20
Non-linear Regression

Sometimes we have a mechanistic model for the relationship between y and x, and we want to estimate the parameters and standard errors of the parameters of a specific non-linear equation from data. Some frequently used non-linear models are shown in Table 20.1. What we mean in this case by 'non-linear' is not that the relationship is curved (it was curved in the case of polynomial regressions, but these were linear models), but that the relationship cannot be linearized by transformation of the response variable or the explanatory variable (or both). Here is an example: it shows jaw bone length as a function of age in deer. Theory indicates that the relationship is an asymptotic exponential with three parameters:

$$y = a - b\mathrm{e}^{-cx}.$$

In R, the main difference between linear models and non-linear models is that we have to tell R the exact nature of the equation as part of the model formula when we use non-linear modelling. In place of `lm` we write `nls` (this stands for 'non-linear least squares'). Then, instead of `y~x`, we write `y~a-b*exp(-c*x)` to spell out the precise nonlinear model we want R to fit to the data.

The slightly tedious thing is that R requires us to specify initial guesses for the values of the parameters a, b and c (note, however, that some common non-linear models have 'self-starting' versions in R which bypass this step; see p. 675). Let's plot the data to work out sensible starting values. It always helps in cases like this to work out the equation's 'behaviour at the limits' – that is to say, to find the values of y when $x = 0$ and when $x = \infty$ (p. 195). For $x = 0$, we have $\exp(-0)$ which is 1, and $1 \times b = b$, so $y = a - b$. For $x = \infty$, we have $\exp(-\infty)$ which is 0, and $0 \times b = 0$, so $y = a$. That is to say, the asymptotic value of y is a, and the intercept is $a - b$.

```
deer<-read.table("c:\\temp\\jaws.txt",header=T)
attach(deer)
names(deer)
```

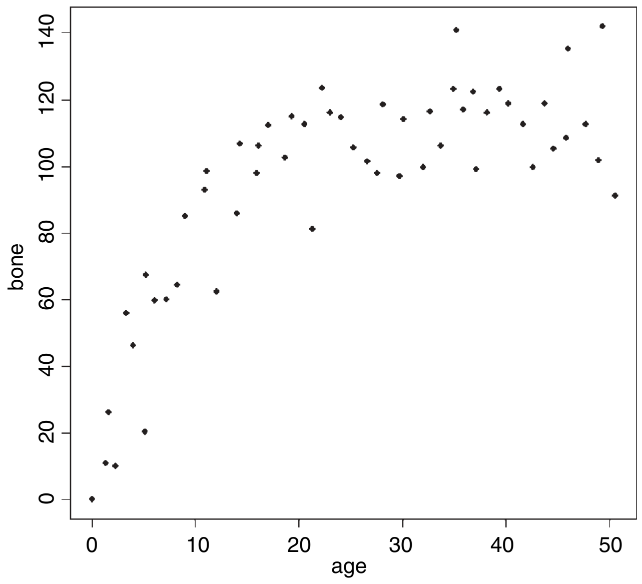
```
[1] "age" "bone"
```

```
plot(age,bone,pch=16)
```

The R Book Michael J. Crawley
© 2007 John Wiley & Sons, Ltd. ISBN: 978-0-470-51024-7

Table 20.1. Useful non-linear functions.

Name	Equation
Asymptotic functions	
Michaelis–Menten	$y = \frac{ax}{1 + bx}$
2-parameter asymptotic exponential	$y = a(1 - e^{-bx})$
3-parameter asymptotic exponential	$y = a - be^{-cx}$
S-shaped functions	
2-parameter logistic	$y = \frac{e^{a+bx}}{1 + e^{a+bx}}$
3-paramerter logistic	$y = \frac{a}{1 + be^{-cx}}$
4-parameter logistic	$y = a + \frac{b - a}{1 + e^{(c-x)/d}}$
Weibull	$y = a - be^{-(cx^d)}$
Gompertz	$y = ae^{-be^{-cx}}$
Humped curves	
Ricker curve	$y = axe^{-bx}$
First-order compartment	$y = k\exp(-\exp(a)x) - \exp(-\exp(b)x)$
Bell-shaped	$y = a\exp(-\lvert bx\rvert^2)$
Biexponential	$y = ae^{bx} - ce^{-dx}$



Inspection suggests that a reasonable estimate of the asymptote is $a \approx 120$ and intercept ≈ 10, so $b = 120 - 10 = 110$. Our guess at the value of c is slightly harder. Where the curve is rising most steeply, jaw length is about 40 where age is 5. Rearranging the equation gives

$$c = -\frac{\log((a-y)/b)}{x} = -\frac{\log(120-40)/110)}{5} = 0.063\,690\,75.$$

Now that we have the three parameter estimates, we can provide them to R as the starting conditions as part of the nls call like this:

```
model<-nls(bone~a-b*exp(-c*age),start=list(a=120,b=110,c=0.064))
summary(model)
```

```
Formula: bone~a - b * exp(-c * age)

Parameters:
   Estimate  Std. Error  t value  Pr(>|t|)
a  115.2528      2.9139    39.55   < 2e-16  ***
b  118.6875      7.8925    15.04   < 2e-16  ***
c    0.1235      0.0171     7.22  2.44e-09  ***

Residual standard error: 13.21 on 51 degrees of freedom
```

All the parameters appear to be significantly different from zero at $p < 0.001$. Beware, however. This does not necessarily mean that all the parameters need to be retained in the model. In this case, $a = 115.2528$ with standard error 2.9139 is clearly not significantly different from $b = 118.6875$ with standard error 7.8925 (they would need to differ by more than 2 standard errors to be significant). So we should try fitting the simpler two-parameter model

$$y = a(1 - \mathrm{e}^{-cx}).$$

```
model2<-nls(bone~a*(1-exp(-c*age)),start=list(a=120,c=0.064))
anova(model,model2)
```

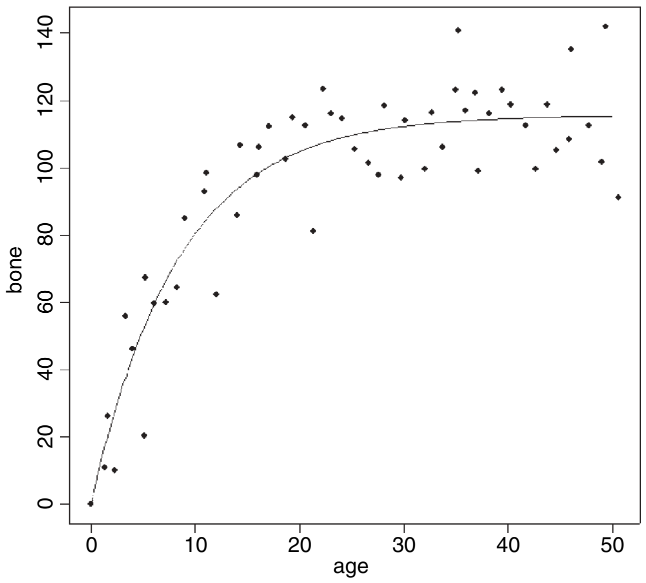
```
Analysis of Variance Table

Model 1: bone~a - b * exp(-c * age)
Model 2: bone~a * (1 - exp(-c * age))

   Res.Df  Res.Sum Sq  Df  Sum Sq  F value  Pr(>F)
1      51      8897.3
2      52      8929.1  −1   −31.8   0.1825   0.671
```

