# 10

# Regression

Regression analysis is the statistical method you use when both the response variable and the explanatory variable are continuous variables (i.e. real numbers with decimal places – things like heights, weights, volumes, or temperatures). Perhaps the easiest way of knowing when regression is the appropriate analysis is to see that a scatterplot is the appropriate graphic (in contrast to analysis of variance, say, where it would have been a box-and-whisker plot or a bar chart). We cover seven important kinds of regression analysis in this book:

- linear regression (the simplest, and much the most frequently used);
- polynomial regression (often used to test for non-linearity in a relationship);
- piecewise regression (two or more adjacent straight lines);
- robust regression (models that are less sensitive to outliers);
- multiple regression (where there are numerous explanatory variables);
- non-linear regression (to fit a specified non-linear model to data);
- non-parametric regression (used when there is no obvious functional form).

The first five cases are covered here, non-linear regression in Chapter 20 and non-parametric regression in Chapter 18 (where we deal with generalized additive models and non-parametric smoothing).

The essence of regression analysis is using sample data to estimate parameter values and their standard errors. First, however, we need to select a model which describes the relationship between the response variable and the explanatory variable(s). The simplest of all is the linear model

$$y = a + bx.$$

There are two variables and two parameters. The response variable is $y$, and $x$ is a single continuous explanatory variable. The parameters are $a$ and $b$: the intercept is $a$ (the value of $y$ when $x = 0$); and the slope is $b$ (the change in $y$ divided by the change in $x$ which brought it about).

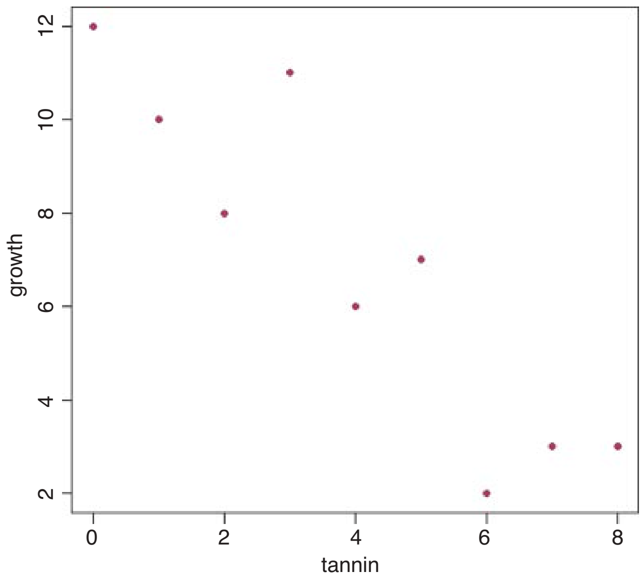
## 10.1 Linear regression

Let us start with an example which shows the growth of caterpillars fed on experimental diets differing in their tannin content:

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

```
[1]  "growth"  "tannin"
```

```
plot(tannin,growth,pch=21,col="blue",bg="red")
```



The higher the percentage of tannin in the diet, the more slowly the caterpillars grew. You can get a crude estimate of the parameter values by eye. Tannin content increased by 8 units, in response to which growth declined from about 12 units to about 2 units, a change of –10 units of growth. The slope, $b$, is the change in $y$ divided by the change in $x$, so

$$b \approx \frac{-10}{8} = -1.25.$$

The intercept, $a$, is the value of $y$ when $x = 0$, and we see by inspection of the scatterplot that growth was close to 12 units when tannin was zero. Thus, our rough parameter estimates allow us to write the regression equation as

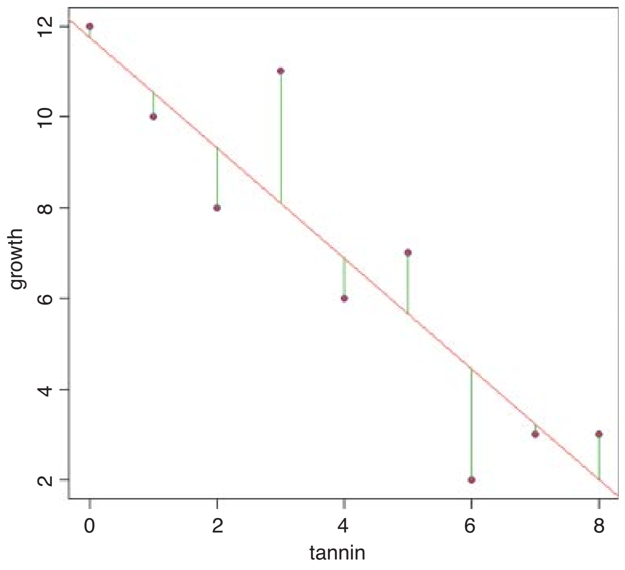
$$y \approx 12.0 - 1.25x.$$

Of course, different people would get different parameter estimates by eye. What we want is an objective method of computing parameter estimates from the data that are in some sense the 'best' estimates of the parameters for these data and this particular model. The convention in modern statistics is to use the **maximum**

**likelihood estimates** of the parameters as providing the 'best' estimates. That is to say that, given the data, and having selected a linear model, we want to find the values of the slope and intercept that make the data most likely. Keep re-reading this sentence until you understand what it is saying.

For the simple kinds of regression models with which we begin, we make several important assumptions:

- The variance in $y$ is constant (i.e. the variance does not change as $y$ gets bigger).
- The explanatory variable, $x$, is measured without error.
- The difference between a measured value of $y$ and the value predicted by the model for the same value of $x$ is called a residual.
- Residuals are measured on the scale of $y$ (i.e. parallel to the $y$ axis).
- The residuals are normally distributed.



```
model <- lm(growth~tannin)
abline(model,col="red")
yhat <- predict(model,tannin=tannin)
join <- function(i)
 lines(c(tannin[i],tannin[i]),c(growth[i],yhat[i]),col="green")
sapply(1:9,join)
```

Under these assumptions, the maximum likelihood is given by the **method of least squares**. The phrase 'least squares' refers to the residuals, as shown in the figure. The residuals are the vertical differences between the data (solid circles) and the fitted model (the straight line). Each of the residuals is a distance, $d$, between a data point, $y$, and the value predicted by the fitted model, $\hat{y}$, evaluated at the appropriate value of the explanatory variable, $x$:

$$d = y - \hat{y}.$$

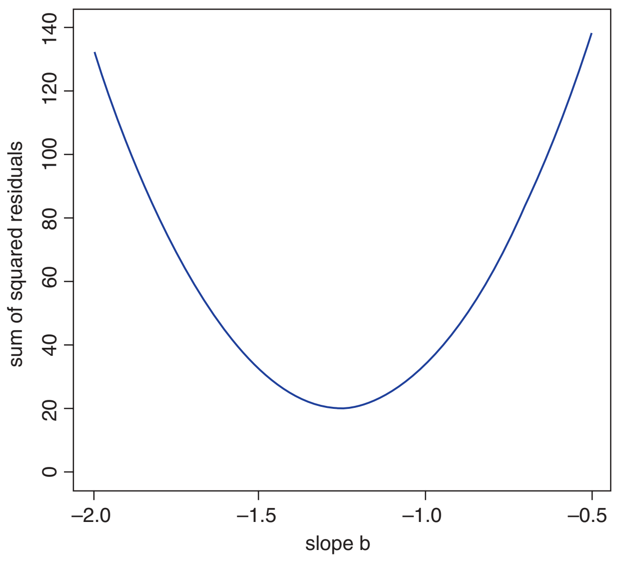
Now we replace the predicted value $\hat{y}$ by its formula $\hat{y} = a + bx$, noting the change in sign:

$$d = y - a - bx.$$

Finally, our measure of lack of fit is the sum of the squares of these distances:

$$\sum d^2 = \sum (y - a - bx)^2.$$

The sum of the residuals will always be zero, because the positive and negative residuals cancel out, so $\sum d$ is no good as a measure of lack of fit (although $\sum |d|$ is useful in computationally intensive statistics; see p. 65). The best fit line is defined as passing through the point defined by the mean value of $x$ ($\bar{x}$) and the mean value of $y$ ($\bar{y}$). The large open circle marks the point($\bar{x}$, $\bar{y}$). You can think of maximum likelihood as working as follows. Imagine that the straight line is pivoted, so that it can rotate around the point($\bar{x}$, $\bar{y}$). When the line is too steep, some of the residuals are going to be very large. Likewise, if the line is too shallow, some of the residuals will again be very large. Now ask yourself what happens to the sum of the squares of the residuals as the slope is rotated from too shallow, through just right, to too steep. The sum of squares will be big at first, then decline to a minimum value, then increase again. A graph of the sum of squares against the value of the slope used in estimating it would look like this:



```
bs <- seq(-2,-0.5,0.01)
SSE <- function(i) sum((growth - 12 - bs[i]*tannin)^2)
plot(bs,sapply(1:length(bs),SSE),type="l",ylim=c(0,140),
            xlab="slope b",ylab="sum of squared residuals",col="blue")
```

The maximum likelihood estimate of the slope is the value of $b$ associated with the minimum value of the sum of the squares of the residuals (i.e. close to –1.25). Ideally we want an analytic solution that gives the maximum likelihood of the slope directly (this is done using calculus in Box 10.1). It turns out, however, that the least-squares estimate of $b$ can be calculated very simply from the covariance of $x$ and $y$ (which we met on p. 304).

### 10.1.1 The famous five in R

We want to find the minimum value of $\sum d^2$. To work this out we need the 'famous five': these are $\sum y^2$ and $\sum y$, $\sum x^2$ and $\sum x$, and the sum of products, $\sum xy$ (introduced on p. 331). The sum of products is worked out pointwise. You can calculate the numbers from the data the long way:

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

```
[1] 36
[1] 204
[1] 62
[1] 536
[1] 175
```

Alternatively, as we saw on p. 332, you can create a matrix and use matrix multiplication:

```
XY <- cbind(1,growth,tannin)
t(XY) %*% XY
```

```
          growth tannin
        9     62     36
growth 62    536    175
tannin 36    175    204
```

### 10.1.2 Corrected sums of squares and sums of products

The next thing is to use the famous five to work out three essential 'corrected sums'. We are already familiar with corrected sums of squares, because these are used in calculating variance: $s^2$ is calculated as the corrected sum of squares divided by the degrees of freedom (p. 333). We shall need the corrected sums of squares of both the explanatory variable, *SSX*, and the response variable, *SSY*:

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

$$SSY = \sum y^2 - \frac{\left(\sum y\right)^2}{n}.$$

The third term is the corrected sum of products, *SSXY*. The covariance of $x$ and $y$ is the expectation of the vector product $\mathrm{E}\left[(x-\bar{x})(y-\bar{y})\right]$, and this depends on the value of the corrected sum of products (p. 334), which is given by

$$SSXY = \sum xy - \frac{\left(\sum x\right)\left(\sum y\right)}{n}.$$

If you look carefully you will see that the corrected sum of products has exactly the same kind of structure as *SSY* and *SSX*. For *SSY*, the first term is the sum of $y$ times $y$ and the second term contains the sum of $y$ times the sum of $y$ (and similarly for *SSX*). For *SSXY*, the first term contains the sum of $x$ times $y$ and the second term contains the sum of $x$ times the sum of $y$.

Note that for accuracy within a computer program it is best *not* to use these shortcut formulae, because they involve differences (minus) between potentially very large numbers (sums of squares) and hence are potentially subject to rounding errors. Instead, when programming, use the following equivalent formulae:

$$SSY = \sum (y - \bar{y})^2,$$
$$SSX = \sum (x - \bar{x})^2,$$
$$SSXY = \sum (y - \bar{y})(x - \bar{x}).$$

The three key quantities *SSY, SSX* and *SSXY* can be computed the long way, substituting the values of the famous five:

$$SSY = 536 - \frac{62^2}{9} = 108.8889,$$
$$SSX = 204 - \frac{36^2}{9} = 60,$$
$$SSXY = 175 - \frac{36 \times 62}{9} = -73.$$

Alternatively, the matrix can be used (see p. 334).

The next question is how we use *SSX*, *SSY* and *SSXY* to find the maximum likelihood estimates of the parameters and their associated standard errors. It turns out that this step is much simpler than what has gone before. The maximum likelihood estimate of the slope, $b$, is just

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

(the detailed derivation of this is in Box 10.1). So, for our example,

$$b = \frac{-73}{60} = -1.216667.$$

Compare this with our by-eye estimate of –1.25. Now that we know the value of the slope, we can use any point that we know to lie on the fitted straight line to work out the maximum likelihood estimate of the intercept, $a$. One part of the definition of the best-fit straight line is that it passes through the point $(\bar{x}, \bar{y})$ determined by the mean values of $x$ and $y$. Since we know that $y = a + bx$, it must be the case that $\bar{y} = a + b\bar{x}$, and so

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

and, using R as a calculator, we get the value of the intercept as

```
mean(growth)+1.216667*mean(tannin)
```

```
[1]  11.75556
```

noting the change of sign. This is reasonably close to our original estimate by eye ($a \approx 12$).

The function for carrying out linear regression in R is `lm` (which stands for 'linear model'). The response variable comes first (`growth` in our example), then the tilde ~, then the name of the continuous explanatory variable (`tannin`). R prints the values of the intercept and slope like this:

```
lm(growth~tannin)


Coefficients:
(Intercept)      tannin
     11.756      -1.217
```

We can now write the maximum likelihood equation like this:

$$\texttt{growth} = 11.755\,56 - 1.216\,667 \times \texttt{tannin}.$$

**Box 10.1 The least-squares estimate of the regression slope, *b***

The **best fit** slope is found by rotating the line until the *error sum of squares*, *SSE*, is minimized, so we want to find the minimum of $\sum(y-a-bx)^2$. We start by finding the derivative of *SSE* with respect to *b*:

$$\frac{\mathrm{d}SSE}{\mathrm{d}b} = -2\sum x\,(y-a-bx)\,.$$

Now, multiplying through the bracketed term by $x$ gives

$$\frac{\mathrm{d}SSE}{\mathrm{d}b} = -2\sum xy - ax - bx^2.$$

Apply summation to 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 cannot solve the equation as it stands because there are two unknowns, $a$ and $b$. However, we know that the value of $a$ is $\bar{y} - b\bar{x}$. 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 bracketed term by $\sum x$ to get:

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

Next, take the two terms containing $b$ to the right-hand side, and note their change of sign:

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

Finally, divide both sides by $\sum x^2 - \left(\sum x\right)^2/n$ to obtain the required estimate $b$:

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

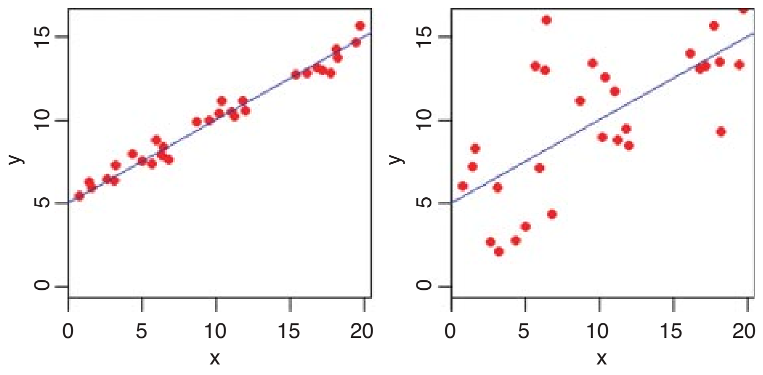
Thus, the value of $b$ that minimizes the sum of squares of the departures is given simply by:

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

This is the **maximum likelihood estimate of the slope** of the linear regression.

