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

D G Rossiter
Department of Earth Systems Analysis
International Institute for Geo-information Science & Earth Observation (ITC)
Enschede (NL)

December 15, 2009

Contents

1 Curve fitting

This is a small introduction to curve fitting in the R environment for statistical computing and visualisation [2, 5] and its dialect of the S language. R provides a sophisticated environment, which gives the user more insight and control than provided by commerical or shareware "push the button" programs such as CurveFit.

> **Note:** For an explanation of the R project, including how to obtain and install the software and documentation, see Rossiter [7]. This also contains an extensive discussion of the S language, R graphics, and many statistical methods, as well as a bibliography of texts and references that use R.

> **Note:** The code in these exercises was tested with Sweave [3, 4] on R version 2.10.1 (2009-12-14), `stats` package Version: 2.10.1, running on Mac OS X 10.6.2. So, the text and graphical output you see here was automatically generated and incorporated into LaTeX by running actual code through R and its packages. Then the LaTeX 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 [6].

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) homoscedascity (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 causitive 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 [1] 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.

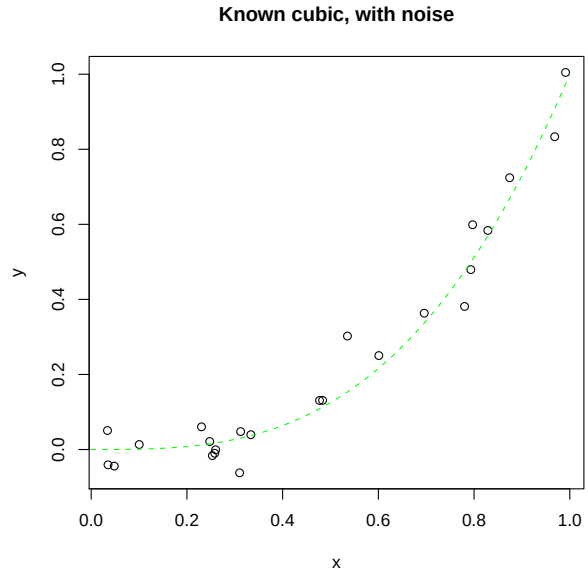
4.1 Fitting a power model

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(520)

> 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  0.1411 0.4925 0.0992 0.0469 0.1131 ...
 $ y: num  0.02586 0.05546 -0.0048 0.0805 0.00764 ...

> 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.

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 ~ I(x^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)

1.539345 :  1
0.2639662 :  1.874984
0.07501804 :  2.584608
0.0673716 :  2.797405
```

```
0.06735321 :  2.808899
0.06735321 :  2.808956

> 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   2.8090     0.1459   19.25 1.11e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.05411 on 23 degrees of freedom

Number of iterations to convergence: 5
Achieved convergence tolerance: 2.163e-07

> summary(m)$coefficients

      Estimate Std. Error  t value     Pr(>|t|)
power 2.808956  0.1459224 19.24966 1.111061e-15
```

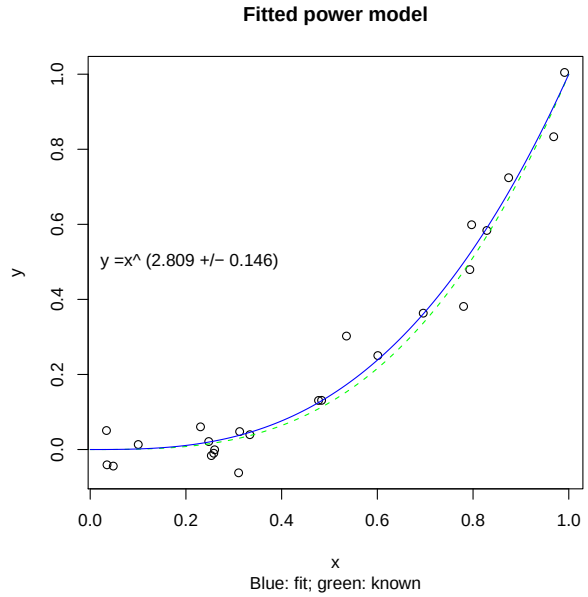
We can see that the estimated power is 2.809 ± 0.146

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 $[0, 0.01, 0.02, \ldots, 0.99, 1]$, 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 quality of the fit. •

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"):

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

[1] 0.06735321

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

[1] 2.219379

> 1 - (RSS.p/TSS)

[1] 0.9696522
```

