Analysis of Nonlinear Models

**Linear Model:**

\[ Y = X\beta + \epsilon \]

where \( E(\epsilon) = 0 \) and \( Var(\epsilon) = \Sigma \)

Then, \( E(Y) = X\beta \) is a linear function of the unknown parameters \( \beta \):

\[ E(Y_i) = X_{i1}\beta_1 + \ldots + X_{ip}\beta_p \]

\[ i = 1, 2, \ldots, n \]

**Nonlinear model:** (additive random errors)

\[ Y_i = f(X_i, \beta) + \epsilon_i, \quad i = 1, \ldots, n \]

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

\( \beta \) is a vector of parameters

**Examples:**

\[ Y_i = \beta_0 + \beta_1X_{i1} + \beta_3X_{i2} + \epsilon_i \]

\[ Y_i = \beta_0 \exp(X_{i1}\beta_1) + \epsilon_i \]

**Intrinsically linear model:**

A model that can be transformed into a linear model.

**Example:**

\[ Y_i = \beta_0 \exp(\beta_1X_i)\epsilon_i \]

where \( \{\epsilon_i : i = 1, \ldots, n\} \) are random errors with \( E(\epsilon_i) = 1 \) and \( Var(\epsilon_i) = \sigma^2 \). Then

\[ E(Y_i) = \beta_0 \exp(\beta_1X_i) \]

and

\[ Var(Y_i) = \sigma^2[\beta_0 \exp(\beta_1X_i)]^2 \]

Transform to a linear model:

\[ \log(Y_i) = \log(\beta_0) + \beta_1X_i + \log(\epsilon_i) \]

\[ \uparrow \quad \uparrow \quad \text{call this } \gamma_0 \quad \text{call this } \eta_i \]

\[ = \gamma_0 + \beta_1X_i + \eta_i \]

**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 \( X_i \)
- \( \epsilon_i \) is a random error with \( E(\epsilon_i) = 0 \) and \( \text{Var}(\epsilon_i) = \sigma^2 \)
- \( \beta_0 > 0 \), \( \beta_1 > 0 \), \( \beta_2 > 0 \).

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
The 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 \( X_i \to \infty \) \( E(Y_i) \to \beta_0 \)

\[ \uparrow \text{this is the limiting response} \]

The Richard Growth Curve:

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

- when \( X_i = 0 \) \( E(Y_i) = \frac{\beta_0}{[1 + \beta_1]^{1/\beta_3}} \)
- as \( X_i \to \infty \) \( E(Y_i) \to \beta_0 \)

Weibull Growth Curve:

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

- when \( X_i = 0 \) \( E(Y_i) = \beta_0 - \beta_1 \)
- as \( X_i \to \infty \) \( E(Y_i) = \beta_0 \)
Mitchellich "Law":

- Used to model how the yield of a process increases with the increase in some factor
  - increase in yield of a crop when 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

**Mitchellich Growth Curves**

\[ Y_i = \beta_0 - (\beta_1)\beta_2^X_i + \epsilon_i \]

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-Menten equation:**

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

\[ Y = \text{velocity of uptake (\mu g/\ell/hr)} \]

\[ \beta_0 = \text{maximum velocity} \]

\[ \beta_1 = \text{transport constant \mu g/\ell} \]

\[ X = \text{concentration of substrate \mu g/\ell} \]

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

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

where

$D$ is the initial dose

$k_e$ is the elimination rate parameter

$k_a$ is the absorption 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.

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

where

$D$ is the initial dose

$\beta_1$ is $\log(k_e)$

$\beta_2$ is $\log(k_a)$

$\beta_0$ is $\log(k_c)$

Least Squares Estimation:

**Data:** $(Y_1, X_1^T)$

$(Y_2, X_2^T)$

$\vdots$

$(Y_n, X_n^T)$

**Model:** $Y_i = f(X_i, \beta) + \epsilon_i$

**Objective:** Find $b = \begin{bmatrix} b_1 \\ \vdots \\ b_k \end{bmatrix}$ to minimize

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

Estimating equations:

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

$$\vdots$$

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

These equations can be expressed as

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

where

$$D = \begin{bmatrix} \frac{\partial f(X_1, b)}{\partial b_1} & \cdots & \frac{\partial f(X_1, b)}{\partial b_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(X_n, b)}{\partial b_1} & \cdots & \frac{\partial f(X_n, b)}{\partial b_k} \end{bmatrix}$$
\[ Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} \quad \text{and} \quad f(X, b) = \begin{bmatrix} f(X_1, b) \\ \vdots \\ f(X_n, b) \end{bmatrix} \]

