Analysis of Nonlinear Models

Any linear model can be expressed as

$$\mathrm{Y} = X\beta + \epsilon$$

where

$$E(\epsilon) = 0 \quad \text{and} \quad Var(\epsilon) = \Sigma$$

Then,

$$E(\mathrm{Y}) = X\beta$$

is a linear function of the unknown parameters β:

$$E(Y_i) = X_{1i}\beta_1 + \ldots + X_{pi}\beta_p$$

$i = 1, 2, \ldots, n$

Examples of linear models:

$$Y_i = \beta_0 + \beta_1 X_{1i} + \ldots + \beta_p X_{pi} + \epsilon_i$$

where $E(\epsilon_i) = 0$, for $i = 1, 2, \ldots, n$

$$Y_i = \beta_0 + \beta_1(X_{1i} - \bar{X}_1) + \beta_2(X_{1i} - \bar{X}_1)^2$$

$$+\beta_3(X_{1i} - \bar{X}_1)(X_{2i} - \bar{X}_2)$$

$$+\beta_4(X_{2i} - \bar{X}_2) + \epsilon_i$$

where $E(\epsilon_i) = 0$, for $i = 1, 2, \ldots, n$

$$Y_i = \beta_0 + \beta_1 log(X_{1i}) + \beta_2 e^{X_{2i}} + \epsilon_i$$

where $E(\epsilon_i) = 0$, for $i = 1, 2, \ldots, n$

Nonlinear model: (with additive random errors)

$$Y_i = f(\mathrm{X}_i, \beta) + \epsilon_i, \quad i = 1, \ldots, n$$

X_i is a vector of values of explanatory variables for the i-th case

β is a vector of parameters

Examples:

$$Y_i = \beta_0 + \beta_1 X_{1i}^{\beta_2} + \beta_3 X_{2i}^{\beta_4} + \epsilon_i$$

$$i = 1, \ldots, n$$

$$Y_i = \beta_0 \exp(X_{1i}\beta_1) + \epsilon_i$$

$$i = 1, \ldots, n$$

Intrinsically Linear Model

A model that can be transformed into a linear model.

Example:

$$Y_i = \beta_0 \exp(\beta_1 X_i)\epsilon_i$$

where $\{\epsilon_i : i = 1, \ldots, n\}$ are random errors with

$E(\epsilon_i) = 1$ and $Var(\epsilon_i) = \sigma^2$

Then

$$E(Y_i) = \beta_0 \exp(\beta_1 X_i)$$

and

$$Var(Y_i) = \sigma^2[\beta_0 \exp(\beta_1 X_i)]^2$$

Transform to a linear model:

$$log(Y_i) = \underline{\log(\beta_0)} + \beta_1 X_i + \underline{\log(\epsilon_i)}$$

$$\uparrow \qquad\qquad \uparrow$$

call this γ_0 call this η_i

$$= \gamma_0 + \beta_1 X_i + \eta_i$$

Growth Curve Models

Model how some response grows with the increase in some variable

- plant growth over time
- animal or human growth over time
- cracks in airplane wings across flight time
- manpower needs over time
- maintenance costs for rental trucks over mileage and time

Logistic growth curve:

- growth is slow in the beginning
- growth becomes more rapid
- growth slows as a limit is approached

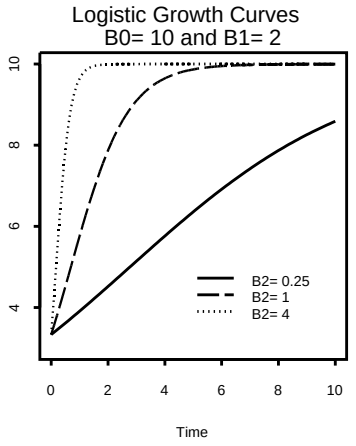
$$Y_i = \frac{\beta_0}{1 + \beta_1 \exp(-\beta_2 X_i)} + \epsilon_i$$

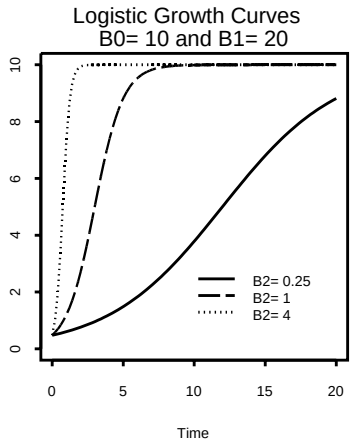
where

Y_i is the population size at time X_i

ϵ_i is a random error with $E(\epsilon_i) = 0$ and $Var(\epsilon_i) = \sigma^2$

$\beta_0 > 0,\ \beta_1 > 0,\ \beta_2 > 0.$





Properties of the logistic growth curve model:

as	$X_i \to -\infty$	$E(Y_i) \to 0$
when	$X_i = 0$	$E(Y_i) = \frac{\beta_0}{1+\beta_1}$
as	$X_i \to \infty$	$E(Y_i) \to \beta_0$

↑
this parameter corresponds to the limiting growth

Gompertz Growth Model

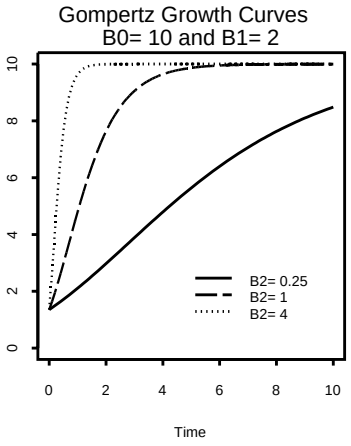
$$Y_i = \beta_0 \exp[-\beta_1 e^{-\beta_2 X_i}] + \epsilon_i$$

where $\beta_0 > 0,\ \beta_1 > 0,\ \beta_2 > 0.$

This is also an S-shaped curve.

when	$X_i = 0$	$E(Y_i) = \beta_0 e^{-\beta_1}$
as	$X_i \to \infty$	$E(Y_i) \to \beta_0$

↑
this is the limiting response



Richard Growth Curve

$$Y_i = \frac{\beta_0}{[1 + \beta_1 e^{-\beta_2 X_i}]^{1/\beta_3}} + \epsilon_i$$

when $X_i = 0$ $\quad E(Y_i) = \frac{\beta_0}{[1+\beta_1]^{1/\beta_3}}$

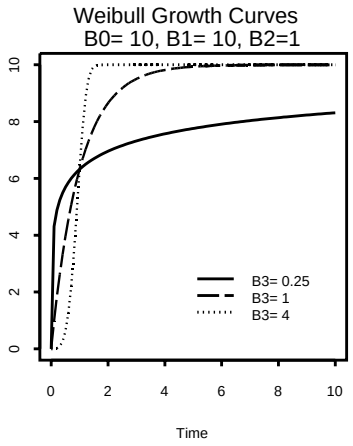
as $X_i \to \infty$ $\quad E(Y_i) \to \beta_0$

Weibull Growth Curve

$$Y_i = \beta_0 - \beta_1 \exp(-\beta_2 X_i^{\beta_3}) + \epsilon_i$$

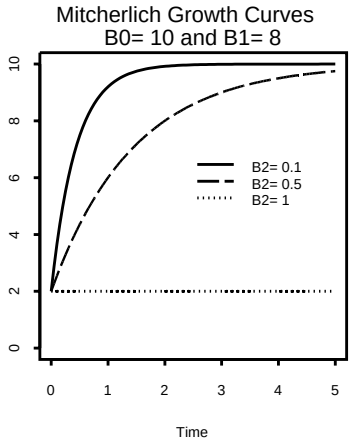
when $X_i = 0$ $\quad E(Y_i) = \beta_0 - \beta_1$

as $X_i \to \infty$ $\quad E(Y_i) = \beta_0$



Mitcherlich "Law"

- Used to model how the yield of a process increases with the increase in some factor
  - increase in yield of a crop when the level of fertilizer is increased
  - increase in yield of a chemical process when temperature or level of a catalyst increases
- No inflection point, this is not an S-shaped curve



$$Y_i = \beta_0 - (\beta_1)\beta_2^{X_i} + \epsilon_i$$

at $X_i = 0$ $\quad E(Y_i) = \beta_0 - \beta_1$

as $X_i \to \infty$ $\quad E(Y_i) \to \beta_0$

when $0 < \beta_2 < 1$

Alternative expressions of the same model are:

$$Y_i = \beta_0 - \beta_1 e^{-\beta_2 X_i} + \epsilon_i$$

$$Y_i = \beta_0[1 - e^{-(\beta_1 + \beta_2 X_i)}] + \epsilon_i$$

Michaelis-Menton equation:

Used to model the rate of uptake of dissolved substrate by organisms of microbial communities.

