16

Nonlinear curve fitting

Curve fitting problems occur in many scientific areas. The typical case is that you wish to fit the relation between some response y and a one-dimensional predictor x, by adjusting a (possibly multidimensional) parameter β. That is,

$$y = f(x; \beta) + \text{error}$$

in which the "error" term is usually assumed to contain independent normally distributed terms with a constant standard deviation σ. The class of models can be easily extended to multivariate x and somewhat less easily to models with nonconstant error variation, but we settle for the simple case.

Chapter 6 described the special case of a linear relation

$$y = \beta_0 + \beta_1 x + \text{error}$$

and we discussed the fitting of polynomials by including quadratic and higher-order terms in Section 12.1. There are other techniques, notably trigonometric regression and spline regression, that can also be formulated in linear form and handled by software for multiple regression analysis like `lm`.

However, sometimes linear methods are inadequate. The common case is that you have a priori knowledge of the form of the function. This may come from theoretical analysis of an underlying physical and chemical

P. Dalgaard, *Introductory Statistics with R*,
DOI: 10.1007/978-0-387-79054-1_16,

system, and the parameters of the relation have a specific meaning in that theory.

The method of least squares makes good sense even when the relation between data and parameters is not linear. That is, we can estimate β by minimizing

$$\text{SSD}(\beta) = \sum (y - f(x;\beta))^2$$

There is no explicit formula for the location of the minimum, but the minimization can be performed numerically by algorithms that we describe only superficially here. This general technique is also known as nonlinear regression analysis. For an in-depth treatment of the topic, a standard reference is Bates and Watts (1988).

If the model is "well-behaved" (to use a deliberately vague term), then the model can be approximated by a linear model in the vicinity of the optimum, and it then makes sense to calculate approximate standard errors for the parameter estimates.

Most of the available optimization algorithms build on the same idea of linearization; i.e.,

$$y - f(x;\beta+\delta) \approx y - f(x;\beta) + Df\delta$$

in which Df denotes the *gradient matrix* of derivatives of f with respect to β. This effectively becomes a design matrix of a linear model, and you can proceed from a starting guess at β to find an approximate least squares fit of δ. Then you replace β by $\beta + \delta$ and repeat until convergence. Variations on this basic algorithm include numerical computation of Df and techniques to avoid instability if the starting guess is too far from the optimum.

To perform the optimization in R, you can use the `nls` function, which is broadly similar to `lm` and `glm`.

16.1 Basic usage

In this section, we use a simulated data set just so that we know what we are doing. The model is a simple exponential decay.

```
> t <- 0:10
> y <- rnorm(11, mean=5*exp(-t/5), sd=.2)
> plot(y ~ t)
```

The simulated data can be seen in Figure 16.1.

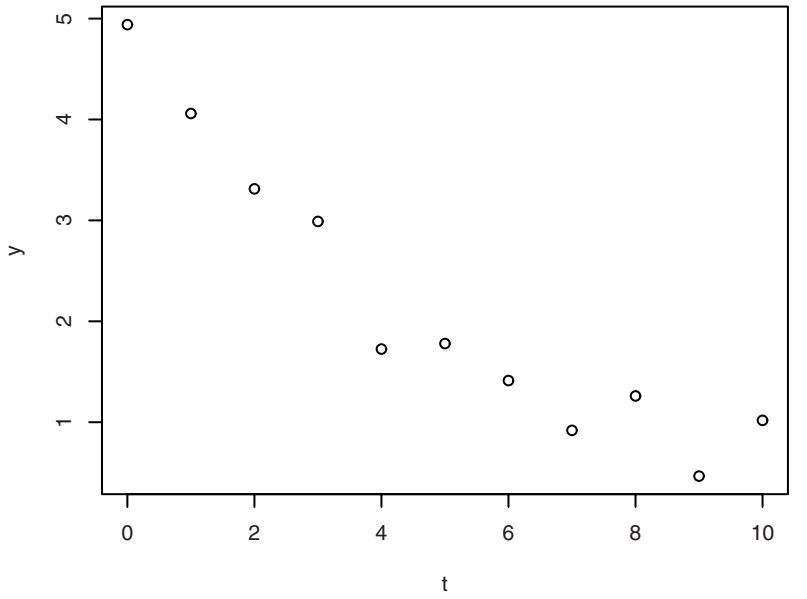

Figure 16.1. Simulated exponential decay.