### 10.1.3 Degree of scatter

There is another very important issue that needs to be considered, because two data sets with exactly the same slope and intercept could look quite different:



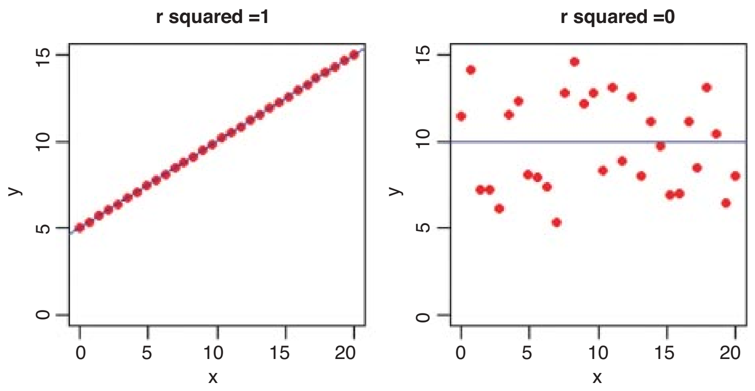
We need a way to quantify the degree of fit, so that the graph on the left has a high value and the graph on the right has a low value. It turns out that we already have the appropriate quantity: it is the sum of squares of the residuals (p. 338). This is referred to as the *error sum of squares*, *SSE*. Here, **error** does not mean 'mistake', but refers to residual variation or *unexplained variation*:

$$SSE = \sum (y - a - bx)^2.$$

Graphically, you can think of *SSE* as the sum of the squares of the lengths of the vertical residuals (the green lines) in the plot on p. 452. By tradition, however, when talking about the degree of scatter we actually quantify the *lack* of scatter, so the graph on the left, with a perfect fit (zero scatter) gets a value of 1, and the graph on the right, which shows no relationship at all between $y$ and $x$ (100% scatter), gets a value of 0. This quantity used to measure the lack of scatter is officially called the 'coefficient of determination', but everybody refers to it as '$r$ squared'. This is an important definition that you should try to memorize: $r^2$ is the fraction of the total variation in $y$ that is explained by variation in $x$. We have already defined the total variation in the response variable as *SSY* (p. 454). The unexplained variation in the response variable is defined above as *SSE* (the error sum of squares) so the explained variation is simply *SSY* – *SSE*. Thus,

$$r^2 = \frac{SSY - SSE}{SSE}.$$

A value of $r^2 = 1$ means that all of the variation in the response variable is explained by variation in the explanatory variable (the left-hand graph below) while a value of $r^2 = 0$ means none of the variation in the response variable is explained by variation in the explanatory variable (the right-hand graph).



```
y <- 5+0.5*x
plot(x,y,pch=16,xlim=c(0,20),ylim=c(0,15),col="red",main="r squared = 1")
abline(5,0.5,col="blue")
y <- 5+runif(30)*10
plot(x,y,pch=16,xlim=c(0,20),ylim=c(0,15),col="red",main="r squared = 0")
abline(h=10,col="blue")
```

You can get the value of *SSY* the long way as on p. 454 ($SSY = 108.8889$), or using R to fit the null model in which growth is described by a single parameter, the intercept *a*. In R, the intercept is called parameter 1, so the null model is expressed as `lm(growth~1)`. There is a function called `deviance` that can be applied to a linear model which returns the sum of the squares of the residuals (in this null case, it returns $\sum(y - \bar{y})^2$, which is *SSY* as we require):

```
deviance(lm(growth~1)

[1]  108.8889
```

The value of *SSE* is worked out longhand from $\sum(y - a - bx)^2$ but this is a pain, and the value can be extracted very simply from the regression model using `deviance` like this:

```
deviance(lm(growth~tannin))

[1]  20.07222
```

Now we can calculate the value of $r^2$:

$$r^2 = \frac{SSY = SSE}{SSY} = \frac{108.8889 - 20.072\ 22}{108.8889} = 0.815\ 663\ 3.$$

You will not be surprised that the value of $r^2$ can be extracted from the model:

```
summary(lm(growth~tannin))[[8]]
```

```
[1] 0.8156633
```

The **correlation coefficient**, $r$, introduced on p. 373, is given by

$$r = \frac{SSXY}{\sqrt{SSX \times SSY}}.$$

Of course $r$ is the square root of $r^2$, but we use the formula above so that we retain the sign of the correlation: $SSXY$ is positive for positive correlations between $y$ and $x$ and negative for negative correlations between $y$ and $x$. For our example, the correlation coefficient is

$$r = \frac{-73}{\sqrt{60 \times 108.8889}} = -0.903\ 140\ 7.$$

### 10.1.4 Analysis of variance in regression: *SSY* = *SSR* + *SSE*

The idea is simple: we take the total variation in $y$, $SSY$, and partition it into components that tell us about the explanatory power of our model. The variation that is explained by the model is called the regression sum of squares (denoted by $SSR$), and the unexplained variation is called the error sum of squares (denoted by $SSE$). Then $SSY = SSR + SSE$. Now, in principle, we could compute $SSE$ because we know that it is the sum of the squares of the deviations of the data points from the fitted model, $\sum d^2 = \sum (y - a - bx)^2$. Since we know the values of $a$ and $b$, we are in a position to work this out. The formula is fiddly, however, because of all those subtractions, squarings and addings-up. Fortunately, there is a very simple shortcut that involves computing $SSR$, the explained variation, rather than $SSE$. This is because

$$SSR = b \times SSXY = \frac{SSXY^2}{SSX},$$

so we can immediately work out $SSR = -1.21667 \times -73 = 88.816\ 67$. And since $SSY = SSR + SSE$, we can get $SSE$ by subtraction:

$$SSE = SSY - SSR = 108.8889 - 88.81667 = 20.07222.$$

Using R to do the calculations, we get:

```
(sse <- deviance(lm(growth~tannin)))
```

```
[1]  20.07222
```

```
(ssy <- deviance(lm(growth~1)))
```

```
[1]  108.8889
```

```
(ssr <- ssy-sse)
```

```
[1]  88.81667
```

We now have all of the sums of squares, and all that remains is to think about the degrees of freedom. We had to estimate one parameter, the overall mean, $\bar{y}$, before we could calculate $SSY = \sum (y - \bar{y})^2$, so the total degrees of freedom are $n - 1$. The error sum of squares was calculated only after two parameters had been estimated from the data (the intercept and the slope) since $SSE = \sum(y - bx)^2$, so the error degrees of freedom are $n - 2$. Finally, the regression model added just one parameter, the slope $b$, compared with the null model, so there is *one* regression degree of freedom. Thus, the ANOVA table looks like this:

| Source | Sum of squares | Degrees of freedom | Mean squares | $F$ ratio |
|---|---|---|---|---|
| Regression | 88.817 | 1 | 88.817 | 30.974 |
| Error | 20.072 | 7 | $s^2 = 2.867\,46$ | |
| Total | 108.889 | 8 | | |

Notice that the component degrees of freedom add up to the total degrees of freedom (this is always true, in any ANOVA table, and is a good check on your understanding of the design of the experiment). The third column, headed 'Mean squares', contains the variances obtained by dividing the sums of squares by the degrees of freedom in the same row. In the row labelled 'Error' we obtain the very important quantity called the error variance, denoted by $s^2$, by dividing the error sum of squares by the error degrees of freedom. Obtaining the value of the error variance is the main reason for drawing up the ANOVA table. Traditionally, one does not fill in the bottom box (it would be the overall variance in $y$, $SSY/(n - 1)$, although this is the basis of the adjusted $r^2$ value; see p. 461). Finally, the ANOVA table is completed by working out the $F$ ratio, which is a ratio between two variances. In most simple ANOVA tables, you divide the treatment variance in the numerator (the regression variance in this case) by the error variance $s^2$ in the denominator. The null hypothesis under test in a linear regression is that the slope of the regression line is zero (i.e. that there is no dependence of $y$ on $x$). The two-tailed alternative hypothesis is that the slope is significantly different from zero (either positive or negative). In many applications it is not particularly interesting to reject the null hypothesis, because we are interested in the estimates of the slope and its standard error (we often know from the outset that the null hypothesis is false). To test whether the $F$ ratio is sufficiently large to reject the null hypothesis, we compare the calculated value of $F$ in the final column of the ANOVA table with the critical value of $F$, expected by chance alone (this is found from quantiles of the $F$ distribution `qf`, with 1 d.f. in the numerator and $n - 2$ d.f. in the denominator, as described below). The table can be produced directly from the fitted model in R by using the `anova` function:

```
anova(lm(growth~tannin))
```

```
Analysis of Variance Table

Response: growth
          Df Sum Sq Mean Sq F value    Pr(>F)
tannin     1 88.817  88.817  30.974 0.0008461 ***
Residuals  7 20.072   2.867
```

The same output can be obtained using `summary.aov(lm(growth~tannin))`. The extra column given by R is the $p$ value associated with the computed value of $F$.

There are two ways to assess our $F$ ratio of 30.974. One way is to compare it with the critical value of $F$, with 1 d.f. in the numerator and 7 d.f. in the denominator. We have to decide on the level of uncertainty that we are willing to put up with; the traditional value for work like this is 5%, so our certainty is 0.95. Now we can use quantiles of the $F$ distribution, `qf`, to find the critical value of $F$:

```
qf(0.95,1,7)
```

```
[1]  5.591448
```

Because our calculated value of $F$ is much larger than this critical value, we can be confident in rejecting the null hypothesis. The other way, which is perhaps better than working rigidly at the 5% uncertainty level, is to ask what is the probability of getting a value for $F$ as big as 30.974 or larger if the null hypothesis is true. For this we use `1-pf` rather than `qf`:

```
1-pf(30.974,1,7)
```

```
[1]  0.0008460725
```

It is very unlikely indeed ($p < 0.001$). This value is in the last column of the R output. Note that the $p$ value is *not* the probability that the null hypothesis is true. On the contrary, it is the probability, given that the null hypothesis *is* true, of obtaining a value of $F$ this large or larger by chance alone.

### 10.1.5 Unreliability estimates for the parameters

Finding the least-squares values of slope and intercept is only half of the story, however. In addition to the parameter estimates, $a = 11.756$ and $b = -1.2167$, we need to measure the unreliability associated with each of the estimated parameters. In other words, we need to calculate the standard error of the intercept and the standard error of the slope. We have already met the standard error of the mean, and we used it in calculating confidence intervals (p. 122) and in doing Student's $t$ test (p. 358). Standard errors of regression parameters are similar in so far as they are enclosed inside a big square root term (so that the units of the standard error are the same as the units of the parameter), and they have the error variance, $s^2$, from the ANOVA table (above) in the numerator. There are extra components, however, which are specific to the unreliability of a slope or an intercept (see Boxes 10.2 and 10.3 for details).

**Box 10.2 Standard error of the slope**

The uncertainty of the estimated slope increases with increasing variance and declines with increasing number of points on the graph. In addition, however, the uncertainty is greater when the range of $x$ values (as measured by $SSX$) is small:

$$se_b = \sqrt{\frac{s^2}{SSX}}.$$

**Box 10.3 Standard error of the intercept**

The uncertainty of the estimated intercept increases with increasing variance and declines with increasing number of points on the graph. As with the slope, uncertainty is greater when the range of $x$ values (as measured by $SSX$) is small. Uncertainty in the estimate of the intercept also increases with the square of the distance between the origin and the mean value of $x$ (as measured by $\sum x^2$):

$$se_a = \sqrt{\frac{s^2 \sum x^2}{n \times SSX}}$$

Longhand calculation shows that the standard error of the slope is

$$se_b = \sqrt{\frac{s^2}{SSX}} = \sqrt{\frac{2.867}{60}} = 0.2186,$$

and the standard error of the intercept is

$$se_a = \sqrt{\frac{s^2 \Sigma x^2}{n \times SSX}} = \sqrt{\frac{2.867 \times 204}{9 \times 60}} = 1.0408.$$

However, in practice you would always use the `summary.lm` function applied to the fitted linear model like this:

```
summary(lm(growth~tannin))
```

```
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 ***
```

I have stripped out the details about the residuals and the explanation of the significance stars in order to highlight the parameter estimates and their standard errors (as calculated above). The residual standard error is the square root of the error variance from the ANOVA table ($1.693 = \sqrt{2.867}$). Multiple $R$-squared is the fraction of the total variance explained by the model ($SSR/SSY = 0.8157$). The adjusted $R$-squared is close to, but different from, the value of $r^2$ we have just calculated. Instead of being based on the explained sum of squares, $SSR$, and the total sum of squares, $SSY$, it is based on the overall variance (a quantity we do not typically calculate), $s_T^2 = SSY/(n-1) = 13.611$, and the error variance $s^2$ (from the ANOVA table, $s^2 = 2.867$) and is worked out like this:

$$\text{adjusted}R\text{-squared} = \frac{s_T^2 - s^2}{s_T^2}.$$

So in this example, adjusted R-squared = (13.611 – 2.867)/13.611 = 0.7893. We discussed the $F$ statistic and $p$ value in the previous section.

