# CHAPTER 10

# Least-Squares Problems

In least-squares problems, the objective function $f$ has the following special form:

$$f(x) = \tfrac{1}{2} \sum_{j=1}^{m} r_j^2(x), \tag{10.1}$$

where each $r_j$ is a smooth function from $\mathbb{R}^n$ to $\mathbb{R}$. We refer to each $r_j$ as a *residual*, and we assume throughout this chapter that $m \geq n$.

Least-squares problems arise in many areas of applications, and may in fact be the largest source of unconstrained optimization problems. Many who formulate a parametrized

model for a chemical, physical, financial, or economic application use a function of the form (10.1) to measure the discrepancy between the model and the observed behavior of the system (see Example 2.1, for instance). By minimizing this function, they select values for the parameters that best match the model to the data. In this chapter we show how to devise efficient, robust minimization algorithms by exploiting the special structure of the function $f$ and its derivatives.

To see why the special form of $f$ often makes least-squares problems easier to solve than general unconstrained minimization problems, we first assemble the individual components $r_j$ from (10.1) into a *residual vector* $r : \mathbb{R}^n \to \mathbb{R}^m$, as follows

$$r(x) = (r_1(x), r_2(x), \ldots, r_m(x))^T. \tag{10.2}$$

Using this notation, we can rewrite $f$ as $f(x) = \frac{1}{2}\|r(x)\|_2^2$. The derivatives of $f(x)$ can be expressed in terms of the *Jacobian* $J(x)$, which is the $m \times n$ matrix of first partial derivatives of the residuals, defined by

$$J(x) = \left[\frac{\partial r_j}{\partial x_i}\right]_{\substack{j=1,2,\ldots,m \\ i=1,2,\ldots,n}} = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix}, \tag{10.3}$$

where each $\nabla r_j(x)$, $j = 1, 2, \ldots, m$ is the gradient of $r_j$. The gradient and Hessian of $f$ can then be expressed as follows:

$$\nabla f(x) = \sum_{j=1}^{m} r_j(x)\nabla r_j(x) = J(x)^T r(x), \tag{10.4}$$

$$\begin{aligned} \nabla^2 f(x) &= \sum_{j=1}^{m} \nabla r_j(x)\nabla r_j(x)^T + \sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x) \\ &= J(x)^T J(x) + \sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x). \end{aligned} \tag{10.5}$$

In many applications, the first partial derivatives of the residuals and hence the Jacobian matrix $J(x)$ are relatively easy or inexpensive to calculate. We can thus obtain the gradient $\nabla f(x)$ as written in formula (10.4). Using $J(x)$, we also can calculate the first term $J(x)^T J(x)$ in the Hessian $\nabla^2 f(x)$ without evaluating any second derivatives of the functions $r_j$. This availability of part of $\nabla^2 f(x)$ "for free" is the distinctive feature of least-squares problems. Moreover, this term $J(x)^T J(x)$ is often more important than the second summation term in (10.5), either because the residuals $r_j$ are close to affine near the solution (that is, the $\nabla^2 r_j(x)$ are relatively small) or because of small residuals (that

is, the $r_j(x)$ are relatively small). Most algorithms for nonlinear least-squares exploit these structural properties of the Hessian.

The most popular algorithms for minimizing (10.1) fit into the line search and trust-region frameworks described in earlier chapters. They are based on the Newton and quasi-Newton approaches described earlier, with modifications that exploit the particular structure of $f$.

Section 10.1 contains some background on applications. Section 10.2 discusses linear least-squares problems, which provide important motivation for algorithms for the nonlinear problem. Section 10.3 describes the major algorithms, while Section 10.4 briefly describes a variant of least squares known as orthogonal distance regression.

Throughout this chapter, we use the notation $\|\cdot\|$ to denote the Euclidean norm $\|\cdot\|_2$, unless a subscript indicates that some other norm is intended.

## 10.1 BACKGROUND

We discuss a simple parametrized model and show how least-squares techniques can be used to choose the parameters that best fit the model to the observed data.

