model if the analysis shows it to have no explanatory power. Anova tables are often published containing a mixture of significant and non-significant effects. This is not a problem in orthogonal designs, because sums of squares can be unequivocally attributed to each factor and interaction term. But as soon as there are missing values or unequal weights, then it is impossible to tell how the parameter estimates and standard errors of the significant terms would have been altered if the non-significant terms had been deleted. The best practice is this

- say whether your data are orthogonal or not
- present a minimal adequate model
- give a list of the non-significant terms that were omitted, and the deviance changes that resulted from their deletion

The reader can then judge for themselves the relative magnitude of the non-significant factors, and the importance of correlations between the explanatory variables.

The temptation to retain terms in the model that are 'close to significance' should be resisted. The best way to proceed is this. If a result would have been *important* if it had been statistically significant, then it is worth repeating the experiment with higher replication and/or more efficient blocking, in order to demonstrate the importance of the factor in a convincing and statistically acceptable way.

Multiple Regression

When there is more than one continuous explanatory variable there are lots of choices that need to be made. At the data inspection stage, there are many more kinds of **plots** we could do:

1) plot the response against each of the explanatory variables separately

2) plot the explanatory variables against one another (e.g. **pairs**)

3) plot the response against pairs of explanatory variables in 3-D plots

4) plot the response against explanatory variables for different combinations of other explanatory variables (e.g. conditioning plots, **coplot**; see p. 16).

At the modelling stage, we need to choose between multiple regression, non parametric surface-fitting (local regression or loess), additive models (with perhaps a mix of parametric and non parametric terms), tree models or multivariate techniques.

At the model checking stage, we need to be particularly concerned with the extent of correlations between the explanatory variables

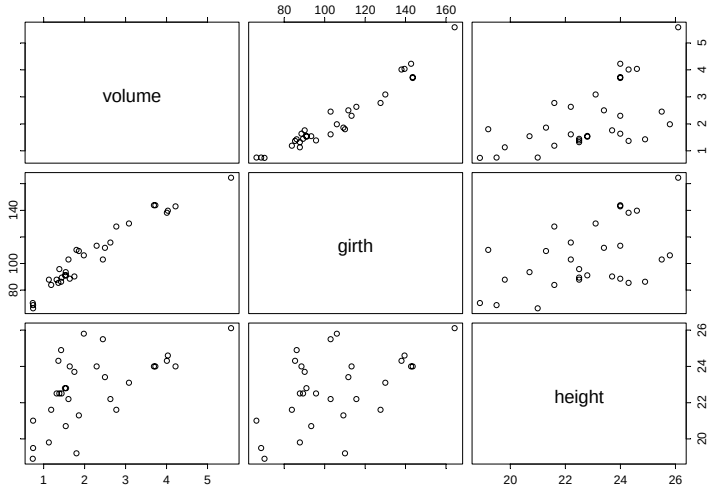
We shall begin with an extremely simple example with just two explanatory variables. The response variable is the volume of utilisable timber in harvested trunks of different lengths and diameters.

```
timber<-read.table("c:\\temp\\timber.txt",header=T)
attach(timber)
names(timber)
```

```
[1] "volume" "girth"  "height"
```

We begin by comparing different kinds of plots. It is a good idea to start by plotting the response against each of the explanatory variables in turn, and by looking at the extent to which the explanatory variables are correlated with each other. The multi-panel **pairs** function is excellent for this (see Practical 1).

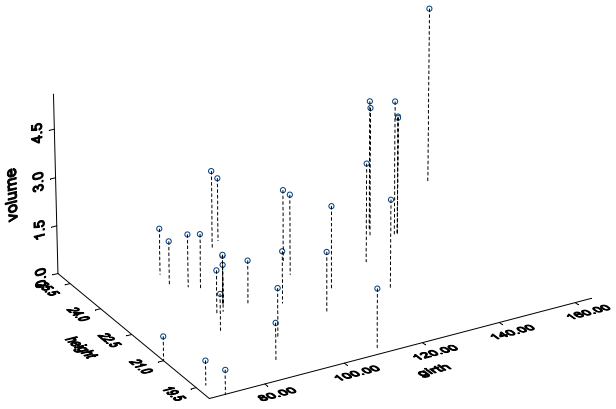
```
pairs(timber)
```



It is clear that girth is a much better predictor of the volume of usable timber than is height (top row; middle and right hand panels). This is what we might have expected, because it is very easy to imagine two tree trunks of exactly the same length that differ enormously in the volume of timber they continue. The long thin one might contain no useful timber at all, but you could build a whole house out of the stout one. The bottom-right quartet of panels shows that there is a positive correlation between the two explanatory variables, but trunks of the same girth appear to differ more in height than trunks of the same height differ in girth.

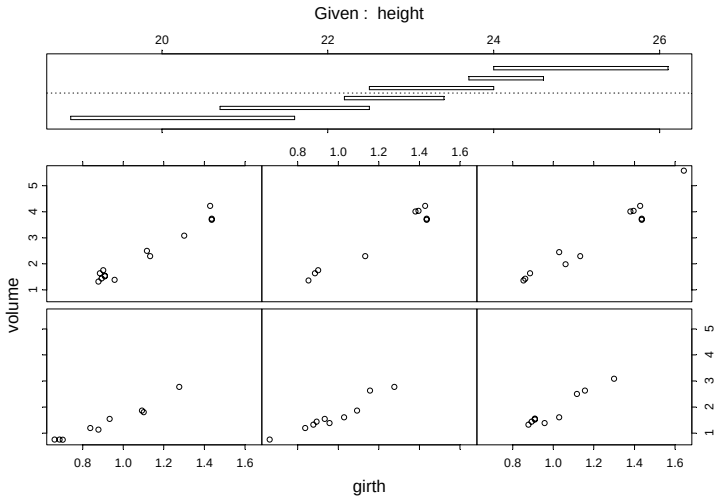
Although experts always advise against it, you might well be tempted to look at a **3D scatterplot** of the data. The reason experts advise against it is a good one: they argue that it is hard to interpret the data correctly from a perspective plot, and the whole idea of plotting is to aid interpretation. The problem with 3D scatterplots is that it is impossible to know where a point is located; you don't know whether it is in the foreground or the background. Plots showing 3D *surfaces* are another matter altogether; they can be very useful in interpreting the models that emerge from multiple regression analysis. For what it is worth, here is the 3D scatterplot of the timber data: gui stands for Graphics User Interface (**R doesn't do this**):

```
guiPlot("Drop Lines Plot",data,frame(girth,height,volume))
```



Perhaps the most useful kinds of plots for this purpose are **conditioning plots**. These allow you to see whether the response depends upon an explanatory variable in different ways at different levels of other explanatory variables. It takes 2 dimensional slices through a potentially multi-dimensional volume of parameter space, and these are often much more informative than the unconditioned scatterplots produced by **pairs**. Here is volume against girth, conditioning on height. Note the use of a model formula in the plotting directive:

```
coplot(volume~girth|height)
```



Much of the scatter that was such a feature of the **pairs** plot has gone. Within a height class, the relationship between volume and girth is relatively close. What seems to be happening is that the *slope* of the relationship depends upon the height, getting steeper for taller trees. This is a very informative plot. It tells us that both girth and height are likely to be required in the minimal model and that there may be an interaction between the two explanatory variables.

The multiple regression model

The assumptions are the same as with simple linear regression. The explanatory variables are assumed to be measured without error, the errors are normally distributed, the errors are confined to the response variable, and the variance is constant. The model for a multiple regression with 2 explanatory variables (x_1 and x_2) looks like this:

$$y_i = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \varepsilon_i$$

The *i*th data point, y_i, is determined by the levels of the 2 continuous explanatory variables $x_{1,i}$ and $x_{2,i}$ by the model's 3 parameters (the intercept β_0 and the two slopes β_1 and β_2), and by the residual ε_i of point *i* from the fitted surface. More generally, the model is presented like this:

$$y_i = \sum \beta_i x_i + \varepsilon_i$$

where the summation term is called the **linear predictor**, and can involve many explanatory variables, non linear terms and interactions. The question that immediately arises is this: how are the values of the parameters determined? In order to see this, we return to the simple linear regression case that we investigated earlier, and re-set that example in terms of matrix algebra. Once the matrix notation is reasonably clear, we can then extend it to deal with an arbitrarily large number of explanatory variables.

N.B. If you are not familiar with matrix algebra, then go directly to p. 230 and skip the next section.

Matrix representation of regression: working through a linear regression in matrix form

The best way to learn how matrix algebra can generalise our ideas about regression is to work through a simple example where the data are already familiar. The example involved weight gain of caterpillars fed diets with differing concentrations of tannin (see Practical 4). The famous five and sample size were:

$$\sum x, \sum x^2, \sum y, \sum y^2, \sum xy, n$$

```
36   204 62 536   175 9
```

The 3 steps involved are these:

1) Display the model in matrix form

$$\mathbf{Y = Xb + e}$$

2) Determine the least squares estimate of **b**

$$\mathbf{b=(X'X)^{-1}X'Y}$$

3) Carry out the analysis of variance

b'X'Y'

We look at each of these in turn. The response variable **Y, 1** and the errors **e** are simple *n* x 1 column vectors, **X** is a *n* x 2 matrix and β is a 2 x 1 vector of coefficients.

$$\mathbf{Y} = \begin{bmatrix} 12 \\ 10 \\ 8 \\ 11 \\ 6 \\ 7 \\ 2 \\ 3 \\ 3 \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \\ 1 & 6 \\ 1 & 7 \\ 1 & 8 \end{bmatrix} \quad \mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \\ e_8 \\ e_9 \end{bmatrix} \quad \mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

The y vector and the 1-vector are entered into the computer like this:

```
y<-c(12,10,8,11,6,7,2,3,3)

one<-rep(1,9)
```

The sample size is **1'1** (transpose of *one* times *one*):

```
t(one) %*% one
```

```
     [,1]
[1,]    9
```

The hard part concerns the matrix of the explanatory variables. This has one more column than there are explanatory variables. The extra column is a column of 1's on the left hand side of the matrix. Because the *x* values are evenly spaced we can generate them rather than type them in (use lower case *x*)

```
x<-0:8
x
```

```
[1] 0 1 2 3 4 5 6 7 8
```

We manufacture the matrix **X** by using **cbind** to tack a column of 1's in front of the vector of *x*'s. Note that the single number 1 is *coerced* into length *n* to match *x* (use upper case X):

```
X<-cbind(1,x)
```

X

```
      x
[1,] 1 0
[2,] 1 1
[3,] 1 2
[4,] 1 3
[5,] 1 4
[6,] 1 5
[7,] 1 6
[8,] 1 7
[9,] 1 8
```

Thus, all of the matrices are of length $n = 9$, except for β which is length $k+1$ (where k = number of explanatory variables; 1 in this example).

The transpose of a matrix (denoted by a prime superscript) is the matrix obtained by writing the rows as columns in the order in which they occur, so that the columns all become rows. So a 9x2 matrix **X** transposes (**t**) to a 2x9 matrix **X'** (called *Xp* for 'x prime') like this:

Xp <- t(X)

```
Xp
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
     1    1    1    1    1    1    1    1    1
x    0    1    2    3    4    5    6    7    8
```

This is useful for regression, because

$$\sum y^2 = y_1^2 + y_2^2 + \ldots + y_n^2 = \mathbf{Y'Y}$$

t(y) %*% y

```
     [,1]
[1,]  536
```

$$\sum y = n\bar{y} = y_1 + y_2 + \ldots + y_n = \mathbf{1'Y}$$

t(one) %*% y

```
   [,1]
[1,]   62
```

$$(\sum y)^2 = \mathbf{Y'11'Y}$$

t(y) %*% one %*% t(one) %*% y

```
     [,1]
[1,] 3844
```

For the matrix of explanatory variables, we see that **X'X** gives a 2x2 matrix containing *n*, $\sum x$ and $\sum x^2$. The numerical values are easy to find using matrix multiplication %*%

```
Xp %*% X
```

```
        x
   9   36
x 36   204
```

Note that **X'X** (a 2x2 matrix) is completely different from **XX'** (a 9x9 matrix). The matrix **X'Y** gives a 2x1 matrix containing $\sum y$, and the sum of products $\sum xy$

```
Xp %*% y
```

```
   [,1]
     62
x   175
```

Using the beautiful symmetry of the normal equations

$$b_0 n + b_1 \sum x = \sum y$$

$$b_0 \sum x + b_1 \sum x^2 = \sum xy$$

we can write the regression directly in matrix form as

$$\mathbf{X'Xb = X'Y}$$

because we already have the necessary matrices to form the left and right hand sides. To find the least squares parameter values **b** we need to divide both sides by **X'X.** This involves calculating the **inverse** of the **X'X** matrix. The inverse exists only when the matrix is square and when its determinant is non-singular. The inverse contains $-\bar{x}$ and $\sum x^2$ as its terms, with SSX as the denominator.

$$\mathbf{(X'X)^{-1}} = \begin{bmatrix} \dfrac{\sum x^2}{n\sum (x-\bar{x})^2} & \dfrac{-\bar{x}}{\sum (x-\bar{x})^2} \\ \dfrac{-\bar{x}}{\sum (x-\bar{x})^2} & \dfrac{1}{\sum (x-\bar{x})^2} \end{bmatrix}$$

When every element of a matrix has a common factor, it can be taken outside the matrix. Here, the term 1/(n.SSX) can be taken outside to give:

$$\mathbf{(X'X)^{-1}} = \frac{1}{n\sum (x-\bar{x})^2} \begin{bmatrix} \sum x^2 & -\sum x \\ -\sum x & n \end{bmatrix}$$

Computing the numerical value of this is easy using the matrix function **ginv** (this stands for generalized inverse):

```
XM<-Xp %*% X

library(MASS)

ginv(XM)
```

```
             [,1]         [,2]
[1,]  0.37777778 -0.06666667
[2,] -0.06666667  0.01666667
```

Now we can solve the normal equations

$$\mathbf{(X'X)^{-1}(X'X)b = (X'X)^{-1}X'Y}$$

using the fact that $\mathbf{(X'X)^{-1}(X'X)=I}$ to obtain the important general result:

$$\mathbf{b = (X'X)^{-1}X'Y}$$

In our example, we have $\mathbf{(X'X)^{-1}}$ as

```
             [,1]         [,2]
[1,]  0.37777778 -0.06666667
[2,] -0.06666667  0.01666667
```

and **X'Y** as

```
[,1]
    62
x  175
```

so **b** is found by the 3-matrix product

```
b<-ginv(XM) %*% Xp %*% y

b
            [,1]
[1,]  11.755556
[2,]  -1.216667
```

which you will recognise as the intercept and slope respectively.

The ANOVA computations are as follows. The correction factor, CF, = **Y'11'Y**/n

```
CF<-t(y) %*% one %*% t(one) %*% y / 9
CF
```

```
          [,1]
[1,] 427.1111
```

SST is **Y'Y –** CF

t(y) %*% y - CF

```
          [,1]
[1,] 108.8889
```

SSR is **b'X'Y** – CF

t(b) %*% t(X) %*% y - CF

```
          [,1]
[1,] 88.81667
```

and SSE is **Y'Y – b'X'Y**

t(y) %*% y - t(b) %*% t(X) %*% y

```
          [,1]
[1,] 20.07222
```

rm(b,y,CF)

Working through a multiple regression by hand

To show how multiple regression works, we shall go through a simple example long-hand. We have two explanatory variables (girth and height) and the response variable is the volume of usable timber logs of this girth and height. Here is the data set (it is loaded on p. 219):

timber

```
   volume  girth height
 1 0.7458  66.23   21.0
 2 0.7458  68.62   19.5
 3 0.7386  70.22   18.9
 4 1.1875  83.79   21.6
 5 1.3613  85.38   24.3
 6 1.4265  86.18   24.9
 7 1.1296  87.78   19.8
 8 1.3179  87.78   22.5
 9 1.6365  88.57   24.0
10 1.4410  89.37   22.5
11 1.7524  90.17   23.7
12 1.5206  90.97   22.8
13 1.5496  90.97   22.8
14 1.5424  93.36   20.7
15 1.3831  95.76   22.5
16 1.6075 102.94   22.2
17 2.4475 102.94   25.5
18 1.9841 106.13   25.8
19 1.8610 109.32   21.3
20 1.8030 110.12   19.2
21 2.4982 111.71   23.4
22 2.2954 113.31   24.0
23 2.6285 115.70   22.2
24 2.7734 127.67   21.6
25 3.0847 130.07   23.1
26 4.0116 138.05   24.3
```

```
27 4.0333 139.64   24.6
28 4.2216 142.84   24.0
29 3.7292 143.63   24.0
30 3.6930 143.63   24.0
31 5.5757 164.38   26.1
```

Note that in the data frame, girth is measured in cm and height in m. To keep things simple, we convert girth to m as well (volume is in m^3 already):

```
girth<-girth/100
```

The response variable (volume) is re-named *y* like this:

```
y<-volume
```

The vector of explanatory variables **X** is made by **cbind** like this

```
X<-cbind(1,girth,height)
```

and its transpose is Xp (X prime) **X'**

```
Xp<-t(X)
```

We obtain the sums of X and the sums of squares of X from **X'X**

```
Xp %*% X
```

```
                    girth     height
        31.0000  32.77230   706.8000
 girth  32.7723  36.52706   754.6654
height 706.8000 754.66542 16224.6600
```

This reads as follows. Top left is n = 31, the number of trees. The second row of the first column shows the sum of the girths = 32.7723, and the third row is the sum of the heights = 706.8. The second row of column 2 shows the sum of the squares of girth = 36.52706 and the third row the sum of the products girth*height = 754.66542 (the symmetry has this figure in the second row of the 3rd column as well). The last figure in the bottom right is the sum of the squares of the heights = 16224.66.

We get the sums of products from **X'Y**

```
Xp %*% y
```

```
             [,1]
         67.72630
 girth   80.24607
height 1584.99573
```

The top number is the sum of the timber volumes $\sum y$ = 67.7263. The second number is the sum of the products volume * girth = 80.24607 and the last number is the sum of the products volume * height = 1584.99573.

Now we need the inverse of **X'X**

```
ginv(Xp%*%X)
```

```
            [,1]         [,2]         [,3]
[1,]  4.9519523  0.35943351 -0.23244197
[2,]  0.3594335  0.72786995 -0.04951388
[3,] -0.2324420 -0.04951388  0.01249064
```

which shows all of the corrected sums of squares of the explanatory variables. For instance, the top left hand number 4.9519523 is

$$\frac{\sum g^2 \sum h^2 - (\sum g.h)^2}{c - d}$$

where *g* stands for girth and *h* for height, *c* is $n\sum g^2 \sum h^2 + 2\sum g \sum h \sum gh$ and *d* is $n(\sum gh)^2 + (\sum g)^2 \sum h^2 + (\sum h)^2 \sum g^2$ (see Draper & Smith (1981) for details of how to calculate determinants for matrices larger than 2x2).

Finally we compute the parameter values, **b**:

```
b<-ginv(Xp %*% X) %*% Xp %*% y
b
```

```
            [,1]
[1,] -4.19899732
[2,]  4.27251096
[3,]  0.08188343
```

and compare the vector **b** with the parameter estimates obtained by multiple regression

```
lm(volume~girth+height)
```

```
Coefficients:
 (Intercept)    girth     height
   -4.198997 4.272511 0.08188343
```

and are hugely relieved to find that they are the same. The first element of **b** is the intercept (-4.199), the second is the slope of the graph of volume against girth (4.27) and the third is the slope of the graph of volume against height (0.082).

To finish, we compute the sums of squares for the anova table, starting with the correction factor

```
CF<-t(y) %*% one %*% t(one) %*% y / length(y)
CF
         [,1]
[1,] 147.963

sst<-t(y) %*% y - CF
sst
          [,1]
[1,] 42.50408

ssr<-t(b) %*% Xp %*% y - CF
ssr
          [,1]
[1,] 40.29156
```

This is the sum of the girth sum of squares (39.75477) and the height (0.53679). Finally sse can be computed by difference:

```
sse<-sst-ssr
sse
          [,1]
[1,] 2.212518
```

These check out with sums of squares in the anova produced by the **aov** fit:

```
model<-aov(volume~girth+height)
summary(model)

            Df Sum of Sq  Mean Sq  F Value      Pr(F)
    girth    1  39.75477 39.75477 503.1070 0.00000000
   height    1   0.53679  0.53679   6.7933 0.01449791
Residuals   28   2.21252  0.07902
```

Note that although the original scatter was greater on the graph of volume against girth, it is girth that explains more of the variation in volume in a model containing both explanatory variables. A more thorough analysis of these data, employing a more realistic scale of measurement, is described later.

Multiple Regression

In multiple regression we have a continuous response variable and two or more continuous explanatory variables (i.e. no categorical explanatory variables). There are several important issues involved in carrying out a multiple regression:

- which explanatory variables to include
- curvature in the response to the explanatory variables
- interactions between explanatory variables
- correlation between explanatory variables
- the risk of over-parameterization

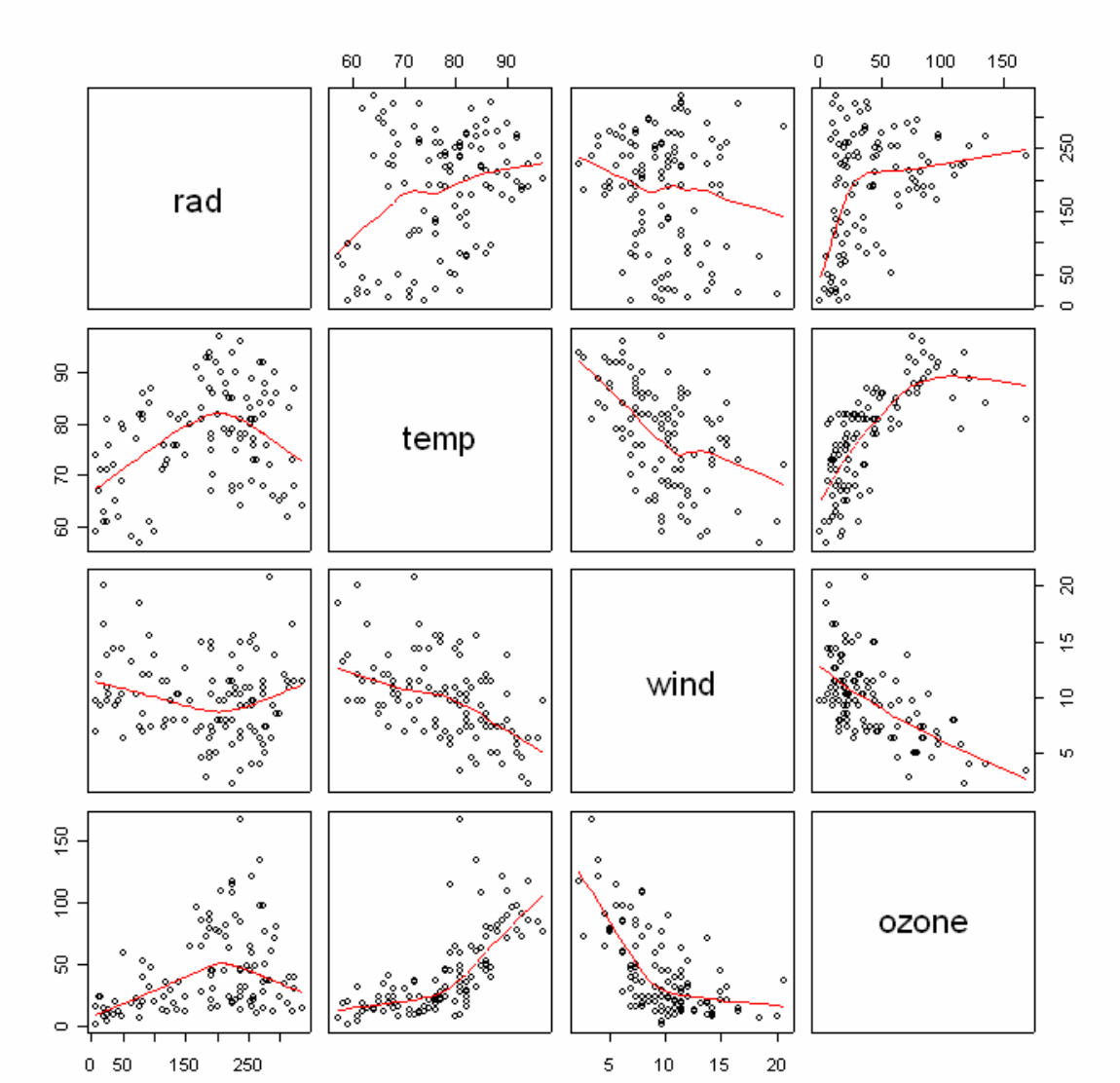
Let's begin with an example from air pollution studies. How is ozone concentration related to wind speed, air temperature and the intensity of solar radiation?

```
ozone.pollution<-read.table("c:\\temp\\ozone.data.txt",header=T)
attach(ozone.pollution)
names(ozone.pollution)
```

```
[1] "rad"   "temp"  "wind"  "ozone"
```

In multiple regression, it is always a good idea to use pairs to look at all the correlations:

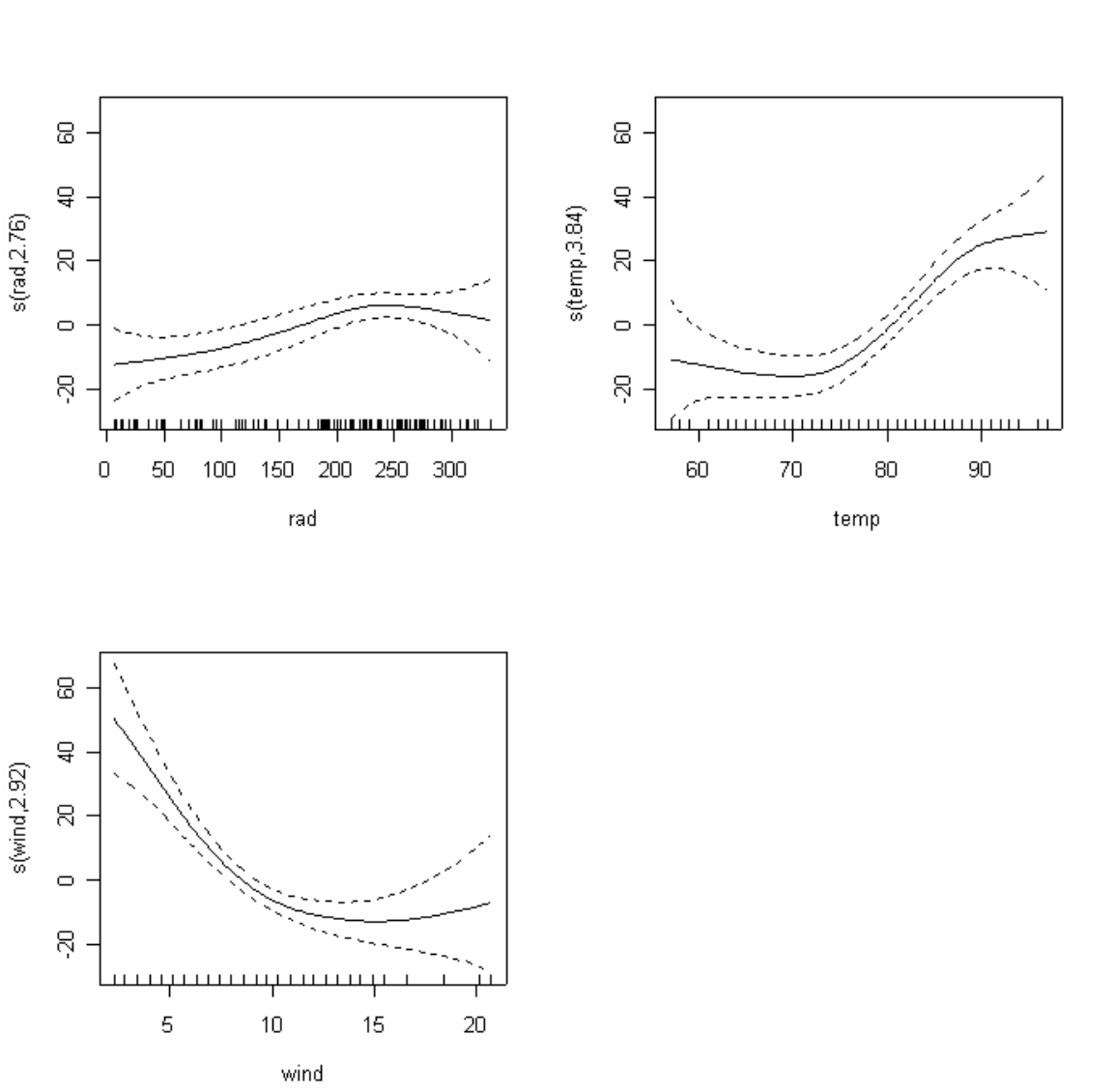
```
pairs(ozone.pollution,panel=panel.smooth)
```



The response variable, ozone concentration, is shown on the *y* axis of the bottom row of panels: there is a strong negative relationship with wind speed, a positive correlation with temperature and a rather unclear, humped relationship with radiation.

A good way to start a multiple regression problem is using non-parametric smoothers in a generalized additive model (gam) like this:

```
library(mgcv)
par(mfrow=c(2,2))
model<-gam(ozone~s(rad)+s(temp)+s(wind))
plot(model)
par(mfrow=c(1,1))
```

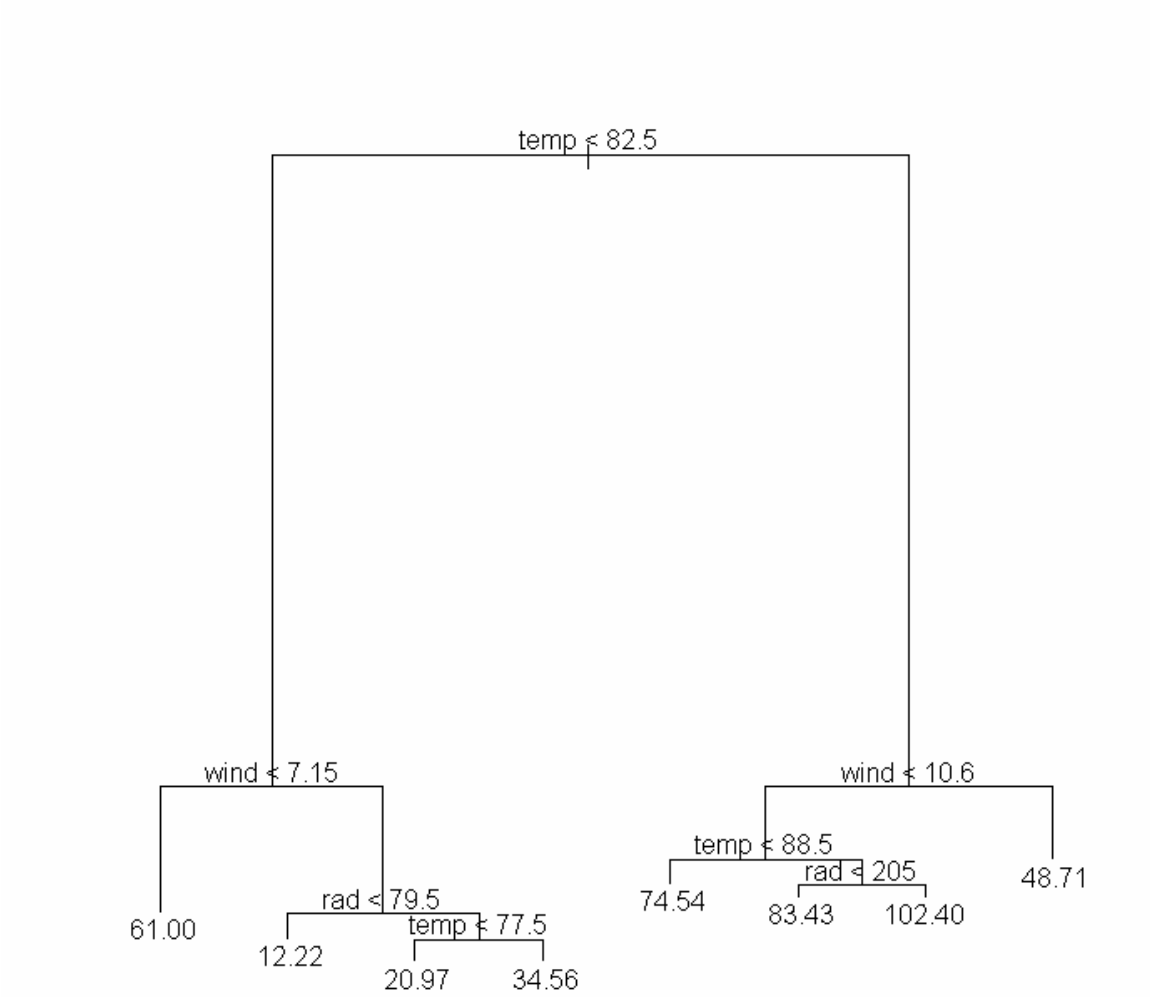


The confidence intervals are sufficiently narrow to suggest that the curvature in the relationship between ozone and temperature is real, but the curvature of the relationship with wind is questionable, and a linear model may well be all that is required for solar radiation.

The next step might be to fit a tree model to see whether complex interactions between the explanatory variables are indicated:

```
library(tree)
```

```
model<-tree(ozone~.,data=ozone.pollution)
plot(model)
text(model)
```



This shows that temperature is far and away the most important factor affecting ozone concentration (the longer the branches in the tree, the greater the deviance explained). Wind speed is important at both high and low temperatures, with still air being associated with higher mean ozone levels (the figures at the ends of the branches). Radiation shows an interesting, but subtle effect. At low temperatures, radiation matters at relatively high wind speeds (> 7.15), whereas at high temperatures, radiation matters at relatively low wind speeds (< 10.6); in both cases, however, higher radiation is associated with higher mean ozone concentration. The tree model therefore indicates that the interaction structure of the data is not complex (a reassuring finding).

