Modern Applied Statistics with S

Fourth edition

by

W. N. Venables and B. D. Ripley

Springer (mid 2002)

Final 15 March 2002

Chapter 8

Non-Linear and Smooth Regression

In linear regression the mean surface is a plane in sample space; in non-linear regression it may be an arbitrary curved surface but in all other respects the models are the same. Fortunately the mean surface in most non-linear regression models met in practice will be approximately planar in the region of highest likelihood, allowing some good approximations based on linear regression to be used, but non-linear regression models can still present tricky computational and inferential problems.

A thorough treatment of non-linear regression is given in Bates and Watts (1988). Another encyclopaedic reference is Seber and Wild (1989), and the books of Gallant (1987) and Ross (1990) also offer some practical statistical advice. The S software is described by Bates and Chambers (1992), who state that its methods are based on those described in Bates and Watts (1988). An alternative approach is described by Huet *et al.* (1996).

Another extension of linear regression is *smooth regression*, in which linear terms are extended to smooth functions whose exact form is not pre-specified but chosen from a flexible family by the fitting procedures. The methods are all fairly computer-intensive, and so only became feasible in the era of plentiful computing power. There are few texts covering this material. Although they do not cover all our topics in equal detail, for what they do cover Hastie and Tibshirani (1990), Simonoff (1996), Bowman and Azzalini (1997) and Hastie *et al.* (2001) are good references, as well as Ripley (1996, Chapters 4 and 5).

8.1 An Introductory Example

Obese patients on a weight reduction programme tend to lose adipose tissue at a diminishing rate. Our dataset `wtloss` has been supplied by Dr T. Davies (personal communication). The two variables are `Days`, the time (in days) since start of the programme, and `Weight`, the patient's weight in kilograms measured under standard conditions. The dataset pertains to a male patient, aged 48, height 193 cm ($6'4''$) with a large body frame. The results are illustrated in Figure 8.1, produced by

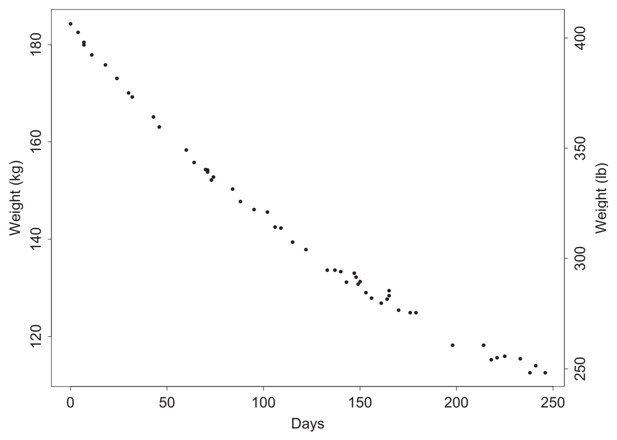

Figure 8.1: Weight loss for an obese patient.

```
attach(wtloss)
# alter margin 4; others are default
oldpar <- par(mar = c(5.1, 4.1, 4.1, 4.1))
plot(Days, Weight, type = "p", ylab = "Weight (kg)")
Wt.lbs <- pretty(range(Weight*2.205))
axis(side = 4, at = Wt.lbs/2.205, lab = Wt.lbs, srt = 90)
mtext("Weight (lb)", side = 4, line = 3)
par(oldpar) # restore settings
```

Although polynomial regression models may describe such data very well within the observed range, they can fail spectacularly outside this range. A more useful model with some theoretical and empirical support is non-linear in the parameters, of the form

$$y = \beta_0 + \beta_1 2^{-t/\theta} + \epsilon \tag{8.1}$$

Notice that all three parameters have a ready interpretation, namely

β_0 is the ultimate lean weight, or asymptote,
β_1 is the total amount to be lost and
θ is the time taken to lose half the amount remaining to be lost,

which allows us to find rough initial estimates directly from the plot of the data.

The parameters β_0 and β_1 are called *linear parameters* since the second partial derivative of the model function with respect to them is identically zero. The parameter, θ, for which this is not the case, is called a *non-linear parameter*.

8.2 Fitting Non-Linear Regression Models

The general form of a non-linear regression model is

$$y = \eta(\boldsymbol{x}, \boldsymbol{\beta}) + \epsilon \tag{8.2}$$

where $\boldsymbol{x}$ is a vector of covariates, $\boldsymbol{\beta}$ is a p-component vector of unknown parameters and ϵ is a $\mathrm{N}(0, \sigma^2)$ error term. In the weight loss example the parameter vector is $\boldsymbol{\beta} = (\beta_0, \beta_1, \theta)^T$. (As $\boldsymbol{x}$ plays little part in the discussion that follows, we often omit it from the notation.)

