
Technical note: Curve fitting with the R Environment for Statistical Computing

D G Rossiter

May 15, 2016

Contents

1	Curve fitting	1
2	Fitting intrinsically linear relations	1
3	Fitting linearizable relations	1
4	Non-linear curve fitting	1
4.1	Fitting a power model	2
4.2	Fitting to a functional form	6
4.3	Fitting an exponential model	10
4.4	Fitting a piecewise model	14
	References	20
	Index of R concepts	21

Version 1.1 Copyright © 2009, 2016 D G Rossiter. All rights reserved. Reproduction and dissemination of the work as a whole (not parts) freely permitted if this original copyright notice is included. Sale or placement on a web site where payment must be made to access this document is strictly prohibited. To adapt or translate please contact the author (dgr2@cornell.edu).

1 Curve fitting

This is a small introduction to curve fitting in the R environment for statistical computing and visualisation [3, 6] and its dialect of the S language. R provides a sophisticated environment, which gives the user more insight and control than provided by commercial or shareware “push the button” programs such as CurveFit.

Note: The code in these exercises was tested with Sweave [5, 4] on R version 3.2.4 (2016-03-10), **stats** package Version: 3.2.4, running on Mac OS X 10.11.4. The text and graphical output you see here was automatically generated and incorporated into L^AT_EX by running actual code through R and its packages. Then the L^AT_EX document was compiled into the PDF version you are now reading. Your output may be slightly different on different versions and on different platforms.

2 Fitting intrinsically linear relations

Relations that are expected to be linear (from theory or experience) are usually fit with R’s **lm** “linear model” method, which by default uses ordinary least squares (OLS) to minimize the sum of squares of the residuals. This is covered in many texts and another tutorial of this series [7].

However, linear relations with some contamination (e.g. outliers) may be better fit by **robust** regression, for example the **lmRob** function in the **robust** package.

After fitting a linear model, the analyst should always check the **regression diagnostics** appropriate to the model, to see if the model **assumptions** are met. For example, the ordinary least squares fit to a linear model assumes, among others: (1) normally-distributed residuals; (2) homoscedasticity (variance of the response does not depend on the value of the predictor); (3) serial independence (no correlation between responses for nearby values of the predictor).

3 Fitting linearizable relations

Some evidently non-linear relations can be **linearized** by transforming either the response or predictor variables. This should generally be done on the basis of theory, e.g. an expected multiplicative effect of a causative variable would indicate an exponential response, thus a logarithmic transformation of the response variable.

An example of a log-linear model is shown in §4.3.

4 Non-linear curve fitting

Equations that can not be linearized, or for which the appropriate linearization is not known from theory, can be fitted with the **nls** method, based on the classic text of Bates and Watts [2] and included in the base R distribution’s **stats** package.

You must have some idea of the functional form, presumably from theory. You can of course try various forms and see which gives the closest fit, but that may result in fitting noise, not model.

4.1 Fitting a power model

We begin with a simple example of a known functional form with some noise, and see how close we can come to fitting it.

Task 1 : Make a data frame of 24 uniform random variates (independent variable) and corresponding dependent variable that is the cube, with noise. Plot the points along with the known theoretical function. •

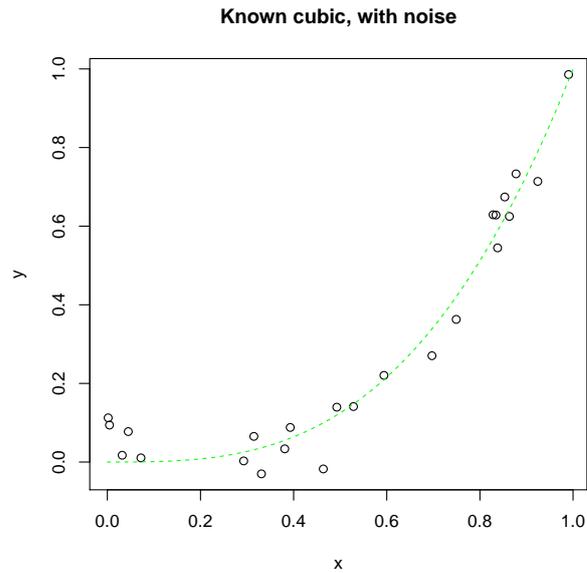
So that your results match the ones shown here, we use the `set.seed` function to initialize the random-number generator; in practice this is not done unless one wants to reproduce a result. The choice of seed is arbitrary. The random numbers are generated with the `runif` (uniform distribution, for the independent variable) and `rnorm` (normal distribution, for the independent variable) functions. These are then placed into a two-column matrix with named columns with the `data.frame` function.

```
> set.seed(1485)

> len <- 24
> x <- runif(len)
> y <- x^3 + rnorm(len, 0, 0.06)
> ds <- data.frame(x = x, y = y)
> str(ds)

'data.frame':      24 obs. of  2 variables:
 $ x: num  0.838 0.5285 0.8344 0.0721 0.9242 ...
 $ y: num  0.5448 0.1412 0.6284 0.0106 0.7139 ...

> plot(y ~ x, main = "Known cubic, with noise")
> s <- seq(0, 1, length = 100)
> lines(s, s^3, lty = 2, col = "green")
```



Suppose this is a dataset collected from an experiment, and we want to determine the most likely value for the exponent. In the simplest case, we assume that the function passes through $(0, 0)$; we suppose there is a physical reason for that. The power model is then

$$y_i = (x_i)^m + \varepsilon_i \quad (1)$$

where m is the exponent and the errors ε_i are assumed to be identically and independently distributed (“i.i.d”), generally as normally-distributed with zero mean and some variance: $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.