Armed with this background information (likely curvature of the temperature response and an uncomplicated interaction structure) we can begin the linear modelling. We start with the most complicated model: this includes interactions between all 3 explanatory variables plus quadratic terms to test for curvature in response to each of the 3 explanatory variables. If you want to do calculations inside

the model formula (e.g. produce a vector of squares for fitting quadratic terms), then you need to use the "as is" function, which is a capital I () like this (i.e. not a 1 or a lower case L):

```
model1<-lm(ozone~temp*wind*rad+I(rad^2)+I(temp^2)+I(wind^2))
summary(model1)
```

```
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    5.683e+02  2.073e+02   2.741  0.00725 **
temp          -1.076e+01  4.303e+00  -2.501  0.01401 *
wind          -3.237e+01  1.173e+01  -2.760  0.00687 **
rad           -3.117e-01  5.585e-01  -0.558  0.57799
I(rad^2)      -3.619e-04  2.573e-04  -1.407  0.16265
I(temp^2)      5.833e-02  2.396e-02   2.435  0.01668 *
I(wind^2)      6.106e-01  1.469e-01   4.157 6.81e-05 ***
temp:wind      2.377e-01  1.367e-01   1.739  0.08519 .
temp:rad       8.402e-03  7.512e-03   1.119  0.26602
wind:rad       2.054e-02  4.892e-02   0.420  0.67552
temp:wind:rad -4.324e-04  6.595e-04  -0.656  0.51358

Residual standard error: 17.82 on 100 degrees of freedom
Multiple R-Squared: 0.7394,     Adjusted R-squared: 0.7133
F-statistic: 28.37 on 10 and 100 DF,  p-value:     0
```

The 3-way interaction is clearly not significant, so we remove it to begin the process of model simplification:

```
model2<-update(model1,~. – temp:wind:rad)
summary(model2)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.245e+02  1.957e+02   2.680   0.0086 **
temp        -1.021e+01  4.209e+00  -2.427   0.0170 *
wind        -2.802e+01  9.645e+00  -2.906   0.0045 **
rad          2.628e-02  2.142e-01   0.123   0.9026
I(rad^2)    -3.388e-04  2.541e-04  -1.333   0.1855
I(temp^2)    5.953e-02  2.382e-02   2.499   0.0141 *
I(wind^2)    6.173e-01  1.461e-01   4.225 5.25e-05 ***
temp:wind    1.734e-01  9.497e-02   1.825   0.0709 .
temp:rad     3.750e-03  2.459e-03   1.525   0.1303
wind:rad    -1.127e-02  6.277e-03  -1.795   0.0756 .
```

Start by removing the least significant interaction term. From model1 this looks like `wind:rad` (p = 0.67552), but the sequence has been altered by removing the 3-way term (it is now `temp:rad` with p = 0.1303):

```
model3<-update(model2,~. - temp:rad)
summary(model3)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.488e+02  1.963e+02   2.796  0.00619 **
temp        -1.144e+01  4.158e+00  -2.752  0.00702 **
wind        -2.876e+01  9.695e+00  -2.967  0.00375 **
rad          3.061e-01  1.113e-01   2.751  0.00704 **
```

```
I(rad^2)    -2.690e-04  2.516e-04  -1.069  0.28755
I(temp^2)    7.145e-02  2.265e-02   3.154  0.00211 **
I(wind^2)    6.363e-01  1.465e-01   4.343 3.33e-05 ***
temp:wind    1.840e-01  9.533e-02   1.930  0.05644 .
wind:rad    -1.381e-02  6.090e-03  -2.268  0.02541 *
```

The `temp:wind` interaction is close to significance, but we are ruthless in our pruning, so we take it out:

```
model4<-update(model3,~. - temp:wind)
summary(model4)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.310e+02  1.082e+02   2.135 0.035143 *
temp        -5.442e+00  2.797e+00  -1.946 0.054404 .
wind        -1.080e+01  2.742e+00  -3.938 0.000150 ***
rad          2.405e-01  1.073e-01   2.241 0.027195 *
I(rad^2)    -2.010e-04  2.524e-04  -0.796 0.427698
I(temp^2)    4.484e-02  1.821e-02   2.463 0.015432 *
I(wind^2)    4.308e-01  1.020e-01   4.225 5.16e-05 ***
wind:rad    -9.774e-03  5.794e-03  -1.687 0.094631 .
```

Now the `wind:rad` interaction, which looked so significant in model3 ($p = 0.02541$), is evidently not significant, so we take it out as well:

```
model5<-update(model4,~. - wind:rad)
summary(model5)
```

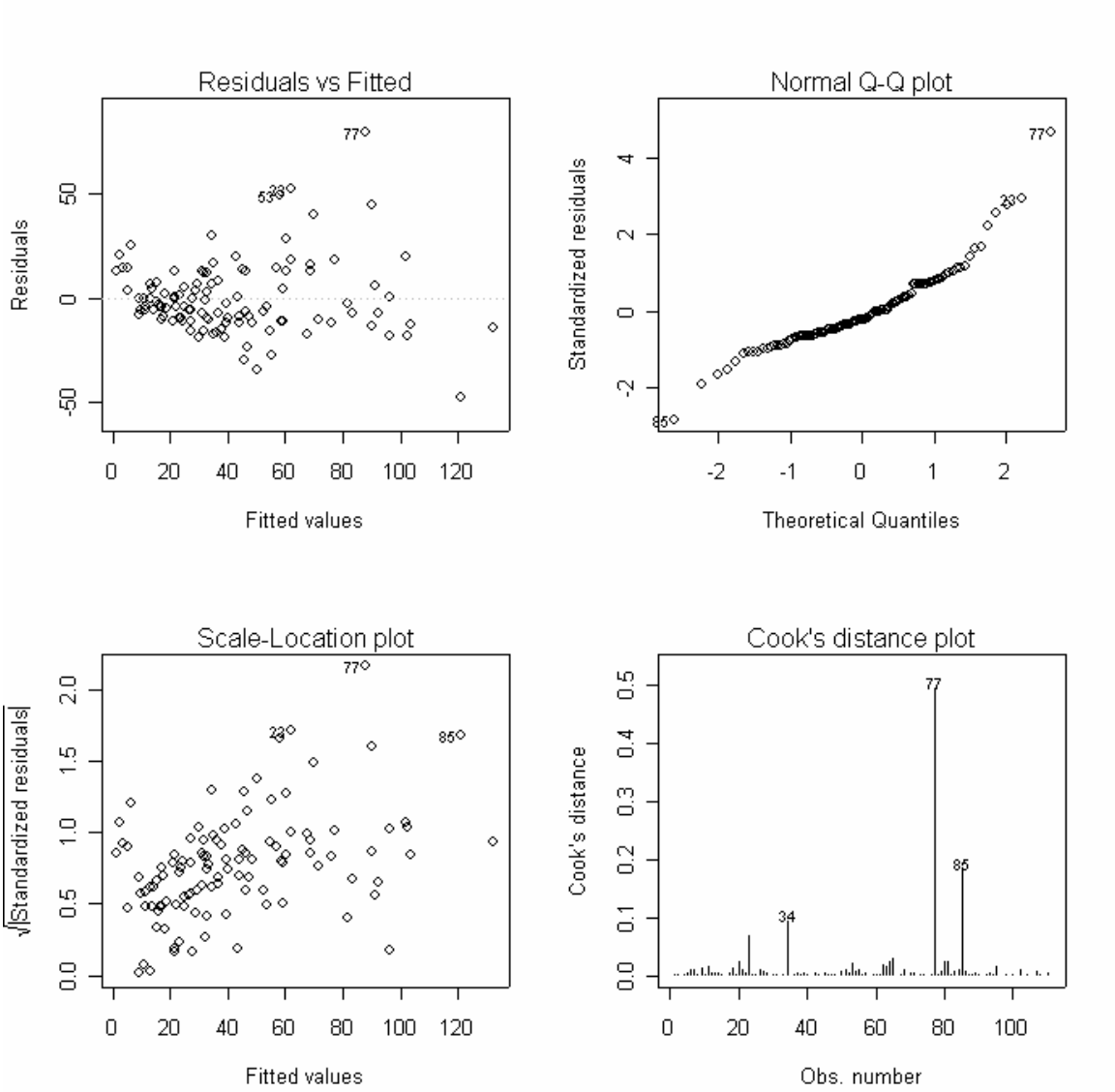
```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.985e+02  1.014e+02   2.942  0.00402 **
temp        -6.584e+00  2.738e+00  -2.405  0.01794 *
wind        -1.337e+01  2.300e+00  -5.810 6.89e-08 ***
rad          1.349e-01  8.795e-02   1.533  0.12820
I(rad^2)    -2.052e-04  2.546e-04  -0.806  0.42213
I(temp^2)    5.221e-02  1.783e-02   2.928  0.00419 **
I(wind^2)    4.652e-01  1.008e-01   4.617 1.12e-05 ***
```

There is no evidence to support retaining any of the 2-way interactions. What about the quadratic terms: the term in `rad^2` looks insignificant, so we take it out:

```
model6<-update(model5,~. - I(rad^2))
summary(model6)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 291.16758  100.87723   2.886  0.00473 **
temp         -6.33955    2.71627  -2.334  0.02150 *
wind        -13.39674    2.29623  -5.834 6.05e-08 ***
rad           0.06586    0.02005   3.285  0.00139 **
I(temp^2)     0.05102    0.01774   2.876  0.00488 **
I(wind^2)     0.46464    0.10060   4.619 1.10e-05 ***

Residual standard error: 18.25 on 105 degrees of freedom
Multiple R-Squared: 0.713,      Adjusted R-squared: 0.6994
F-statistic: 52.18 on 5 and 105 DF,  p-value: < 2.2e-16
```

Now we are making progress. All the terms in model6 are significant. We should check the assumptions, using plot(model6):



There is a clear pattern of variance increasing with the mean of the fitted values. This is bad news (heteroscedasticity). Also, the normality plot is distinctly curved; again, this is bad news. Let's try transformation of the response variable. There are no zeros in the response, so a log transformation is worth trying:

```
model7<-lm(log(ozone) ~ temp + wind + rad + I(temp^2) + I(wind^2))
summary(model7)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.5538486  2.7359735   0.933  0.35274
temp        -0.0041416  0.0736703  -0.056  0.95528
wind        -0.2087025  0.0622778  -3.351  0.00112 **
rad          0.0025617  0.0005437   4.711 7.58e-06 ***
I(temp^2)    0.0003313  0.0004811   0.689  0.49255
I(wind^2)    0.0067378  0.0027284   2.469  0.01514 *

Residual standard error: 0.4949 on 105 degrees of freedom
```

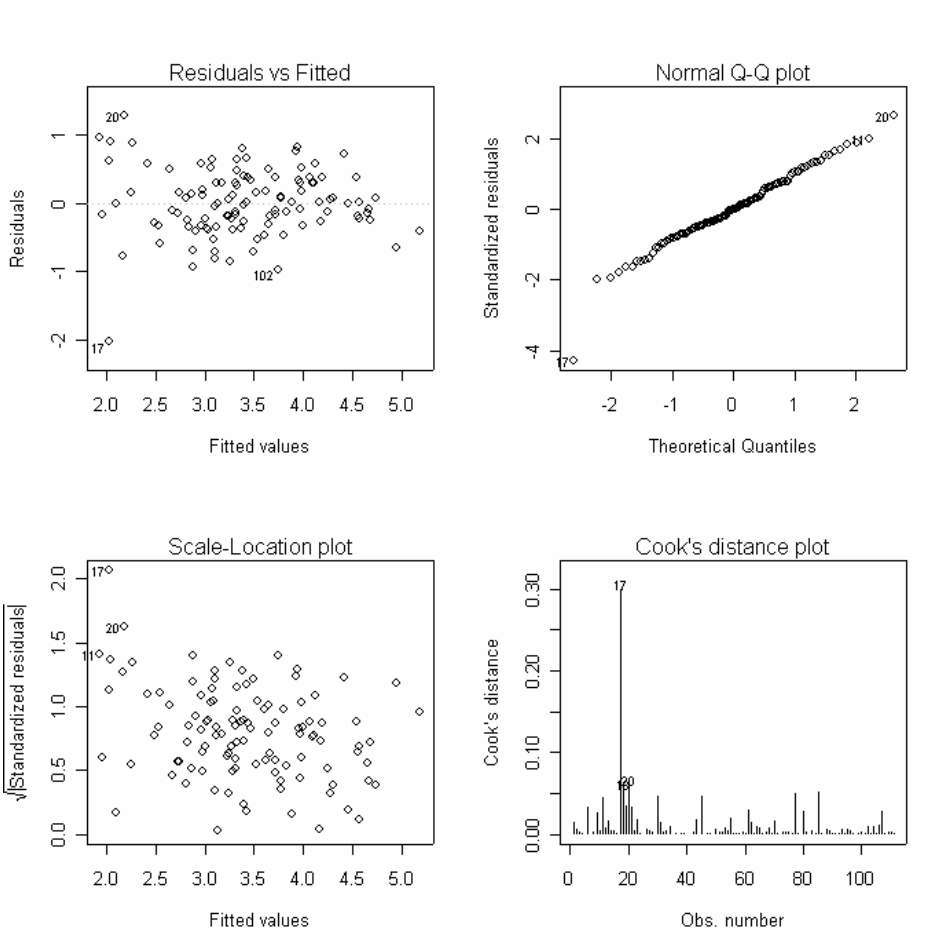
```
Multiple R-Squared: 0.6882,     Adjusted R-squared: 0.6734
F-statistic: 46.36 on 5 and 105 DF,  p-value:     0
```

On the log(ozone) scale, there is no evidence for a quadratic term in temperature, so let's remove that:

```
model8<-update(model7,~. - I(temp^2))
summary(model8)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.7231644  0.6457316   1.120  0.26528
temp         0.0464240  0.0059918   7.748 5.94e-12 ***
wind        -0.2203843  0.0597744  -3.687  0.00036 ***
rad          0.0025295  0.0005404   4.681 8.49e-06 ***
I(wind^2)    0.0072233  0.0026292   2.747  0.00706 **

Residual standard error: 0.4936 on 106 degrees of freedom
Multiple R-Squared: 0.6868,     Adjusted R-squared: 0.675
F-statistic: 58.11 on 4 and 106 DF,  p-value:     0
```

```
plot(model8)
```



The heteroscedasticity and the non-normality have been cured, but there is now a highly influential data point (number 17 on the Cook's plot). We should refit the

model with this point left out, to see if the parameter estimates or their standard errors are greatly affected:

```
model9<-lm(log(ozone) ~ temp + wind + rad + I(wind^2),subset=(1:length(ozone)!=17))
summary(model9)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.1932358  0.5990022   1.992 0.048963 *
temp         0.0419157  0.0055635   7.534 1.81e-11 ***
wind        -0.2208189  0.0546589  -4.040 0.000102 ***
rad          0.0022097  0.0004989   4.429 2.33e-05 ***
I(wind^2)    0.0068982  0.0024052   2.868 0.004993 **

Residual standard error: 0.4514 on 105 degrees of freedom
Multiple R-Squared: 0.6974,     Adjusted R-squared: 0.6859
F-statistic:  60.5 on 4 and 105 DF,  p-value:     0
```

Finally, plot(model9) shows that the variance and normality are well behaved, so we can stop at this point. We have found the minimal adequate model. It is on a scale of log(ozone concentration), all the main effects are significant, but there are no interactions, and there is a single quadratic term for wind speed (5 parameters in all, with 105 d.f. for error).

A more realistically complex example of multiple regression

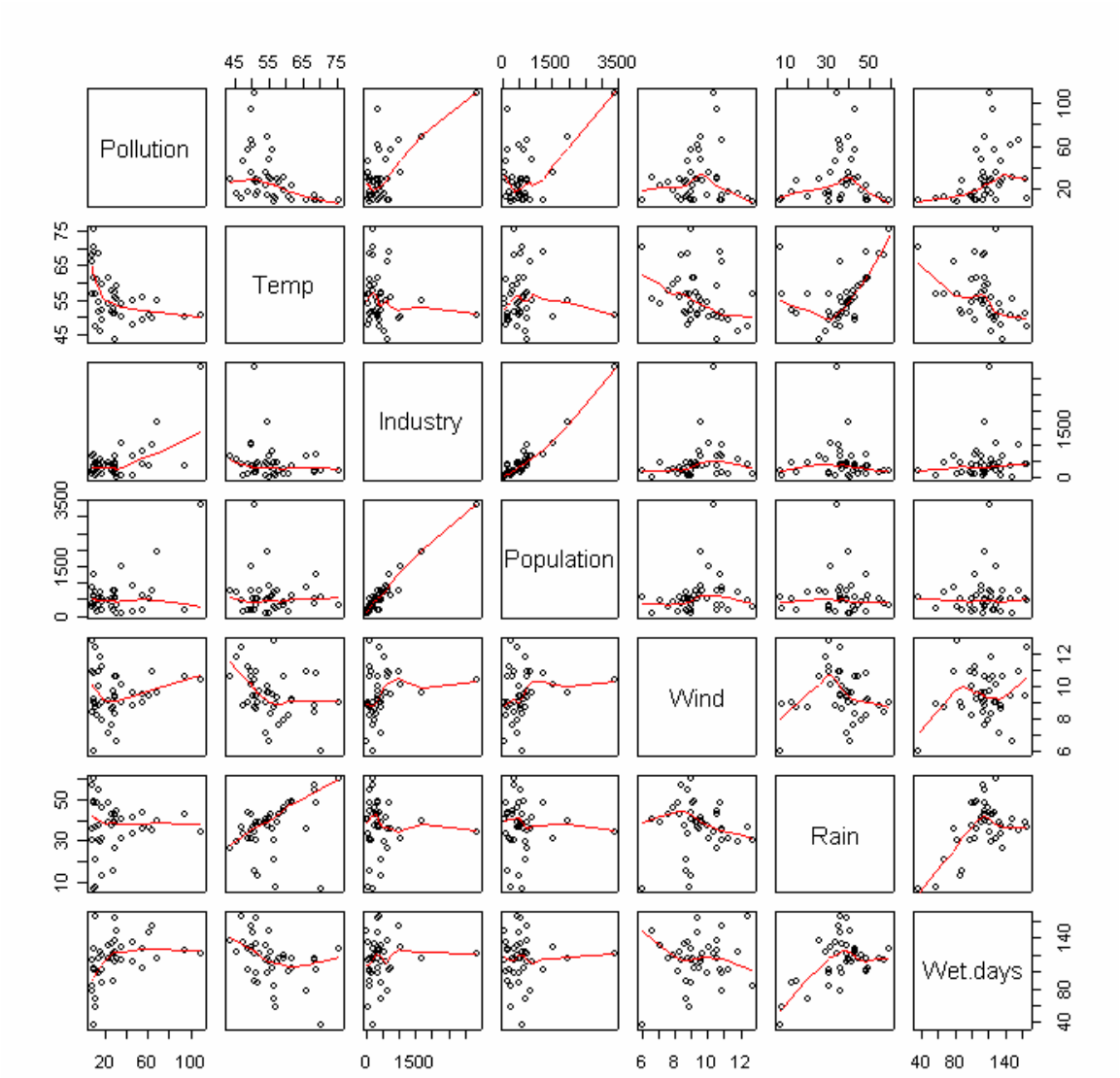
In the next example we introduce two new difficulties: more explanatory variables, and fewer data points. It is another air pollution data frame, but the response variable in this case is sulphur dioxide concentration. There are 6 continuous explanatory variables:

```
pollute<-read.table("c:\\temp\\sulphur.dioxide.txt",header=T)
attach(pollute)
names(pollute)
```

```
[1] "Pollution"  "Temp"       "Industry"   "Population" "Wind"
[6] "Rain"       "Wet.days"
```

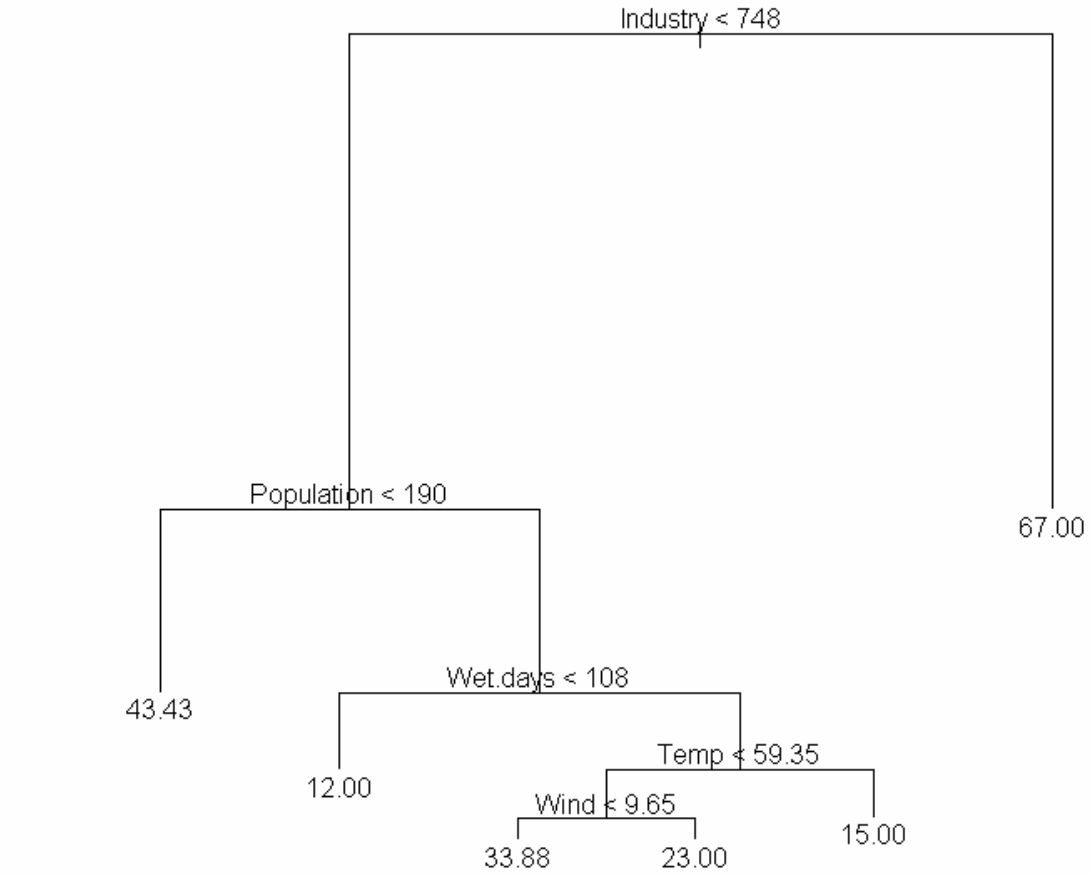
Here are the 36 scatter plots:

```
pairs(pollute,panel=panel.smooth)
```



This time, let's begin with the tree model rather than the generalized additive model. A look at the pairs plots suggests that interactions may be more important than non-linearity in this case.

```
library(tree)
model<-tree(Pollution~.,data=pollute)
plot(model)
text(model)
```



This is interpreted as follows. The most important explanatory variable is Industry, and the threshold value separating low and high values of Industry is 748. The right hand branch of the tree indicates the mean value of air pollution for high levels of industry (67.00). The fact that this limb is unbranched means that no other variables explain a significant amount of the variation in pollution levels for high values of Industry. The left-hand limb does not show the mean values of pollution for low values of industry, because there are other significant explanatory variables. Mean values of pollution are only shown at the extreme ends of branches. For low values of Industry, the tree shows us that Population has a significant impact on air pollution. At low values of Population (<190) the mean level of air pollution was 43.43. For high values of Population, the number of Wet.days is significant. Low numbers of wet days (< 108) have mean pollution levels of 12.00 while Temperature has a significant impact on pollution for places where the number of wet days is large. At high temperatures (> 59.35 ‘F) the mean pollution level was 15.00 while at lower temperatures the run of Wind is important. For still air (Wind < 9.65) pollution was higher (33.88) than for higher wind speeds (23.00).

The virtues of tree-based models are numerous:

- they are easy to appreciated and to describe to other people
- the most important variables stand out
- interactions are clearly displayed
- non-linear effects are captured effectively
- the complexity of the behaviour of the explanatory variables is plain to see

We conclude that the interaction structure is highly complex. We shall need to carry out the linear modelling with considerable care.

Start with some elementary calculations. With 6 explanatory variables, how many interactions might we fit? Well, there are 5 + 4 + 3 + 2 + 1 = 15 two-way interactions for a start. Plus 20 three-way, 15 four-way and 6 five-way interactions, plus one six-way interaction for good luck. Then there are quadratic terms for each of the 6 explanatory variables. So we are looking at about 70 parameters that might be estimated from the data. But how many data points have we got?

```
length(Pollution)
```

```
[1] 41
```

Oops! We are planning to estimate almost twice as many parameters as there are data points. That's taking over-parameterization to new heights. We already know that you can't estimate more parameter values than there are data points (i.e. a maximum of 41 parameters). But we also know that when we fit a saturated model to the data, it has no explanatory power (there are no degrees of freedom, so the model, by explaining everything, ends up explaining nothing at all). There is a useful rule of thumb: *don't try to estimate more than n/3 parameters during a multiple regression*. In the present case $n = 41$ so the rule of thumb is suggesting that we restrict ourselves to estimating about $41/3 \approx 13$ parameters at any one time. We know from the tree model that the interaction structure is going to be complicated so we shall concentrate on that. We begin, therefore, by looking for curvature, to see if we can eliminate this:

```
model1<-
lm(Pollution~Temp+I(Temp^2)+Industry+I(Industry^2)+Population+I(Populatio
n^2)+Wind+I(Wind^2)+Rain+I(Rain^2)+Wet.days+I(Wet.days^2))
summary(model1)
```

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)     -6.641e+01  2.234e+02  -0.297  0.76844
Temp             5.814e-01  6.295e+00   0.092  0.92708
I(Temp^2)       -1.297e-02  5.188e-02  -0.250  0.80445
Industry         8.123e-02  2.868e-02   2.832  0.00847 **
I(Industry^2)   -1.969e-05  1.899e-05  -1.037  0.30862
Population      -7.844e-02  3.573e-02  -2.195  0.03662 *
I(Population^2)  2.551e-05  2.158e-05   1.182  0.24714
Wind             3.172e+01  2.067e+01   1.535  0.13606
I(Wind^2)       -1.784e+00  1.078e+00  -1.655  0.10912
Rain             1.155e+00  1.636e+00   0.706  0.48575
I(Rain^2)       -9.714e-03  2.538e-02  -0.383  0.70476
```

```
Wet.days        -1.048e+00  1.049e+00  -0.999  0.32615
I(Wet.days^2)    4.555e-03  3.996e-03   1.140  0.26398

Residual standard error: 14.98 on 28 degrees of freedom
Multiple R-Squared: 0.7148,     Adjusted R-squared: 0.5925
F-statistic: 5.848 on 12 and 28 DF,  p-value: 5.868e-005
```

So that's our first bit of good news. There is no evidence of curvature for any of the 6 explanatory variables. Only the main effects of Industry and Population are significant in this (over-parameterized) model. Now we need to consider the interaction terms. We do not fit interaction terms without both the component main effects, so we can not fit all the two-way interaction terms at the same time (that would be 15 + 6 = 21 parameters; well above the rule of thumb value of 13). One approach is to fit the interaction terms in randomly selected pairs. With all 6 main effects, we can afford to try 13 – 6 = 7 interaction terms at a time. We'll try this. Make a vector containing the names of the 15 two-way interactions:

```
interactions<-c("ti","tp","tw","tr","td","ip","iw","ir","id","pw","pr","pd","wr","wd","rd")
```

Now shuffle the interactions into random order using sample without replacement:

```
sample(interactions)
```

```
[1] "wr" "wd" "id" "ir" "rd" "pr" "tp" "pw" "ti" "iw" "tw" "pd" "tr" "td" "ip"
```

It would be sensible and pragmatic to test the two-way interactions in 3 models, each containing 5 different two-way interaction terms:

```
model2<-
lm(Pollution~Temp+Industry+Population+Wind+Rain+Wet.days+Wind:Rain+
Wind:Wet.days+Industry:Wet.days+Industry:Rain+Rain:Wet.days)
model3<-
lm(Pollution~Temp+Industry+Population+Wind+Rain+Wet.days+Population:R
ain+Temp:Population+Population:Wind+Temp:Industry+Industry:Wind)
model4<-
lm(Pollution~Temp+Industry+Population+Wind+Rain+Wet.days+Temp:Wind+
Population:Wet.days+Temp:Rain+Temp:Wet.days+Industry:Population)
```

Extracting only the interaction terms from the 3 models, we see:

```
Industry:Rain     -1.616e-04  9.207e-04  -0.176 0.861891
Industry:Wet.days  2.311e-04  3.680e-04   0.628 0.534949
Wind:Rain          9.049e-01  2.383e-01   3.798 0.000690 ***
Wind:Wet.days     -1.662e-01  5.991e-02  -2.774 0.009593 **
Rain:Wet.days      1.814e-02  1.293e-02   1.403 0.171318

Temp:Industry   -1.643e-04  3.208e-03  -0.051   0.9595
Temp:Population  1.125e-03  2.382e-03   0.472   0.6402
Industry:Wind    2.668e-02  1.697e-02   1.572   0.1267
Population:Wind -2.753e-02  1.333e-02  -2.066   0.0479 *
Population:Rain  6.898e-04  1.063e-03   0.649   0.5214

Temp:Wind            1.261e-01  2.848e-01   0.443  0.66117
Temp:Rain           -7.819e-02  4.126e-02  -1.895  0.06811 .
```

```
Temp:Wet.days        1.934e-02  2.522e-02   0.767  0.44949
Industry:Population  1.441e-06  4.178e-06   0.345  0.73277
Population:Wet.days  1.979e-05  4.674e-04   0.042  0.96652
```

The next step might be to put all of the significant or close-to-significant interactions into the same model, and see which survive:

```
model5<-
lm(Pollution~Temp+Industry+Population+Wind+Rain+Wet.days+Wind:Rain+
Wind:Wet.days+Population:Wind+Temp:Rain)
summary(model5)
```

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)     323.054546 151.458618   2.133 0.041226 *
Temp             -2.792238   1.481312  -1.885 0.069153 .
Industry          0.073744   0.013646   5.404 7.44e-06 ***
Population        0.008314   0.056406   0.147 0.883810
Wind            -19.447031   8.670820  -2.243 0.032450 *
Rain             -9.162020   3.381100  -2.710 0.011022 *
Wet.days          1.290201   0.561599   2.297 0.028750 *
Temp:Rain         0.017644   0.027311   0.646 0.523171
Population:Wind  -0.005684   0.005845  -0.972 0.338660
Wind:Rain         0.997374   0.258447   3.859 0.000562 ***
Wind:Wet.days    -0.140606   0.053582  -2.624 0.013530 *
```

We certainly don't need Temp:Rain

```
model6<-update(model5,~. – Temp:Rain)
```

or Population:Wind

```
model7<-update(model6,~. – Population:Wind)
```

All the terms in model7 are significant. Time for a check on the behaviour of the model:

```
plot(model7)
```

That's not bad at all. But what about the higher-order interactions? One way to proceed is to specify the interaction level using ^3 in the model formula, but if you do this, you will find that we run out of degrees of freedom straight away. A pragmatic option is to fit three way terms for the variables that already appear in 2-way interactions: in our case, that is just one term: Wind:Rain:Wet.days

```
model8<-update(model7,~. + Wind:Rain:Wet.days)
summary(model8)
```

```
Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)        278.464474  68.041497   4.093 0.000282 ***
Temp                -2.710981   0.618472  -4.383 0.000125 ***
Industry             0.064988   0.012264   5.299  9.1e-06 ***
Population          -0.039430   0.011976  -3.293 0.002485 **
```

```
Wind                -7.519344   8.151943  -0.922 0.363444
Rain                -6.760530   1.792173  -3.772 0.000685 ***
Wet.days             1.266742   0.517850   2.446 0.020311 *
Wind:Rain            0.631457   0.243866   2.589 0.014516 *
Wind:Wet.days       -0.230452   0.069843  -3.300 0.002440 **
Wind:Rain:Wet.days   0.002497   0.001214   2.056 0.048247 *

Residual standard error: 11.2 on 31 degrees of freedom
Multiple R-Squared: 0.8236,     Adjusted R-squared: 0.7724
F-statistic: 16.09 on 9 and 31 DF,  p-value: 2.231e-009
```

That's enough for now. I'm sure you get the idea. Multiple regression is difficult, time consuming, and always vulnerable to subjective decisions about what to include and what to leave out. The linear modelling confirms the early impression from the tree model: for low levels of industry, the SO_2 level depends in a simple way on population (people tend to want to live where the air is clean) and in a complicated way on daily weather (the 3-way interaction between wind, total rainfall and the number of wet days (i.e. on rainfall intensity)). Note that the relationship between pollution and population in the initial scatterplot suggested a positive correlation between these two variables, not the negative relationship we discovered by statistical modelling. This is one of the great advantages of multiple regression.

Common problems arising in multiple regression

- differences in the measurement scales of the explanatory variables, leading to large variation in the sums of squares and hence to an ill-conditioned matrix
- multicollinearity is which there is a near linear relation between two of the explanatory variables leading to unstable parameter estimates
- rounding errors during the fitting procedure
- non-independence of groups of measurments
- temporal or spatial correlation amongst the explanatory variables
- pseudoreplication and a host of others (see Wetherill et al. 1986 for a detailed discussion).

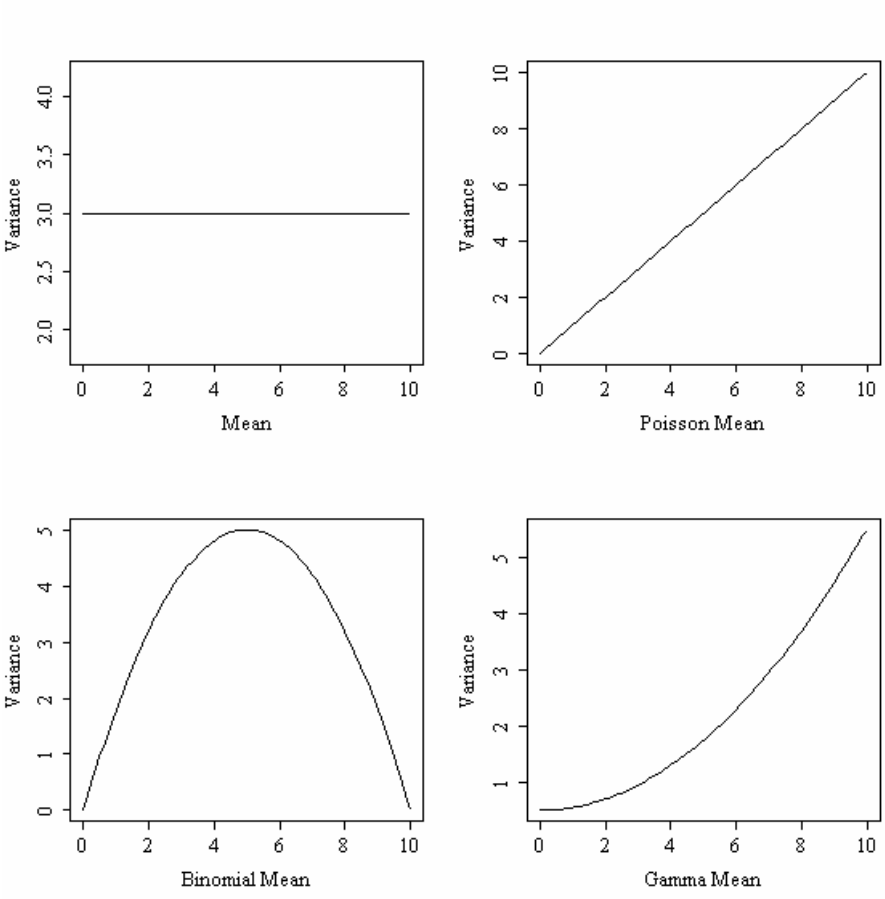
STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

9. GENERALISED LINEAR MODELS

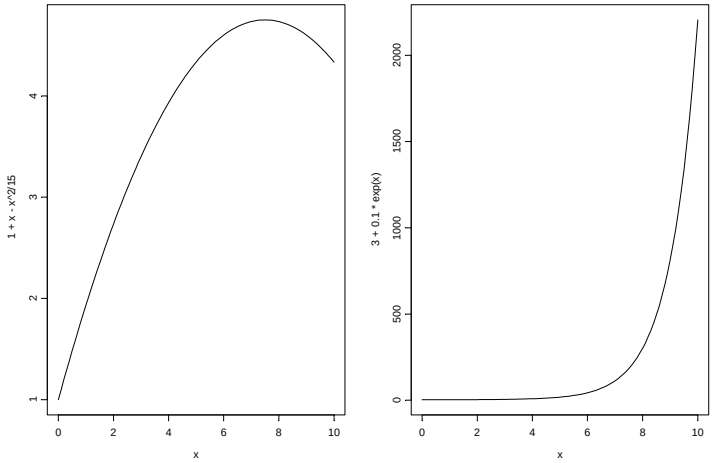
So far, we have assumed that the variance is constant and that the errors are normally distributed. For many kinds of data, one or both of these assumptions is wrong, and we need to be able to deal with this if our analysis is to be unbiased and our interpretations scientifically correct. In count data, for example, where the response variable is an integer and there are often lots of zero's in the data frame, the variance may increase linearly with the mean. With proportion data, where we have a count of the number of failures of an event as well as the number of successes, the variance will be a n-shaped function of the mean. Where the response variable follows a gamma distribution (e.g. in data on time-to-death) the variance increases faster than linearly with the mean. So our assumption has been like the top left panel (where variance is constant) but the data are often like one of the other three panels:



The way to deal with all these problems in a single theoretical framework was discovered by John Nelder who christened the technique Generalised Linear Models (or glims for short). The directive to fit a Generalised Linear Model in S-Plus is **glm**. It is used in exactly the same way as the model fitting directives that are now familiar to you: **aov** and **lm**. There are yet more ways of fitting models that you can discover later if you want to (Generalised Additive Models (**gam**), non-parametric surface fitting models (**loess**), tree models (**tree**) and so on).

A common misconception about generalised linear models (hereafter **glm**'s) is that linear models involve a straight-line relationship between the response variable and the explanatory variables. This is not the case, as you can see from these two linear models. In the plot command use "l" = "lower case L" to generate a line:

```
par(mfrow=c(1,2))
x<-seq(0,10,0.1)
plot(x,1+x-x^2/15,type="l")
plot(x,3+0.1*exp(x),type="l")
```



The definition of a linear model is an equation that contains mathematical variables, parameters and random variables that is *linear in the parameters and in the random variables*. What this means is that if *a*, *b* and *c* are parameters then obviously

$$y = a + bx$$

is a linear model, but so is

$$y = a + bx - cx^2$$

because x^2 can be replaced by z which gives a linear relationship

$$y = a + bx + cz$$

and so is

$$y = a + be^x$$

because we can create a new variable $z = \exp(x)$, so that

$$y = a + bz$$

Some models are non-linear but can be readily linearized by transformation. For example:

$$y = \exp(a + bx)$$

on taking logs of both sides, becomes

$$\ln y = a + bx$$

This kind of relationship is handled in a glm by specifying the log link (see below).

Again, the much-used asymptotic relationship (known in different disciplines as Michaelis-Menten, Briggs-Haldane or Holling 'disk equation') given by:

$$y = \frac{x}{b + ax}$$

is non-linear in the parameter a, but it is readily linearized by taking reciprocals:

$$\frac{1}{y} = a + b\frac{1}{x}$$

Generalised linear models handle this family of equations by a transformation of the explanatory variable ($z = 1/x$) and using the reciprocal link (often, for data like this, associated with gamma rather than normal errors).

Other models are *intrinsically non-linear* because there is no transformation that can linearize them in all the parameters. Some important examples include the hyperbolic function

$$y = a + \frac{b}{c + x}$$

and the asymptotic exponential

$$y = a(1 - be^{-cx})$$

where both models are non-linear unless the parameter c is known in advance. In cases like this, a **glm** is unable to estimate the full set of parameters, and we need to resort to non linear modelling.

Generalised linear models

A generalised linear model has three important properties:

- the *error structure*
- the *linear predictor*
- the *link function*

These are all likely to be unfamiliar concepts. The ideas behind them are straightforward, however, and it is worth learning what each of the concepts involves.

The error structure

Up to this point, we have dealt with the statistical analysis of data with normal errors. In practice, however, many kinds of data have non-normal errors: for example:

- errors that are strongly skewed
- errors that are kurtotic
- errors that are strictly bounded (as in proportions)
- errors that can not lead to negative fitted values (as in counts)

In the past, the only tools available to deal with these problems were transformation of the response variable or the adoption of non-parametric methods. A **glm** allows the specification of a variety of different error distributions:

- Poisson errors, useful with count data
- binomial errors, useful with data on proportions
- gamma errors, useful with data showing a constant coefficient of variation
- exponential errors, useful with data on time to death (survival analysis)

The error structure is defined by means of the **family** directive, used as part of the model formula like this:

```
glm( y ~ z, family = poisson )
```

which means that the response variable *y* has Poisson errors. Or

```
glm(y ~ z, family = binomial )
```

which means that the response is binary, and the model has binomial errors. As with previous models, the explanatory variable *z* can be continuous (leading to a regression analysis) or categorical (leading to an anova-like procedure called analysis of deviance; see below).

The linear predictor

The structure of the model relates each observed y-value to a predicted value. The predicted value is obtained *by transformation of the value emerging from the linear*

predictor. The linear predictor, η (eta), is a linear sum of the effects of one or more explanatory variables, x_j:

$$\eta_i = \sum_{j=1}^{p} x_{ib}\beta_j$$

where the *x*'s are the values of the *p* different explanatory variables, and the β's are the (usually) unknown parameters to be estimated from the data. The right hand side of the equation is called the *linear structure*.

There are as many terms in the linear predictor as there are parameters, *p*, to be estimated from the data. Thus with a simple regression, the linear predictor is the sum of two terms; the intercept and the slope. With a 1-way ANOVA with 4 treatments, the linear predictor is the sum of 4 terms; a mean for each level of the factor. If there are covariates in the model, they add one term each to the linear predictor (the slope of the relationship). Each interaction term in a factorial ANOVA adds one or more parameters to the linear predictor, depending upon the degrees of freedom of each factor (e.g. 3 extra parameters for the interaction between a 2-level factor and a 4-level factor (2-1) x (4-1) = 3).

Fit

To determine the fit of a given model, a **glm** evaluates the linear predictor for each value of the response variable, then compares the observed value with a *back-transformed* value of the linear predictor. The transformation to be employed is specified in the link function (see below). The fitted value is computed by applying the reciprocal of the link function, in order to get back to the original scale of measurement of the response variable. Thus, with a log link, the fitted value is the antilog of the linear predictor, and with the reciprocal link, the fitted value is the reciprocal of the linear predictor.

The link function

One of the difficult things to grasp about **glm** is the relationship between the values of the response variable (as measured in the data and predicted by the model in fitted values) and the linear predictor. The thing to remember is that the *link function relates the mean value of y to its linear predictor*. In symbols, this means that:

$$\eta = g(\mu)$$

which is simple, but needs thinking about. The linear predictor, η (eta), emerges from the linear model as a sum of the terms for each of the *p* parameters. *This is not a value of y* (except in the special case of the *identity link* that we have been using (implicitly) up to now). The value of η is obtained by transforming the value of *y* by the link function, and the predicted value of *y* is obtained by applying the inverse link function to η.

The most frequently used link functions are shown below. An important criterion in the choice of link function is to ensure that the fitted values stay within reasonable bounds. We would want to ensure, for example, that counts were all greater than or equal to zero (negative count data would be nonsense). Similarly, if the response variable was the proportion of animals that died, then the fitted values would have to lie between zero and one (fitted values greater than 1 or less than 0 would be meaningless). In the first case, a log link is appropriate because the fitted values are antilogs of the linear predictor, and all antilogs are greater than or equal to zero. In the second case, the logit link is appropriate because the fitted values are calculated as the antilogs of the log-odds, log(p/q).

By using different link functions, the performance of a variety of models can be compared directly. The total deviance is the same in each case and we can investigate the consequences of altering our assumptions about precisely how a given change in the linear predictor brings about a response in the fitted value of y. The most appropriate link function is the one which produces the minimum residual deviance.