Suppose $\boldsymbol{y}$ is a sample vector of size n and $\boldsymbol{\eta}(\boldsymbol{\beta})$ is its mean vector. It is easy to show that the maximum likelihood estimate of $\boldsymbol{\beta}$ is a least-squares estimate, that is, a minimizer of $\|\boldsymbol{y} - \boldsymbol{\eta}(\boldsymbol{\beta})\|^2$. The variance parameter σ^2 is then estimated by the residual mean square as in linear regression.

For varying $\boldsymbol{\beta}$ the vector $\boldsymbol{\eta}(\boldsymbol{\beta})$ traces out a p-dimensional surface in $\mathbb{R}^n$ that we refer to as the *solution locus*. The parameters $\boldsymbol{\beta}$ define a coordinate system within the solution locus. From this point of view a linear regression model is one for which the solution locus is a plane through the origin and the coordinate system within it defined by the parameters is affine; that is, it has no curvature. The computational problem in both cases is then to find the coordinates of the point on the solution locus closest to the sample vector $\boldsymbol{y}$ in the sense of Euclidean distance.

The process of fitting non-linear regression models in S is similar to that for fitting linear models, with two important differences:

1. there is no explicit formula for the estimates, so iterative procedures are required, for which initial values must be supplied;
2. linear model formulae that define only the model matrix are not adequate to specify non-linear regression models. A more flexible protocol is needed.

The main S function for fitting a non-linear regression model is `nls`.[1] We can fit the weight loss model by

```
> # R: library(nls)
> wtloss.st <- c(b0 = 90, b1 = 95, th = 120)
> wtloss.fm <- nls(Weight ~ b0 + b1*2^(-Days/th),
    data = wtloss, start = wtloss.st, trace = T)
67.5435 : 90 95 120
40.1808 : 82.7263 101.305 138.714
39.2449 : 81.3987 102.658 141.859
39.2447 : 81.3737 102.684 141.911
> wtloss.fm
Residual sum of squares : 39.245
parameters:
     b0     b1     th
 81.374 102.68 141.91
formula: Weight ~ b0 + b1 * 2^( - Days/th)
52 observations
```

The arguments to `nls` are the following.

`formula` A non-linear model formula. The form is `response ~ mean`, where the right-hand side can have either of two forms. The standard form is an ordinary algebraic expression containing both parameters and determining variables. Note that the operators now have their usual arithmetical

[1]In package `nls` in R.

meaning. (The second form is used with the `plinear` fitting algorithm, discussed in Section 8.3 on page 218.)

`data` An optional data frame for the variables (and sometimes parameters).

`start` A list or numeric vector specifying the starting values for the parameters in the model.
The `names` of the components of `start` are also used to specify which of the variables occurring on the right-hand side of the model formula are parameters. All other variables are then assumed to be determining variables.[2]

`control` An optional argument allowing some features of the default iterative procedure to be changed.

`algorithm` An optional character string argument allowing a particular fitting algorithm to be specified. The default procedure is simply `"default"`.

`trace` An argument allowing tracing information from the iterative procedure to be printed. By default none is printed.

In our example the names of the parameters were specified as `b0`, `b1` and `th`. The initial values of 90, 95 and 120 were found by inspection of Figure 8.1. From the trace output the procedure is seen to converge in three iterations.

Weighted data

The `nls` function has no `weights` argument, but non-linear regressions with *known* weights may be handled by writing the formula as `~ sqrt(W)*(y - M)` rather than `y ~ M`. (The algorithm minimizes the sum of squared differences between left- and right-hand sides and an empty left-hand side counts as zero.) If `W` contains unknown parameters to be estimated the log-likelihood function has an extra term and the problem must be handled by the more general optimization methods such as those discussed in Chapter 16.

Using function derivative information

Most non-linear regression fitting algorithms operate in outline as follows. The first-order Taylor-series approximation to η_k at an initial value $\boldsymbol{\beta}^{(0)}$ is

$$\eta_k(\boldsymbol{\beta}) \approx \eta_k(\boldsymbol{\beta}^{(0)}) + \sum_{j=1}^{p} (\beta_j - \beta_j^{(0)}) \left. \frac{\partial \eta_k}{\partial \beta_j} \right|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(0)}}$$

In vector terms these may be written

$$\boldsymbol{\eta}(\boldsymbol{\beta}) \approx \boldsymbol{\omega}^{(0)} + Z^{(0)}\boldsymbol{\beta} \qquad (8.3)$$

where

$$Z_{kj}^{(0)} = \left. \frac{\partial \eta_k}{\partial \beta_j} \right|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(0)}} \qquad \text{and} \qquad \omega_k^{(0)} = \eta_k(\boldsymbol{\beta}^{(0)}) - \sum_{j=1}^{p} \beta_j^{(0)} Z_{kj}^{(0)}$$

[2]In S-PLUS there is a bug that may be avoided if the order in which the parameters appear in the `start` vector is the same as the order in which they first appear in the model. It is as if the order in the `names` attribute were ignored.

