Nonlinear Least Squares

Introduction

As an application of some of the optimization techniques presented earlier, the nonlinear least squares problem is discussed in this section. For simplicity of discussion a homoscedastic error distribution will be assumed. The nonlinear regression model considered is

\[ y_i = h(x_i, \theta) + \epsilon_i, \quad i = 1, \ldots, n \]  

(1)

where \( y_i \) are observed responses, \( x_i \) is a \( k \times 1 \) vector of known fixed values, \( \theta \) is a \( p \times 1 \) vector of unknown parameters and \( h() \) is a known function of \( x_i \) and \( \theta \) which is nonlinear in \( \theta \), respectively. The random errors \( \epsilon_i \) are assumed to be uncorrelated and have mean 0 and unknown variance \( \sigma^2 \). The least squares estimator of \( \theta \) minimizes

\[ f(\theta) = \sum_{i=1}^{n} [y_i - h(x_i, \theta)]^2, \]

the sum of squared residuals. Thus \( f(\theta) \) becomes the objective function to be minimized.

If the usual Newton-Raphson iteration

\[ \theta_{(i+1)} = \theta_{(i)} - H_{(i)}^{-1} g_{(i)} \]

is to be applied to this problem, the gradient vector \( g(\theta) = f'(\theta) \) and the Hessian \( H(\theta) = f''(\theta) \), must be computed and evaluated at each iteration. These are given by

\[ g(\theta) = f'(\theta) = \left( \frac{\partial f}{\partial \theta_1}, \frac{\partial f}{\partial \theta_2}, \ldots, \frac{\partial f}{\partial \theta_p} \right)^T, \]

where

\[ \frac{\partial f}{\partial \theta_j} = \sum_{i=1}^{n} \frac{\partial [y_i - h(x_i, \theta)]^2}{\partial \theta_j} \]

\[ = -2 \sum_{i=1}^{n} [y_i - h(x_i, \theta)] \frac{\partial h(x_i, \theta)}{\partial \theta_j}, \]

and

\[ H(\theta) = f''(\theta) = (h_{rs})_{p \times p}, \]

where

\[ h_{rs} = \frac{\partial^2 f}{\partial \theta_r \partial \theta_s} \]

\[ = -2 \sum_{i=1}^{n} \frac{\partial \left\{ [y_i - h(x_i, \theta)] \frac{\partial h(x_i, \theta)}{\partial \theta_s} \right\}}{\partial \theta_r} \]

\[ = -2 \sum_{i=1}^{n} \left\{ [y_i - h(x_i, \theta)] \frac{\partial^2 h(x_i, \theta)}{\partial \theta_r \partial \theta_s} - \frac{\partial h(x_i, \theta)}{\partial \theta_r} \frac{\partial h(x_i, \theta)}{\partial \theta_s} \right\}. \]
First, \( g(\theta) \) can be expressed in the form
\[
g(\theta) = f'(\theta) = -2J(\theta)^T r(\theta)
\]
where
\[
r(\theta) = (y_1 - h(x_1, \theta), y_2 - h(x_2, \theta), \ldots, y_n - h(x_n, \theta))^T
\]
and \( J(\theta) \) is the \( n \times p \) Jacobian of the \( n \times 1 \) vector \( h(\theta) = (h(x_1, \theta), h(x_2, \theta), \ldots h(x_n, \theta))^T \) given by
\[
J(\theta) = \begin{bmatrix}
\frac{\partial h(x_1, \theta)}{\partial \theta_1} & \frac{\partial h(x_1, \theta)}{\partial \theta_2} & \cdots & \frac{\partial h(x_1, \theta)}{\partial \theta_p} \\
\vdots & \ddots & \cdots & \vdots \\
\frac{\partial h(x_n, \theta)}{\partial \theta_1} & \frac{\partial h(x_n, \theta)}{\partial \theta_2} & \cdots & \frac{\partial h(x_n, \theta)}{\partial \theta_p}
\end{bmatrix}.
\]