- 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 \( f(X_j, \beta) \) with a linear function of \( \beta \).
- Apply OLS estimation to the linear approximation to "update" the estimate of \( \beta \).
- Repeat this until convergence

Apply a Taylor series expansion:

For the \( j \)-th case at the \( i \)-th iteration, we have

\[ f(X_j, \beta) \approx f(X_j, b^{(i)}) + \sum_{\ell=1}^{k} \left[ \frac{\partial f(X_j, \beta)}{\partial \beta_\ell} \right]_{\beta = b^{(i)}} (\beta_\ell - b^{(i)}_\ell) + (\text{approximation error}) \]

Then

\[ Y_j = f(X_j, \beta) + \epsilon_j \]

\[ = f(X_j, b^{(i)}) + \sum_{\ell=1}^{k} \left[ \frac{\partial f(X_j, \beta)}{\partial \beta_\ell} \right]_{\beta = b^{(i)}} (\beta_\ell - b^{(i)}_\ell) + \epsilon_j \]
and

\[ Y = f(X, \beta) + \epsilon \]

\[ = f(X, b^{(i)}) + D^{(i)}(\beta - b^{(i)}) + \epsilon^* \]

or

\[ Y - f(X, b^{(i)}) = D^{(i)}(\beta - b^{(i)}) + \epsilon^* \]

This is

\[ D = \begin{bmatrix}
\frac{\partial f(X_1, \beta)}{\partial \beta_1} & \cdots & \frac{\partial f(X_1, \beta)}{\partial \beta_k} \\
\vdots & \ddots & \vdots \\
\frac{\partial f(X_n, \beta)}{\partial \beta_1} & \cdots & \frac{\partial f(X_n, \beta)}{\partial \beta_k}
\end{bmatrix} \]

evaluated at \( \beta = b^{(i)} \)

Selection of starting values:

- speed of convergence
- convergence to "local" minimum or a "global minimum"
- divergence
  - at each iteration check if
    \[ SSE^{(i+1)} < SSE^{(i)} \]
  - if not, you might use a "halving" step
    \[ b_{new}^{(i+1)} = b^{(i)} + .5(b^{(i+1)} - b^{(i)}) \]
- try different starting values
• Grid search: evaluate
  \[ g(b) = \sum_{j=1}^{n} (Y_j - f(X_j, b))^2 \]
  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 \( \sigma^2 \) with
  \[ MSE = \frac{SSE}{n-k} = \frac{\sum_{j=1}^{n} (Y_j - \hat{Y}_j)^2}{n-k} \]
  where \( \hat{Y}_j = f(X_j, b) \).

For large samples, when \( \epsilon_i \sim NID(0, \sigma^2) \), we have
  \[ \frac{(n-k)MSE}{\sigma^2} \sim \chi^2_{(n-k)} \]

Approximate t-tests:
  \[ H_0 : \beta_i = c_i \quad \text{vs.} \quad H_A : \beta_i \neq c_i \]

Reject \( H_0 \) if
  \[ |t| = \frac{|b_i - c_i|}{S_{b_i}} > t_{(n-k),a/2} \]

Approximate \((1 - a) \times 100\%\) confidence intervals:
  \[ b_i \pm t_{(n-k),a/2} S_{b_i} \]
  \[ \text{Square root of the } \ (i,i) \text{ element of } S_b \]
Example 11.1:
Weight loss from an obese patient
(V&R, Section 8.1)

\[ Y = \text{weight (in kg)} \]
\[ X = \text{time (in days)} \]

Obese patients in a weight rehabilitation 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 \texttt{nl( )} function and the \texttt{deriv( )} function for symbolic differentiation. Code is stored in the file wtloss.ssc

SAS: Use the \texttt{NLIN} procedure in SAS. Code is stored in the file wtloss.sas

```splus
# 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
```

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

# The following lines are for adding
# an axis for weight on the "lb" scale.
# pretty(): Returns a vector of ordered
# and equally spaced values that span
# the range of the input.