The `summary.lm` table shows everything you need to know about the parameters and their standard errors, but there is a built-in function, `confint`, which produces 95% confidence intervals for the estimated parameters from the model directly like this:

```
confint(model)
```

```
                2.5 %     97.5 %
(Intercept)  9.294457 14.2166544
tannin      -1.733601 -0.6997325
```

These values are obtained by subtracting from, and adding to, each parameter estimate an interval which is the standard error times Student's $t$ with 7 degrees of freedom (the appropriate value of $t$ is given by `qt(.975,7)` = 2.364 624). The fact that neither interval includes 0 indicates that both parameter values are significantly different from zero, as established by the earlier $F$ tests.

Of the two sorts of summary table, `summary.lm` is by far the more informative, because it shows the effect sizes (in this case the slope of the graph) and their unreliability estimates (the standard error of the slope). Generally, you should resist the temptation to put ANOVA tables in your written work. The important information such as the $p$ value and the error variance can be put in the text, or in figure legends, much more efficiently. ANOVA tables put far too much emphasis on hypothesis testing, and show nothing directly about effect sizes.

**Box 10.4 Standard error for a predicted value**

The standard error of a predicted value $\hat{y}$ is given by:

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

It increases with the *square* of the difference between mean $x$ and the value of $x$ at which the prediction is made. As with the standard error of the slope, the wider the range of $x$ values, $SSX$, the lower the uncertainty. The bigger the sample size, $n$, the lower the uncertainty. 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).

For predictions made on the basis of the regression equation we need to know the standard error for a predicted single sample of $y$,

$$se_y\sqrt{s^2\left[1 + \frac{1}{n} + \frac{(x-\bar{x})^2}{SSX}\right]},$$

while the standard error for a predicted mean for $k$ items at a given level of $x_i$ is

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

### 10.1.6 Prediction using the fitted model

It is good practice to save the results of fitting the model in a named object. Naming models is very much a matter of personal taste: some people like the name of the model to describe its structure, other people like the name of the model to be simple and to rely on the formula (which is part of the structure of the model) to describe what the model does. I like the second approach, so I might write

```
model <- lm(growth~tannin)
```

The object called `model` can now be used for all sorts of things. For instance, we can use the `predict` function to work out values for the response at values of the explanatory variable that we did not measure. Thus, we can ask for the predicted growth if tannin concentration was 5.5%. The value or values of the explanatory variable to be used for prediction are specified in a list like this:

```
predict(model,list(tannin=5.5))
```

```
[1]  5.063889
```

indicating a predicted growth rate of 5.06 if a tannin concentration of 5.5% had been applied. To predict growth at more than one level of tannin, the list of values for the explanatory variable is specified as a vector. Here are the predicted growth rates at 3.3, 4.4, 5.5 and 6.6% tannin:

```
predict(model,list(tannin=c(3.3,4.4,5.5,6.6)))
```

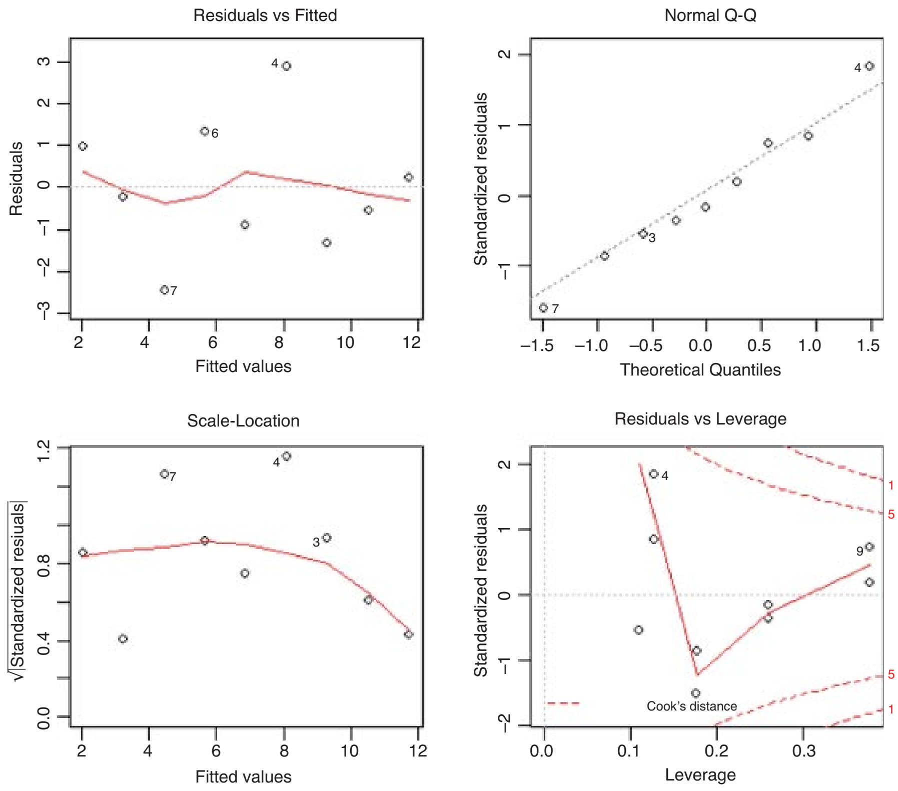
```
       1        2        3        4
7.740556 6.402222 5.063889 3.725556
```

For drawing smooth curves through a scatterplot we use `predict` with a vector of 100 or so closely-spaced *x* values, as illustrated on p. 207.

### 10.1.7 Model checking

The final thing you will want to do is to expose the model to critical appraisal. The assumptions we really want to be sure about are constancy of variance and normality of errors. The simplest way to do this is with model-checking plots. Six plots (selectable by `which`) are currently available: a plot of residuals against fitted values; a scale–location plot of $\sqrt{|\text{residuals}|}$ against fitted values; a normal qunatile–quantile plot; a plot of Cook's distances versus row labels; a plot of residuals against leverages; and a plot of Cook's distances against leverage/(1 – leverage). By default four plots are provided (the first three plus the fifth):

```
windows(7,7)
par(mfrow=c(2,2))
plot(model)
```



The first graph (top left) shows residuals on the *y* axis against fitted values on the *x* axis. It takes experience to interpret these plots, but what you *do not* want to see is lots of structure or pattern in the plot. Ideally, as here, the points should look like the sky at night. It is a major problem if the scatter increases as the fitted values get bigger; this would look like a wedge of cheese on its side (see p. 405). But in our present case, everything is OK on the constancy of variance front.

The next plot (top right) shows the normal `qqnorm` plot (p. 406) which should be a straight line if the errors are normally distributed. Again, the present example looks fine. If the pattern were S-shaped or banana-shaped, we would need to fit a different model to the data.

The third plot (bottom left) is a repeat of the first, but on a different scale; it shows the square root of the standardized residuals (where all the values are positive) against the fitted values. If there was a problem, such as the variance increasing with the mean, then the points would be distributed inside a triangular shape, with the scatter of the residuals increasing as the fitted values increase. The red line would then show a pronounced upward trend. But there is no such pattern here, which is good.

The fourth and final plot (bottom right) shows standardized residuals as a function of leverage, along with Cook's distance (p. 419) for each of the observed values of the response variable. The point of this plot is to highlight those *y* values that have the biggest effect on the parameter estimates (high influence; p. 409). You can see that point 9 has the highest leverage, but point 7 is quite influential (it is closest to the Cook's distance contour). You might like to investigate how much this influential point (6, 2) affected the parameter estimates and their standard errors. To do this, we repeat the statistical modelling but leave out the point in question, using `subset` like this (recall that `!=` means 'not equal to'):

```
model2 <- update(model,subset=(tannin != 6))
summary(model2)
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  11.6892     0.8963  13.042 1.25e-05 ***
tannin       -1.1171     0.1956  -5.712  0.00125 **
```

First of all, notice that we have lost one degree of freedom, because there are now eight values of *y* rather than nine. The estimate of the slope has changed from –1.2167 to –1.1171 (a difference of about 9%) and the standard error of the slope has changed from 0.2186 to 0.1956 (a difference of about 12%). What you do in response to this information depends on the circumstances. Here, we would simply note that point (6, 2) was influential and stick with our first model, using all the data. In other circumstances, a data point might be so influential that the structure of the model is changed completely by leaving it out. In that case, we might gather more data or, if the study was already finished, we might publish both results (with and without the influential point) so that the reader could make up their own mind about the interpretation. The important point is that we always do model checking; the `summary.lm(model)` table is not the end of the process of regression analysis.

You might also want to check for lack of serial correlation in the residuals (e.g. time series effects) using the `durbin.watson` function from the `car` package (see p. 484), but there are too few data to use it with this example.

## 10.2 Polynomial approximations to elementary functions

Elementary functions such sin($x$), log($x$) and exp($x$) can be expressed as Maclaurin series:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots,$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots,$$

$$\exp(x) = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots,$$

$$\log(x+1) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \ldots.$$

In fact, we can approximate any smooth continuous single-valued function by a polynomial of sufficiently high degree. To see this in action, consider the graph of sin($x$) against $x$ in the range $0 < x < \pi$ (where $x$ is an angle measured in radians):

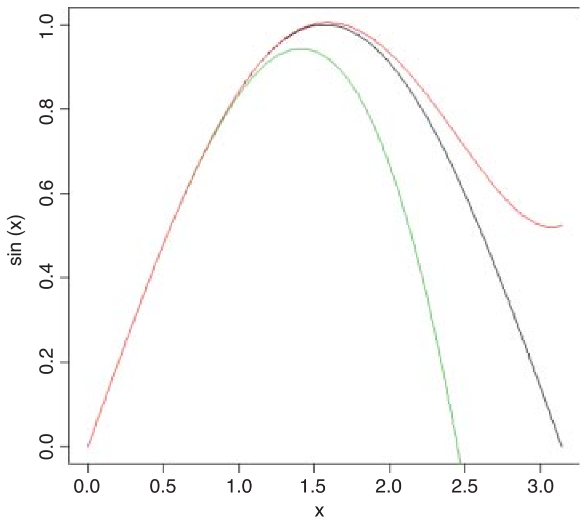
```
x <- seq(0,pi,0.01)
y <- sin(x)
plot(x,y,type="l",ylab="sin(x)")
```

Up to about $x = 0.3$ the very crude approximation sin($x$) = $x$ works reasonably well. The first approximation, including a single extra term for $-x^3/3!$, extends the reasonable fit up to about $x = 0.8$:

```
a1 <- x-x^3/factorial(3)
lines(x,a1,col="green")
```

Adding the term in $x^5/5!$ captures the first peak in sin($x$) quite well. And so on.

```
a2 <- x-x^3/factorial(3)+x^5/factorial(5)
lines(x,a2,col="red")
```

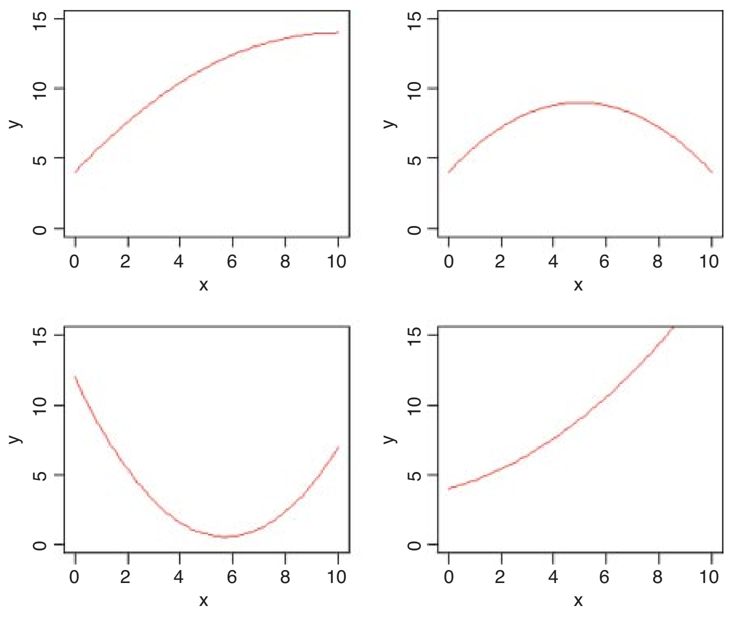


## 10.3 Polynomial regression

The relationship between $y$ and $x$ often turns out not to be a straight line. However, Occam's razor requires that we fit a straight-line model unless a non-linear relationship is significantly better at describing the data. So this begs the question: how do we assess the significance of departures from linearity? One of the simplest ways is to use polynomial regression.

The idea of polynomial regression is straightforward. As before, we have just one continuous explanatory variable, $x$, but we can fit higher powers of $x$, such as $x^2$ and $x^3$, to the model in addition to $x$ to explain curvature in the relationship between $y$ and $x$. It is useful to experiment with the kinds of curves that can be generated with very simple models. Even if we restrict ourselves to the inclusion of a quadratic term, $x^2$, there are many curves we can describe, depending upon the signs of the linear and quadratic terms:

```
par(mfrow=c(2,2))
x <- seq(0,10,0.1)
y1 <- 4 + 2 * x - 0.1 * x^2
y2 <- 4 + 2 * x - 0.2 * x^2
y3 <- 12 - 4 * x + 0.35 * x^2
y4 <- 4 + 0.5 * x + 0.1 * x^2
plot(x,y1,type="l",ylim=c(0,15),ylab="y",col="red")
plot(x,y2,type="l",ylim=c(0,15),ylab="y",col="red")
plot(x,y3,type="l",ylim=c(0,15),ylab="y",col="red")
plot(x,y4,type="l",ylim=c(0,15),ylab="y",col="red")
```



In the top left-hand panel, there is a curve with positive but declining slope, with no hint of a hump ($y = 4 + 2x - 0.1x^2$). The top right-hand graph shows a curve with a clear maximum ($y = 4 + 2x - 0.2x^2$), and at bottom left we have a curve with a clear minimum ($y = 12 - 4x + 0.35x^2$). The bottom right-hand curve shows a positive association between $y$ and $x$ with the slope increasing as $x$ increases ($y = 4 + 0.5x + 0.1x^2$). So you can see that a simple quadratic model with three parameters (an intercept, a slope for $x$, and a slope for $x^2$) is capable of describing a wide range of functional relationships between $y$ and $x$. It is very important to understand that the quadratic model *describes* the relationship between $y$ and $x$; it does not pretend to *explain* the mechanistic (or causal) relationship between $y$ and $x$.