Secondly, \( H(\theta) \) reduces to the form
\[
H(\theta) = f''(\theta) = -2 \left\{ \sum_{i=1}^{n} r_i(\theta) H_i(\theta) - J(\theta)^T J(\theta) \right\}
\]
where
\[
H_i(\theta) = \left( \frac{\partial^2 h(x_i, \theta)}{\partial \theta_r \partial \theta_s} \right)_{p \times p}.
\]

From these representations it must be carefully noted that in the nonlinear least squares problem, the function \( h \) must be evaluated \( n \) times for just a single evaluation of the objective function \( f \). This leads to the evaluation of \( n \) gradient vectors \((J(\theta))\) for a single evaluation of \( f'(\theta) \) and \( n \) Hessian matrices \( H_i(\theta) \) for a single evaluation of \( f''(\theta) \). Thus even for moderate sample sizes, enormous amount of computations are involved for just a single iteration of the standard Newton-Raphson if directly applied to the nonlinear least square problem. One approach to overcome this problem is to take advantage of the special form of the objective function, which in this case is a sum of squares, in order to construct specialized algorithms that are more efficient than the direct application of the general method. In practice, the following methods have been successfully used to solve the nonlinear least squares problem.

**Gauss-Newton Method**

This method involves approximating the function \( h \) in model 1 by the leading terms of its Taylor series expansion. Stopping at the linear term (second term) of the Taylor series expansion of \( h(x_i, \theta) \) about an initial guess \( \theta_{(0)} \), produces a linear regression problem that is an approximation to the nonlinear regression problem. Hence this is sometimes called the linearization method. Thus:
\[
h(x_i, \theta) \approx h(x_i, \theta_{(0)}) + (\theta - \theta_{(0)})^T h'(x_i, \theta_{(0)})
\]
where
\[ h'(x_i, \theta_{(0)}) = \left( \frac{\partial h(x_i, \theta)}{\partial \theta_1}, \frac{\partial h(x_i, \theta)}{\partial \theta_2}, \ldots, \frac{\partial h(x_i, \theta)}{\partial \theta_p} \right)^T \mid_{\theta = \theta_{(0)}}. \]

Substituting this approximation to \( h(x_i, \theta) \) in 1 and rearranging gives
\[ y_i - h(x_i, \theta_{(0)}) = h'(x_i, \theta_{(0)})^T (\theta - \theta_{(0)}) + \epsilon_i \]
or, equivalently
\[ y_i^* = z_i^T d + \epsilon_i \quad (i = 1, \ldots, n) \]
and in matrix notation
\[ y^* = Z d + \epsilon \]
where
\[ y^* = (y_1 - h(x_1, \theta_{(0)}), y_2 - h(x_2, \theta_{(0)}), \ldots, y_n - h(x_n, \theta_{(0)}))^T = r(\theta_{(0)}), \]
\[ Z_{n \times p} = \left( \begin{array}{cccc} \frac{\partial h(x_1, \theta)}{\partial \theta_1} & \frac{\partial h(x_1, \theta)}{\partial \theta_2} & \ldots & \frac{\partial h(x_1, \theta)}{\partial \theta_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h(x_n, \theta)}{\partial \theta_1} & \frac{\partial h(x_n, \theta)}{\partial \theta_2} & \ldots & \frac{\partial h(x_n, \theta)}{\partial \theta_p} \end{array} \right) \mid_{\theta = \theta_{(0)}} = J(\theta_{(0)}), \]
and,
\[ d = \theta - \theta_{(0)}. \]

The least squares estimate of \( d \) for model 2 is
\[ \hat{d} = (Z^T Z)^{-1} Z^T y^*, \]
that is
\[ \theta_{(1)} = \hat{\theta} = \theta_{(0)} + (Z^T Z)^{-1} Z^T y^*, \]
which suggests the following iterative method for computing the estimate of \( \theta \):
\[ \theta_{(i+1)} = \theta_{(i)} + (Z_{(i)}^T Z_{(i)})^{-1} Z_{(i)}^T y_{(i)}^*. \]
where \( Z_{(i)}, y_{(i)^*} \) are the same as \( Z \) and \( y^* \) above except that they are evaluated at \( \theta_{(i)} \) instead of \( \theta_{(0)} \). Noting that in our previous notation, \( Z^T Z = J(\theta_{(0)})^T J(\theta_{(0)}) \) and, \( Z^T y^* = J(\theta_{(0)})^T r(\theta_{(0)}) \), we recognize that the new iterative method selects the direction vector \( d_{(i)} \) as the solution to the set of equations
\[ J_{(i)}^T J_{(i)} d_{(i)} = J_{(i)}^T r_{(i)} \]
(3)
where \( \mathbf{J}(i) = \mathbf{J}(\theta(i)) \) and \( \mathbf{r}(i) = \mathbf{r}(\theta(i)) \), are the Jacobian and the residuals at the \( i^{th} \) step. The required iteration is thus expressed as