Task 2 : Fit a power model, with zero intercept, to this data. •

We use the workhorse `nls` function, which is analogous to `lm` for linear models. This requires at least:

1. a **formula** of the functional form;
2. the **environment** of the variable names listed in the formula;
3. a named list of **starting guesses** for these.

We’ll specify the power model: $y \sim I(x^{\text{power}})$ and make a starting guess that it’s a linear relation, i.e. that the power is 1.

Note: Note the use of the `I` operator to specify that the `^` exponentiation operator is a mathematic operator, not the `^` formula operator (factor crossing). In this case there is no difference, because there is only one predictor, but in the general case it must be specified.

We use the optional `trace=T` argument to see how the non-linear fit converges.

```
> m <- nls(y ~ I(x^power), data = ds, start = list(power = 1),
+         trace = T)
```

```

1.814289 : 1
0.3012573 : 2.050361
0.08280959 : 2.881157
0.07643193 : 3.091093
0.07643191 : 3.091473
0.07643191 : 3.091458

```

```
> class(m)
```

```
[1] "nls"
```

The `nls` function has returned an object of class `nls`, for which many further functions are defined.

Task 3 : Display the solution. •

The generic `summary` method specializes to the non-linear model:

```
> summary(m)
```

```
Formula: y ~ I(x^power)
```

```
Parameters:
```

```

      Estimate Std. Error t value Pr(>|t|)
power    3.091      0.161    19.2 1.17e-15 ***
---

```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.05765 on 23 degrees of freedom
```

```
Number of iterations to convergence: 5
```

```
Achieved convergence tolerance: 8.035e-07
```

```
> summary(m)$coefficients
```

```

      Estimate Std. Error t value Pr(>|t|)
power 3.091458    0.161011 19.20028 1.174765e-15

```

We can see that the estimated power is 3.091 ± 0.161

The standard error of the coefficient shows how uncertain we are of the solution.

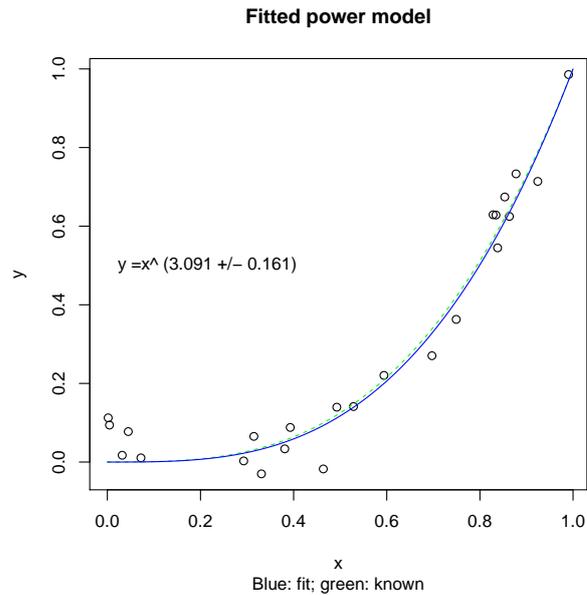
Task 4 : Plot the fitted curve against the known curve. •

We use the `predict` method to find the function value for the fitted power function along the sequence $[0, 0.01, 0.02, \dots, 0.99, 1]$, and use these to plot the fitted power function.

```

> power <- round(summary(m)$coefficients[1], 3)
> power.se <- round(summary(m)$coefficients[2], 3)
> plot(y ~ x, main = "Fitted power model", sub = "Blue: fit; green: known")
> s <- seq(0, 1, length = 100)
> lines(s, s^3, lty = 2, col = "green")
> lines(s, predict(m, list(x = s)), lty = 1, col = "blue")
> text(0, 0.5, paste("y =x^ (", power, " +/- ", power.se,
+      ")", sep = ""), pos = 4)

```



Despite the noise, the fit is quite close to the known power.

Task 5 : Determine the goodness of the fit. •

R^2

We compute the residual sum-of-squares (lack of fit) and the complement of its proportion to the total sum-of-squares (coefficient of determination, “ R^2 ”):

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}} \quad (2)$$

```
> (RSS.p <- sum(residuals(m)^2))
```

```
[1] 0.07643191
```

```
> (TSS <- sum((y - mean(y))^2))
```

```
[1] 2.172195
```

```
> 1 - (RSS.p/TSS)
```

```
[1] 0.9648135
```

We can compare this to the lack-of-fit to the known cubic, where the lack of fit is due to the noise:

```
> 1 - sum((x^3 - y)^2)/TSS
```

```
[1] 0.9642825
```

They are hardly distinguishable; the known cubic will not necessarily be better, this varies with each simulation.

AIC

Another way to evaluate the success of a model fit is with Akaike’s Information Criterion (AIC) [1], defined as:

$$\text{AIC} = -2 \log(L) + 2k \quad (3)$$

where k is the number of degrees of freedom used in the model and L is the value of the *likelihood function*, i.e., the likelihood of the observations having been generated by the assumed model.

likelihood

The likelihood function is:

$$L(\theta; Z) = \prod_{i=1}^N g_{\theta}(z_i) \quad (4)$$

where $Z = (z_1, z_2, \dots, z_N)$ are the N observed values, θ is the vector of parameters of the assumed model, and g_{θ} is the model equation with the specified parameters. In this example θ has two elements, the power m and the variance of the errors σ^2 , so that $g_{\theta}(z_i) = (x_i)^m + \mathcal{N}(0, \sigma^2)$. As m is varied, so is $g_{\theta}(z_i)$ of each observation, and hence the likelihood. The values of (m, σ^2) at which the product of the $g_{\theta}(z_i)$ is maximized are called the *maximum likelihood* values of the parameters, and provides a general method for fitting equations.

Since the likelihood in the AIC formula has a negative sign, the smaller (with sign) the better.

Task 6 : Compute the AIC for this model fit. •

This is computed with (surprise!) the AIC function:

```
> AIC(m)
```

```
[1] -65.87676
```

This has no interpretation by itself, because (1) it depends on the set of observations; (2) the AIC is unique up to a constant. However, the AIC is useful to compare models, see below.

4.2 Fitting to a functional form

The more general way to use `nls` is to define a **function** for the right-hand side of the non-linear equation. We illustrate for the power model, but without assuming that the curve passes through $(0, 0)$.

Task 7 : Fit a power model and intercept. •

First we define a function, then use it in the formula for `nls`. The function takes as arguments:

1. the input vector, i.e. independent variable(s);
2. the parameters; these must match the call and the arguments to the `start=` initialization argument, but they need not have the same names.

```

> rhs <- function(x, b0, b1) {
+   b0 + x^b1
+ }
> m.2 <- nls(y ~ rhs(x, intercept, power), data = ds, start = list(intercept = 0,
+   power = 2), trace = T)

```

```

0.3299172 : 0 2
0.09528284 : 0.05152784 3.22261047
0.06804034 : 0.03275645 3.46465245
0.0679837 : 0.0300599 3.4248749
0.06798214 : 0.03042144 3.43209077
0.06798209 : 0.03035352 3.43079872
0.06798209 : 0.03036561 3.43103066
0.06798209 : 0.03036343 3.43098899

```

```

> summary(m.2)

```

Formula: $y \sim \text{rhs}(x, \text{intercept}, \text{power})$

Parameters:

	Estimate	Std. Error	t value	Pr(> t)
intercept	0.03036	0.01898	1.60	0.124
power	3.43099	0.29164	11.77	5.81e-11 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.05559 on 22 degrees of freedom

Number of iterations to convergence: 7

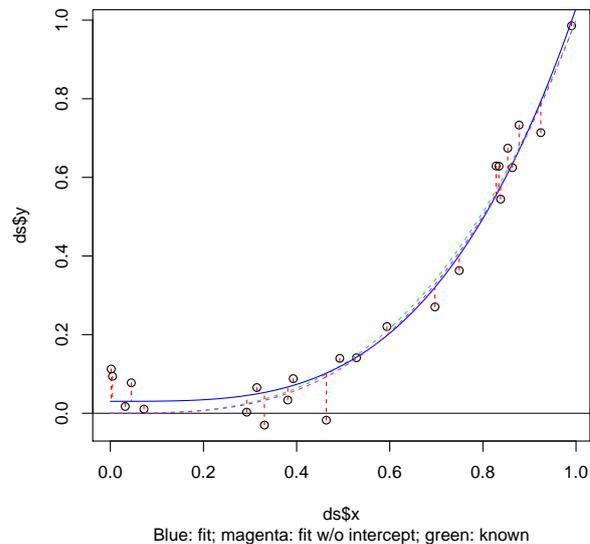
Achieved convergence tolerance: 5.512e-06

```

> plot(ds$y ~ ds$x, main = "Fitted power model, with intercept",
+   sub = "Blue: fit; magenta: fit w/o intercept; green: known")
> abline(h = 0, lty = 1, lwd = 0.5)
> lines(s, s^3, lty = 2, col = "green")
> lines(s, predict(m.2, list(x = s)), lty = 1, col = "blue")
> lines(s, predict(m, list(x = s)), lty = 2, col = "magenta")
> segments(x, y, x, fitted(m.2), lty = 2, col = "red")

```

Fitted power model, with intercept



This example shows the effect of forcing the equation through a known point, in this case (0,0). Since the model has one more parameter, it will by definition fit better near the origin. However, in this case it is fitting noise, not structure.

Task 8 : Compare the fit with the known relation and the power-only model. •