Y = velocity of uptake (μg/ℓ/hr)

β_0 = maximum velocity

β_1 = transport constant (μg/ℓ)

X = substrate conc. (μg/ℓ)

The model is

$$Y_i = \frac{\beta_0}{\beta_1 + X_i} + \epsilon_i$$

at $X_i = 0$ $\quad E(Y_i) = \beta_0/\beta_1$

as $X_i \to \infty$ $\quad E(Y_i) \to 0$

Pharmacokinetics

Plasma concentration of a drug at time T after an intravenous injection

$$E(Y|T) = \frac{\alpha k_e k_a}{k_c(k_a - k_e)}[\exp(-k_e T) - \exp(-k_a T)]$$

where

α is the initial dose

k_e is the elimination rate parameter

k_a is the absorbtion rate parameter

k_c is the clearance parameter

For this model to be meaningful, k_e, k_a, and k_c must all be positive.

Consequently, least squares estimation will result in a constrained minimization problem. This can be avoided by reparameterizing the model in terms of the natural logarithms of the parameters.

$$E(Y|T) = \frac{\alpha \exp(\beta_1 + \beta_2 - \beta_0)}{\exp(\beta_2) - \exp(\beta_1)} \times [\exp(-T\exp(\beta_1)) - \exp(-T\exp(\beta_2))]$$

where

α is the initial dose

$\beta_1 = \log(k_e)$, $\beta_2 = \log(k_a)$,

$\beta_0 = \log(k_c)$

Least Squares Estimation:

Data: (Y_1, X_1^T)
(Y_2, X_2^T)
$\vdots$
(Y_n, X_n^T)

Model: $Y_i = f(\mathrm{X}_i, \beta) + \epsilon_i$

Objective: Find $\mathrm{b} = \begin{bmatrix} b_1 \\ \cdots \\ b_k \end{bmatrix}$

to minimize

$$g(\mathrm{b}) = \sum_{i=1}^{n} [Y_i - f(\mathrm{X}_i, \mathrm{b})]^2$$

Estimating Equations

$$0 = \frac{\partial g(\mathrm{b})}{\partial b_1} = 2 \sum_{i=1}^{n} [Y_i - f(\mathrm{X}_i, \mathrm{b})] \frac{\partial f(\mathrm{X}_i, b)}{\partial b_1}$$

$$\cdots$$

$$0 = \frac{\partial g(\mathrm{b})}{\partial b_k} = 2 \sum_{i=1}^{n} [Y_i - f(\mathrm{X}_i, \mathrm{b})] \frac{\partial f(\mathrm{X}_i, b)}{\partial b_k}$$

These equations can be expressed as

$$0_{k \times 1} = D^T [\mathrm{Y} - \mathrm{f}(X, \mathrm{b})]$$

where

$$D = \begin{bmatrix} \frac{\partial f(\mathrm{X}_1, \mathrm{b})}{\partial b_1} & \cdots & \frac{\partial f(\mathrm{X}_1, \mathrm{b})}{\partial b_k} \\ \vdots & & \vdots \\ \frac{\partial f(\mathrm{X}_n, \mathrm{b})}{\partial b_1} & \cdots & \frac{\partial f(\mathrm{X}_n, \mathrm{b})}{\partial b_k} \end{bmatrix}$$

$$\mathrm{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} \quad \text{and} \quad \mathrm{f}(X, \mathrm{b}) = \begin{bmatrix} f(\mathrm{X}_1, \mathrm{b}) \\ \vdots \\ f(\mathrm{X}_n, \mathrm{b}) \end{bmatrix}$$

- These equations are not linear in b
- There is often no analytic formula for the solution as a function of Y and X
- An iterative procedure must be used to solve these equations

Gauss-Newton procedure

- Use a Taylor series expansion to approximate $f(\mathrm{X}_i, \beta)$ with a linear function of β.
- Apply OLS estimation to the linear approximation to "update" the estimate of β.
- Repeat this until convergence

Start the iterations with a vector of "starting values"

$$\mathrm{b}^{(0)} = \begin{bmatrix} b_1^{(0)} \\ \vdots \\ b_k^{(0)} \end{bmatrix}$$

At the end of the r-th iteration the values of the parameter estimates are

$$\mathrm{b}^{(r)} = \begin{bmatrix} b_1^{(r)} \\ \vdots \\ b_k^{(r)} \end{bmatrix}$$

Apply a Taylor series expansion:

For the i-th case at the r-th iteration, we have

$$f(\mathrm{X}_i, \beta) \doteq f(\mathrm{X}_i, \mathrm{b}^{(r)}) + \sum_{\ell=1}^{k} \left[\frac{\partial f(\mathrm{X}_i, \beta)}{\partial \beta_\ell}\right]_{\beta=\mathrm{b}^{(r)}} (\beta_\ell - b_\ell^{(r)}) + (\text{approximation error})$$

Then

$$\begin{aligned} Y_i &= f(X_i, \beta) + \epsilon_i \\ &= f(X_i, \mathrm{b}^{(r)}) + \sum_{\ell=1}^{k} \left[\frac{\partial f(\mathrm{X}_i, \beta)}{\partial \beta_\ell}\right]_{\beta=\mathrm{b}^{(r)}} (\beta_\ell - b_\ell^{(r)}) + \epsilon_i^* \end{aligned}$$

and

$$\begin{aligned} \mathrm{Y} &= \mathrm{f}(X, \beta) + \epsilon \\ &= \mathrm{f}(X, \mathrm{b}^{(r)}) + D^{(r)}(\beta - \mathrm{b}^{(r)}) + \epsilon^* \end{aligned}$$

or

$$\mathrm{Y} - \mathrm{f}(X, \mathrm{b}^{(r)}) = D^{(r)}(\beta - \mathrm{b}^{(r)}) + \epsilon^*$$

where $D^{(r)}$ is

$$D = \begin{bmatrix} \frac{\partial f(\mathrm{X}_1, \beta)}{\partial \beta_1} & \cdots & \frac{\partial f(\mathrm{X}_1, \beta)}{\partial \beta_k} \\ \vdots & & \vdots \\ \frac{\partial f(\mathrm{X}_n, \beta)}{\partial \beta_1} & \cdots & \frac{\partial f(\mathrm{X}_n, \beta)}{\partial \beta_k} \end{bmatrix}$$

evaluated at $\beta = \mathrm{b}^{(r)}$

$$\underbrace{(\mathrm{Y} - \mathrm{f}(X, \mathrm{b}^{(r)}))}_{\text{vector of responses}} = \underbrace{D^{(r)}}_{\text{model matrix}} \underbrace{(\beta - \mathrm{b}^{(r)})}_{\text{parameter vector}} + \epsilon^*$$

Use OLS estimation to obtain

$$(\mathrm{b}^{(r+1)} - \mathrm{b}^{(r)}) = ([D^{(r)}]^T D^{(r)})^{-1} [D^{(r)}]^T (\mathrm{Y} - \mathrm{f}(X, \mathrm{b}^{(r)}))$$

Then

$$\mathrm{b}^{(r+1)} = \mathrm{b}^{(r)} + ([\boldsymbol{D}^{(r)}]^T \boldsymbol{D}^{(r)})^{-1} \times [\boldsymbol{D}^{(r)}]^T (\mathrm{Y} - \mathrm{f}(\boldsymbol{X}, \mathrm{b}^{(r)})$$

and the "deviance" or sum of squared residuals at the end of the $(r+1)$-th iteration is

$$\boldsymbol{SSE}^{(r+1)} = \sum_{i=1}^{n} [\boldsymbol{Y}_i - \boldsymbol{f}(\mathrm{X}_i, \mathrm{b}^{(r+1)})]^2$$

Continue the iterations until some convergence criterion is satisfied:

$\boldsymbol{SSE}^{(r-1)} - \boldsymbol{SSE}^{(r)} <$ some constant

or

$$\max_{\ell=1,\ldots,k} \left\{ \frac{\left| b_\ell^{(r+1)} - b_\ell^{(r)} \right|}{\left| b_\ell^{(r)} + c \right|} \right\} < \text{constant}$$

or ??

Selection of starting values:

- speed of convergence
- convergence to a "local" minimum or a "global" minimum
- divergence
  - at each iteration check if
    $$\boldsymbol{SSE}^{(r+1)} < \boldsymbol{SSE}^{(r)}$$
  - if not, you might use a "halving" step
    $$\mathrm{b}_{new}^{(r+1)} = \mathrm{b}^{(r)} + .5(\mathrm{b}^{(r+1)} - \mathrm{b}^{(r)})$$
- try different starting values

- Grid search: evaluate
  $$g(\mathrm{b}) = \sum_{i=1}^{n} (\boldsymbol{Y}_i - \boldsymbol{f}(\mathrm{X}_i, \mathrm{b}))^2$$
  for values of b on a grid, and make a contour plot.
- Obtain starting values from
  - past experience
  - scientific theory
  - linear model approximations

Inference

Estimate σ_e^2 with

$$MSE = \frac{SSE}{n-k} = \frac{\sum_{i=1}^{n}(Y_i - \bar{Y}_i)^2}{n-k}$$

where $\bar{Y}_i = f(\mathrm{X}_i, \mathrm{b})$.