Wt.lbs <- pretty(range(wtloss$Weight*2.05))
axis(side=4, at=Wt.lbs/2.205,
     lab=Wt.lbs, srt=90)
mtext("Weight (lb)", side=4, line=3)
A negative exponential decay model:

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

\[ \beta_0 > 0, \beta_1 > 0, \beta_2 > 0 \]

where

- \( X_i \) is time (in days) since the start of the weight loss program
- \( Y_i \) is the observed weight at time \( X_i \)
- \( \beta_0 \) is the limiting weight (stable lean weight) as \( X \to \infty \)
- \( \beta_1 \) 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 derivatives.

```r
tlloss.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 value of the parameter estimates for successive iterations of the minimization procedure. This output was requested by the trace=TRUE option. Other information is obtained in the following way:

```r
summary(tlloss.fm)
```

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

Parameters:

<table>
<thead>
<tr>
<th>Value</th>
<th>Std. Error</th>
<th>t value</th>
</tr>
</thead>
<tbody>
<tr>
<td>b0 81.3736</td>
<td>2.26903 35.8627</td>
<td></td>
</tr>
<tr>
<td>b1 102.6840</td>
<td>2.08279 49.3013</td>
<td></td>
</tr>
<tr>
<td>b2 204.7350</td>
<td>7.63848 26.8031</td>
<td></td>
</tr>
</tbody>
</table>
Residual standard error:

0.894937 on 49 degrees of freedom

Correlation of Parameter Estimates:

\[
\begin{align*}
& \text{b0} & \text{b1} \\
& \text{b1} & -0.989 \\
& \text{b2} & -0.986 \quad 0.956
\end{align*}
\]

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

deviance(wtloss.fm)

[1] 39.2447

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

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

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

vcov(wtloss.fm)

\[
\begin{align*}
& \text{b0} & \text{b1} & \text{b2} \\
& \text{b0} & 5.148517 & -4.674626 & -17.08394 \\
& \text{b1} & -4.674626 & 4.338010 & 15.21100 \\
& \text{b2} & -17.083939 & 15.211003 & 58.34644
\end{align*}
\]

# 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"

# Plot the curve

par(fin=c(7.0,7.0),pch=18,mar=.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,fitted(wtloss.fm),lty=1,lwd=3)

Weight Loss Data

![Weight Loss Data Graph](image)
# 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=1, degree=1,
    main="Residuals")

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

# Now 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)
Obtain first partial derivatives of the mean weight at $X$ days after the start of the program

$$E(Y_i) = f(X_i, \beta) + \epsilon_i$$

where

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

We have

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

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

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

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

```r
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 create 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"] <- .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)))).value
```

```r
10s2
```

```r
10s3
```

```r
10s4
```

```r
10s5
```
# Now fit the model

```r
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
39.2947 : 81.3737 102.684 204.734

Confidence intervals:

\[ b_i \pm t_{(n-k),\alpha/2} \frac{S}{\sqrt{n}} \]

Using \( \alpha = 0.05 \), we have \( t_{49, 0.025} = 2.0096 \)

For \( \beta_0 \):

\[ 81.374 \pm (2.0096)(2.26902) \]
\[ \Rightarrow (76.81, 85.93) \]

For \( \beta_1 \):

\[ 102.684 \pm (2.0096)(2.08277) \]
\[ \Rightarrow (98.50, 106.87) \]

For \( \beta_2 \):

\[ 204.734 \pm (2.0096)(7.6384) \]
\[ \Rightarrow (189.4, 220.1) \]

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

- Partition the parameter vector \( \beta \) in \( E(Y) = f(X, \beta) \) 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 \( \hat{\beta} \) be the least squares estimator of \( \beta \) obtained by minimizing

\[ \sum_{j=1}^{n} (Y_j - f(X_j; \hat{\beta}))^2 \]

The minimum sum of squared residuals is

\[ RSS(\hat{\beta}) = \sum_{j=1}^{n} [Y_j - f(X_j; [\beta_{1,0}^{(1)} \beta_{2}^{(1)}])]^2 \]

Let \( \hat{\beta}_{2|1} \) be the “conditional” least squares estimate of \( \beta \) obtained by fixing \( \hat{\beta}_{1} = \beta_{1,0} \) and optimizing with respect to \( \beta_2 \), the remaining parameters. Minimize

\[ \sum_{j=1}^{n} [Y_j - f(X_j; [\beta_{1,0} \beta_{2}])]^2 \]

Then,

\[ \hat{\beta}_{2|1} = \begin{bmatrix} \beta_{1,0}^{(1)} \\ \beta_{2}^{(1)} \end{bmatrix} \]