\[
\theta(i+1) = \theta(i) + \mathbf{d}(i).
\]

where \( \mathbf{d}(i) \) is called the Gauss-Newton correction. How well does Gauss-Newton work relative to the Newton-Raphson applied directly? Gauss-Newton method also requires the same number of function and gradient evaluations as does Newton-Raphson, but it avoids the \( n \) Hessian computations, \( \mathbf{H}_i, i = 1, \ldots, n \), required by Newton-Raphson. This is because the Hessian matrix is essentially approximated in the G-N method by \( \mathbf{J}^T(i)\mathbf{J}(i) \) as compared to the Hessian computed in the direct application of Newton’s algorithm (see Equation 2).

However, these computational savings may not be of much help because Gauss-Newton may converge very slowly or fail to converge at all, even from starting values close to the solution. This is because the term omitted in the approximation for the Hessian used above, \( \sum \mathbf{r}_i \mathbf{H}_i \), can become negligible only when the residuals \( \mathbf{r}_i = y_i - h(x_i, \theta) \approx 0 \). When the residuals are large near the solution \( \hat{\theta} \) the Gauss-Newton iteration will fail to converge or become extremely slow. This might be a consequence of taking too large a step in each iteration. Modified Gauss-Newton method consists of a step-length parameter \( \lambda(i) \) with \( 0 \leq \lambda_i \leq 1 \), included in the usual Gauss-Newton iteration:

\[
\theta(i+1) = \theta(i) + \lambda(i)\mathbf{d}(i)
\]

Then choose \( \lambda^* \) to minimize

\[
Q(\lambda) = f(\theta(i) + \lambda\mathbf{d}(i)).
\]

A consequence of this procedure is that at least

\[
f(\theta(i) + \lambda\mathbf{d}(i)) \leq f(\theta(i)),
\]

and thus a descent direction will always be used. One way to do the minimization of \( Q(\lambda) \) is to fit a parabola to the 3 points. \( \lambda = 0, 1/2, 1 \) The minimum of the fitted function can be directly computed:

\[
\lambda^* = \frac{1}{2} + \frac{1}{4} \frac{Q(0) - Q(1)}{Q(1) + Q(0) - 2Q(1/2)}
\]

Hartley (1961) originally suggested the procedure where he proposed using \( \lambda(i) = (\frac{1}{2})^{i+1} \) at each step \( i \). This procedure is known as step-halving in the literature. Hartley based his argument on the following assumption: Let \( \mathbf{J}^T(i)\mathbf{J}(i) \) be positive definite for all \( \theta \) in a bounded convex subset \( T \) in \( E^p \). Then it is possible to find \( \theta \in T \) such that

\[
f(\theta) \leq \inf_{\theta \in T} f(\theta)
\]

i.e., as long as \( \theta \) is in \( T \) the algorithm will converge to a solution. However, using a quadratic approximation like the parabola fit to determine \( \lambda_i \) at each step may perform better than step-halving. This method and earlier approaches are illustrated in a simple example.
**Example:** The Michaelis-Menten enzyme kinetic model

\[ y_i = \frac{\beta_1 x_i}{\beta_2 + x_i} + \epsilon_i \]

is fitted to the following data using nonlinear least squares.