We can see how polynomial regression works by analysing an example where diminishing returns in output (`yv`) are suspected as inputs (`xv`) are increased:

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

```
[1]  "xv"  "yv"
```

We begin by fitting a straight-line model to the data:

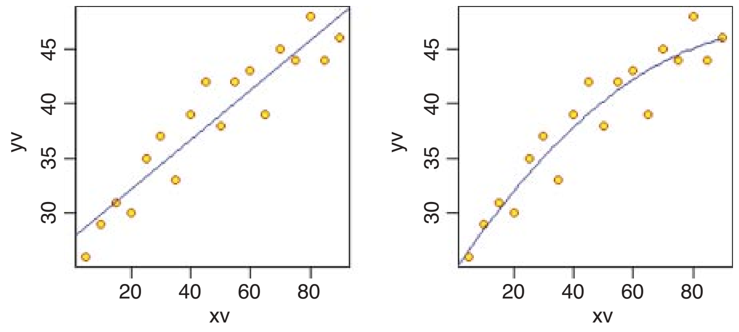
```
windows(7,4)
par(mfrow=c(1,2))
model1 <- lm(yv~xv)
plot(xv,yv,pch=21,col="brown",bg="yellow")
abline(model1,col="navy")
```

This is not a bad fit to the data ($r^2 = 0.8725$), but there is a distinct hint of curvature (diminishing returns in this case). Next, we fit a second explanatory variable which is the square of the $x$ value (the so-called 'quadratic term'). Note the use of `I` (for 'as is') in the model formula; see p. 210.

```
model2 <- lm(yv~xv+I(xv^2))
```

Now we use `model2` to predict the fitted values for a smooth range of $x$ values between 0 and 90:

```
plot(xv,yv,pch=21,col="brown",bg="yellow")
x <- 0:90
y <- predict(model2,list(xv=x))
lines(x,y,col="navy")
```



This looks like a slightly better fit than the straight line ($r^2 = 0.9046$), but we shall choose between the two models on the basis of an $F$ test using `anova`:

```
anova(model1,model2)
```

```
Analysis of Variance Table

Model 1: yv ~ xv
Model 2: yv ~ xv + I(xv^2)
```

```
  Res.Df     RSS Df Sum of Sq       F Pr(>F)
1     16 91.057
2     15 68.143  1    22.915 5.0441 0.0402 *
```

The more complicated curved model is a significant improvement over the linear model ($p = 0.04$) so we accept that there is evidence of curvature in these data.

## 10.4 Fitting a mechanistic model to data

Rather than fitting some arbitrary model for curvature (as above, with a quadratic term for inputs), we sometimes have a mechanistic model relating the value of the response variable to the explanatory variable (e.g. a mathematical model of a physical process). In the following example we are interested in the decay of organic material in soil, and our mechanistic model is based on the assumption that the fraction of dry matter lost per year is a constant. This leads to a two-parameter model of exponential decay in which the amount of material remaining ($y$) is a function of time ($t$):

$$y = y_0 e^{-bt}.$$

Here $y_0$ is the initial dry mass (at time $t = 0$) and $b$ is the decay rate (the parameter we want to estimate by linear regression). Taking logs of both sides, we get

$$\log(y) = \log(y_0) - bt.$$

Now you can see that we can estimate the parameter of interest, $b$, as the slope of a linear regression of log($y$) on $t$ (i.e. we log-transform the $y$ axis but not the $x$ axis) and the value of $y_0$ as the antilog of the intercept.

We begin by plotting our data:

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

```
[1]  "time"  "amount"
```

```
attach(data)
plot(time,amount,pch=21,col="blue",bg="brown")
abline(lm(amount~time),col="green")
```

The curvature in the relationship is clearly evident from the poor fit of the straight-line (green) model through the scatterplot (there are groups of positive residuals for low and high values of time, and a large group of negative residuals at intermediate times). Now we fit the linear model of `log(amount)` as a function of `time`:

```
model <- lm(log(amount)~time)
summary(model)
```

```
Call:
lm(formula = log(amount) ~ time)

Residuals:
  Min      1Q  Median      3Q     Max
-0.5935 -0.2043  0.0067  0.2198  0.6297
```

```
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.547386   0.100295   45.34  < 2e-16 ***
time         -0.068528   0.005743  -11.93 1.04e-12 ***

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 DF,  p-value: 1.038e-12
```

Thus, the slope is – 0.068 528 and $y_0$ is the antilog of the intercept: $y_0 = \exp(4.547\,386) = 94.385\,36$. The equation can now be parameterized (with standard errors in brackets) as

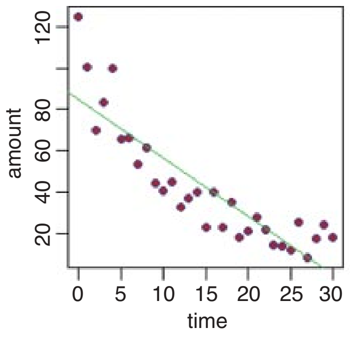
$$y = e^{4.5474(\pm 0.1003) - 0.0685(\pm 0.00574)t},$$

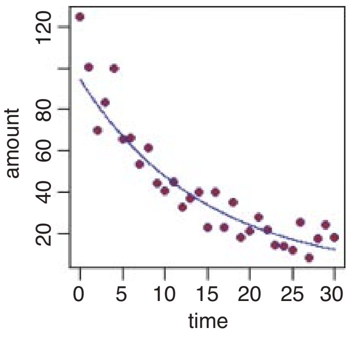
or written in its original form, without the uncertainty estimates, as

$$y = 94.385^{-0.0685t},$$

and we can draw the fitted line through the data, remembering to take the antilogs of the predicted values (the model predicts `log(amount)` and we want `amount`), like this:

```
ts <- seq(0,30,0.02)
left <- exp(predict(model,list(time=ts)))
plot(time,amount,pch=21,col="blue",bg="brown")
lines(ts,left,col="blue")
```





## 10.5 Linear regression after transformation

Many mathematical functions that are non-linear in their parameters can be linearized by transformation (see p. 258). The most frequent transformations (in order of frequency of use), are logarithms, antilogs and reciprocals. Here is an example of linear regression associated with a power law (p. 261):

$$y = ax^b.$$

This is a two-parameter function, where the parameter $a$ describes the slope of the function for low values of $x$ and $b$ is the shape parameter. For $b = 0$ we have a horizontal relationship $y = a$, for $b = 1$ we have a straight line through the origin $y = ax$ with slope $a$, for $b > 1$ the slope is positive but increases with increasing $x$, for

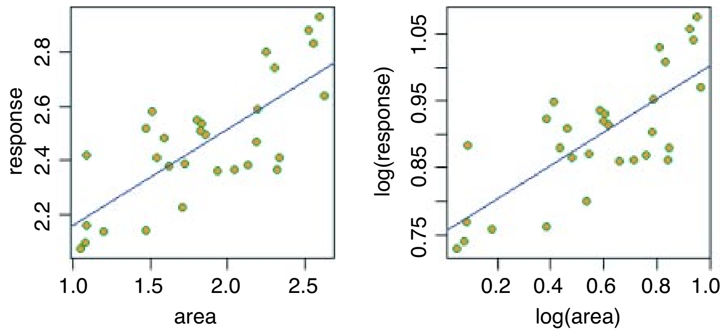
$0 < b < 1$ the slope is positive but decreases with increasing $x$, while for $b < 0$ (negative powers) the curve is a negative hyperbola that is asymptotic to infinity as $x$ approaches 0 and asymptotic to zero as $x$ approaches infinity.

Let us load a new dataframe and plot the data:

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

```
[1] "area"     "response"
```

```
plot(area,response,pch=21,col="green",bg="orange")
abline(lm(response~area),col="blue")
plot(log(area),log(response),pch=21,col="green",bg="orange")
abline(lm(log(response)~log(area)),col="blue")
```



The two plots look very similar (this is not always the case), but we need to compare the two models:

```
model1 <- lm(response~area)
```

```
model2 <- lm(log(response)~log(area))
```

```
summary(model2)
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.75378    0.02613  28.843  < 2e-16 ***
log(area)    0.24818    0.04083   6.079 1.48e-06 ***
```

We need to do a $t$ test to see whether the estimated shape parameter, $b = 0.248\,18$, is significantly less than $b = 1$ (a straight line):

$$t = \frac{|0.24818 - 1.0|}{0.04083} = 18.41342.$$

This is highly significant ($p < 0.0001$), so we conclude that there is a non-linear relationship between `response` and `area`.

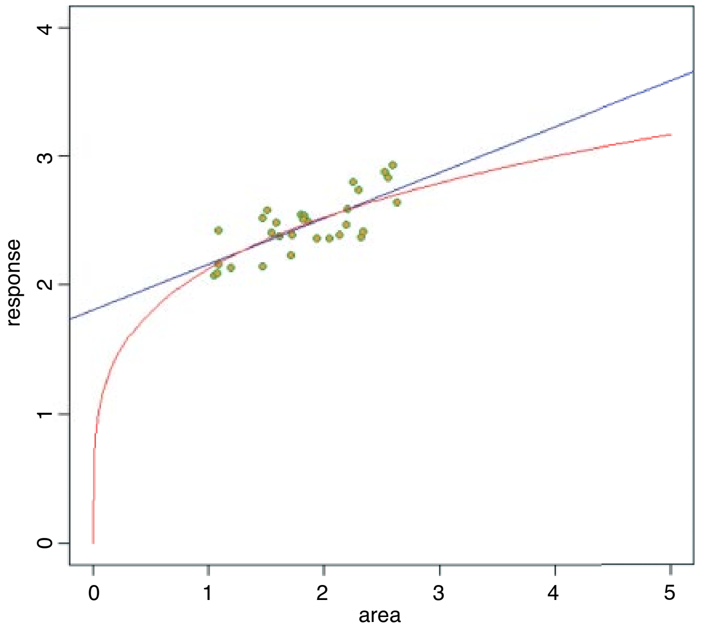
Let us get a visual comparison of the two models:

```
windows(7,7)
plot(area,response,pch=21,col="green",bg="orange")
```

```
abline(lm(response~area),col="blue")
xv <- seq(1,2.7,0.01)
yv <- exp(0.75378)*xv^0.24818
lines(xv,yv,col="red")
```

This is a nice example of the distinction between statistical significance and scientific importance. The power law transformation shows that the curvature is highly significant ($b < 1$ with $p$<0.0001) but over the range of the data, and given the high variance in $y$, the effect of the curvature is very small; the straight line and the power function are very close to one another. However, the choice of model makes an enormous difference if the function is to be used for prediction. Here are the two functions over an extended range of values for $x$:

```
plot(area,response,xlim=c(0,5),ylim=c(0,4),pch=21,col="green",bg="orange")
abline(lm(response~area),col="blue")
xv <- seq(0,5,0.01)
yv <- exp(0.75378)*xv^0.24818
lines(xv,yv,col="red")
```



The moral is clear: you need to extremely careful when using regression models for prediction. If you know that `response` *must* be zero when `area` is zero (the graph has to pass through the origin) then obviously the power function is likely to be better for extrapolation to the left of the data. But if we have no information on non-linearity other than that contained within the data, then parsimony suggests that errors will be smaller using the simpler, linear model for prediction. Both models are equally good at describing the data (the linear model has $r^2 = 0.574$ and the power law model has $r^2 = 0.569$), but extrapolation beyond the range of the data is always fraught with difficulties. Targeted collection of new data for `response` at values of `area` close to 0 and close to 5 might resolve the issue.

## 10.6 Prediction following regression

The popular notion is that predicting the future is impossible, and that attempts at prediction are nothing more that crystal-gazing. However, all branches of applied science rely upon prediction. These predictions may be based on extensive experimentation (as in engineering or agriculture) or they may be based on detailed, long-term observations (as in astronomy or meteorology). In all cases, however, the main issue to be confronted in prediction is how to deal with uncertainty: uncertainty about the suitability of the fitted model, uncertainty about the representativeness of the data used to parameterize the model, and uncertainty about future conditions (in particular, uncertainty about the future values of the explanatory variables).

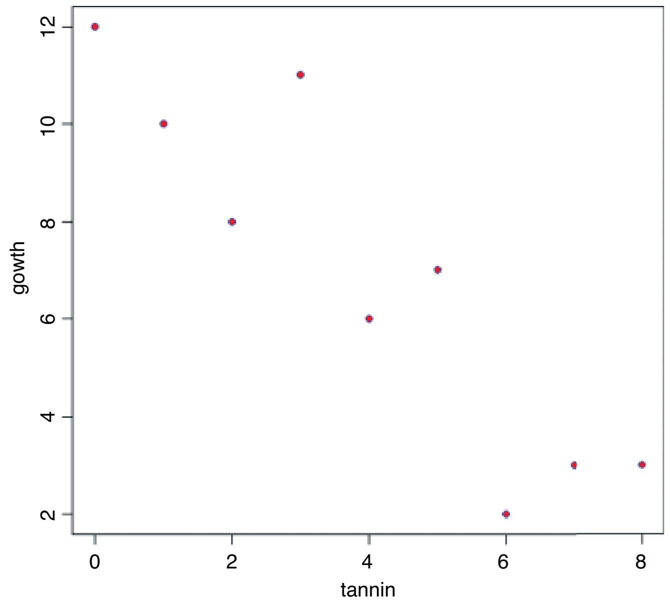
There are two kinds of prediction, and these are subject to very different levels of uncertainty. **Interpolation**, which is prediction *within* the measured range of the data, can often be very accurate and is not greatly affected by model choice. **Extrapolation**, which is prediction *beyond* the measured range of the data, is far more problematical, and model choice is a major issue. Choice of the wrong model can lead to wildly different predictions (see p. 471).