We now fit the model to data using `nls`. Unlike `lm` and `glm`, the model formula for `nls` does *not* use the special codings for linear terms, grouping factors, interactions, etc. Instead, the right-hand side is an explicit expression to calculate the expected value of the left-hand side. This can depend on external variables as well as the parameters, so we need to specify which is which. The simplest way to do this is to specify a named vector (or a named list) of starting values.

```
> nlsout <- nls(y ~ A*exp(-alpha*t), start=c(A=2, alpha=0.05))
> summary(nlsout)

Formula: y ~ A * exp(-alpha * t)

Parameters:
      Estimate Std. Error t value Pr(>|t|)
A      4.97204    0.21766   22.84 2.80e-09 ***
alpha  0.20793    0.01572   13.23 3.35e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2805 on 9 degrees of freedom

Number of iterations to convergence: 5
```

```
Achieved convergence tolerance: 2.223e-06
```

Notice that `nls` treats `t` as a variable and not a parameter because it is not mentioned in the `start` argument. Whenever the fitting algorithm needs to evaluate `A * exp(-alpha * t)`, `t` is taken from the variable in the global environment, whereas `A` and `alpha` are varied by the algorithm.

The general form of the output is quite similar to that of `glm`, so we shall not dwell too long upon it. One thing that might be noted is that the t test and p-value stated for each parameter are tests for a hypothesis that the parameter is *zero*, which is often quite meaningless for nonlinear models.

16.2 Finding starting values

In the previous section, we had quite fast convergence, even though the initial guess of parameters was (deliberately) rather badly off. Unfortunately, things are not always that simple; convergence of nonlinear models can depend critically on having good starting values. Even when the algorithm is fairly robust, we at least need to get the order of magnitude right.

Methods for obtaining starting values will most often rely on an analysis of the functional form; common techniques involve transformation to linearity and the estimation of "landmarks" such asymptotes, maximum points, and initial slopes.

To illustrate this, we again consider the Juul data. This time we focus on the relation between age and height. To obtain a reasonably homogeneous data set, we look at males only and subset the data to the ages between 5 and 20.

```
> attach(subset(juul2, age<20 & age>5 & sex==1))
> plot(height ~ age)
```

A plot of the data is shown in Figure 16.2. The plot looks linear over a large portion of its domain, but there is some levelling off at the right end and of course it is basic human biology that we stop growing at some point in the later teens.

The Gompertz curve is often used to describe growth. It can be expressed in the following form:

$$y = \alpha e^{-\beta e^{-\gamma x}}$$

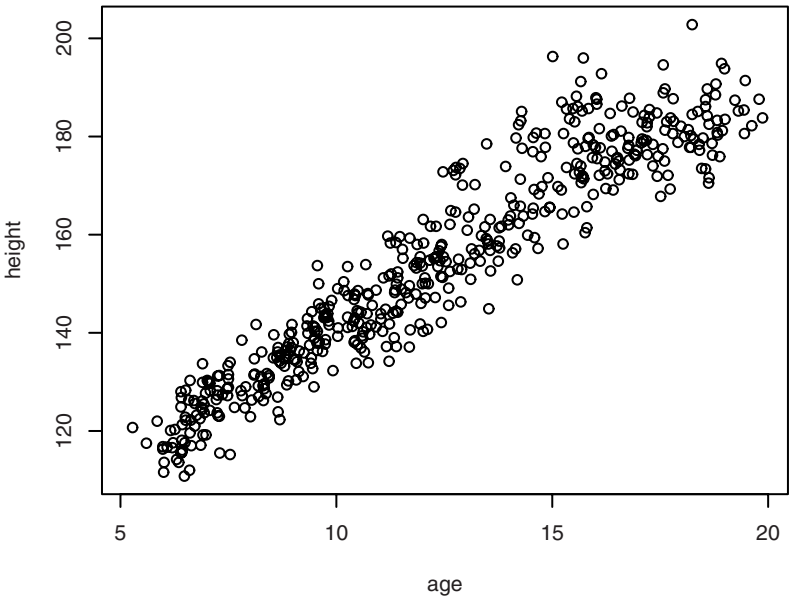

Figure 16.2. Relationship between age and height in `juul2` data set.