<table>
<thead>
<tr>
<th>y</th>
<th>1.4</th>
<th>2</th>
<th>2.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

where \( y_i \) = initial reaction velocity and \( x = \) molar concentration of substrate.

\[
\frac{\partial h(x_i, \beta)}{\partial \beta_1} = \frac{x_i}{\beta_2 + x_i}, \quad \frac{\partial h(x_i, \beta)}{\partial \beta_2} = -\frac{\beta_1 x_i}{(\beta_2 + x_i)^2}
\]

Assume \( \beta_{(0)} = (2, 2)^T \). Then

\[
J_{(0)} = \begin{bmatrix}
\frac{\partial h(x_1, \beta)}{\partial \beta_1} & \frac{\partial h(x_1, \beta)}{\partial \beta_2} \\
\frac{\partial h(x_2, \beta)}{\partial \beta_1} & \frac{\partial h(x_2, \beta)}{\partial \beta_2} \\
\frac{\partial h(x_3, \beta)}{\partial \beta_1} & \frac{\partial h(x_3, \beta)}{\partial \beta_2}
\end{bmatrix} = \begin{bmatrix}
.333 & -.222 \\
.5 & -.25 \\
.6 & -.24
\end{bmatrix}
\]

\[
J_{(0)}^T J_{(0)} = \begin{bmatrix}
.7211 & -.3431 \\
-.3431 & .1695
\end{bmatrix}, \quad (J_{(0)}^T J_{(0)})^{-1} = \begin{bmatrix}
37.593 & 76.095 \\
76.095 & 159.93
\end{bmatrix}
\]

\[
h(\beta_{(0)}) = (h(x_1, \beta_{(0)}), h(x_2, \beta_{(0)}), h(x_3, \beta_{(0)}))^T = (.667, 1.0, 1.2)^T
\]

\[
r_{(0)} = y - h(\beta_{(0)}) = \begin{pmatrix}
1.4 & -.667 \\
2 & -1.0 \\
2.3 & -1.2
\end{pmatrix} = \begin{pmatrix}
.733 \\
1.0 \\
1.1
\end{pmatrix}
\]

\[
J_{(0)}^T r_{(0)} = \begin{pmatrix}
1.4043 \\
-.6767
\end{pmatrix}
\]

\[
d = (J_{(0)}^T J_{(0)})^{-1} J_{(0)}^T r_{(0)} = \begin{pmatrix}
1.298 \\
-1.364
\end{pmatrix}
\]

Let \( Q(\lambda) = f(\beta_{(0)} + \lambda d) \). The modified Gauss-Newton method requires the computation of an updated estimator \( \beta_{(1)} = \beta_{(0)} + \lambda^* d \) where \( \lambda^* \) minimizes \( Q(\lambda) \). It is not simple to minimize \( Q(\lambda) \). Alternately, a polynomial could be fitted to values of \( Q(\lambda) \) at \( \lambda = 0, 1/2, 1 \)
and the value of \( \lambda \) that minimizes the polynomial could be found. If the polynomial fitted is a parabola, then the value of \( \lambda \) that is the minimum is given once-and-for-all by the formula

\[
\lambda^* = \frac{1}{2} + \frac{1}{4} \left[ \left( Q(0) - Q(1) \right) / \left( Q(1) + Q(0) - 2Q(1/2) \right) \right].
\]

\[
Q(0) = f(\beta_0) = \sum_{i=1}^{3} (y_i - h(x_i, \beta_{(0)}))^2 = \sum_{i=1}^{3} r_i^2 = .733^2 + 1.0^2 + 1.1^2 = 2.7429
\]

\[
Q(1/2) = f(\beta_0 + (1/2)d) = \sum_{i=1}^{3} \left( y_i - \frac{2.649x_i}{1.318 + x_i} \right)^2 = .4400
\]

\[
Q(1) = f(\beta_0 + d) = \sum_{i=1}^{3} \left( y_i - \frac{3.298x_i}{.636 + x_i} \right)^2 = .8089
\]

Thus

\[
\lambda^* = .5 + .25(1.934/2.6718) = .681 \quad \text{and}
\]

\[
Q(.681) = f(\beta_0 + .681d) = \sum_{i=1}^{3} \left( y_i - \frac{2.884x_i}{1.071 + x_i} \right)^2 = .0454
\]

Thus the residual sum of squares after one iteration of M-G-N is .0454. However, if the Gauss-Newton procedure is used the residual SS is \( Q(1) = .8089 \), considerably larger.