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] 0.9675771
```

They are hardly distinguishable; the known cubic will not necessarily be better, this depends on the particular simulation.

4.2 Fitting to a functional form

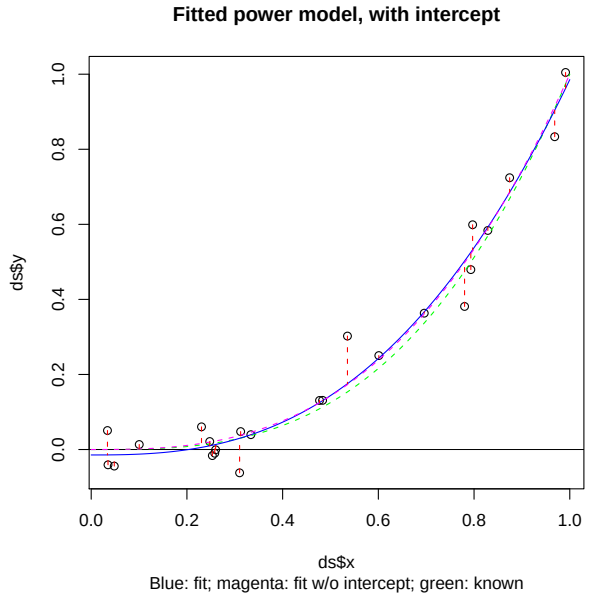
The more general way to use `nls` is to define a **function** for the right-hand side of the non-linear equation. We illustrate for the power model, but without assuming that the curve passes through $(0, 0)$.

Task 6 : 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)

0.2038798 :  0 2
0.06829972 :  -0.01228041  2.55368164
0.06558916 :  -0.01466565  2.65390708
0.06558607 :  -0.01447859  2.65989000
0.06558607 :  -0.01446096  2.66017092
0.06558607 :  -0.01446011  2.66018398

> summary(m.2)

Formula: y ~ rhs(x, intercept, power)

Parameters:
          Estimate Std. Error t value Pr(>|t|)
intercept -0.01446    0.01877   -0.77    0.449
power      2.66018    0.23173   11.48 9.27e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0546 on 22 degrees of freedom

Number of iterations to convergence: 5
Achieved convergence tolerance: 5.288e-07

> 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")
```



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

```
> (RSS.pi <- sum(residuals(m.2)^2))
[1] 0.06558607
> (RSS.p)
[1] 0.06735321
> 1 - (RSS.pi/TSS)
[1] 0.9704485
> 1 - (RSS.p/TSS)
[1] 0.9696522
> 1 - sum((x^3 - y)^2)/TSS
[1] 0.9675771
```

Task 8 : Compare the two models (with and without intercept) with an Analysis of Variance. •

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

```
Analysis of Variance Table

Model 1: y ~ rhs(x, intercept, power)
Model 2: y ~ I(x^power)
  Res.Df Res.Sum Sq Df     Sum Sq F value Pr(>F)
1     22   0.065586
2     23   0.067353 -1 -0.0017671  0.5928 0.4495
```

The `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 (as it in fact is, in this case).