The curve has a sigmoid shape, approaching a constant level α as x increases and (in principle) zero for large negative x. The β and γ parameters determine the location and steepness of the transition.

To obtain starting values for a nonlinear fit, one approach is to notice that the relation between y and x is something like log-log linear. Specifically, we can rewrite the relation as

$$\log y = \log \alpha - \beta e^{-\gamma x}$$

which we may rearrange and take logarithms on both sides again, yielding

$$\log(\log \alpha - \log y) = \log \beta - \gamma x$$

That means that if we can come up with a guess for α, then we can guess the two other parameters by a linear fit to transformed data. Since α is the asymptotic maximum, a guess of $\alpha = 200$ could be reasonable. With this guess, we can make a plot that should show an approximately linear relationship ($\log 200 \approx 5.3$):

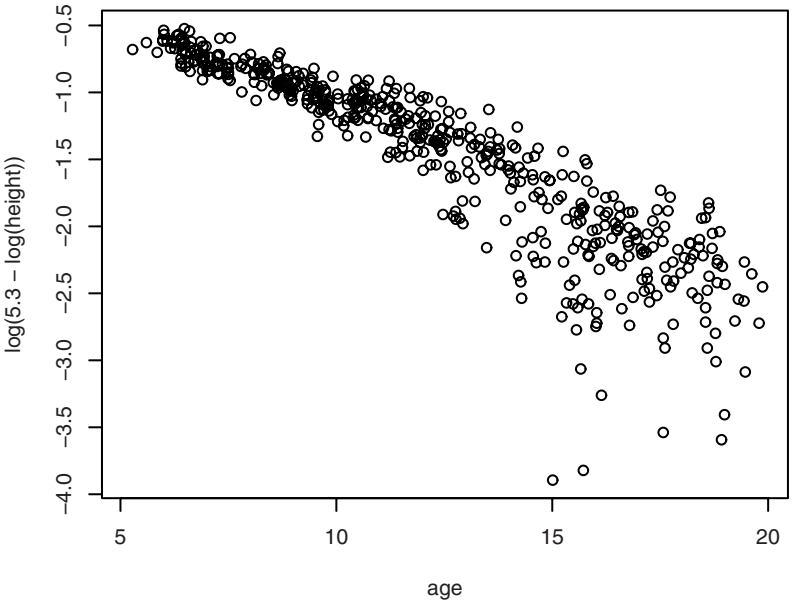

Figure 16.3. Linearized plot of the Gompertz relation when assuming $\alpha \approx 200$.

```
> plot(log(5.3-log(height))~age)
Warning message:
In log(5.3 - log(height)) : NaNs produced
```

Notice that we got a warning that an `NaN` (Not a Number) value was produced. This is because one individual was taller than 200 cm, and we therefore tried to take the logarithm of a negative value. The linearized plot shows a clearly nonconstant variance and probably also some asymmetry of the residual distribution, so the assumptions for linear regression analysis are clearly violated. However, it is good enough for our purpose, and a linear fit gives

```
> lm(log(5.3-log(height))~age)

Call:
lm(formula = log(5.3 - log(height)) ~ age)

Coefficients:
(Intercept)          age
     0.4200      -0.1538

Warning message:
In log(5.3 - log(height)) : NaNs produced
```