**Hybrid Methods**

These methods combine Gauss-Newton with gradient methods such as *steepest descent* by linearly combining the normalized versions of the respective directions. Consider, for example, a hybrid of these two methods:

steepest descent direction \( d_{sd} = -g(\theta) \)

Gauss-Newton direction \( d_{gn} = (J^TJ)^{-1} J^T r \)
Normalize \( \mathbf{d}_{sd} \) and multiply by \(( \mathbf{d}_{gn}^T \mathbf{d}_{gn} )^{1/2} \) giving

\[
\mathbf{d}_{sd}^* = ( \mathbf{d}_{gn}^T \mathbf{d}_{gn} )^{1/2} \frac{\mathbf{d}_{sd}}{\sqrt{\mathbf{d}_{sd}^T \mathbf{d}_{sd}}}
\]

As a result, \( \mathbf{d}_{sd}^T \mathbf{d}_{sd}^* = \mathbf{d}_{gn}^T \mathbf{d}_{gn} \), i.e., the two normalized directions will have the same length. The hybrid direction is thus:

\[
\mathbf{d}^* = \delta \mathbf{d}_{gn} + (1 - \delta) \mathbf{d}_{sd}^*.
\]

Again, a linear search is performed to find the best mixing value \( \delta \). Select \( \delta \) such that \( f(\theta_{(i)}) + \delta \mathbf{d}_{gn} + (1 - \delta) \mathbf{d}_{sd}^* \) is a minimum and is less than \( f(\theta_{(i)}) \).

Gauss-Newton method is slow when the starting point further away but improves when closer to the minimum. Steepest descent is fast at the beginning but slows up later. So a good technique would be one that combines steepest descent at the early stages and switches to Gauss-Newton in latter iterations.

**Marquardt’s Procedure**

This is a special application of the Levenberg-Marquardt adjustment of the Hessian that was introduced as a modification to Newton’s method in the discussion on optimization methods. Here the adjustment is to be made to the matrix \( \mathbf{J}_{(i)}^T \mathbf{J}_{(i)} \) used in the Gauss-Newton scheme. The equation to be solved for obtaining step direction as given in equation 3 is now modified to

\[
( \mathbf{J}_{(i)}^T \mathbf{J}_{(i)} + \tau_{(i)} \mathbf{I} ) \mathbf{d}_{(i)} = \mathbf{J}_{(i)}^T \mathbf{r}_{(i)}
\]

where \( \mathbf{J}_{(i)} \) and \( \mathbf{r}_{(i)} \) are as defined before and \( \tau_{(i)} \) is a scalar.

Marquardt noted that gradient methods are not scale invariant: The transformation of the model to \( ay_i = ah(x_i, \theta) + a \epsilon_i \) results in

\[
\mathbf{d}_{sd} = -\mathbf{z}(\theta) = -2a^2 \mathbf{J}(\theta)^T \mathbf{r}(\theta)
\]

while Gauss-Newton is scale invariant. Thus Marquardt transforms \( \mathbf{J}_{(i)}^T \mathbf{J}_{(i)} \) to the correlation from and normalizes \( \mathbf{d}_{sd} \). Let \( \mathbf{C}_{(i)} = \mathbf{J}_{(i)}^T \mathbf{J}_{(i)} \) and \( \mathbf{z}_{(i)} = \mathbf{J}_{(i)}^T \mathbf{r}(\theta) \). Transforming \( \mathbf{C} \) to \( \mathbf{C}^* \) by \( \mathbf{C}^*_{(i)} = \mathbf{D}_{(i)} \mathbf{C}_{(i)} \mathbf{D}_{(i)}^T \) and \( \mathbf{z} \) to \( \mathbf{z}^* \) by \( \mathbf{z}^*_{(i)} = \mathbf{D}_{(i)} \mathbf{z}_{(i)} \), using \( \mathbf{D}_{(i)} = \text{diagonal}(1/\sqrt{\tau_{(i)}}) \), the problem is changed to solving the system

\[
( \mathbf{C}^*_{(i)} + \tau_{(i)} \mathbf{I} ) \mathbf{d}^*_{(i)} = \mathbf{z}^*_{(i)}
\]

If the solution is \( \mathbf{d}^*_{(i)} \) at the \( i \)th iteration then the correction vector is

\[
\mathbf{d}_{(i)} = \mathbf{D}_{(i)} \mathbf{d}^*_{(i)}
\]

i.e., \( \theta_{(i+1)} = \theta_{(i)} + \mathbf{d}_{(i)} \) where \( \mathbf{d}_{(i)} = \mathbf{D}_{(i)} \mathbf{d}^*_{(i)} \).