Model simplification was clearly justified ($p = 0.671$), so we accept the two-parameter version, model2, as our minimal adequate model. We finish by plotting the curve through the scatterplot. The age variable needs to go from 0 to 50 in smooth steps:

```
av<-seq(0,50,0.1)
```

and we use predict with model2 to generate the predicted bone lengths:

```
bv<-predict(model2,list(age=av))
lines(av,bv)
```



The parameters of this curve are obtained from model2:

```
summary(model2)
```

```
Parameters:
     Estimate   Std. Error   t value   Pr(>|t|)
a  115.58056      2.84365    40.645     < 2e-16   ***
c    0.11882      0.01233     9.635    3.69e-13   ***

Residual standard error: 13.1 on 52 degrees of freedom
```

which we could write as $y = 115.58(1 - e^{-0.1188x})$ or as $y = 115.58(1 - \exp(-0.1188x))$ according to taste or journal style. If you want to present the standard errors as well as the parameter estimates, you could write: 'The model $y = a(1 - \exp(-bx))$ had $a = 115.58 \pm 2.84$ (1 standard error) and $b = 0.1188 \pm 0.0123$ (1 standard error, $n = 54$) and explained 84.6% of the total variation in bone length'. Note that because there are only two parameters in the minimal adequate model, we have called them a and b (rather than a and c as in the original formulation).

Comparing Michaelis–Menten and Asymptotic Exponential

Model choice is always an important issue in curve fitting. We shall compare the fit of the asymptotic exponential (above) with a Michaelis–Menten with parameter values estimated from the same deer jaws data. As to starting values for the parameters, it is clear that a reasonable estimate for the asymptote would be 100 (this is a/b; see p. 202). The curve passes close to the point $(5, 40)$ so we can guess a value of a of $40/5 = 8$ and hence $b = 8/100 = 0.08$. Now use nls to estimate the parameters:

```
(model3<-nls(bone~a*age/(1+b*age),start=list(a=8,b=0.08)))
```

```
Nonlinear regression model
  model: bone~a * age/(1 + b * age)
```

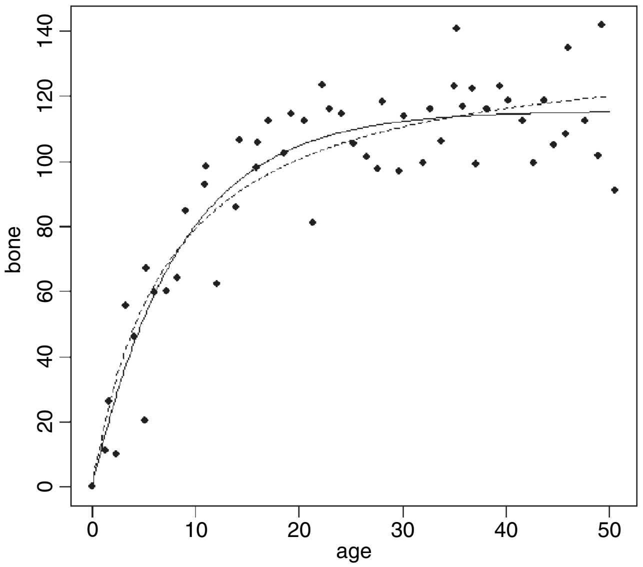
```
   data: parent.frame()
        a          b
18.7253859  0.1359640

residual sum-of-squares: 9854.409
```

Finally, we can add the line for Michaelis–Menten to the original plot. You could draw the best-fit line by transcribing the parameter values

```
ymm<-18.725*av/(1+0.13596*av)
lines(av,ymm,lty=2)
```

Alternatively, you could use predict with the model name, using list to allocate *x* values to age:

```
ymm<-predict(model3, list(age=av))
lines(av,ymm,lty=2)
```



You can see that the asymptotic exponential (solid line) tends to get to its asymptote first, and that the Michaelis–Menten (dotted line) continues to increase. Model choice, therefore would be enormously important if you intended to use the model for prediction to ages much greater than 50 months.

Generalized Additive Models

Sometimes we can see that the relationship between *y*and *x* is non-linear but we don't have any theory or any mechanistic model to suggest a particular functional form (mathematical equation) to describe the relationship. In such circumstances, generalized additive models GAMs are particularly useful because they fit non-parametric smoothers to the data without requiring us to specify any particular mathematical model to describe the non-linearity (background and more examples are given in Chapter 18).

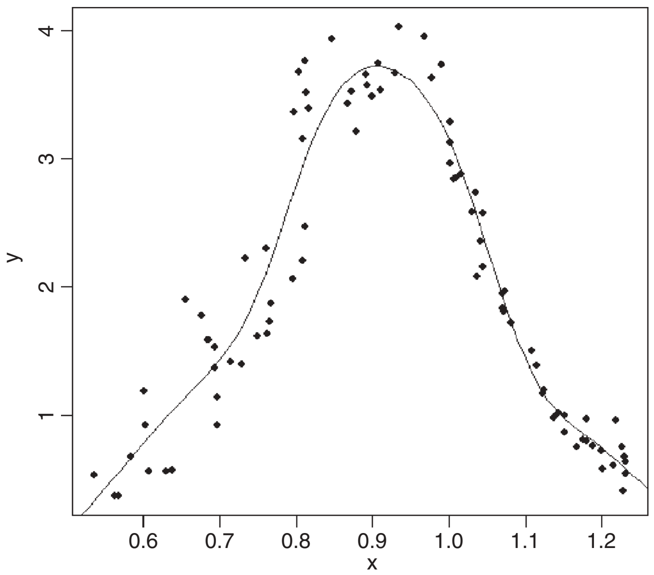
```
humped<-read.table("c:\\temp\\hump.txt",header=T)
attach(humped)
names(humped)

[1] "y" "x"

plot(x,y,pch=16)
library(mgcv)
```

The model is specified very simply by showing which explanatory variables (in this case just x) are to be fitted as smoothed functions using the notation y~s(x):

```
model<-gam(y~s(x))
```

Now we can use predict in the normal way to fit the curve estimated by gam:

```
xv<-seq(0.5,1.3,0.01)
yv<-predict(model,list(x=xv))
lines(xv,yv)
```



```
summary(model)

Family: gaussian
Link function: identity

Formula:
y ~ s(x)

Parametric coefficients:
              Estimate  Std. Error  t value  Pr(>|t|)
(Intercept)    1.95737    0.03446     56.8    <2e-16  ***
```

```
Approximate significance of smooth terms:
        edf  Est.rank      F  p-value
s(x)  7.452         9  110.0   <2e-16  ***

R-sq.(adj) = 0.919   Deviance explained = 92.6%
GCV score = 0.1156   Scale est. = 0.1045  n = 88
```

Fitting the curve uses up 7.452 degrees of freedom (i.e. it is quite expensive) but the resulting fit is excellent and the model explains more than 92% of the deviance in *y*.