For large samples, when

$$\epsilon_i \sim NID(0, \sigma_e^2)$$

we have

$$\frac{(n-k)MSE}{\sigma_e^2} \dot{\sim} \chi^2_{(n-k)}$$

For large samples, when

$$\epsilon_i \sim iid(0, \sigma_e^2)$$

we have

$$\mathrm{b} \dot{\sim} N(\beta,\ \sigma_e^2(D^TD)^{-1})$$

↑

Estimate this covariance matrix as

$$S_{\mathrm{b}} = MSE(\bar{D}^T\bar{D})^{-1}$$

where $\bar{D}$ is D evaluated at $\beta = \mathrm{b}$.

Approximate t-tests

$$H_0 : \beta_\ell = c_\ell \quad \text{vs.} \quad H_A : \beta_\ell \neq c_\ell$$

Reject H_0 if

$$|t| = \frac{|b_\ell - c_\ell|}{S_{b_\ell}} > t_{(n-k),\alpha/2}$$

Approximate $(1-\alpha)\times 100\%$ confidence intervals:

$$b_\ell \ \pm \ t_{(n-k)\alpha/2}\ S_{b_\ell}$$

↗

Square root of the (ℓ,ℓ) element of S_{b}

Example 11.1:

Weight loss from an obese patient (V&R, Section 8.1)

Y = weight (in kg)

X = time (in days)

Obese patients in a weight loss program tend to lose tissue at a diminishing rate as the program progresses. We have data from one patient

- male, age 48
- height 193 cm (6'4")
- initial weight 184.35 kg (406.42 lb)

Data file: wtloss.dat

S-PLUS: Use the nls( ) function and the deriv( ) function for symbolic differentiation. Code is stored in the file

wtloss.ssc

SAS: Use the NLIN procedure. Code is stored in the file

wtloss.sas


```
# This is SPLUS code for fitting
# nonlinear models to the weight loss
# data in Venables and Ripley (Section 8.1).
# This file stored as   wtloss.ssc

# First access the MASS library

library(MASS, first=T)


# Enter the data stored in the file
#
#               wtloss.dat
#
# There are three numbers on each
# line in the following order:
#    Identification code
#    Days since beginning of the study
#    Weight (in kg)

wtloss <- read.table("wtloss.dat")

wtloss
```



```
   Days Weight
1     0 184.35
2     4 182.51
3     7 180.45
4     7 179.91
5    11 177.91
6    18 175.81
7    24 173.11
8    30 170.06
9    32 169.31
10   43 165.10
11   46 163.11
12   60 158.30
13   64 155.80
14   70 154.31
15   71 153.86
16   71 154.20
17   73 152.20
18   74 152.80
19   84 150.30
20   88 147.80
```



```
21   95 146.10
22  102 145.60
23  106 142.50
24  109 142.30
25  115 139.40
26  122 137.90
27  133 133.70
28  137 133.70
29  140 133.30
30  143 131.20
31  147 133.00
32  148 132.20
33  149 130.80
34  150 131.30
35  153 129.00
36  156 127.90
37  161 126.90
38  164 127.70
39  165 129.50
40  165 128.40
```

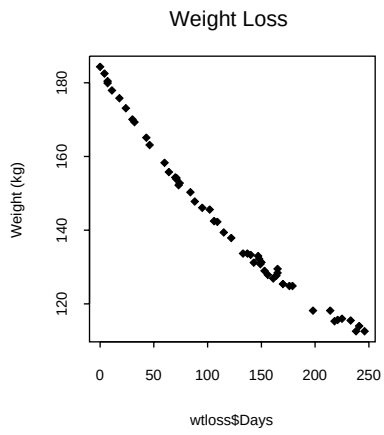
```
41  170 125.40
42  176 124.90
43  179 124.90
44  198 118.20
45  214 118.20
46  218 115.30
47  221 115.70
48  225 116.00
49  233 115.50
50  238 112.60
51  241 114.00
52  246 112.60
```



```
# Code for plotting weight against time.
# Unix users should insert the motif( )
# command here to open a graphics window.

# Specify plotting symbol and size of graph
# in inches.
# fin=c(w,h) specifies a plot that is w inches
#            wide and h inches high, not
#            including labels
# pch=18  requests a filled rectangle as a
#          plotting symbol
# mkh=b   requests plotting symbols that are
#          b inches high
# mex=a   sets the spacing between lines
#          printed in the margins
# plt     plt=c(.2,.8,.2,.8) defines the
#          fraction of figure region to use
#          for plotting.  This can provide
#          more space for labels.
```



```
par(fin=c(7.0,7.0),pch=18,mkh=.1,mex=1.5,
     plt=c(.2,.8,.2,.8))
plot(wtloss$Days, wtloss$Weight, type="p",
         ylab="Weight (kg)",
         main="Weight Loss")
```

An Exponential Decay Model

$$Y_i = \beta_0 + \beta_1 e^{-(X_i/\beta_2)} + \epsilon_i$$

where $\beta_0 > 0,\ \beta_1 > 0,\ \beta_2 > 0$ and

X_i	is time (in days) since the start of the weight loss program
Y_i	is the observed weight at time X_i
β_0	is the limiting weight (stable lean weight) as $X \to \infty$
β_1	is the total weight to be lost
$-\beta_2 log(1-p)$	is the time needed to lose 100p% of the remaining weight to be lost


```
# Code for fitting an exponential decay
# model for weight loss.

# The initial values of 90, 95, and 190 are
# specified by the user.  Since no formulas
# for derivatives are specified, the
# Gauss-Newton optimization procedure
# is applied with numerical approximations
# to the first partial derviatives.

wtloss.fm <- nls(
       formula = Weight ~ b0 + b1*exp(-Days/b2),
          data = wtloss,
         start = c(b0=90, b1=95, b2=190),
         trace = T)

305.628 : 90 95 190
39.5054 : 81.6598 102.392 204.343
39.2447 : 81.3736 102.684 204.735
```



```
# The first line of output comes from the
# initial values and the last two lines
# of output show the value of the sum of
# squared errors and the values of the
# parameter estimates for successive
# iterations of the minimization procedure.
# This output was  requested by the  trace=T
# option.  Other information is obtained in
# the following way:


summary(wtloss.fm)


Formula: Weight ~ b0 + b1 * exp( - Days/b2)


Parameters:
      Value Std. Error t value
b0  81.3736    2.26903 35.8627
b1 102.6840    2.08279 49.3013
b2 204.7350    7.63848 26.8031
```



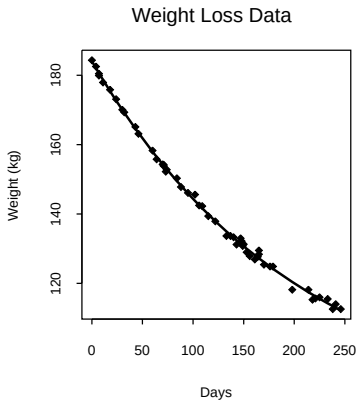
```
Residual standard error:

      0.894937 on 49 degrees of freedom


Correlation of Parameter Estimates:
       b0     b1
b1 -0.989
b2 -0.986  0.956


# Print the sum of squared residuals. This
# will not work if you do not attach the
# MASS library.

deviance(wtloss.fm)

[1] 39.2447
```

```
# Print the estimated covariance matrix of
# the estimated parameters.  This will not
# work if you do not attach the MASS library.

vcov(wtloss.fm)

           b0         b1        b2
b0   5.148517  -4.674626 -17.08394
b1  -4.674626   4.338010  15.21100
b2 -17.083939  15.211003  58.34644


#  Other information in the object created
#  by the nls( ) function

names(wtloss.fm)
[1] "parameters"   "formula"   "call"
[4] "residuals"    "R"         "fitted.values"
[7] "assign"       "trace"
```

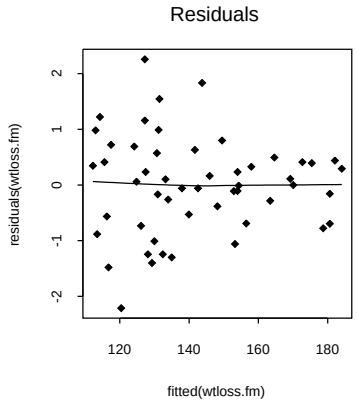

Some functions you can apply to objects made by the nls function:

coef	estimates of coefficients
fitted	fitted values
intervals	confidence intervals for coeffcients
plot	diagnostic plots
predict	predictions
qqnorm	normal probability plots
resid	residuals
summary	summary information
update	update the model