Equation (8.3) defines the tangent plane to the surface at the coordinate point $\boldsymbol{\beta} = \boldsymbol{\beta}^{(0)}$. The process consists of regressing the observation vector $\boldsymbol{y}$ onto the tangent plane defined by $Z^{(0)}$ with *offset* vector $\boldsymbol{\omega}^{(0)}$ to give a new approximation, $\boldsymbol{\beta} = \boldsymbol{\beta}^{(1)}$, and iterating to convergence. For a linear regression the offset vector is $\mathbf{0}$ and the matrix $Z^{(0)}$ is the model matrix X, a constant matrix, so the process converges in one step. In the non-linear case the next approximation is

$$\boldsymbol{\beta}^{(1)} = \big(Z^{(0)\,T} Z^{(0)}\big)^{-1} Z^{(0)\,T}\big(\boldsymbol{y} - \boldsymbol{\omega}^{(0)}\big)$$

With the `default` algorithm the Z matrix is computed approximately by numerical methods unless formulae for the first derivatives are supplied. Providing derivatives often (but not always) improves convergence.

Derivatives can be provided as an attribute of the model. The standard way to do this is to write an S function to calculate the mean vector $\boldsymbol{\eta}$ and the Z matrix. The result of the function is $\boldsymbol{\eta}$ with the Z matrix included as a `gradient` attribute.

For our simple example the three derivatives are

$$\frac{\partial\eta}{\partial\beta_0} = 1, \qquad \frac{\partial\eta}{\partial\beta_1} = 2^{-x/\theta}, \qquad \frac{\partial\eta}{\partial\theta} = \frac{\log(2)\,\beta_1 x 2^{-x/\theta}}{\theta^2}$$

so an S function to specify the model including derivatives is

```
expn <- function(b0, b1, th, x) {
    temp <- 2^(-x/th)
    model.func <- b0 + b1 * temp
    Z <- cbind(1, temp, (b1 * x * temp * log(2))/th^2)
    dimnames(Z) <- list(NULL, c("b0", "b1", "th"))
    attr(model.func, "gradient") <- Z
    model.func
}
```

Note that the gradient matrix must have column names matching those of the corresponding parameters.

We can fit our model again using first derivative information:

```
> wtloss.gr <- nls(Weight ~ expn(b0, b1, th, Days),
    data = wtloss, start = wtloss.st, trace = T)
67.5435 : 90 95 120
40.1808 : 82.7263 101.305 138.714
39.2449 : 81.3987 102.658 141.859
39.2447 : 81.3738 102.684 141.911
```

This appears to make no difference to the speed of convergence, but tracing the function `expn` shows that only 6 evaluations are required when derivatives are supplied compared with 21 if they are not supplied.

Functions such as `expn` can often be generated automatically using the symbolic differentiation function `deriv`. It is called with three arguments:

(a) the model formula, with the left-hand side optionally left blank,

(b) a character vector giving the names of the parameters and

(c) an empty function with an argument specification as required for the result.

An example makes the process clearer. For the weight loss data with the exponential model, we can use:

```
expn1 <- deriv(y ~ b0 + b1 * 2^(-x/th), c("b0", "b1", "th"),
               function(b0, b1, th, x) {})
```

The result in S-PLUS is the function (R's result is marginally different)

```
expn1 <- function(b0, b1, th, x)
{
   .expr3 <- 2^(( - x)/th)
   .value <- b0 + (b1 * .expr3)
   .grad <- array(0, c(length(.value), 3),
               list(NULL, c("b0", "b1", "th")))
   .grad[, "b0"] <- 1
   .grad[, "b1"] <- .expr3
   .grad[, "th"] <- b1 *
       (.expr3 * (0.693147180559945 * (x/(th^2))))
   attr(.value, "gradient") <- .grad
   .value
}
```

Self-starting non-linear regressions

Very often reasonable starting values for a non-linear regression can be calculated by some fairly simple automatic procedure. Setting up such a *self-starting* non-linear model is somewhat technical, but several examples[3] are supplied.

Consider once again a negative exponential decay model such as that used in the weight loss example but this time written in the more usual exponential form:

$$y = \beta_0 + \beta_1 \exp(-x/\theta) + \epsilon$$

One effective initial value procedure follows.

(i) Fit an initial quadratic regression in x.

(ii) Find the fitted values, say, y_0, y_1 and y_2 at three equally spaced points x_0, $x_1 = x_0 + \delta$ and $x_2 = x_0 + 2\delta$.