### ❑ EXAMPLE 10.1

We would like to study the effect of a certain medication on a patient. We draw blood samples at certain times after the patient takes a dose, and measure the concentration of the medication in each sample, tabulating the time $t_j$ and concentration $y_j$ for each sample.

Based on our previous experience in such experiments, we find that the following function $\phi(x; t)$ provides a good prediction of the concentration at time $t$, for appropriate values of the five-dimensional parameter vector $x = (x_1, x_2, x_3, x_4, x_5)$:

$$\phi(x; t) = x_1 + tx_2 + t^2x_3 + x_4e^{-x_5t}. \tag{10.6}$$

We choose the parameter vector $x$ so that this model best agrees with our observation, in some sense. A good way to measure the difference between the predicted model values and the observations is the following least-squares function:

$$\tfrac{1}{2}\sum_{j=1}^{m}[\phi(x; t_j) - y_j]^2, \tag{10.7}$$

which sums the squares of the discrepancies between predictions and observations at each $t_j$. This function has precisely the form (10.1) if we define

$$r_j(x) = \phi(x; t_j) - y_j. \tag{10.8}$$

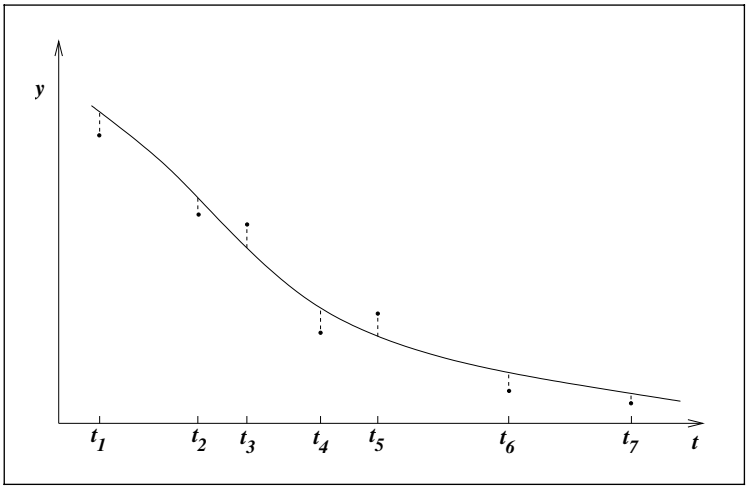


**Figure 10.1** Model (10.7) (smooth curve) and the observed measurements, with deviations indicated by vertical dotted lines.

Graphically, each term in (10.7) represents the square of the vertical distance between the curve $\phi(x;t)$ (plotted as a function of $t$) and the point $(t_j, y_j)$, for a fixed choice of parameter vector $x$; see Figure 10.1. The minimizer $x^*$ of the least-squares problem is the parameter vector for which the sum of squares of the lengths of the dotted lines in Figure 10.1 is minimized. Having obtained $x^*$, we use $\phi(x^*;t)$ to estimate the concentration of medication remaining in the patient's bloodstream at any time $t$.

❐

This model is an example of what statisticians call a *fixed-regressor model*. It assumes that the times $t_j$ at which the blood samples are drawn are known to high accuracy, while the observations $y_j$ may contain more or less random errors due to the limitations of the equipment (or the lab technician!)

In general data-fitting problems of the type just described, the ordinate $t$ in the model $\phi(x;t)$ could be a vector instead of a scalar. (In the example above, for instance, $t$ could have two dimensions, with the first dimension representing the time since the drug was admistered and the second dimension representing the weight of the patient. We could then use observations for an entire population of patients, not just a single patient, to obtain the "best" parameters for this model.)