Accordingly, an initial guess of the parameters is

$$\log \alpha = 5.3$$
$$\log \beta = 0.42$$
$$\gamma = 0.15$$

Supplying these guesses to `nls` and fitting the Gompertz curve yields

```
> fit <- nls(height~alpha*exp(-beta*exp(-gamma*age)),
+            start=c(alpha=exp(5.3),beta=exp(0.42),gamma=0.15))
> summary(fit)

Formula: height ~ alpha * exp(-beta * exp(-gamma * age))

Parameters:
       Estimate Std. Error t value Pr(>|t|)
alpha 2.428e+02  1.157e+01  20.978   <2e-16 ***
beta  1.176e+00  1.892e-02  62.149   <2e-16 ***
gamma 7.903e-02  8.569e-03   9.222   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.811 on 499 degrees of freedom

Number of iterations to convergence: 8
Achieved convergence tolerance: 5.283e-06
  (3 observations deleted due to missingness)
```

The final estimates are quite a bit different from the starting values. This reflects the crudeness of the estimation methods used for the starting values. In particular, we used transformations that were based on the mathematical form of the function but did not take the structure of the error variation into account. Also, the important parameter α was obtained by eye.

Looking at the fitted model, however, it is not reassuring that the final estimate for α suggests that boys would continue growing until they are 243 cm tall (for readers in nonmetric countries, that is almost eight feet!). Possibly, the Gompertz curve is just not a good fit for these data.

We can overlay the original data with the fitted curve as follows (Figure 16.4)

```
> plot(age, height)
> newage <- seq(5,20,length=500)
> lines(newage, predict(fit,newdata=data.frame(age=newage)),lwd=2)
```

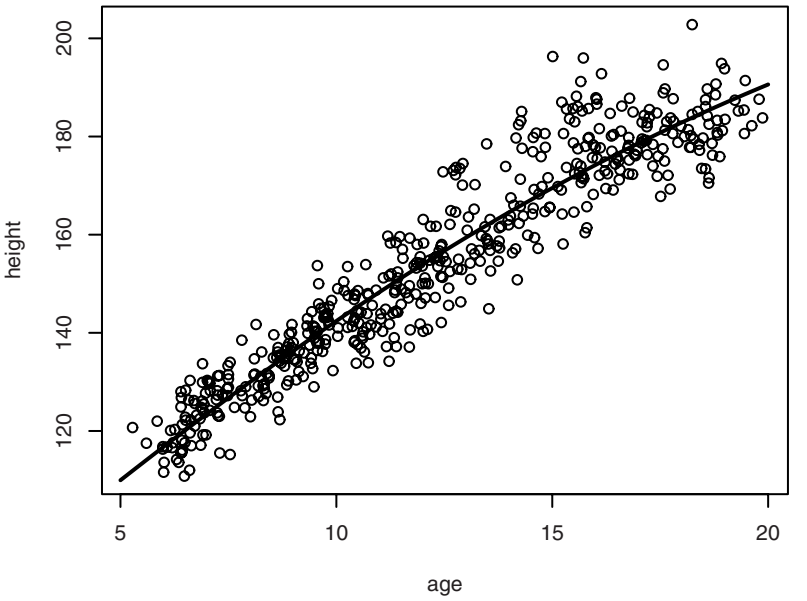

Figure 16.4. The fitted Gompertz curve.

The plot suggests that there is a tendency for the dispersion to increase with increasing fitted values, so we attempt a log-scale fit. This can be done expediently by transforming both sides of the model formula.

```
>
> fit <- nls(log(height)~log(alpha*exp(-beta*exp(-gamma*age))),
+ start=c(alpha=exp(5.3),beta=exp(.12),gamma=.12))
> summary(fit)

Formula: log(height) ~ log(alpha * exp(-beta * exp(-gamma * age)))

Parameters:
       Estimate Std. Error t value Pr(>|t|)
alpha 255.97694   15.03920  17.021   <2e-16 ***
beta    1.18949    0.02971  40.033   <2e-16 ***
gamma   0.07033    0.00811   8.673   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.04307 on 499 degrees of freedom

Number of iterations to convergence: 8
Achieved convergence tolerance: 2.855e-06
  (3 observations deleted due to missingness)
```