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 9 : 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 $0.1, 0.01, 0.001\ldots$ that is just large enough to bring all the negative values above zero. Here the minimum is:

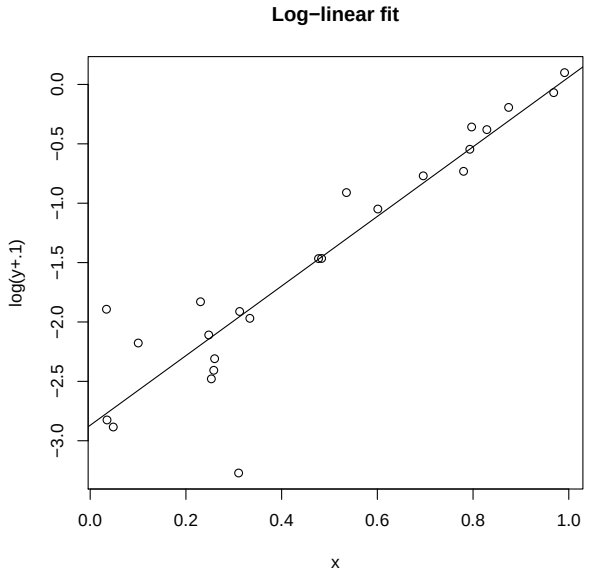
```
> min(y)

[1] -0.06205655
```

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
-1.31083 -0.09495  0.01993  0.07643  0.87749

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -2.8698     0.1458  -19.69 1.85e-15 ***
x             2.9311     0.2613   11.22 1.43e-10 ***
---
```

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

Residual standard error: 0.3874 on 22 degrees of freedom
Multiple R-squared: 0.8512,        Adjusted R-squared: 0.8445
F-statistic: 125.9 on 1 and 22 DF,  p-value: 1.431e-10
```

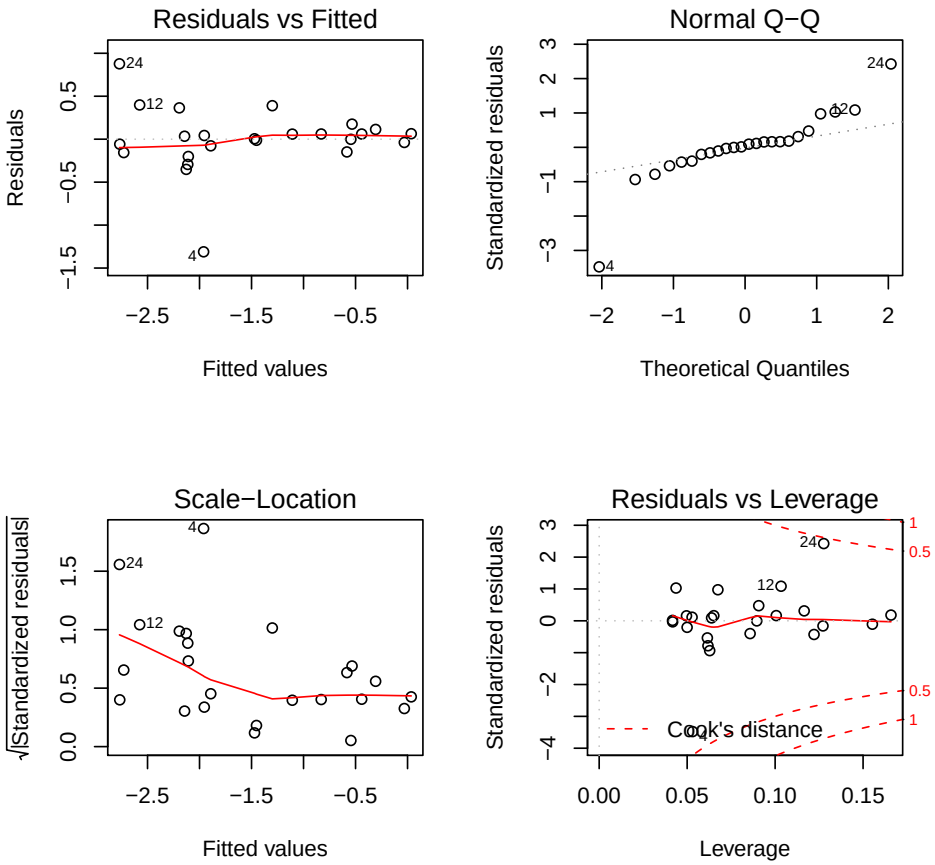
```
> plot(ds$ly ~ ds$x, xlab = "x", ylab = "log(y+.1)", main = "Log-linear fit")
> abline(m.1)
> text(0, 0.4, pos = 4, paste("log(y) = ", round(coefficients(m.1)[1],
+     3), "+", round(coefficients(m.1)[2], 3)))
```



Here the adjusted R^2 is 0.844, but this can not be compared to the non-linear fit, because of 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.