The minimum sum of squared residuals is

\[ RSS(\hat{\beta}_{2|1}(\beta_{1,0})) = \sum_{j=1}^{n} [Y_j - f(X_j; \hat{\beta}_{2|1})]^2 \]
Approximate F-test:

\[ F(b_{1,0}) = \frac{RSS(\hat{\beta}_2|b_{1,0}) - RSS(\hat{\beta})}{(RSS(\hat{\beta})/(n - k))} \]

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

\[ \uparrow \]

number of parameters in \(\beta\)

Approximate t-statistic:

\[ t(b_{1,0}) = \text{sign}(b_{1,0} - \hat{b}_1) \sqrt{F(b_{1,0})} \]

with \(n - k\) d.f.

- 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

- Approximate \((1 - \alpha) \times 100\%\) confidence interval for \(\beta_1\):
  Find all values of \(b_{1,0}\) such that \(H_0: \beta_1 = b_{1,0}\) is not rejected at the \(\alpha\) level of significance, i.e.,
  find all values of \(b_{1,0}\) such that

\[ t_{(n-k),1-\alpha/2} \leq t(b_{1,0}) \leq t_{(n-k),\alpha/2} \]

# 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"))

\[
\begin{array}{cc}
2.5\% & 97.5\% \\
b0 & 76.49233 \quad 85.63038 \\
b1 & 98.78522 \quad 107.18805 \\
b2 & 190.42694 \quad 221.22585
\end{array}
\]
Inferences about functions of parameters:

Suppose you are interested in a function of the 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 $Y_0$:

\[ Y_0 = \beta_0 + \beta_1 e^{-(X_0/\beta_2)} \]
\[ \Rightarrow X_0 = h(\beta) = -\beta_2 \log \left( \frac{Y_0 - \beta_0}{\beta_1} \right) \]

Delta method:

If \( b \sim N(\beta, V) \) then

\[ h(b) \sim N(h(\beta), GVG^T) \]

where

\[
G = \begin{bmatrix}
\frac{\partial h_1(\beta)}{\partial \beta_1} & \frac{\partial h_1(\beta)}{\partial \beta_2} & \cdots & \frac{\partial h_1(\beta)}{\partial \beta_k} \\
\frac{\partial h_2(\beta)}{\partial \beta_1} & \frac{\partial h_2(\beta)}{\partial \beta_2} & \cdots & \frac{\partial h_2(\beta)}{\partial \beta_k} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial h_k(\beta)}{\partial \beta_1} & \frac{\partial h_k(\beta)}{\partial \beta_2} & \cdots & \frac{\partial h_k(\beta)}{\partial \beta_k}
\end{bmatrix}
\]

This is a large sample approximation, it is more accurate for larger sample sizes.
\[ S^2_Y = \hat{G} \tilde{V} \hat{G}^T \]
\[ = \hat{G} \begin{bmatrix} 5.14841 & -4.67453 & -17.08394 \\ -4.67453 & 4.33792 & 15.21101 \\ -17.08394 & 15.21101 & 58.34646 \end{bmatrix} \hat{G}^T \]
\[ = 0.033378 \]

Standard error:
\[ S_{\hat{Y}} = \sqrt{0.033378} = 0.182696 \]

Approximate 95% confidence interval:
\[ \hat{Y} \pm t_{49.025} S_{\hat{Y}} \]
\[ \Rightarrow (144.01, 144.75) \]

(ii) Time until weight reaches \( Y_0 = 100 \) kg.
\[ \bar{X} = h(b) = -b_2 \log \left( \frac{Y_0 - b_0}{b_1} \right) \]
\[ = (-141.911) \log \left( \frac{100 - 81.3738}{102.684} \right) \]
\[ = 349.5 \text{ days} \]

Apply the delta method:
\[ G = \begin{bmatrix} \frac{\partial h(\beta)}{\partial \beta_0} & \frac{\partial h(\beta)}{\partial \beta_1} & \frac{\partial h(\beta)}{\partial \beta_2} \end{bmatrix} \beta = b \]
\[ = \begin{bmatrix} b_2 & b_2 & -\log \left( \frac{Y - b_0}{b_1} \right) \end{bmatrix} \]
\[ = [10.991737 \ 1.993829 \ 1.70708] \]