Here are two kinds of plots involved in prediction following regression: the first illustrates uncertainty in the parameter estimates; the second indicates uncertainty about predicted values of the response. We continue with the tannin example:

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

```
[1]  "growth"  "tannin"
```

```
plot(tannin,growth,pch=21,col="blue",bg="red")
```



```
model <- lm(growth~tannin)
abline(model,col="blue")
```

The first plot is intended to show the uncertainty associated with the estimate of the slope. It is easy to extract the slope from the vector of coefficients:

```
coef(model)[2]

   tannin
-1.216667
```

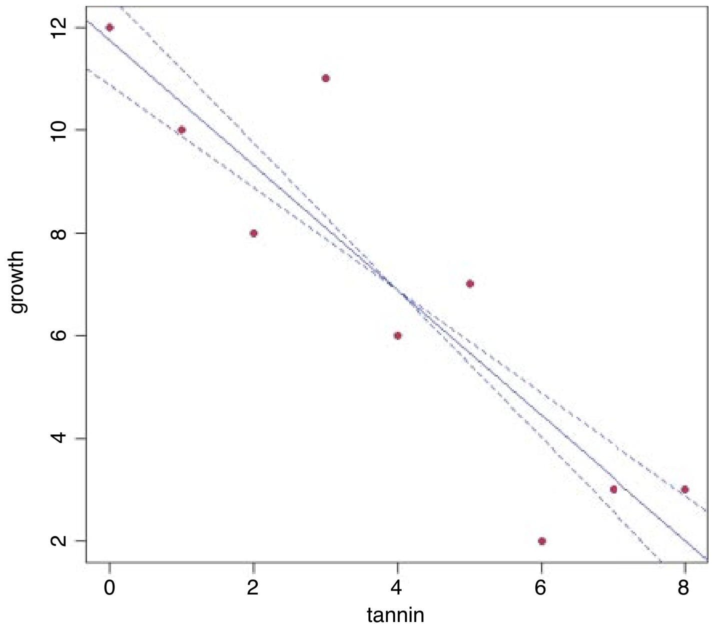
The standard error of the slope is a little trickier to find. After some experimentation, you will discover that it is in the fourth element of the list that is `summary(model)`:

```
summary(model)[[4]][4]

[1]  0.2186115
```

Here is a function that will add dotted lines showing two extra regression lines to our existing plot – the estimated slope plus and minus one standard error of the slope:

```
se.lines <- function(model){
      b1 <- coef(model)[2]+ summary(model)[[4]][4]
      b2 <- coef(model)[2]- summary(model)[[4]][4]
      xm <- sapply(model[[12]][2],mean)
      ym <- sapply(model[[12]][1],mean)
      a1 <- ym-b1*xm
      a2 <- ym-b2*xm
            abline(a1,b1,lty=2,col="blue")
            abline(a2,b2,lty=2,col="blue")
}
se.lines(model)
```



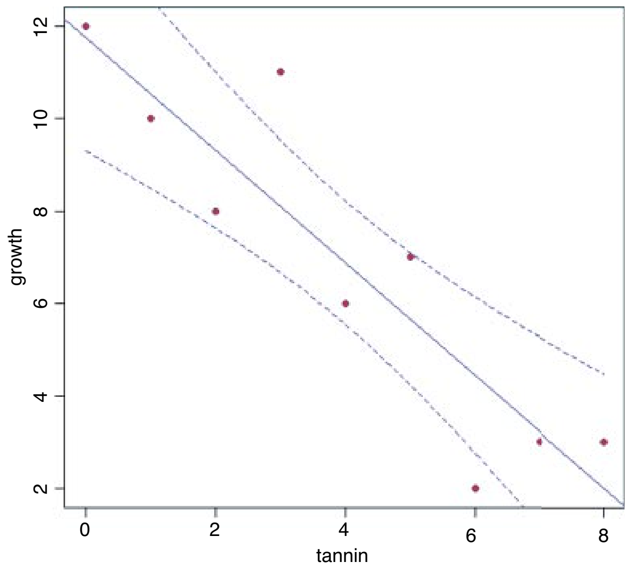
More often, however, we are interested in the uncertainty about predicted values (rather than uncertainty of parameter estimates, as above). We might want to draw the 95% confidence intervals associated with predictions of *y* at different values of *x*. As we saw on p. 460, uncertainty increases with the *square* of the difference between the mean value of *x* and the value of *x* at which the value of *y* is to be predicted. Before we can draw these lines we need to calculate a vector of *x* values; you need 100 or so values to make an attractively smooth curve. Then we need the value of Student's *t* (p. 122). Finally, we multiply Student's *t* by the standard error of the predicted value of *y* (p. 462) to get the confidence interval. This is added to the fitted values of *y* to get the upper limit and subtracted from the fitted values of *y* to get the lower limit. Here is the function:

```
ci.lines <- function(model){
      xm <- sapply(model[[12]][2],mean)
      n <- sapply(model[[12]][2],length)
      ssx <-  sum(model[[12]][2]^2)-sum(model[[12]][2])^2/n
      s.t <-  qt(0.975,(n-2))
xv <- seq(min(model[[12]][2]),max(model[[12]][2]),length=100)
yv <- coef(model)[1]+coef(model)[2]*xv
se <- sqrt(summary(model)[[6]]^2*(1/n+(xv-xm)^2/ssx))
ci <- s.t*se
      uyv <- yv+ci
      lyv <- yv-ci
      lines(xv,uyv,lty=2,col="blue")
      lines(xv,lyv,lty=2,col="blue")
}
```

We replot the linear regression, then overlay the confidence intervals (Box 10.4):

```
plot(tannin,growth,pch=21,col="blue",bg="red")
abline(model, col="blue")

ci.lines(model)
```



This draws attention to the points at `tannin` = 3 and `tannin` = 6 that fall outside the 95% confidence limits of our fitted values.

You can speed up this procedure by using the built-in ability to generate confidence intervals coupled with `matlines`. The familiar 95% confidence intervals are `int="c"`, while prediction intervals (fitted values plus or minus 2 standard deviations) are `int="p"`.

```
plot(tannin,growth,pch=16,ylim=c(0,15))
model <-lm(growth~tannin)
```

As usual, start by generating a series of *x* values for generating the curves, then create the scatterplot. The *y* values are predicted from the model, specifying `int="c"`, then `matlines` is used to draw the regression line (solid) and the two confidence intervals (dotted), producing exactly the same graph as our last plot (above) without writing a special function:

```
xv <- seq(0,8,0.1)
yv <- predict(model,list(tannin=xv),int="c")
matlines(xv,yv,lty=c(1,2,2),col="black")
```

A similar plot can be obtained using the `effects` library (see p. 968).

## 10.7 Testing for lack of fit in a regression

The unreliability estimates of the parameters explained in Boxes 10.2 and 10.3 draw attention to the important issues in optimizing the efficiency of regression designs. We want to make the error variance as small as possible (as always), but in addition, we want to make *SSX* as large as possible, by placing as many points as possible at the extreme ends of the *x* axis. Efficient regression designs allow for:

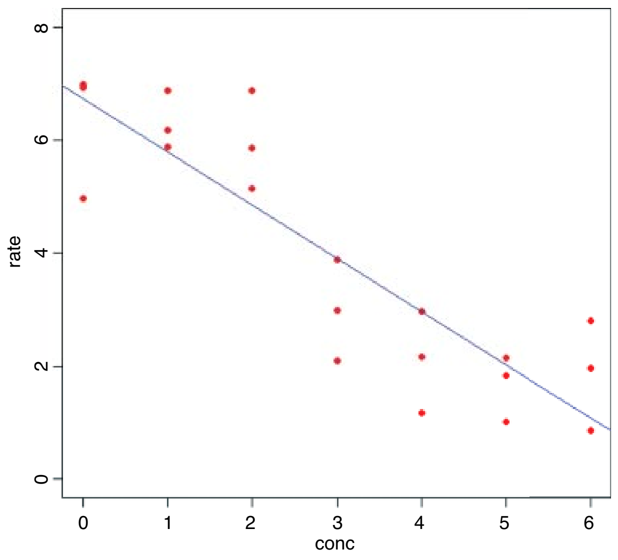
- replication of least some of the levels of *x*;
- a preponderance of replicates at the extremes (to maximize *SSX*);
- sufficient levels of *x* to allow testing for non-linearity;
- sufficient different values of *x* to allow accurate location of thresholds.

Here is an example where replication allows estimation of pure sampling error, and this in turn allows a test of the significance of the data's departure from linearity. As the concentration of an inhibitor is increased, the reaction rate declines:

```
data <- read.delim("c:\\temp\\lackoffit.txt")
attach(data)
names(data)
```

```
[1]  "conc"  "rate"
```

```
plot(conc,jitter(rate),pch=16,col="red",ylim=c(0,8),ylab="rate")
abline(lm(rate~conc),col="blue")
```



The linear regression does not look too bad, and the slope is highly significantly different from zero:

```
model.reg <- lm(rate~conc)
summary(model.reg)
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   6.7262     0.4559  14.755 7.35e-12 ***
conc         -0.9405     0.1264  -7.439 4.85e-07 ***
Residual standard error: 1.159 on 19 degrees of freedom
Multiple R-squared: 0.7444,     Adjusted R-squared: 0.7309
F-statistic: 55.33 on 1 and 19 DF,  p-value: 4.853e-07
```

Because there is replication at each level of $x$ we can do something extra, compared with a typical regression analysis. We can estimate what is called the **pure error variance**. This is the sum of the squares of the differences between the $y$ values and the *mean* values of $y$ for the relevant level of $x$. This should sound somewhat familiar. In fact, it is the definition of *SSE* from a one-way analysis of variance (see p. 501). By creating a factor to represent the seven levels of $x$, we can estimate this *SSE* simply by fitting a one-way ANOVA:

```
fac.conc <- factor(conc)
model.aov <- aov(rate~fac.conc)
summary(model.aov)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)
fac.conc     6  87.81  14.635   17.07 1.05e-05 ***
Residuals   14  12.00   0.857
```

This shows that the pure error sum of squares is 12.0 on 14 degrees of freedom (three replicates, and hence 2 d.f., at each of seven levels of $x$). See if you can figure out why this sum of squares is less than the observed in the `model.reg` regression (25.512). If the means from the seven different concentrations all fell exactly on the same straight line then the two sums of squares would be identical. It is the fact that the means do *not* fall on the regression line that causes the difference. The difference between these two sums of squares

(25.512 – 12.9 = 13.512) is a measure of lack of fit of the `rate` data to the straight-line model. We can compare the two models to see if they differ in their explanatory powers:

```
anova(model.reg,model.aov)
```

```
Analysis of Variance Table
Model 1: rate ~ conc
Model 2: rate ~ fac.conc

  Res.Df    RSS Df Sum of Sq      F  Pr(>F)
1     19 25.512
2     14 12.000  5    13.512 3.1528 0.04106 *
```

A single ANOVA table showing the lack-of-fit sum of squares on a separate line is obtained by fitting both the regression line (1 d.f.) and the lack of fit (5 d.f.) in the same model:

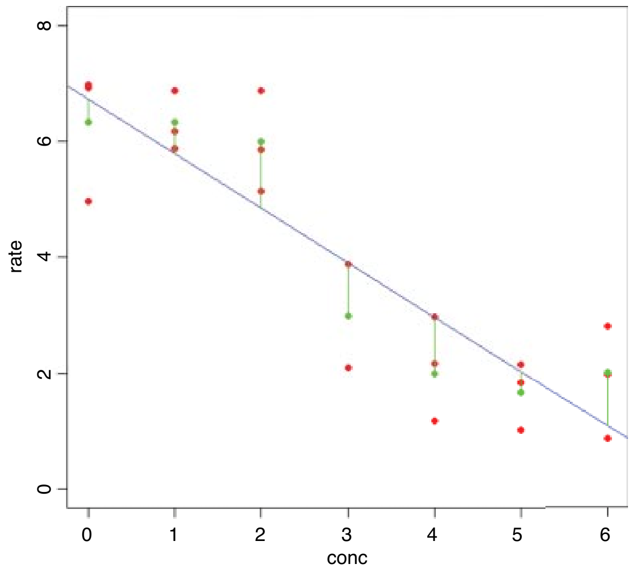
```
anova(lm(rate~conc+fac.conc))
```

```
Analysis of Variance Table

Response: rate
          Df Sum Sq Mean Sq F value    Pr(>F)
conc       1 74.298  74.298 86.6806 2.247e-07 ***
fac.conc   5 13.512   2.702  3.1528   0.04106 *
Residuals 14 12.000   0.857
```

To get a visual impression of this lack of fit we can draw vertical lines from the mean values to the fitted values of the linear regression for each level of *x*:

```
my <- as.vector(tapply(rate,fac.conc,mean))
for (i in 0:6)
 lines(c(i,i),c(my[i+1],predict(model.reg,list(conc=0:6))[i+1]),col="green")
points(0:6,my,pch=16,col="green")
```



This significant lack of fit indicates that the straight-line model is *not* an adequate description of these data ($p < 0.05$). A negative S-shaped function is likely to fit the data better (see p. 301).

There is an R package called `lmtest` on CRAN, which is full of tests for linear models.

## 10.8 Bootstrap with regression

An alternative to estimating confidence intervals on the regression parameters from the pooled error variance in the ANOVA table (p. 459) is to use bootstrapping. There are two ways of doing this:

- sample cases with replacement, so that some points are left off the graph while others appear more than once in the dataframe;
- calculate the residuals from the fitted regression model, and randomize which fitted $y$ values get which residuals.

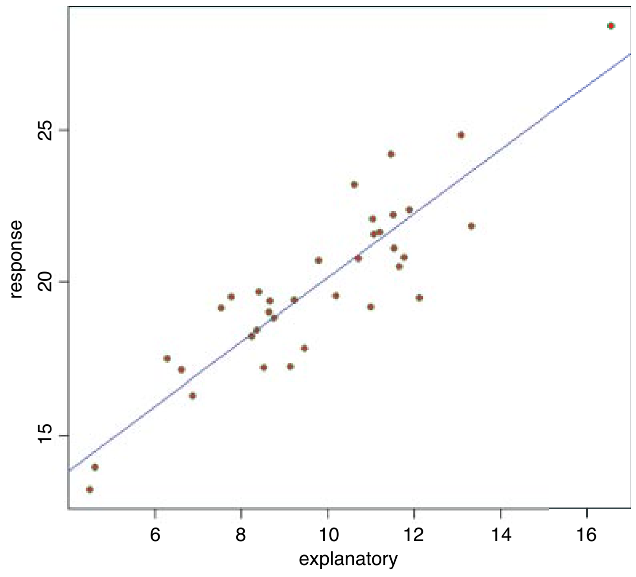
In both cases, the randomization is carried out many times, the model fitted and the parameters estimated. The confidence interval is obtained from the quantiles of the distribution of parameter values (see p. 41).

The following dataframe contains a response variable (profit from the cultivation of a crop of carrots for a supermarket) and a single explanatory variable (the cost of inputs, including fertilizers, pesticides, energy and labour):

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

```
[1]  "explanatory"  "response"
```

```
plot(explanatory,response,pch=21,col="green",bg="red")
model <- lm(response~explanatory)
abline(model,col="blue")
```



The response is a reasonably linear function of the explanatory variable, but the variance in the response is quite large. For instance, when the explanatory variable is about 12, the response variable ranges between less than 20 and more than 24.