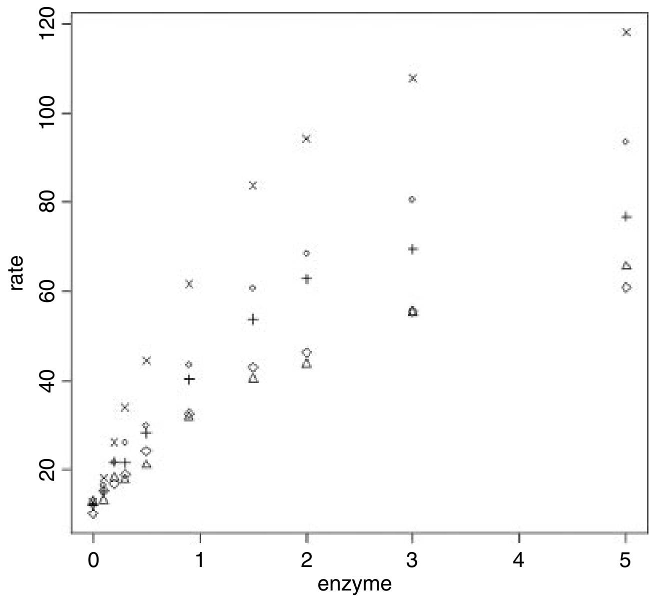
Grouped Data for Non-linear Estimation

Here is a dataframe containing experimental results on reaction rates as a function of enzyme concentration for five different bacterial strains, with reaction rate measured just once for each strain at each of ten enzyme concentrations. The idea is to fit a family of five Michaelis–Menten functions with parameter values depending on the strain.

```
reaction<-read.table("c:\\temp\\reaction.txt",header=T)
attach(reaction)
names(reaction)
```

```
[1] "strain" "enzyme" "rate"
```

```
plot(enzyme,rate,pch=as.numeric(strain))
```



Clearly the different strains will require different parameter values, but there is a reasonable hope that the same functional form will describe the response of the reaction rate of each strain to enzyme concentration.

```
library(nlme)
```

The function we need is nlsList which fits the same functional form to a group of subjects (as indicated by the ‘given’ operator |):

```
model<-nlsList(rate~c+a*enzyme/(1+b*enzyme)|strain,
                                        data=reaction,start=c(a=20,b=0.25,c=10))
```

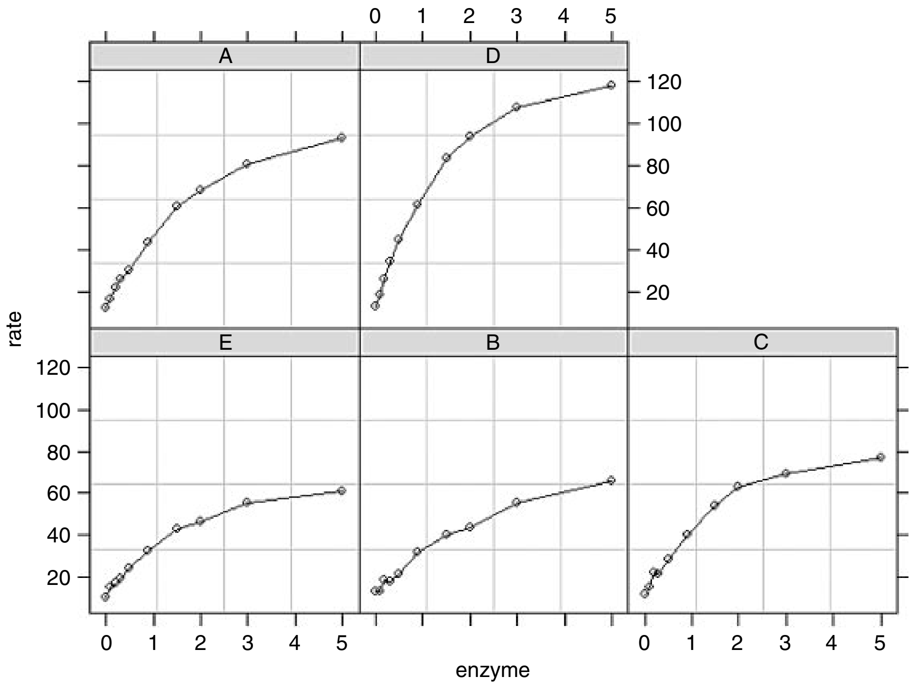
Note the use of the groupedData style formula rate~enzyme | strain.

```
summary(model)

Call:
   Model: rate ~ c + a * enzyme/(1 + b * enzyme) | strain
    Data: reaction

Coefficients:
   a
   Estimate  Std. Error    t value      Pr(>|t|)
A  51.79746   4.093791  12.652686  1.943005e-06
B  26.05893   3.063474   8.506335  2.800344e-05
C  51.86774   5.086678  10.196781  7.842353e-05
D  94.46245   5.813975  16.247482  2.973297e-06
E  37.50984   4.840749   7.748767  6.462817e-06
   b
    Estimate  Std. Error    t value      Pr(>|t|)
A  0.4238572  0.04971637   8.525506  2.728565e-05
B  0.2802433  0.05761532   4.864041  9.173722e-04
C  0.5584898  0.07412453   7.534479  5.150210e-04
D  0.6560539  0.05207361  12.598587  1.634553e-05
E  0.5253479  0.09354863   5.615774  5.412405e-05
   c
   Estimate  Std. Error    t value      Pr(>|t|)
A  11.46498   1.194155   9.600916  1.244488e-05
B  11.73312   1.120452  10.471780  7.049415e-06
C  10.53219   1.254928   8.392663  2.671651e-04
D  10.40964   1.294447   8.041768  2.909373e-04
E  10.30139   1.240664   8.303123  4.059887e-06

Residual standard error: 1.81625 on 35 degrees of freedom
```

There is substantial variation from strain to strain in the values of *a* and *b*, but we should test whether a model with a common intercept of, say, 11.0 might not fit equally well.

The plotting is made much easier if we convert the dataframe to a grouped data object:

```
reaction<-groupedData(rate~enzyme|strain,data=reaction)
library(lattice)
plot(reaction)
```



This plot has just joined the dots, but we want to fit the separate non-linear regressions. To do this we fit a non-linear mixed-effects model with nlme, rather than use nlsList:

```
model<-nlme(rate~c+a*enzyme/(1+b*enzyme),fixed=a+b+c~1,
        random=a+b+c~1|strain,data=reaction,start=c(a=20,b=0.25,c=10))
```