The sum-of-squares function (10.7) is not the only way of measuring the discrepancy between the model and the observations. Other common measures include the maximum absolute value

$$\max_{j=1,2,\ldots,m} |\phi(x;t_j) - y_j| \tag{10.9}$$

and the sum of absolute values

$$\sum_{j=1}^{m} |\phi(x; t_j) - y_j|. \tag{10.10}$$

By using the definitions of the $\ell_\infty$ and $\ell_1$ norms, we can rewrite these two measures as

$$f(x) = \|r(x)\|_\infty, \quad f(x) = \|r(x)\|_1, \tag{10.11}$$

respectively. As we discuss in Chapter 17, the problem of minimizing the functions (10.11) can be reformulated a smooth constrained optimization problem.

In this chapter we focus only on the $\ell_2$-norm formulation (10.1). In some situations, there are statistical motivations for choosing the least-squares criterion. Changing the notation slightly, let the discrepancies between model and observation be denoted by $\epsilon_j$, that is,

$$\epsilon_j = \phi(x; t_j) - y_j.$$

It often is reasonable to assume that the $\epsilon_j$'s are independent and identically distributed with a certain variance $\sigma^2$ and probability density function $g_\sigma(\cdot)$. (This assumption will often be true, for instance, when the model accurately reflects the actual process, and when the errors made in obtaining the measurements $y_j$ do not contain a systematic bias.) Under this assumption, the likelihood of a particular set of observations $y_j$, $j = 1, 2, \ldots, m$, given that the actual parameter vector is $x$, is given by the function

$$p(y; x, \sigma) = \prod_{j=1}^{m} g_\sigma(\epsilon_j) = \prod_{j=1}^{m} g_\sigma(\phi(x; t_j) - y_j). \tag{10.12}$$

Given the observations $y_1, y_2, \ldots, y_m$, the "most likely" value of $x$ is obtained by maximizing $p(y; x, \sigma)$ with respect to $x$. The resulting value of $x$ is called the *maximum likelihood estimate*.

When we assume that the discrepancies follow a *normal* distribution, we have

$$g_\sigma(\epsilon) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right).$$

Substitution in (10.12) yields

$$p(y; x, \sigma) = (2\pi\sigma^2)^{-m/2} \exp\left(-\frac{1}{2\sigma^2} \sum_{j=1}^{m} [\phi(x; t_j) - y_j]^2\right).$$

For any fixed value of the variance $\sigma^2$, it is obvious that $p$ is maximized when the sum of squares (10.7) is minimized. To summarize: When the discrepancies are assumed to be independent and identically distributed with a normal distribution function, the maximum likelihood estimate is obtained by minimizing the sum of squares.

The assumptions on $\epsilon_j$ in the previous paragraph are common, but they do not describe the only situation for which the minimizer of the sum of squares makes good statistical sense. Seber and Wild [280] describe many instances in which minimization of functions like (10.7), or generalizations of this function such as

$$r(x)^T W r(x), \quad \text{where } W \in \mathbb{R}^{m \times m} \text{ is symmetric,}$$

is the crucial step in obtaining estimates of the parameters $x$ from observed data.

## 10.2 LINEAR LEAST-SQUARES PROBLEMS

Many models $\phi(x; t)$ in data-fitting problems are linear functions of $x$. In these cases, the residuals $r_j(x)$ defined by (10.8) also are linear, and the problem of minimizing (10.7) is called a *linear least-squares problem.* We can write the residual vector as $r(x) = Jx - y$ for some matrix $J$ and vector $y$, both independent of $x$, so that the objective is

$$f(x) = \tfrac{1}{2}\|Jx - y\|^2, \tag{10.13}$$

where $y = r(0)$. We also have

$$\nabla f(x) = J^T(Jx - y), \qquad \nabla^2 f(x) = J^T J.$$

(Note that the second term in $\nabla^2 f(x)$ (see (10.5)) disappears, because $\nabla^2 r_j = 0$ for all $j = 1, 2, \ldots, m$.) It is easy to see that the $f(x)$ in (10.13) is convex—a property that does not necessarily hold for the nonlinear problem (10.1). Theorem 2.5 tells us that any point $x^*$ for which $\nabla f(x^*) = 0$ is the global minimizer of $f$. Therefore, $x^*$ must satisfy the following linear system of equations:

$$J^T J x^* = J^T y. \tag{10.14}$$

These are known as the *normal equations* for (10.13).