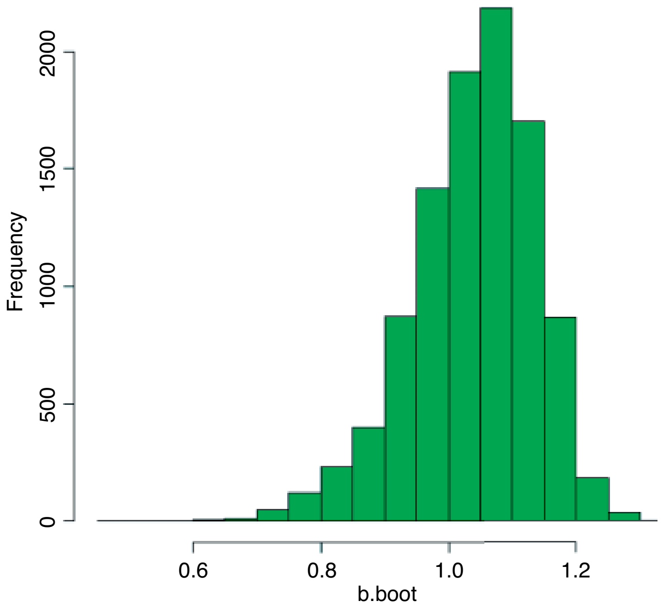
```
model
```

```
Coefficients:
(Intercept)  explanatory
      9.630        1.051
```

Theory suggests that the slope should be 1.0, and our estimated slope is very close to this (1.051). We want to establish a 95% confidence interval on the estimate. Here is a home-made bootstrap which resamples the data points 10 000 times and gives a bootstrapped estimate of the slope:

```
b.boot <- numeric(10000)

for (i in 1:10000){
      indices <- sample(1:35,replace=T)
      xv <- explanatory[indices]
      yv <- response[indices]
      model <- lm(yv~xv)
      b.boot[i] <- coef(model)[2]
}
hist(b.boot,main="",col="green")
```



Here is the 95% interval for the bootstrapped estimate of the slope:

```
quantile(b.boot,c(0.025,0.975))
```

```
     2.5%     97.5%
0.8137637 1.1964226
```

Evidently, the bootstrapped data provide no support for the hypothesis that the slope is significantly greater than 1.0.

We now repeat the exercise, using the `boot` function from the `boot` package:

```
library(boot)
```

The first step is to write what is known as the ‘statistic’ function. This shows `boot` how to calculate the statistic we want from the resampled data (the slope in this case). The resampling of the data is achieved by a subscript provided by `boot` (here called `index`). The point is that every time the model is fitted within the bootstrap it uses a different data set (`yv` and `xv`): we need to describe how these data are constructed and how they are to be used in the model fitting:

```
reg.boot <- function(regdat, index){
      xv <- explanatory[index]
      yv <- response[index]
      model <- lm(yv~xv)
      coef(model)
}
```

Now we can run the `boot` function, then extract the intervals with the `boot.ci` function:

```
reg.model <- boot(regdat,reg.boot,R=10000)
boot.ci(reg.model,index=2)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = reg.model, index = 2)

Intervals :
Level      Normal               Basic
95%   ( 0.870,  1.254 )   ( 0.903,  1.287 )

Level     Percentile             BCa
95%   ( 0.815,  1.198 )   ( 0.821,  1.202 )

Calculations and Intervals on Original Scale
Warning message:
In boot.ci(reg.model, index = 2) :
  bootstrap variances needed for studentized intervals
```

All the intervals are reasonably similar: statisticians typically prefer the bias-corrected, accelerated (BCa) intervals. These indicate that if we were to repeat the data-collection exercise we can be 95% confident that the regression slope for those new data would be between 0.821 and 1.202.

The other way of bootstrapping with a model is to randomize the allocation of the residuals to fitted *y* values estimated from the original regression model. We start by calculating the residuals and the fitted values:

```
model <- lm(response~explanatory)
fit <- fitted(model)
res <- resid(model)
```

What we intend to do is to randomize which of the `res` values is added to the `fit` values to get a reconstructed response variable, *y*, which we regress as a function of the original explanatory variable. Here is the statistic function to do this:

```
residual.boot <- function(res, index){
y <- fit+res[index]
model <- lm(y~explanatory)
coef(model) }
```

Note that the data passed to the statistic function are `res` in this case (rather than the original dataframe `regdat` as in the first example, above). Now use the `boot` function and the `boot.ci` function to obtain the 95% confidence intervals on the slope (this is `index=2`; the intercept is `index=1`):

```
res.model <- boot(res,residual.boot,R=10000)
boot.ci(res.model,index=2)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = res.model, index = 2)

Intervals :
Level      Normal              Basic
95%   ( 0.878,  1.224 )   ( 0.884,  1.225 )

Level     Percentile            BCa
95%   ( 0.876,  1.218 )   ( 0.872,  1.215 )

Calculations and Intervals on Original Scale
Warning message:
In boot.ci(res.model, index = 2) :
  bootstrap variances needed for studentized intervals
```

The BCa from randomizing the residuals is from 0.872 to 1.215, while from selecting random *x* and *y* points with replacement it was from 0.821 to 1.202 (above). The two rather different approaches to bootstrapping produce reassuringly similar estimates of the same parameter.

## 10.9 Jackknife with regression

A second alternative to estimating confidence intervals on regression parameters is to **jackknife** the data. Each point in the data set is left out, one at a time, and the parameter of interest is re-estimated. The `regdat` dataframe (above) has `length(response)` data points:

```
names(regdat)
```

```
[1] "explanatory"  "response"
```

```
length(response)
```

```
[1]  35
```

We create a vector to contain the 35 different estimates of the slope:

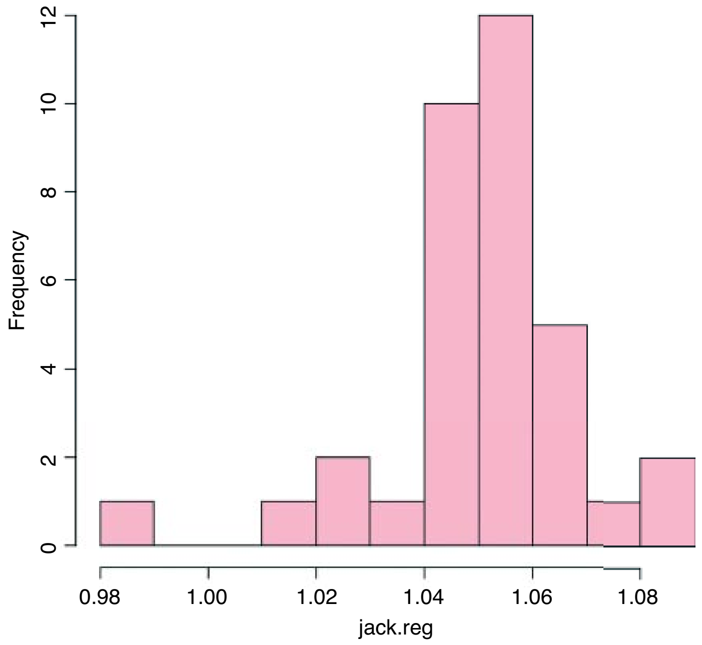
```
jack.reg <- numeric(35)
```

Now carry out the regression 35 times, leaving out a different *x*, *y* pair each time:

```
for (i in 1:35) {
model <- lm(response[-i]~explanatory[-i])
jack.reg[i] <- coef(model)[2] }
```

Here is a histogram of the different estimates of the slope of the regression:

```
hist(jack.reg,main="",col="pink")
```



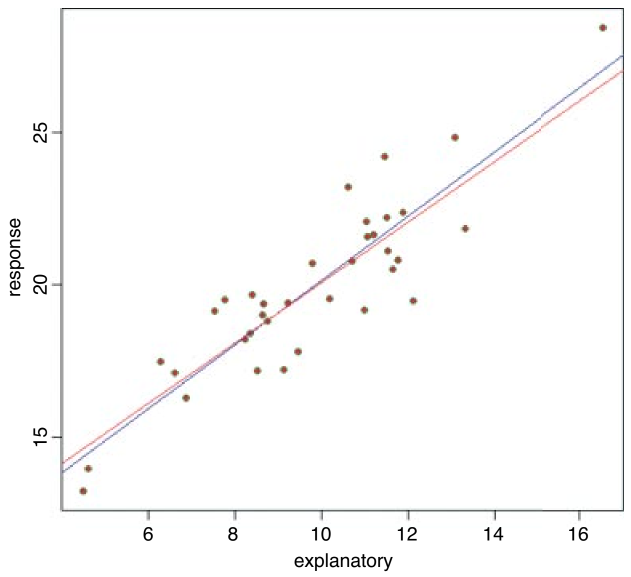
As you can see, the distribution is strongly skew to the left. The quantiles of `jack.reg` are not particularly informative because the sample is so small (just 35). However, the jackknife does draw attention to one particularly influential point (the extreme left-hand bar) which, when omitted from the dataframe, causes the estimated slope to fall below 1.0. We say the point is **influential** because it is the only one of the 35 points whose omission causes the estimated slope to fall below 1.0. But which data point is this? We extract Cook's distance `$infmat[,5]` from the influence matrix from the model (`influence.measures(model)$infmat`) and ask which data point has the maximum value of this influence measure:

```
model <- lm(response~explanatory)
which(influence.measures(model)$infmat[,5]
     == max(influence.measures(model)$infmat[,5]))
```

```
22
```

Now we can draw regression lines for the full data set (blue line) and for the model with the influential point number 22 omitted (red line) to see just how influential (or not) this point really is for the location of the line:

```
plot(explanatory,response,pch=21,col="green",bg="red")
abline(model,col="blue")
abline(lm(response[-22]~explanatory[-22]),col="red")
```

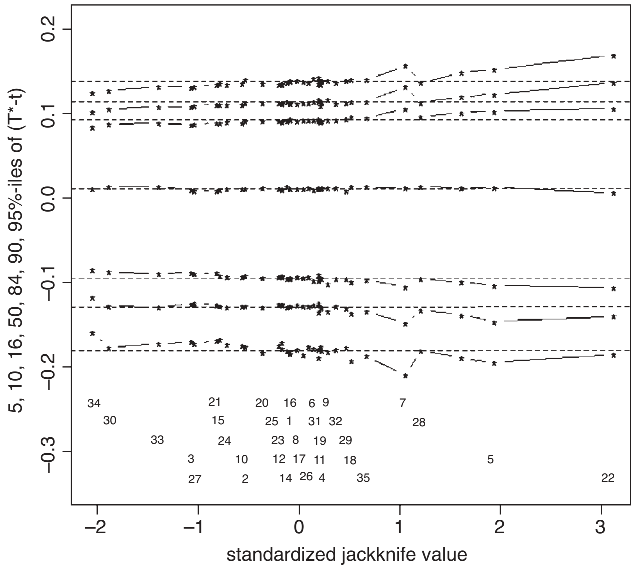


Neither model describes at all well the location of the response for the two lowest values of the explanatory variable (and the fit is worse with the most influential point removed).

## 10.10 Jackknife after bootstrap

The `jack.after.boot` function calculates the jackknife influence values from a bootstrap output object, and plots the corresponding jackknife-after-bootstrap plot. We illustrate its use with the `boot` object calculated earlier called `reg.model`. We are interested in the slope, which is `index=2`:

```
jack.after.boot(reg.model,index=2)
```



The centred jackknife quantiles for each observation are estimated from those bootstrap samples in which *the particular observation did not appear*. These are then plotted against the influence values. From the top downwards, the horizontal dotted lines show the 95th, 90th, 84th, 50th, 16th, 10th and 5th percentiles. The numbers at the bottom identify the 35 points by their index values within regdat. Again, the influence of point no. 22 shows up clearly (this time on the right-hand side), indicating that it has a strong positive influence on the slope, and the two left-hand outliers are identified as points nos 34 and 30.

## 10.11 Serial correlation in the residuals

The Durbin–Watson function is used for testing whether there is autocorrelation in the residuals from a linear model or a generalized linear model, and is implemented as part of the car package (see Fox, 2002):

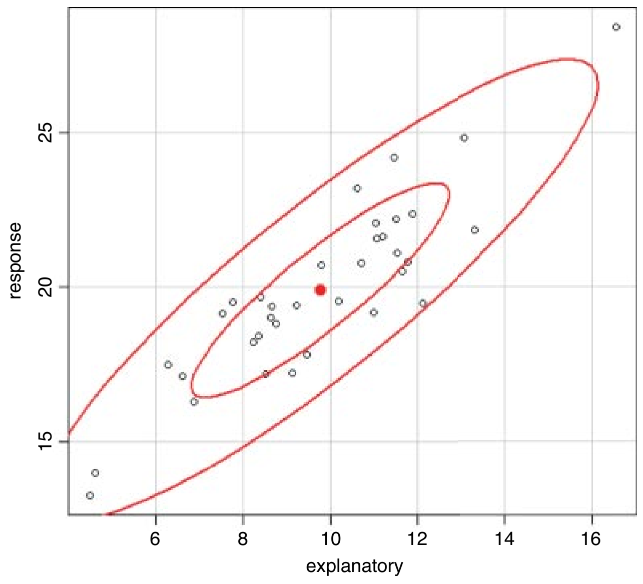
```
library("car")
durbinWatsonTest(model)
```

```
lag Autocorrelation D-W Statistic p-value
  1      -0.07946739      2.049899   0.874
Alternative hypothesis: rho != 0
```

There is no evidence of serial correlation in these residuals ($p = 0.874$).

The car package also contains functions for drawing ellipses, including data ellipses, and confidence ellipses for linear and generalized linear models. Here is the dataEllipse function for the present example: by default, the ellipses are drawn at 50% and 90%:

```
dataEllipse(explanatory,response)
```



## 10.12 Piecewise regression

This kind of regression fits different functions over different ranges of the explanatory variable. For example, it might fit different linear regressions to the left- and right-hand halves of a scatterplot. Two important questions arise in piecewise regression:

- how many segments to divide the line into;
- where to position the break points on the *x* axis.