(iii) Equate the three fitted values to their expectation under the non-linear model to give an initial value for θ as

$$\theta_0 = \delta \;/\; \log\left(\frac{y_0 - y_1}{y_1 - y_2}\right)$$

(iv) Initial values for β_0 and β_1 can then be obtained by linear regression of y on $\exp(-x/\theta_0)$.

[3]Search for objects with names starting with `SS`, in R in package `nls`.

An S function to implement this procedure (with a few extra checks) called `negexp.SSival` is supplied in `MASS`; interested readers should study it carefully.

We can make a self-starting model with both first derivative information and this initial value routine by.

```
negexp <- selfStart(model = ~ b0 + b1*exp(-x/th),
    initial = negexp.SSival, parameters = c("b0", "b1", "th"),
    template = function(x, b0, b1, th) {})
```

where the first, third and fourth arguments are the same as for `deriv`. We may now fit the model without explicit initial values.

```
> wtloss.ss <- nls(Weight ~ negexp(Days, B0, B1, theta),
                   data = wtloss, trace = T)
     B0     B1  theta
 82.713 101.49 200.16
39.5453 : 82.7131 101.495 200.160
39.2450 : 81.3982 102.659 204.652
39.2447 : 81.3737 102.684 204.734
```

(The first two lines of output come from the initial value procedure and the last three from the `nls` trace.)

8.3 Non-Linear Fitted Model Objects and Method Functions

The result of a call to `nls` is an object of class `nls`. The standard method functions are available.

For the preceding example the summary function gives:

```
> summary(wtloss.gr)
Formula: Weight ~ expn1(b0, b1, th, Days)
Parameters:
     Value Std. Error t value
b0  81.374     2.2690  35.863
b1 102.684     2.0828  49.302
th 141.911     5.2945  26.803
Residual standard error: 0.894937 on 49 degrees of freedom
Correlation of Parameter Estimates:
       b0     b1
b1 -0.989
th -0.986  0.956
```

Surprisingly, no working deviance method function exists but such a method function is easy to write and is included in `MASS`. It merely requires

```
> deviance.nls <- function(object) sum(object$residuals^2)
> deviance(wtloss.gr)
[1] 39.245
```

`MASS` also has a generic function `vcov` that will extract the estimated variance matrix of the mean parameters:

```
> vcov(wtloss.gr)
         b0       b1      th
b0   5.1484  -4.6745 -11.841
b1  -4.6745   4.3379  10.543
th -11.8414  10.5432  28.032
```

Taking advantage of linear parameters

If all non-linear parameters were known the model would be linear and standard linear regression methods could be used. This simple idea lies behind the "plinear" algorithm. It requires a different form of model specification that combines aspects of linear and non-linear model formula protocols. In this case the right-hand side expression specifies a *matrix* whose columns are functions of the non-linear parameters. The linear parameters are then implied as the regression coefficients for the columns of the matrix. Initial values are only needed for the non-linear parameters. Unlike the linear model case there is no implicit intercept term.

There are several advantages in using the partially linear algorithm. It can be much more stable than methods that do not take advantage of linear parameters, it requires fewer initial values and it can often converge from poor starting positions where other procedures fail.

Asymptotic regressions with different asymptotes

As an example of a case where the partially linear algorithm is very convenient, we consider a dataset first discussed in Linder, Chakravarti and Vuagnat (1964). The object of the experiment was to assess the influence of calcium in solution on the contraction of heart muscle in rats. The left auricle of 21 rat hearts was isolated and on several occasions electrically stimulated and dipped into various concentrations of calcium chloride solution, after which the shortening was measured. The data frame muscle in MASS contains the data as variables Strip, Conc and Length.

The particular model posed by the authors is of the form

$$\log y_{ij} = \alpha_j + \beta\rho^{x_{ij}} + \varepsilon_{ij} \tag{8.4}$$

where i refers to the concentration and j to the muscle strip. This model has 1 non-linear and 22 linear parameters. We take the initial estimate for ρ to be 0.1. Our first step is to construct a matrix to select the appropriate α.

```
> A <- model.matrix(~ Strip - 1, data = muscle)
> rats.nls1 <- nls(log(Length) ~ cbind(A, rho^Conc),
    data = muscle, start = c(rho = 0.1), algorithm = "plinear")
> (B <- coef(rats.nls1))
      rho  .lin1  .lin2  .lin3  .lin4  .lin5  .lin6 .lin7
 0.077778 3.0831 3.3014 3.4457 2.8047 2.6084 3.0336 3.523
  .lin8  .lin9 .lin10 .lin11 .lin12 .lin13 .lin14 .lin15 .lin16
 3.3871 3.4671 3.8144 3.7388 3.5133 3.3974 3.4709  3.729 3.3186
 .lin17 .lin18 .lin19 .lin20 .lin21  .lin22
 3.3794 2.9645 3.5847 3.3963   3.37 -2.9601
```