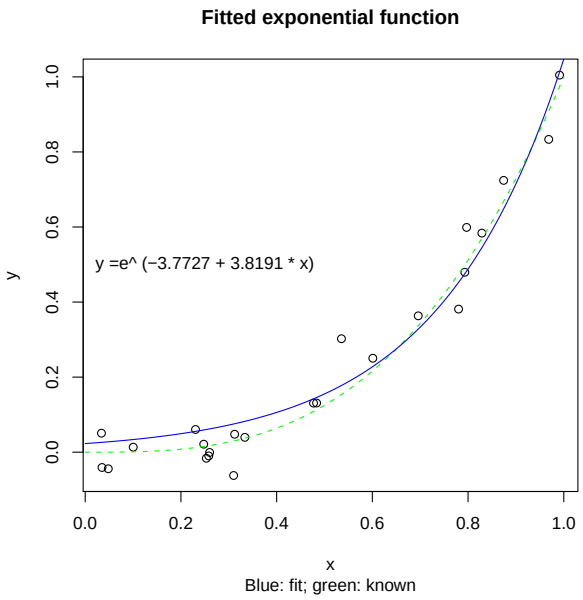
The second way is with `nls`.

Task 10 : 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)

50.47337 :  0 1
5.529667 :  -1.095364  1.427397
0.5618565 :  -2.280596  2.298405
0.1172045 :  -3.390888  3.419927
0.1012589 :  -3.802324  3.855069
0.1011533 :  -3.766443  3.812051
0.1011499 :  -3.773935  3.820469
0.1011498 :  -3.772497  3.818849
0.1011498 :  -3.772775  3.819162
0.1011498 :  -3.772722  3.819102

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

```
   Estimate Std. Error   t value     Pr(>|t|)
a -3.772722  0.2516577 -14.99148 4.968731e-13
b  3.819102  0.2805423  13.61328 3.400880e-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:

```
> RSS.p

[1] 0.06735321

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

[1] 0.1011498

> TSS

[1] 2.219379

> 1 - RSS.p/TSS

[1] 0.9696522

> 1 - RSS.e/TSS

[1] 0.9544243
```

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

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 necessariy 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 11 : 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 12 : 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}. •

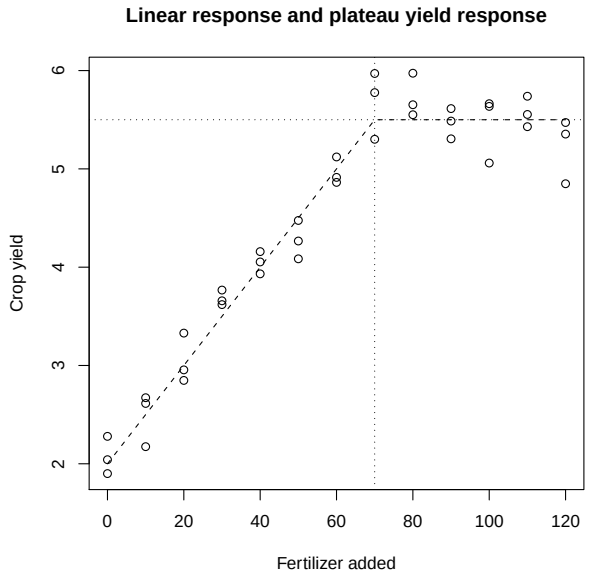
In nature there are always random factors; we account for this by adding normally-distributed noise with the `rnorm` function. Again we use `set.seed` so your results will be the same, but you can experiment with other random values.