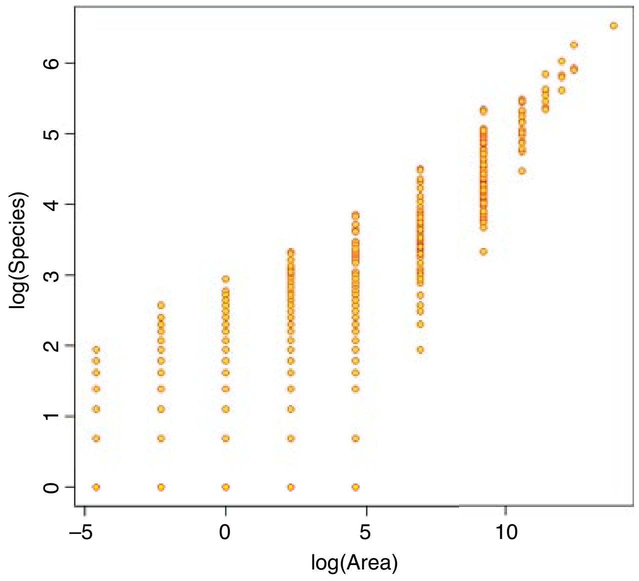
Suppose we want to do the simplest piecewise regression, using just two linear segments. Where do we break up the *x* values? A simple, pragmatic view is to divide the *x* values at the point where the piecewise regression best fits the response variable. Let us take an example using a linear model where the response is the log of a count (the number of species recorded) and the explanatory variable is the log of the size of the area searched for the species:

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

```
[1]  "Species"  "Area"
```

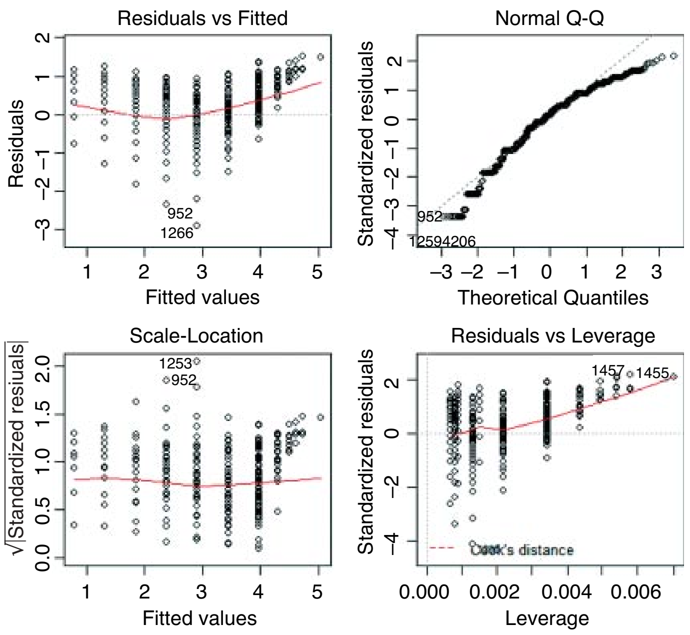
A quick scatterplot suggests that the relationship between `log(Species)` and `log(Area)` is not linear:

```
plot(log(Species)~log(Area),pch=21,col="red",bg="yellow")
```



The slope appears to be shallower at small scales than at large. The overall regression highlights this at the model-checking stage:

```
model1 <- lm(log(Species)~log(Area))
par(mfrow=c(2,2))
plot(model1)
```



The residuals are very strongly U-shaped (this plot should look like the sky at night) and the errors are profoundly non-normal (the top right-hand line should be straight).

If we are to use piecewise regression, then we need to work out how many straight-line segments to use and where to put the breaks. Visual inspection of the scatterplot suggests that two segments would be an improvement over a single straight line and that the break point should be about `log(Area)` = 5. The choice of break point is made more objective by choosing a range of values for the break point and selecting the break that produces the minimum deviance. We should have a minimum of two *x* values for each of the pieces of the regression, so the areas associated with the first and last breaks can be obtained by examination of the table of *x* values:

```
table(Area)
```

```
Area
0.01    0.1      1     10    100   1000  10000  40000  90000 160000 250000  1e+06
 346    345    259    239     88     67    110     18      7      4      3      1
```

The leftmost break could be between areas 0.1 and 1, and the rightmost between 160 000 and 250 000 (i.e. between indices 2 and 3 and 10 and 11)

Piecewise regression is extremely simple in R: we just include a logical statement as part of the model formula, with as many logical statements as we want straight-line segments in the fit. In the present example with two linear segments, the two logical statements are `Area<Break` to define the left-hand regression and `Area>=Break` to define the right-hand regression. We want to fit the model for all values of `Break` between 1 and 250 000, so we create a vector of breaks like this:

```
Break <- sort(unique(Area))[3:11]
```

Now we use a loop to fit the two-segment piecewise model nine times and to store the value of the residual standard error in a vector called `d`. This quantity is the sixth element of the list that is the model summary object, `d[i] <- summary(model)[[6]]:`

```
d <- numeric(9)
for (i in 1:9) {
model <-
lm(log(Species)~(Area<Break[i])*log(Area)+(Area>=Break[i])*log(Area))

d[i] <- summary(model)[[6]] }
```

A plot shows where the minimum value of d occurs:

```
windows(7,4)
par(mfrow=c(1,2))
plot(log(Break),d,typ="l",col="red")
```

Where exactly does the minimum of d occur? We use the which function for this:

```
Break[which(d==min(d))]

 [1] 100
```

The best piecewise regression will fit one line up to Area $= 100$ and a different line for Area $> 100$. The model formula looks like this:

```
model2 <- lm(log(Species)~log(Area)*(Area<100)+log(Area)*(Area>=100))
```

The piecewise regression is a massive improvement over the linear model:

```
anova(model1,model2)

Analysis of Variance Table

Model 1: log(Species) ~ log(Area)
Model 2: log(Species) ~ log(Area) * (Area < 100) + log(Area) * (Area >=
  100)
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1   1485 731.98
2   1483 631.36  2    100.62 118.17 < 2.2e-16 ***
```

The summary of the piecewise regression takes some getting used to. We have fitted two linear regressions, so there are four parameters. Like an analysis of covariance, the table of coefficients contains one slope and one intercept, along with one difference between slopes and one difference between intercepts. The table has six rows because of the intentional aliasing, which we contrived by providing zeros for the explanatory variables where the two logical expressions evaluate to FALSE:

```
summary(model2)

Coefficients: (2 not defined because of singularities)

                          Estimate Std. Error t value Pr(>|t|)
 (Intercept)               0.61682    0.13059   4.723 2.54e-06 ***
log(Area)                  0.41019    0.01655  24.787  < 2e-16 ***
Area < 100TRUE             1.07854    0.13246   8.143 8.12e-16 ***
Area >= 100TRUE                 NA         NA      NA       NA
log(Area):Area < 100TRUE  -0.25611    0.01816 -14.100  < 2e-16 ***
log(Area):Area >= 100TRUE       NA         NA      NA       NA

Residual standard error: 0.6525 on 1483 degrees of freedom
```

```
Multiple R-squared: 0.724,      Adjusted R-squared: 0.7235
F-statistic:  1297 on 3 and 1483 DF,  p-value: < 2.2e-16
```

The intercept is for the factor level that comes first in the alphabet: this is the right-hand part of the graph `log(Area):Area < 100FALSE` and the slope (`log(Area)`) is for this factor level too. The difference between the two intercepts is labelled `Area < 100TRUE`. The next row is labelled `Area >= 100TRUE` and contains `NA`s because there were no *x* values (they were all zeros because logical FALSE was coerced to numeric zero by the multiplication). The difference between two slopes is labelled `log(Area):Area < 100TRUE` while the last row labelled `log(Area):Area >= 100TRUE` contains `NA`s because there were no *x* values. We cannot use `abline`, because we want two separate lines through different parts of the scatterplot (not two lines across the whole plotting area; try it and see). To make plotting the two lines easier it is a good idea to calculate the two slopes and two intercepts in advance. The parameters are in the fourth element of the list that makes up `summary(model2)`. It is worth looking at this separately:

```
summary(model2)[[4]]
```

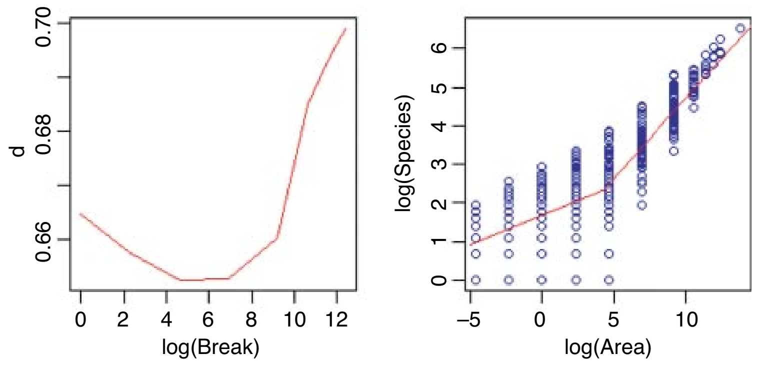
```
                           Estimate Std. Error    t value      Pr(>|t|)
(Intercept)               0.6168168 0.13058554   4.723469  2.537983e-06
log(Area)                 0.4101943 0.01654883  24.786903 9.081618e-114
Area < 100TRUE            1.0785395 0.13245572   8.142642  8.117406e-16
log(Area):Area < 100TRUE -0.2561147 0.01816373 -14.100333  1.834740e-42
```

Note that the two rows with `NA`s have been excluded. Using subscripts, we extract the parameter estimates and calculate two intercepts (`a1` and `a2`) and two slopes (`b1` and `b2`) for the left and right hand pieces of the regression:

```
a1 <- summary(model2)[[4]][1]+summary(model2)[[4]][3]
a2 <- summary(model2)[[4]][1]
b1 <- summary(model2)[[4]][2]+summary(model2)[[4]][4]
b2 <- summary(model2)[[4]][2]
```

Finally, we need to decide on the *x* values between which to draw the two lines. Inspection of the scatterplot indicates that –5 would be a good minimum value and 15 would be a good maximum. The break point (4.6 = log(100)) is the obvious point at which to stop the first line and start the second line.

```
plot(log(Area),log(Species),col="blue")
lines(c(-5,4.6),c(a1+b1*-5,a1+b1*4.6),col="red")
lines(c(4.6,15),c(a2+b2*4.6,a2+b2*15),col="red")
```



Of course, for spatial scales even smaller than studied here, the slope of the plot must go asymptotically to zero, because once the plot is so small that it can contain only one individual, making the plot even smaller is bound to contain that same individual (thus, species richness will be one for all subsequent smaller spatial scales).

## 10.13 Multiple regression

A multiple regression is a statistical model with two or more continuous explanatory variables. We contrast multiple regression with analysis of variance, where all the explanatory variables are categorical (Chapter 11) and analysis of covariance, where the explanatory variables are a mixture of continuous and categorical (Chapter 12). Multiple regressions models provide some of the most profound challenges faced by the analyst because of some crucial issues:

- over-fitting (we often have more explanatory variables than data points);
- parameter proliferation (we might want to fit parameters for curvature and interaction);
- correlation between explanatory variables (called collinearity);
- choice between contrasting models of roughly equal explanatory power.

The *principle of parsimony* (Occam's razor), discussed in Section 9.2, is again relevant here. It requires that the model should be as simple as possible. This means that the model should not contain any redundant parameters. Ideally, we achieve this by fitting a maximal model and 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.
- Amalgamate explanatory variables that have similar parameter values.

Of course, such simplifications must make good scientific sense, and must not lead to significant reductions in explanatory power. It is likely that many of the explanatory variables are correlated with each other, and so *the order in which variables are deleted from the model* will influence the explanatory power attributed to them. The thing to remember about multiple regression is that, in principle, there is no end to it. The number of combinations of interaction terms and curvature terms is endless. There are some simple rules (like parsimony) and some automated functions (like `step`) to help. But, in principle, you could spend a very great deal of time in modelling a single dataframe. There are no hard-and-fast rules about the best way to proceed, but we shall typically carry out simplification of a complex model by stepwise deletion: non-significant terms are left out, and significant terms are added back (see Chapter 9).

At the data inspection stage, there are many more kinds of plots we could do:

- Plot the response against each of the explanatory variables separately.
- Plot the explanatory variables against one another (e.g. `pairs`; see Section 10.13.1).
- Plot the response against pairs of explanatory variables in three-dimensional plots.

- Plot the response against explanatory variables for different combinations of other explanatory variables (e.g. conditioning plots, `coplot`; see p. 236).
- Fit non-parametric smoothing functions (e.g. using generalized additive models, to look for evidence of curvature).
- Fit tree models to investigate whether interaction effects are simple or complex.

### 10.13.1 The multiple regression model

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 overparameterization.

The assumptions about the response variable are the same as with simple linear regression: the errors are normally distributed, the errors are confined to the response variable, and the variance is constant. The explanatory variables are assumed to be measured without error. The model for a multiple regression with two explanatory variables ($x_1$ and $x_2$) looks like this:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \varepsilon_i .$$

The $i$th data point, $y_i$, is determined by the levels of the two continuous explanatory variables $x_{1i}$ and $x_{2i}$, by the model's three parameters (the intercept $\beta_0$ and the two slopes $\beta_1$ and $\beta_2$), and by the residual $\varepsilon_i$ of point $i$ from the fitted surface. For each of the $i$ rows of the dataframe, there are $k + 1$ parameters, $\beta_j$, so that

$$y_i = \sum_{j=0}^{k} \beta_j x_{ji} + \varepsilon_i ,$$

where $x_{0i} = 1$.