```
> (RSS.pi <- sum(residuals(m.2)^2))
[1] 0.06798209
> (r2.pi <- (1 - (RSS.pi/TSS)))
[1] 0.9687035
> (r2.p <- 1 - (RSS.p/TSS))
[1] 0.9648135
> (r2.3 <- 1 - sum((x^3 - y)^2)/TSS)
[1] 0.9642825
```

In this case the model with intercept explains more of the variation in the observations.

adjusted R^2

The unadjusted R^2 can be empirically adjusted to account for the number of model parameters p (more parameters allow a closer fit) and the number of observations n (more observations allow more parameters without overfitting):

$$R_{\text{adj}}^2 = 1 - \left[\frac{(n-1)}{(n-p)} \cdot (1 - R^2) \right] \quad (5)$$

For the power model and the known cubic there is only one parameter ($p = 1$), so the adjustment expression $(n-1)/(n-p) = 1$ and there is no adjustment needed. However for the model with both power and intercept, there is an additional parameter (the intercept), and we see its effect in allowing the curve to fit better near the origin.

Task 9 : Compute the adjusted R^2 of the with-intercept model. •

The `dim` function gives the dimensions of an array; here the first dimension is the number of rows, i.e., observations. The `length` function gives the number of elements in a vector, here the coefficients vector, extracted with the `coefficients` function.

```
> (n <- dim(ds)[1])
[1] 24
> (p <- length(coefficients(m.2)))
```

```

[1] 2
> (r2.adj.pi <- 1 - (((n - 1)/(n - p)) * (1 - r2.pi)))
[1] 0.9672809
> r2.pi
[1] 0.9687035
> r2.pi - r2.adj.pi
[1] 0.001422568

```

Adding one parameter to even this small dataset results in very little adjustment to the R^2 .

Task 10 : Compare the two models (with and without intercept) with an Analysis of Variance. •

```
> anova(m.2, m)
```

Analysis of Variance Table

Model 1: $y \sim \text{rhs}(x, \text{intercept}, \text{power})$

Model 2: $y \sim I(x^{\text{power}})$

	Res.Df	Res.Sum Sq	Df	Sum Sq F value	Pr(>F)
1	22	0.067982			
2	23	0.076432	-1	-0.0084498	2.7345

The $\text{Pr}(>F)$ value is the probability that rejecting the null hypothesis (the more complex model does not fit better than the simpler model) would be a mistake; in this case since we know there shouldn't be an intercept, we hope that this will be high.

The use of adjusted R^2 to compare nonlinear models was criticized by Spiess and Neumeyer [8]; these authors recommend using the AIC.

Task 11 : Compare the two models (with and without intercept) by their AIC. •

Note that comparison by AIC is only valid if the models were fit on the same data, since AIC is based on the likelihood of observing these data, conditional on the assumed model.

```
> AIC(m.2)
```

```
[1] -66.68851
```

```
> AIC(m)
```

```
[1] -65.87676
```

The AIC of the model with intercept is in this case (i.e., with these observations) less (here, more negative), showing that it is a better model, even after compensating for the additional parameter. We know this is not “true”, since

the known model does not have an intercept. If you do this comparison with other simulated values you will get different AIC values and in some cases a different ranking of the models.

4.3 Fitting an exponential model

Looking at the scatterplot we might suspect an exponential relation. This can be fit in two ways:

- linearizing by taking the logarithm of the response;
- with non-linear fit, as in the previous section.

The first approach works because:

$$y = e^{a+bx} \equiv \log(y) = a + bx$$

Task 12 : Fit a log-linear model. •

The logarithm is not defined for non-positive numbers, so we have to add a small offset if there are any of these (as here). One way to define this is as the decimal 0.1, 0.01, 0.001... that is just large enough to bring all the negative values above zero. Here the minimum is:

```
> min(y)
```

```
[1] -0.03000078
```

and so we should add 0.1.

```
> offset <- 0.1
> ds$ly <- log(ds$y + offset)
> m.l <- lm(ly ~ x, data = ds)
> summary(m.l)
```

Call:

```
lm(formula = ly ~ x, data = ds)
```

Residuals:

```
      Min       1Q   Median       3Q      Max
-1.1356 -0.1421  0.0578  0.2590  0.8064
```

Coefficients:

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -2.3589      0.1813  -13.011 8.28e-12 ***
x              2.1554      0.2977   7.241 2.97e-07 ***
---
```