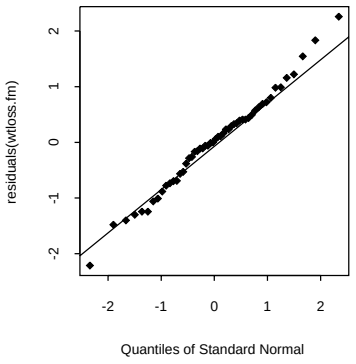

```
#  Plot the curve

 par(fin=c(7.0,7.0),pch=18,mkh=.1,mex=1.5,
       plt=c(.2,.8,.2,.8))
 plot(wtloss$Days, wtloss$Weight, type="p",
        xlab="Days",
        ylab="Weight (kg)",
        main="Weight Loss Data")
 lines(wtloss$Days,wtloss.fm$fitted.values,lty=1,lwd=3)
```




```
#  Check residual plots.  The scatter.smooth
#  passes a smooth curve through the points
#  on the residual plot

  scatter.smooth(fitted(wtloss.fm),
      residuals(wtloss.fm), span=.75, degree=1,
      main="Residuals")
```



```
qqnorm(residuals(wtloss.fm))
qqline(residuals(wtloss.fm))
```



Determination of starting values

For each case, expand

$$E(Y_i) = f(X_i, \beta) = \beta_0 + \beta_1 e^{-(X_i/\beta_2)}$$

in a Taylor series expansion about $X_i = \bar{X}$ to obtain

$$E(Y_i) \doteq f(\bar{X}, \beta) + \left.\frac{\partial f(X_i, \beta)}{\partial X_i}\right|_{X_i = \bar{X}} (X_i - \bar{X})$$

$$+ \left.\frac{\partial^2 f(X_i, \beta)}{\partial X_i^2}\right|_{X_i = \bar{X}} \frac{(X_i - \bar{X})^2}{2}$$

For the exponential decay model

$$f(X_i, \beta) = \beta_0 + \beta_1 e^{-(X_i/\beta_2)}$$

and

$$Y_i \doteq \left(\beta_0 + \beta_1\ e^{-(\bar{X}/\beta_2)}\right) - \frac{\beta_1}{\beta_2} e^{-(\bar{X}/\beta_2)}(X_i - \bar{X})$$

$$+ \frac{\beta_1}{\beta_2^2}\ e^{-(\bar{X}/\beta_2)} \frac{(X_i - \bar{X})^2}{2} + error_i$$

$$= \alpha_0 + \alpha_1 (X_i - \bar{X}) + \alpha_2 \frac{(X_i - \bar{X})^2}{2} + error_i$$

Obtain OLS estimators $\hat{\alpha}_0$, $\hat{\alpha}_1$, $\hat{\alpha}_2$. Then,

$$\beta_2^* = -\frac{\hat{\alpha}_1}{\hat{\alpha}_2}$$

provides a starting value for the estimate of β_2.

Next, starting values for β_0 and β_1 are obtained as OLS estimators of the coefficents in

$$Y_i = \beta_0 + \beta_1 e^{-(X_i/\beta_2^*)} + error_i$$

```
# Create a function to generate starting
# values for parameter estimates in the
# negative exponential decay model.

negexp.sv <- function(x, y ) {
 mx<-mean(x)
 x1<-x-mx
 x2<-((x-mx)^2)/2
 b <- as.vector(lm(y~x1+x2)$coef)
 b2<- -b[2]/b[3]
   b <- as.vector(lm(y~exp(-x/b2))$coef,ncol=1)
   parms <- cbind(b[1],b[2],b2)
   parms }

b.start <- negexp.sv(wtloss$Days, wtloss$Weight)

wtloss.ss <- nls(
   formula = Weight ~ b0 + b1*exp(-Days/b2),
      data = wtloss,
      start = c(b0=b.start[1], b1=b.start[2],
                b2=b.start[3]), trace = T)
```



```
39.6455 : 79.7542 104.130 210.266
39.2455 : 81.4217 102.636 204.573
39.2447 : 81.3737 102.684 204.734


# You can supply formulas for first partial
# derivatives to the nls() function.  Sometimes
# this makes the minimization of the sum
# of squared residuals more stable.

# Derivatives can only be provided as an
# attribute of the model.  We will create a
# function to evaluate the mean response and
# the D matrix of values of first partial
# derivatives evaluated at the current values
# of the parameter estimates.  The D matrix is
# included as a "gradient" attribute.  Note
# that the gradient matrix must have column
# names matching those of the corresponding
# parameters.
```


Formulas for the first partial derivatives of the function for mean weight at X days after the start of the program

$$E(Y_i) = f(X_i, \beta) = \beta_0 + \beta_1 e^{-(X_i/\beta_2)}$$

The i-th row of D is

$$\frac{\partial f(x_i, \beta)}{\partial \beta_0} = 1$$

$$\frac{\partial f(x_i, \beta)}{\partial \beta_1} = e^{-(X_i/\beta_2)}$$

$$\frac{\partial f(x_i, \beta)}{\partial \beta_2} = \frac{\beta_1 X_i}{\beta_2^2} e^{-(X_i/\beta_2)}$$


```
expn <- function(b0, b1, b2, x)
{
    temp <- exp(-x/b2)
    model.func <- b0 + b1 * temp
    D <- cbind(1, temp, (b1 * x * temp)/b2^2)
    dimnames(Z) <- list(NULL, c("b0","b1","b2"))
    attr(model.func, "gradient") <- D
    model.func
}


wtloss.gr <- nls(
     formula=Weight ~ expn(b0, b1, b2, Days),
       data = wtloss,
      start = c(b0=b.start[1], b1=b.start[2],
                b2=b.start[3]), trace = T)


39.6455 : 79.7542 104.130 210.266
39.2455 : 81.4217 102.636 204.573
39.2447 : 81.3737 102.684 204.734
```

```
# Use the symbolic differentiation function,
# deriv3( ), to createe a function to specify
# the model and compute derivatives.  You must
# attach the MASS library to use deriv3( ).

expn1 <- deriv3(expr = y ~ b0 + b1 * exp(-x/b2),
          namevec = c("b0", "b1", "b2"),
     function.arg = function(b0, b1, b2, x) NULL)

# List the contents of expn1
expn1

function(b0, b1, b2, x) {
.expr3 <- exp((( - x)/b2))
.expr7 <- 0
.expr8 <- b2^2
.expr9 <- x/.expr8
.expr10 <- .expr3 * .expr9
.value <- b0 + (b1 * .expr3)
```



```
.grad <- array(0, c(length(.value), 3),
               list(NULL, c("b0", "b1", "b2")))
.hess <- array(0, c(length(.value), 3, 3),
               list(NULL, c("b0", "b1", "b2"),
                     c("b0", "b1", "b2")))
.grad[, "b0"] <- 1
.grad[, "b1"] <- .expr3
.grad[, "b2"] <- b1 * .expr10
.hess[, "b0", "b0"] <- .expr7
.hess[, "b1", "b0"] <- .expr7
.hess[, "b2", "b0"] <- .expr7
.hess[, "b0", "b1"] <- .expr7
.hess[, "b1", "b1"] <- .expr7
.hess[, "b2", "b1"] <- .expr10
.hess[, "b0", "b2"] <- .expr7
.hess[, "b1", "b2"] <- .expr10
.hess[, "b2", "b2"] <- b1 *
          ((.expr10 * .expr9) - (.expr3 *
     ((x * (2 * b2))/(.expr8^2))))
attr(.value, "gradient") <- .grad
attr(.value, "hessian") <- .hess
.value }
```



```
# Now fit the model

  wtloss.gr <- nls(
      formula = Weight ~ expn1(b0, b1, b2, Days),
         data = wtloss,
        start = c(b0=b.start[1], b1=b.start[2],
                  b2=b.start[3]), trace = T)


39.6455 : 79.7542 104.130 210.266
39.2455 : 81.4217 102.636 204.573
39.2447 : 81.3737 102.684 204.734
```



```
# Print final parameter estimates

 wtloss.gr$parameters
           b0       b1       b2
     81.37369 102.6842 204.7338

# Print large sample estimate of the
# covariance matrix

   vcov(wtloss.gr)
              b0         b1        b2
    b0   5.148431  -4.674544 -17.08362
    b1  -4.674544   4.337933  15.21070
    b2 -17.083616  15.210699  58.34522

# Print large sample estimates of
# standard errors for the parameter estimates

  sqrt(diag(covb))
   [1] 2.269015 2.082771 7.638405
```

Confidence intervals

$$b_i \pm t_{(n-k),\alpha/2}\, S_{b_i}$$

Using $\alpha = .05$, we have $t_{49,.025} = 2.0096$

For β_0:

$$81.374 \pm (2.0096)(2.26902)$$

$$\Rightarrow (76.81, 85.93)$$

For β_1:

$$102.684 \pm (2.0096)(2.08277)$$

$$\Rightarrow (98.50, 106.87)$$

For β_2:

$$204.734 \pm (2.0096)(7.6384)$$

$$\Rightarrow (189.4, 220.1)$$

Approximate profile likelihood confidence intervals (V& R, Section 8.4):

- Partition the parameter vector β in $E(\mathrm{Y}) = f(\mathrm{X}, \beta)$ as

$$\beta = \begin{bmatrix} \beta_1 \\ \beta_2^* \end{bmatrix}$$

- Derive an approximate F-test of $H_0 : \beta_1 = \beta_{1,0}$ vs. $H_A : \beta_1 \neq \beta_{1,0}$