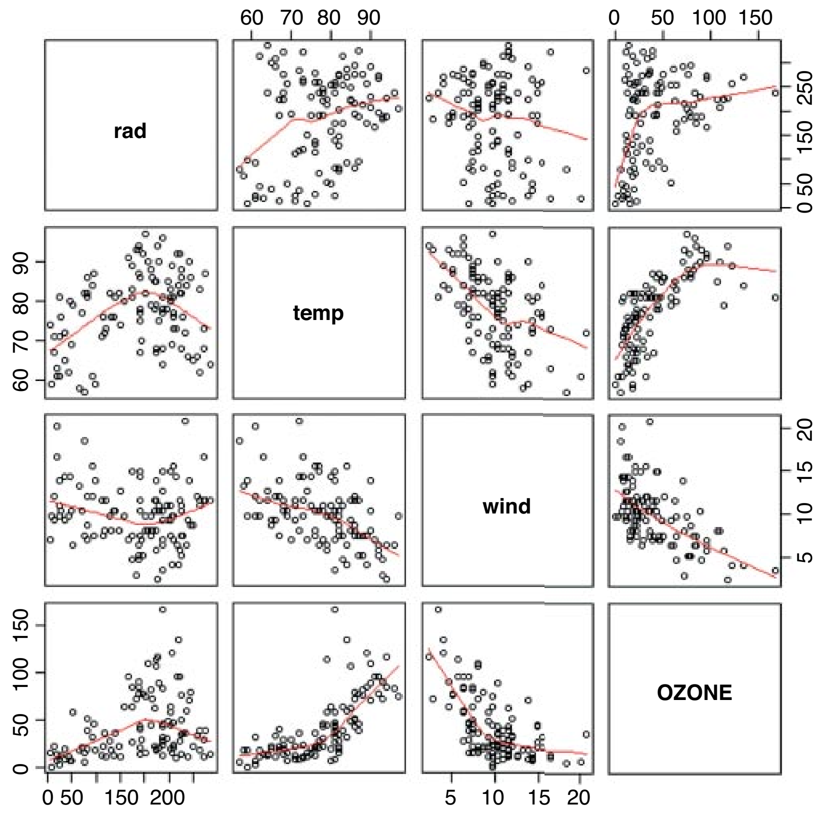
Let us begin with an example from air pollution studies. How is ozone concentration related to wind speed, air temperature and the intensity of solar radiation?

```
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:

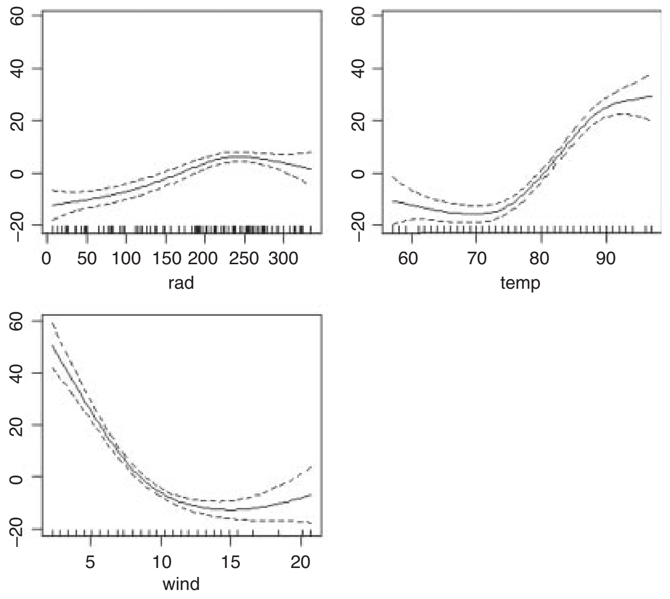
```
pairs(ozone.pollution,panel=panel.smooth)
```



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.

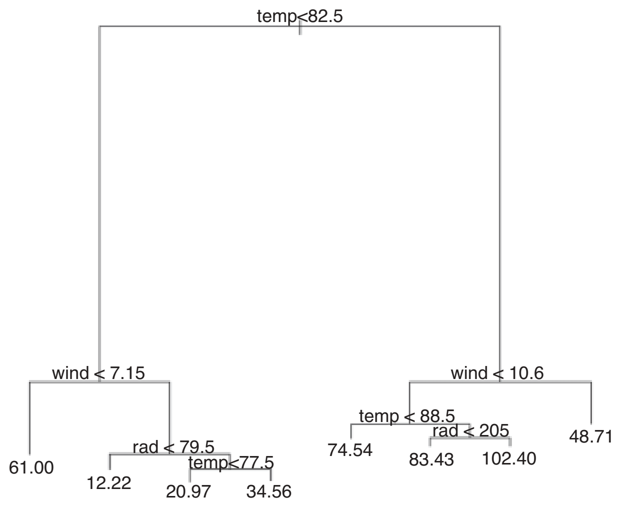
A good way to tackle a multiple regression problem is using non-parametric smoothers in a generalized additive model like this:

```
library(mgcv)
par(mfrow=c(2,2))
model <- gam(ozone~s(rad)+s(temp)+s(wind))
plot(model)
```



The confidence intervals are sufficiently narrow to suggest that the curvature in the relationships between ozone and temperature and ozone and wind are real, but the curvature of the relationship with solar radiation is marginal. The plots lead us to anticipate that quadratic terms for temperature and wind should be included in our initial model. What about interactions? This is where tree models can help:

```
library(tree)
model <- tree(ozone~.,data=ozone.pollution)
par(mfrow=c(1,1))
plot(model)
text(model)
```



This shows that temperature is by far 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). The interaction structure is relatively simple (compare with the other air pollution example on p. 768), but there is a hint of an interaction between wind and radiation and between wind and temperature. We could include these in an initial complex model, degrees of freedom permitting.

```
w2 <- wind^2
t2 <- temp^2
r2 <- rad^2
tw <- temp*wind
wr <- wind*rad
tr <- temp*rad
wtr <- wind*temp*rad
```

Armed with this background information we can begin the linear modelling. We start with the most complicated model: this includes curvature terms for each variable, all three two-way interactions and a three-way interaction:

```
model1 <- lm(ozone~rad+temp+wind+t2+w2+r2+wr+tr+tw+wtr)
summary(model1)

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.683e+02  2.073e+02   2.741  0.00725 **
rad         -3.117e-01  5.585e-01  -0.558  0.57799
temp        -1.076e+01  4.303e+00  -2.501  0.01401 *
wind        -3.237e+01  1.173e+01  -2.760  0.00687 **
t2           5.833e-02  2.396e-02   2.435  0.01668 *
w2           6.106e-01  1.469e-01   4.157 6.81e-05 ***
r2          -3.619e-04  2.573e-04  -1.407  0.16265
wr           2.054e-02  4.892e-02   0.420  0.67552
tr           8.403e-03  7.512e-03   1.119  0.26602
tw           2.377e-01  1.367e-01   1.739  0.08519 .
wtr         -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: < 2.2e-16
```

There looks to be rather little scope for model simplification, so we shall do it all by hand (rather than using `step`, for instance, not least because this is prone to remove main effects that are still present in interactions, or the linear parts of quadratic terms that would best be retained).

We start by removing the highest-order interaction. An excellent feature of R is that the *p* values are ‘*p* values on deletion’ so we do not have to use `anova` to compare the models produced by stepwise deletions:

```
model2 <- update(model1,~.-wtr)
summary(model2)

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.245e+02  1.957e+02   2.680   0.0086 **
```

```
rad           2.628e-02  2.142e-01   0.123   0.9026
temp         -1.021e+01  4.209e+00  -2.427   0.0170 *
wind         -2.802e+01  9.645e+00  -2.906   0.0045 **
t2            5.953e-02  2.382e-02   2.499   0.0141 *
w2            6.173e-01  1.461e-01   4.225 5.25e-05 ***
r2           -3.388e-04  2.541e-04  -1.333   0.1855
wr           -1.127e-02  6.277e-03  -1.795   0.0756 .
tr            3.750e-03  2.459e-03   1.525   0.1303
tw            1.734e-01  9.497e-02   1.825   0.0709 .
```

The least significant term is the quadratic term for radiation, so we remove that:

```
model3 <- update(model2,~.-r2)
summary(model3)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 486.346603 194.333075   2.503  0.01392 *
rad          -0.043163   0.208535  -0.207  0.83644
temp         -9.446780   4.185240  -2.257  0.02613 *
wind        -26.471461   9.610816  -2.754  0.00697 **
t2            0.056966   0.023835   2.390  0.01868 *
w2            0.599709   0.146069   4.106 8.14e-05 ***
wr           -0.011359   0.006300  -1.803  0.07435 .
tr            0.003160   0.002428   1.302  0.19600
tw            0.157637   0.094595   1.666  0.09869 .
```

The temperature by radiation interaction is not significant, so it goes next:

```
model4 <- update(model3,~.-tr)
summary(model4)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 514.401470 193.783580   2.655  0.00920 **
rad           0.212945   0.069283   3.074  0.00271 **
temp        -10.654041   4.094889  -2.602  0.01064 *
wind        -27.391965   9.616998  -2.848  0.00531 **
t2            0.067805   0.022408   3.026  0.00313 **
w2            0.619396   0.145773   4.249 4.72e-05 ***
wr           -0.013561   0.006089  -2.227  0.02813 *
tw            0.169674   0.094458   1.796  0.07538 .
```

The temperature by wind interaction is the next to go (it is marginally significant but we are ruthless):

```
model5 <- update(model4,~.-tw)
summary(model5)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 223.573855 107.618223   2.077 0.040221 *
rad           0.173431   0.066398   2.612 0.010333 *
```

```
temp          -5.197139   2.775039  -1.873 0.063902 .
wind         -10.816032   2.736757  -3.952 0.000141 ***
t2             0.043640   0.018112   2.410 0.017731 *
w2             0.430059   0.101767   4.226 5.12e-05 ***
wr            -0.009819   0.005783  -1.698 0.092507 .
```

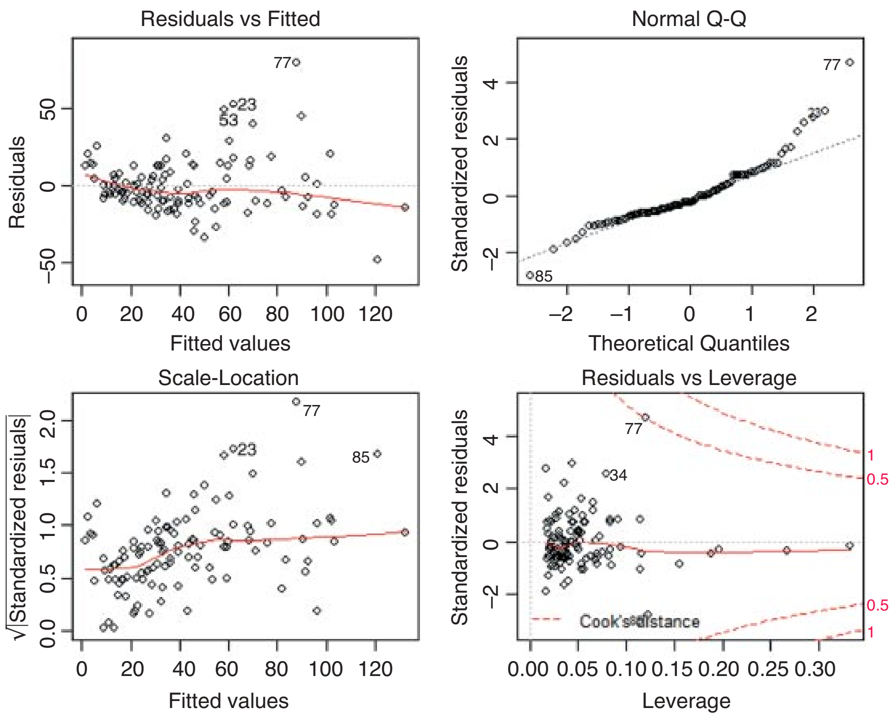
There is no place for the wind by rain interaction:

```
model6 <- update(model5,~.-wr)
summary(model6)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 291.16758  100.87723   2.886  0.00473 **
rad           0.06586    0.02005   3.285  0.00139 **
temp         -6.33955    2.71627  -2.334  0.02150 *
wind        -13.39674    2.29623  -5.834 6.05e-08 ***
t2            0.05102    0.01774   2.876  0.00488 **
w2            0.46464    0.10060   4.619 1.10e-05 ***
```

The next job is to subject `model6` to criticism:

```
par(mfrow=c(2,2))
plot(model6)
```



This is quite seriously badly behaved. The residuals increase with the fitted values (non-constant variance) and the errors are not normal. Let us try transforming the response variable. Having done this we need to start the modelling from scratch with all of the original explanatory variables included. Having transformed the response variable, we should expect that the curvature has been altered:

```
model7 <- lm(log(ozone) ~ rad+temp+wind+t2+w2+r2+wr+tr+tw+wtr)
summary(model7)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.803e+00  5.676e+00   0.494   0.6225
rad          2.771e-02  1.529e-02   1.812   0.0729 .
temp        -3.018e-02  1.178e-01  -0.256   0.7983
wind        -9.812e-02  3.211e-01  -0.306   0.7605
t2           6.034e-04  6.559e-04   0.920   0.3598
w2           8.732e-03  4.021e-03   2.172   0.0322 *
r2          -1.489e-05  7.043e-06  -2.114   0.0370 *
wr          -2.001e-03  1.339e-03  -1.494   0.1382
tr          -2.507e-04  2.056e-04  -1.219   0.2256
tw          -1.985e-03  3.742e-03  -0.530   0.5971
wtr          2.535e-05  1.805e-05   1.404   0.1634
```

```
model8 <- update(model7,~.-wtr)
summary(model8)

model9 <- update(model8,~.-tr)
summary(model9)

model10 <- update(model9,~.-tw)
summary(model10)

model11 <- update(model10,~.-t2)
summary(model11)

model12 <- update(model11,~.-wr)
summary(model12)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  7.724e-01  6.350e-01   1.216 0.226543
rad          7.466e-03  2.323e-03   3.215 0.001736 **
temp         4.193e-02  6.237e-03   6.723 9.52e-10 ***
wind        -2.211e-01  5.874e-02  -3.765 0.000275 ***
w2           7.390e-03  2.585e-03   2.859 0.005126 **
r2          -1.470e-05  6.734e-06  -2.183 0.031246 *

Residual standard error: 0.4851 on 105 degrees of freedom
Multiple R-squared: 0.7004,     Adjusted R-squared: 0.6861
F-statistic:  49.1 on 5 and 105 DF,  p-value: < 2.2e-16
```

```
plot(model12)
```

This is the minimum adequate model. It has five consequential parameters (the intercept of a multiple regression model is usually meaningless; it is the value of the response when every one of the explanatory variables is zero). As predicted by our initial plots, none of the interactions survived the model simplification.

The curvature on the scale of `log(ozone)` is different, of course (we plotted `ozone` against the explanatory variables, not `log(ozone)`). Log transformation of the response improved both the non-constancy of variance and the non-normality of errors. The model explains just over of 70% of the variation in log(ozone concentration).

### 10.13.2 Common problems arising in multiple regression

The following are some of the problems and difficulties that crop up when we do 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, in which there is a near-linear relation between two of the explanatory variables, leading to unstable parameter estimates;
- parameter proliferation where quadratic and interaction terms soak up more degrees of freedom than our data can afford;
- rounding errors during the fitting procedure;
- non-independence of groups of measurements;
- temporal or spatial correlation amongst the explanatory variables;
- pseudoreplication.

Wetherill *et al*. (1986) give a detailed discussion of these problems. We shall encounter other examples of multiple regressions in the context of generalized linear models (Chapter 13), generalized additive models (Chapter 18), survival models (Chapter 27) and mixed-effects models (Chapter 19).