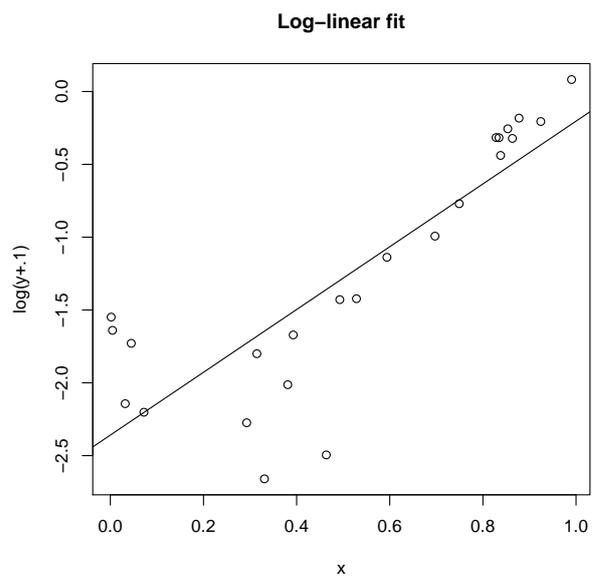
```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 0.4701 on 22 degrees of freedom

Multiple R-squared: 0.7044, Adjusted R-squared: 0.691

F-statistic: 52.43 on 1 and 22 DF, p-value: 2.966e-07

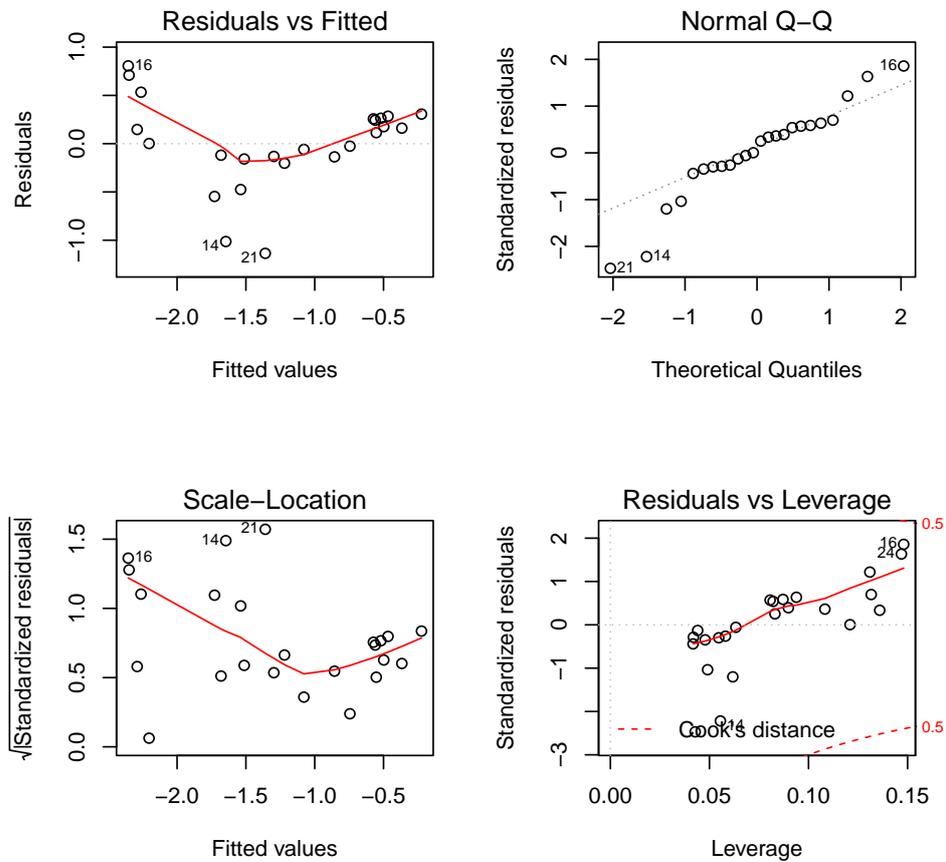
```
> plot(ds$ly ~ ds$x, xlab = "x", ylab = "log(y+.1)", main = "Log-linear fit")
> abline(m.l)
> text(0, 0.4, pos = 4, paste("log(y) = ", round(coefficients(m.l)[1],
+ 3), "+", round(coefficients(m.l)[2], 3)))
```



Here the adjusted R^2 is 0.691, but this can not be compared to the non-linear fit, because of the transformation. Neither can the AIC, here 35.79, be compared, because the observations have different values before and after the transformation.

Since this is a linear model, we can evaluate the regression diagnostics:

```
> par(mfrow = c(2, 2))
> plot(m.1)
> par(mfrow = c(1, 1))
```



Clearly the log-linear model is not appropriate.

The second way is with `nls`.

Task 13 : Directly fit an exponential model. •

The functional form is $y = e^{a+bx}$. A reasonable starting point is $a = 0, b = 1$, i.e. $y = e^x$.

```
> m.e <- nls(y ~ I(exp(1)^(a + b * x)), data = ds, start = list(a = 0,
+   b = 1), trace = T)
```