The log link

The log link has many uses, but the most frequent are:

- for count data, where negative fitted values are prohibited
- for explanatory variables that have multiplicative effects, where the log link introduces additivity (remember that “the log of times is plus”)

The model parameters inspected with **summary(model)** are in natural logarithms, and the fitted values are the natural antilogs (**exp**) of the linear predictor.

The logit link

This is the link used for proportion data, and the logit link is generally preferred to the more old fashioned **probit** link. If a fraction *p* of the insects in a bioassay died, then a fraction *q* = (1 - *p*) must have survived out of the original cohort of *n* animals. The logit link is

$$\text{logit} = \ln\left(\frac{p}{q}\right)$$

This is beautifully simple, and it ensures that the fitted values are bounded both above and below (the predicted proportions may not be greater than 1 or less than 0). The details of how the logit link linearizes proportion data are explained in Practical 10.

The logit link does, however, make it a little tedious to calculate *p* from the parameter estimates. Suppose we had a predicted value *x* on the logit scale of -0.328. To get the value of *p* we evaluate (taking care with the signs)

$$p = \frac{1}{1 + e^{-x}}$$

which gives $p = 0.42$. Note that confidence intervals on p will be asymmetric when back-transformed, and it is good practice to draw barcharts and error bars on the logit scale rather than the proportion (or percentage) scale to avoid this problem.

Other link functions

Two other commonly used link functions are the **probit** and the **complementary log-log** links. They are used in bioassay and in dilution analysis respectively, and examples of their use are discussed later.

The names of the links that can be used in R include:

`"logit"`, `"probit"`, `"cloglog"`, `"identity"`, `"log"`, `"sqrt"`, `"1/mu^2"`, `"inverse"`

Use of probits for bioassay is largely traditional, because probit paper used to be available for converting percentage mortality to a linear scale against log dose. Since computers have become widely available the need for the probit transformation

$$\frac{y_i}{n_i} = \Phi(\eta_i) + \varepsilon_i$$

has declined. The proportion responding (y/n) is linked to the linear predictor by $\Phi(.)$, the unit normal probability integral. Because the logit is so much simpler to interpret, and because the results of modelling with the two transformations are almost always identical, the logit link function is nowadays recommended for bioassay work, even though probits are based on a reasonable distributional argument for the tolerance levels of individuals.

The complementary log-log link:

$$\theta = \ln[-\ln(1-p)]$$

is not symmetrical about p=0.5 and is often used in simple dilution assay. If the proportion of tubes containing bacteria p is related to dilution x like this:

$$p = 1 - e^{-\lambda x}$$

then the complementary log-log transformation gives

$$\eta = \ln[-\ln(1-p)] = \ln\lambda + \ln x$$

which means that the linear predictor has a slope of 1 when plotted against $\ln(x)$. We fit the model, therefore, with $\ln(x)$ as an offset, and **glm** estimates the maximum likelihood value of $\ln(\lambda)$.

The complementary log-log link should be assessed during model criticism for binary data and for data on proportional responses (see Practical 10). It will sometimes lead to a lower residual deviance than the symmetrical logit link.

Canonical link functions

The canonical link functions are the default options employed when a particular error structure is specified in the **family** directive in the model formula. Omission of a **link** directive means that the following settings are used:

Error	Canonical link
normal	identity
Poisson	log
binomial	logit
gamma	reciprocal

You should try to memorise these canonical links and to understand why each is appropriate to its associated error distribution.

The likelihood function

The concept of maximum likelihood is unfamiliar to most non-statisticians. Fortunately, the methods that scientists have encountered in linear regression and traditional ANOVA (i.e. least squares) are the maximum likelihood estimators when the data have normal errors and the model has an identity link. For other kinds of error structure and different link functions, however, the methods of least squares do not give unbiased parameter estimates, and maximum likelihood methods are preferred. It is easiest to see what maximum likelihood involves by working through two simple examples based on the binomial and Poisson distributions.

The binomial distribution

Suppose we have carried out a single trial, and have found $r = 5$ parasitised animals out of a sample of $n = 9$ insects. Our intuitive estimate of the proportion parasitised is $5/9 = r/n = 0.555$. What is the maximum likelihood estimate of the proportion parasitised? With n=9 and r=5 the formula for the binomial looks like this:

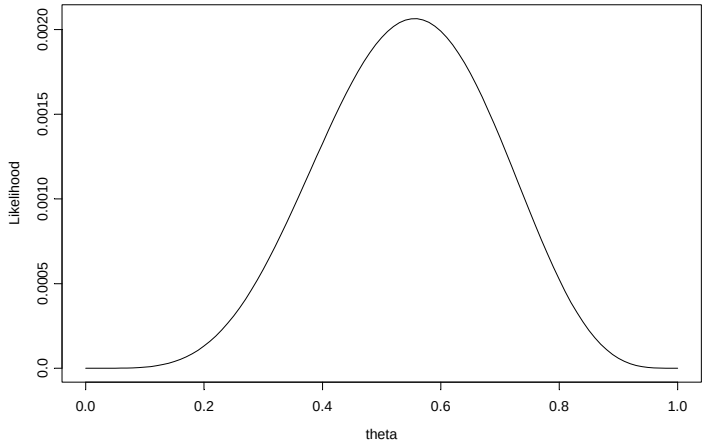
$$P(5) = \left(\frac{9!}{(9-5)! \times 5!} \right) \theta^5 (1-\theta)^{(9-5)}$$

Now the likelihood L does not depend upon the combinatorial part of the formula, because θ , the parameter we are trying to estimate, does not appear there. This simplifies the problem, because all we need to do now is to find the value of θ which maximises the likelihood

$$L(\theta) = \theta^5 (1-\theta)^{(9-5)}$$

To do this we might plot L(θ) against θ like this

```
theta<-seq(0,1,.01)
par(mfrow=c(1,1))
plot(theta,theta^5*(1-theta)^4,type="l",ylab="Likelihood")
```



from which it is clear that the maximum likelihood occurs at = *r/n* = 5/9 = 0.555. It is reassuring that our intuitive estimate of the proportion parasitised is *r/n* as well.

A more general way to find the maximum likelihood estimate of θ is to use calculus. We need to find the derivative of the likelihood with respect to θ, then set this to zero, and solve for θ. In the present case it is easier to work with the log of the likelihood. Obviously, the maximum likelihood and the maximum log likelihood will occur at the same value of θ .

$$\text{log likelihood} = r\ln(\theta) + (n-r)\ln(1-\theta)$$

so the derivative of the log likelihood with respect to θ is:

$$\frac{dL(\theta)}{d\theta} = \frac{r}{\theta} - \frac{n-r}{1-\theta}$$

remembering that the derivative of $\ln\theta$ is $1/\theta$ and of ln (1-θ) is -1/(1-θ). We set this to zero, rearrange, then take reciprocals to find θ:

$$\frac{r}{\theta} = \frac{n-r}{1-\theta} \quad \text{so} \quad \theta = \frac{r}{n}$$

The maximum likelihood estimate of the binomial parameter is the same as our intuitive estimate.

The Poisson distribution

As a second example, we take the problem of finding the maximum likelihood estimate of μ for a Poisson process in which we observed, say, *r* lightening strikes per parish in *n* parishes, giving a total of $\sum r$ lightening strikes in all. The probability density function for the number of strikes per parish is

$$f(r) = \frac{e^{-\mu}\mu^{r}}{r!}$$

so the initial likelihood is the density function multiplied by itself as many times as there are individual parishes:

$$L(\mu) = \prod_{1}^{n} \frac{e^{-\mu}\mu^{r}}{r!} = e^{-n\mu}\mu^{\sum r}$$

because the constant r! can be ignored. Note that *nr* is replaced by $\sum r$ the observed total number of lightening strikes. Now it is straightforward to obtain the log likelihood:

$$L(\mu) = -n\mu + \sum r \ln \mu$$

The next step is to find the derivative of the log likelihood with respect to μ :

$$\frac{dL(\mu)}{d\mu} = -n + \frac{\sum r}{\mu}$$

We set this to zero, and rearrange to obtain

$$\mu = \frac{\sum r}{n}$$

Again, the maximum likelihood estimator for the single parameter of the Poisson distribution conforms with intuition; it is the mean (in this case, the mean number of lightning strikes per parish).

Maximum likelihood estimation

The object is to determine the values for the parameters of the model that lead to the best fit of the model to the data. It is in the definition of what constitutes 'best' that maximum likelihood methods can differ from the more familiar, least squares estimates. This is how it works

- given the data
- and given our choice of model
- what values of the parameters

- make the observed data most likely ?

The data are sacrosanct, and they tell us what actually happened under a given set of circumstances. It is a common mistake to say 'the data were fitted to the model' as if the data were something flexible, and we had a clear picture of the structure of the model. On the contrary, what we are looking for is the minimal adequate model to describe the data. The model is fit to data, not the other way around. The best model is the model that produces the minimal residual deviance, subject to the constraint that all the parameters in the model should be statistically significant.

You have to specify the model. It embodies your best hypothesis about the factors involved, and the way they are related to the response variable. We want the model to be minimal because of the principle of parsimony, and adequate because there is no point in retaining an inadequate model that does not describe a significant fraction of the variation in the data. It is very important to understand that *there is not one model*; this is one of the common implicit errors involved in traditional regression and anova, where the same models are used, often uncritically, over and over again. In most circumstances, there will be a large number of different, more or less plausible models that might be fit to any given set of data. Part of the job of data analysis is to determine which, if any, of the possible models are adequate, and then, out of the set of adequate models, which is the minimal adequate model. In some cases there may be no single best model and a set of different models may all describe the data equally well.

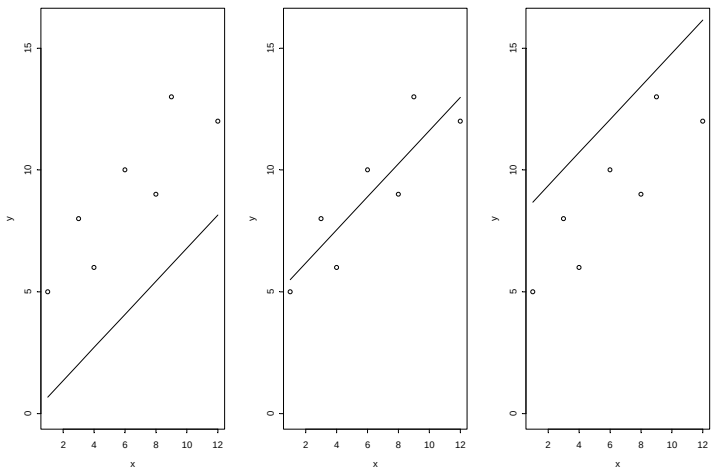
Here are the data:

```
x<-c(1,3,4,6,8,9,12)
y<-c(5,8,6,10,9,13,12)
```

and here is the model

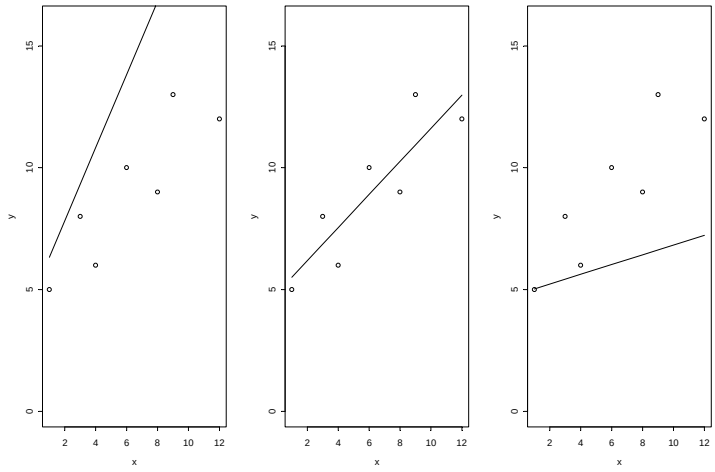
$$y = a + bx$$

Suppose that we know that the slope is 0.68, then the maximum likelihood question can be applied to the intercept *a*. If the intercept were 0 (left graph), would the data be likely?



The answer of course, is no. If the intercept were 8 (right graph) would the data be likely? Again, the answer is obviously no. The maximum likelihood estimate of the intercept is shown in the central graph (its value turns out to be 4.827).

We could have a similar debate about the slope. Suppose we knew that the intercept was 4.827, would the data be likely if the graph had a slope of 1.5 (left graph)?



The answer, of course, is no. What about a slope of 0.2 (right graph)? Again, the data are not at all likely if the graph has such a gentle slope. The maximum likelihood of the data is obtained with a slope of 0.679 (centre graph).

This is not how the procedure is carried out, but it makes the point that we judge the model of the basis *how likely the data would be if the model were correct*. In practice of course, the parameters are estimated simultaneously.

Parameter estimation in generalised linear models

The method of parameter estimation is *iterative, weighted least-squares*. You know about least-squares methods from Practicals 4, 5 and 6. A **glm** is different in that the regression is not carried out on the response variable, y, but on *a linearized version of the link function applied to y*. The *weights* are functions of the fitted values. The procedure is *iterative* because both the adjusted response variable and the weight depend upon the fitted values.

This is how it works (the technical details are on pp 31-34 in McCullagh & Nelder, 1983). Take the data themselves as starting values for estimates of the fitted values. Use this to derive the linear predictor, the derivative of the linear predictor ($d\eta / d\mu$) and the variance function. Then re-estimate the adjusted response variable *z* and the weight W, as follows:

$$z_0 = \eta_0 + (y - \mu_0)\left(\frac{d\eta}{d\mu}\right)_0$$

where the derivative of the link function is evaluated at μ_0 and

$$W_0^{-1} = \left(\frac{d\eta}{d\mu}\right)_0^2 V_0$$

where V_O is the variance function of *y* (see below). Keep repeating the cycle until the changes in the parameter estimates are sufficiently small. It is the difference ($y - \mu_0$) between the data *y* and the fitted values μ_0 that lies at the heart of the procedure. The maximum likelihood parameter estimates are given by

$$\sum W(y - \mu) \frac{d\eta}{d\mu} x_i = 0$$

for each explanatory variable x_i (summation is over the rows of the data frame). For more detail, see McCullagh & Nelder (1989) and Aitkin et al. (1989); a good general introduction to the methods of maximum likelihood is to be found in Edwards (1972).

Deviance: measuring the goodness of fit of a glm.

The fitted values produced by the model are most unlikely to match the values of the data perfectly. The size of the discrepancy between the model and the data is a measure of the inadequacy of the model; a small discrepancy may be tolerable, but a large one will not be. The measure of discrepancy in a **glm** used to assess the goodness of fit of the model to the data is called the *deviance*. Deviance is defined as –2 times the difference in log likelihood between the current model and a saturated model (i.e. a model that fits the data perfectly). Because the latter does not depend on the parameters of the model, minimising the deviance is the same as maximising the likelihood.

Deviance is estimated in different ways for different families within glm. Numerical examples of the calculation of deviance for different glm families are in Practical 11 (Poisson errors), Practical 10 (binomial errors) and Practical 12 (gamma errors).

Table. Deviance formulas for different families of glm. y is observed data, $\bar{y}$ mean value of y, μ fitted values of y from the maximum likelihood model, n is the binomial denominator in a binomial error glm.

Family (Error structure)	Deviance
Normal	$\sum (y - \bar{y})^2$
Poisson	$2\sum y \ln(y / \mu) - (y - \mu)$
Binomial	$2\sum y \ln(y / \mu) + (n - y) \ln(n - y) / (n - \mu)$
Gamma	$2\sum (y - \mu) / y - \ln(y / \mu)$
Inverse Gaussian	$\sum (y - \mu)^2 / (\mu^2 y)$

The following 3 practicals deal with the 3 main applications of **glm**'s

- proportion data using binomial errors
- count data using Poisson errors
- survival data using various error distributions

In this practical, we investigate the use of a range of different **link functions**

```
timber<-read.table("c:\\temp\\timber.txt", header=T)
attach(timber)
names(timber)
```

```
[1] "volume" "girth"  "height"
```

We begin with data inspection: how does timber volume depend upon girth and height? Volume of a cylinder is

$$v = \pi r^2 h$$

(i.e. the cross-sectional area times the height). This means that we expect volume to be proportional to height and proportional to the square of girth (since $g = 2\pi r$). Also, we note that the model for volume is multiplicative rather than additive. Taking logs of both sides gives

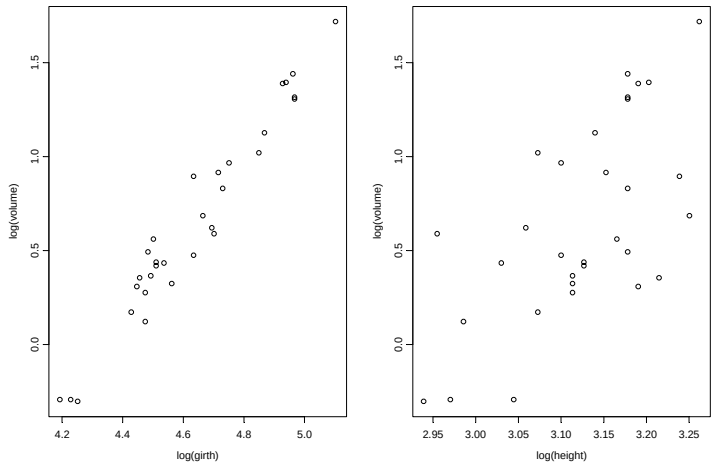
$$\ln(v) = \ln(\pi) + 2\ln(r) + \ln(h)$$

which is useful because it makes the relationship linear and additive. If we regress log volume on log height we expect the slope of the graph to be 1. If we regress log volume on girth we expect the slope of the graph to be 2. And if the timber really is cylindrical (rather than tapered or conical) then the intercept will be a function of pi. Let's rewrite the equation with girth in place of radius and see if we can predict what the intercept of a multiple regression should be:

$$\ln(v) = \ln(\pi) + 2\ln(r) + \ln(h) = \ln(\pi) + 2\ln(g/2\pi) + \ln(h) = \ln(1/4\pi) + 2\ln(g) + \ln(h)$$

so we expect the intercept of a log-log plot to be $\ln(1/4\pi) =$ -2.531024. Let's see what these data look like:

```
par(mfrow=c(1,2))
plot(log(girth),log(volume))
plot(log(height),log(volume))
```



The relationship between timber volume and girth is very close (left), but the relationship with height is much more noisy (right). It is sensible to convert girth into metres, so that all of the units are in the same currency:

```
girth<-girth/100
```

We begin by testing the prediction that the slopes of the graph should be 2 and 1 for the regressions of log volume on log girth and log height respectively:

```
model1<-lm(log(volume)~log(girth)+log(height))
summary(model1)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.89938    0.63767  -4.547 9.56e-05 ***
log(girth)   1.98267    0.07503  26.426  < 2e-16 ***
log(height)  1.11714    0.20448   5.463 7.83e-06 ***
```

The values are close to our predictions: log(girth) has a slope of 1.98 against a predicted value of 2.0, log(height) has 1.12 against a predicted value of 1.0 and the intercept is -2.8994 against a predicted value of -2.531024. A simple test of whether the observed and predicted values are significantly different is to carry out 3 t-tests, dividing the differences between observed and predicted parameter values by their standard errors:

```
(2-1.9827)/0.075
```

```
[1] 0.2306667
```

```
(1-1.1171)/0.2045
```

```
[1] -0.5726161
```

```
(2.8994 -2.531024)/0.6377
```

```
[1] 0.5776635
```

So there is no evidence of any significant difference there: none of the t values is bigger than 2.0. An alternative is to test whether a different model explains the data any less well than the full regression. In this alternative model, we constrain the two slopes to be exactly 2.0 and1.0 by the use of an **offset**, like this:

```
model1<-glm(log(volume)~log(girth)+log(height))
model2<-glm(log(volume)~offset(2*log(girth)+log(height)))
```

Now we can compare the 3 models using **anova** to see how the full model and the model with the offset constraining the 2 slopes to exactly 2.0 and 1.0 compare with

```
anova(model1,model2,test="F")
```

```
Analysis of Deviance Table

Model 1: log(volume) ~ log(girth) + log(height)
Model 2: log(volume) ~ offset(2 *log(girth)+log(height))

  Resid. Df Resid. Dev Df  Deviance      F Pr(>F)
1        28   0.185548
2        30   0.187772 -2 -0.002223 0.1677 0.8464
```

Note that in order for **offset** to work properly, we need to fit the models as **glm** rather than **lm** (an offset is strictly redundant in a Gaussian model, because in principle one can work directly with the residuals).

The **anova** shows that the reduction in explanatory power of the offset model is only 0.002223 compared with the full model, despite a saving of 2 degrees of freedom. This is assessed by an F test as (0.002223/2) / (0.185548/28) = 0.1677 compared with a value of 3.34 in tables (p = 0.8464).

What about the shape of the timber? Theory might have predicted that the taper on the logs would be enough to prefer a conical model to a cylindrical model. A cone would have an intercept of –3.630 on a log-log plot, while as we have seen, a cylinder would have –2.531. We can specify this full model in an offset and fit the model without an intercept (by specifying –1 in the model formula to remove the intercept):

```
model3<-glm(log(volume)~-1+offset(-2.531 + 2*log(girth)+log(height)))
anova(model1,model2,model3,test="F")
```

```
Analysis of Deviance Table

Model 1: log(volume) ~ log(girth) + log(height)
Model 2: log(volume) ~ offset(2 * log(girth)+log(height))
Model 3: log(volume) ~ -1 +
                       offset(-2.531+2*log(girth)+log(height))

  Resid. Df Resid. Dev Df  Deviance      F Pr(>F)
1        28   0.185548
2        30   0.187772 -2 -0.002223 0.1677 0.8464
3        31   0.188057 -1 -0.000285 0.0430 0.8372
```

The model3 in which we specify all 3 parameters is not significantly worse than model1 in which all 3 parameters were estimated from the data. The data therefore provide no support for a model for the volume of this timber that is any more complicated than a cylinder.

Transformations of the response and explanatory variables

We can now assess the explanatory power of a variety of different transformations. The first model transformed both explanatory variables to a log scale, making the model additive and dimensionally consistent. A different way of making the model dimensionally consistent is to take the cube root of the volume as the response variable, in which case all of the variables on both sides of the equation have units of metres.

```
model4<-glm(volume^(0.3333)~girth+height)

summary(model4)
```

```
Coefficients:
             Estimate Std. Error t value  Pr(>|t|)
(Intercept) -0.035478   0.076824  -0.462     0.648
girth        0.791345   0.029453  26.868  < 2e-016 ***
height       0.020104   0.003858   5.211 1.56e-005 ***

    Null deviance: 1.492964  on 30  degrees of freedom
Residual deviance: 0.033371  on 28  degrees of freedom
AIC: -115.88
```

so r^2 is (1.492964 – 0.033371) / 1.492964 = 0.9776478.

Finally, we can use the **quasi** function to allow the use of **anova** to compare different link functions like log, power or reciprocal. Recall that if we transform the response variable in different ways (e.g. log in one case, cube root in another case) then we can not compare the resulting models using anova. Models 5-7 fit the same model

volume~girth+height

using a different link function in each case: model5 uses the cube root link (this is specified as **power(0.333)**), model6 uses the **log** link, while model7 uses the reciprocal link (this is specified as **inverse**).

```
model5<-glm(volume~girth+height,family=quasi(link=power(0.3333)))

model6<-glm(volume~girth+height,family=quasi(link=log))

model7<-glm(volume~girth+height,family=quasi(link=inverse))
```

The beauty of this approach is that we can now compare the models using **anova** because they all have the same response variable (i.e. untransformed volume).

anova(model5,model6,model7,test="F")

```
Analysis of Deviance Table

Model 1: volume ~ girth + height
Model 2: volume ~ girth + height
Model 3: volume ~ girth + height

  Resid. Df Resid. Dev Df Deviance
1        28     0.9655
2        28     1.4291  0  -0.4636
3        28     5.3189  0  -3.8898
```

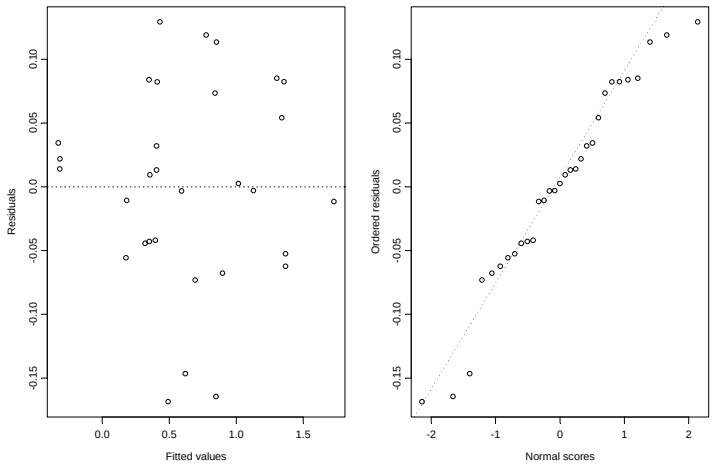
The cube root link is better than the log link (residual deviance of 0.966 compared with 1.429) and both are much better than the inverse link (deviance = 5.319). Note that the **anova** table gives us no indication of how the 3 models differ from one another in their link functions, so we need to be very careful in our book-keeping.

So which is the best model for these data ? We have 3 serious candidates:

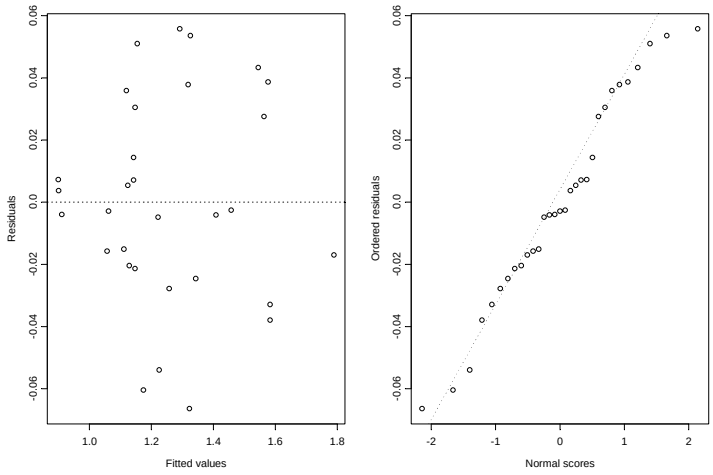
Response	**Model**	**Link**	**r^2**
log(volume)	1: log(girth)+log(volume)	identity	0.9777
(volume)^0.333	4: girth+volume	identity	0.9776
volume	5: girth+volume	power(0.333)	0.9773

We can not compare them using **anova** because they all have different response variables. We can compute their r^2 values, and this suggests that our original log-log transformed model is best (but there is precious little in it). We can inspect their residuals using **plot(model**) : model 1, then model 6, then model 8

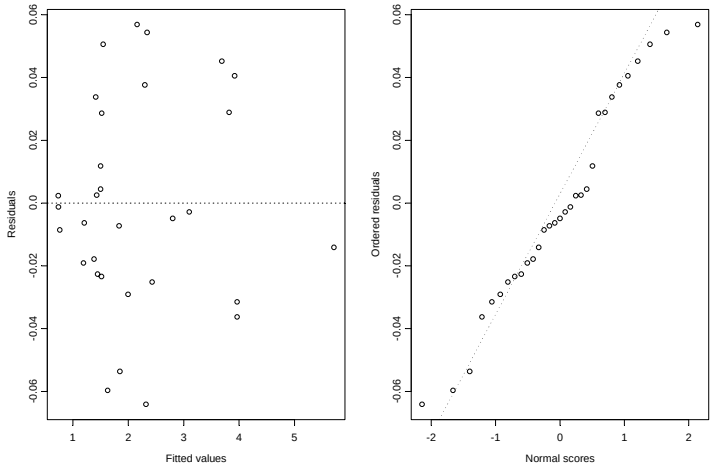
log(volume) ~ log(girth) + log(height)



(volume^0.333) ~ girth + height



volume ~ girth + height, family = quasi(link = power(0.3333))



Again, there is little to choose between them, but the power link has the worst qq-plot (the slope in the central part is shallower than the 1 to 1 line). Overall, be would probably go for the original model, involving log-log transformation, because:

- it is dimensionless on both sides
- it is additive in a way that makes good mathematical sense ($vol \propto g^2 \times h$)
- it has the highest r^2
- its residuals are well behaved

Box Cox transformations

Sometimes it is not clear from theory what the optimal transformation of the response variable should be. In these circumstances, the Box Cox transformation offers a simple empirical solution. The idea is to find the power transformation, λ (lambda), that maximises the likelihood when a specified set of explanatory variables is fit to

$$\frac{y^{\lambda}-1}{\lambda}$$

as the response. The value of lambda can be positive or negative, but it can't be zero (you would get a zero-divide error when the formula was applied to the response variable, y). For the case $\lambda = 0$ the Box Cox transformation is defined as log(y). Suppose that $\lambda = -1$. The formula now looks like this:

$$\frac{y^{-1}-1}{-1} = \frac{\frac{1}{y}-1}{-1} = 1-\frac{1}{y}$$

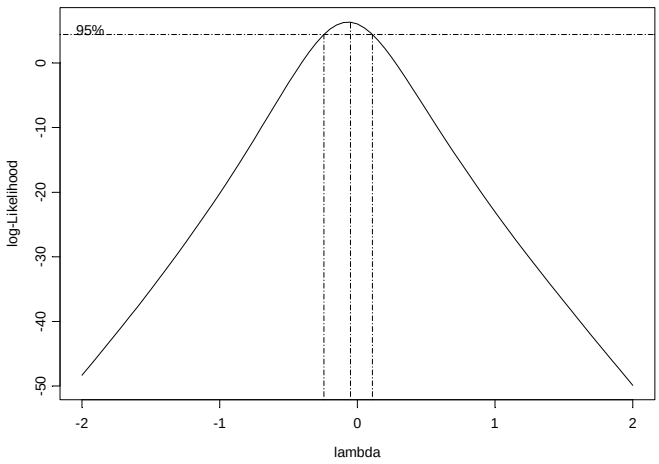
and this quantity is regressed against the explanatory variables and the log likelihood computed.

We start by loading the MASS library of Venables and Ripley:

```
library(MASS)
```

The **boxcox** function is very easy to use: just specify the model formula, and the default options take care of everything else

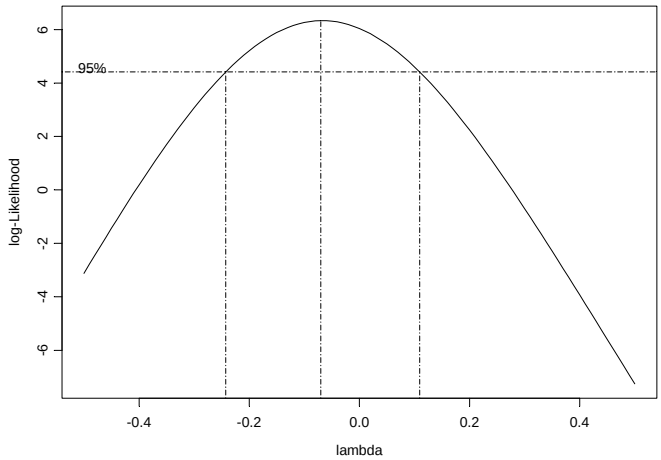
```
boxcox(volume~log(girth)+log(height))
```



It is clear that the optimal value of lambda is close to zero (i.e. the log transformation). We can zoom in to get a more accurate estimate by specifying our

own, non-default, range of lambda values. It looks as if it would be sensible to plot from –0.5 to +0.5:

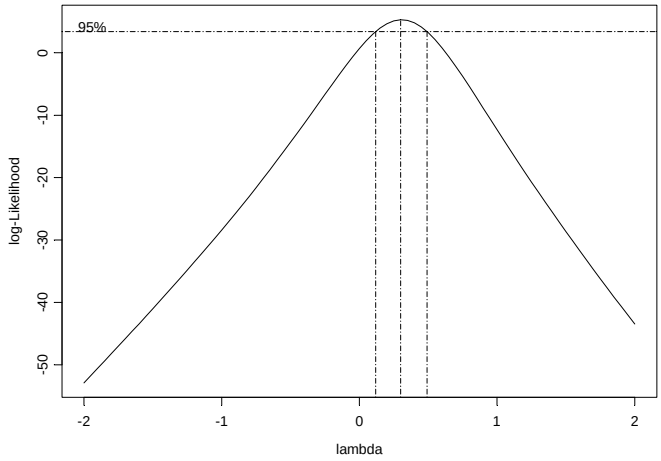
```
boxcox(volume~log(girth)+log(height),lambda=seq(-0.5,0.5,0.01))
```



The likelihood is maximised at lambda about –0.08, but the log likelihood for lambda = 0 is very close to the maximum. This also gives a much more straightforward interpretation, so we would go with that, and model log(volume) as a function of log(girth) and log(height).

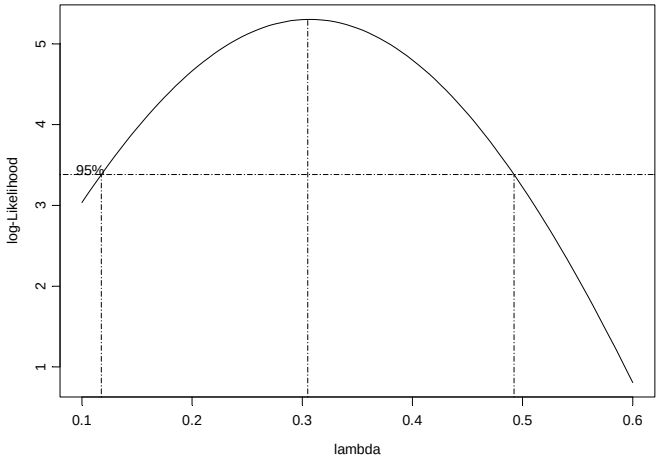
What if we had not log-transformed the explanatory variables? What would have been the optimal transformation of volume in that case? To find out, we re-run the **boxcox** function, simply changing the model formula like this:

```
boxcox(volume~girth+height)
```



We can zoom in from 0.1 to 0.6 like this:

```
boxcox(volume~girth+height,lambda=seq(0.1,0.6,0.01))
```



This suggests that the cube root transformation would be best ($\lambda = 1/3$). Again, this accords with dimensional arguments, since the response and explanatory variables would all have dimensions L in this case.

STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

10. ANALYSING PROPORTION DATA: BINOMIAL ERRORS

An important class of problems involves data on proportions:

- data on percentage mortality
- infection rates of diseases
- proportion responding to clinical treatment
- proportion admitting to particular voting intentions
- data on proportional response to an experimental treatment

What all these have in common is that we know how many of the experimental objects are in one category (say dead, insolvent or infected) and we know how many are in another (say alive, solvent or uninfected). This contrasts with Poisson count data, where we knew how many times an event occurred, but not how many times it did not occur.

We model processes involving proportional response variables in S-Plus by specifying a **glm** with **family=binomial** . The only complication is that whereas with Poisson errors (Practical 11) we could simply say **family=poisson**, with binomial errors we must specify the number of failures as well as the numbers of successes in a 2-vector response variable. To do this we bind together two vectors using **cbind** into a single object, *y*, comprising the numbers of successes and the number of failures. The *binomial denominator, n,* is the total sample, and the

number.of.failures = binomial.denominator – number.of.successes

y <- cbind(number.of.successes, number.of.failures)

The old fashioned way of modelling this sort of data was to use the percentage mortality as the response variable. There are 3 problems with this:

- the errors are not normally distributed
- the variance is not constant
- by calculating the percentage, we lose information of the size of the sample, *n*, from which the proportion was estimated

In S-Plus, we use the *number responding* and the *number not responding* bound together as the response variable. S-Plus then carries out weighted regression, using the individual sample sizes as weights, and the *logit link function* to ensure linearity (as described below).

If the response variable takes the form of a **percentage change** in some continuous measurement (such as the percentage change in weight on receiving a particular diet), then the data should be **arc-sine transformed** prior to analysis, especially if many of the percentage changes are smaller than 20% or larger than 80%. Note, however, that data of this sort are probably better treated by either

- analysis of covariance (see Practical 6), using final weight as the response variable and initial weight as a covariate
- or by specifying the response variable to be a relative growth rates, where the response variable is ln(final weight/initial weight).

There are some proportion data, like **percentage cover**, which are best analysed using conventional models (normal errors and constant variance) following **arc-sine transformation**. The response variable, *y*, measured in radians, is $\sin^{-1}\sqrt{0.01\times p}$ where *p* is cover in %.

Count data on proportions

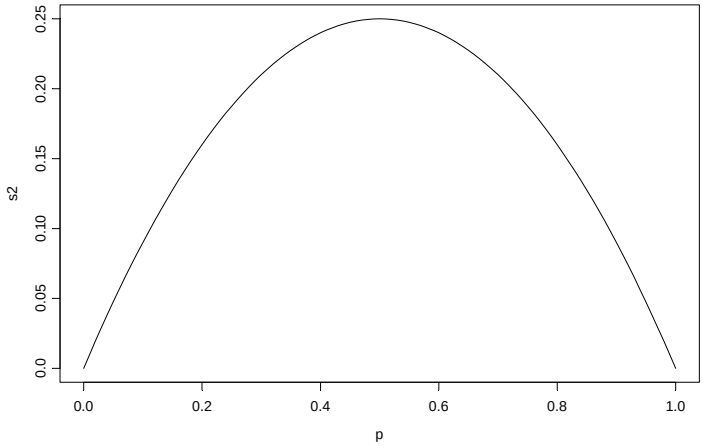
The traditional transformations of proportion data were arcsine and probit. The arcsine transformation took care of the error distribution, while the probit transformation was used to linearize the relationship between percentage mortality and log dose in a bioassay. There is nothing wrong with these transformations, and they are available within S-Plus, but a simpler approach is often preferable, and is likely to produce a model that is easier to interpret.

The major difficulty with modelling proportion data is that the responses are *strictly bounded*. There is no way that the percentage dying can be greater than 100% or less than 0%. But if we use simple techniques like regression or analysis of covariance, then the fitted model could quite easily predict negative values, especially if the variance was high and many of the data were close to 0.