  Let $\bar{\beta}$ be the least squares estimator of β obtained by minimizing

$$\sum_{j=1}^{n} (Y_j - f(\mathrm{X}_j; \beta))^2$$

The minimum sum of squared residuals is

$$RSS(\bar{\beta}) = \sum_{j=1}^{n} [Y_j - f(\mathrm{X}_j; \bar{\beta})]^2$$

Let $\bar{\beta}_{2|1}$ be the "conditional" least squares estimate of β obtained by fixing $\beta_1 = \beta_{1,0}$ and optimizing with respect to β_2^*, the remaining parameters. Minimize

$$\sum_{j=1}^{n} \left[Y_j - f(\mathrm{X}_j; \begin{bmatrix} \beta_{1,0} \\ \beta_2^* \end{bmatrix}) \right]^2$$

Then,

$$\bar{\beta}_{2|1} = \begin{bmatrix} \beta_{1,0} \\ \bar{\beta}_2^* \end{bmatrix} .$$

The minumum sum of squared residuals is

$$RSS(\bar{\beta}_{2|1}(\beta_{1,0})) = \sum_{j=1}^{n} [Y_j - f(\mathrm{X}_j; \bar{\beta}_{2|1})]^2$$

Approximate F-test:

$$F(\beta_{1,0}) = \frac{RSS(\bar{\beta}_{2|1}(\beta_{1,0})) - RSS(\bar{\beta})}{(RSS(\bar{\beta})/(n-k))}$$

with $(1, n-k)$ degrees of freedom

$\uparrow$

number of parameters in β

Approximate t-statistic:

$$t(\beta_{1,0}) = \text{sign}(\beta_{1,0} - \bar{\beta}_1)\sqrt{F(\beta_{1,0})}$$

with $n-k$ d.f.

- Approximate $(1-\alpha)\times 100\%$ confidence interval for β_1:

  Find all values of $\beta_{1,0}$ such that
  $H_0 : \beta_1 = \beta_{1,0}$ is not rejected
  at the α level of significance, i.e.,

  find all values of $\beta_{1,0}$ such that

$$t_{(n-k),1-\alpha/2} \;\leq\; t(\beta_{1,0}) \;\leq\; t_{(n-k),\alpha/2}$$

- This is a large sample method:
  - It uses the limiting normal distribution of parameter estimates
  - This procedure may fail if the sum of squared residuals (or profile likelihood)
    becomes too flat
  - These confidence intervals are not necessarily centered at the point estimate of the parameter
  - These confidence intervals are produced by confint( ) in S-PLUS


```
# Compute 95% confidence intervals for the
# parameters using a profile likelihood
# method

 rbind(b0 = confint(wtloss.gr,parm="b0"),
       b1 = confint(wtloss.gr,parm="b1"),
       b2 = confint(wtloss.gr,parm="b2"))
```

```
        2.5%     97.5%
b0  76.48233  85.63038
b1  98.78522 107.18805
b2 190.42694 221.22585
```



```
# Compute confidence intervals for parameters
# using standard large sample normal theory.
# We will have to create our own function.
# The intervals( )  function cannot be
# applied to objects created by nls( ).

 est.b <- function(obj, level=0.95) {
     b <- coef(obj)
     n <- length(obj$fitted.values) -
                     length(obj$parameters)
     stderr <- sqrt(diag(vcov(obj)))
     low <- b - qt(1 - (1 - level)/2, n)*stderr
     high <- b + qt(1 - (1 - level)/2, n)*stderr
     a <- cbind(b, stderr, low, high)
     a }

> est.b(wtloss.gr)
```

```
           b   stderr       low      high
b0  81.37369 2.269015  76.81394  85.93345
b1 102.68422 2.082771  98.49874 106.86971
b2 204.73381 7.638405 189.38387 220.08376
```

Inferences about functions of parameters

(i) Expected weight after X days on the program

$$h(\beta) = f(X, \beta) = \beta_0 + \beta_1 e^{-(X/\beta_2)}$$

(ii) Time until expected weight is Y_0:

$$Y_0 = \beta_0 + \beta_1 e^{-(X_0/\beta_2)}$$

$$\Rightarrow \quad X_0 = h(\beta) = -\beta_2 \log\left(\frac{Y_0 - \beta_0}{\beta_1}\right)$$

Delta Method

If $\mathrm{b} \dot{\sim} N(\beta, V)$ then

$$\mathrm{h(b)} \dot{\sim} N(\mathrm{h}(\beta), GVG^T)$$

where

$$G = \begin{bmatrix} \frac{\partial h_1(\beta)}{\partial \beta_1} & \frac{\partial h_1(\beta)}{\partial \beta_2} \cdots & \frac{\partial h_1(\beta)}{\partial \beta_k} \\ \vdots & \vdots & \vdots \\ \frac{\partial h_q(\beta)}{\partial \beta_1} & \cdots\cdots & \frac{\partial h_q(\beta)}{\partial \beta_k} \end{bmatrix}$$

This is a large sample approximation. It is more accurate for larger sample sizes.

(i) Expected weight after $X = 100$ days in the program

Point estimate:

$$\widehat{Y} = h(\mathrm{b}) = b_0 + b_1^{-(X/b_2)}$$

$$= 81.3737 + 102.684\ e^{(100/204.73473)}$$

$$= 144.379 \text{ kg.}$$

Apply the delta method:

$$\widehat{G} = \left[\frac{\partial h(\beta)}{\partial \beta_1} \quad \frac{\partial h(\beta)}{\partial \beta_2} \quad \frac{\partial h(\beta)}{\partial \beta_3}\right]_{\beta=\mathrm{b}}$$

$$= \left[1 \qquad e^{-(100/b_2)} \qquad \frac{b_1(100)}{b_2^2} e^{(-(100/b_2)}\right]$$

$$= [1 \quad .6135847 \quad .1503129]$$

Variance:

$$S^2_{\widehat{Y}} = \widehat{G}\widehat{V}\widehat{G}^T$$

$$= \widehat{G} \begin{bmatrix} 5.14841 & -4.67453 & -17.08394 \\ -4.67453 & 4.33792 & 15.21101 \\ -17.08394 & 15.21101 & 58.34646 \end{bmatrix} \widehat{G}^T$$

$$= .033378$$

Standard error:

$$S_{\widehat{Y}} = \sqrt{.033378} = .182696$$

Approximate 95% confidence interval:

$$\widehat{Y} \pm t_{49,.025} S_{\widehat{Y}}$$

$$\Rightarrow (144.01, 144.75)$$

(ii) Time until weight reaches $Y_0 = 100$ kg.

$$\bar{X} = h(\mathrm{b}) = -b_2 \log\left(\frac{Y_0 - b_0}{b_1}\right)$$

$$= (-141.911) \log\left(\frac{100 - 81.3738}{102.684}\right)$$

$$= 349.5 \text{ days}$$

Apply the delta method:

$$G = \left[\frac{\partial h(\beta)}{\partial \beta_0} \quad \frac{\partial h(\beta)}{\partial \beta_1} \quad \frac{\partial h(\beta)}{\partial \beta_2}\right]_{\beta=\mathrm{b}}$$

$$= \left[\frac{b_2}{(Y_0 - b_0)} \quad \frac{b_2}{b_1} \quad -\log\left(\frac{Y - b_0}{b_1}\right)\right]$$

$$= [10.991737 \quad 1.993829 \quad 1.70708]$$

Variance:

$$S^2_{\bar{X}} = \hat{G} \begin{bmatrix} 5.148941 & -4.67453 & -17.08394 \\ -4.67453 & 4.33792 & 15.21101 \\ -17.08394 & 15.21101 & 58.34646 \end{bmatrix} \hat{G}^T$$

$$= 66.8393$$

Standard error:

$$S_{\bar{X}} = \sqrt{66.8393} = 8.1755$$

Approximate 95% confidence interval

$$\bar{X} \pm t_{49,.025}\, S_{\bar{X}}$$

$$\Rightarrow (333.1,\ 365.9) \text{ days}$$


```
# Construct a confidence interval for the
# time needed to achieve specific predicted
# weights: (110,100,90) kg.

# Compute the estimated times and their
# standard errors. First use the deriv3( )
# function to get values of the first
# partial derviatives of the mean function
# needed to apply the delta method.

 time.pds <- deriv3(~ -b2*log((y0-b0)/b1),
              c("b0","b1","b2"),
              function(y0, b0, b1, b2) NULL)


#  list the function

 time.pds
```



```
function(y0, b0, b1, b2)
{
.expr2 <- y0 - b0
.expr3 <- .expr2/b1
.expr4 <- log(.expr3)
.expr6 <- 1/b1
.expr7 <- .expr6/.expr3
.expr10 <- .expr3^2
.expr13 <- b1^2
.expr16 <- .expr2/.expr13
.expr21 <-  - (b2 * (((1/.expr13)/.expr3)
                - ((.expr6 *.expr16)/.expr10)))
.expr22 <- .expr16/.expr3
.value <- ( - b2) * .expr4
.grad <- array(0, c(length(.value), 3),
              list(NULL, c("b0", "b1", "b2")))
```

```
    .hess <- array(0, c(length(.value), 3, 3),
                list(NULL,c("b0", "b1", "b2"),
                  c("b0", "b1", "b2")))
    .grad[, "b0"] <- b2 * .expr7
    .grad[, "b1"] <- b2 * .expr22
    .grad[, "b2"] <-  - .expr4
    .hess[, "b0", "b0"] <- b2 *
              ((.expr6 * .expr6)/.expr10)
    .hess[, "b1", "b0"] <- .expr21
    .hess[, "b2", "b0"] <- .expr7
    .hess[, "b0", "b1"] <- .expr21
    .hess[, "b1", "b1"] <-  - (b2 *
           ((((.expr2 * (2 * b1))/(.expr13^2))
           /.expr3) - ((.expr16 *
                     .expr16)/.expr10)))
    .hess[, "b2", "b1"] <- .expr22
    .hess[, "b0", "b2"] <- .expr7
    .hess[, "b1", "b2"] <- .expr22
    .hess[, "b2", "b2"] <- 0
      attr(.value, "gradient") <- .grad
      attr(.value, "hessian") <- .hess
.value
}
```



```
#  Compute the estimates of the times and
#  the large sample covariance matrix and
#  standard errors

est.time <- function(y0, obj,level=.95) {
    b <- coef(obj)
    tmp <- time.pds(y0, b["b0"], b["b1"], b["b2"])
    x0 <- as.vector(tmp)
    lam <- attr(tmp, "gradient")
    np <- length(p)
    v <- (lam%*%vcov(obj)*lam) %*% matrix(1,np,1)
    n <- length(obj$fitted.values)-np
    low <- x0 - qt(1 - (1 - level)/2, n)*sqrt(v)
    high <- x0 + qt(1 - (1 - level)/2, n)*sqrt(v)
    a <- cbind(x0, sqrt(v), low, high)
    dimnames(a) <- list(paste(y0, "kg: "),
                    c("x0", "SE", "low", "high"))
    a
}
```



```
est.time(c(110, 100, 90), wtloss.gr)
```

```
              x0        SE      low     high
110 kg:  261.5131  2.795151 255.8961 267.1302
100 kg:  349.4977  8.175374 333.0687 365.9267
 90 kg:  507.0933 31.217666 444.3591 569.8275
```


Maximum Likelihood Estimation

Example 11.2:

Specific gravity of a three component mixture. Myers, R. H. (1964) Technometrics, 343-356.

$$\boldsymbol{Y}_i|(\boldsymbol{X}_{1i} = \boldsymbol{x}_{1i}, \boldsymbol{X}_{2i} = \boldsymbol{x}_{2i}, \boldsymbol{X}_{3i} = \boldsymbol{x}_{3i}) \sim \boldsymbol{N}(\mu_i,\ \sigma^2)$$

where

$$\mu_i = \frac{1}{\beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i}}$$

and

Y is the specific gravity measurement of a mixture

X_1 is the percentage nitroglycerine (NG)

X_2 is the percentage triacetin (TA)

X_3 is the percentage 2-nitrodiphenylamine (NDPA)

and Y_1, Y_2, ... , Y_n are independent.

Joint likelihood function:

$$L(\beta_1,\ \beta_2,\ \beta_3, \sigma^2 \mid data)$$

$$= \prod_{i=1}^{n} \left(\frac{1}{\sqrt{(2\pi\sigma^2)}} \exp\left(\frac{-1}{2\sigma^2}[Y_i - \frac{1}{\beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i}}]^2\right)\right)$$

$$= \prod_{i=1}^{n} \left(\frac{1}{\sqrt{(2\pi\theta)}} \exp\left(\frac{-1}{2\theta}[Y_i - \frac{1}{\beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i}}]^2\right)\right)$$

$$= \prod_{i=1}^{n} f(Y_i;\ \beta_1,\ \beta_2,\ \beta_3,\ \theta)$$

Joint log-likelihood function

$$\ell(\beta_1,\ \beta_2,\ \beta_3,\ \theta \mid data)$$
$$= \sum_{i=1}^{n} log[f(Y_i;\ \beta_1,\ \beta_2,\ \beta_3,\ \theta)]$$

The score function

$$\mathrm{u}(\theta) = \begin{bmatrix} u_1(\theta) \\ \vdots \\ u_r(\theta) \end{bmatrix} = \begin{bmatrix} \frac{\partial \ell(\theta; \mathrm{Y}_1, \ldots, \mathrm{Y}_n)}{\partial \theta_1} \\ \vdots \\ \frac{\partial \ell(\theta; \mathrm{Y}_1, \ldots, \mathrm{Y}_n)}{\partial \theta_r} \end{bmatrix}$$

is the vector of first partial derivatives of the log-likelihood function with respect to the elements of

$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_r \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \theta \end{bmatrix}.$$

The likelihood equations are

$$u(\theta; Y_1, \ldots, Y_n) = 0$$

Maximum Likelihood Estimator (MLE)

$$\hat{\theta} = \begin{bmatrix} \hat{\theta}_1 \\ \vdots \\ \hat{\theta}_r \end{bmatrix}$$

is a solution to the likelihood equations, that maximizes the log-likelihood function.

Fisher information matrix

$$\begin{aligned} i(\theta) &= \text{Var}(u(\theta; \mathrm{Y}_1, \ldots, \mathrm{Y}_n)) \\ &= E\left(u(\theta; Y_1 \ldots, Y_n)[u(\theta; Y_1, \ldots, Y_n)]^T\right) \\ &= -E\left(\left[\frac{\partial \ell(\theta;\ Y_1, \ldots, Y_n)}{\partial \theta_r \partial \theta_k}\right]\right) \end{aligned}$$

Local approximation to the Fisher information matrix (this is the negative of the Hessian matrix):

$$i(\hat{\theta}) = -\left[\frac{\partial \ell(\theta; \mathrm{Y}_1, \ldots, \mathrm{Y}_n)}{\partial \theta_r \partial \theta_k}\right]$$

Newton-Raphson algorithm

$$\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} + \alpha \left[i(\hat{\theta}^{(k)})\right]^{-1} u(\hat{\theta}^{(k)})$$

where

(i) $\alpha = 1$ if $\hat{\theta}^{(k+1)}$ provides a larger value of the log-likelihood than $\hat{\theta}^{(k)}$, otherwise

(ii) Cycle through $\alpha = (0.5)^h$, for h=1,2,....,H, until $\hat{\theta}^{(k+1)}$ provides a larger value of the log-likelihood than $\hat{\theta}^{(k)}$.

Fisher-Scoring algorithm

$$\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} + \alpha \left[i(\hat{\theta}^{(k)})\right]^{-1} u(\hat{\theta}^{(k)})$$

where

(i) $\alpha = 1$ if $\hat{\theta}^{(k+1)}$ provides a larger value of the log-likelihood than $\hat{\theta}^{(k)}$, otherwise

(ii) Cycle through $\alpha = (0.5)^h$, for h=1,2,....,H, until $\hat{\theta}^{(k+1)}$ provides a larger value of the log-likelihood than $\hat{\theta}^{(k)}$.

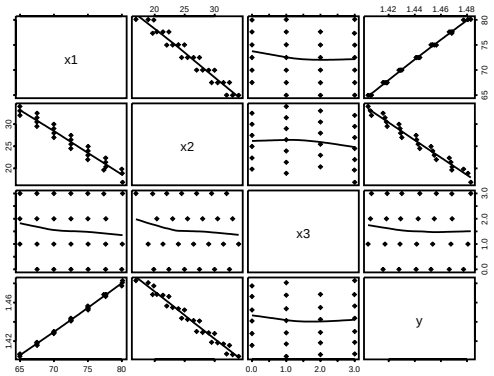
Large sample normal theory:

$$\hat{\theta} \overset{\bullet}{\sim} N\left(\theta, [i(\theta)]^{-1}\right)$$

```
# This is SPLUS code for fitting a nonlinear
# model to the mixture data from Myers (1994)
# Technometrics, 343-356. Maximum likelihood
# estimation is used.
# This file stored as     myers.ssc

# First access the MASS library

library(MASS, first=T)

# Enter the data stored in the file
#            myers.dat

# There are five numbers on each line in the
# following order:
#        mixture: Identification code
#        x1: percent nitroglycerine (NG)
#        x2: percent triacetin (TA)
#        x3: percent 2-nitrodiphenylamine (2N)
#         y: specific gravity of the mixture

myers <- read.table("myers.dat",
   col.names=c("mixture","x1","x2","x3","y"))
myers
```