```
53.5851 : 0 1
5.713618 : -1.073559 1.390495
0.5851148 : -2.232398 2.219397
0.1222333 : -3.397036 3.416349
0.09275322 : -4.006025 4.092473
0.09271058 : -4.016730 4.101149
0.09271053 : -4.015782 4.100066
0.09271053 : -4.015905 4.100206
0.09271053 : -4.015889 4.100188
```

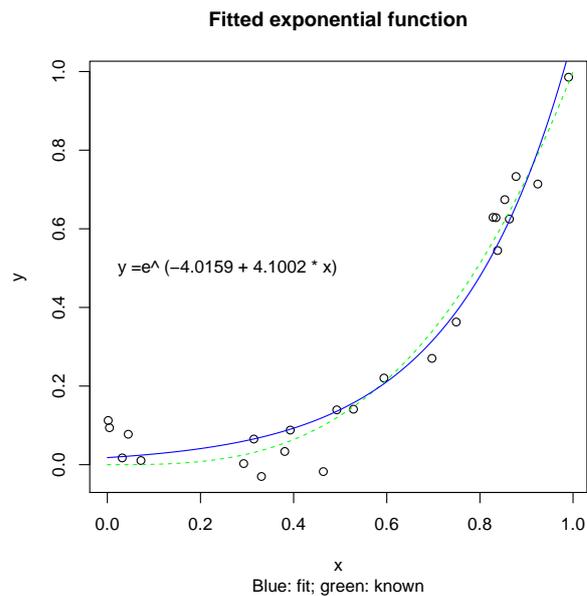
```
> summary(m.e)$coefficients
```

```

      Estimate Std. Error  t value    Pr(>|t|)
a -4.015889   0.2742684 -14.64219 7.976698e-13
b  4.100188   0.3094644  13.24930 5.801485e-12

> a <- round(summary(m.e)$coefficients[1, 1], 4)
> b <- round(summary(m.e)$coefficients[2, 1], 4)
> plot(y ~ x, main = "Fitted exponential function", sub = "Blue: fit; green: known")
> s <- seq(0, 1, length = 100)
> lines(s, s^3, lty = 2, col = "green")
> lines(s, predict(m.e, list(x = s)), lty = 1, col = "blue")
> text(0, 0.5, paste("y = e^ (", a, " + ", b, " * x)", sep = ""),
+       pos = 4)

```



Here the goodness-of-fit can be compared directly to that for the power model, since they both have one parameter and untransformed observations:

```

> RSS.p
[1] 0.07643191
> (RSS.e <- sum(residuals(m.e)^2))
[1] 0.09271053
> TSS
[1] 2.172195
> 1 - RSS.p/TSS
[1] 0.9648135
> 1 - RSS.e/TSS
[1] 0.9573194

```

The fit is not as good as for the power model, which suggests that the exponential model is an inferior functional form.

The AICs can also be compared, since the original observations are used in both cases:

```
> AIC(m)
[1] -65.87676
> AIC(m.e)
[1] -59.2428
```

Clearly, the exponential model is much inferior.

4.4 Fitting a piecewise model

A big advantage of the `nls` method is that any function can be optimized. This must be continuous in the range of the predictor but not necessarily differentiable.

An example is the **linear-with-plateau** model sometimes used to predict crop yield response to fertilizers. The theory is that up to some **threshold**, added fertilizer linearly increases yield, but once the maximum yield is reached (limited by light and water, for example) added fertilizer makes no difference. So there are four parameters: (1) intercept: yield with no fertilizer; (2) slope: yield increase per unit fertilizer added; (3) threshold yield: maximum attainable; (4) threshold fertilizer amount: where this yield is attained.

Note that one parameter is redundant: knowing the linear part and the threshold yield we can compute the threshold amount, or with the amount the yield.

Task 14 : Define a linear-response-with-plateau function. •

We define the function with three parameters, choosing to fit the maximum fertilizer amount, from which we can back-compute the maximum yield (plateau). We use the `ifelse` operator to select the two parts of the function, depending on the threshold.

```
> f.lrp <- function(x, a, b, t.x) {
+   ifelse(x > t.x, a + b * t.x, a + b * x)
+ }
```

Task 15 : Generate a synthetic data set to represent a fertilizer experiment with 0, 10, ...120 kg ha⁻¹ added fertilizer, with three replications, with known linear response $y = 2 + 0.5x$ and maximum fertilizer which gives response of 70 kg ha⁻¹. •

In nature there are always random factors; we account for this by adding normally-distributed noise with the `rnorm` function. Again we use `set.seed` so your results will be the same, but you can experiment with other random values.

```

> f.lvls <- seq(0, 120, by = 10)
> a.0 <- 2
> b.0 <- 0.05
> t.x.0 <- 70
> test <- data.frame(x = f.lvls, y = f.lrp(f.lvls, a.0,
+   b.0, t.x.0))
> test <- rbind(test, test, test)
> set.seed <- 1040
> test$y <- test$y + rnorm(length(test$y), 0, 0.2)
> str(test)