The *logistic* curve is commonly used to describe data on proportions, because, unlike the straight line model, it asymptotes at 0 and 1 so that negative proportions, and responses of more than 100% can not be predicted. Throughout this discussion we shall use *p* to describe the proportion of individuals observed to respond in a given way. Because much of their jargon was derived from the theory of gambling, statisticians call these *successes* although, to an ecologist measuring death rates this may seem somewhat macabre. The individuals that respond in other ways (the statistician's *failures*) are therefore (1-*p*) and we shall call the proportion of failures *q*. The third variable is the size of the sample, *n*, from which *p* was estimated (it is the binomial denominator, and the statistician's *number of attempts*).
An important point about the binomial distribution is that the variance is not constant. In fact, the variance of the binomial distribution is:

$$s^2 = npq$$

so that the variance changes with the mean like this:



The variance is low when p is very high or very low, and the variance is greatest when p = q = 0.5. As p gets smaller, so the binomial distribution gets closer and closer to the Poisson distribution. You can see why this is so by considering the formula for the variance of the binomial: $s^2 = npq$. Remember that for the Poisson, the variance is equal to the mean; $s^2 = np$. Now, as p gets smaller, so q gets closer and closer to 1, so the variance of the binomial converges to the mean:

$$s^2 = npq \approx np \qquad (q \approx 1)$$

Odds

The logistic model for p as a function of x looks like this:

$$p = \frac{e^{(a+bx)}}{1+e^{(a+bx)}}$$

and there are no prizes for realising that the model is not linear. But if $x = -\infty$, then p=0; if $x = +\infty$ then p=1 so the model is strictly bounded. When x = 0 then p = exp(a)/(1 + exp(a)). The trick of linearising the logistic actually involves a very simple transformation. You may have come across the way in which bookmakers specify probabilities by quoting the *odds* against a particular horse winning a race (they might give odds of 2 to 1 on a reasonably good horse or 25 to 1 on an outsider). This is a rather different way of presenting information on probabilities than scientists are used to dealing with. Thus, where the scientist might state a proportion as 0.666 (2 out of 3), the bookmaker would give odds of 2 to 1 (2 successes to 1 failure). In symbols, this is the difference between the scientist stating the probability p, and the bookmaker stating the odds, p/q. Now if we take the *odds* p/q and substitute this into the formula for the logistic, we get:

$$\frac{p}{q} = \frac{e^{(a+bx)}}{1+e^{(a+bx)}}\left[1 - \frac{e^{(a+bx)}}{1+e^{(a+bx)}}\right]^{-1}$$

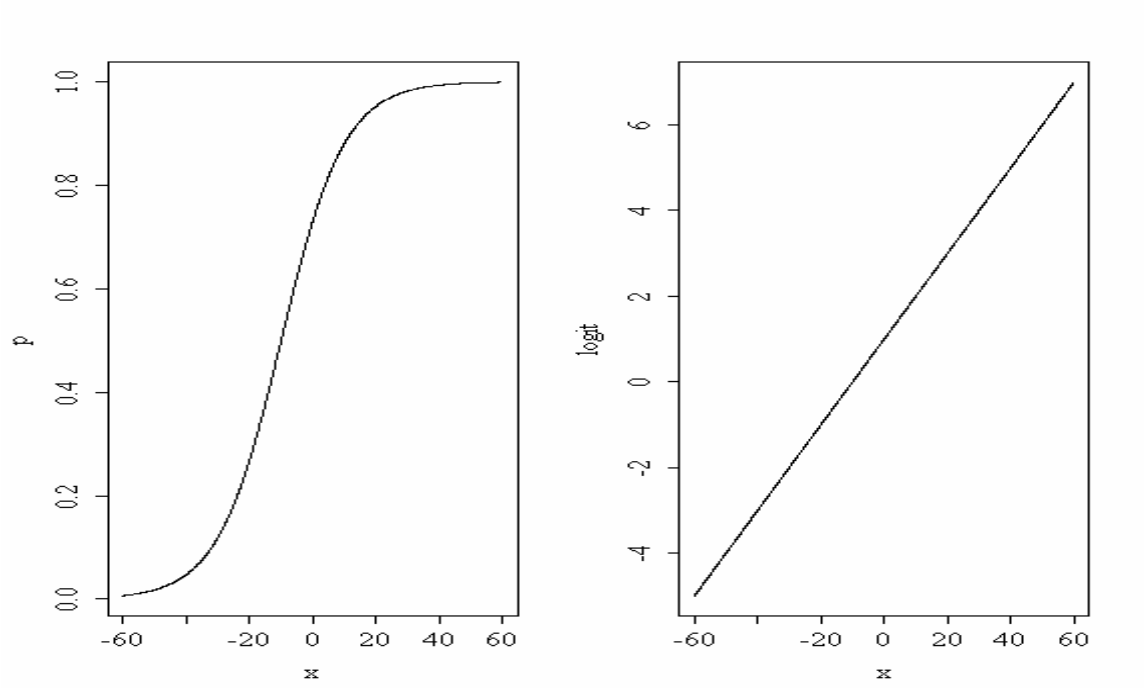
which looks awful. But a little algebra shows that:

$$\frac{p}{q} = \frac{e^{(a+bx)}}{1+e^{(a+bx)}}\left[\frac{1}{1+e^{(a+bx)}}\right]^{-1} = e^{(a+bx)}$$

Now, taking natural logs, and recalling that $\ln(e^x) = x$ will simplify matters even further, so that

$$\ln\left(\frac{p}{q}\right) = a + bx$$

This gives a *linear predictor*, $a + bx$, not for p but for the *logit* transformation of p, namely $\ln(p/q)$. In the jargon of SPlus, the logit is the *link function* relating the linear predictor to the value of p.



You might ask at this stage 'why not simply do a linear regression of $\ln(p/q)$ against the explanatory x-variable'? SPlus has three great advantages here:

- it allows for the non-constant binomial variance;
- it deals with the fact that logits for p's near 0 or 1 are infinite;
- it allows for differences between the sample sizes by weighted regression.

Deviance with binomial errors

Before we met **glm**'s we always used SSE as the measure of lack of fit (see p. xxx). For data on proportions, the maximum likelihood estimate of lack of fit is different from this: if y is the observed count and $\hat{y}$ is the fitted value predicted by the current

model, then if n is the binomial denominator (the sample size from which we obtained y successes), the deviance is :

$$2\sum y\log\left[\frac{y}{\hat{y}}\right]+(n-y)\log\left[\frac{(n-y)}{(n-\hat{y})}\right]$$

We should try this on a simple numerical example. In a trial with two treatments and two replicates per treatment we got 3 out of 6 and 4 out of 10 in one treatment, and 5 out of 9 and 8 out of 12 in the other treatment. Are the proportions significantly different across the two treatments? One thing you need to remember is how to calculate the average of two proportions. You don't work out the two proportions, add them up and divide by 2. What you do is count up the total number of successes and divide by the total number of attempts. In our case, the overall mean proportion is calculated like this. The total number of successes, *y*, is (3 + 4 + 5 + 8) = 20. The total number of attempts, *N*, is (6 + 10 + 9 + 12) = 37. So the overall average proportion is 20/37 = 0.5405. We need to work out the 4 expected counts. We multiply each of the sample sizes by the mean proportion: 6 x 0.5405 etc.

```
y<-c(3,4,5,8)
N<-c(6,10,9,12)
N*20/37
```

```
[1] 3.243243 5.405405 4.864865 6.486486
```

Now we can calculate the total deviance:

$$2\times\left\{3\log\left(\frac{3}{3.243}\right)+3\log\left(\frac{3}{6-3.243}\right)+4\log\left(\frac{4}{5.405}\right)+6\log\left(\frac{6}{10-5.405}\right)+5\log\left(\frac{5}{4.865}\right)+4\log\left(\frac{4}{9-4.865}\right)+8\log\left(\frac{8}{6.486}\right)+4\log\left(\frac{4}{12-6.486}\right)\right\}$$

You wouldn't want to have to do this too often, but it is worth it to demystify deviance.

```
p1<-2*(3*log(3/3.243)+3*log(3/(6-3.243))+4*log(4/5.405)+6*log(6/(10-5.405)))
p2<-2*(5*log(5/4.865)+4*log(4/(9-4.865))+8*log(8/6.486)+4*log(4/(12-6.486)))

p1+p2
```

```
[1] 1.629673
```

So the total deviance is 1.6297. Let's see what a **glm** with binomial errors gives:

```
rv<-cbind(y,N-y)
glm(rv~1,family=binomial)
```

```
Degrees of Freedom: 3 Total (i.e. Null);  3 Residual
Null Deviance:      1.63
Residual Deviance: 1.63         AIC: 14.28
```

Great relief all round. What about the residual deviance? We need to compute a new set of fitted values based on the two individual treatment means. Total successes in treatment A were 3 + 4 = 7 out of a total sample of 6 + 10 = 16. So the mean proportion was 7/16 = 0.4375. In treatment B the equivalent figures were 13/21 = 0.619. So the fitted values are

```
p<-c(7/16,7/16,13/21,13/21)
p*N
```

```
[1] 2.625000 4.375000 5.571429 7.428571
```

and the residual deviance is

$$2\times\left\{3\log\left(\frac{3}{2.625}\right)+3\log\left(\frac{3}{6-2.625}\right)+4\log\left(\frac{4}{4.375}\right)+6\log\left(\frac{6}{10-4.375}\right)+5\log\left(\frac{5}{5.571}\right)+4\log\left(\frac{4}{9-5.571}\right)+8\log\left(\frac{8}{7.429}\right)+4\log\left(\frac{4}{12-7.429}\right)\right\}$$

Again, we calculate this in 2 parts

```
p1<-2*(3*log(3/2.625)+3*log(3/(6-2.625))+4*log(4/4.375)+6*log(6/(10-4.375)))
p2<-2*(5*log(5/5.571)+4*log(4/(9-5.571))+8*log(8/7.429)+4*log(4/(12-7.429)))

p1+p2
```

```
[1] 0.4201974
```

```
treatment<-factor(c("A","A","B","B"))

glm(rv~treatment,family=binomial)
```

```
Degrees of Freedom: 3 Total (i.e. Null);  2 Residual
Null Deviance:      1.63
Residual Deviance: 0.4206       AIC: 15.07
```

So there really is no mystery in the values of deviance. It is just another way of measuring lack of fit. The difference between 0.4201974 and 0.4206 is just due to our use of only 3 decimal places in calculating the deviance components. If you were perverse, you could always work it out for yourself. Everything we have done with sums of squares (e.g. F tests and r^2), you can do with deviance.

S-Plus commands for Proportional Data

In addition to specifying that the y-variable contains the counted data on the number of successes, SPlus requires the user to provide a second vector containing the number of failures. The response is then a 2-column data frame, y, produced like this:

y <- **cbind**(successes,failures)

The sum of the successes and failures is the *binomial denominator* (the size of the sample from which each trial was drawn). This is used as a weight in the statistical modelling.

Overdispersion and hypothesis testing

All the different statistical procedures that we have met in earlier chapters can also be used with data on proportions. Factorial analysis of variance, multiple regression, and a variety of mixed models in which different regression lines are fit in each of several levels of one or more factors, can be carried out. The only difference is that we assess the significance of terms on the basis of chi-squared; the increase in scaled deviance that results from removal of the term from the current model.

The important point to bear in mind is that hypothesis testing with binomial errors is less clear-cut than with normal errors. While the chi squared approximation for changes in scaled deviance is reasonable for large samples (i.e. bigger than about 30), it is poorer with small samples. Most worrisome is the fact that the degree to which the approximation is satisfactory is itself unknown. This means that considerable care must be exercised in the interpretation of tests of hypotheses on parameters, especially when the parameters are marginally significant or when they explain a very small fraction of the total deviance. With binomial or Poisson errors we can not hope to provide exact P-values for our tests of hypotheses.

As with Poisson errors, we need to address the question of overdispersion (see Practical 11). When we have obtained the minimal adequate model, the residual scaled deviance should be roughly equal to the residual degrees of freedom. When the residual deviance is larger than the residual degrees of freedom there are two possibilities: either the model is mis-specified, or the probability of success, p, is not constant within a given treatment level. The effect of randomly varying *p* is to increase the binomial variance from *npq* to

$$s^2 = npq + n(n-1)\sigma^2$$

leading to a large residual deviance. This occurs even for models that would fit well if the random variation were correctly specified.

One simple solution is to assume that the variance is not npq but $npqs$, where *s* is an unknown *scale parameter* (s > 1). We obtain an estimate of the scale parameter by dividing the Pearson chi-square by the degrees of freedom, and use this estimate of *s* to compare the resulting scaled deviances for terms in the model using an F test (just as in conventional anova, Practical 5). While this procedure may work reasonably well for small amounts of overdispersion, it is no substitute for proper model specification. For example, it is not possible to test the goodness of fit of the model.

Model Criticism

Next, we need to assess whether the standardised residuals are normally distributed and whether there are any trends in the residuals, either with the fitted values or with the explanatory variables. It is necessary to deal with *standardised residuals* because with error distributions like the binomial, Poisson or gamma distributions, the variance changes with the mean. To obtain plots of the standardised residuals, we need to calculate:

$$r_s = \frac{y-\mu}{\sqrt{V}} = \frac{y-\mu}{\sqrt{\mu\left(1-\frac{\mu}{bd}\right)}}$$

where V is the formula for the *variance function* of the binomial, μ are the fitted values, and *bd* is the binomial denominator (successes plus failures).

Other kinds of standardised residuals (like Pearson residuals or deviance residuals) are available as options like this:

resid(model.name, type = "pearson")

or you could use other types including "deviance", "working" or "response". To plot the residuals against the fitted values, we might arc-sine transform the x axis (because the fitted data are proportions):

Summary

The most important points to emphasise in modelling with binomial errors are:

- create a 2-column object for the response, using **cbind** to join together the two vectors containing the counts of success and failure
- check for overdispersion (residual deviance > residual degrees of freedom), and correct for it by using F tests rather than chi square if necessary
- remember that you do not obtain exact P-values with binomial errors; the chi squared approximations are sound for large samples, but small samples may present a problem
- the fitted values are counts, like the response variable
- the linear predictor is in logits (the log of the odds = p/q)
- you can back transform from logits (z) to proportions (p) by $p = 1/(1+1/\exp(z))$

Applications

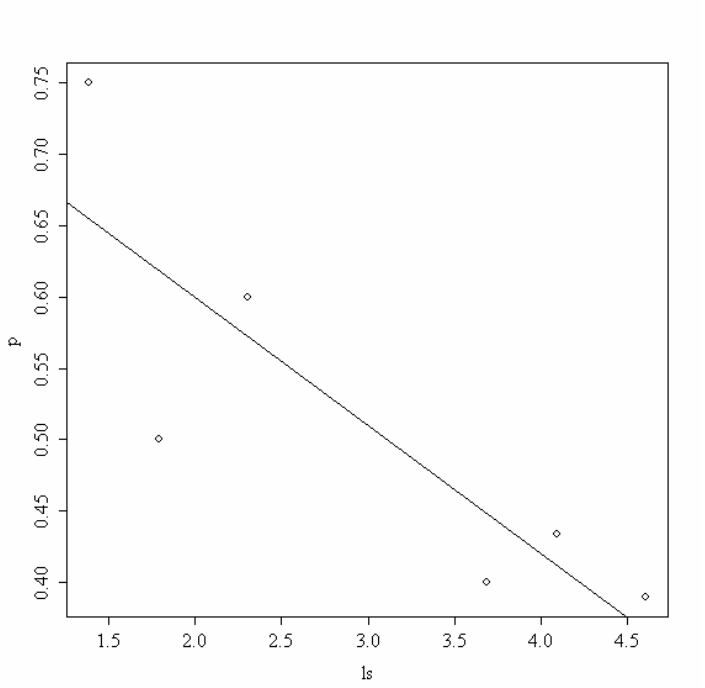
1) Regression analysis with proportion data

In a study of density dependent seed mortality in trees, seeds were placed in 6 groups of different densities (size from 4 to 100). The scientist returned after 48 hours and counted the number of seeds taken away by animals. We are interested to see if the proportion of seeds taken is influenced by the size of the cache of seeds. These are the data:

```
taken<-c(3,3,6,16,26,39)

size<-c(4,6,10,40,60,100)
```

Now calculate the proportion removed (*p*) and the logarithm of pile size (*ls*):

```
p<-taken/size
ls<-log(size)
plot(ls,p)
abline(lm(p~ls))
```



It certainly looks as if the proportion taken declines as density increases. Let's do the regression to test the significance of this:

```
summary(lm(p~ls))

Call:
lm(formula = p ~ ls)

Residuals:
       1        2        3        4        5        6
 0.09481 -0.11877  0.02710 -0.04839  0.02135  0.02390

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.77969    0.08939   8.723 0.000952 ***
ls          -0.08981    0.02780  -3.230 0.031973 *

Residual standard error: 0.08246 on 4 degrees of freedom
Multiple R-Squared: 0.7229,     Adjusted R-squared:
0.6536
F-statistic: 10.43 on 1 and 4 degrees of freedom,
p-value: 0.03197
```

So the density dependent effect is significant (p = 0.032). But these were proportion data. Perhaps we should have transformed them before doing the regression ? We can repeat the regression using the logit transformation.

```
logit<-log(p/(1-p))

summary(lm(logit~ls))

Call:
lm(formula = logit ~ ls)

Residuals:
       1        2        3        4        5        6
 0.43065 -0.51409  0.08524 -0.19959  0.09149  0.10630

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.1941     0.3895   3.065   0.0375 *
ls           -0.3795     0.1212  -3.132   0.0351 *

Residual standard error: 0.3593 on 4 degrees of freedom
Multiple R-Squared: 0.7104,     Adjusted R-squared: 0.638
F-statistic: 9.811 on 1 and 4 degrees of freedom,
p-value: 0.03511
```

The regression is still significant, suggesting density dependence in removal rate.

Now we shall carry out the analysis properly, taking account of the fact that the very high death rates were actually estimated from very small samples of seeds (4, 6, and 10 individuals respectively). A **glm** with binomial errors takes care of this by giving low weight to estimates with small binomial denominators. First we need to construct a response vector (using **cbind**) that contains the number of successes and the number of failures. In our example, this means the number of seeds taken away and the number of seeds left behind. We do it like this:

```
left<-size - taken

y<-cbind(taken,left)
```

Now the modelling. We use **glm** instead of **lm**, but otherwise everything else is the same as usual.

```
model<-glm(y~ls,binomial)

summary(model)

Call:
glm(formula = y ~ ls, family = binomial)

Deviance Residuals:
[1]   0.5673  -0.4324   0.3146  -0.6261   0.2135   0.1317

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.8815     0.7488   1.177    0.239
ls           -0.2944     0.1817  -1.620    0.105
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3.7341  on 5  degrees of freedom
Residual deviance: 1.0627  on 4  degrees of freedom
AIC: 25.543
```

When we do the analysis properly, using weighted regression in glim, we see that there is no significant effect of density on the proportion of seeds removed ($p = 0.105$). This analysis was carried out by a significance test on the parameter values. It is usually better to assess significance by a deletion test. To do this, we fit a simpler model with an intercept but no slope:

```
model2<-glm(y~1,binomial)

anova(model,model2)

Analysis of Deviance Table

Response: y


   Resid. Df Resid. Dev Df Deviance
ls         4     1.0627
1          5     3.7341 -1  -2.6713
```

The thing to remember is that the change in deviance is a chi squared value. In this case the chi-square on 1 d.f. is only 2.671 and this is less than the value in tables (3.841), so we conclude that ***there is no evidence for density dependence*** in the death rate of these seeds. It is usually more convenient to have the chi squared test printed out as part of the analysis of deviance table. This is very straightforward. We just include the option test="Chi" in the **anova** directive

```
anova(model,model2,test="Chi")

Analysis of Deviance Table
Response: y


   Resid. Df Resid. Dev Df Deviance P(>|Chi|)
ls         4     1.0627
1          5     3.7341 -1  -2.6713    0.1022
```

The moral is clear. Tests of density dependence need to be carried out using **weighted regression**, so that the death rates estimated from small samples do not have undue influence on the value of the regression slope. In this example, using the correct linearization technique (logit transformation) did not solve the problem; logit regression agreed (wrongly) with simple regression that there was significant inverse density dependence (i.e. the highest death rates were estimated at the lowest densities). But these high rates were estimated from tiny samples (3 out of 4 and 3 out of 6 individuals). The correct analysis uses weighted regression and binomial errors, in which case these points are given less influence. The result is that the data no longer provided convincing evidence of density dependence. In fact, the data are consistent with the hypothesis that the death rate was the same at all densities.

There are lots of examples in the published ecological literature of alleged cases of density dependence that do not stand up to a proper statistical analysis. In many key factor analyses, for example, the k factors (log N(t+1) - log N(t)) at low densities are often based on very small sample sizes. These points are highly influential and can produce spurious positive or negative density dependence unless weighted regression is employed.

2) Analysis of deviance with proportion data

This example concerns the germination of seeds of 2 genotypes of the parasitic plant *Orobanche* and 2 plant extracts (bean and cucumber) that were used to stimulate germination.

```
germination<-read.table("c:\\temp\\germination.txt",header=T)
attach(germination)
names(germination)
```

```
[1] "count"     "sample"    "Orobanche" "extract"
```

Count is the number of seeds that germinated out of a batch of size = sample. So the number that didn't germinate is sample – count, and we can construct the response vector like this

```
y<-cbind(count , sample-count)
```

Each of the categorical explanatory variables has 2 levels

```
levels(Orobanche)
```

```
[1] "a73" "a75"
```

```
levels(extract)
```

```
[1] "bean"     "cucumber"
```

We want to test the hypothesis that there is no interaction between *Orobanche* genotype ("a73" or "a75") and plant extract ("bean" or "cucumber") on the germination rate of the seeds. This requires a factorial analysis using the asterisk * operator like this

```
model<-glm(y ~ Orobanche * extract, binomial)
```

```
summary(model)
```

```
Call:
glm(formula = y ~ Orobanche * extract, family = binomial)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.01617  -1.24398   0.05995   0.84695   2.12123
```

```
(Intercept)                   -0.4122     0.1842  -2.238   0.0252 *
Orobanchea75                  -0.1459     0.2232  -0.654   0.5132
extractcucumber                0.5401     0.2498   2.162   0.0306 *
Orobanchea75:extractcucumber   0.7781     0.3064   2.539   0.0111 *

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 98.719  on 20  degrees of freedom
Residual deviance: 33.278  on 17  degrees of freedom
AIC: 117.87
```

At first glance, it looks as if there is a highly significant interaction (p = 0.0111). But we need to check that the model is sound. The **first thing is to check for overdispersion**. The residual deviance is 33.278 on 17 d.f. so the model is quite badly overdispersed:

```
33.279 / 17

[1] 1.957588
```

the overdispersion factor is almost 2. The simplest way to take this into account is to use what is called an "empirical scale parameter" to reflect the fact that the errors are not binomial as we assumed, but were larger than this (overdispersed) by a factor of 1.9576. We re-fit the model using quasibinomial to account for the overdispersion.

```
model<-glm(y ~ Orobanche * extract, quasibinomial)
```

Then we use update to remove the interaction term in the normal way.

```
model2<-update(model, ~ . - Orobanche:extract)
```

The only difference is that we use an F-test instead of a Chi square test to compare the original and simplified models:

```
anova(model,model2,test="F")

Analysis of Deviance Table

Model 1: y ~ Orobanche * extract
Model 2: y ~ Orobanche + extract

  Resid. Df Resid. Dev Df Deviance      F  Pr(>F)
1        17     33.278
2        18     39.686 -1   -6.408 3.4419 0.08099 .
```

Now you see that the interaction is not significant ($p = 0.081$). There is no indication that different genotypes of *Orobanche* respond differently to the two plant extracts.

The next step is to see if any further model simplification is possible.

```
anova(model2,test="F")

Analysis of Deviance Table
```

```
Model: quasibinomial, link: logit

Response: y

Terms added sequentially (first to last)


          Df Deviance Resid. Df Resid. Dev       F    Pr(>F)
NULL                        20     98.719
Orobanche  1    2.544        19     96.175  1.1954    0.2887
extract    1   56.489        18     39.686 26.5419 6.691e-05 ***
```

There is a highly significant difference between the two plant extracts on germination rate, but it is not obvious that we need to keep *Orobanche* genotype in the model. We try removing it.

```
model3<-update(model2, ~ . - Orobanche)
anova(model2,model3,test="F")
```

```
Analysis of Deviance Table

Model 1: y ~ Orobanche + extract
Model 2: y ~ extract

  Resid. Df Resid. Dev Df Deviance      F Pr(>F)
1        18     39.686
2        19     42.751 -1   -3.065 1.4401 0.2457
```

There is no justification for retaining *Orobanche* in the model. So the minimal adequate model contains just two parameters:

```
coef(model3)
```

```
(Intercept)     extract
 -0.5121761   1.0574031
```

What, exactly, do these two numbers mean ? Remember that the coefficients are from the linear predictor. They are on the transformed scale, so because we are using binomial errors, they are in logits (ln(p/(1-p)). To turn them into the germination rates for the two plant extracts requires us to do a little calculation. To go from a logit *x* to a proportion *p*, you need to do the following sum

$$p = \frac{1}{1 + e^{-x}}$$

So our first x value is –0.5122 and we calculate

```
1/(1+(exp(0.5122)))
```

```
[1] 0.3746779
```

This says that the mean germination rate of the seeds with the first plant extract was 37%. What about the parameter for extract (1.057). Remember that with categorical

explanatory variables *the parameter values are differences between means*. So to get the second germination rate we *add 1.057 to the intercept* before back-transforming:

```
1/(1+1/(exp(-0.5122+1.0574)))
```

```
[1] 0.6330212
```

This says that the germination rate was nearly twice as great (63%) with the second plant extract (cucumber). Obviously we want to generalise this process, and also to speed up the calculations of the estimated mean proportions. We can use **predict** to help here.

```
tapply(predict(model3),extract,mean)
```

```
      bean   cucumber
-0.5121761  0.5452271
```

This extracts the average logits for the two levels of extract, showing that seeds germinated better with cucumber extract than with bean extract. To get the mean proportions we just apply the back transformation to this tapply (use Up Arrow to edit)

```
tapply(1/(1+1/exp(predict(model3))),extract,mean)
```

```
     bean  cucumber
0.3746835 0.6330275
```

These are the two proportions we calculated earlier. There is an even easier way to get the proportions, because there is an option for **predict** called type="response" which makes predictions on the back-transformed scale automatically:

```
tapply(predict(model3,type="response"),extract,mean)
```

```
     bean  cucumber
0.3746835 0.6330275
```

It is interesting to compare these figures with the average of the raw proportions and the overall average. First we need to calculate the proportion germinating, *p*, in each sample

```
p<-count/sample
```

then we can average it by extract

```
tapply(p,extract,mean)
```

```
     bean  cucumber
0.3487189 0.6031824
```

You see that **this gives a different answer**. Not too different in this case, but different none the less. The correct way to average proportion data is to add up the total counts for the different levels of abstract, and only then to turn them into proportions:

```
tapply(count,extract,sum)
```

```
 bean  cucumber
  148       276
```

This means that 148 seeds germinated with bean extract and 276 with cucumber

```
tapply(sample,extract,sum)
```

```
  bean cucumber
   395      436
```

This means that there were 395 seeds treated with bean extract and 436 seeds treated with cucumber. So the answers we want are 148/395 and 276/436. We automate the calculations like this:

```
ct<-as.vector(tapply(count,extract,sum))

sa<-as.vector(tapply(sample,extract,sum))

ct/sa
```

```
[1] 0.3746835 0.6330275
```

These are the correct mean proportions that were produced by **glm**. The moral here is that you calculate the average of proportions by using total counts and total samples and **not by averaging the raw proportions**.

To summarise this analysis:

- make a 2-column response vector containing the successes and failures
- use **glm** with **family=binomial** (you don't need to include the 'family=' bit)
- fit the maximal model (in this case it had 4 parameters)
- test for **overdispersion**
- if, as here, you find overdispersion then use F tests not Chi square tests of deletion
- begin model simplification by removing the interaction term
- this was non significant once we adjusted for overdispersion
- try removing main effects (we didn't need *Orobanche* genotype in the model)
- use **plot** to obtain your model-checking diagnostics
- back transform using **predict** with the option type="response" to obtain means

3) Analysis of covariance with proportion data

Here we carry out a logistic analysis of covariance to compare 3 different insecticides in a toxicology bioassay. Three products (A, B and C) were applied to batches of insects in circular arenas. The initial numbers of insects (*n*) varied between 15 and 24. The insecticides were each applied at 17 different doses, varying on a log scale from 0.1 to 1.7 units of active ingredient m^{-2}. The number of insects found dead after 24 hours, *c*, was counted.

```
logistic<-read.table("c:\\temp\\logistic.txt",header=T)
attach(logistic)
names(logistic)
```

```
[1] "logdose" "n"       "dead"    "product"
```

First we calculate the proportion dying:

```
p<-dead/n
```

Now we plot the scattergraph

```
plot(logdose,p,type="n")
```

We used **type=”n”** to make sure that all the points from all 3 products fit onto the plotting surface. Now we add the points for each product, one at a time:

```
points(logdose[product=="A"],p[product=="A"],pch=16,col=2)
points(logdose[product=="B"],p[product=="B"],pch=16,col=4)
points(logdose[product=="C"],p[product=="C"],pch=16,col=1)
```

It looks as if the slope of the response curve is shallower for the blue product (“B”), but there is no obvious difference between products A and C. We shall test this using analysis of covariance in which we begin by fitting 3 separate logistic regression lines (one for each product) then ask whether all the parameters need to be retained in the minimum adequate model.

Begin be creating the response vector using **cbind** like this:

```
y<-cbind(dead,n-dead)
```

Now we fit the maximal model using the * notation to get 3 slopes and 3 intercepts:

```
model<-glm(y~product*logdose,binomial)
```

```
summary(model)
```

```
Call:
glm(formula = y ~ product * logdose, family = binomial)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.39802  -0.60733   0.09697   0.65918   1.69300
```

```
Coefficients:
                Estimate Std. Error z value  Pr(>|z|)
(Intercept)      -2.0410     0.3028  -6.741 1.57e-011 ***
productB          1.2920     0.3841   3.364  0.000768 ***
productC         -0.3080     0.4301  -0.716  0.473977
logdose           2.8768     0.3423   8.405  < 2e-016 ***
productB.logdose -1.6457     0.4213  -3.906 9.38e-005 ***
productC.logdose  0.4319     0.4887   0.884  0.376845

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 311.241  on 50  degrees of freedom
Residual deviance:  42.305  on 45  degrees of freedom
AIC: 201.31
```

First we check for overdispersion. The residual deviance was 42.305 on 45 degrees of freedom, so that is perfect. There is no evidence of overdispersion at all, so we can use Chi square tests in our model simplification. There looks to be a significant difference in slope for product B (as we suspected from the scatter plot); its slope is shallower by –1.6457 than the slope for product A (labelled "*logdose*" in the table above). But what about product C? Is it different from product A in its slope or its intercept? The z tests certainly suggest that it is not significantly different. Let's remove it by creating a new factor called *newfac* that combines A and C:

```
newfac<-factor(1+(product=="B"))
```

Now we create a new model in which we replace the 3-level factor called "*product*" by the 2-level factor called "*newfac*":

```
model2<-update(model , ~ . - product*logdose + newfac*logdose)
```

and then we ask whether the simpler model is significantly less good at describing the data than the more complex model?

```
anova(model,model2,test="Chi")

Analysis of Deviance Table

Model 1: y ~ product * logdose
Model 2: y ~ logdose + newfac + logdose:newfac

  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1        45     42.305
2        47     43.111 -2   -0.805     0.668
```

Evidently, this model simplification was completely justified (p = 0.668). So is this the minimal adequate model ? We need to look at the coefficients:

```
summary(model2)

Coefficients:
```

```
               Estimate Std. Error z value  Pr(>|z|)
(Intercept)     -2.1997     0.2148 -10.240  < 2e-016 ***
logdose          3.0988     0.2438  12.711  < 2e-016 ***
newfac           1.4507     0.3193   4.543 5.54e-006 ***
logdose.newfac  -1.8676     0.3461  -5.397 6.79e-008 ***
```

Yes it is. All of the 4 coefficients are significant (3-star in this case); we need 2 different slopes and 2 different intercepts. So now we can draw the fitted lines through our scatterplot. You may need to revise the steps involved in this. We need to make a vector of values for the x axis (we might call it *xv*) and a vector of values for the 2-level factor to decide which graph to draw (we might call it *nf*).

```
max(logdose)

[1] 1.7

min(logdose)

[1] 0.1
```

So we need to generate a sequence of values of *xv* between 0.1 and 1.7

```
xv<-seq(0.1,1.7,0.01)

length(xv)

[1] 161
```

This tells us that the vector of factor levels in *nf* needs to be made up of 161 repeats of factor level = 1 then 161 repeats of factor level = 2, like this

```
nf<-factor(c(rep(1,161),rep(2,161)))
```

Now we double the length of xv to account for both levels of nf:

```
xv<-c(xv,xv)
```

Inside the **predict** directive we combine *xv* and *nf* into a data frame where we give them *exactly the same names* as the variables had in model2 . Remember that we need to back-transform from predict (which is in logits) onto the scale of proportions, using type=”response”.

```
yv<-predict(model2,type="response",data.frame(logdose=xv,newfac=nf))
```

It simplifies matters if we use split to get separate lists of coordinated for the two regression curves:

```
yvc<-split(yv,nf)
xvc<-split(xv,nf)
```

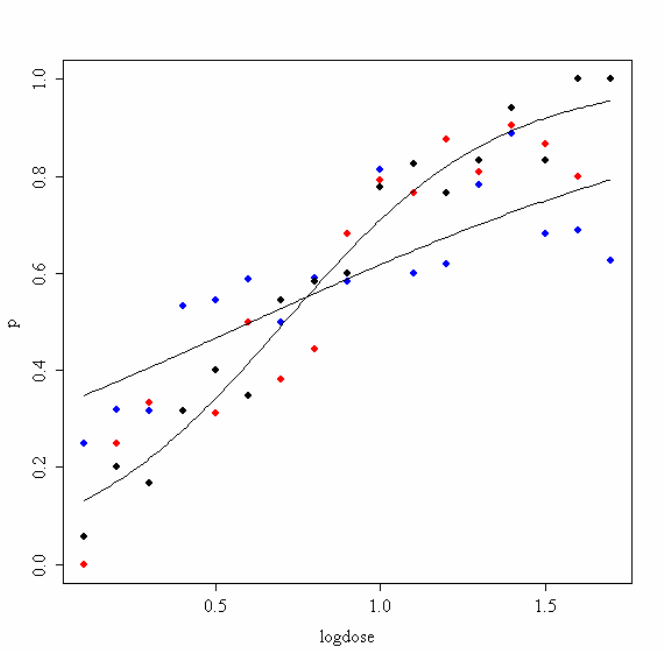
We shall draw the graph in two steps, first for factor level 1 (i.e. for products A and C combined):

```
lines(xvc[[1]],yvc[[1]])
```

and then for the second factor level (product B)

```
lines(xvc[[2]],yvc[[2]])
```

Note the need for double subscripts [[2]] because split produces 2 lists not 2 vectors.



All 3 products have very similar LD50's (roughly logdose = 0.8), but higher doses of A and C are much better than B at killing insects in large numbers.

Binary Response Variables

Many statistical problems involve binary response variables. For example, we often classify individuals as

- dead or alive
- occupied or empty
- healthy or diseased
- wilted or turgid
- male or female
- literate or illiterate
- mature or immature
- solvent or insolvent
- employed or unemployed

and it is interesting to understand the factors that are associated with an individual being in one class or the other (Cox & Snell 1989). In a study of company insolvency, for instance, the data would consist of a list of measurements made on the insolvent companies (their age, size, turnover, location, management experience, workforce training, and so on) and a similar list for the solvent companies. The question then becomes which, if any, of the explanatory variables increase the probability of an individual company being insolvent.

The response variable contains only 0's or 1's; for example, 0 to represent dead individuals and 1 to represent live ones. There is a single column of numbers, in contrast to proportion data (see above). The way SPlus treats this kind of data is to assume that the 0's and 1's come from *a binomial trial with sample size 1*. If the probability that an animal is dead is *p*, then the probability of obtaining *y* (where *y* is either dead or alive, 0 or 1) is given by an abbreviated form of the binomial distribution with $n = 1$, known as the Bernoulli distribution:

$$P(y) = p^{y}(1-p)^{(1-y)}$$

The random variable *y* has a mean of *p* and a variance of *p(1-p)*, and the object is to determine how the explanatory variables influence the value of *p*.

The trick to using binary response variables effectively is to know when it is worth using them and when it is better to lump the successes and failures together and analyse the *total counts* of dead individuals, occupied patches, insolvent firms or whatever. The question you need to ask yourself is this:

do I have unique values of one or more explanatory variables for each and every individual case?

If the answer is ‘yes’, then analysis with a binary response variable is likely to be fruitful. If the answer is ‘no’, then there is nothing to be gained, and you should reduce your data by aggregating the counts to the resolution at which each count *does* have a unique set of explanatory variables. For example, suppose that all your explanatory variables were categorical (say sex (male or female), employment

(employed or unemployed) and region (urban or rural)). In this case there is nothing to be gained from analysis using a binary response variable because none of the individuals in the study have *unique* values of any of the explanatory variables. It might be worthwhile if you had each individual's body weight, for example, then you could ask the question "when I control for sex and region, are heavy people more likely to be unemployed than light people? ". In the absence of *unique* values for any explanatory variables, there are two useful options:

- analyse the data as a 2 x 2 x 2 contingency table using Poisson errors, with the count of the total number of individuals in each of the 8 contingencies is the response variable (see Practical 11) in a data frame with just 8 rows.

- decide which of your explanatory variables is the key (perhaps you are interested in gender differences), then express the data as proportions (the number of males and the number of females) and re-code the binary response as count of a 2-level factor. The analysis is now of proportion data (the proportion of all individuals that are female, for instance) using binomial errors (see Practical 11).

If you *do* have unique measurements of one or more explanatory variables for each individual, these are likely to be continuous variables (that's what makes them unique to the individual in question). They will be things like body weight, income, medical history, distance to the nuclear reprocessing plant, and so on. This being the case, successful analyses of binary response data tend to be multiple regression analyses or complex analyses of covariance, and you should consult the relevant Practicals for details on model simplification and model criticism.

In order to carry out linear modelling on a binary response variable we take the following steps:

- create a vector of 0's and 1's as the response variable
- use **glm** with **family=binomial**
- you can change the link function from default logit to complementary log-log
- fit the model in the usual way
- test significance by deletion of terms from the maximal model, and compare the change in deviance with chi-square
- note that there is no such thing as overdispersion with a binary response variable, and hence no need to change to using F tests.

Choice of link function is generally made by trying both links and selecting the link that gives the lowest deviance. The logit link that we used earlier is symmetric in p and q, but the complementary log-log link is asymmetric.

Death rate and reproductive effort (an ecological trade-off)

We shall use a binary response variable to analyse some questions about the costs of reproduction in a perennial plant. Suppose that we have one set of measurements on plants that survived through the winter to the next growing season, and another set of measurements on otherwise similar plants that died between one growing season and the next. We are particularly interested to know whether, for a plant of a given size,

the probability of death is influenced by the number of seeds it produced during the previous year.