When \( \tau_{(i)} = 0 \) the system is

\[
\mathbf{C}^*_{(i)} \mathbf{d}_{(i)} = \mathbf{z}^*_{(i)}
\]
and the correction vector is identical to the G-N correction.

As \( \tau_{(i)} \) becomes larger and dominates the maximum eigenvalue of \( C^* \), \( \tau_{(i)} I \) dominates \( C^* + \tau I \) and the solution approaches

\[
d^* = \frac{1}{\tau_{(i)}} z^*_{(i)}
\]

which is a scalar times the gradient; thus large \( \tau_{(i)} \) implies that the direction would have steepest descent properties.

**The Marquardt Algorithm:**

1. Set starting values of \( \theta_{(0)}, \nu, \tau_{(0)} \) and compute \( f_0 \equiv f(\theta_0) \).

2. Set \( i = 0 \)

   (a) Solve

   \[
   \left( C^*_{(i)} + \frac{\tau_{(i)}}{\nu} I \right) d^*_{(i)} = z^*_{(i)}
   \]

   \[
   d_{(i)} = D_{(i)} d^*_{(i)}
   \]

   \[
   \theta_{(i+1)} = \theta_{(i)} + d_{(i)}
   \]

   Set \( f_1 = f(\theta_{(i+1)}) \)

   If \( f_1 < f_0 \), set \( \tau_{(i+1)} = \tau_{(i)}/\nu, \quad i = i + 1, \quad f_0 = f_1 \), go to (a).

   Else go to (b).

   (b) Solve

   \[
   \left( C^*_{(i)} + \tau_{(i)} I \right) d^*_{(i)} = z^*_{(i)}
   \]

   \[
   d_{(i)} = D_{(i)} d^*_{(i)}
   \]

   \[
   \theta_{(i+1)} = \theta_{(i)} + d_{(i)}
   \]

   Set \( f_2 = f(\theta_{(i+1)}) \)

   If \( f_2 < f_0 \), set \( \tau_{(i+1)} = \tau_{(i)}, \quad i = i + 1, \quad f_0 = f_2 \), go to (a).

   Else set \( \tau_{(i)} = \nu \tau_{(i)} \), go to (b).

**Notes:**

- Select \( \nu \) to be a value \( > 1 \) such as 2, 10, or 100.
- Select \( \tau \) to be large initially such as 1.
- Choice of these values depend on the starting values; for e.g. if starting values are far from the least squares solution, larger starting \( \tau \) may necessary to make progress initially.
- Remember that for smaller \( \tau \), the correction is closer to G-N correction and for larger \( \tau \) it is closer to the Steepest Descent correction.
- Stopping rules have not been incorporated into the above description.
Equivalency of Least Squares and Maximum Likelihood Estimation

If it is assumed that the \( \epsilon_i \)'s are i.i.d. Normal with variance \( \sigma^2 \), then \( y_i \)'s are Normal with mean \( h(x_i, \theta) \) and variance \( \sigma^2 \). Thus the log likelihood function of \((\theta, \sigma^2)\) is

\[
\ell(\theta, \sigma^2) = \text{constant} - (n/2) \log \sigma^2 - (1/2\sigma^2)f(\theta),
\]

where \( f(\theta) \) is the sum of squared residuals:

\[
f(\theta) = \sum_{i=1}^{n} [y_i - h(x_i, \theta)]^2
\]
as before. Given \( \theta \), the maximum likelihood estimate of \( \sigma^2 \) is obtained by taking the derivative of \( \ell(\theta, \sigma^2) \) with respect to \( \sigma^2 \) and equating to zero. Clearly,

\[
\hat{\sigma}^2 = f(\theta)/n.
\]

Hence the concentrated likelihood function for \( \theta \) is

\[
\ell_c(\theta) = \text{constant} - (n/2) \log f(\theta).
\]

It is clear thus that maximizing the likelihood for \( \theta \) is equivalent to minimizing the sum of squared residuals \( f(\theta) \).