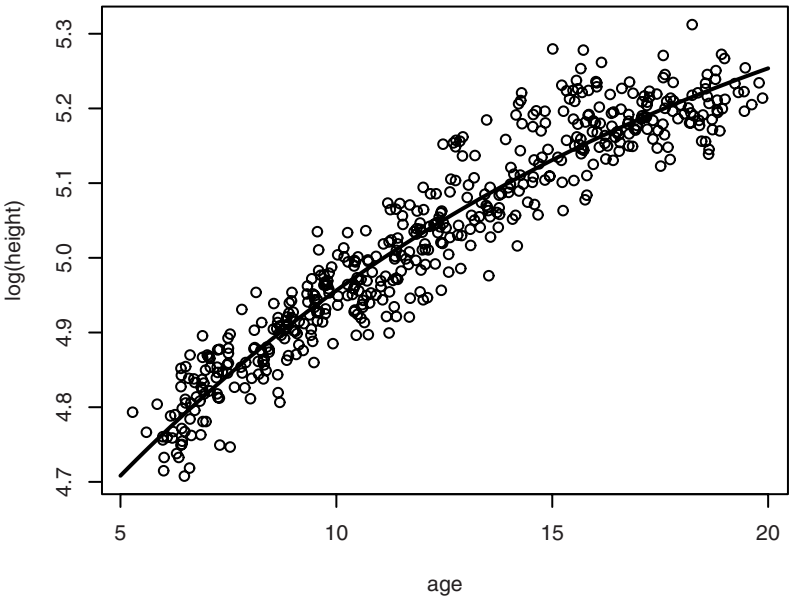

Figure 16.5. Fitted Gompertz curve on log scale.

```
> plot(age, log(height))
> lines(newage, predict(fit,newdata=data.frame(age=newage)),lwd=2)
```

On the log-scale plot (Figure 16.5), the distribution around the curve appears to be more stable. The parameter estimates did not change much, although the maximum height is now increased by a further 13 cm (5 inches) and the γ parameter is reduced to compensate.

Closer inspection of the plots (whether on log scale or not), however, reveals that the Gompertz curve tends to overshoot the data points at the right end, where a much flatter curve would fit the data in the range from 15 years upwards. Although visually there is a nice overall fit, this is not hard to obtain for a three-parameter family of curves, and the Gompertz curves seem unable to fit the characteristic patterns of human growth.

16.3 Self-starting models

Finding starting values is an art rather than a craft, but once a stable method has been found, it may be reasonable to assume that will apply to most data sets from a given model. `nls` allows the nice feature that the procedure for calculating starting values can be embodied in the expressions that are used on the right-hand side of the model formula. Such functions are by convention named starting with "SS", and R 2.6.2 comes with 10 of these built-in. In particular, there is in fact an `SSgompertz` function, so we could have saved ourselves much of the trouble of the previous section by just writing

```
> summary(nls(height~SSgompertz(age, Asym, b2, b3)))

Formula: height ~ SSgompertz(age, Asym, b2, b3)

Parameters:
      Estimate Std. Error t value Pr(>|t|)
Asym 2.428e+02  1.157e+01   20.98   <2e-16 ***
b2   1.176e+00  1.892e-02   62.15   <2e-16 ***
b3   9.240e-01  7.918e-03  116.69   <2e-16 ***
...
```

Notice, though, that the parameterization is different: The parameter `b3` is actually e^{γ}, whereas the two other parameters are recognized as α and β.

One minor drawback of self-starting models is that you cannot just transform them if you want to see if the model fits better on, for example, a log scale. In other words, this fails:

```
> nls(log(height) ~ log(SSgompertz(age, Asym, b2, b3)))
Error in nlsModel(formula, mf, start, wts) :
  singular gradient matrix at initial parameter estimates
Calls: nls -> switch -> nlsModel
In addition: Warning message:
In nls(log(height) ~ log(SSgompertz(age, Asym, b2, b3))) :
  No starting values specified for some parameters.
Intializing 'Asym', 'b2', 'b3' to '1.'.
Consider specifying 'start' or using a selfStart model
```

The error message means, in essence, that the self-start machinery is turned off, so `nls` tries a wild guess, setting all parameters to 1, and then fails to converge from that starting point.