# 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 derivatives 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),
                 list(NULL, 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", "b0"] <- .expr22
  .hess[,"b0", "b2"] <- .expr7
  .hess[,"b1", "b2"] <- .expr22
  .hess[,"b2", "b2"] <- 0
  attr(.value, "gradient") <- .grad
  attr(.value, "hessian") <- .hess
}

# 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")
  v <- (lam%*%vcov(obj)*lam)%*%matrix(1,3,1)
  n <- length(obj$fitted.values)
  - length(obj$parameters)
  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: 261.5140 2.795183 255.8966 267.1311
100 kg: 349.4987 8.175451 333.0696 365.9279
90 kg: 507.0944 31.217816 444.3598 569.8289

# Calculate confidence intervals by inverting
# large sample F-tests. Reparameterize the
# model in terms of the desired quantity. Then
# use deriv( ) to construct a function that
# provides the model formula and partial
# derivatives. This method may fail for some
# values of some quantities. Try obtaining a
# confidence interval for days until weight
# reaches 85 lb in the following example.

expn2 <- deriv("b0 + b1*(w0-b0)/b1"/(x/d0),
               c("b0","b1","d0"),
               function(b0,b1,d0,x,w0) { })
Relative to the ‘true’ curve, the time at which weight reaches $Y_0 = 100 \text{ kg}$ is

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

Then,

$$-\frac{1}{\beta_2} = \frac{1}{X_0} \log \left( \frac{Y_0 - \beta_0}{\beta_1} \right)$$

Substitute this formula for $\beta_2$ into the formula for the curve

$$Y = \beta_0 + \beta_1 e^{-X/\beta_2} + error$$

to obtain

$$Y = \beta_0 + \beta_1 \left( \frac{Y_0 - \beta_0}{\beta_1} \right)^{X/X_0} + error$$

We have expressed the model as a function of $\beta_0$, $\beta_1$, and $X_0$, the time at which the weight reaches $Y_0$.

---

# Define a function to calculate initial # values.

```r
wtloss.init <- function(obj, w0) {
  p <- coef(obj)
  d0 <- -log((w0 - p["b0"] / p["b1"]) * p["b2"])
  c(p[c("b0","b1")], d0 = as.vector(d0))
}
```

# Construct the confidence intervals

```r
result <- NULL
w0s <- c(110,100,90)
```

---

## Maximum Likelihood Estimation

**Example 11.2:** Specific gravity of a three component mixture.


$$Y_i|X_{1i} = x_{1i}, X_{2i} = x_{2i}, X_{3i} = x_{3i}) \sim N(\mu_i, \sigma^2)$$

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 $Y_1, Y_2, \ldots, Y_i$ are independent.

---

for (w0 in w0s) {
  fm <- nls(Weight ~ expn2(b0, b1, d0, Days, w0), data = wtloss, start = wtloss.init(wtloss.gr,w0))
  result <- rbind(result, c(coef(fm)["d0"],
                          confint(fm,"d0")) )
}
dimnames(result) <- list(paste(w0s,"kg:"),
                          c("time","low","high"))
result

<table>
<thead>
<tr>
<th>time</th>
<th>low</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>110 kg:</td>
<td>261.5140</td>
<td>256.2309</td>
</tr>
<tr>
<td>100 kg:</td>
<td>349.4987</td>
<td>334.7384</td>
</tr>
<tr>
<td>90 kg:</td>
<td>507.0944</td>
<td>457.5568</td>
</tr>
</tbody>
</table>
Joint likelihood function:
\[
L(\beta_1, \beta_2, \beta_3, \sigma^2 \mid data) = \prod_{i=1}^{n} \left( \frac{1}{2\pi \sigma^2} \exp \left( -\frac{1}{2\sigma^2} \left( Y_i - \beta_{2i} + \beta_{3i} + \frac{1}{\beta_1 \beta_2 \beta_3} \right)^2 \right) \right) \\
= \prod_{i=1}^{n} \left( \frac{1}{2\pi} \exp \left( -\frac{1}{2} \left( Y_i - \beta_{2i} + \beta_{3i} + \frac{1}{\beta_1 \beta_2 \beta_3} \right)^2 \right) \right) \\
= \prod_{i=1}^{n} f(Y_i; \beta_1, \beta_2, \beta_3, \sigma^2)
\]

