10-Nonlinear Regression

2026-01-20

1 Introduction

The following examples demonstrate how to perform non-linear regression in R. This is quite different from linear regression, not only because the regression functions show a curve, but also due to the applied numerical techniques. While in the linear case, the coefficients of the regression line can be calculated directly (analytically) by solving a system of linear equations, iterative numerical optimization needs to be used instead.

This means that the coefficients are approximated step by step until convergence, beginning with start values specified by the user. There is no guarantee that a globally optimal solution can be found.

The following examples are intended as a starting point, the last example (logistic growth) is left as an exercise.

1.1 Exponential growth

The first example shows an exponentially growing data set that is fitted by nonlinear least squares (`nls`). Here, the exponential function is given directly at the right hand side of the formula and the start values for the parameters are specified in `pstart`. An optional argument `trace = TRUE` is set to watch the progress of iteration.

Function `predict` can be used to create dense x and y-values (here: `x1` and `y1`) for a smooth curve.

The statistical results of the regression are displayed with `summary`: parameters (a, b) of the curve, standard errors and the correlation between a and b. Note the optional `correlation=TRUE` argument. High correlations can indicate problems in model fitting, especially due to so called non-identifiablility of the parameter set.

Here everything went through even with high correlation, because the data do not vary much around the exponential curve.

Finally, the nonlinear coefficient of determination R^2 can be calculated from the variances of the residuals and the original data:

```r
x <- 1:10
y <- c(1.6, 1.8, 2.1, 2.8, 3.5, 4.1, 5.1, 5.8, 7.1, 9.0)
plot(x, y)

pstart <- c(a = 1, b = 1)
model_fit <- nls(y ~ a * exp(b * x), start = pstart, trace = TRUE)
```

```r
x1 <- seq(1, 10, 0.1)
y1 <- predict(model_fit, data.frame(x = x1))
lines(x1, y1, col = "red")

summary(model_fit, correlation = TRUE)

## R squared
1 - var(residuals(model_fit))/var(y)
```

1.2 Regression model as user-defined function

Instead of putting the regression model directly into `nls` it is also possible to use a user-defined function, that we call `f` here for example. This approach is especially useful for more complicated regression models:

```r
f <- function(x, a, b) {a * exp(b * x)}

plot(x, y)

pstart <- c(a = 1, b = 1)
model_fit <- nls(y ~ f(x, a, b), start = pstart)

x1 <- seq(1, 10, 0.1)
y1 <- predict(model_fit, data.frame(x = x1))
lines(x1, y1, col = "red")
summary(model_fit, correlation = TRUE)
1 - var(residuals(model_fit))/var(y)
```

2 Michaelis-Menten kinetics

The following data from an experiment show the dependency of carbon uptake by micro-organisms on substrate concentration:

```r
# Substrate ug C /l
S  <- c(25, 25, 10, 10, 5, 5, 2.5, 2.5, 1.25, 1.25)
# Microbial uptake rate ug C /(l*h)
V <- c(0.0998, 0.0948, 0.076, 0.0724, 0.0557,
        0.0575, 0.0399, 0.0381, 0.017, 0.0253)
plot(S, V)
```

The plot shows a typical saturation at high substrate concentrations and the most common function used for such dependencies is the Michaelis-Menten kinetics, sometimes also called Monod function.

```r
## Michaelis-Menten kinetics
f <- function(S, Vm, K) {
 Vm * S/(K + S)
}
```

Fitting the Michaelis-Menten function is essentially similar to the former example:

```r
pstart <- c(Vm = max(V), K = 5)
model_fit   <- nls(V ~ f(S, Vm, K), start = pstart, trace = TRUE)
plot(S, V, xlim = c(0, 25), ylim = c(0, 0.1))
x1 <- seq(0, 25, length = 100)
lines(x1, predict(model_fit, data.frame(S = x1)), col = "red")
summary(model_fit)
(Rsquared <- 1 - var(residuals(model_fit))/var(V))
```

..., but especially for Michaelis-Menten, **R** contains also a so-called "self-start" function, so that it is not necessary to define an own "f" and, even more useful, not necessary to provide start values for the parameters:

```r
model_fit <- nls(V ~ SSmicmen(S, Vm, K), trace = TRUE)
plot(S, V, xlim = c(0, 25), ylim = c(0, 0.1))
x1 <- seq(0, 25, length = 100)
lines(x1, predict(model_fit, data.frame(S = x1)), col = "red")
summary(model_fit)
(Rsquared <- 1 - var(residuals(model_fit))/var(V))
```

3 Logistic growth of a cyanobacteria population

The following data describe the growth of a blue-green algae culture in a batch culture experiment:

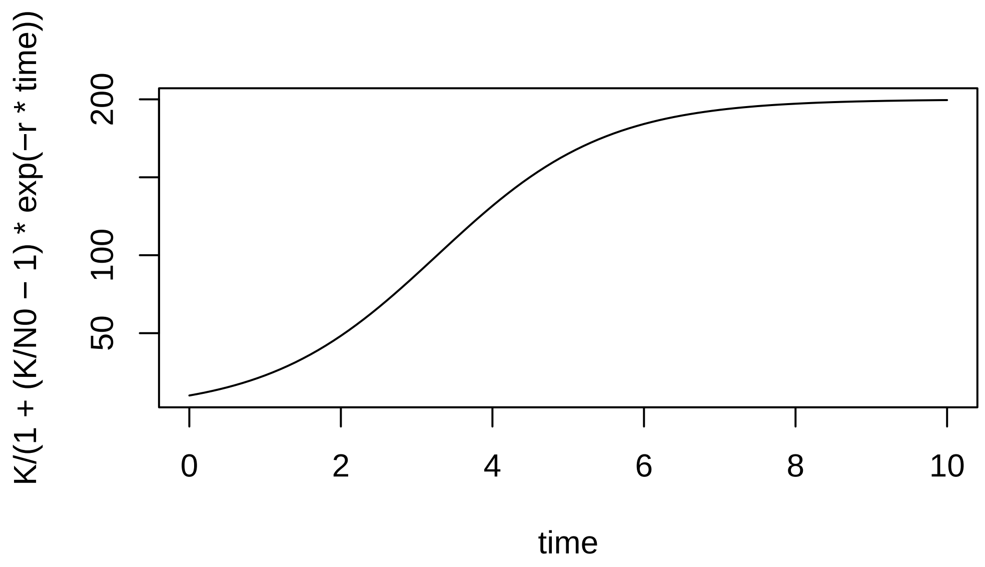
```r
# time (t)
x <- c(0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20)

# Algae cell counts (per ml)
y <- c(0.88, 1.02, 1.43, 2.79, 4.61, 7.12,
       6.47, 8.16, 7.28, 5.67, 6.91) * 1e6
plot(x, y)
```

The growth is exponential in the beginning and then approaches a saturation. The points show a sigmoid shape and the usual function applied for this type of growth is the "logistic equation", with two parameters r (maximum growth rate) and K (carrying capacity). This is the function after which the r and K strategists are named in ecology.

The following example shows the general behavior of the logistic and the formula in R notation:

```r
N0 <- 10
r  <- 0.9
K  <- 200
time <- seq(0, 10, length=101)
plot(time, K /(1 + (K/N0 - 1) * exp(-r * time)), type="l")
```



4 Exercise

Fit a logistic curve to the cell counts of the growth experiment. Note however, that the large cell counts (10^6) can lead to numerical errors. Here it may be neccessary to adapt the `control` options of `nls` or, much simpler, to rescale the `y`-variable to more convenient values.