Using expression `log(SSgompertz(age, Asym, b2, b3))` to compute the expected value of `log(height)` is not a problem (in itself). We can take the starting values from the untransformed fit but this is still not enough to make things work.

There is a hitch: `SSgompertz` returns a *gradient attribute* along with the fitted values. This is the derivative of the fitted value with respect to each of the model parameters. This speeds up the convergence process for the original model but is plainly wrong for the transformed model, where it causes convergence failure. We could patch this up by calculating the correct gradient, but it is expedient simply to discard the attribute by taking `as.vector`.

```
> cf <- coef(nls(height ~ SSgompertz(age, Asym, b2, b3)))
> summary(nls(log(height) ~
+                log(as.vector(SSgompertz(age,Asym, b2, b3))),
+             start=as.list(cf)))

Formula: log(height) ~ log(as.vector(SSgompertz(age, Asym, b2, b3)))

Parameters:
      Estimate Std. Error t value Pr(>|t|)
Asym 2.560e+02  1.504e+01   17.02   <2e-16 ***
b2   1.189e+00  2.971e-02   40.03   <2e-16 ***
b3   9.321e-01  7.559e-03  123.31   <2e-16 ***
...
```

It is possible to write your own self-starting models. It is not hard once you have some experience with R programming, but we shall not go into details here. The essence is that you need two basic items: the model expression and a function that calculates the starting values. You must ensure that these adhere to some formal requirements, and then a constructor function `selfStart` can be called to create the actual self-starting function.

16.4 Profiling

We discussed profiling before in connection with `glm` and logistic regression in Section 13.3. For nonlinear regression, there are some slight differences: The function that is being profiled is not the likelihood function but the sum of squared deviations, and the approximate confidence intervals are based on the t distribution rather than the normal distribution. Also, the plotting method does not by default use the signed version of the profile, just the square root of the difference in the sum of squared deviations.

Profiling is designed to eliminate *parameter curvature*. The same model can be formulated using different parameterizations (such as when Gompertz curves could be defined using γ or $b_3 = e^{\gamma}$). The choice of parameterization can have a substantial influence on whether the distribution of the

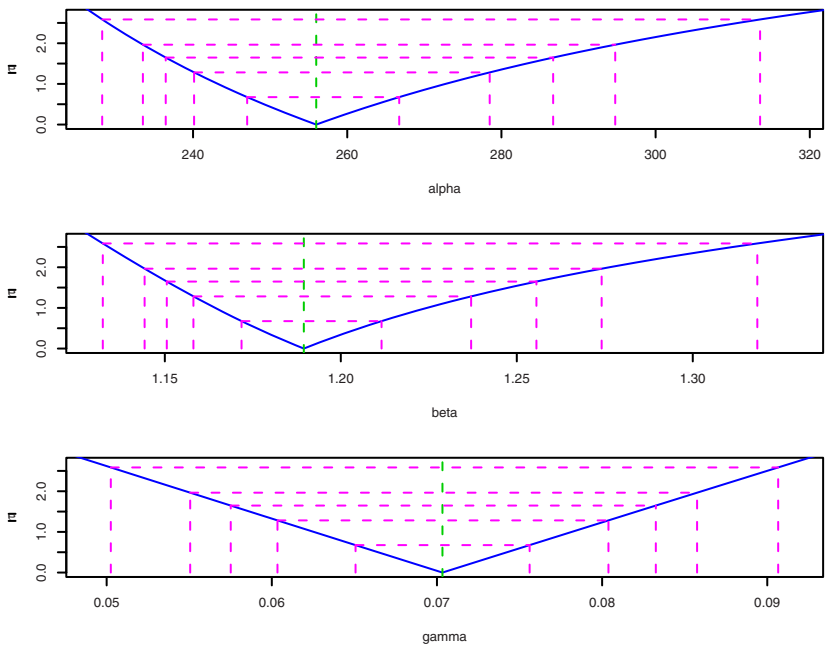

Figure 16.6. Parameter profiles of the Gompertz fit on log scale.

estimate is approximately normal or not, and this in turn means that the use of symmetric confidence intervals based on the standard errors from the model summary can be misleading. Profile-based confidence intervals do not depend on parameterization — if you transform a parameter, the ends of the confidence interval are just transformed in the same way.