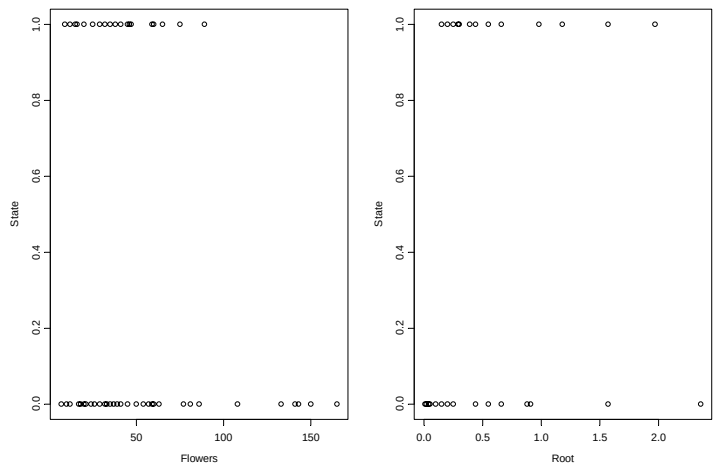
Because the binomial denominator, *n*, is always equal to 1, we don't need to specify it. Instead of binding together 2 vectors containing the numbers of successes and failures, we provide a single vector of 0's and 1's as the response variable. The aim of the modelling exercise is to determine which of the explanatory variables influences the probability of an individual being in one state or the other.

```
flowering<-read.table("c:\\temp\\flowering.txt",header=T)
attach(flowering)
names(flowering)
```

```
[1] "State"   "Flowers" "Root"
```

The first thing to get used to is how odd the plots look. Because all of the values of the response variable are just 0's or 1's we get to rows of points: one across the top of the plot at y = 1 and another across the bottom at y = 0. We have two explanatory variables in this example, so let's plot the state of the plant next spring (dead or alive) as a function of its seed production this summer (Flowers) and the size of its rootstock (Root). The expectation is that plants that produce more flowers are more likely to die during the following winter, and plants with bigger roots are less likely to die than plants with smaller rootas. We use **par(mfrow=c(1,2))** to get 2 plots, side-by-side:

```
par(mfrow=c(1,2))
plot(Flowers,State)
plot(Root,State)
```



It is very hard to see any pattern in plots of binary data without the fitted models. Rather few of the plants with the biggest roots died (bottom right of the right hand plot), and rather few of the plants with big seed production survived (top right of the left-hand plot). The modelling is very straightforward. It is a multiple regression with 2 continuous explanatory variables (Flowers and Root), where the response variable,

State, is binary with values either 0 or 1. We begin by fitting the full model, including an interaction term, using the * operator:

```
model<-glm(State~Flowers*Root,binomial)

summary(model)
```

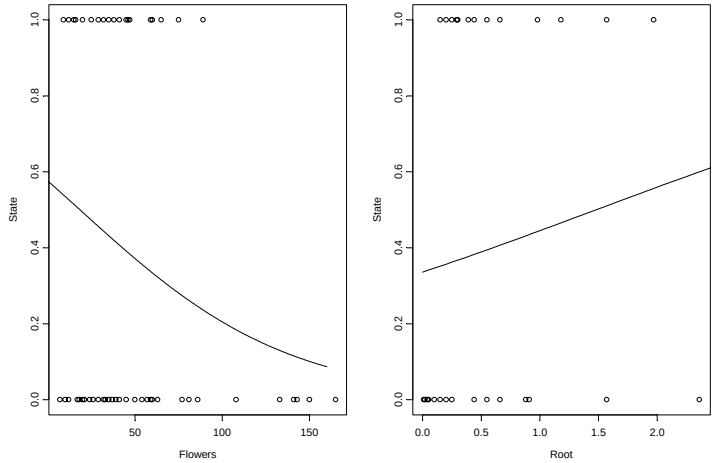
```
Call:
glm(formula = State ~ Flowers * Root, family = binomial)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.74539  -0.44300  -0.03415   0.46455   2.69430

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -2.96084    1.52124  -1.946  0.05161 .
Flowers      -0.07886    0.04150  -1.901  0.05736 .
Root         25.15152    7.80222   3.224  0.00127 **
Flowers.Root -0.20911    0.08798  -2.377  0.01746 *

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 78.903  on 58  degrees of freedom
Residual deviance: 37.128  on 55  degrees of freedom
AIC: 45.128
```

The first question is whether the interaction term is required in the model. It certainly looks as if it is required (p = 0.01746), but we need to check this by deletion. We use **update** to compute a simpler model, leaving out the interaction *Flower:Root*

```
model2<-update(model , ~ . - Flowers:Root)

anova(model,model2,test="Chi")

Analysis of Deviance Table

Model 1: State ~ Flowers * Root
Model 2: State ~ Flowers + Root

  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1        55     37.128
2        56     54.068 -1  -16.940 3.858e-05
```

This is highly significant, so we conclude that the *way* that flowering affects survival depends upon the size of the rootstock. Inspection of the model coefficients indicates that a given level of flowering has a bigger impact on reducing survival on plants with small roots than on larger plants.

Both explanatory variables and their interaction are important, so we can not simplify the original model. Here are separate 2-D graphs for the effects of *Flowers* and *Root* on *State*:



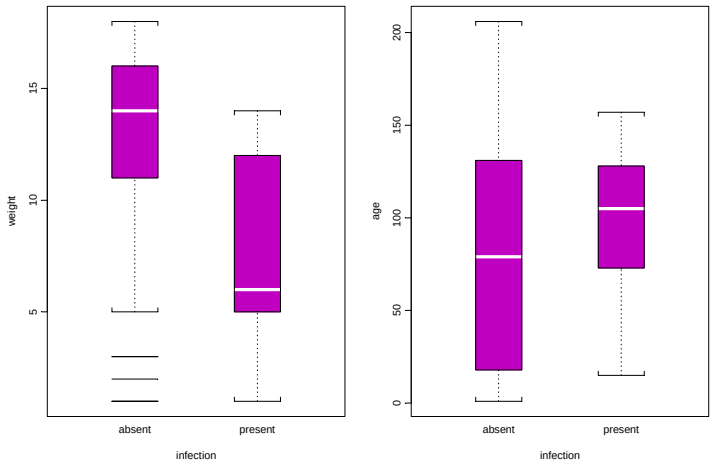
Binary response variable with both continuous and categorical explanatory variables: Logistic Ancova

```
parasite<-read.table("c:\\temp\\parasite.txt",header=T)
attach(parasite)
names(parasite)
```

```
[1] "infection" "age"       "weight"    "sex"
```

In this example the binary response variable is parasite infection (infected or not) and the explanatory variables are weight and age (continuous) and sex (categorical). We begin with data inspection.

```
par(mfrow=c(1,2))
plot(infection,weight)
plot(infection,age)
```



Infected individuals are substantially lighter than uninfected individuals, and occur in a much narrower range of ages. To see the relationship between infection and gender (both categorical variables) we can use table:

```
table(infection,sex)
```

```
        female male
 absent     17   47
present     11    6
```

which indicates that the infection is much more prevalent in females (11/28) than in males (6/53).

We begin, as usual, by fitting a maximal model with different slopes for each level of the categorical variable:

```
model<-glm(infection~age*weight*sex,family=binomial)
```

```
summary(model)
```

```
Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)        -0.109124   1.375388  -0.079    0.937
age                 0.024128   0.020874   1.156    0.248
weight             -0.074156   0.147678  -0.502    0.616
sexmale            -5.969133   4.275952  -1.396    0.163
age:weight         -0.001977   0.002006  -0.985    0.325
age:sexmale         0.038086   0.041310   0.922    0.357
weight:sexmale      0.213835   0.342825   0.624    0.533
age:weight:sexmale -0.001651   0.003417  -0.483    0.629

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 83.234  on 80  degrees of freedom
Residual deviance: 55.706  on 73  degrees of freedom
AIC: 71.706
```

Remember that with a binary response variable, the null and residual deviance don't mean anything, and there is no such thing as overdispersion.

It certainly does not look as if any of the high order interactions are significant. Instead of using update and anova for model simplification, we can use **step** to compute the AIC for each term in turn:

step(model)

```
Start:  AIC= 71.71
 infection ~ age + weight + sex + age:weight + age:sex +
weight:sex +  age:weight:sex
```

First, it tests whether the 3-way interaction is required

```
                 Df Deviance    AIC
- age:weight:sex  1   55.943 69.943
<none>                55.706 71.706

Step:  AIC= 69.94
 infection ~ age + weight + sex + age:weight + age:sex +
weight:sex
```

A reduction in AIC of just 71.7 – 69.9 = 1.8 and hence not significant. Next, it looks at the 3 2-way interactions and decides which to delete first:

```
Df Deviance    AIC
- weight:sex  1   56.122 68.122
- age:sex     1   57.828 69.828
<none>            55.943 69.943
- age:weight  1   58.674 70.674

Step:  AIC= 68.12
 infection ~ age + weight + sex + age:weight + age:sex

             Df Deviance    AIC
<none>            56.122 68.122
- age:sex     1   58.142 68.142
- age:weight  1   58.899 68.899

Call:  glm(formula = infection ~ age + weight + sex +
age:weight + age:sex,      family = binomial)

Coefficients:

(Intercept)          age       weight      sexmale   age:weight  age:sexmale
  -0.391572     0.025764    -0.036493    -3.743698    -0.002221     0.020464

Degrees of Freedom: 80 Total (i.e. Null);  75 Residual
Null Deviance:      83.23
Residual Deviance: 56.12        AIC: 68.12
```

The **step** procedure suggests that we retain two 2-way interactions: age:weight and age:sex. Let's see if we would have come to the same conclusion using **update** and **anova**.

```
model2<-update(model,~.-age:weight:sex)
anova(model,model2,test="Chi")[,-1]
```

-age:weight:sex p = 0.626 so no evidence of 3-way interaction. Now for the 2-ways:

```
model3<-update(model2,~.-age:weight)
anova(model2,model3,test="Chi")[,-1]
```

-age:weight p = 0.098 so no really persuasive evidence of an age:weight term

```
model4<-update(model2,~.-age:sex)
anova(model2,model4,test="Chi")[,-1]
```

Note that we are testing all the 2-way interactions by deletion from the model that contains all 2-way interactions (model2):

-age:sex p = 0.170, so nothing there, then.

```
model5<-update(model2,~.-weight:sex)
anova(model2,model5,test="Chi")[,-1]
```

-weight:sex p = 0.672 or thereabouts. As one often finds, **step** has been relatively generous in leaving terms in the model. It is a good idea to try adding back marginally significant terms, to see if they have greater explanatory power when fitted on their own.

```
model6<-update(model2,~.-weight:sex-weight:age-sex:age)
model7<-update(model6,~.+weight:age)
anova(model7,model6,test="Chi")[,-1]
```

-weight:age p = 0.190. So no evidence for this interaction. Now for the main effects:

```
model8<-update(model6,~. -weight)
anova(model6,model8,test="Chi")[,-1]
```

-weight p = 0.0003221 so weight is highly significant, as we expected from the initial boxplot:

```
model9<-update(model6,~. -sex)
anova(model6,model9,test="Chi")[,-1]
```

-sex p = 0.020 so sex is quite significant:

```
model10<-update(model6,~. -age)
anova(model6,model10,test="Chi")[,-1]
```

-age p = 0.048 so age is marginally significant. Note that all the main effects were tested by deletion from the model that contained all the main effects (model6).

It is worth establishing whether there is any evidence of non-linearity in the response of infection to weight or age. We might begin by fitting quadratic terms for the 2 continuous explanatory variables:

```
model11<-
glm(infection~sex+age+weight+I(age^2)+I(weight^2),family=binomial)
```

Then dropping each of the quadratic terms in turn:

```
model12<-glm(infection~sex+age+weight+I(age^2),family=binomial)
anova(model11,model12,test="Chi")[,-1]
```

-I(weight^2) p = 0.034 so the weight^2 term is needed.

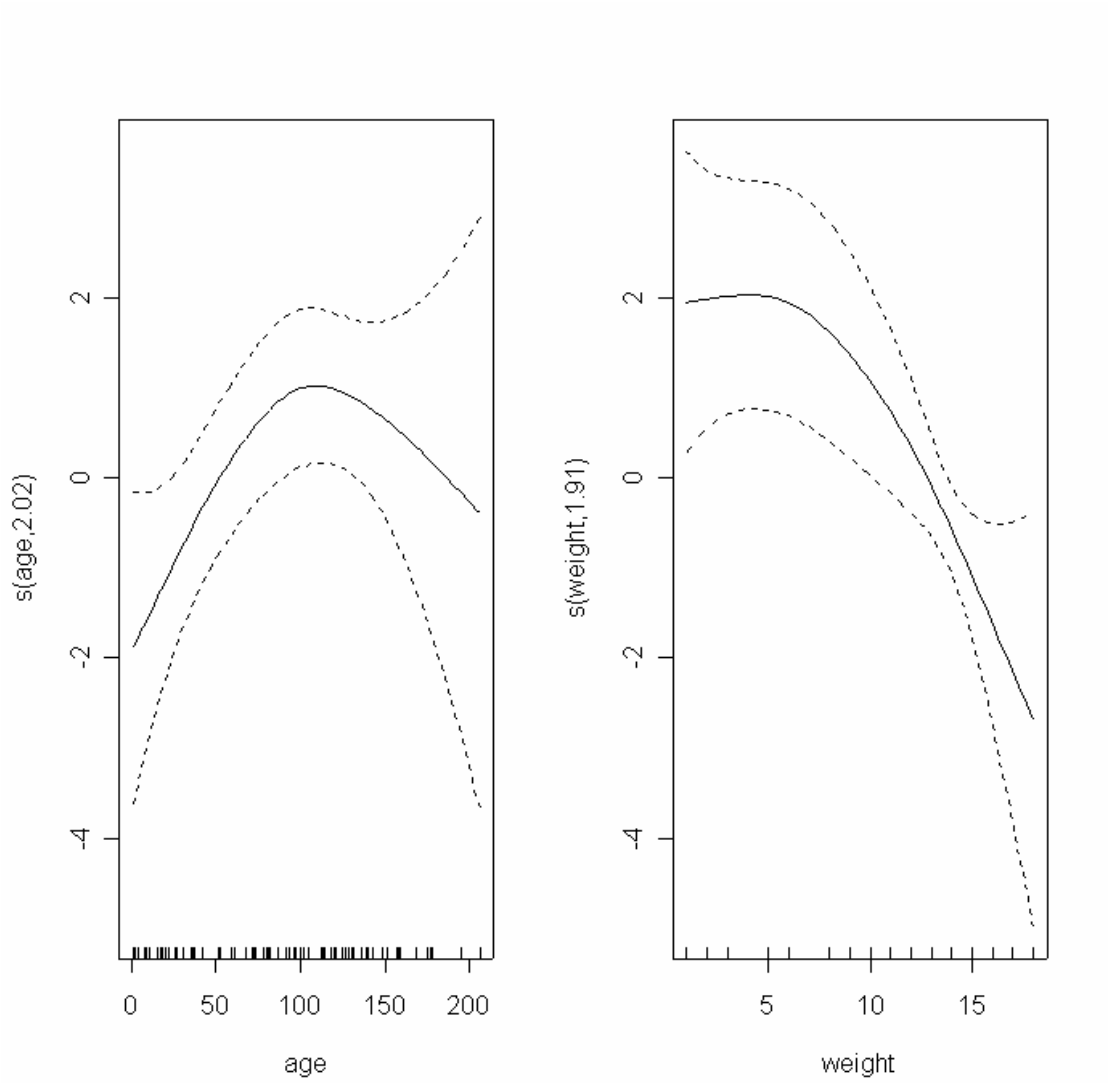
```
model13<-glm(infection~sex+age+weight+(weight^2),family=binomial)
anova(model11,model13,test="Chi")[,-1]
```

-I(age^2) p = 0.00990 and so is the age^2 term.

It is worth looking at these non-linearities in more detail, to see if we can do better with other kinds of models (e.g. non-parametric smoothers, piece-wise linear models or step functions).

An alternative approach is to use a **gam** rather than a **glm** for the continuous covariates.

```
library(mgcv)

gam1<-gam(infection~sex+s(age)+s(weight),family=binomial)

plot.gam(gam1)
```



These non-parametric smoothers are excellent at showing the humped relationship between infection and age, and the suggestion of a threshold at weight ≈ 12 in the relationship between weight and infection. We can now return to a **glm** to incorporate these ideas. We shall fit *age* and *age*^2 as before, but try a **piecewise linear fit** for *weight*, estimating the threshold weight as 12 (see above: we can try a range of values and see which gives the lowest deviance). The piecewise regression is specified by the term:

I((weight – 12) * (weight > 12))

The I is necessary to stop the * as being evaluated as an interaction term in the model formula. What this expression says is regress infection on the value of weight-12, but only do this when (weight>12) is true. Otherwise, assume that infection is independent of weight (that is to say, for values of weight less than or equal to 12) and produce a string of 0's (numeric FALSE) for the explanatory variable.

```
model14<-glm(infection~sex+age+I(age^2)+I((weight-
        12)*(weight>12)),family=binomial)
```

The effect of sex on infection is not quite significant (p = 0.07 for a chi square test on deletion), so we leave it out:

```
model15<-update(model14,~.-sex)

anova(model14,model15,test="Chi")

Analysis of Deviance Table

  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1        76     48.687
2        77     51.953 -1   -3.266     0.071

summary(model15)

Coefficients:
                                  Estimate Std. Error z value Pr(>|z|)
(Intercept)                     -3.1207575  1.2663090  -2.464  0.01372 *
age                              0.0765785  0.0323274   2.369  0.01784 *
I(age^2)                        -0.0003843  0.0001845  -2.082  0.03732 *
I((weight - 12) * (weight > 12)) -1.3511514  0.5112930  -2.643  0.00823 **

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 83.234  on 80  degrees of freedom
Residual deviance: 51.953  on 77  degrees of freedom
AIC: 59.953
```

This is our minimal adequate model. All the terms are significant when tested by deletion, the model is parsimonious, and the residuals are well behaved.

A 3x3 crossover trial

In a crossover experiment, the same set of treatments is applied to different individuals (or locations) but in different sequences ***to control for order effects***. So a set of farms might have predator control for 3 years followed by no control for 3 years, while another set may have three years of no control followed by 3 years of predator control. Crossover trials are most commonly used on human subjects, as in the present example of a drug trial on a new pain killer using a group of headache sufferers. There were 3 experimental treatments: a control pill that had no pain killer in it (the 'placebo'), neurofen, and brandA, the new product to be compared with the established standard. The interesting thing about crossover trials is that they involve repeated measures on the same individual, so a number of important questions need to be addressed:

1) does the order in which the treatments are administered to subjects matter?

2) if so, are there any carryover effects from treatments administered early on ?

3) are there any time series trends through the trial as a whole?

The response variable, y, is binary: 1 means pain relief was reported and 0 means no pain relief.

```
rm(y)

crossover<-read.table("c:\\temp\\crossover.txt",header=T)
attach(crossover)
names(crossover)

[1] "y" "drug" "order" "time"  "n"  "patient" “carryover"
```

In a case like this, data inspection is best carried out by summarising the totals of the response variable over the explanatory variables. Your first reaction might be to use **table** to do this, but this just counts the number of cases

```
table(drug)

 A  B  C
86 86 86
```

which shows only that all 86 patients were given each of the 3 drugs: A is the placebo, B neurofen and C BrandA. What we want is the sum of the y values (the number of cases of expressed pain relief) for each of the drugs. The tool for this is **tapply**: the order of its 3 arguments are tapply(response variable, explanatory variable(s), action) so we want

```
tapply(y,drug,sum)

 A  B  C
22 61 69
```

Before we do any complicated modelling, it is sensible to see if there are any significant differences between these 3 counts. It looks (as we hope that it would) as if pain relief is reported less frequently (22 times) for the placebo than for the pain killers (61 and 69 reports respectively). It is much less clear, however, whether BrandA (69) is significantly better than neurofen (61). Looking on the bright side, there is absolutely no evidence that it is significantly worse than neurofen ! To do a really quick chi square test on these three counts we just type:

```
glm(c(22,61,69)~1,family=poisson)
```

and we see

```
Degrees of Freedom: 2 Total (i.e. Null);  2 Residual
Null Deviance:      28.56
Residual Deviance: 28.56        AIC: 47.52
```

which is a chi square of 28.56 on just 2 degrees of freedom, and hence highly significant. Now we look to see whether there are any obvious effects of treatment sequence on reports of pain relief. In a 3-level crossover like this there are 6 orders: ABC, ACB, BAC, BCA, CAB and CBA. Patients are allocated to order at random.

There were different numbers of replications in this trial, which we can see using **table** on *order* and *drug*:

```
table(drug,order)
```

```
   1  2  3  4  5  6
A 15 16 15 12 15 13
B 15 16 15 12 15 13
C 15 16 15 12 15 13
```

which shows that replication varied from a low of 12 patients for order 4 (BCA) to a high of 16 for order 2 (ACB). A simple way to get a feel for whether or not order matters is to tabulate the total reports of relief by order and by drug:

```
tapply(y,list(drug,order),sum)
```

```
   1  2  3  4  5  6
A  2  5  5  2  4  4
B 13 13 10 10 10  5
C 12 13 13 10  9 12
```

Nothing obvious here. C (our new drug) did worst in order 5 (CAB) when it was administered first (9 cases), but this could easily be due to chance. We shall see. It is time to begin the modelling.

```
order<-factor(order)
```

With a binary response, we can make the single vector of 0's and 1's into the response variable directly, so we type:

```
model<-glm(y~drug*order+time,family=binomial)
```

There are lots of different ways of modelling an experiment like this. Here we are concentrating on the treatment effects and worrying less about the correlation and time series structure of the repeated measurements. We fit the main effects of drug (a 3-level factor) and order of administration (a 6-level factor) and their interaction (which has 2 x 5 = 10 d.f.). We also fit time to see if there is any pattern to the reporting of pain relief through the course of the experiment as a whole. This is what we find:

```
summary(model)
```

```
Call:
glm(formula = y ~ drug * order + time, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2642  -0.8576   0.5350   0.6680   2.0074

Coefficients:
             Estimate Std. Error z value  Pr(>|z|)
(Intercept)  -1.862803   1.006118  -1.851  0.064101 .
```

```
drugB         3.752595   1.006118   3.730  0.000192 ***
drugC         3.276086   0.771058   4.249 2.15e-005 ***
order2        1.083344   0.930765   1.164  0.244453
order3        1.187650   0.858054   1.384  0.166322
order4        0.280355   1.092937   0.257  0.797553
order5        0.869198   0.881368   0.986  0.324038
order6        1.078864   0.771058   1.399  0.161753
time         -0.008997   0.385529  -0.023  0.981383
drugB.order2 -1.479810   1.512818  -0.978  0.327985
drugC.order2 -1.012297   1.159837  -0.873  0.382776
drugB.order3 -2.375300   1.330330  -1.785  0.074181 .
drugC.order3 -0.702144   1.232941  -0.569  0.569026
drugB.order4 -0.551713   1.680933  -0.328  0.742747
drugC.order4 -0.066207   1.428863  -0.046  0.963043
drugB.order5 -2.038855   1.338492  -1.523  0.127697
drugC.order5 -1.868020   1.208188  -1.546  0.122072
drugB.order6 -3.420667   1.222788  -2.797  0.005151 **

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 349.42  on 257  degrees of freedom
Residual deviance: 268.85  on 240  degrees of freedom
AIC: 304.85
```

Remember that **with binary response data we are *not* concerned with overdispersion**. Looking up the list of coefficients, starting with the highest order interactions, it looks as if drugB.order6 with its t value of 2.768 is significant, and drugB.order3, perhaps. We test this by deleting the drug:order interaction from the model

```
model2<-update(model,~. - drug:order)

anova(model,model2,test="Chi")

Analysis of Deviance Table

Model 1: y ~ drug * order + time
Model 2: y ~ drug + order + time

  Resid. Df Resid. Dev  Df Deviance P(>|Chi|)
1       240    268.849
2       249    283.349  -9  -14.499     0.106
```

There is no compelling reason to retain this interaction (p > 10%). So we leave it out and test for an effect of time:

```
model3<-update(model2,~. - time)

anova(model2,model3,test="Chi")

Analysis of Deviance Table
Model 1: y ~ drug + order + time
Model 2: y ~ drug + order
```

```
  Resid. Df Resid. Dev  Df Deviance P(>|Chi|)
1       249    283.349
2       250    283.756  -1   -0.407     0.523
```

Nothing at all. What about the effect of order of administration of the drugs?

```
model4<-update(model3,~. - order)

anova(model3,model4,test="Chi")

Analysis of Deviance Table

Model 1: y ~ drug + order
Model 2: y ~ drug

  Resid. Df Resid. Dev  Df Deviance P(>|Chi|)
1       250    283.756
2       255    286.994  -5   -3.238     0.663
```

Nothing. What about the differences between the drugs?

```
summary(model4)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.0678     0.2469  -4.325 1.53e-05 ***
drugB         1.9598     0.3425   5.723 1.05e-08 ***
drugC         2.4687     0.3664   6.739 1.60e-11 ***

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 349.42  on 257  degrees of freedom
Residual deviance: 286.99  on 255  degrees of freedom
AIC: 292.99
```

Notice that, helpfully, the names of the treatments appear in the coefficients list . Now it is clear that our drug C had log odds greater than neurofen by 2.4687 – 1.9598 = 0.5089 . The standard error of the difference between two treatments is about 0.366 (taking the larger (less well replicated) of the two values) so a t test between neurofen and our drug gives t = 0.5089/0.366 = 1.39. This is not significant.

The coefficients are now simply the logits of our original proportions that we tabulated during data exploration: 22/86, 61/86 and 69/86 (you might like to check this). The analysis tells us that 69/86 is not significantly greater than 61/86. If we had known to begin with that none of the interactions, sequence effects or temporal effects had been significant, we could have analysed these results much more simply, like this

```
relief<-c(22,61,69)
patients<-c(86,86,86)
treat<-factor(c("A","B","C"))
```

```
model<-glm(cbind(relief,patients-relief)~treat,binomial)

summary(model)

Coefficients:
            Estimate Std. Error z value  Pr(>|z|)
(Intercept)  -1.0678     0.2471  -4.321 1.55e-005 ***
treatB        1.9598     0.3427   5.718 1.08e-008 ***
treatC        2.4687     0.3666   6.734 1.65e-011 ***

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6.2424e+001  on 2  degrees of freedom
Residual deviance: 1.5479e-014  on 0  degrees of freedom
AIC: 19.824
```

Treatments B and C differ by only about 0.5 with a standard error of 0.367, so we may be able to combine them. Note the use of != for “not equal to”

```
t2<-factor(1+(treat != "A"))

model2<-glm(cbind(relief,patients-relief)~t2,binomial)

anova(model,model2,test="Chi")

Analysis of Deviance Table

Model 1: cbind(relief, patients - relief) ~ treat
Model 2: cbind(relief, patients - relief) ~ t2

  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1         0  6.376e-15
2         1    2.02578 -1 -2.02578   0.15465
```

We can reasonably combine these two factor levels, so the minimum adequate model as only 2 parameters, and we conclude that the new drug is not significantly better than neurofen. But if they cost the same, which of them would *you* take if you had a headache?

STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

11. ANALYSING COUNT DATA: POISSON ERRORS

Up to Practical 9, the data were all continuous measurements like weights, heights, lengths, temperatures, growth rates and so on. A great deal of the data collected by scientists, medical statisticians and economists, however, is in the form of *counts* (whole numbers or integers). The number of individuals that died, the number of firms going bankrupt, the number of days of frost, the number of red blood cells on a microscope slide, or the number of craters in a sector of lunar landscape are all potentially interesting variables for study. With count data, the number 0 often appears as a value of the response variable (consider, for example, what a 0 would mean in the context of the examples just listed).

For our present purposes, it is useful to think of count data as coming in four types:

- data on *frequencies*, where we count how many times something happened, but we have no way of knowing how often it did not happen (e.g. lightening strikes, bankruptcies, deaths, births)
- data on *proportions*, where both the number doing a particular thing, and the total group size are known (insects dying in an insecticide bioassay, sex ratios at birth, proportions responding in a questionnaire)
- *category* data, in which the response variable is a count distributed across a categorical variable with two or more levels
- *binary* response variables (dead or alive, solvent or insolvent, infected or immune)

We dealt with proportion data and binary response variables in Practical 10. Here we are concerned with pure counts rather than proportions. Straightforward linear regression methods (constant variance, normal errors) are not appropriate for count data for 5 main reasons:

- the linear model might lead to the prediction of negative counts
- the variance of the response variable is likely to increase with the mean
- the errors will not be normally distributed
- zeros are difficult to handle in transformations
- some distributions (e.g. log-normal or gamma) don't allow zeros

In S-Plus, count data are handled very elegantly in a **glm** by specifying **family=poisson** which sets errors = Poisson and link = log. The log link ensures that

all the fitted values are positive, while the Poisson errors take account of the fact that the data are integer and have variances equal to their means.

The Poisson distribution

The Poisson distribution is widely used for the description of count data that refer to cases where we know how many times something happened (e.g. kicks from cavalry horses, lightening strikes, bomb hits), but we have no way of knowing how many times it did not happen. This is in contrast to the binomial distribution (Practical 10) where we know how many times something did not happen as well as how often it did happen (e.g. if we got 6 heads out of 10 tosses of a coin, we must have got 4 tails).

The Poisson is a 1-parameter distribution, specified entirely by the mean. The variance is identical to the mean, so the variance/mean ratio is equal to one. Suppose we are studying the ecology of a leaf miner on birch trees, and our data consist of the numbers of mines per leaf (*x*). Many leaves have no mines at all, but some may have as many as 5 or 6 mines. If the mean number of mines per leaf is λ, then the probability of observing *x* mines per leaf is given by:

$$P(x) = \frac{e^{-\lambda}\lambda^{x}}{x!}$$

This can be calculated very simply on a hand calculator because:

$$P(x) = P(x-1)\frac{\lambda}{x}$$

This means that if you start with the *zero term*

$$P(0) = e^{-\lambda}$$

then each successive probability is obtained simply by multiplying by the mean and dividing by *x*.

Because the data are whole numbers (integers), it means that the residuals (y - the fitted values) can only take a restricted range of values. If the estimated mean was 0.5, for example, then the residuals for counts of 0, 1, 2, and 3 could only be -0.5, 0.5, 1.5 and 2.5. The normal distribution assumes that the residuals can take any values (it is a continuous distribution).

Similarly, the normal distribution allows for negative fitted values. Since we can not have negative counts, this is clearly not appropriate. SPlus deals with this by using the log link function when Poisson errors are specified

$$y = \exp(\sum \beta_i x_i)$$

so the fitted values are antilogs and can never go negative. Even if the linear predictor was $-\infty$ the fitted value would be exp(-∞) = 0 .

A further difference from the examples in previous chapters is that SPlus uses maximum likelihood methods other than least squares to estimate the parameters values and their standard errors. Given a set of data and a particular model, then the maximum likelihood estimates of the parameter values are *those values that make the observed data most likely* (hence the name maximum likelihood).

Because the Poisson is a one-parameter distribution (the mean is equal to the variance), SPlus does not attempt to estimate a scale parameter (it is set to a default value of 1.0). If the error structure of the data really is Poisson, then the ratio of residual deviance to degrees of freedom after model-fitting should be 1.0. If the ratio is substantially greater than one, then the data are said to show overdispersion, and remedial measures may need to be taken (e.g. the use of an empirical scale parameter or the specification of negative binomial errors).

With Poisson errors, the change in deviance attributable to a given factor is distributed asymptotically as chi-squared. This makes hypothesis testing extremely straightforward. We simply remove a given factor from the maximal model and note the resulting change in deviance and in the degrees of freedom. If the change in deviance is larger than the critical value of chi squared (**qchisq**) we retain the term in the model, while if the change in deviance is less than the value of chi squared in tables, the factor is insignificant and can be left out of the model.

Thus, the only important differences you will need to remember in modelling with Poisson rather than normal errors are:

- use **glm** rather than **aov** or **lm**
- the **family=poisson** directive must be specified (but the "family =" bit is optional)
- hypothesis testing involves deletion followed by chi-squared tests
- beware of overdispersion, and correct for it if necessary
- do not collapse contingency tables over explanatory variables

Deviance with Poisson errors

Up to this point, lack of fit has always been measured by SSE; the residual or error sum of squares

$$\text{SSE} = \sum (y - \hat{y})^2$$

where $\hat{y}$ are the fitted values estimated by the model. With **glm**'s SSE is only the maximum likelihood estimate of lack of fit when the model has normal errors and the identity link (in which case, you have to ask "why am I doing a glm?"). Generally, we use *deviance* to measure lack of fit in a **glm**. For Poisson errors the deviance is this:

$$\text{Poisson deviance} = 2\sum O \ln\left[\frac{O}{E}\right]$$

where O is the Observed count, and E is the Expected count as predicted by the current model. Let's see how this works. Suppose you have 4 counts, 2 from each of 2 locations. Location A produced counts of 3 and 6. Location B produced counts of 4

and 7. The total deviance (like SST) is based on the whole sample of 4 numbers. The expected count is the overall mean

```
mean(c(3,6,4,7))
```

```
[1] 5
```

This allows us to calculate the total deviance:

$$2 \times \left\{ 3 \times \ln\left[\frac{3}{5}\right] + 6 \times \ln\left[\frac{6}{5}\right] + 4 \times \ln\left[\frac{4}{5}\right] + 7 \times \ln\left[\frac{7}{5}\right] \right\}$$

Calculating the value gives

```
2*(3*log(3/5)+6*log(6/5)+4*log(4/5)+7*log(7/5))
```

```
[1] 2.048368
```

Now the 2 different location means are 9/2 = 4.5 and 11/2 =5.5 respectively. The residual deviance after fitting a 2–level factor for location should therefore be

$$2 \times \left\{ 3 \times \ln\left[\frac{3}{4.5}\right] + 6 \times \ln\left[\frac{6}{4.5}\right] + 4 \times \ln\left[\frac{4}{5.5}\right] + 7 \times \ln\left[\frac{7}{5.5}\right] \right\}$$

```
2*(3*log(3/4.5)+6*log(6/4.5)+4*log(4/5.5)+7*log(7/5.5))
```

```
[1] 1.848033
```

Given that the total deviance is 2.048 and the residual deviance is 1.848 we calculate the treatment deviance (due to differences between locations) as 2.048 – 1.848 = 0.2. Let's see if a **glm** with Poisson errors gives the same answers. Data entry first

```
y<-c(3,6,4,7)
```

```
location<-factor(c("A","A","B","B"))
```

now the statistical modelling

```
glm(y~location,poisson)
```

```
Degrees of Freedom: 3 Total (i.e. Null);  2 Residual
Null Deviance:      2.048
Residual Deviance: 1.848        AIC: 19.57
```

So far so good. The residual deviance (1.848) is as we calculated it to be. How about the total and treatment deviances ? We can get the total deviance if we fit the null model y ~ 1

```
glm(y~1,family=poisson)
```

```
Degrees of Freedom: 3 Total (i.e. Null);  3 Residual
Null Deviance:      2.048
Residual Deviance: 2.048        AIC: 17.77
```

So there is no mystery to the values of deviance. We could calculate them if we wanted to, at least for models as simple as this (more complex models require *iterative fits* to estimate the parameter values).

Using Poisson errors in modelling

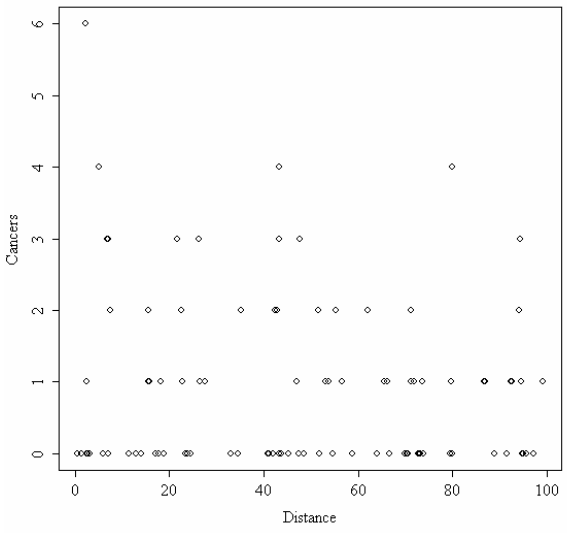
1) Continuous explanatory variables with count data: log-linear regression

This example involves counts of prostate cancer (cancer 'clusters') and distance from a nuclear processing plant. The response variable contains many zeros and is completely unsuitable for standard regression analysis. The single explanatory variable is continuous: distance to the nuclear plant from the clinic where the diagnosis was made (not from where the patients actually lived or worked). The issue is whether or not the data provide any evidence that the number of cancers increases with proximity to the plant.

```
clusters<-read.table("c:\\temp\\clusters.txt",header=T)
attach(clusters)
names(clusters)
```

```
[1] "Cancers"  "Distance"
```

```
plot(Distance,Cancers)
```



The first thing that you notice about plots of count data is that all the data are in sharp horizontal rows reflecting the integer values of the response. There are lots of zero's at all distances away from the nuclear plant. The largest value (the cluster of 6 cases) is close to the plant. But is there evidence for any trend in the number of cases as distance from the plant increases ?

The modelling is exactly like any other regression except that we replace **lm** by **glm**, and add the phrase family=poisson after the model formula (family= is optional):

```
model<-glm(Cancers~Distance,family=poisson)
summary(model)
```

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.186865   0.188728   0.990   0.3221
Distance    -0.006138   0.003667  -1.674   0.0941 .
```

There is a negative trend in the data (slope = -0.00614) but it is not significant under a 2-tailed test (t = 1.68). If you were desperate, you might say that a 1-tailed test is appropriate, because we expected *a priori* that the slope would be negative. I don't buy that. Anyway, we are not finished yet. Poisson errors is an assumption not a fact.

```
(Dispersion parameter for poisson family taken to be 1 )
```

We need to test for overdispersion. This would be evidenced if the residual deviance was larger than the residual degrees of freedom:

```
Null deviance: 149.48  on 93  degrees of freedom
Residual deviance: 146.64  on 92  degrees of freedom
AIC: 262.41
```

The dispersion parameter is actually 146.64 / 92 = 1.594 and we should make allowance for this overdispersion in our analysis. As a rule of thumb, the standard error of the parameters will increase as $\sqrt{\text{Dispersion parameter}}$ (i.e. they will be 1.26–fold larger in this case). A better way to adjust for overdispersion is to **use an F test with an empirical scale parameter instead of a chi square test**. This is carried out in R by using the family **quasipoisson** in place of poisson errors. We can accomplish this by deleting distance from the model (fitting only the intercept, ~1), then comparing the model fits using **anova** in which we specify test="F"

```
model<-glm(Cancers~Distance,family=quasipoisson)
model2<-glm(Cancers~1,family=quasipoisson)
anova(model,model2,test="F")
```

```
Analysis of Deviance Table

Model 1: Cancers ~ Distance
Model 2: Cancers ~ 1
  Resid. Df Resid. Dev Df Deviance      F Pr(>F)
1        92    146.643
2        93    149.484 -1   -2.841 1.8269 0.1798
```

There is no evidence for a decline in the number of cancers with distance. An F value as large as 1.83 will arise by chance alone with probability p = 0.18 when there is no trend in cancers with distance.

Analysis of deviance: categorical explanatory variables with count data

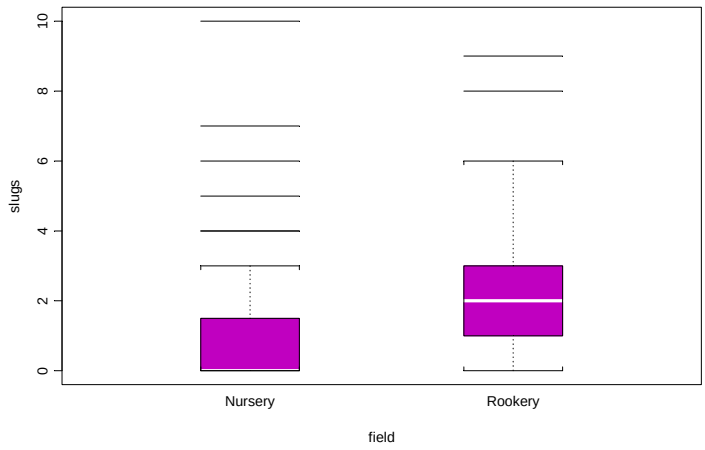
Count data were obtained on the numbers of slugs found beneath 40 tiles placed in a stratified random grid over each of two permanent grasslands. This was part of a pilot study in which the question was simply whether the mean slug density differs significantly between the two grasslands.