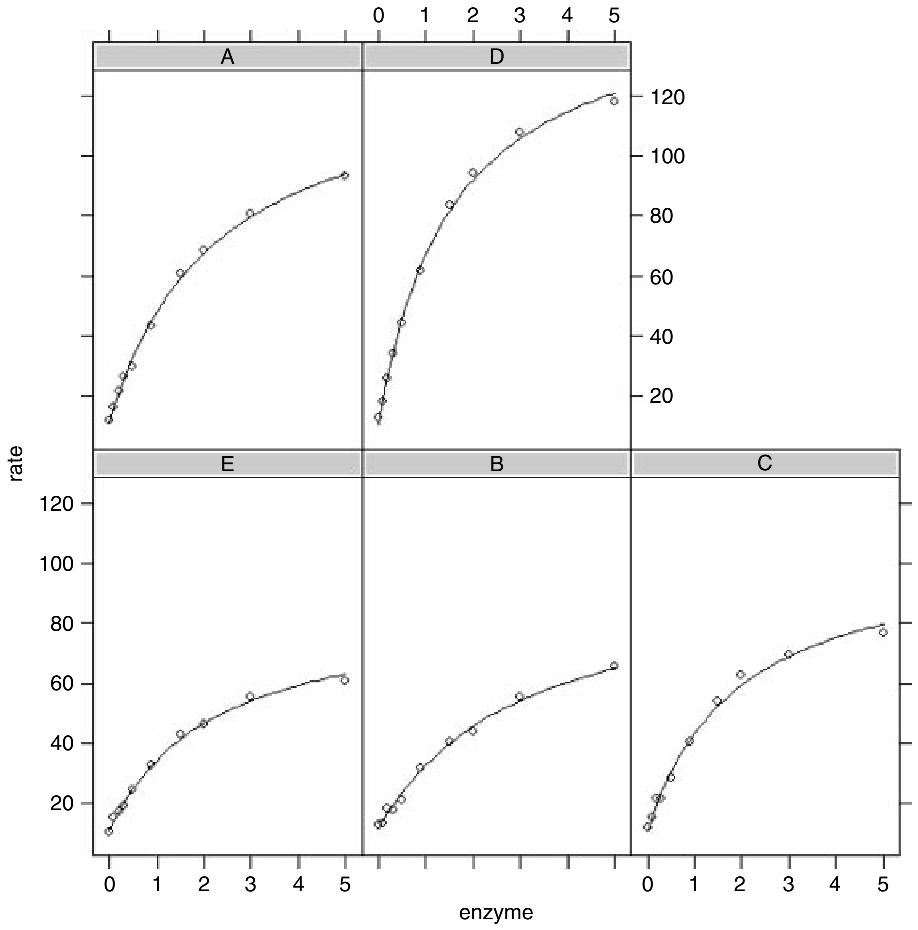
Now we can employ the very powerful augPred function to fit the curves to each panel:

```
plot(augPred(model))
```

Here is the summary of the non-linear mixed model:

```
summary(model)
```

```
Nonlinear mixed-effects model fit by maximum likelihood
Model: rate ~ c + a * enzyme/(1 + b * enzyme)
 Data: reaction
     AIC        BIC      logLik
253.4805   272.6007   −116.7403

Random effects:
  Formula: list(a ~ 1, b ~ 1, c ~ 1)
  Level: strain
  Structure: General positive-definite, Log-Cholesky parametrization

           StdDev      Corr
a          22.9151522  a        b
b           0.1132367   0.876
c           0.4230049  -0.537   -0.875
Residual    1.7105945
```



```
Fixed effects: a + b + c ~ 1
      Value   Std.Error  DF    t-value  p-value
a  51.59880   10.741364  43   4.803747        0
b   0.47665    0.058786  43   8.108295        0
c  10.98537    0.556452  43  19.741797        0

Correlation:
   a      b
b   0.843
c  -0.314  -0.543

Standardized Within-Group Residuals:
       Min          Q1         Med          Q3         Max
-1.79186411  -0.65635614  0.05687126  0.74269371  2.02721778

Number of Observations: 50
Number of Groups: 5
```

The fixed effects in this model are the means of the parameter values. To see the separate parameter estimates for each strain use coef:

```
coef(model)
          a         b        c
E  34.09051  0.4533456  10.81722
B  28.01273  0.3238688  11.54813
C  49.63892  0.5193772  10.67189
A  53.20468  0.4426243  11.23613
D  93.04715  0.6440384  10.65348
```

Note that the rows of this table are no longer in alphabetical order but sequenced in the way they appeared in the panel plot (i.e. ranked by their asymptotic values). The parameter estimates are close to, but not equal to, the values estimated by nlsList (above) as a result of 'shrinkage' in the restricted maximum likelihood estimates (see p. 631).

Non-linear Time Series Models (Temporal Pseudoreplication)

The previous example was a designed experiment in which there was no pseudoreplication. However, we often want to fit non-linear models to growth curves where there is temporal pseudoreplication across a set of subjects, each providing repeated measures on the response variable. In such a case we shall want to model the temporal autocorrelation.

```
nl.ts<-read.table("c:\\temp\\nonlinear.txt",header=T)
attach(nl.ts)
names(nl.ts)

[1] "time" "dish" "isolate" "diam"

growth<-groupedData(diam~time|dish,data=nl.ts)
```

Here, we model the temporal autocorrelation as first-order autoregessive, corAR1():

```
model<-nlme(diam~a+b*time/(1+c*time),
fixed=a+b+c~1,
random=a+b+c~1,
data=growth,
correlation=corAR1(),
start=c(a=0.5,b=5,c=0.5))


summary(model)


Nonlinear mixed-effects model fit by maximum likelihood
Model: diam ~ a + b * time/(1 + c * time)
Data: growth

     AIC       BIC     logLik
129.7694  158.3157  -53.88469

Random effects:
Formula: list(a ~ 1, b ~ 1, c ~ 1)
Level: dish
Structure: General positive-definite, Log-Cholesky parametrization
```

```
           StdDev     Corr
a          0.1014474  a       b
b          1.2060379  -0.557
c          0.1095790  -0.958  0.772
Residual   0.3150068
```

```
Correlation Structure: AR(1)
Formula: ~1 | dish
Parameter estimate(s):
        Phi
-0.03344944
Fixed effects: a + b + c ~ 1
       Value    Std.Error  DF   t-value  p-value
a   1.288262  0.1086390  88  11.85819        0
b   5.215251  0.4741954  88  10.99810        0
c   0.498222  0.0450644  88  11.05578        0
```

```
Correlation:
   a      b
b  -0.506
c  -0.542  0.823
```

```
Standardized Within-Group Residuals:
       Min           Q1           Med          Q3          Max
-1.74222882  -0.64713657  -0.03349834  0.70298805  2.24686653
```

```
Number of Observations: 99
Number of Groups: 9
```

```
coef(model)
```