	mixture	x1	x2	x3	y
1	1	79.98	19.85	0.00	1.4774
2	2	80.06	18.91	1.00	1.4807
3	3	80.10	16.87	3.00	1.4829
4	4	77.61	22.36	0.00	1.4664
5	5	77.60	21.38	1.00	1.4677
6	6	77.63	20.35	2.00	1.4686
7	7	77.34	19.65	2.99	1.4684
8	8	75.02	24.96	0.00	1.4524
9	9	75.03	23.95	1.00	1.4537
10	10	74.99	22.99	2.00	1.4549
11	11	74.98	22.00	3.00	1.4565
12	12	72.50	27.47	0.00	1.4410
13	13	72.50	26.48	1.00	1.4414
14	14	72.50	25.48	2.00	1.4426
15	15	72.49	24.49	3.00	1.4438
16	16	69.98	29.99	0.00	1.4279
17	17	69.98	29.00	1.00	1.4287
18	18	69.99	27.99	2.00	1.4291
19	19	69.99	26.99	3.00	1.4301
20	20	67.51	32.47	0.00	1.4157
21	21	67.50	31.47	1.00	1.4172
22	22	67.48	30.50	2.00	1.4183
23	23	67.49	29.49	3.00	1.4188
24	24	64.98	34.00	1.00	1.4042
25	25	64.98	33.00	2.00	1.4060
26	26	64.99	31.99	3.00	1.4068


```
#  Create a scatterplot matrix with smooth
#  curves.  Unix users should first use motif( )
#  to open a graphics window

 points.lines <- function(x, y){
 points(x, y)
 lines(loess.smooth(x, y, 0.90))}

 par(din=c(7,7), pch=18, mkh=.15, cex=1.2, lwd=3)
 pairs(myers[ ,-1], panel=points.lines)
```




```
# Use maximum likelihood estimation to model
# specific gravity of the mixture as the
# inverse of a linear combination of the
# percentages of NG, TA, and 2N plus a
# random error with a normal distribution.
# Initial values for coefficients are obtained
# from a regression of the inverse of the
# specific gravity of the mixture on a
# linear combination of the percentages of
# NG, TA, and 2N.

  myers$d <- 1/myers$y;
  myers.lm <- lm(d~x1+x2+x3-1,data=myers)
  b <- as.matrix(coef(myers.lm),ncol=1)
  b

          [,1]
x1 0.006289329
x2 0.008680710
x3 0.008073777
```

```
# Use the sum of squared errors to get a
# starting value for the variance estimate

  n <- length(myers$y)
  ss <- var(myers$y-1/fitted(myers.lm))*
          (n-1.)/(n-length(b))
  ss

[1] 7.5867e-007


# Compute maximum likelihood estimates for
# a model where the conditional responses
# are independent and normally distributed
# with homogeneous variances. Use deriv3( )
# to create a function to specify the
# log-likelihood and compute first and
# second partial derivatives. You must
# access the MASS library to use deriv3( ).
# Note that S-PLUS stores the value of
# pi in an object called pi. Furthermore,
# ms( ) is a minimization function so we
# must give it the negative of the log-
# likelihood.
```



```
lmyers <- deriv3( ~ 0.50*(log(2*pi)+log(ss)+
            ((y-1/(b1*x1+b2*x2+b3*x3))^2)/ss),
              c("b1", "b2", "b3", "ss"),
   function(y, x1, x2, x3, b1, b2, b3, ss) NULL)

# List the contents of this function
lmyers

function(y, x1, x2, x3, b1, b2, b3, ss)
{
    .expr9 <- ((b1 * x1) + (b2 * x2)) + (b3 * x3)
    .expr11 <- y - (1/.expr9)
    .expr12 <- .expr11^2
    .expr16 <- .expr9^2
    .expr17 <- x1/.expr16
    .expr19 <- 2 * (.expr17 * .expr11)
    .expr26 <- .expr16^2
    .expr33 <- x2/.expr16
    .expr36 <- 2 * (x2 * .expr9)
    .expr43 <- 0.5 * ((2 * ((.expr17 * .expr33) -
       (((x1 * .expr36)/.expr26) * .expr11)))/ss)
```



```
    .expr44 <- x3/.expr16
    .expr47 <- 2 * (x3 * .expr9)
    .expr54 <- 0.5 * ((2 * ((.expr17 * .expr44) -
        (((x1 * .expr47)/.expr26) * .expr11)))/ss)
    .expr55 <- ss^2
    .expr58 <-  - (0.5 * (.expr19/.expr55))
    .expr60 <- 2 * (.expr33 * .expr11)
    .expr78 <- 0.5 * ((2 * ((.expr33 * .expr44) -
        (((x2 * .expr47)/.expr26) * .expr11)))/ss)
    .expr81 <-  - (0.5 * (.expr60/.expr55))
    .expr83 <- 2 * (.expr44 * .expr11)
    .expr96 <-  - (0.5 * (.expr83/.expr55))
    .value <- 0.5 * (((log((2 * pi))) +
              (log(ss))) + (.expr12/ss))
    .grad <- array(0, c(length(.value), 4),
           list(NULL, c("b1", "b2", "b3", "ss")))
    .hess <- array(0, c(length(.value), 4, 4),
           list(NULL, c("b1", "b2", "b3", "ss"),
               c("b1", "b2", "b3", "ss")))
    .grad[, "b1"] <- 0.5 * (.expr19/ss)
    .grad[, "b2"] <- 0.5 * (.expr60/ss)
    .grad[, "b3"] <- 0.5 * (.expr83/ss)
    .grad[, "ss"] <- 0.5 * ((1/ss)
                      - (.expr12/.expr55))
```



```
  .hess[, "b1", "b1"] <- 0.5 * ((2 *
            ((.expr17 * .expr17) - (((x1 *
     (2 * (x1 * .expr9)))/.expr26) * .expr11)))/ss)
    .hess[, "b2", "b1"] <- .expr43
    .hess[, "b3", "b1"] <- .expr54
    .hess[, "ss", "b1"] <- .expr58
    .hess[, "b1", "b2"] <- .expr43
    .hess[, "b2", "b2"] <- 0.5 *
         ((2 * ((.expr33 * .expr33) - (((
     2 * .expr36)/.expr26) * .expr11)))/ss)
    .hess[, "b3", "b2"] <- .expr78
    .hess[, "ss", "b2"] <- .expr81
    .hess[, "b1", "b3"] <- .expr54
    .hess[, "b2", "b3"] <- .expr78
    .hess[, "b3", "b3"] <- 0.5 *
        ((2 * ((.expr44 * .expr44) - (((
       x3 * .expr47)/.expr26) * .expr11)))/ss)
    .hess[, "ss", "b3"] <- .expr96
    .hess[, "b1", "ss"] <- .expr58
    .hess[, "b2", "ss"] <- .expr81
    .hess[, "b3", "ss"] <- .expr96
    .hess[, "ss", "ss"] <-  - (0.5 *
           ((1/.expr55) - ((.expr12 * (2 *ss))
              /(.expr55^2))))
    attr(.value, "gradient") <- .grad
    attr(.value, "hessian") <- .hess
    .value
}
```

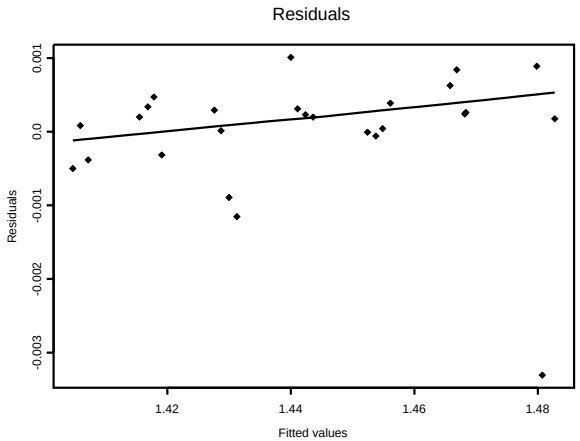
```
# We will trace the iterative process that
# minimizes the negative of the log-likelihood
# with the a function that provides the value
# of the negative of the log-likelihood,
# parameter estimates and the gradient.

tr.ms <- function(info, theta, grad, scale,
              flags, fit.pars){
 cat(signif(info[3]),":",signif(theta),"\n",
     ":",signif(grad[1:length(theta)]),"\n")
 invisible(list())
}

# Now maximize the likelihood

myers.ms <- ms(
       ~ lmyers(y, x1, x2, x3, b1, b2, b3, ss),
   data = myers,
  start = c(b1=b[1], b2=b[2], b3=b[3], ss=ss),
  trace = tr.ms,
 control=list(maxiter=50,tolerance=10^(-10),
       rel.tolerance=10^(-10),miniscale=1000))
```