```
slugsurvey<-read.table("c:\\temp\\slugsurvey.txt",header=T)
attach(slugsurvey)
names(slugsurvey)
```

```
[1] "slugs" "field"
```

These count data seem ideally suited for analysis using **glm** with Poisson errors.

```
plot(field,slugs)
```



It certainly looks as if median slug numbers are higher in Rookery Field than in Nursery Field, but the range of counts is very large in both fields, so the significance of the difference must be in some doubt. We fit the model first as a simple 1-way analysis of deviance, using Poisson errors and the log link:

```
model<-glm(slugs~field,poisson)
```

```
summary(model)
```

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)    0.2429     0.1399   1.737 0.082446 .
fieldRookery   0.5790     0.1748   3.312 0.000925 ***
```

The t-test for difference between fields is highly significant, so it looks like mean slug numbers really are different. But just a minute. We have not yet checked for overdispersion.

```
(Dispersion parameter for poisson family taken to be 1)
```

```
    Null deviance: 224.86  on 79  degrees of freedom
Residual deviance: 213.44  on 78  degrees of freedom
AIC: 346.26
```

Oops! The scale parameter (or Dispersion Parameter as it is labelled here) is 213.44/78 = 2.74, not 1 as was assumed by the model. The simplest thing we can do to compensate for this is to carry out an F test rather than a chi square test of deletion. The F test uses the empirical scale parameter as an estimate equivalent to the error variance, and performs a test much harsher than the chi square test. We can compare the two: chi square first:

```
model2<-update(model,~.-field)

anova(model,model2,test="Chi")
```

```
Analysis of Deviance Table

Model 1: slugs ~ field
Model 2: slugs ~ 1
  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1        78    213.438
2        79    224.859 -1  -11.422     0.001
```

This suggests that the difference between the fields is highly significant (p = 0.001). Now do exactly the same deletion, but use the F test with the empirical scale parameter. This requires that we re-fit the model using **family = quasipoisson**:

```
model<-glm(slugs~field,quasipoisson)
model2<-update(model,~.-field)
anova(model,model2,test="F")
```

```
Analysis of Deviance Table

Model 1: slugs ~ field
Model 2: slugs ~ 1

  Resid. Df Resid. Dev Df Deviance      F  Pr(>F)
1        78    213.438
2        79    224.859 -1  -11.422 3.6041 0.06134 .
```

A big change. Under the F test the difference in mean slug density is **not** significant.

We could try a parametric transformation, and analyse the data using a linear model (**aov**). For instance,

```
model3<-aov(log(slugs+1)~field)

summary(model3)
```

```
            Df Sum of Sq  Mean Sq  F Value    Pr(F)
```

```
    field  1    4.4750 4.475004 8.961176 0.003692791
Residuals 78   38.9514 0.499377
```

This says the difference is highly significant ($p < 0.004$), so clearly the adjustment for overdispersion in the F test is extremely unforgiving.

The alternative parametric transformation for count data is the square root. Here we compare a straightforward analysis of variance on the raw count data (normal errors and constant variance (wrongly) assumed) with a model based on the square root transformation.

```
model4<-aov(slugs~field)

anova(model4)

Analysis of Variance Table

Response: slugs

Terms added sequentially (first to last)
          Df Sum of Sq  Mean Sq  F Value      Pr(F)
    field  1     20.00 20.00000 3.900488 0.05181051
Residuals 78    399.95  5.12756
```

The untransformed anova suggests that the difference is not quite significant. Now for a square root transformation of the counts, assuming (now with much greater justification) that the errors are normal and the variance constant:

```
model5<-aov(slugs^0.5~field)

anova(model5)

Analysis of Variance Table
Response: slugs^0.5

          Df Sum of Sq  Mean Sq  F Value       Pr(F)
    field  1   7.44812 7.448123 9.312308 0.003110717
Residuals 78  62.38557 0.799815
```

So the linear model with square root transformed counts indicates that the difference in slug density between the two fields is highly significant.

The next option in dealing with overdispersion is to use **quasi** to define a different family of error structures. For instance, we might retain the log link (this is good for constraining the predicted counts to be non-negative), but we might allow that the variance increases with the square of the mean (like a discrete version of a gamma distribution), rather than as the mean (as assumed by Poisson errors). This is how we write the model :

```
model6<-glm(slugs~field,family=quasi(link=log,variance=mu^2))
```

The software recognises “mu” as the mean of the distribution. The result of the fit is this:

```
summary(model6)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.2429     0.2300   1.056   0.2941
fieldRookery   0.5790     0.3252   1.780   0.0789 .

Dispersion parameter for quasi family taken to be 2.115615)
```

Note the estimate of the Dispersion Parameter, above. The t-test does not indicate a significant difference between fields. Model checking plots indicate that the errors are much more nearly normal under the log(slugs+1) transformation than in the quasi model.

```
    Null deviance: 40.144  on 79  degrees of freedom
Residual deviance: 45.606  on 78  degrees of freedom
AIC: NA
```

Let’s check the deletion test.

```
model7<-glm(slugs~1,family=quasi(link=log,variance=mu^2))
anova(model6,model7,test="F")
```

```
Analysis of Deviance Table

Model 1: slugs ~ field
Model 2: slugs ~ 1
  Resid. Df Resid. Dev Df Deviance      F Pr(>F)
1        78     45.606
2        79     40.144 -1    5.462 2.5815 0.1122
```

Not significant. What about a non-parametric test? We can use the Wilcoxon rank sum test to compare the two (unpaired) samples. Load the “classic tests” library:

```
library(ctest)
```

```
wilcox.test(slugs[field=="Nursery"],slugs[field=="Rookery"])
```

Notice that we need to split the response variable into two separate vectors, one for each field, in order to use this function (in general, the 2 vectors might be of different lengths).

```
Wilcoxon rank-sum test

data:slugs[field=="Nursery"]and slugs[field=="Rookery"]

rank-sum normal statistic with correction Z = -3.0581,
p-value = 0.0022
alternative hypothesis: true mu is not equal to 0
```

The non-parametric test says that mean slug numbers are significantly different in the 2 fields ($p = 0.0022$). **This p value is not particularly reliable, however, because there are so many ties in the data** (all those zeros).

Thus, several of the tests suggest that the differences between the mean slug densities are highly significant, while other tests suggest that the difference is insignificant. What is clear is that there is more to this than mere differences between the fields. Differences between the means explain only 5% of the variation in slug counts from tile to tile (deviance change of 11.42 out of 224.86). Within fields it is clear that the data are highly aggregated. It is very common for field data to have this kind of overdispersed, spatially aggregated structure, and it would be naive of us to assume that simple error structures will always work perfectly.

This kind of problem, where one kind of test says a difference is significant and another test says the same difference is not significant, comes up all the time when dealing with data with a low mean and a high variance. There is no obvious right answer, but the analyses correcting for overdispersion are clearly signalling that the result, so significant with linear models, should be treated with a more than usual degree of circumspection.

Summary of Overdispersion with Poisson errors

Overdispersion occurs when the residual deviance is greater than the residual degrees of freedom once the minimal adequate model has been fit to the data. It means that the errors are not, in fact, Poisson (i.e. equal to the mean) but are actually greater than assumed. This means that the estimated standard errors are too small, and the significance of model terms is more or less severely overestimated. We need to take care of overdispersion, and there are several options we can follow. In order of simplicity, these are

1) carry out significance tests using "**F**" rather than "**Chi**" in the **anova** directive after specifying family = **quasipoisson** rather than poisson

2) use **family = quasi** instead of family = poisson and specify a variance function

3) use negative binomial errors

ANCOVA with count data: categorical and continuous explanatory variables.

A long-term agricultural experiment had 90 grassland plots, each 25m x 25m, differing in biomass, soil pH and species richness (the count of species in the whole plot). It is well known that species richness declines with increasing biomass, but the question addressed here is whether the slope of that relationship differs with soil pH. The plots were classified according to a 3-level factor as high, medium or low pH with 30 plots in each level. The response variable is the count of species, so a **glm** with Poisson errors is a sensible choice. The continuous explanatory variable is long-term average biomass measured in June, and the categorical explanatory variable is soil pH. With a mixture of continuous and categorical explanatory variables, analysis of covariance is the appropriate method.

```
species<-read.table("c:\\temp\\species.txt",header=T)
attach(species)
names(species)
```

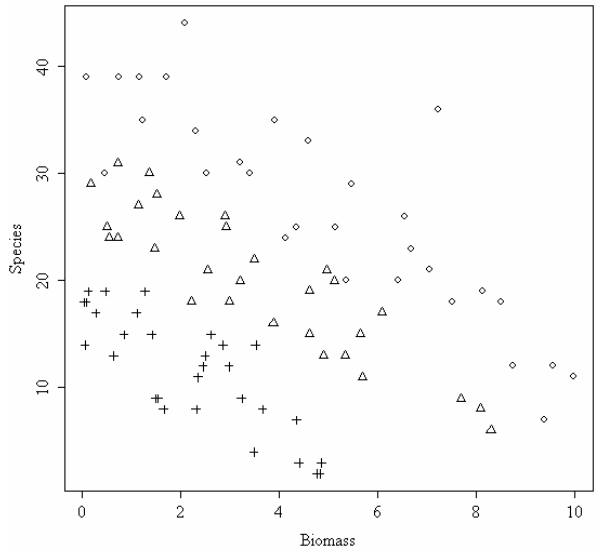
```
[1] "pH"      "Biomass" "Species"
```

We begin by plotting the data, using different symbols for each level of soil pH. It is important in cases like this to ensure that the axes in the first plot directive are scaled appropriately to accommodate all of the data. The trick here is to plot the axes with nothing between them: the **type="n"** option:

```
plot(Biomass,Species,type="n")
```

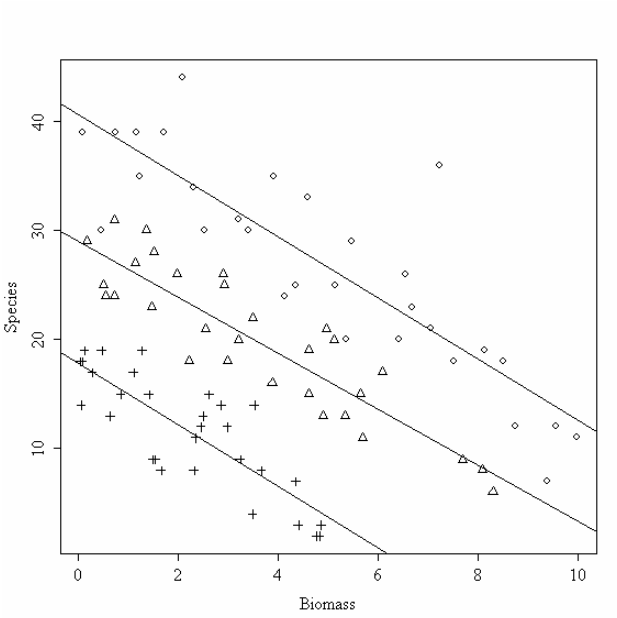
Now we can add a scatterplot of points for each level of soil pH, using different plotting characters **pch** for each:

```
points(Biomass[pH=="high"],Species[pH=="high"])
points(Biomass[pH=="mid"],Species[pH=="mid"],pch=2)
points(Biomass[pH=="low"],Species[pH=="low"],pch=3)
```



To help interpret the plot, we can fit linear regression lines through the clouds of points for each level of soil pH using **abline**. We shall not use linear regression in the statistical modelling, if only because a linear model would predict nonsense values (e.g. negative counts of species at high biomass). Note that the order of the variables is reversed in the lm directive (y then x) and that the variables are separated by ~ (tilde).

```
abline(lm(Species[pH=="high"]~Biomass[pH=="high"]))
abline(lm(Species[pH=="mid"]~Biomass[pH=="mid"]))
abline(lm(Species[pH=="low"]~Biomass[pH=="low"]))
```



There is certainly a clear difference in mean species richness with declining soil pH, but there is little evidence of any substantial difference in the slope of the relationship between species richness and biomass on soils of differing pH. Now we shall do the modelling properly, as a **log-linear analysis of covariance.**

The procedure is exactly the same as in standard analysis of covariance: we fit a full model with different slopes and intercepts for each factor level, then we simplify the model by assessing whether a common slope would describe the data equally well. The only difference is that we replace **lm** by **glm** and add the phrase family=poisson after the model formula:

```
model1<-glm(Species~pH*Biomass,poisson)
model2<-update(model1,~.-pH:Biomass)
```

Don’t forget the punctuation in update: comma tilde dot minus. Now we can get a full anova table comparing the 2 different models, with and without the interaction term (differences between slopes):

```
anova(model1,model2,test="Chi")
```

```
Analysis of Deviance Table

Model 1: Species ~ pH * Biomass
Model 2: Species ~ pH + Biomass
  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1        84     83.201
2        86     99.242 -2  -16.040 0.0003288
```

Our initial impression that the slopes were the same was completely wrong: in fact, the slopes are very significantly different ($p < 0.00035$). So we need to retain a model with different slopes. Note that the model containing different slopes (Model 1) is not overdispersed (residual deviance = 83.2 on 84 d.f.), so we do not need to correct by using quasipoisson.

Plotting the fitted values as smooth lines, separately for each level of soil pH, demonstrates the boundedness of log linear models. Our first plot, using linear regression, absurdly predicted negative values for species richness at high biomass. In contrast, the **glm** predicts species richness declining asymptotically towards low values. In reality of course, it is impossible that plant species richness could ever fall below 1 so long as the plots were vegetated.

```
plot(Biomass,Species,type="n")
```

Now we can add a scatterplot of points for each level of soil pH, using different plotting characters pch for each:

```
points(Biomass[pH=="high"],Species[pH=="high"])
points(Biomass[pH=="mid"],Species[pH=="mid"],pch=2)
points(Biomass[pH=="low"],Species[pH=="low"],pch=3)
```

The curve fitting requires that we make a data frame to contain values for all of the explanatory variables (the continuous variable Biomass for the x axis and the categorical explanatory variable pH for the 3 different graphs). First we generate the values for the x axis

```
x<-seq(0,10,0.1)
length(x)
```

```
[1] 101
```

This means that we need to generate101 repeats of each of the factor levels for soil pH

```
levels(pH)
```

```
[1] "high" "low"  "mid"
```

We created a new vector of length 3 x 101 to contain the factor levels:

```
acid<-factor(c(rep("low",101),rep("mid",101),rep("high",101)))
```

and now we need to make the vector of x values (Biomass) the same length

```
x<-c(x,x,x)
```

Finally, we use **predict** to predict the fitted values. We could back transform the predicted values manually (using the antilog function exp) but here we use type="response" to do this for us.

```
yv<-predict(model1,list(Biomass=x,pH=acid),type="response")
```

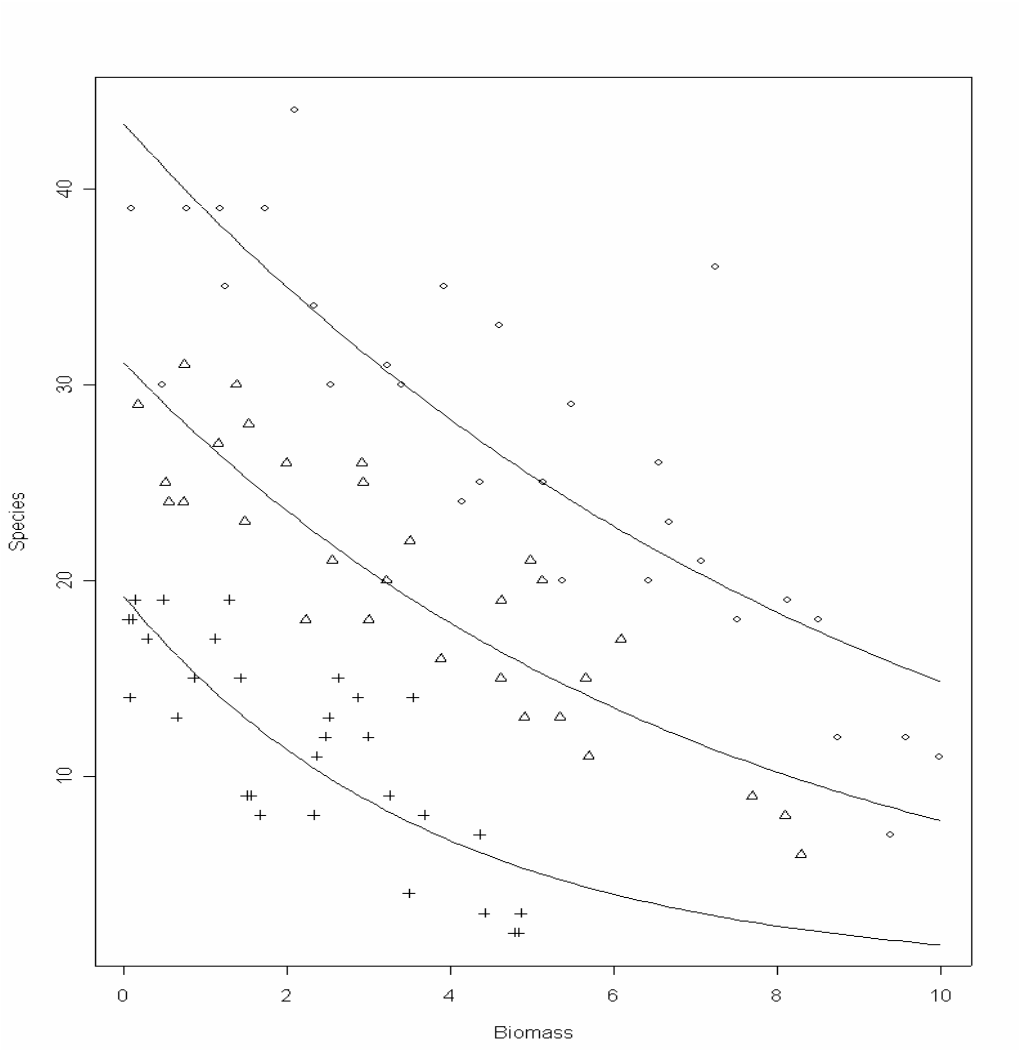
With complicated plots like this, where we want to fit different curves for different levels of one or more factors, it is useful to employ the split function to reduce the y-values and x-values into separate lists (one for each fitted line)

```
yvs<-split(yv,acid)
bvs<-split(x,acid)
```

Now we can fit the three lines. Note the use of double subscripts, [[1]], because the split function produces **lists** not vectors:

```
lines(bvs[[1]],yvs[[1]])
lines(bvs[[2]],yvs[[2]])
lines(bvs[[3]],yvs[[3]])
```

```
detach(species)
rm(species)
```



3) Categorical explanatory variables with count data: Contingency Tables

a) Fisher's Exact Test

This test is used for the analysis of contingency tables in which **one or more expected frequencies is less than 5**. The individual counts are a, b, c and d like this:

2x2 Table	Col. 1	Col. 2	Row totals
Row 1	a	b	a+b
Row 2	c	d	c+d
Column Totals	A+c	b+d	n

The probability of any one outcome is this

$$p = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{a!b!c!d!n!}$$

where n is the grand total. Our example concerns a test with ant colonies forming nests (nor not) on two species of trees:

	Tree A	Tree B	Row totals
With ants	6	2	8
Without ants	4	8	12
Column totals	10	10	20

It is easy to calculate the probability of this particular outcome (but **not in R** because, for some obscure reason, it doesn't have a factorial function!). This means that we need to start by writing a factorial function of our own:

```
factorial<-function(x) max(cumprod(1:x))
```

Now we can work out the probabilities for Fisher's Exact Test:

```
factorial(8)*factorial(12)*factorial(10)*factorial(10) /
    (factorial(6)*factorial(2)*factorial(4)*factorial(8)*factorial(20))
```

```
[1] 0.07501786
```

but that is only part of the story. We need to compute the probability of outcomes more extreme than this. There are two of them. Suppose only 1 ant colony had been found on Tree B. Then the table values would be 7, 1, 3, 9 but the row and column totals would be exactly the same (*the marginal totals are constrained*). The numerator always stays the same, so this case has probability

```
factorial(8)*factorial(12)*factorial(10)*factorial(10)/
    (factorial(7)*factorial(3)*factorial(1)*factorial(9)*factorial(20))
```

```
[1] 0.009526078
```

There is an even more extreme case if no ant colonies at all had been found on Tree B. Now the table elements become 8, 0, 2, 10 with probability

```
factorial(8)*factorial(12)*factorial(10)*factorial(10)/
    (factorial(8)*factorial(2)*factorial(0)*factorial(10)*factorial(20))
```

```
[1] 0.0003572279
```

and we need to add these 3 probabilities together

```
0.07501786+0.009526078+0.000352279
```

```
[1] 0.08489622
```

But there was no *a priori* reason for expecting the result to be in this particular direction (more ants on A than B). We need to allow for extreme counts in the opposite direction (more ants on B than on A), by doubling this probability (in fact, all Fisher's Exact Tests are 2-tailed).

```
2*(0.07501786+0.009526078+0.000352279)
```

```
[1] 0.1697924
```

We conclude that there is no evidence of any association between Tree Type and Ant Colonies. The observed pattern could have arisen by chance alone with probability = 0.17.

There is a built in function called **fisher.test**, which saves us all this computation. First, however, we need get the "classical tests" from the library called "ctest"

```
library(ctest)
```

Fishers exact test takes as its argument a 2x2 matrix containing the counts of the 4 contingencies. We make the matrix (column-wise) like this

```
x<-as.matrix(c(6,4,2,8))
```

```
dim(x)<-c(2,2)
```

To see what the matrix looks like type x

```
     [,1] [,2]
[1,]    6    2
[2,]    4    8
```

and run the test like this

```
fisher.test(x)
```

```
     Fisher's exact test

data:  x
p-value = 0.1698
alternative hypothesis: two.sided
```

It gives the same non-significant *p* value that we calculated longhand (above).

b) A simple 2x2 contingency table: the G-test

The simplest analysis of count data is the 2-by-2 contingency table. The traditional analysis of such tables used Pearson's chi-square to test the null hypothesis that two factors were independent in their effects on the response variable. Here, we show the alternative (often called the G-test) which uses log linear models to address the same question. As with Pearson, the test statistic is chi square with 1 degree of freedom.

Suppose we have the following contingency table of counts of oak trees supporting two species of cynipid gall formers

	Andricus present	*Andricus* absent
Biorhiza present	13	44
Biorhiza absent	25	29

The response variable is a count of the number of trees falling into each of 4 categories: with and without the 2 gall formers, *Andricus* and *Biorhiza*. The question to be addressed is whether there is any evidence of ecological association or separation between these two insects on different oak trees. We set up the data like this. There are 3 variables: the response, which we'll call *count*, a 2-level factor (presence or absence) for *Biorhiza*, and a 2-level factor (presence or absence) for *Andricus*. The data frame is so small, we may as well type in the values directly:

```
count<-c(13,44,25,29)
Biorhiza<-factor(c("present","present","absent","absent"))
Andricus<-factor(c("present","absent","present","absent"))
```

Carrying out the log-linear modelling (the **glm** with Poisson errors) is straightforward. We give a name to the model object (*ct* stands for contingency table), then specify the model formula in the usual way. The only extra detail is that we need to specify family=poisson:

```
ct<-glm(count~Biorhiza+Andricus,poisson)
```

We shall not bother with the summary of *ct*, because most of the information is superfluous to present purposes. All we need is the value of the residual deviance:

```
ct
```

```
Call:  glm(formula = count ~ Biorhiza + Andricus, family
= poisson)


Degrees of Freedom: 3 Total (i.e. Null);  1 Residual
Null Deviance:           18.19
Residual Deviance: 6.878     AIC: 33.19
```

The coefficients are of no interest. All we want is the value of the residual deviance (6.878) to compare with chi square tables with 1 d.f. (3.841). Because the calculated value of chi square is larger than the value in tables, we reject the null hypothesis of independence in the distribution of these two gall formers.

Note that the test does not tell us whether there is a positive or a negative correlation between the 2 species' distributions. For this, we need to look at the data and the fitted values (these are the counts we would have expected if the distributions really had been independent):

```
fitted(ct)

    1         2         3         4
 19.51355 37.48649 18.48652 35.51351

count

[1] 13     44         25         29
```

This tells us that if the distributions had been independent, then we should have expected to find the two galls together on the same tree on more than 19 occasions. We actually found them together only 13 times, so the significant association between the two taxa is negative. Positive correlations would be indicated when the observed count of co-occurrence was greater than the expected frequency.

c) A complex contingency table: Schoener's lizards

It is all too easy to analyse complex contingency tables in the wrong way and to produce answers that are actually not supported by the data. Problems typically occur because people fail (or forget) to include all the interactions between the nuisance variables that are necessary to constrain the marginal totals. Note that the problems almost never arise if the same analysis can be structured so that it can be carried out as the analysis of proportion data using binomial errors (compare the example explained below with its re-analysis as proportion data later on, where there are no nuisance variables at all).

The example concerns niche differences between two lizard species:

```
lizards<-read.table("c:\\temp\\lizards.txt",header=T)
attach(lizards)
names(lizards)

[1] "n"    "sun"    "height"  "perch"    "time"     "species"
```

Schoener collected information on the distribution of two *Anolis* lizard species (*A. opalinus* and *A. grahamii*) to see if their ecological niches were different in terms of where and when they perched to prey on insects. Perches were classified by twig diameter, their height in the bush, whether the perch was in sun or shade when the lizard was counted, and the time of day at which they were foraging. The response variable is a count of the number of times a lizard of each species was seen in each of the contingencies. The modelling looks like this. First we fit a saturated model which has a parameter for every data point. There are no degrees of freedom and hence the model has no explanatory power.

```
model1<-glm(n~species*sun*height*perch*time,family=poisson)
```

Prepare yourself for a nasty shock !

```
summary(model1)
```

```
Coefficients:
                                          Estimate Std. Error    z value Pr(>|z|)
(Intercept)                            1.386e+000 5.000e-001      2.773  0.00556 **
species                               -1.000e-014 7.071e-001 -7.73e-015  1.00000
sun                                    9.163e-001 5.916e-001      1.549  0.12143
height                                -1.369e+001 2.847e+002     -0.048  0.96165
perch                                 -2.877e-001 7.638e-001     -0.377  0.70642
timeMid.day                           -1.386e+000 1.118e+000     -1.240  0.21500
timeMorning                           -6.931e-001 8.660e-001     -0.800  0.42349
species.sun                            5.878e-001 8.097e-001      0.726  0.46786
species.height                         1.479e+001 2.847e+002      0.052  0.95857
species.perch                          5.108e-001 1.017e+000      0.503  0.61530
species.timeMid.day                    2.079e+000 1.275e+000      1.631  0.10284
species.timeMorning                    2.303e+000 1.025e+000      2.247  0.02463 *
sun.height                             1.248e+001 2.847e+002      0.044  0.96502
sun.perch                              6.454e-002 8.991e-001      0.072  0.94277
sun.timeMid.day                        2.079e+000 1.183e+000      1.757  0.07884 .
sun.timeMorning                        7.885e-001 9.700e-001      0.813  0.41631
height.perch                           1.259e+001 2.847e+002      0.044  0.96472
height.timeMid.day                     1.386e+000 4.026e+002      0.003  0.99725
height.timeMorning                     6.931e-001 4.026e+002      0.002  0.99863
perch.timeMid.day                      2.877e-001 1.607e+000      0.179  0.85795
perch.timeMorning                      6.931e-001 1.190e+000      0.582  0.56032
species.sun.height                    -1.391e+001 2.847e+002     -0.049  0.96103
species.sun.perch                     -1.099e+000 1.200e+000     -0.916  0.35974
species.sun.timeMid.day               -1.429e+000 1.358e+000     -1.052  0.29283
species.sun.timeMorning               -1.762e+000 1.151e+000     -1.530  0.12598
species.height.perch                  -1.530e+001 2.847e+002     -0.054  0.95714
species.height.timeMid.day            -2.485e+000 4.026e+002     -0.006  0.99508
species.height.timeMorning            -2.223e+000 4.026e+002     -0.006  0.99559
species.perch.timeMid.day             -1.204e+000 1.846e+000     -0.652  0.51431
species.perch.timeMorning             -1.833e+000 1.429e+000     -1.283  0.19965
sun.height.perch                      -1.208e+001 2.847e+002     -0.042  0.96615
sun.height.timeMid.day                -1.792e+000 4.026e+002     -0.004  0.99645
sun.height.timeMorning                -2.776e-001 4.026e+002     -0.001  0.99945
sun.perch.timeMid.day                  4.055e-001 1.700e+000      0.239  0.81147
sun.perch.timeMorning                 -1.598e-001 1.341e+000     -0.119  0.90514
height.perch.timeMid.day              -1.259e+001 4.930e+002     -0.026  0.97963
height.perch.timeMorning              -1.300e+001 4.930e+002     -0.026  0.97897
species.sun.height.perch               1.442e+001 2.847e+002      0.051  0.95960
species.sun.height.timeMid.day         2.989e+000 4.026e+002      0.007  0.99408
species.sun.height.timeMorning         2.040e+000 4.026e+002      0.005  0.99596
species.sun.perch.timeMid.day          1.182e+000 1.981e+000      0.597  0.55083
species.sun.perch.timeMorning          1.417e+000 1.641e+000      0.864  0.38786
species.height.perch.timeMid.day       1.609e+000 5.693e+002      0.003  0.99774
species.height.perch.timeMorning       1.585e+001 4.930e+002      0.032  0.97436
sun.height.perch.timeMid.day           1.183e+001 4.930e+002      0.024  0.98085
sun.height.perch.timeMorning           1.057e+001 4.930e+002      0.021  0.98290
species.sun.height.perch.timeMid.day -1.307e+000 5.693e+002     -0.002  0.99817
species.sun.height.perch.timeMorning -1.330e+001 4.931e+002     -0.027  0.97847

 (Dispersion parameter for poisson family taken to be 1)

Null deviance: 7.3756e+002  on 47  degrees of freedom
Residual deviance: 5.4480e-005  on  0  degrees of freedom
AIC: 259.25
```

There are two key things to note here: (1) the residual deviance and degrees of freedom are both zero because we have intentionally fitted a *saturated model* which has as many parameters (48) as there are data points; and (2) all but 23 of the 48 estimated parameters are nuisance variables. The nuisance variables are of absolutely no scientific interest and are included only to **ensure that all of the marginal totals are correctly constrained.** The only parameters that are of scientific interest are 23 ***interaction terms involving species***.

The only way to model a data set like this successfully is to be fantastically well organised. Terms involving species are deleted stepwise from the current model, starting with the highest order interactions. Non-significant terms are left out. Significant terms are added back in to the model. We need to keep very accurate track of which terms we have deleted and which terms remain to be deleted. Remember,

you can't remove any 3-way interactions until the 4-way interactions containing the relevant factors have been removed.

The 23 parameters to be tested are involved in the following 15 potentially interesting interaction terms that involve species:

```
species.sun
species.height
species.perch
species.timeMid.day
species.timeMorning
species.sun.height
species.sun.perch
species.sun.timeMid.day
species.sun.timeMorning
species.height.perch
species.height.timeMid.day
species.height.timeMorning
species.perch.timeMid.day
species.perch.timeMorning
species.sun.height.perch
species.sun.height.timeMid.day
species.sun.height.timeMorning
species.sun.perch.timeMid.day
species.sun.perch.timeMorning
species.height.perch.timeMid.day
species.height.perch.timeMorning
species.sun.height.perch.timeMid.day
species.sun.height.perch.timeMorning
```

I suggest that you start at the bottom of this list and tick off the interaction terms as you delete them. Once you have tested, say, all of the 4-way interactions involving species and found them to be non significant, then leave them all out (but leave the high order interactions involving the nuisance variables in the model).

Using automatic model simplification with **step** is not much use in a case like this because:

- **step** tends to be too lenient, and to leave non-significant terms in the model
- it is likely to remove interaction terms between nuisance variables and these must stay in the model in order to constrain the marginal totals properly

There is no way round it. We have to do the model simplification the long way, using **update** to delete terms and **anova** to test the significance of deleted terms. I have used cut and paste to make the following table of deletion tests:

```
model2<-update(model1,~.-species:sun:height:perch:time)
anova(model1,model2,test="Chi")
```

```
  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1         0    0.00005
2         2    0.00010 -2 -0.00004   0.99998
```

This means that we can adopt model2 as the base model and systematically delete the various 4-way interactions involving species:

```
model3<-update(model2,~.-species:sun:height:perch)
anova(model2,model3,test="Chi")
```

```
Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1         2     0.0001
2         3     2.7089 -1  -2.7088    0.0998
```

And so on.

```
Resid. Df Resid. Dev   Test                            Df  Deviance      Pr(Chi)
       2 0.00014458   -species:sun:height:perch:time -2 -0.00004    0.9999766
       3   2.708911   -species:sun:height:perch      -1 -2.708766    0.09979817
       5   3.111413   -species:sun:height:time       -2 -0.4025021   0.8177071
       7   7.928383   -species:height:perch:time     -2 -4.816971    0.08995144
       9   8.573263   -species:sun:perch:time        -2 -0.6448801   0.7243793
      10   8.587620   -species:sun:perch             -1 -0.01435679  0.9046259
      11   11.97484   -species:sun:height            -1 -3.387218    0.06570373
      13   13.34891   -species:sun:time              -2 -1.374076    0.5030639
      15   13.37456   -species:perch:time            -2 -0.02564741  0.9872582
      16   13.68243   -species:height:perch          -1 -0.3078676   0.5789916
      18   14.20496   -species:height:time           -2 -0.522531    0.7700764
  *** 20   25.80257   -species:time                  -2 -11.59761   0.003031181
  *** 19   36.27283   -species:height                -1 -22.06787   2.6317e-006
  *** 19   21.89253   -species:sun                   -1 -7.687569   0.005560247
  *** 19   27.33579   -species:perch                 -1 -13.13083   0.000290476
```

There are no significant interactions between the explanatory variables that have a significant effect on the distribution of these two lizard species. All of the factors have highly significant main effects, however. The two lizard species utilise different parts of the bush, occupy perches of different diameters, are active at different times of day, and are found in differing levels of shade. The published analyses that report significant interactions between factors all made the mistake of unintentionally leaving out one or more interactions involving the nuisance variables. This is a very easy thing to do, unless a strict regime of *simplifying downwards from the saturated model* is followed religiously. If you always simplify down from a complex model to a simple model this problem will never arise, and you will save yourself from making embarrassing mistakes of interpretation. With this many comparisons being done, I would always work at $p = 0.01$ for significance, rather than $p = 0.05$.

d) The Danger of Contingency Tables

In observational studies we quantify only a limited number of explanatory variables. It is inevitable that we shall fail to note (or to measure) a number of factors that have an important influence on the ecological behaviour of the system in question. That's life, and given that we make every effort to note the important factors, there is little we can do about it. The problem comes when we ignore factors that have an important influence on ecological behaviour. This difficulty can be particularly acute if we *aggregate data over important explanatory variables*. An example should make this clear.

Suppose we are carrying out a study of induced defences in trees. A preliminary trial has suggested that early feeding on a leaf by aphids may cause chemical changes in the leaf which reduce the probability of that leaf being attacked later in the season by hole-making insects. To this end we mark a large cohort of leaves, then score whether they were infested by aphids early in the season and whether they were holed by

insects later in the year. The work was carried out on two different trees and the results were as follows:

Tree	Aphids	Holed	Intact	Total leaves	Proportion Holed
Tree 1	Without	35	1750	1785	0.0196
	With	23	1146	1169	0.0197
Tree 2	Without	146	1642	1788	0.0817
	With	30	333	363	0.0826

There are 4 variables: the response variable, count, with 8 values (highlighted above), a 2-level factor for late season feeding by caterpillars (holed or intact), a 2-level factor for early season aphid feeding (With or Without aphids) and a 2-level factor for tree (the observations come from two separate trees, imaginatively named Tree1 and Tree2).

```
induced<-read.table("c:\\temp\\induced.txt",header=T)
attach(induced)
names(induced)
```

```
[1] "Tree"        "Aphid"       "Caterpillar"   "Count"
```

We call the model *id* (for induced defences) and fit what is known as a **saturated model**. This is a curious thing, which has as many parameters as there are values of the response variable. The fit of the model is perfect, so there are no residual degrees of freedom and no residual deviance. The reason that we fit a saturated model is that it is always the best place to start modelling complex contingency tables. If we fit the saturated model, then there is no risk that we inadvertently leave out important interactions between the nuisance variables.

```
id<-glm(Count~Tree*Aphid*Caterpillar,family=poisson)
```

The asterisk notation ensures that the saturated model is fitted, because all of the main effects and 2-way interactions are fitted, along with the 3-way interaction *Tree* by *Aphid* by *Caterpillar*. The model fit involves the estimation of 2 x 2 x 2 = 8 parameters, and exactly matches the 8 values of the response variable, *Count*. There is no point looking at the saturated model in any detail, because the reams of information it contains are all superfluous. The first real step in the modelling is to use **update** to remove the 3-way interaction from the saturated model, and then to use **anova** to test whether the 3-way interaction is significant or not.

```
id2<-update(id , ~ . - Tree:Aphid:Caterpillar)
```

The punctuation here is very important (it is comma, tilde, dot, minus) and note the use of colons rather than asterisks to denote interaction terms rather than main effects plus interaction terms. Now we can see whether the 3-way interaction was significant by specifying test=”Chi” like this:

```
anova(id,id2,test="Chi")
```

```
Analysis of Deviance Table
Response: Count

  Resid. Df  Resid. Dev Df Deviance P(>|Chi|)
1         0 -3.975e-013
2         1     0.00079 -1 -0.00079   0.97756
```

This shows clearly that the interaction between caterpillar attack and leaf holing does not differ from tree to tree (p = 0.97756). Note that if this interaction had been significant, then we would have stopped the modelling at this stage. But it wasn't, so we leave it out and continue. What about the main question? Is there an interaction between caterpillar attack and leaf holing? To test this we delete the *Caterpillar:Aphid* interaction from the model, call it *id3*, and assess the results using **anova**:

```
id3<-update(id2 , ~ . - Aphid:Caterpillar)
anova(id2,id3,test="Chi")
```

```
Analysis of Deviance Table
Response: Count

  Resid. Df  Resid. Dev               Test Df    Deviance   Pr(Chi)
1         1 0.000791372
2         2 0.004085322 -Aphid:Caterpillar -1 -0.00329395 0.9542322
```

There is absolutely no hint of an interaction (p = 0.954). The interpretation is clear. This work provides no evidence for induced defences caused by early season caterpillar feeding.