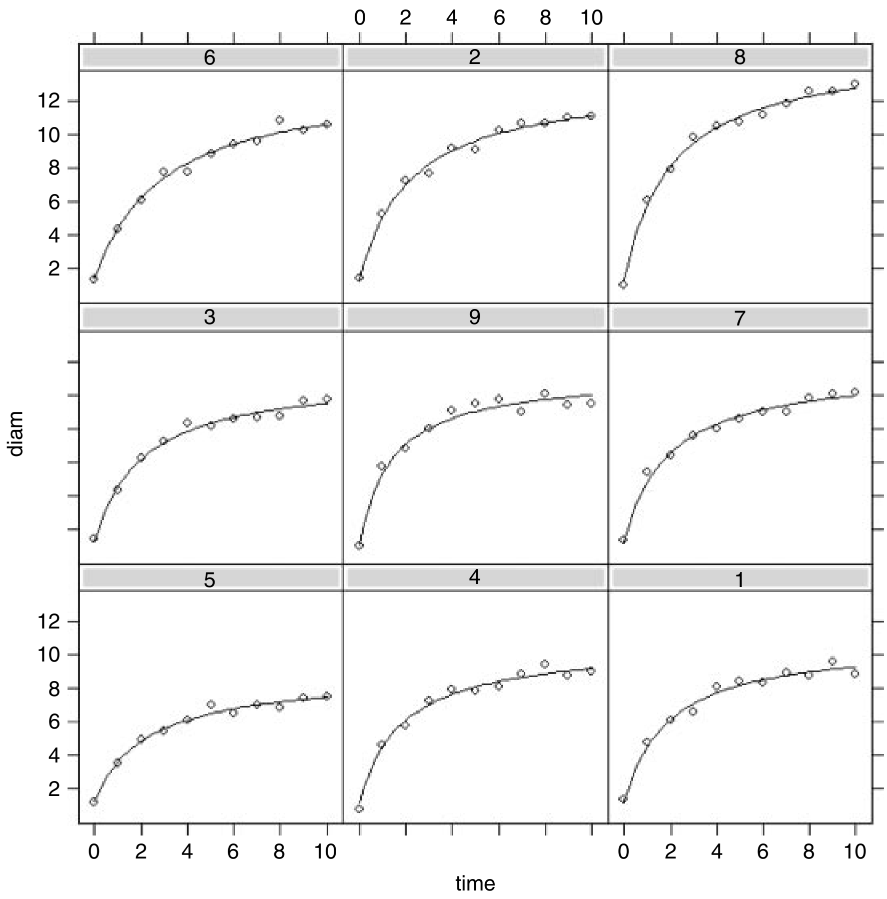
```
          a         b          c
5  1.288831  3.348752  0.4393772
4  1.235632  5.075219  0.5373948
1  1.252725  5.009538  0.5212435
3  1.285847  4.843221  0.4885947
9  1.111135  7.171305  0.7061053
7  1.272570  5.361570  0.5158167
6  1.435784  4.055242  0.3397510
2  1.348523  5.440494  0.4553723
8  1.363310  6.631920  0.4803384
```

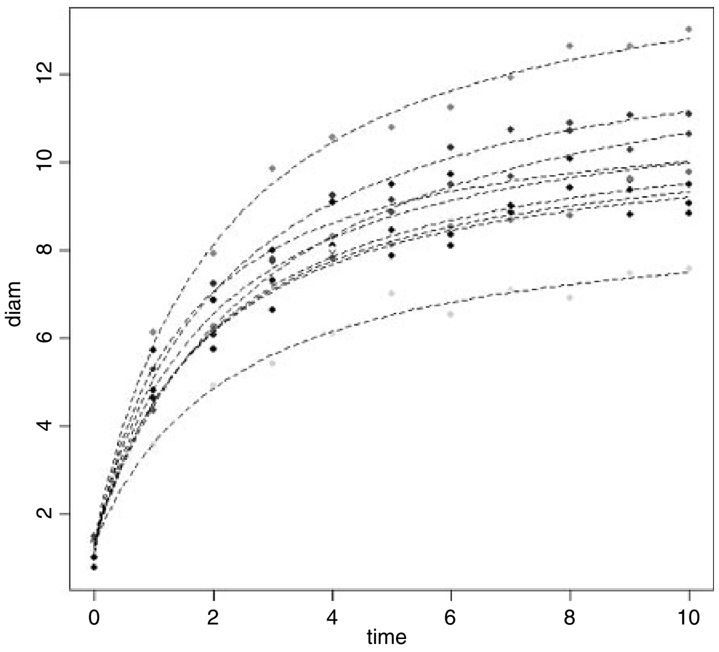
It could not be simpler to plot the family of non-linear models in a panel of scatterplots. We just use augPred like this:

```
plot(augPred(model))
```

To get all the curves in a single panel we could use predict instead of augPred:

```
xv<-seq(0,10,0.1)
plot(time,diam,pch=16,col=as.numeric(dish))
sapply(1:9,function(i) lines(xv,predict(model,list(dish=i,time=xv)),lty=2))
```


6
2
8
3
9
7
5
4
1
diam
time
0
2
4
6
8
10
12


diam
time
0
2
4
6
8
10
12

Self-starting Functions

One of the most likely things to go wrong in non-linear least squares is that the model fails because your initial guesses for the starting parameter values were too far off. The simplest solution is to use one of R's 'self-starting' models, which work out the starting values for you automatically. These are the most frequently used self-starting functions:

SSasymp	asymptotic regression model
SSasympOff	asymptotic regression model with an offset
SSasympOrig	asymptotic regression model through the origin
SSbiexp	biexponential model
SSfol	first-order compartment model
SSfpl	four-parameter logistic model
SSgompertz	Gompertz growth model
SSlogis	logistic model
SSmicmen	Michaelis–Menten model
SSweibull	Weibull growth curve model

Self-starting Michaelis–Menten model

In our next example, reaction rate is a function of enzyme concentration; reaction rate increases quickly with concentration at first but asymptotes once the reaction rate is no longer enzyme-limited. R has a self-starting version called SSmicmen parameterized as

$$y = \frac{ax}{b+x},$$

where the two parameters are a (the asymptotic value of y) and b (which is the x value at which half of the maximum response, $a/2$, is attained). In the field of enzyme kinetics a is called the Michaelis parameter (see p. 202; in R help the two parameters are called Vm and K respectively).

Here is SSmicmen in action:

```
data<-read.table("c:\\temp\\mm.txt",header=T)
attach(data)
names(data)
```

```
[1] "conc" "rate"
```

```
plot(rate~conc,pch=16)
```

To fit the non-linear model, just put the name of the response variable (rate) on the left of the tilde ~ then put SSmicmen(conc,a,b)) on the right of the tilde, with the name of your explanatory variable first in the list of arguments (conc in this case), then your names for the two parameters (a and b, above):

```
model<-nls(rate~SSmicmen(conc,a,b))
summary(model)
```

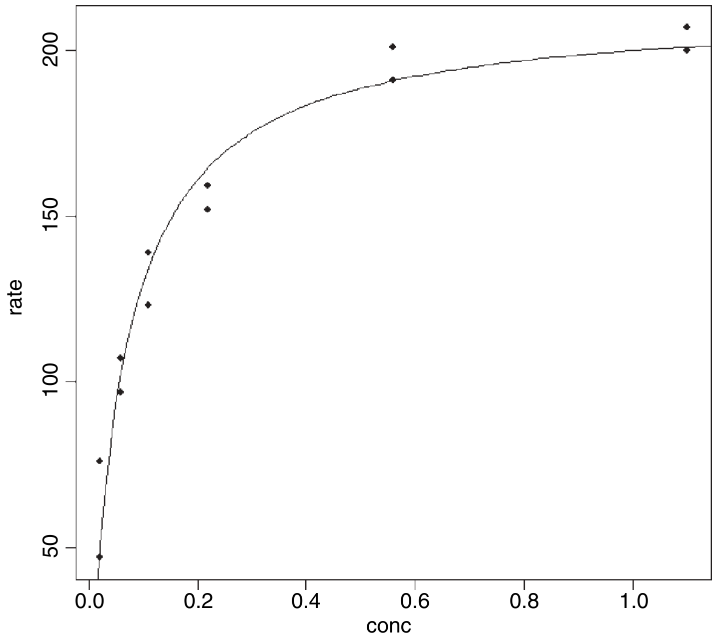
```
Formula: rate ~ SSmicmen(conc, a, b)
Parameters:
     Estimate   Std. Error   t value   Pr(>|t|)
a   2.127e+02   6.947e+00    30.615    3.24e-11   ***
b   6.412e-02   8.281e-03     7.743    1.57e-05   ***
```

So the equation looks like this:

$$\text{rate} = \frac{212.7 \times \text{conc}}{0.064\,12 + \text{conc}}$$

and we can plot it like this:

```
xv<-seq(0,1.2,.01)
yv<-predict(model,list(conc=xv))
lines(xv,yv)
```



Self-starting asymptotic exponential model

In Chapter 7 we wrote the three-paramter asymptotic exponential like this:

$$y = a - b\mathrm{e}^{-cx}.$$

In R's self-starting version SSasymp, the parameters are:

- a is the horizontal asymptote on the right-hand side (called Asym in R help);
- $b = a - \text{R0}$ where R0 is the intercept (the response when x is zero);
- c is the rate constant (the log of lrc in R help).