We outline briefly three major algorithms for the unconstrained linear least-squares problem. We assume in most of our discussion that $m \geq n$ and that $J$ has full column rank.

The first and most obvious algorithm is simply to form and solve the system (10.14) by the following three-step procedure:

- compute the coefficient matrix $J^T J$ and the right-hand-side $J^T y$;
- compute the Cholesky factorization of the symmetric matrix $J^T J$;
- perform two triangular substitutions with the Cholesky factors to recover the solution $x^*$.

The Cholesky factorization

$$J^T J = \bar{R}^T \bar{R} \tag{10.15}$$

(where $\bar{R}$ is an $n \times n$ upper triangular with positive diagonal elements) is guaranteed to exist when $m \geq n$ and $J$ has rank $n$. This method is frequently used in practice and is often effective, but it has one significant disadvantage, namely, that the condition number of $J^T J$ is the square of the condition number of $J$. Since the relative error in the computed solution of a problem is usually proportional to the condition number, the Cholesky-based method may result in less accurate solutions than those obtained from methods that avoid this squaring of the condition number. When $J$ is ill conditioned, the Cholesky factorization process may even break down, since roundoff errors may cause small negative elements to appear on the diagonal during the factorization process.

A second approach is based on a QR factorization of the matrix $J$. Since the Euclidean norm of any vector is not affected by orthogonal transformations, we have

$$\|Jx - y\| = \|Q^T(Jx - y)\| \tag{10.16}$$

for any $m \times m$ orthogonal matrix $Q$. Suppose we perform a QR factorization with column pivoting on the matrix $J$ (see (A.24)) to obtain

$$J\Pi = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} = Q_1 R, \tag{10.17}$$

where

$\Pi$ is an $n \times n$ permutation matrix (hence, orthogonal);
$Q$ is $m \times m$ orthogonal;
$Q_1$ is the first $n$ columns of $Q$, while $Q_2$ contains the last $m - n$ columns;
$R$ is $n \times n$ upper triangular with positive diagonal elements.

By combining (10.16) and (10.17), we obtain

$$\begin{aligned}\|Jx - y\|_2^2 &= \left\| \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} (J\Pi\Pi^T x - y) \right\|_2^2 \\ &= \left\| \begin{bmatrix} R \\ 0 \end{bmatrix} (\Pi^T x) - \begin{bmatrix} Q_1^T y \\ Q_2^T y \end{bmatrix} \right\|^2 \\ &= \left\| R(\Pi^T x) - Q_1^T y \right\|_2^2 + \left\| Q_2^T y \right\|^2. \end{aligned} \tag{10.18}$$

No choice of $x$ has any effect on the second term of this last expression, but we can minimize $\|Jx - y\|$ by driving the first term to zero, that is, by setting

$$x^* = \Pi R^{-1} Q_1^T y.$$

(In practice, we perform a triangular substitution to solve $Rz = Q_1^T y$, then permute the components of $z$ to obtain $x^* = \Pi z$.)

This QR-based approach does not degrade the conditioning of the problem unnecessarily. The relative error in the final computed solution $x^*$ is usually proportional to the condition number of $J$, not its square, and this method is usually reliable. Some situations, however, call for greater robustness or more information about the sensitivity of the solution to perturbations in the data ($J$ or $y$). A third approach, based on the singular-value decomposition (SVD) of $J$, can be used in these circumstances. Recall from (A.15) that the SVD of $J$ is given by

$$J = U \begin{bmatrix} S \\ 0 \end{bmatrix} V^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} S \\ 0 \end{bmatrix} V^T = U_1 S V^T, \tag{10.19}$$

where

$U$ is $m \times m$ orthogonal;
$U_1$ contains the first $n$ columns of $U$, $U_2$ the last $m - n$ columns;
$V$ is $n \times n$ orthogonal;
$S$ is $n \times n$ diagonal, with diagonal elements $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$.