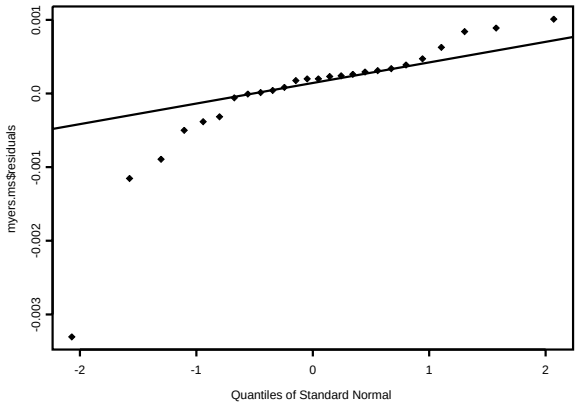
```
-147.8   : 0.00629  0.00868 0.008073 7.5867e-007
         : -7871.99 1124.22 1680.63  1977140
-147.893 : 0.00629  0.00868 0.008062 6.43125e-007
         : 1404.37 -205.418 -302.131 -862364
-147.905 : 0.00629  0.00868 0.008064 6.68382e-007
         : 52.9135 -7.81551 -11.4196 -63072.5
-147.905 : 0.00629  0.00868 0.008064 6.70535e-007
         : 0.16968 -0.02519 -0.036679 -404.412

#  Check the gradient

myers.ms$gradient

            b1               b2               b3
 4.500951e-006   -1.771914e-007   -7.388685e-007
          SS
 -0.01687477

# Compute fitted values and residuals

myers.ms$fitted <- (as.matrix(myers[ ,
       c("x1","x2","x3")])%*%
       as.vector(myers.ms$parameters[
                 c("b1","b2","b3")]))^(-1)
myers.ms$residuals <- myers$y-myers.ms$fitted
```



```
# Construct residual plots

scatter.smooth(myers.ms$fitted,
    myers.ms$residuals, span=1, degree=1,
    xlab="Fitted values",
    ylab="Residuals",
    main="Residuals")
```




```
qqnorm(myers.ms$residuals)
qqline(myers.ms$residuals)
```



```
# Compute confidence intervals for parameters
# using standard large sample normal theory.
# We will have to create our own function.
# The intervals( )  function cannot be
# applied to objects created by ms( ).

 conf.ms <- function(obj, level=0.95) {
    b <- coef(obj)
    thessian <- obj$hessian+t(obj$hessian)-
                diag(diag(obj.hessian))
    vcov <- solve(thessian)
    stderr <- sqrt(diag(vcov))
    low <- b - qnorm(1 - (1 - level)/2)*stderr
    high <- b + qnorm(1 - (1 - level)/2)*stderr
    a <- cbind(b, stderr, low, high)
    a }


conf.ms(myers.ms)

               b        stderr           low          high
b1 6.289741e-003 4.486347e-006 6.280948e-003 6.298534e-003
b2 8.680121e-003 1.201090e-005 8.656580e-003 8.703662e-003
b3 8.064233e-003 6.791295e-005 7.931126e-003 8.197339e-003
ss 6.705490e-007 1.859768e-007 3.060411e-007 1.035057e-006
```



```
#  In this case, the results for least squares
#  estimation gives essentially the same estimates

  myers$d <- 1/myers$y;
  myers.lm <- lm(d~x1+x2+x3-1,data=myers)
  b <- as.matrix(coef(myers.lm),ncol=1)

  myers.nls <- nls(
    formula = y ~ 1/(b1*x1+b2*x2+b3*x3),
       data = myers,
      start = c(b1=b[1], b2=b[2], b3=b[3]),
      trace = T)


1.74494e-005 : 0.00628933 0.00868071 0.00807378
1.74343e-005 : 0.00628974 0.00868012 0.00806423
```



```
summary(myers.nls)


Formula: y ~ 1/(b1 * x1 + b2 * x2
                          + b3 * x3)

Parameters:
        Value   Std. Error  t value
b1 0.00628974 4.77088e-006 1318.360
b2 0.00868012 1.27708e-005  679.682
b3 0.00806423 7.22151e-005  111.670

Residual standard error: 0.000870639
   on 23 degrees of freedom

Correlation of Parameter Estimates:
        b1      b2
b2 -0.9190
b3 -0.2950 -0.0228
```



```
#  The first case may be an outlier.  Delete
#  the first case and refit the model.

myers.1 <- myers[-1, ]

 myers.1$d <- 1/myers.1$y;
  myers.lm <- lm(d~x1+x2+x3-1,data=myers.1)
  b <- as.matrix(coef(myers.lm),ncol=1)
  b

          [,1]
x1 0.006281328
x2 0.008694754
x3 0.008166900

 n <- length(myers.1$y)
  ss <- var(myers.1$y-1/fitted(myers.lm))*
           (n-1.)/(n-length(b))
  ss
[1] 1.692968e-007
```

```
myers.ms <- ms(
   ~ lmyers(y, x1, x2, x3, b1, b2, b3, ss),
  data = myers.1,
  start = c(b1=b[1], b2=b[2], b3=b[3], ss=ss),
  trace = tr.ms,
  control=list(maxiter=50,tolerance=10^(-10)))
```

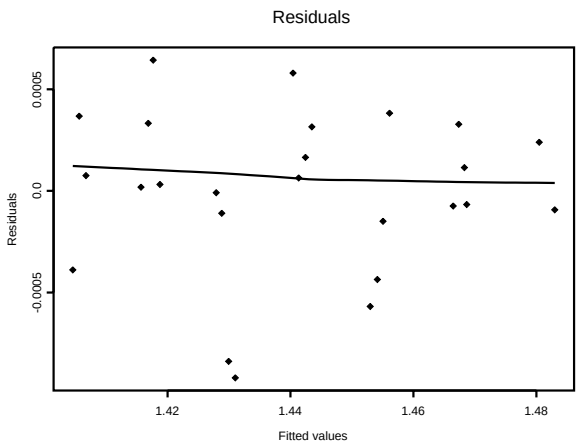
```
-160.922 : 0.00628133 0.00869475 0.0081669 1.69297e-007
         : 10524.6 -3105.81 495.381 8860170
-161.008 : 0.00628122 0.00869506 0.00816662 1.42527e-007
         : -1977.72 583.024 -93.0682 -3963970
-161.021 : 0.00628124 0.00869502 0.00816665 1.48434e-007
         : -78.7239 23.1976 -3.70426 -303121
-161.021 : 0.00628124 0.00869502 0.00816665 1.48965e-007
         : -0.280366 0.0825972 -0.0131917 -2151.45
```

```
#  Check the gradient
  myers.ms$gradient
```

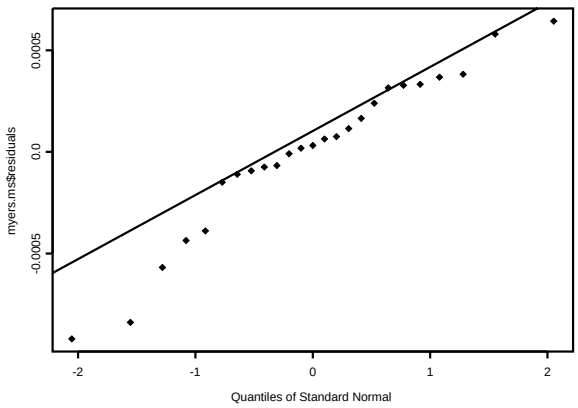
```
             b1              b2              b3
-7.452989e-006   1.975943e-006  -3.383961e-007
           ss
-0.1103133
```



```
#  Check residual plots.  The scatter.smooth
#  passes a smooth curve through the points
#  on the residual plot

myers.ms$fitted <-
   (as.matrix(myers.1[ ,c("x1","x2","x3")])%*%
    as.vector(myers.ms$parameters[
              c("b1","b2","b3")]))^(-1)
myers.ms$residuals <- myers.1$y-myers.ms$fitted

scatter.smooth(myers.ms$fitted,
    myers.ms$residuals, span=1, degree=1,
    xlab="Fitted values",
    ylab="Residuals",
    main="Residuals")
```


```
qqnorm(myers.ms$residuals)
qqline(myers.ms$residuals)
```



```
# Compute confidence intervals for parameters
# using standard large sample normal theory.
# We will have to create our own function.
# The intervals( )  function cannot be
# applied to objects created by ms( ).

 conf.ms <- function(obj, level=0.95) {
    b <- coef(obj)
    thessian<-obj$hessian+t(obj$hessian)-diag(diag(obj$hessian))
    vcov <- solve(thessian)
    stderr <- sqrt(diag(vcov))
    low <- b - qnorm(1 - (1 - level)/2)*stderr
    high <- b + qnorm(1 - (1 - level)/2)*stderr
    a <- cbind(b, stderr, low, high)
    a
}

conf.ms(myers.ms)
```

```
              b        stderr           low          high
b1 6.281238e-003 5.113040e-007 6.280236e-003 6.282240e-003
b2 8.695017e-003 1.412531e-006 8.692248e-003 8.697785e-003
b3 8.166655e-003 1.913770e-005 8.129146e-003 8.204164e-003
ss 1.489684e-007 4.213463e-008 6.638606e-008 2.315508e-007
```