Here is SSasymp applied to the jaws data (p. 151):

```
deer<-read.table("c:\\temp\\jaws.txt",header=T)
attach(deer)
names(deer)
```

```
[1] "age" "bone"
```

```
model<-nls(bone~SSasymp(age,a,b,c))
plot(age,bone,pch=16)
xv<-seq(0,50,0.2)
yv<-predict(model,list(age=xv))
lines(xv,yv)
summary(model)
```

```
Formula: bone~SSasymp(age, a, b, c)

Parameters:
   Estimate  Std. Error  t value  Pr(>|t|)
a  115.2527      2.9139   39.553    <2e-16  ***
b   -3.4348      8.1961   -0.419     0.677
c   -2.0915      0.1385  -15.101    <2e-16  ***

Residual standard error: 13.21 on 51 degrees of freedom
```

The plot of this fit is on p. 664 along with the simplified model without the non-significant parameter *b*.

Alternatively, one can use the two-parameter form that passes through the origin, SSasympOrig, which fits the function $y = a(1 - \exp(-bx))$. The final form of the asymptotic exponential allows one to specify the function with an offset, *d*, on the *x* values, using SSasympOff, which fits the function $y = a - b\exp(-c(x - d))$.

Profile likelihood

The profile function is a generic function for profiling models, by investigating the behaviour of the objective function near the solution represented by the model's fitted values. In the case of nls, it investigates the profile log-likelihood function:

```
par(mfrow=c(2,2))
plot(profile(model))
```

The profile *t*-statistic (tau) is defined as the square root of change in sum-of-squares divided by residual standard error with an appropriate sign.

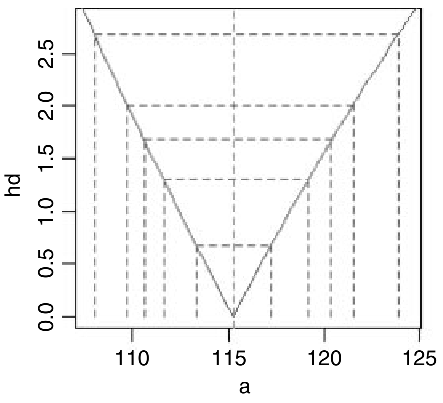
Self-starting logistic

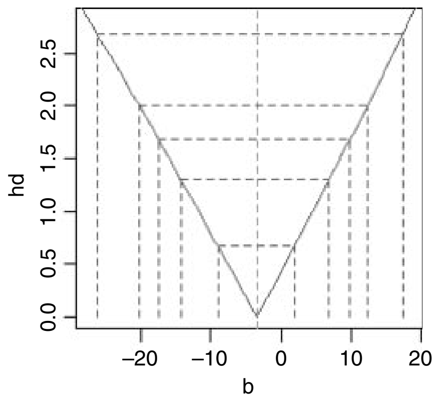
This is one of the most commonly used three-parameter growth models, producing a classic S-shaped curve:

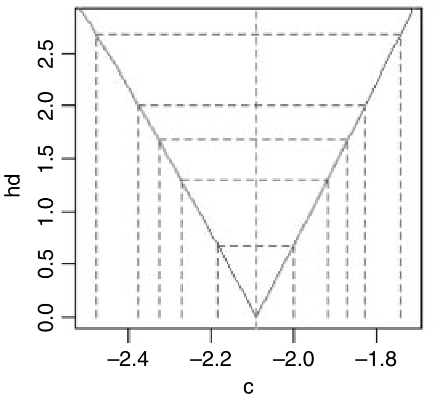
```
sslogistic<-read.table("c:\\temp\\sslogistic.txt",header=T)
attach(sslogistic)
names(sslogistic)
```

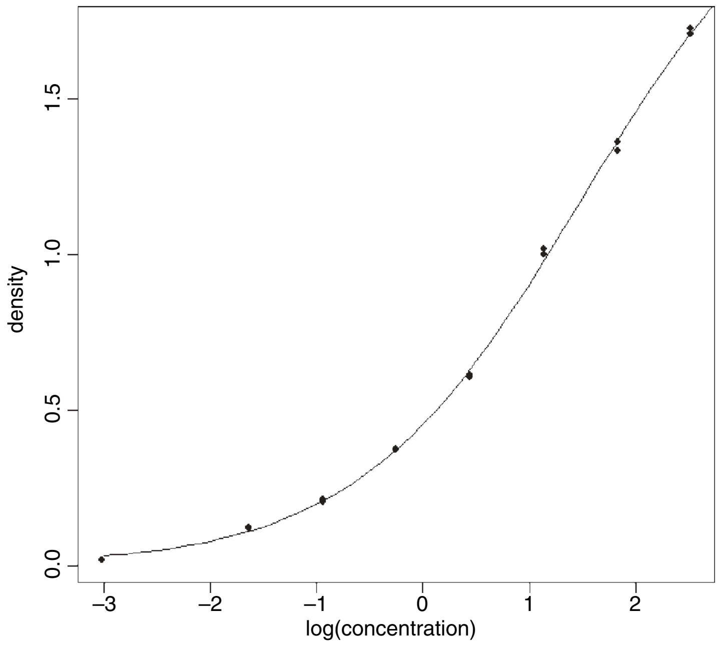
```
[1] "density" "concentration"
```

```
plot(density~log(concentration),pch=16)
```


hd
0.0
0.5
1.0
1.5
2.0
2.5
110
115
120
125
a


hd
0.0
0.5
1.0
1.5
2.0
2.5
−20
−10
0
10
20
b


hd
0.0
0.5
1.0
1.5
2.0
2.5
−2.4
−2.2
−2.0
−1.8
c


density
0.0
0.5
1.0
1.5
−3
−2
−1
0
1
2
log(concentration)

We estimate the three parameters (a, b, c) using the self-starting function SSlogis:

```
model<-nls( density ~ SSlogis(log(concentration), a, b, c ))
```

Now draw the fitted line using predict (note the antilog of xv in list):

```
xv<-seq(-3,3,0.1)
yv<-predict(model,list(concentration=exp(xv)))
lines(xv,yv)
```

The fit is excellent, and the parameter values and their standard errors are given by:

```
summary(model)
```

```
Parameters:
   Estimate  Std. Error  t value  Pr(>|t|)
a   2.34518     0.07815    30.01  2.17e-13  ***
b   1.48309     0.08135    18.23  1.22e-10  ***
c   1.04146     0.03227    32.27  8.51e-14  ***
```

Here a is the asymptotic value, b is the mid-value of x when y is $a/2$, and c is the scale.