Now we shall do the analysis the wrong way, in order to show the danger of collapsing a contingency table over important explanatory variables. Suppose we went straight for the interaction of interest, *Aphid:Caterpillar*. We might proceed like this:

```
wrong<-glm(Count~Aphid*Caterpillar,family=poisson)
wrong1<-update(wrong,~. - Aphid:Caterpillar)
anova(wrong,wrong1,test="Chi")
```

```
Analysis of Deviance Table
Response: Count

  Resid. Df Resid. Dev               Test Df  Deviance     Pr(Chi)
1         4   550.1917
2         5   556.8511 -Aphid:Caterpillar -1 -6.659372 0.009863566
```

The *Aphid* by *Caterpillar* interaction is highly significant (p < 0.01) providing strong evidence for induced defences. **Wrong !!** By failing to include *Tree* in the model we have omitted an important explanatory variable. As it turns out, and we should really have determined by more thorough preliminary analysis, the trees differ enormously in their average levels of leaf holing:

as.vector(tapply(Count,list(Caterpillar,Tree),sum))[1]/ tapply(Count,Tree,sum) [1]

```
  Tree1
0.01963439
```

as.vector(tapply(Count,list(Caterpillar,Tree),sum))[3]/ tapply(Count,Tree,sum) [2]

```
   Tree2
0.08182241
```

Tree2 has more than 4 times the proportion of its leaves with holes made by caterpillars. If we had been paying more attention when we did the modelling the wrong way, we should have noticed that the model containing only Aphid and Caterpillar had massive **overdispersion**, and this should have alerted us that all was not well. The moral is simple and clear. Always fit a saturated model first, containing all the variables of interest and all the interactions involving the nuisance variables (*Tree* in this case). Only delete from the model those interactions that involve the variables of interest (*Aphid* and *Caterpillar* in this case). Main effects are meaningless in contingency tables, as are the model summaries. **Always test for overdispersion**. It will never be a problem if you follow the advice of simplifying down from a saturated model, because you only ever leave out non-significant terms.

4) Bird Ring Recoveries

Certain kinds of survival analysis involve the periodic recovery of small numbers of dead animals. Bird ringing records are a good example of this kind of data. Out of a reasonably large, known number of ringed birds, a usually small number of dead birds carrying rings is recovered each year. The number of rings recovered each year declines because the pool of ringed birds declines each year as a result of natural mortality. There are two important variables in this case: 1) the probability of a bird dying in a given time period; and 2) the probability that, having died, the ring will be discovered. It is possible that one or both of these parameters varies with the age of the bird (i.e. with the time elapsed since the ring was applied). We can use SPlus with Poisson errors to fit log-linear models to data like this, and to compare estimated survival and ring-recovery rates from different cohorts of animals.

The probability that a bird survives from ringing up to time t_{j-1} is the survivorship s_{j-1} . The number of ringed birds still alive at time t_{j-1} is therefore $N_o s_{j-1}$, where N_o is the number of birds initially ringed and released at time t_o. Thus, the number of birds that die in the present time interval D_j is:

$$D_j = N_0 s_{j-1} d_j$$

where d_j is the probability of an animal that survived to time t_{j-1} dying during the interval $(t_j - t_{j-1})$. Now, out of these D_j dead animals, we expect to recover a small proportion p_j, so the expected number of recoveries, R_j, is:

$$R_j = N_0 s_{j-1} d_j p_j$$

Maximum likelihood estimates of the death and recovery probabilities are possible only if assumptions are made about the way in which the two parameters change with age and with time after ringing (e.g. Seber, 1982). Common assumptions are that the probability of death is constant, once a given age has been reached, and that the probability of discovery of a dead bird is constant over time.

In order to analyse ring return data by log-linear modelling, we must combine the two probabilities of death and recovery into a single parameter z_j. This is the *probability of a death being recorded* in year j for an animal that was alive at the beginning of year j.

$$R_j = N_0 s_{j-1} z_j$$

Now, taking logs, rearranging and then taking antilogs we can write this expression as:

$$R_j = \exp[\ln(N_0) - \lambda_{j-1} t_{j-1} + \ln(z_j)]$$

replacing the survivorship term S_{j-1} by the proportion dying λt. Since we know the number of birds originally marked, we can use $\ln(N_0)$ as an offset. Then, using the log link, we are left with a graph of log of recoveries $\ln R_j$ against time t_j in which the intercept is given by the log of the recovery probability ($\ln(z_j)$) and the slope λ_{j-1} is the survival rate per unit time over the period 0 to t_{j-1}.

S-Plus can now be used to analyse count data on the number of recoveries of dead ringed birds, and to compare models based on different assumptions about survivorship and recovery:

- different cohorts have constant death rates and recovery rates, and these rates are the same in all cohorts (the null model);
- different cohorts have the same death rates but different recovery probabilities;
- different cohorts differ in both their death rates and their recovery rates;
- the death rates may be time dependent; or
- a variety of more complex models.

Suppose that tawny owls were ringed in three different kinds of woodland in the same county; oak, birch and mixed woodland. The numbers marked in the three habitats were 75, 49 and 128 respectively. The numbers of dead, ringed birds recovered in subsequent years were as follows:

```
owlrings<-read.table("c:\\temp\\owlrings.txt",header=T)
attach(owlrings)
names(owlrings)
```

```
[1] "recovered" "year"      "wood"      "marked"
```

The model has the number of rings recovered as the response variable (a count with Poisson errors, with the number of birds marked appearing in the model as an offset

(on the log scale of the linear predictor, because the log link is the default for Poisson models: hence the name, log-linear models).

```
model<-glm(recovered~wood*year+offset(log(marked)),family=poisson)

summary(model)
```

```
Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)    -2.34595    0.58719  -3.995 6.46e-05 ***
woodbirch       0.14340    0.70377   0.204   0.8385
woodoak        -0.08609    0.77883  -0.111   0.9120
year           -0.35839    0.18323  -1.956   0.0505 .
woodbirch:year -0.12476    0.22886  -0.545   0.5856
woodoak:year   -0.02906    0.24617  -0.118   0.9060

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 42.839  on 20  degrees of freedom
Residual deviance: 15.670  on 15  degrees of freedom
AIC: 69.859
```

There is clearly no justification for keeping different slopes for the different woodlands ($p > 0.5$), so we remove the interaction term, then compare the complex and simpler models using anova. Note the lack of overdispersion.

```
model2<-glm(recovered~wood+year+offset(log(marked)),family=poisson)

anova(model,model2,test="Chi")
```

```
Analysis of Deviance Table

Model 1: recovered ~ wood * year + offset(log(marked))
Model 2: recovered ~ wood + year + offset(log(marked))

  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1        15    15.6703
2        17    16.0353 -2  -0.3650    0.8332
```

This confirms that there is no justification for retaining different slopes in different woodlands. What about differences in the intercepts of different woodlands? We simplify model2, like this:

```
model3<-glm(recovered~year+offset(log(marked)),family=poisson)
anova(model2,model3,test="Chi")
```

```
Analysis of Deviance Table

Model 1: recovered ~ wood + year + offset(log(marked))
Model 2: recovered ~ year + offset(log(marked))

  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
```

```
1        17    16.0353
2        19    16.2502 -2  -0.2148    0.8982
```

Not even close (p = 0.898). What about the slope?

```
model4<-glm(recovered~offset(log(marked)),family=poisson)
anova(model3,model4,test="Chi")
```

```
Analysis of Deviance Table

Model 1: recovered ~ year + offset(log(marked))
Model 2: recovered ~ offset(log(marked))
  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1        19     16.250
2        20     42.839 -1  -26.589 2.517e-07
```

No doubt about the significance of the slope. So model3 looks to be minimal adequate:

```
summary(model3)
```

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.30547    0.27314  -8.441  < 2e-16 ***
year        -0.42522    0.09114  -4.665 3.08e-06 ***

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 42.839  on 20  degrees of freedom
Residual deviance: 16.250  on 19  degrees of freedom
AIC: 62.439
```

The final model is not overdispersed (16.25 on 19 d.f.). The interpretation is that the log of the recovery rate (the intercept) is –2.30547 and that the log of the survival rate (the slope) is – 0.42522. Taking antilogs we get

```
exp(-2.30547)
```

```
[1] 0.09971193
```

```
exp(-0.42522)
```

```
[1] 0.653626
```

We conclude that the annual survivorship of the owls is about 65% and that survival does not differ significantly from one woodland to another. The annual probability of recovery of a ring from a dead ringed bird is estimated to be about 10%.

Error checking by **plot(model3)** shows a few problems: the deviance declines markedly with the fitted values, and there is substantial non-normality in the error distribution, particularly for the largest negative residuals. But this will not affect the parameter estimates greatly, and parameter estimation is our main purpose here.

Reanalysis of Schoener's lizards as proportion data:
getting rid of the nuisance variables

The analysis of count data in complex contingency tables is difficult, tedious and very easy to get wrong. If the response can be reformulated as a proportion, then the analysis is much more straightforward, because there is no longer any need to include the vast numbers of nuisance parameters necessary to constrain the marginal totals.

The computational part of the exercise here is data compression. We need to reduce the data frame called lizards so that it has the proportion of all lizards that are *Anolis grahamii* as the response, with the same set of explanatory variables (but each with half as many rows as before). It is essential to ensure that all of the cases are in exactly the same sequence for both of the lizard species. To do this, we create a sorted version of the data frame:

```
sorted.lizards<-lizards[order(lizards[,2],lizards[,3],lizards[,4],lizards[,5],lizards[,6]),1:6]
```

So now we can save the ordered *grahamii* and *opalinus* vectors as *ng* and *no* and bind them together to use as the response vector in the binomial analysis

```
ng<-sorted.lizards[,1][sorted.lizards[,6]=="grahamii"]
no<-sorted.lizards[,1][sorted.lizards[,6]=="opalinus"]
y<-cbind(ng,no)
```

The 4 shortened explanatory variables (the factors s, h, p, and t) are now produced:

```
s<-sorted.lizards[,2][sorted.lizards[,6]=="grahamii"]
h<-sorted.lizards[,3][sorted.lizards[,6]=="grahamii"]
p<-sorted.lizards[,4][sorted.lizards[,6]=="grahamii"]
t<-sorted.lizards[,5][sorted.lizards[,6]=="grahamii"]
```

You should check that these have been automatically declared to be factors (hint: use **is.factor**). Now we are ready to fit the saturated model:

```
model<-glm(y ~ s*h*p*t , binomial)
```

Let's see how good **step** is at simplifying this saturated model:

```
step(model)
```

```
Start:  AIC= 102.82
 y ~ s + h + p + t + s:h + s:p + s:t + h:p + h:t + p:t + s:h:p +
    s:h:t + s:p:t + h:p:t + s:h:p:t
```

(out goes the 4-way interaction s:h:p:t)

```
          Df  Deviance    AIC
- s:h:p:t  1 0.0002621 100.82
<none>       0.0001585 102.82

Step:  AIC= 100.82
```

```
 y ~ s + h + p + t + s:h + s:p + s:t + h:p + h:t + p:t + s:h:p +
    s:h:t + s:p:t + h:p:t
```

(out goes the 3-way interaction s:h:t)

```
        Df  Deviance     AIC
- s:h:t  2     0.442  97.266
- s:p:t  2     0.810  97.635
- h:p:t  2     3.222 100.046
<none>     0.0002621 100.824
- s:h:p  1     2.709 101.533

Step:  AIC= 97.27
 y ~ s + h + p + t + s:h + s:p + s:t + h:p + h:t + p:t + s:h:p +
    s:p:t + h:p:t
```

(out goes the 3-way interaction s:p:t)

```
        Df Deviance    AIC
- s:p:t  2    1.071 93.896
<none>        0.442 97.266
- h:p:t  2    4.648 97.472
- s:h:p  1    3.111 97.936

Step:  AIC= 93.9
 y ~ s + h + p + t + s:h + s:p + s:t + h:p + h:t + p:t + s:h:p +
    h:p:t

        Df Deviance    AIC
- s:t    2    3.340 92.165
<none>        1.071 93.896
- s:h:p  1    3.302 94.126
- h:p:t  2    5.791 94.615

Step:  AIC= 92.16
 y ~ s + h + p + t + s:h + s:p + h:p + h:t + p:t + s:h:p + h:p:t
```

(AIC has *increased,* so this deletion was not accepted)

```
        Df Deviance    AIC
<none>        3.340 92.165
- s:h:p  1    5.827 92.651
- h:p:t  2    8.542 93.366

Degrees of Freedom: 22 Total (i.e. Null);  7 Residual
Null Deviance:      70.1
Residual Deviance: 3.34         AIC: 92.16
Start:  AIC= 48.0002
```

(it wants to keep the 3-way interaction h:p:t)

```
model1<-step(model)
summary(model1)
```

```
Coefficients:
                      Estimate Std. Error z value Pr(>|z|)
(Intercept)            -0.8297     0.5120  -1.620   0.1051
sSun                    0.4791     0.4744   1.010   0.3126
hLow                  -11.9069    72.7886  -0.164   0.8701
pNarrow                 0.1751     0.7481   0.234   0.8149
tMid.day               -0.9101     0.4272  -2.130   0.0331 *
tMorning               -0.9314     0.4637  -2.009   0.0445 *
sSun:hLow              10.7912    72.7884   0.148   0.8821
sSun:pNarrow            0.2416     0.6987   0.346   0.7295
hLow:pNarrow           12.1239    72.8016   0.167   0.8677
hLow:tMid.day          -0.2446     0.9278  -0.264   0.7920
hLow:tMorning           0.5732     0.9260   0.619   0.5359
pNarrow:tMid.day        0.2140     0.6486   0.330   0.7415
pNarrow:tMorning        0.7106     0.6980   1.018   0.3087
sSun:hLow:pNarrow     -10.9665    72.8028  -0.151   0.8803
hLow:pNarrow:tMid.day  -0.6021     1.3489  -0.446   0.6553
hLow:pNarrow:tMorning  -3.2054     1.6106  -1.990   0.0466 *
```

As often happens, **step** has left a rather complicated model with two 3-way interactions and six 2-way interactions in it. Let's see how we get on by hand in simplifying the reduced model (model1) that **step** has bequeathed us. We remove the terms in sequence and use **anova** to compare the simpler model with its more complex predecessor:

```
model2<-update(model1,~.-h:p:t)
anova(model1,model2,test="Chi")
```

```
Analysis of Deviance Table

  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1         7     3.3405
2         9     8.5422 -2  -5.2017    0.0742
```

This simplification caused a non-significant increase in deviance (p = 0.07) so we make model2 the current model, and remove the next 3-way interaction, s:h:p (see below). We keep removing terms until a significant term is discovered, as follows:

```
                          Resid. Df Resid. Dev Df Deviance P(>|Chi|)

model2 <-update(model1,~.-h:p:t)  9     8.5422 -2  -5.2017    0.0742
model3 <-update(model2,~.-s:h:p) 10    10.9032 -1  -2.3610    0.1244
model4 <-update(model3,~.-p:t)   12    10.9090 -2  -0.0058    0.9971
model5 <-update(model4,~.-h:t)   14    11.7667 -2  -0.8577    0.6513
model6 <-update(model5,~.-h:p)   15    11.9789 -1  -0.2122    0.6450
model7 <-update(model6,~.-s:p)   16    11.9843 -1  -0.0054    0.9414
model8 <-update(model7,~.-s:h)   17    14.2046 -1  -2.2203    0.1362
model9 <-update(model8,~.-t)     19     25.802 -2  -11.597     0.003 ***
model10<-update(model8,~.-p)     18    27.3346 -1 -13.1300    0.0003 ****
model11<-update(model8,~.-h)     18     36.271 -1  -22.066 2.634e-06 ****
model12<-update(model8,~.-s)     18    21.8917 -1  -7.6871    0.0056 ***
```

Note that because there was a significant increase in deviance when time was deleted from model8, we use model 8 not model9 in assessing the significance of perch diameter (and of the other two main effects as well). The main effect probabilities are identical to those obtained by contingency table analysis with Poisson errors (above). We need to look at the model summary for model8 to see whether all of the factor levels need to be retained:

```
summary(model8)
```

```
Call:
glm(formula = y ~ s + h + p + t, family = binomial)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.48718  -0.62644  -0.04488   0.37800   1.66015

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.2079     0.3534  -3.418 0.000631 ***
sSun          0.8473     0.3222   2.629 0.008554 **
hLow         -1.1300     0.2570  -4.397 1.10e-05 ***
pNarrow       0.7626     0.2112   3.610 0.000306 ***
tMid.day     -0.9639     0.2815  -3.424 0.000618 ***
tMorning     -0.7368     0.2989  -2.465 0.013712 *

 (Dispersion parameter for binomial family taken to be 1)

    Null deviance: 70.102  on 22  degrees of freedom
Residual deviance: 14.205  on 17  degrees of freedom
AIC: 83.029
```

Mid.day and Morning are not significantly different from one another (their parameters are -0.96 and -0.74 with a standard error of a difference of 0.299), so we lump them together in a new factor called t3:

```
t3<-t
levels(t3)[c(2,3)]<-"other"
levels(t3)
```

```
[1] "Afternoon" "other"
```

```
model9<-glm(y~s+h+p+t3,binomial)
anova(model8,model9,test="Chi")
```

```
Analysis of Deviance Table

Response: y

          Terms Resid. Df Resid. Dev    Test Df   Deviance   Pr(Chi)
1  s + h + p + t       17   14.20457
2 s + h + p + t3       18   15.02320 1 vs. 2 -1 -0.8186255 0.3655824
```

This simplification was justified (p = 0.37), so we accept model9.

```
summary(model9)
```

```
Call:
glm(formula = y ~ s + h + p + t3, family = binomial)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.1595     0.3482  -3.330 0.000870 ***
sSun          0.7871     0.3158   2.493 0.012682 *
hLow         -1.1188     0.2565  -4.362 1.29e-05 ***
pNarrow       0.7485     0.2104   3.557 0.000375 ***
t3other      -0.8717     0.2611  -3.339 0.000842 ***
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 70.102  on 22  degrees of freedom
Residual deviance: 15.023  on 18  degrees of freedom
AIC: 81.847
```

All the parameters are significant, so this is the minimal adequate model. There are just 5 parameters, and the model contains no nuisance variables (compare this with the massive contingency table model on p. 322). The ecological interpretation is straightforward: the two lizard species differ significantly in their niches on all the niche-axes that were measured. However, there were no significant interactions (nothing subtle was happening like swapping perch sizes at different times of day).

STATISTICS: AN INTRODUCTION USING R

By M.J. Crawley

Exercises

12. SURVIVAL ANALYSIS

A great many studies in statistics deal with deaths or with failures of components: the numbers of deaths, the timing of death, and the risks of death to which different classes of individuals are exposed. The analysis of survival data is a major focus of the statistics business (see Kalbfleisch & Prentice 1980, Miller 1981, Fleming & Harrington 1991), and S-Plus supports a wide range of tools for the analysis of survival data. The main theme of this chapter is the analysis of data that take the form of measurements of the *time to death*, or the *time to failure* of a component. Up to now, we have dealt with mortality data by considering the proportion of individuals that were dead *at a given time*. In this chapter each individual is followed until it dies, then the time of death is recorded (this will be the response variable). Individuals that survive to the end of the experiment will die at an unknown time in the future; they are said to be *censored* (see below).

A Monte Carlo experiment

With data on time-to-death, the most important decision to be made concerns the error distribution. The key point to understand is that the variance in age at death is almost certain to increase with the mean, and hence standard models (assuming constant variance and normal errors) will be inappropriate. You can see this at once with a simple Monte Carlo experiment. Suppose that the per-week probability of failure of a component is 0.1 from one factory but 0.2 from another. We can simulate the fate of an individual component in a given week by generating a uniformly distributed random number between 0 and 1. If the value of the random number is less than or equal to 0.1 (or 0.2 for the second factory) the component fails during that week and its lifetime can be calculated. If the random number is larger than 0.1 the component survives to the next week. The lifetime of the component is simply the number of the week in which it finally failed. Thus, a component that failed in the first week has an age at failure of 1 (this convention means that there are no zeros in the data frame).

The simulation is very simple. We create a vector of random numbers, *rnos*, that is long enough to be almost certain to contain a value that is less than our failure probabilities of 0.1 and 0.2. Remember that the mean life expectancy is the reciprocal of the failure rate, so our mean lifetimes will be 1/0.1 = 10 and 1/0.2 = 5 weeks respectively. A length of 100 should be more than sufficient.

```
rnos<-runif(100)
```

The trick is to find the week number in which the component failed; this is the lowest subscript for which *rnos* <= 0.1 for factory 1. We can do this very efficiently using the **which** function: **which** returns *a vector of subscripts* for which the specified logical condition is true. So for factory 1 we would write

```
which(rnos<= 0.1)
```

```
 [1]  5  8  9 19 29 33 48 51 54 63 68 74 80 83 94 95
```

This means that for my first set of 100 random numbers, 16 of them were less than or equal to 0.1. The important point is that the *first* such number occurred in week 5. So the simulated value of the age of death of this first component is 5 and is obtained from the vector of failure ages using the subscript [1]

```
which(rnos<= 0.1)[1]
```

```
[1] 5
```

All we need to do to simulate the life spans of a sample of 20 components, death1, is to repeat the above procedure 20 times

```
death1<-numeric(20)

for  (i in 1:20)  {
rnos<-runif(100)
death1[i]<- which(rnos<= 0.1)[1]
 }

death1
```

```
 [1]  5  8 12 23 11  3  8  3 12 13  1  5  9  1  7  9 11  1  2  8
```

The 4th component survived for a massive 23 weeks but the 11th component failed during its first week. The simulation has roughly the right average weekly failure rate:

```
1/mean(death1)
```

```
[1] 0.1315789
```

which is as close to 0.1 as we could reasonably expect from a sample of only 20 components. Now we do the same for the second factory with its failure rate of 0.2:

```
death2<-numeric(20)

for  (i in 1:20)  {
```

```
rnos<-runif(100)
death2[i]<- which(rnos<= 0.2)[1]
 }
```

The sample mean is again quite reasonable (if a little on the low side):

```
1/mean(death2)
```

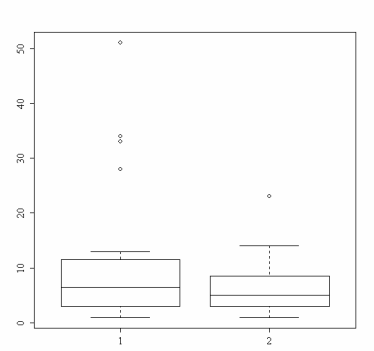
```
[1] 0.1538462
```

We now have the simulated raw data to carry out a comparison in age at death between factories 1 and 2. We combine the two vectors into one, and generate a vector to represent the factory identities:

```
death<-c(death1,death2)
factory<-factor(c(rep(1,20),rep(2,20)))
```

We get a visual assessment of the data using plot

```
plot(factory,death)
```



The median age at death for factory 1 is somewhat greater, but the variance in age a death is much higher than from factory 2. For data like this we expect the variance to be proportional to the square of the mean, so an appropriate error structure is the gamma (as explained below). We model the data very simply as a one-way analysis of deviance with a **glm** of family = Gamma (note the upper case G)

```
model1<-glm(death~factory,Gamma)
summary(model1)
```

```
Coefficients:
            Estimate Std. Error t value  Pr(>|t|)
(Intercept)  0.08474    0.01861   4.554 5.29e-005 ***
factory      0.06911    0.03884   1.779    0.0832 .

(Dispersion parameter for Gamma family taken to be 0.983125)
```

```
    Null deviance: 37.175  on 39  degrees of freedom
Residual deviance: 33.670  on 38  degrees of freedom
```

We conclude that the factories are not significantly different in mean age at failure of these components ($p = 0.0832$). So, even with a 2-fold difference in the true failure rate, we are unable to detect a significant difference in mean age at death with samples of size $n = 20$. But the **glm** with gamma errors comes closer to detecting the difference than did a conventional analysis with normal errors and constant variance (see if you can work out how to demonstrate this: for my data, $p = 0.113$ with the inappropriate anova). The moral is that *for data like this on age at death you are going to need really large sample sizes in order to find significant differences*.

rm(death)

Background

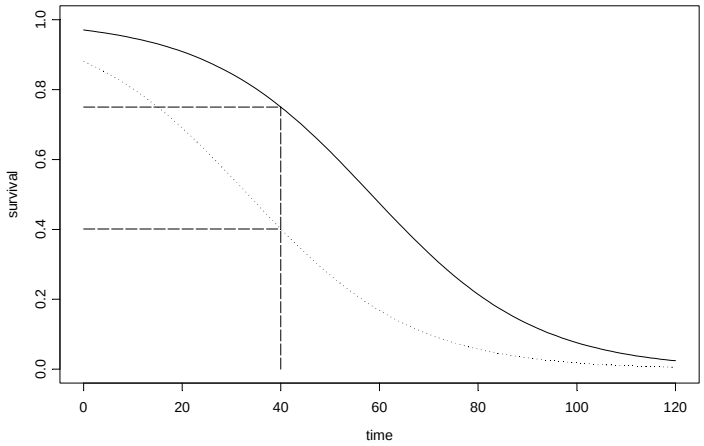
We are unlikely to know much about the error distribution in advance of the study, except that it will certainly not be normal! In S-Plus we are offered several choices for the analysis of survival data:

- gamma
- exponential
- piece-wise exponential
- extreme value
- log-logistic
- log normal
- Weibull

and, in practice, it is often difficult to choose between them. In general, the best solution is to try several distributions and to pick the error structure that produces the minimum error deviance. Parametric survival models are used in circumstances where prediction is the object of the exercise (e.g. in analyses where extreme conditions are used to generate accelerated failure times). Alternatively, we could use nonparametric methods, the most important of which are Kaplan-Meier and Cox proportional hazards, which are excellent for comparing the effects of different treatments on survival, but they do not predict beyond the last observation and hence can not be used for extrapolation.

Some theoretical demography

Since everything dies eventually, it is often not possible to analyse the results of survival experiments in terms of the proportion that were killed (as we did in Practical 10); in due course, they *all* die. Look at the following figure:

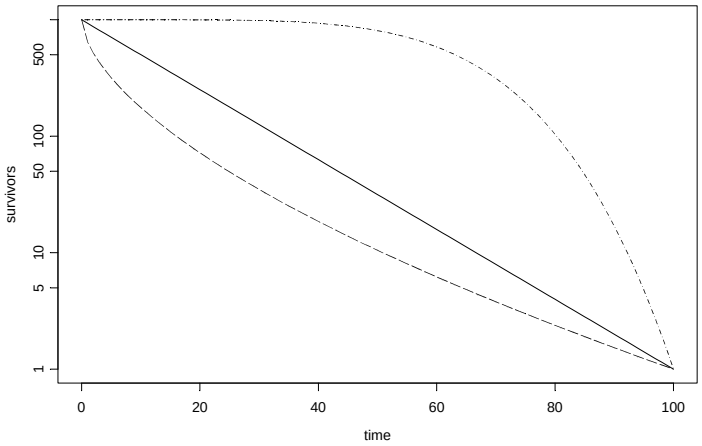


It is clear that the two treatments caused different patterns of mortality, but both start out with 100% survival and both end up with zero. We could pick some arbitrary point in the middle of the distribution at which to compare the percentage survival (say at time = 40), but this may be difficult in practice, because one or both of the treatments might have few observations at the same location. Also, the choice of when to measure the difference is entirely subjective and hence open to bias. It is much better to use S-Plus's powerful facilities for the analysis of survival data than it is to pick an *arbitrary* time at which to compare two proportions.

Demographers, actuaries and ecologists use three interchangeable concepts when dealing with data on the timing of death:

- survivorship
- age at death
- instantaneous risk of death

There are 3 broad patterns of survivorship. Type I: most of the mortality occurs late in life (e.g. humans); Type II: mortality occurs at a roughly constant rate throughout life; Type III: most of the mortality occurs early in life (e.g. salmonid fishes). On a log scale, the numbers surviving from an initial cohort of 1000, say, would look like this:



1) The survivor function

The survivorship curve plots the natural log of the proportion of a cohort of individuals that started out at time 0 that is still alive at time t. For the so-called Type II survivorship curve, there is a linear decline in log numbers with time. This means that a constant proportion of the individuals alive at the beginning of a time interval will die during that time interval (i.e. the proportion dying is density independent and constant for all ages). When the death rate is highest for the younger age classes we get a steeply descending, Type III survivorship curve. When it is the oldest animals that have the highest risk of death, we obtain the Type I curve (characteristic of human populations in affluent societies where there is low infant mortality).

2) The density function

The density function describes the fraction of all deaths from our initial cohort that are likely to occur in a given instant of time. For the Type II curve this is a negative exponential. Because the fraction of individuals dying is constant with age, the number dying declines exponentially as the number of survivors (the number of individuals at risk of death) declines exponentially with the passage of time. The density function declines more steeply than exponentially for Type III survivorship curves. In the case of Type I curves, however, the density function has a maximum at the time when the product of the risk of death and the number of survivors is greatest (see below).

3) The hazard function

The hazard is the instantaneous risk of death; i.e. the derivative of the survivorship curve. It is the instantaneous rate of change in the log of the number of survivors per unit time (it is the slope of the survivorship curves). Thus, for the Type II survivorship the hazard function is a horizontal line, because the risk of death is constant with age. Although this sounds highly unrealistic, it is a remarkably robust assumption in many applications. It also has the substantial advantage of parsimony. In some cases, however, it is clear that the risk of death changes substantially with the age of the individuals, and we need to be

able to take this into account in carrying out our statistical analysis. In the case of Type III survivorship, the risk of death declines with age, while for Type I survivorship (as in humans) the risk of death increases with age.

The Exponential Distribution

This is a 1-parameter distribution in which the hazard function is independent of age (i.e. it describes a Type II survivorship curve). The exponential is a special case of the gamma distribution in which the shape parameter α is equal to 1 .

Density function

The density function is the probability of dying in the small interval of time between *t* and *t+dt*; a plot of the number dying in the interval around time *t* as a function of *t* (i.e. the proportion of the original cohort dying at a given age) declines exponentially:

$$f(t) = \frac{e^{-\frac{t}{\mu}}}{\mu}$$

where both μ and $t > 0$. Note that the density function has an intercept of $1/\mu$ (remember that e^0 is 1). The probability of dying declines exponentially with time and a fraction $1/\mu$ dies during the first time interval (and, indeed, during every subsequent time interval).

Survivor function

This shows the proportion of individuals from the initial cohort still alive at time t:

$$S(t) = e^{-\frac{t}{\mu}}$$

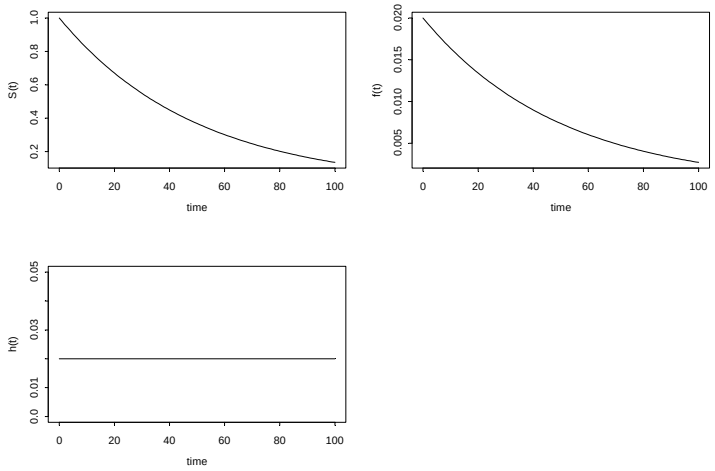
The survivor function has an intercept of 1 (i.e. all the cohort is alive at time 0), and shows the probability of surviving at least as long as *t*.

Hazard function

This is the statisticians equivalent of the ecologist's *instantaneous death rate*. It is defined as the ratio between the density function and the survivor function, and is the conditional density function at time *t*, given survival up to time *t*. In the case of Type II curves this has an extremely simple form:

$$h(t) = \frac{f(t)}{S(t)} = \frac{e^{-t/\mu}}{\mu e^{-t/\mu}} = \frac{1}{\mu}$$

because the exponential terms cancel out. Thus, with the exponential distribution the *hazard is the reciprocal of the mean time to death*, and vice versa. For example, if the mean time to death is 3.8 weeks, then the hazard is 0.2632; if the hazard were to increase to 0.32, then the mean time of death would decline to 3.125 weeks.

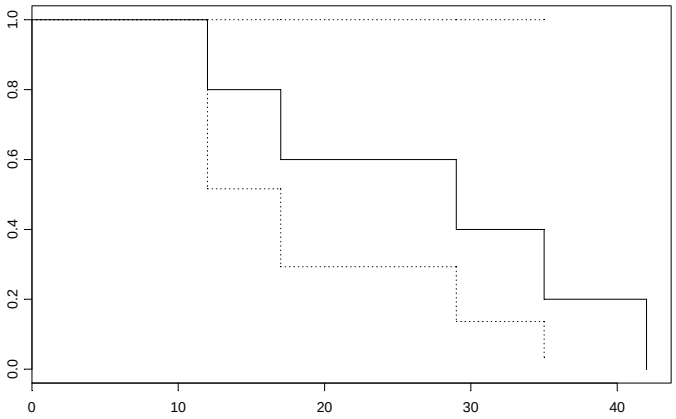


These are the **survivor**, **density** and **hazard** functions of the **exponential distribution**.

Of course, the death rate may not be a linear function of age. For example, the death rate may be high for very young as well as for very old animals, in which case the survivorship curve is like an S-shape on its side.

Kaplan-Meier survival distributions

This is a discrete stepped survivorship curve that adds information as each death occurs. Suppose the times at death were 12, 17, 29, 35 and 42 weeks after the beginning of a trial. Survivorship is 1 at the outset, and stays at 1 until time 12, when it steps down to 4/5 = 0.8. It stays level until time 17 when it drops to 0.8 x 3/4 = 0.6. Then there is a long level period until time 29, when survivorship drops to 0.6 x 2/3 = 0.4, then drops at time 35 to 0.4 x 1/2 = 0.2 then to zero at time 42.



In general, therefore, we have two groups at any one time: the number of deaths $d(t_i)$ and the number at risk $r(t_i)$ (i.e. those that have not died yet: the survivors). The Kaplan-Meier survivor function is

$$\hat{S}_{KM} = \prod_{t_i<t} \frac{r(t_i) - d(t_i)}{r(t_i)}$$

which as we have seen, produces a step at every time at which one or more deaths occurs. The censored individuals that survive beyond the end of the study are shown by a + on the plot or after their age in the data frame (thus 65 means died at time 65, but 65+ means still alive when last seen at age 65).

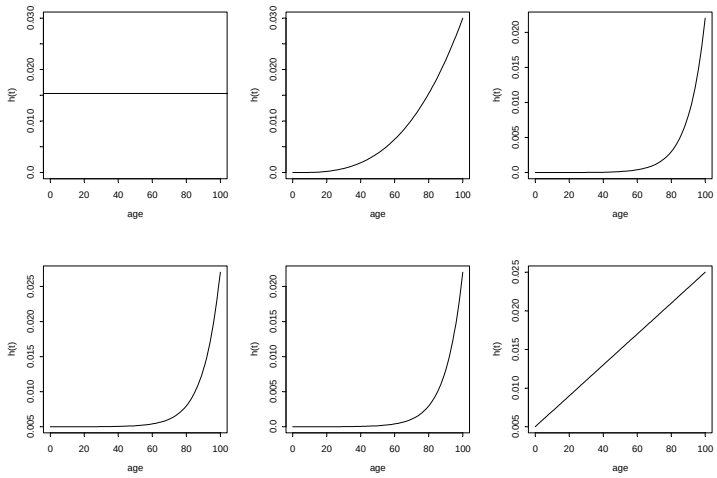
Age-specific hazard models

In many circumstances, the risk of death increases with age. There are many models to choose from:

Distribution	Hazard
Exponential	constant $= \frac{1}{\mu}$
Weibull	$\alpha\lambda(\lambda t)^{\alpha-1}$
Gompertz	be^{ct}
Makeham	$a + be^{ct}$
Extreme value	$\frac{1}{\sigma} e^{(t-\eta)/\sigma}$
Rayleigh	$a + bt$

The Rayleigh is obviously the simplest model in which hazard increases with time, but the Makeham is widely regarded as the best description of hazard for human subjects. Post infancy, there is a constant hazard (a) which is due to age independent accidents,

murder, suicides, etc., with an exponentially increasing hazard in later life. The Gompertz assumption was that "the average exhaustion of a man's power to avoid death is such that at the end of equal infinitely small intervals of time he has lost equal portions of his remaining power to oppose destruction which he had at the commencement of these intervals". Note that the Gompertz differs from the Makeham only by the omission of the extra background hazard (a), and this becomes negligible in old age.



These plots show how hazard changes with age for the following distributions: from top left to bottom right: **exponential, Weibull, Gompertz, Makeham, Extreme value** and **Rayleigh**.

Survival analysis in R

There are three cases that concern us here:

- constant hazard and no censoring
- constant hazard with censoring
- age-specific hazard, with or without censoring

The first case is dealt with in R by specifying a **glm** with exponential errors. This involves using gamma errors with the scale factor fixed at 1 .

The second case involves the use of a **glm** with Poisson errors and a log link, where *the censoring indicator is the response variable*, and log(time of death) is an offset (see below).

The third case is the one that concerns us mainly in this Practical. We can choose to use **parametric** models, based round the **Weibull** distribution, or **non parametric** techniques, based round **Cox proportional hazard** model.

Cox proportional hazards model

This is the most widely used regression model for survival data. It assumes that the hazard is of this form

$$\lambda(t; Z_i) = \lambda_0(t) r_i(t)$$

where $Z_i(t)$ is the set of explanatory variables for individual *i* at time *t*. The *risk score* for subject *i* is

$$r_i(t) = e^{\beta Z_i(t)}$$

in which β is a vector of parameters from the linear predictor and $\lambda_0(t)$ is an *unspecified baseline hazard function* that will cancel out in due course. The antilog guarantees that λ is positive for any regression model $\beta Z_i(t)$. If a death occurs at time t*, then conditional on this death occurring, the likelihood that it is individual *i* that dies rather than any other individual at risk, is

$$L_i(\beta) = \frac{\lambda_0(t^*) r_i(t^*)}{\sum_j Y_j(t^*) \lambda_0(t^*) r_j(t^*)} = \frac{r_i(t^*)}{\sum_j Y_j(t^*) r_j(t^*)}$$

The product of these terms over all times of death $L(\beta) = \prod L_i(\beta)$ was christened a partial likelihood by Cox (1972). This is clever, because maximising $\log(L(\beta))$ allows an estimate of β without knowing anything about the baseline hazard function ($\lambda_0(t)$ is a nuisance variable in this context). The proportional hazards model is nonparametric in the sense that it depends only on the **ranks** of the survival times.

An example of survival analysis without censoring

To see how the exponential distribution is used in modelling we take an example from plant ecology, in which individual seedlings were followed from germination until death. We have the times to death measured in weeks for two cohorts, each of 30 seedlings. The plants were germinated at two times (cohorts), in early September (treatment 1) and mid October (treatment 2). We also have data on the size of the gap into which each seed was sown (a covariate *x*). The questions are these:

- is an exponential distribution suitable to describe these data?
- was survivorship different between the 2 planting dates?
- did gap size affect the time to death of a given seedling?