Now computing the gradient of \( \ell(\theta, \sigma^2) \) we obtain the \((p+1)\) score vector

\[
u(\theta) = \frac{\partial \ell(\theta)}{\partial \theta} = \begin{pmatrix} (1/\sigma^2)J^T r \\ -n/2\sigma^2 + f(\theta)/2\sigma^4 \end{pmatrix},
\]

where \( J = J(\theta) \) is the \( n \times p \) Jacobian of the vector-valued function \( h(\theta) \) as before and \( r = r(\theta) = (y - h(\theta)) \) where \( h(\theta) = (h(x_1, \theta), h(x_2, \theta), \ldots h(x_n, \theta))^T \). By comparing \( u(\theta) \) the gradient of \( \beta, \sigma^2 \) in the regression model \( y = X\beta + \epsilon \):

\[
\begin{pmatrix} (1/\sigma^2)X^T r \\ -n/2\sigma^2 + r^T r/2\sigma^4 \end{pmatrix},
\]

where \( r = y - X\beta \), we note that the role of \( J \) is similar to that of the design matrix \( X \) in regression. Next, we note that the Hessian of \( \ell(\theta, \sigma^2) \) also resembles its regression counterpart:

\[
H(\theta) = \begin{pmatrix} (1/\sigma^2) \sum_i r_i H_i - J^T J & -J^T r \\ -J^T J & n/2\sigma^4 + (r^T r)/\sigma^6 \end{pmatrix},
\]

where \( r \) here is \( r(\theta) = y - h(\theta), \) \( H_i = H_i(\theta), \) and \( J = J(\theta) \) al defined previously. The expected information, \( E[H(\theta)] \) is

\[
I(\theta) = \begin{pmatrix} (1/\sigma^2) J^T J & 0 \\ 0 & n/2\sigma^4 \end{pmatrix},
\]
is almost identical to that in the regression case. Thus a scoring algorithm for estimating $\theta$ can be constructed immediately:

$$
\theta_{(i+1)} = \theta_{(i)} - I(\theta_{(i)})^{-1}u(\theta_{(i)})
$$

$$
= \theta_{(i)} - (J^T_{(i)}J_{(i)})^{-1}J^T_{(i)}r_{(i)}
$$

where $J_{(i)} = J(\theta_{(i)})$ and $r_{(i)} = y - h(\theta_{(i)})$.

****** R function nls() applied to fit Michaelis-Menten Model ****** to Bliss data

```r
> conc
[1] 0.19700 0.13850 0.06780 0.04170 0.02720 0.01450 0.00976 0.00816
> rate
[1] 21.5 21.0 19.0 16.5 14.5 11.0 8.5 7.0

> bliss=data.frame(Conc=conc,Rate=rate)
> bliss
   Conc Rate
1 0.19700 21.5
2 0.13850 21.0
3 0.06780 19.0
4 0.04170 16.5
5 0.02720 14.5
6 0.01450 11.0
7 0.00976  8.5
8 0.00816  7.0

> nls(Rate~(beta*Conc)/(Conc+gamma),data=bliss,start=list(beta=23.0,gamma=0.017),trace=T)
Nonlinear regression model
  model: Rate ~ (beta * Conc)/(Conc + gamma)
  data: bliss
   beta    gamma
23.61802771 0.01749682
residual sum-of-squares: 0.4758522
```

10
##### Functions to compute and plot contours of the Residual
##### Sum of squares surface from fitting the Michaelis-Menten Model

# Define function f(theta,x)
mmf=function(x,beta,gamma)
{
  return((beta*x)/(x+gamma))
}

# Compute Sum of Squared Residuals for data vectors (x,y) given beta, gamma
ssr=function(beta,gamma,x,y)
{
  r=y-mmff(x,beta,gamma)
  return(sum(r^2))
}