Self-starting four-parameter logistic

This model allows a lower asymptote (the fourth parameter) as well as an upper:

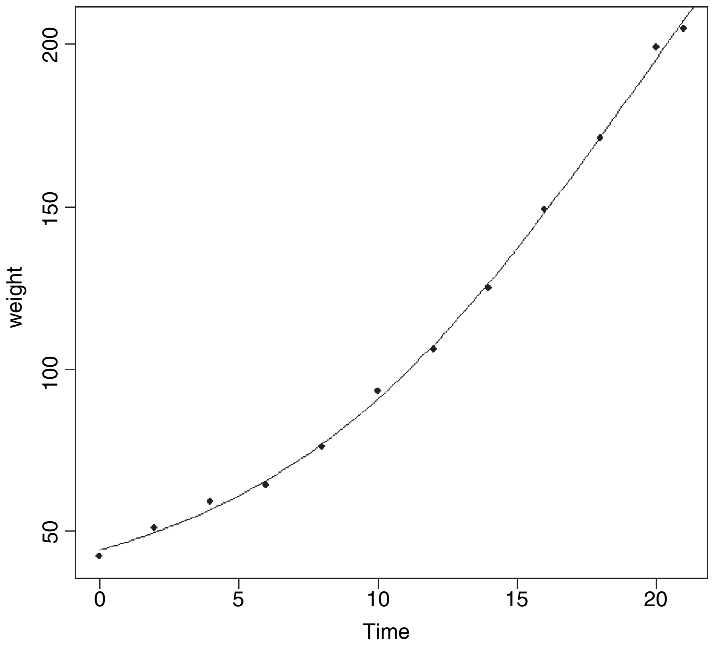
```
data<-read.table("c:\\temp\\chicks.txt",header=T)
attach(data)
names(data)
```

```
[1]  "weight"  "Time"
```

```
model <- nls(weight~SSfpl(Time, a, b, c, d))
xv<-seq(0,22,.2)
yv<-predict(model,list(Time=xv))
plot(weight~Time,pch=16)
lines(xv,yv)

summary(model)
Formula: weight~SSfpl(Time, a, b, c, d)

Parameters:
   Estimate  Std. Error  t value  Pr(>|t|)
a    27.453       6.601    4.159  0.003169  **
b   348.971      57.899    6.027  0.000314  ***
c    19.391       2.194    8.836  2.12e-05  ***
d     6.673       1.002    6.662  0.000159  ***

Residual standard error: 2.351 on 8 degrees of freedom
```

The four-parameter logistic is given by

$$y = A + \frac{B - A}{1 + e^{(D-x)/C}}.$$



This is the same formula as we used in Chapter 7, but note that C above is $1/c$ on p. 203. A is the horizontal asymptote on the left (for low values of x), B is the horizontal asymptote on the right (for large values of x), D is the value of x at the point of inflection of the curve (represented by xmid in our model for the chicks data), and C is a numeric scale parameter on the x axis (represented by scal). The parameterized model would be written like this:

$$y = 27.453 + \frac{348.971 - 27.453}{1 + \exp((19.391 - x)/6.673)}.$$

Self-starting Weibull growth function

R's parameterization of the Weibull growth function is

```
Asym-Drop*exp(-exp(lrc)*x^pwr)
```

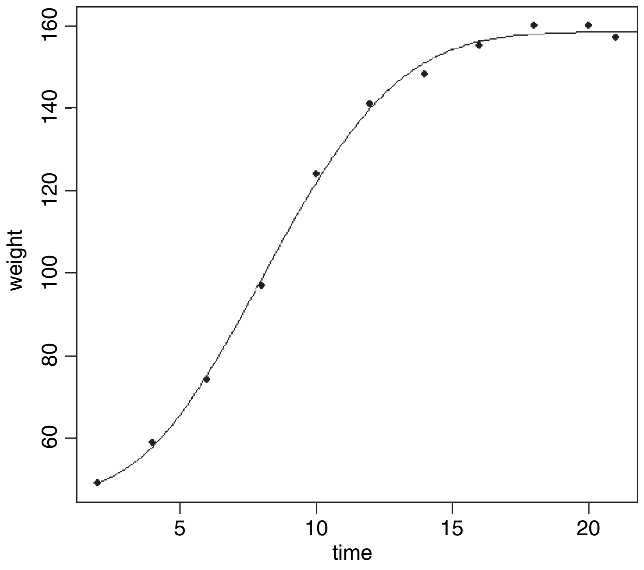
where Asym is the horizontal asymptote on the right, Drop is the difference between the asymptote and the intercept (the value of y at $x = 0$), lrc is the natural logarithm of the rate constant, and pwr is the power to which x is raised.

```
weights<-read.table("c:\\temp\\weibull.growth.txt",header=T)
attach(weights)
model <- nls(weight ~ SSweibull(time, Asym, Drop, lrc, pwr))
summary(model)
```

```
Formula: weight ~ SSweibull(time, Asym, Drop, lrc, pwr)
```

```
Parameters:
      Estimate  Std. Error  t value  Pr(>|t|)
Asym  158.5012      1.1769   134.67  3.28e-13  ***
Drop  110.9971      2.6330    42.16  1.10e-09  ***
lrc    -5.9934      0.3733   -16.06  8.83e-07  ***
pwr     2.6461      0.1613    16.41  7.62e-07  ***

Residual standard error: 2.061 on 7 degrees of freedom
```

```
plot(time,weight,pch=16)
xt<-seq(2,22,0.1)
yw<-predict(model,list(time=xt))
lines(xt,yw)
```



The fit is good, but the model cannot accommodate a drop in y values once the asymptote has been reached (you would need some kind of humped function).

Self-starting first-order compartment function

In the following, the response, drug concentration in the blood, is modelled as a function of time after the dose was administered. There are three parameters (a, b, c) to be estimated:

```
foldat<-read.table("c:\\temp\\fol.txt",header=T)
attach(foldat)
```

The model looks like this:

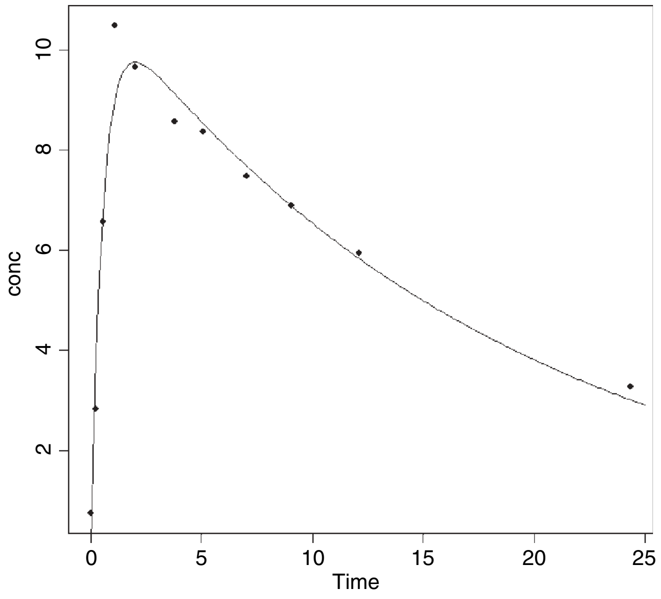
$$y = k\exp(-\exp(a)x) - \exp(-\exp(b)x),$$

where $k = Dose \times \exp(a + b - c)/(\exp(b)\text{-}\exp(a))$ and Dose is a vector of identical values provided to the fit (4.02 in this example):

```
model<-nls(conc~SSfol(Dose,Time,a,b,c))
summary(model)
```

```
Formula: conc ~ SSfol(Dose, Time, a, b, c)

Parameters:
   Estimate  Std. Error  t value  Pr(>| t| )
a   -2.9196      0.1709  -17.085    1.40e-07  ***
b    0.5752      0.1728    3.328      0.0104  *
c   -3.9159      0.1273  -30.768    1.35e-09  ***
```

```
plot(conc~Time,pch=16)
xv<-seq(0,25,0.1)
yv<-predict(model,list(Time=xv))
lines(xv,yv)
```



As you can see, this is a rather poor model for predicting the value of the peak concentration, but a reasonable description of the ascending and declining sections.