```
> 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 <- 619
> 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  1.9 2.67 2.96 3.77 3.93 ...
```

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 13 : 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)
```



Note: 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  1.9 2.67 2.96 3.77 3.93 ...
 $ 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$rep: 1
[1] 4.422042
------------------------------------------------------
test$rep: 2
[1] 4.488869
------------------------------------------------------
test$rep: 3
[1] 4.405799
```

Task 14 : 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))

32.56051 :   0.0  0.1 50.0
8.951927 :   2.10193890  0.04665956 60.07251046
2.265017 :   2.09041476  0.04735101 72.64469433
2.187349 :   2.04412992  0.04966525 69.20738193
2.169194 :   2.06727234  0.04850813 70.75843813
2.149853 :   2.05570113  0.04908669 70.04640466
2.149082 :   2.05497793  0.04912285 70.00347614
2.149081 :   2.05489318  0.04912709 69.99845311
2.149070 :   2.05496255  0.04912362 70.00308909
2.149026 :   2.05492024  0.04912574 70.00057914
2.149018 :   2.05490970  0.04912626 69.99995416

> summary(m.lrp)

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

Parameters:
     Estimate Std. Error t value Pr(>|t|)
a    2.054927   0.091055   22.57   <2e-16 ***
b    0.049125   0.002177   22.57   <2e-16 ***
t.x 70.001112   2.254942   31.04   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2443 on 36 degrees of freedom

Number of iterations till stop: 10
Achieved convergence tolerance: 0.1495
Reason stopped: step factor 0.000244141 reduced below 'minFactor' of 0.000488281

> coefficients(m.lrp)
```

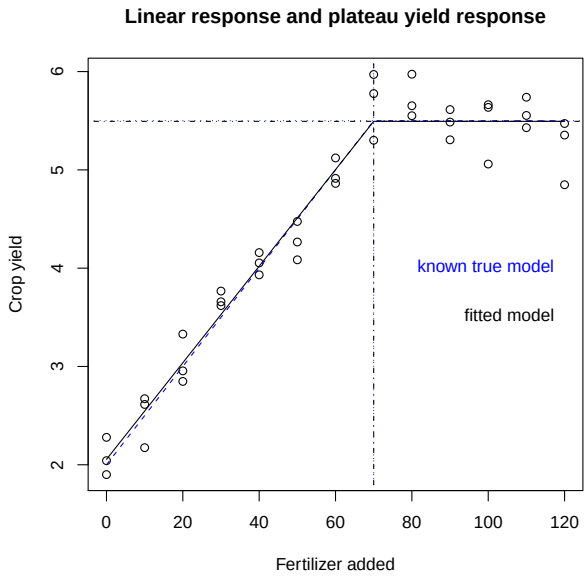
```
        a         b        t.x
2.0549270 0.0491254 70.0011125
```

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 15 : 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
5.493759

> 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)
```



References

[1] D. M. Bates and D. G. Watts. *Nonlinear Regression Analysis and Its Applications*. Wiley, 1988. 2

[2] 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

[3] F Leisch. *Sweave User's Manual*. TU Wein, Vienna (A), 2.1 edition, 2006. URL http://www.ci.tuwien.ac.at/~leisch/Sweave. 1

[4] F Leisch. Sweave, part I: Mixing R and LaTeX. *R News*, 2(3):28–31, December 2002. URL http://CRAN.R-project.org/doc/Rnews/. 1

[5] 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

[6] D G Rossiter. *Technical Note: An example of data analysis using the R environment for statistical computing*. International Institute for Geo-information Science & Earth Observation (ITC), Enschede (NL), 0.9 edition, 2008. URL http://www.itc.nl/personal/rossiter/teach/R/R_corregr.pdf. 1

[7] D G Rossiter. *Introduction to the R Project for Statistical Computing for use at ITC*. International Institute for Geo-information Science & Earth Observation (ITC), Enschede (NL), 3.6 edition, 2009. URL http://www.itc.nl/personal/rossiter/teach/R/RIntro_ITC.pdf. 1

Index of R Concepts