'data.frame':      39 obs. of  2 variables:
 $ x: num  0 10 20 30 40 50 60 70 80 90 ...
 $ y: num  1.82 2.53 2.99 3.5 3.72 ...

```

In this example the maximum attainable yield is 5.5, for any fertilizer amount from 70 on. No fertilizer gives a yield of 2 and each unit of fertilizer added increases the yield 0.05 units. The noise represents the intrinsic error in field experiments. Note that the amount of fertilizer added is considered exact, since it is under the experimenter's control.

Task 16 : Plot the experiment with the known true model. •

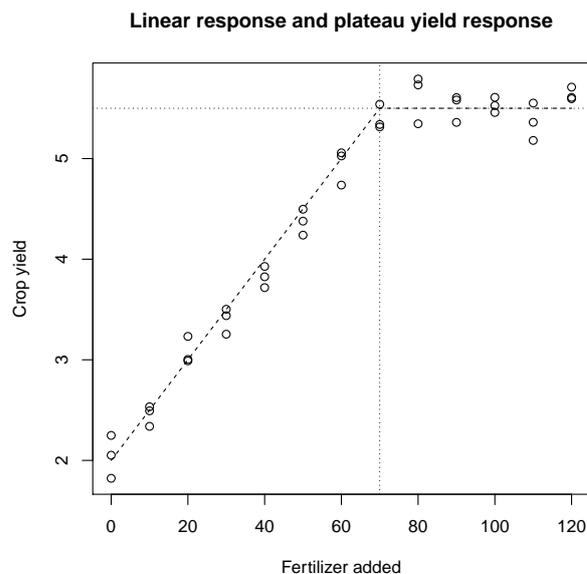
```

> plot(test$y ~ test$x, main = "Linear response and plateau yield response",
+   xlab = "Fertilizer added", ylab = "Crop yield")
> (max.yield <- a.0 + b.0 * t.x.0)

[1] 5.5

> lines(x = c(0, t.x.0, 120), y = c(a.0, max.yield, max.yield),
+   lty = 2)
> abline(v = t.x.0, lty = 3)
> abline(h = max.yield, lty = 3)

```



Although it's not needed for this example, the replication number should be added to the dataframe as a factor; we use the `rep` "replicate" function to

create the vector of replication numbers, and then `as.factor` to convert to a factor. The `table` function gives the count of each replicate.

```
> test$rep <- as.factor(rep(1:3, each = length(test$y)/3))
> str(test)

'data.frame':      39 obs. of  3 variables:
 $ x  : num  0 10 20 30 40 50 60 70 80 90 ...
 $ y  : num  1.82 2.53 2.99 3.5 3.72 ...
 $ rep: Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...

> table(test$rep)

 1  2  3
13 13 13
```

The different replications have slightly different mean yields, due to random error; we see this with the `by` function to split a vector by a factor and then apply a function per-factor; in this case `mean`:

```
> by(test$y, test$rep, mean)

test$rep: 1
[1] 4.375666
-----
test$rep: 2
[1] 4.444215
-----
test$rep: 3
[1] 4.37456
```

Task 17 : Fit the model to the experimental data. •

Now we try fit the model, as if we did not know the parameters. Starting values are from the experimenter's experience. Here we say zero fertilizer gives no yield, the increment is 0.1, and the maximum fertilizer that will give any result is 50.

```
> m.lrp <- nls(y ~ f.lrp(x, a, b, t.x), data = test, start = list(a = 0,
+   b = 0.1, t.x = 50), trace = T, control = list(warnOnly = T,
+   minFactor = 1/2048))

29.55894 :   0.0  0.1 50.0
8.9431  :   2.04540088  0.04595019 60.87229119
0.8166921 :   2.02080700  0.04742582 74.01871968
0.8123844 :   2.01121237  0.04790556 73.54241857
0.8123836 :   2.01121237  0.04790556 73.54718832

> summary(m.lrp)

Formula: y ~ f.lrp(x, a, b, t.x)

Parameters:
      Estimate Std. Error t value Pr(>|t|)
a    2.011212   0.055984   35.92  <2e-16 ***
b    0.047906   0.001338   35.80  <2e-16 ***
t.x  73.547188   1.491585   49.31  <2e-16 ***
---

```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.1502 on 36 degrees of freedom
```

```
Number of iterations to convergence: 4
```

```
Achieved convergence tolerance: 1.609e-09
```

```
> coefficients(m.lrp)
```

```
          a          b          t.x  
2.01121237 0.04790556 73.54718832
```

Task 18 : Evaluate the goodness-of-fit with the unadjusted and adjusted R^2 . •

```
> (RSS.lrp <- sum(residuals(m.lrp)^2))
```

```
[1] 0.8123836
```

```
> (TSS <- sum((test$y - mean(test$y))^2))
```

```
[1] 61.20578
```

```
> (r2.m.lrp <- 1 - (RSS.lrp/TSS))
```

```
[1] 0.986727
```

```
> (n <- dim(test)[1])
```

```
[1] 39
```

```
> (p <- length(coefficients(m.lrp)))
```

```
[1] 3
```

```
> (r2.adj.m.lrp <- 1 - (((n - 1)/(n - p)) * (1 - r2.m.lrp)))
```

```
[1] 0.9859896
```

```
> r2.m.lrp - r2.adj.m.lrp
```

```
[1] 0.0007373882
```

The fit is quite close to the known true values. Note that the summary gives the standard error of each parameter, which can be used for simulation or sensitivity analysis. In this case all “true” parameters are well within one standard error of the estimate.