Bootstrapping a Family of Non-linear Regressions

There are two broad applications of bootstrapping to the estimation of parameters in non-linear models:

- Select certain of the data points at random with replacement, so that, for any given model fit, some data points are duplicated and others are left out.
- Fit the model and estimate the residuals, then allocate the residuals at random, adding them to different fitted values in different simulations

Our next example involves the viscosity data from the MASS library, where sinking time is measured for three different weights in fluids of nine different viscosities:

$$\text{Time} = \frac{b \times \text{Viscosity}}{Wt - c}.$$

We need to estimate the two parameters b and c and their standard errors.

```
library(MASS)
data(stormer)
attach(stormer)
```

Here are the results of the straightforward non-linear regression:

```
model<-nls(Time~b*Viscosity/(Wt-c),start=list(b=29,c=2))
summary(model)
```

```
Formula: Time ~ b * Viscosity/(Wt - c)

Parameters:
   Estimate  Std. Error  t value  Pr(>|t|)
b   29.4013      0.9155   32.114    < 2e-16  ***
c    2.2182      0.6655    3.333    0.00316  **

Residual standard error: 6.268 on 21 degrees of freedom
```

Here is a home-made bootstrap which leaves out cases at random. The idea is to sample the indices (subscripts) of the 23 cases at random with replacement:

```
sample(1:23,replace=T)
[1] 4  4 10 10 12  3 23 22 21 13  9 14  8  5 15 14 21 14 12  3 20 14 19
```

In this realization cases 1 and 2 were left out, case 3 appeared twice, and so on. We call the subscripts *ss* as follows, and use the subscripts to select values for the response (y_1) and the two explanatory variables (x_1 and x_2) like this:

```
ss<-sample(1:23,replace=T)
y<-Time[ss]
x1<-Viscosity[ss]
x2<-Wt[ss]
```

Now we put this in a loop and fit the model

```
model<-nls(y~b*x1/(x2-c),start=list(b=29,c=2))
```

one thousand times, storing the coefficients in vectors called *bv* and *cv*:

```
bv<-numeric(1000)
cv<-numeric(1000)
for(i in 1:1000){
ss<-sample(1:23,replace=T)
y<-Time[ss]
x1<-Viscosity[ss]
x2<-Wt[ss]
model<-nls(y~b*x1/(x2-c),start=list(b=29,c=2))
bv[i]<-coef(model)[1]
cv[i]<-coef(model)[2]
}
```

This took 7 seconds for 1000 iterations. The 95% confidence intervals for the two parameters are obtained using the quantile function:

```
quantile(bv,c(0.025,0.975))
```

```
    2.5%     97.5%
27.91842  30.74411
```

```
quantile(cv,c(0.025,0.975))
```

```
     2.5%      97.5%
0.9084572  3.7694501
```

Alternatively, you can randomize the locations of the residuals while keeping all the cases in the model for every simulation. We use the built-in functions in the boot library to illustrate this procedure.

```
library(boot)
```

First, we need to calculate the residuals and the fitted values from the nls model we fitted on p. 682:

```
rs<-resid(model)
fit<-fitted(model)
```

and make the fit along with the two explanatory variables Viscosity and *Wt* into a new dataframe called storm that will be used inside the 'statistic' function

```
storm<-data.frame(fit,Viscosity,Wt)
```

Next, you need to write a statistic function (p. 320) to describe the model fitting:

```
statistic<-function(rs,i){
storm$y<-storm$fit+rs[i]
coef(nls(y~b*Viscosity/(Wt-c),storm,start=coef(model)))}
```

The two arguments to statistic are the vector of residuals, *rs*, and the randomized indices, *i*. Now we can run the boot function over 1000 iterations:

```
boot.model<-boot(rs,statistic,R=1000)
boot.model
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = rs, statistic = statistic, R = 1000)

Bootstrap Statistics :

      original        bias   std. error
t1*  29.401294   0.6915554   0.8573951
t2*   2.218247  -0.2552968   0.6200594
```

The parametric estimates for *b* (t1) and *c* (t2) in boot.model are reasonably unbiased, and the bootstrap standard errors are slightly smaller than when we used nls. We get the bootstrapped confidence intervals with the boot.ci function: *b* is index = 1 and *c* is index = 2:

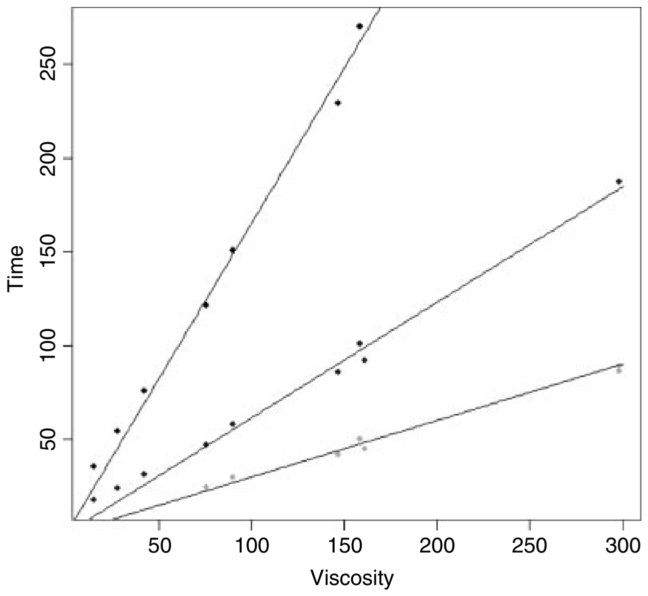
```
boot.ci(boot.model,index=1)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = storm.boot)

Intervals :
Level      Normal            Basic          Studentized
95%     (26.33, 29.65 )   (26.43, 29.78 )   (25.31, 29.63 )

Level     Percentile            BCa
95%     (27.65, 31.00 )   (26.92, 29.60 )
```

```
boot.ci(boot.model,index=2)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.model, index = 2)

Intervals :
Level      Normal             Basic
95%     ( 1.258, 3.689 )   ( 1.278, 3.637 )

Level     Percentile            BCa
95%     ( 0.800, 3.159 )   ( 1.242, 3.534 )
```

For comparison, here are the parametric confidence intervals (from model): for *b*, from 28.4858 to 30.3168; and for *c*, from 0.8872 to 3.5492.


Time
Viscosity