# Surface of Sum of Squared Residuals for a grid of values of beta, gamma
z=function(b,g,x,y)
{
  zurf=NULL
  for (i in seq(along=b)){
    for (j in seq(along=g))
      zurf = c(zurf,ssr(b[i],g[j],x,y))
  }
  return(matrix(zurf,length(b),length(g)))
}

# Data and grid of parameter values
conc=c(0.197, 0.1385, 0.0678, 0.0417, 0.0272, 0.0145, 0.00976, 0.00816)
rate=c(21.5, 21.0, 19.0, 16.5, 14.5, 11.0, 8.5, 7.0)

beta=seq(22.5,24.5,,100)
gamma=seq(0.016,0.02,,100)

# Compute and plot contours of surface
zdata=z(beta,gamma,conc,rate)
contour(beta,gamma,zdata,levels=c(seq(.4,.9,by=.1),1.0,1.2,1.5,2.0,4.0)
,ylab="Beta",ylab="Gamma")
title("Contour plot of the Sum of Squared Residuals \
Model fitted to Bliss data")
Contour plot of the Sum of Squared Residuals
Michaelis–Menten Model fitted to Bliss data
**Gauss-Newton Function and Usage**

```r
> micmen
function(xx, theta, derivs = T)
{
  x = xx[, 1]
  y = xx[, 2]
  denom = theta[2] + x
  r = y - ((theta[1] * x)/denom)
  if(!derivs)
    return(r)
  g = matrix(c(x/denom, (( - theta[1]) * x)/denom^2), ncol = 2)
  list(residual = r, gradient = g)
}

> gaussnewton
function(x, theta, eps, delta)
{
  k = 0
  theta.old = theta
  step = micmen(x, theta.old)
  res = step[[1]]
  g = step[[2]]
  ssr.old = sum(res^2)
  cat("Iter= ",k, " Residuals=",res,
       " Theta = ", theta.old,"SSRes=",ssr.old,fill=T)
  write(c(k, signif(ssr.old, 6), signif(theta.old, 6)), file = "micmen.d",
       ncol = 4, append = T)
  repeat {
    k = k + 1
    theta.new = theta.old + solve(t(g) %*% g, t(g) %*% res)
    step = micmen(x, theta.new)
    res = step[[1]]
    g = step[[2]]
    ssr.new = sum(res^2)
    write(c(k, signif(ssr.new, 6), signif(theta.new, 6)), file =
          "micmen.d", ncol = 4, append = T)
    if(sqrt(sum((theta.new - theta.old)^2) < eps && abs(ssr.old -
               ssr.new) < delta))
      break
    cat("Iter= ",k, " Residuals=",res,
         " Theta = ", theta.new," SSRes=",ssr.new,fill=T)
    theta.old = theta.new
    ssr.old = ssr.new
  }
  return(list(Residuals = res, SSR = ssr.new, theta = theta.new))
}
```

12
***** Gauss-Newton applied to fit Michaelis-Menten Model
***** to Bliss data

```r
> conc
[1] 0.19700 0.13850 0.06780 0.04170 0.02720 0.01450 0.00976 0.00816
> rate
[1] 21.5 21.0 19.0 16.5 14.5 11.0  8.5  7.0

> bliss.data=cbind(conc,rate)
> bliss.data
     conc rate
[1,] 0.19700 21.5
[2,] 0.13850 21.0
[3,] 0.06780 19.0
[4,] 0.04170 16.5
[5,] 0.02720 14.5
[6,] 0.01450 11.0
[7,] 0.00976  8.5
[8,] 0.00816  7.0

> gaussnewton(bliss.data,c(23.0,0.017),.00001,1.e-10)

$Residuals
 [1] -0.19146974  0.03100390  0.22670677 -0.13724020  0.12738188  0.29701810
 [7]  0.04296286 -0.51157378

$SSR
 [1] 0.4758522

$theta
     [,1]
[1,] 23.61802684
[2,]  0.01749682

Contents of micmen.d

0 1.28438 23 0.017
1 0.475901 23.619 0.0175081
2 0.475852 23.618 0.0174966
3 0.475852 23.618 0.0174968
4 0.475852 23.618 0.0174968
```