Joint log-likelihood function:
\[
\ell(\beta_1, \beta_2, \beta_3, \sigma^2 \mid data) = \sum_{i=1}^{n} \log[f(Y_i; \beta_1, \beta_2, \beta_3)]
\]

The score function
\[
u(\theta) = \begin{bmatrix} u_1(\theta) \\ \vdots \\ u_n(\theta) \end{bmatrix} = \begin{bmatrix} \frac{\partial \ell(\theta; Y_1, \ldots, Y_n)}{\partial \beta_1} \\ \vdots \\ \frac{\partial \ell(\theta; Y_1, \ldots, Y_n)}{\partial \beta_n} \end{bmatrix}
\]

is the vector of first partial derivatives of the log-likelihood function with respect to the elements of
\[
\theta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}.
\]

The likelihood equations are
\[
u(\theta; Y_1, \ldots, Y_n) = 0
\]

The maximum likelihood estimator (MLE)
\[
\hat{\theta} = \begin{bmatrix} \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_n \end{bmatrix}
\]
is a solution to the likelihood equations, that maximizes the log-likelihood function.

Fisher information matrix:
\[
i(\theta) = \text{Var}(\nu(\theta; Y_1, \ldots, Y_n))
\]

\[
= E \left( \nu(\theta; Y_1, \ldots, Y_n) \nu(\theta; Y_1, \ldots, Y_n)^T \right)
\]

\[
= -E \left( \frac{\partial^2 \ell(\theta; Y_1, \ldots, Y_n)}{\partial \theta_r \partial \theta_k} \right)
\]

Local approximation to the Fisher information matrix (this is the negative of the Hessian matrix):
\[
i(\theta) = -\frac{\partial^2 \ell(\theta; Y_1, \ldots, Y_n)}{\partial \theta_r \partial \theta_k}
\]

Newton-Raphson algorithm:
\[
\theta^{(k+1)} = \theta^{(k)} + \alpha \left[ i(\theta^{(k)}) \right]^{-1} u(\theta^{(k)})
\]

where

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

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

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

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 \( h=1,2,\ldots,H \), until \( \hat{\theta}^{(k+1)} \) provides a larger value of the log-likelihood than \( \hat{\theta}^{(k)} \).

Large sample normal theory:

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

for "large" sample sizes

---

# 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

mixture x1  x2  x3  y
1  1 79.98 19.88 0.00 1.4774
2  2 80.06 18.91 1.00 1.4307
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 1.4694
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 74.96 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 1.4436
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 67.49 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

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

myers
# 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, msh=.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$y <- 1/myers$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(~ 0.5*(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))/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 <- array0(c(length(value), 4),  
list(NULL, c("b1", "b2", "b3", "ss")))  
.hess <- array0(c(length(value), 4),  
list(NULL, c("b1", "b2", "b3", "ss"),  
(c("b1", "b2", "b3", "ss")))  
.grad[1, "b1"] <- 0.5 * (.expr19/ss)  
.grad[1, "b2"] <- 0.5 * (.expr60/ss)  
.grad[1, "b3"] <- 0.5 * (.expr83/ss)  
.grad[1, "ss"] <- 0.5 * ((1/ss)  
- (.expr12/.expr55))  
}  

# 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",  
":\n",signif(grad[1:length(theta)]),"\n")  
invisible(list())  
}  

# Now maximize the likelihood

myers.ms <- ms(  
  layers(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, minscale=1000))  

# 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

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

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

---

# Compute confidence intervals for parameters

```r
conf.ms <- function(obj, level=0.95) {
  b <- coef(obj)
  vcov <- solve(obj\$hessian)
  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
}
```

```r
conf.ms(myers.ms)
```

```r
b stderr low high
b1 5.49741e-003 1.06899e-006 6.287670e-003 6.29153e-003
b2 4.50318e-003 1.95986e-006 4.674320e-003 4.68592e-003
b3 8.06433e-003 4.05974e-005 7.964559e-003 8.143907e-003
ss 6.70590e-007 1.865768e-007 3.060411e-007 1.035057e-006
```

---

# In this case, the results for least squares estimation are very similar, but not exactly the same.

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

```r
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)
```

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

```r
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)
```

```r
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 0.00864237 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

myers.ms <- ms(
  lm(myers(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)))

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$fitted <-
  as.matrix(myers.1[, c("x1","x2","x3")]) %*% as.vector(myers.ms$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")
# 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)
  vcov <- solve(obj$hessian)
  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)