Task 19 : Plot the experiment with the fitted model and the known model. •

```
> plot(test$y ~ test$x, main = "Linear response and plateau yield response",  
+       xlab = "Fertilizer added", ylab = "Crop yield")
```

```
> (max.yield <- a.0 + b.0 * t.x.0)
```

```
[1] 5.5
```

```

> lines(x = c(0, t.x.0, 120), y = c(a.0, max.yield, max.yield),
+       lty = 2, col = "blue")
> abline(v = t.x.0, lty = 3, col = "blue")
> abline(h = max.yield, lty = 3, col = "blue")
> (max.yield <- coefficients(m.lrp)["a"] + coefficients(m.lrp)["b"] *
+   coefficients(m.lrp)["t.x"])

```

```

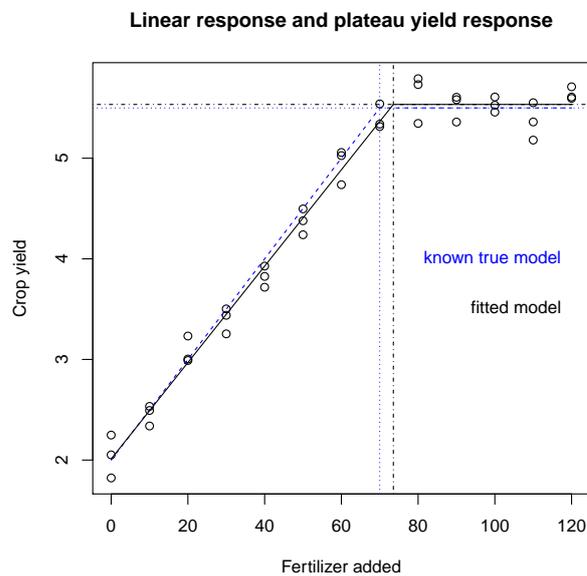
a
5.534531

```

```

> lines(x = c(0, coefficients(m.lrp)["t.x"], 120), y = c(coefficients(m.lrp)["a"],
+   max.yield, max.yield), lty = 1)
> abline(v = coefficients(m.lrp)["t.x"], lty = 4)
> abline(h = max.yield, lty = 4)
> text(120, 4, "known true model", col = "blue", pos = 2)
> text(120, 3.5, "fitted model", col = "black", pos = 2)

```



The AIC of this model can also be calculated and compared to, for example, a second-order power model, often used to model response curves:

```

> m.quad <- lm(y ~ I(x^2) + x, data = test)
> summary(m.quad)

```

Call:

```
lm(formula = y ~ I(x^2) + x, data = test)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.43532	-0.13110	0.00171	0.13853	0.45238

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	1.828e+00	9.339e-02	19.58	< 2e-16 ***
I(x^2)	-3.149e-04	2.904e-05	-10.84	6.82e-13 ***
x	6.907e-02	3.616e-03	19.10	< 2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2251 on 36 degrees of freedom
Multiple R-squared: 0.9702, Adjusted R-squared: 0.9685
F-statistic: 586.1 on 2 and 36 DF, p-value: < 2.2e-16

```
> summary(m.quad)$adj.r.squared  
[1] 0.9685497  
  
> r2.adj.m.lrp  
[1] 0.9859896  
  
> r2.adj.m.lrp - summary(m.quad)$adj.r.squared  
[1] 0.01743988  
  
> AIC(m.quad)  
[1] -0.7694702  
  
> AIC(m.lrp)  
[1] -32.30522
```

The linear-response with plateau model is much superior to the quadratic for this set of observations; this difference is more clearly shown with AIC rather than with the adjusted R^2 .

References

- [1] H Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, Dec 1974. doi: 10.1109/TAC.1974.1100705. 5
- [2] DM Bates and DG Watts. *Nonlinear Regression Analysis and Its Applications*. Wiley, 1988. 1
- [3] R Ihaka and R Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996. 1
- [4] F Leisch. Sweave, part I: Mixing R and L^AT_EX. *R News*, 2(3):28–31, December 2002. URL <http://CRAN.R-project.org/doc/Rnews/>. 1
- [5] F Leisch. *Sweave User's Manual*. TU Wien, Vienna (A), 2.1 edition, 2006. URL <http://www.ci.tuwien.ac.at/~leisch/Sweave>. 1
- [6] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. URL <http://www.R-project.org>. ISBN 3-900051-07-0. 1
- [7] DG Rossiter. *Technical Note: An example of data analysis using the R environment for statistical computing; Version 1.3*. self-published online, 2014. URL http://www.css.cornell.edu/faculty/dgr2/teach/R/R_corregr.pdf. 1
- [8] A-N Spiess and N Neumeyer. An evaluation of R-2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a monte carlo approach. *BMC Pharmacology*, 10:6, Jun 2010. doi: 10.1186/1471-2210-10-6. 9

Index of R Concepts

`^` formula operator, 3

`^` operator, 3

AIC, 6

`as.factor`, 16

`by`, 16

coefficients, 8

`data.frame`, 2

`dim`, 8

I operator, 3

`ifelse`, 14

`length`, 8

`lm`, 1, 3

`lmRob`, 1

`mean`, 16

`nls`, 3, 4, 6, 12, 14

`nls` package, 1

`predict`, 4

`rep`, 15

`rnorm`, 2, 14

`robust` package, 1

`runif`, 2

`set.seed`, 2, 14

`stats` package, 1

`summary`, 4

`table`, 16