```
seedlings<-read.table("c:\\temp\\seedlings.txt",header=T)
attach(seedlings)
names(seedlings)
```

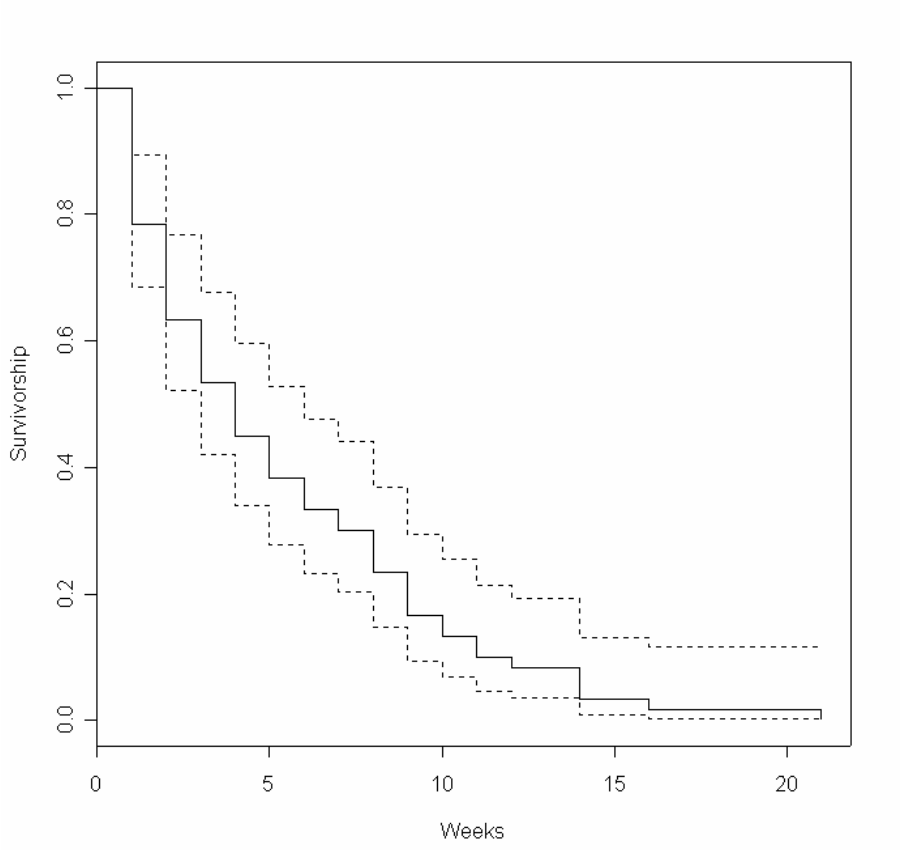
```
[1] "cohort"  "death"   "gapsize"
```

We need to open the library of survival analysis functions:

```
library(survival)
```

There are several important functions for plotting and analysing survival data. The function **Surv** (note the capital S) takes 2 vectors. The first contains the time (or age) at which the individual was last seen, and the second indicates the status of that individual (i.e. whether it was dead or alive when it was last seen: 1 = dead, 0 = alive). Individuals are said to be censored if they were alive at the time they were last seen. All the seedlings died in this example, so status = 1 for all the cases. The function **survfit** calculates the survivorship curve on the basis of the age at death and censoring information in **Surv**. Use of **plot** with **survift** as its argument produces a graph of the survivorship curve.

```
status<-1*(death>0)

plot(survfit(Surv(death,status)),ylab="Survivorship",xlab="Weeks")
```



This shows the survivorship of seedlings over the 20 weeks of the study period until the last of the seedlings died in week 21. The dotted lines show the confidence limits and are the default when only 1 survivorship curve is plotted. No axis labels are plotted unless we provide them (as here).

Statistical modelling is extremely straightforward, but somewhat limited, because interaction effects and continuous explanatory variables are not allowed with **survfit** (but see below). We begin with a simple model for the effect of cohort on its own:

```
model1<-survfit(Surv(death,status)~cohort)
```

Just typing the name of the model object produces a useful summary of the two survivorship curves, their means, standard errors and 95% confidence intervals for the age at death:

```
model1
```

```
Call: survfit(formula = Surv(death, status) ~ cohort)

                   n events mean se(mean) median 0.95LCL 0.95UCL
  cohort=October  30     30 5.83    0.903      5       3       9
cohort=September 30     30 4.90    0.719      4       2       7
```

Survival in the two cohorts is clearly not significantly different (just look at the overlap in the confidence intervals for median age at death). Using the **summary** function produces fully documented survival schedules for each of the two cohorts:

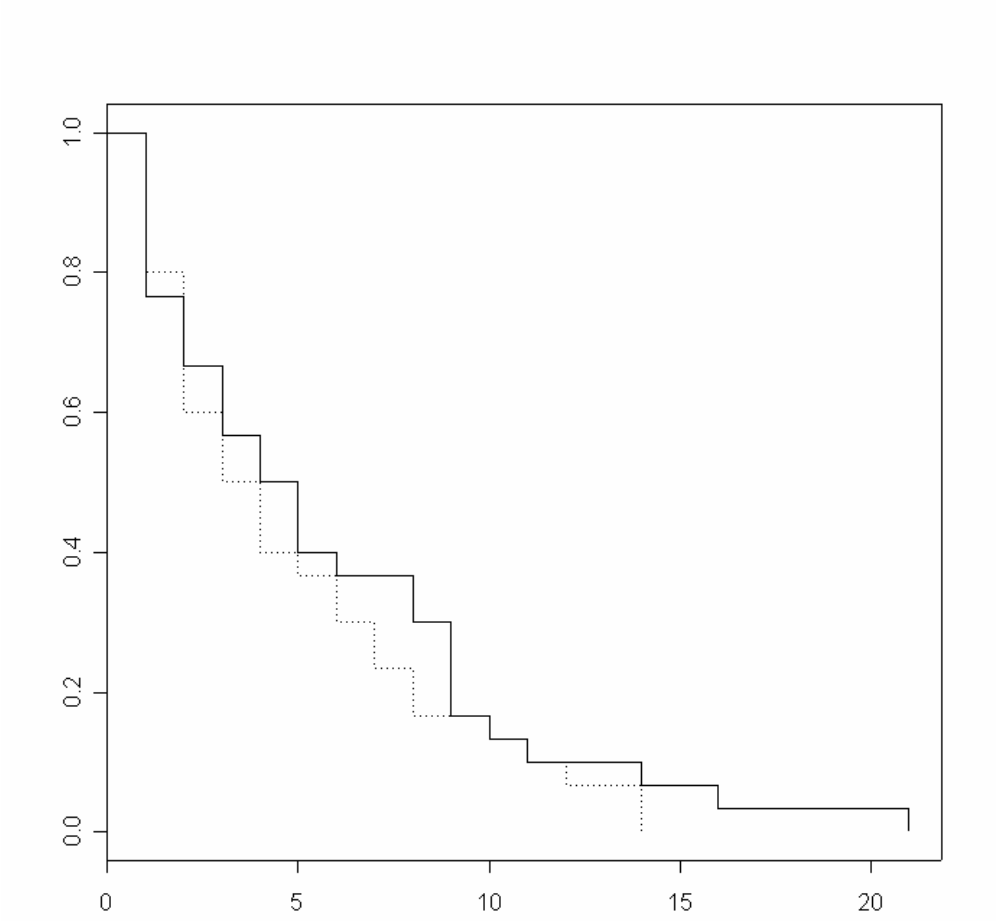
```
summary(model1)
```

```
          cohort=October
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    1     30       7   0.7667  0.0772      0.62932        0.934
    2     23       3   0.6667  0.0861      0.51763        0.859
    3     20       3   0.5667  0.0905      0.41441        0.775
    4     17       2   0.5000  0.0913      0.34959        0.715
    5     15       3   0.4000  0.0894      0.25806        0.620
    6     12       1   0.3667  0.0880      0.22910        0.587
    8     11       2   0.3000  0.0837      0.17367        0.518
    9      9       4   0.1667  0.0680      0.07488        0.371
   10      5       1   0.1333  0.0621      0.05355        0.332
   11      4       1   0.1000  0.0548      0.03418        0.293
   14      3       1   0.0667  0.0455      0.01748        0.254
   16      2       1   0.0333  0.0328      0.00485        0.229
   21      1       1   0.0000      NA           NA           NA

                  cohort=September
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    1     30       6   0.8000  0.0730       0.6689        0.957
    2     24       6   0.6000  0.0894       0.4480        0.804
    3     18       3   0.5000  0.0913       0.3496        0.715
    4     15       3   0.4000  0.0894       0.2581        0.620
```

5	12	1	0.3667	0.0880	0.2291	0.587
6	11	2	0.3000	0.0837	0.1737	0.518
7	9	2	0.2333	0.0772	0.1220	0.446
8	7	2	0.1667	0.0680	0.0749	0.371
10	5	1	0.1333	0.0621	0.0535	0.332
11	4	1	0.1000	0.0548	0.0342	0.293
12	3	1	0.0667	0.0455	0.0175	0.254
14	2	2	0.0000	NA	NA	NA

Using the **plot** function produces a set of survivorship curves. Note the use of the vector of different line types lty=c(1,3) for plotting

```
plot(model1,lty=c(1,3))
```



This produces a plot of the two survivorship curves using line types 1 and 3 for the October and September cohorts respectively (the factor levels are in alphabetic order, as usual). Note that the axes are unlabelled unless you specify **xlab** and **ylab**.

To investigate the effects of a continuous explanatory variable like gap size we use Cox proportional hazards to give the survivorship curve at the average gap size:

```
model2<-survfit(coxph(Surv(death,status)~gapsize))
```

Typing the model name alone gives this:

```
model2
```

Call: survfit.coxph(object = coxph(Surv(death, status) ~ gapsize))

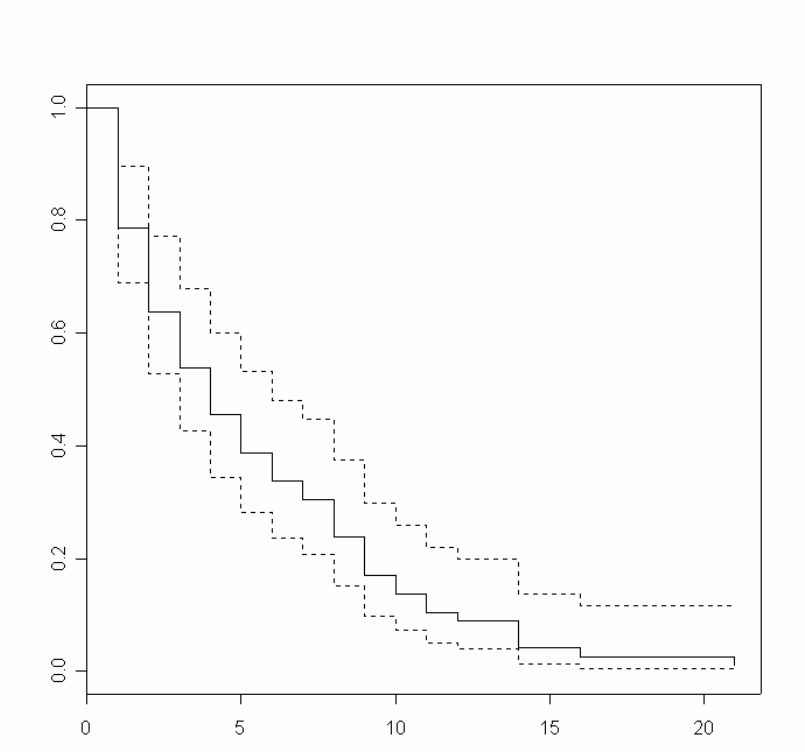
```
  n events mean se(mean) median 0.95LCL 0.95UCL
 60     60 5.49    0.619      4       3       6
```

Using the summary function gives the full survival table and confidence intervals

```
summary(model2)
```

```
Call: survfit.coxph(object = coxph(Surv(death, status) ~ gapsize))

 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    1     60      13  0.78600  0.0527     0.689271        0.896
    2     47       9  0.63769  0.0618     0.527386        0.771
    3     38       6  0.53812  0.0641     0.426011        0.680
    4     32       5  0.45467  0.0641     0.344883        0.599
    5     27       4  0.38793  0.0628     0.282503        0.533
    6     23       3  0.33807  0.0609     0.237442        0.481
    7     20       2  0.30474  0.0593     0.208086        0.446
    8     18       4  0.23773  0.0549     0.151129        0.374
    9     14       4  0.17098  0.0487     0.097862        0.299
   10     10       2  0.13785  0.0446     0.073147        0.260
   11      8       2  0.10501  0.0396     0.050169        0.220
   12      6       1  0.08885  0.0366     0.039606        0.199
   14      5       3  0.04132  0.0252     0.012487        0.137
   16      2       1  0.02565  0.0199     0.005607        0.117
   21      1       1  0.00935  0.0119     0.000771        0.113
```

```
plot(model2)
```



To test for differences in baseline survival for each cohort, we use **survdiff** like this:

survdiff(Surv(death,status)~cohort)

```
Call:
survdiff(formula = Surv(death, status) ~ cohort)

                   N Observed Expected (O-E)^2/E (O-E)^2/V
   cohort=October 30       30     32.9     0.259     0.722
 cohort=September 30       30     27.1     0.315     0.722

 Chisq= 0.7  on 1 degrees of freedom, p= 0.395
```

The baseline survival is not significantly different for the two cohorts.

For a full analysis of covariance, fitting gap size separately for each cohort we use the **strata** option in the model formula with Cox proportional hazards:

coxph(Surv(death,status)~strata(cohort)*gapsize)

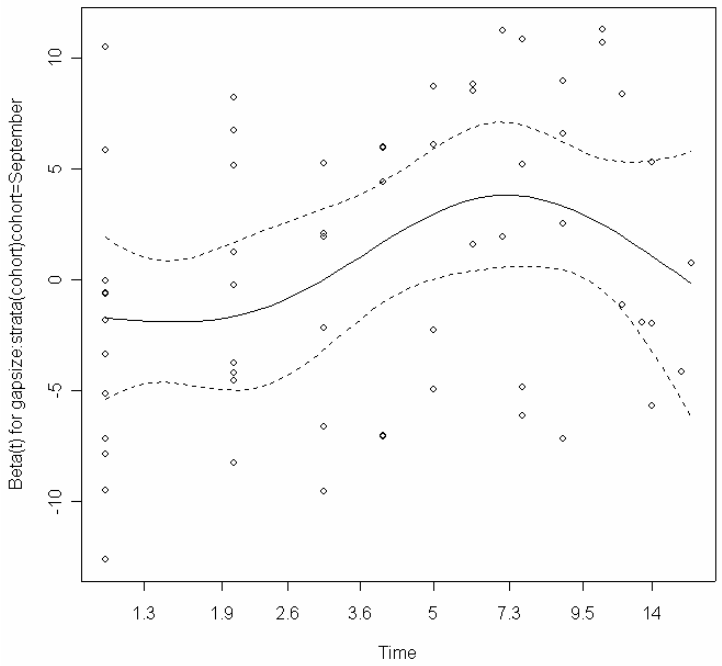
```
Call:
coxph(formula = Surv(death, status) ~ strata(cohort) * gapsize)

                                  coef exp(coef) se(coef)        z   p
```

```
gapsize                          -0.00189     0.998    0.593 -0.00319 1.0
gapsize:strata(cohort)Septembr    0.71741     2.049    0.861  0.83341 0.4

Likelihood ratio test=1.35  on 2 df, p=0.51  n= 60
```

Gapsize has no effect on survival in either cohort.

To test whether the coefficients are a function of time, we use the **cox.zph** function

```
model3<-cox.zph(coxph(Surv(death,status)~strata(cohort)*gapsize))
plot(model3)
```



There is no evidence of temporal changes in the parameters (you can easily fit a horizontal line between the confidence intervals). This is a truly dull data set. Nothing is happening at all.

```
detach(seedlings)
```

Censoring

Censoring occurs when we do not know the time to death for all of the individuals. This comes about principally because some individuals outlive the experiment. We can say they survived for the duration of the study but we have no way of knowing at what age they will die. These individuals contribute something to our knowledge of the survivor function, but nothing to our knowledge of the age at death. Another reason for censoring occurs when individuals are lost from the study; they may be killed in accidents, they may emigrate, or they may lose their identity tags.

In general, then, our survival data may be a mixture of times at death and times after which we have no more information on the individual. We deal with this by setting up an extra vector called the *censoring indicator* to distinguish between the two kinds of numbers. If a time really is a time to death, then the censoring indicator takes the value 1. If a time is just the last time we saw an individual alive, then the censoring indicator is set to 0. Thus, if we had the time data T and censoring indicator W:

```
T  4 7 8 8 12 15 22
W  1 1 0 1 1  0  1
```

this would mean that all the data were genuine times at death except for two cases, one at time 8 and another at time 15, when animals were seen alive but never seen again.

With repeated sampling in survivorship studies, it is usual for the degree of censoring to decline as the study progresses. Early on, many of the individuals are alive at the end of each sampling interval, whereas few if any survive to the end of the last study period.

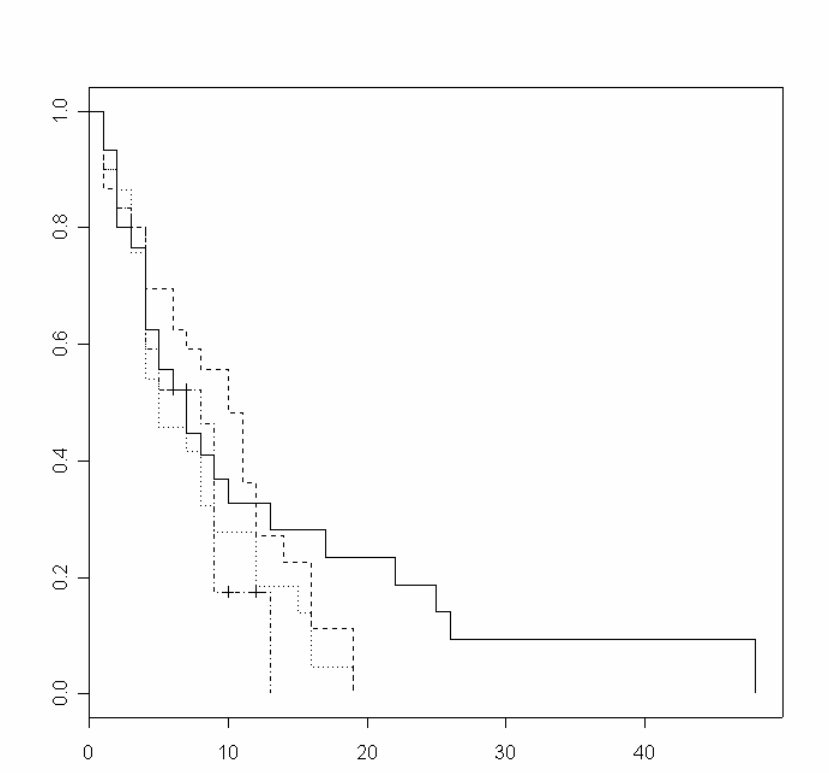
An example with censoring and non-constant hazard

This study involved 4 cohorts of cancer patients each of 30 individuals. They were allocated to Drug A, Drug B, Drug C or a placebo at random. The year in which they died (recorded as time after treatment began) is the response variable. Some patients left the study before their age at death was known (these are the censored individuals with status = 0). Remember to **remove the variables status and death** between each analysis:

```
rm(status)
rm(death)

cancer<-read.table("c:\\temp\\cancer.txt",header=T)
attach(cancer)
names(cancer)

[1] "death"     "treatment" "status"
```

We start by plotting the survivorship curves for patients in the 4 different treatments:

```
plot(survfit(Surv(death,status)~treatment),lty=c(1,2,3,4))
```



The mean age at death for the 4 treatments (ignoring censoring) was

```
tapply(death[status==1],treatment[status==1],mean)
```

```
   DrugA    DrugB    DrugC  placebo
9.480000 8.360000 6.800000 5.238095
```

The patients receiving Drug A lived an average of more than 4 years longer than those receiving the placebo. We need to test the significance of these differences, remembering that the variance in age-at-death is very high:

```
tapply(death[status==1],treatment[status==1],var)
```

```
    DrugA     DrugB     DrugC   placebo
117.51000  32.65667  27.83333  11.39048
```

We start with the simplest model, assuming exponential errors and constancy in the risk of death:

```
model<-survreg(Surv(death,status)~treatment,dist="exponential")
summary(model)
```

```
                  Value Std. Error      z        p
(Intercept)       2.448      0.200 12.238 1.95e-34
treatmentDrugB   -0.125      0.283 -0.442 6.58e-01
```

```
treatmentDrugC   -0.430      0.283 -1.520 1.28e-01
treatmentplacebo -0.333      0.296 -1.125 2.61e-01

Scale fixed at 1

Exponential distribution
Loglik(model)= -310.1   Loglik(intercept only)= -311.5
        Chisq= 2.8 on 3 degrees of freedom, p= 0.42
```

This analysis does not distinguish between the 4 treatments, despite the large differences in mean age at death. The placebo is not significantly different from Drug A ($p = 0.261$). Next we try a model with a different error assumption: the extreme value distribution:

```
model<-survreg(Surv(death,status)~treatment,dist="extreme")
summary(model)
```

```
                  Value Std. Error     z         p
(Intercept)       22.91     2.0686 11.07  1.69e-28
treatmentDrugB   -11.16     2.7548 -4.05  5.13e-05
treatmentDrugC   -13.38     2.7487 -4.87  1.12e-06
treatmentplacebo -13.29     2.9357 -4.53  5.97e-06
Log(scale)         2.21     0.0717 30.76 8.32e-208

Scale= 9.08

Extreme value distribution
Loglik(model)= -371.7   Loglik(intercept only)= -383.7
        Chisq= 23.94 on 3 degrees of freedom, p= 2.6e-05
```

This model indicates significant effects of treatment on death rate: Drug A gave improved survival compared to Drugs B and C and the placebo, but the difference between Drug C and the placebo was not significant (t= (13.38 – 13.29)/ 2.9357). The scale parameter = 9.08 indicates that mortality is significantly age dependent, so the earlier exponential distribution (scale = 1), assuming that mortality was not age dependent, was not appropriate. A full analysis of these data would fit more covariates (details about each patient) and test for time-dependency in the model parameters. The present point is that if, as here, the death risk is a function of age, then assuming the simpler exponential model does not allow us to detect the significant differences that exist between the treatments.

```
detach(cancer)
```

The likelihood function for censored data

A given individual contributes to the likelihood function depending upon whether it is alive or dead at time t. If it has died, then the censoring indicator is 1 and we learn more about *f(t)*; if it is still alive, then w_i is 0 and we learn only about *S(t)*:

$$L(\beta) = \prod_{i=1}^{n} [f(t_i)]^{w_i} [S(t_i)]^{1-w_i}$$

Now recall that the hazard function *h(t)* is given by *f(t)/S(t)*. Thus we have $S(t)^{w}$ in the denominator and $S(t)^{1-w}$ in the numerator, so the likelihood function becomes:

$$L(\beta) = \prod_{i=1}^{n} [h(t_i)]^{w_i} S(t_i)$$

which involves data only from the hazard function of the uncensored individuals. If we replace $1/\mu_i$ by λ_i, then we can write the likelihood function for the exponential distribution as follows:

$$L(\beta) = \prod_{i=1}^{n} \lambda_i^{w_i} e^{-\lambda t_i}$$

For reasons that will become clear in a moment, it is convenient to multiply both the numerator and the denominator by $\prod t_i^{w_i}$. This gives

$$L(\beta) = \frac{\prod_{i=1}^{n} (\lambda_i t_i)^{w_i} e^{-\lambda t_i}}{\prod_{i=1}^{n} t_i^{w_i}}$$

Because the denominator is not a function of the estimated parameters β, it can be omitted from the likelihood formula, leaving only a term for the likelihood of a set of *n* observations w_i, having independent Poisson distributions, with means $\lambda_i t_i$, where w_i is either 0 or 1. The model can be fit to the hazard rate λ in one of two ways. Let θ_i represent the Poisson mean $\lambda_i t_i$,. Using the *linear hazard model*

$$\lambda_i = \beta' x_i$$

and

$$\theta_i = \beta'(t_i x_i)$$

The modelling proceeds as follows:

- use the censoring indicator w_i as the response variable
- declare the error as Poisson
- declare the link function as the identity link
- multiply each of the explanatory variables, including the unit vector, by t_i
- fit the model as usual.

This is somewhat long-winded, and the fitting is easier if the *log linear hazard model is employed*, because

$$-\log \mu_i = \log \lambda = \beta' x_i$$

and so

$$\log \theta_i = \log \lambda_i + \log t_i = \beta' x_i + \log t_i$$

This is easier to fit, because we do not need to multiply through all the explanatory vectors by t_i. Instead, we use log t_i as an offset, and proceed as follows:

- use the censoring indicator w_i as the response variable;
- declare the error as Poisson;
- declare the link function as the log link;
- declare log t_i as an offset;
- fit the model as usual.

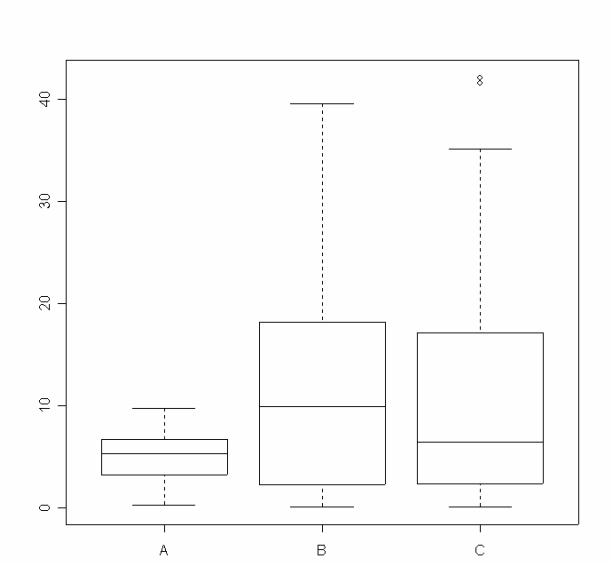
This rather curious procedure of using the censoring indicator full of 0's and 1's as the response variable should become clearer with an example.

An example with censoring

The next example comes from a study of mortality in 150 adult cockroaches. There were three experimental *group*s, and the animals were followed for 50 days. The groups were treated with three different insecticidal BT toxins added to their diet. The initial body mass of each insect (*weight*) was recorded as a covariate. The day on which each animal died (*death*) was recorded, and animals which survived up to the 50th day were recorded as being censored (for them, the censoring indicator *status* = 0).

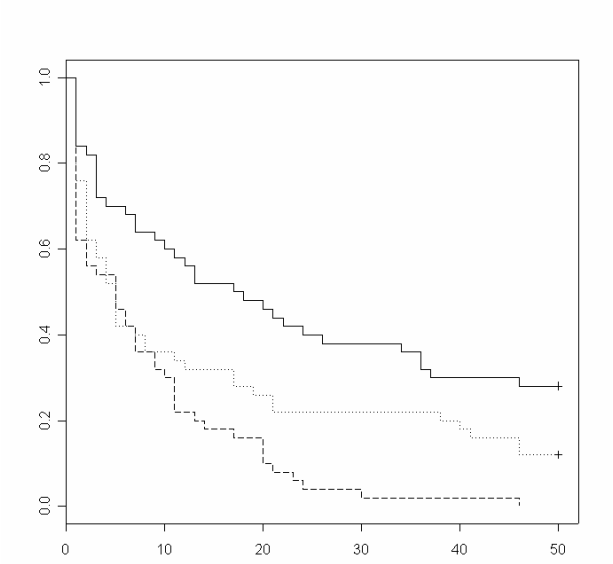
```
rm(status,death)
roaches<-read.table("c:\\temp\\roaches.txt",header=T)
attach(roaches)
names(roaches)
[1] "death"  "status" "weight" "group"
```

```
plot(group,weight)
```



The insects in batches B and C are much more variable than those in batch A. The overall survivorship curves for the 3 groups are obtained as before:

```
plot(survfit(Surv(death,status)~group),lty=c(1,3,5))
```



The crosses + at the end of the survivorship curves for groups A and B indicate that there was censoring in these groups (not all of the individuals were dead at the end of the experiment). Parametric regression in survival models uses the **survreg** function, for which you can specify a wide range of different error distributions. Here we use the exponential distribution for the purposes of demonstration (we can chose from dist =

"extreme", "logistic", "gaussian" or "exponential" and from link = "log" or "identity"), and fit the full analysis of covariance model to begin with:

```
model<-survreg(Surv(death,status)~weight*group,dist="exponential")
```

```
model
```

```
Call:
survreg(formula = Surv(death, status) ~ weight * group, dist =
"exponential")

Coefficients:
  (Intercept)        weight        groupB        groupC weight:groupB
   3.87018131   -0.08030300   -0.88529869   -1.78043935    0.06425282
weight:groupC
   0.07957341

Scale fixed at 1

Loglik(model)= -480.6   Loglik(intercept only)= -502.1
        Chisq= 43.11 on 5 degrees of freedom, p= 3.5e-08
n= 150
```

To see the parameter estimates and their standard errors, use **summary**:

```
summary(model)
```

```
                Value Std. Error      z        p
(Intercept)    3.8702     0.3854 10.041 1.00e-23
weight        -0.0803     0.0659 -1.219 2.23e-01
groupB        -0.8853     0.4508 -1.964 4.95e-02
groupC        -1.7804     0.4386 -4.059 4.92e-05
weight:groupB  0.0643     0.0674  0.954 3.40e-01
weight:groupC  0.0796     0.0674  1.180 2.38e-01

Scale fixed at 1

Exponential distribution
Loglik(model)= -480.6   Loglik(intercept only)= -502.1
        Chisq= 43.11 on 5 degrees of freedom, p= 3.5e-08
Number of Newton-Raphson Iterations: 4
n= 150
```

Model simplification proceeds in the normal way. You could use **update**, but here (for variety only) we re-fit progressively simpler models and test them using **anova**. First we take out the different slopes for each group:

```
model2<-survreg(Surv(death,status)~weight+group,dist="exponential")
```

```
anova(model,model2,test="Chi")
```

```
Terms Resid. Df    -2*LL          Test Df  Deviance P(>|Chi|)
1 weight * group       144 961.1800               NA        NA        NA
2 weight + group       146 962.9411 -weight:group -2 -1.761142 0.4145462
```

The interaction is not significant so we leave it out and try deleting weight:

model3<-survreg(Surv(death,status)~group,dist="exponential")

anova(model2,model3,test="Chi")

```
Terms Resid. Df    -2*LL    Test Df   Deviance P(>|Chi|)
1 weight + group       146 962.9411         NA        NA        NA
2          group       147 963.9393 -weight -1 -0.9981333 0.3177626
```

This is not significant, so we leave it out and try deleting group:

model4<-survreg(Surv(death,status)~1,dist="exponential")

anova(model3,model4,test="Chi")

```
Terms Resid. Df     -2*LL Test Df  Deviance    P(>|Chi|)
1 group       147  963.9393      NA        NA           NA
2     1       149 1004.2865      -2 -40.34721 1.732661e-09
```

This is highly significant, so we add it back . The minimal adequate model is model3 with the 3-level factor *group*, but there is no evidence that initial body *weight* had any influence on survival.

summary(model3)

```
             Value Std. Error     z        p
(Intercept)  3.467      0.167 20.80 3.91e-96
groupB      -0.671      0.225 -2.99 2.83e-03
groupC      -1.386      0.219 -6.34 2.32e-10

Scale fixed at 1

Exponential distribution
Loglik(model)= -482   Loglik(intercept only)= -502.1
        Chisq= 40.35 on 2 degrees of freedom, p= 1.7e-09
```

You can immediately see the advantage of doing proper survival analysis when you compare the predicted mean ages at death for *model3* with the crude arithmetic averages of the raw data on age at death:

tapply(predict(model3,type="response"),group,mean)

```
       A        B        C
32.05555 16.38635  8.02000
```

```
tapply(death,group,mean)

    A     B     C
23.08 14.42  8.02
```

If there is no censoring (all the individuals died, as in Group C) then the estimated mean ages at death are identical. But when there is censoring, the arithmetic mean underestimates the age at death, and when the censoring is substantial (as in Group A) this underestimate is very large (23.08 vs. 32.06).

Weibull distribution

The origin of the Weibull distribution is in *weakest link analysis*. If there are *r* links in a chain, and the strengths of each link Z_i are independently distributed $(0, \infty)$, then the distribution of weakest links $V = \min(Z_j)$ approaches the Weibull distribution as the number of links increases.

The Weibull is a 2-parameter model that has the exponential distribution as a special case. Its value in demographic studies and survival analysis is that it allows for the death rate to increase or to decrease with age, so that all 3 kinds of survivorship curve can be analysed (see above). The density, hazard and survival functions with $\lambda = \mu^{-\alpha}$ are:

$$f(t) = \alpha\lambda t^{\alpha-1} e^{-\lambda t^{\alpha}}$$

$$h(t) = \alpha\lambda t^{\alpha-1}$$

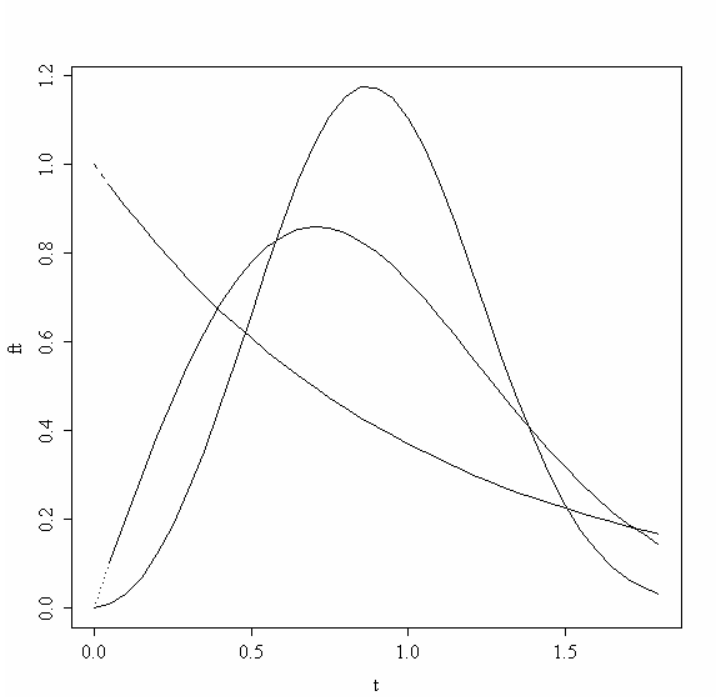
$$S(t) = e^{-\lambda t^{\alpha}}$$

The mean of the Weibull distribution is $\Gamma(1+\alpha^{-1})\mu$ and the parameter α describes the shape of the hazard function (the background to determining the likelihood equations is given by Aitkin et al. (1989) pp 281-283). For $\alpha = 1$ (the exponential distribution) the hazard is constant, while for $\alpha > 1$ the hazard increases with age and for $\alpha < 1$ the hazard decreases with age.

Because the Weibull, lognormal and log-logistic all have positive skewness, it will be difficult to discriminate between them with small samples. This is an important problem, because each distribution has differently shaped hazard functions, and it will be hard, therefore, to discriminate between different ecological assumptions about the age-specificity of death rates. In survival studies, parsimony requires that we fit the exponential rather than the Weibull when the shape parameter α is not significantly different from 1.

Here is a family of 3 Weibull distributions with $\alpha = 1, \alpha = 2$ and $\alpha = 3$ (dotted, dashed and solid lines, respectively). Note that for large values of α the distribution becomes symmetrical, while for $\alpha \leq 1$ the distribution has its mode at t = 0. The variable name is lower-case L (for lambda)

```
a<-3
l<-1
t<-seq(0,1.8,.05)
ft<-a*l*t^(a-1)*exp(-l*t^a)
plot(t,ft,type="l")
a<-1
ft<-a*l*t^(a-1)*exp(-l*t^a)
lines(t,ft,type="l",lty=2)
a<-2
ft<-a*l*t^(a-1)*exp(-l*t^a)
lines(t,ft,type="l",lty=3)
```



Recall that that the shape parameter of the Weibull reflects the way that the risk of death changes as a function of age.

An example of censored survival data analysed using glm with Poisson errors

Suppose that in another GM trial we have two groups of caterpillars, each comprising 21 larvae of the same initial size and age. They are fed on leaves from two kinds of plants; the first group get leaves from a plant that has been genetically engineered to express BT toxin in its foliage, while the second group get leaves from an otherwise identical, but

non-transgenic strain. The data consist of the time at death in days (when $w = 1$) or the time when the animal was lost to the study ($w = 0$). The survival analysis is very simple. The response variable is the censoring indicator (*status*) with Poisson errors and log(*time*) as an offset:

```
rm(status)

transgenic<-read.table("c:\\temp\\transgenic.txt",header=T)
attach(transgenic)
names(transgenic)
[1] "time"   "status" "diet"
```

where *time* is age at death (days), *status* is the censoring indicator (1 = dead, 0 = alive at the end of the experiment), and *diet* is a 2-level factor. Preliminary data inspection involves calculating the mean age at death for insect fed on control and transgenic Bt expressing leaves using **tapply**:

```
tapply(time,diet,mean)


   control transgenic
 17.095238   8.666667
```

Evidently the control insects lived much longer on average. It is useful to know how the censoring is distributed across individuals in the different treatments. We use **table** to count the cases:

```
table(status,diet)

status control transgenic
     0 12       0
     1  9      21
```

That is very revealing. All of the censoring (12 cases of *status* = 0) occurred in the control group of insects. None of the insects fed on Bt expressing leaves survived until the end of the experiment. Nine control insects died compared with all 21 of the insects fed a diet of transgenic leaves. The model has the **censoring indicator as the response variable**, and **the variation to be explained by the model is introduced by the offset** (log time of death):

```
model<-glm(status~diet+offset(log(time)),family=poisson)
```

Note how the **offset** appears as an additive part of the model formula.

```
summary(model)

Coefficients:
```

```
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -3.6861     0.3333 -11.060 < 2e-016 ***
diet          1.5266     0.3984   3.832 0.000127 ***

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 54.503  on 41  degrees of freedom
Residual deviance: 38.017  on 40  degrees of freedom
AIC: 102.02
```

Diet looks highly significant, but we prefer to test by deletion, using chi square:

```
model2<-glm(status~1+offset(log(time)),family=poisson)
anova(model,model2,test="Chi")
```

```
Analysis of Deviance Table

Model 1: status ~ diet + offset(log(time))
Model 2: status ~ 1 + offset(log(time))
  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1        40     38.017
2        41     54.503 -1  -16.485 4.903e-05
```

So, no doubt there then. Diet had an enormously significant effect on mean time at death. It is useful to know how to back transform these coefficient tables when there are offsets.

The antilogs of the estimates give the hazard. The mean age at death is the reciprocal of the hazard. So mean age at death is given by

```
1/exp(tapply(predict(model)-log(time),diet,mean))
```

```
   control transgenic
 39.888889   8.666667
```

Note that where there was lots of censoring (as in the case of the control insects) the estimated mean age at death is substantially *greater* than the arithmetic mean age at death

```
tapply(time,diet,mean)
```

```
   control transgenic
 17.095238   8.666667
```

(39.89 vs. 17.10) whereas the non-censored means are identical (8.67 vs. 8.67).