There is, however, also *intrinsic curvature* of the models. This describes how far the model is from an approximating linear model. This kind of curvature is independent of parameterization and is harder to adjust for than parameter curvature. The effect of intrinsic curvature is that the t distribution used for the calculation of profile-based confidence intervals is not exactly the right distribution to use. Experience suggests that this effect is usually much smaller than the distortions caused by parameter curvature.

For the Gompertz fit (after log transformation), we get the plots shown in Figure 16.6.

```
> par(mfrow=c(3,1))
> plot(profile(fit))
```

The plots show that there is a marked curvature for the α and β parameters, reflected in the curved and asymmetric profiles, whereas the γ profile is more linear and symmetric. This is also seen when comparing the profile-based confidence intervals with those of `confint.default`, which uses the normal approximation and the approximate standard errors.

```
> confint(fit)
Waiting for profiling to be done...
              2.5%        97.5%
alpha 233.49688706 294.76696435
beta    1.14429894   1.27416518
gamma   0.05505754   0.08575007
> confint.default(fit)
              2.5 %       97.5 %
alpha 226.50064512 285.45322721
beta    1.13125578   1.24772846
gamma   0.05443819   0.08622691
```

16.5 Finer control of the fitting algorithm

The Juul example that has been used in this chapter has been quite benign because there are a large number of observations and an objective function that is relatively smooth as a function of the parameters. However, convergence problems easily come up in less nice examples. Nonlinear optimization is simply a tricky topic, to which we have no chance of doing justice in this short chapter. The algorithms have several parameters that can be adjusted in order to help convergence, but since we are not describing the algorithms, it is hardly possible to give more than a feeling for what can be done.

The possibility of supplying a gradient of the fitted curve with respect to parameters was mentioned earlier. If the curve is given by a simple mathematical expression, then the `deriv` function can even be used to generate the gradient automatically. If a gradient is not available, then the algorithm will estimate it numerically; in practice, this often turns out to be equally fast.

The `nls` function features a `trace` argument that, if set to `TRUE`, allows you to follow the parameters and the SSD iteration by iteration. This is sometimes useful to get a handle on what is happening, for instance whether the algorithm is making unreasonably large jumps. To actually modify the behaviour, there is a single `control` argument, which can be set to the return value of `nls.control`, which in turn has arguments to set iteration limits and tolerances (and more).

You can switch out the entire fitting method by using the `algorithm` argument. Apart from the default algorithm, this allows the settings `"plinear"` and `"port"`. The former allows models of the form

$$y = \sum_i \alpha_i f_i(x; \beta_i)$$

that are partially linear since the α_i can be determined by multiple linear regression if the β_i are considered fixed. To specify models with more than one term, you let the expression on the right-hand side of the model formula return a matrix instead of a vector. The latter algorithm uses a routine from the PORT library from Lucent Technologies; this in particular allows you to set contraints on parameters by using the `upper` and `lower` arguments to `nls`.

It should be noted that all the available algorithms operate under the implicit assumption that the SSD(β) is fairly smooth and well behaved, with a well-defined global minimum and no other local minima nearby. There are cases where this assumption is not warranted. In such cases, you might attack the minimization problem directly using the `optim` function.

16.6 Exercises

16.1 Try fitting the Gompertz model for girls in the Juul data. How would you go about testing whether the same model fits both genders?

16.2 The `philion` data contain four small-sample EC50 experiments that are somewhat tricky to handle. We suggest the model $y = y_{\max}/(1 + (x/\beta)^{\alpha})$. It may be useful to transform y by the square root since the data are counts, and this stabilizes the variance of the Poisson distribution. Consider how to obtain starting values for the model, and fit it with `nls`. The `"port"` algorithm seems more stable for these data. For profiling and confidence intervals, it seems to help if you set the `alphamax` argument to 0.2.

16.3 (Theoretical) Continuing with the `philion` data, consider what happens if you modify the model to be $y = y_{\max}/(1 + x/\beta)^{\alpha}$.