(Note that $J^T J = V S^2 V^T$, so that the columns of $V$ are eigenvectors of $J^T J$ with eigenvalues $\sigma_j^2$, $j = 1, 2, \ldots, n$.) By following the same logic that led to (10.18), we obtain

$$\begin{aligned}\|Jx - y\|^2 &= \left\| \begin{bmatrix} S \\ 0 \end{bmatrix} (V^T x) - \begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix} y \right\|^2 \\ &= \|S(V^T x) - U_1^T y\|^2 + \|U_2^T y\|^2. \end{aligned} \tag{10.20}$$

Again, the optimum is found by choosing $x$ to make the first term equal to zero; that is,

$$x^* = V S^{-1} U_1^T y.$$

Denoting the $i$th columns of $U$ and $V$ by $u_i \in \mathbb{R}^m$ and $v_i \in \mathbb{R}^n$, respectively, we have

$$x^* = \sum_{i=1}^{n} \frac{u_i^T y}{\sigma_i} v_i. \tag{10.21}$$

This formula yields useful information about the sensitivity of $x^*$. When $\sigma_i$ is small, $x^*$ is particularly sensitive to perturbations in $y$ that affect $u_i^T y$, and also to perturbations in $J$ that affect this same quantity. Such information is particularly useful when $J$ is nearly rank-deficient, that is, when $\sigma_n/\sigma_1 \ll 1$. It is sometimes worth the extra cost of the SVD algorithm to obtain this sensitivity information.

All three approaches above have their place. The Cholesky-based algorithm is particularly useful when $m \gg n$ and it is practical to store $J^T J$ but not $J$ itself. It can also be less expensive than the alternatives when $m \gg n$ and $J$ is sparse. However, this approach must be modified when $J$ is rank-deficient or ill conditioned to allow pivoting of the diagonal elements of $J^T J$. The QR approach avoids squaring of the condition number and hence may be more numerically robust. While potentially the most expensive, the SVD approach is the most robust and reliable of all. When $J$ is actually rank-deficient, some of the singular values $\sigma_i$ are exactly zero, and any vector $x^*$ of the form

$$x^* = \sum_{\sigma_i \neq 0} \frac{u_i^T y}{\sigma_i} v_i + \sum_{\sigma_i = 0} \tau_i v_i \tag{10.22}$$

(for arbitrary coefficients $\tau_i$) is a minimizer of (10.20). Frequently, the solution with smallest norm is the most desirable, and we obtain it by setting each $\tau_i = 0$ in (10.22). When $J$ has full rank but is ill conditioned, the last few singular values $\sigma_n, \sigma_{n-1}, \ldots$ are small relative to $\sigma_1$. The coefficients $u_i^T y/\sigma_i$ in (10.22) are particularly sensitive to perturbations in $u_i^T y$ when $\sigma_i$ is small, so an approximate solution that is less sentitive to perturbations than the true solution can be obtained by omitting these terms from the summation.

When the problem is very large, it may be efficient to use iterative techniques, such as the conjugate gradient method, to solve the normal equations (10.14). A direct implementation of conjugate gradients (Algorithm 5.2) requires one matrix vector multiplication with $J^T J$ to be performed at each iteration. This operation can be performed by means of successive multiplications by $J$ and $J^T$; we need only the ability to perform matrix-vector multiplications with these two matrices to implement this algorithm. Several modifications of the conjugate gradient approach have been proposed that involve a similar amount of work per iteration (one matrix-vector multiplication each with $J$ and $J^T$) but that have superior numerical properties. Some alternatives are described by Paige and Saunders [234],

who propose in particular an algorithm called LSQR which has become the basis of a highly successful code.

## 10.3 ALGORITHMS FOR NONLINEAR LEAST-SQUARES PROBLEMS

### THE GAUSS–NEWTON METHOD

We now describe methods for minimizing the nonlinear objective function (10.1) that exploit the structure in the gradient $\nabla f$ (10.4) and Hessian $\nabla^2 f$ (10.5). The simplest of these methods—the Gauss–Newton method—can be viewed as a modified Newton's method with line search. Instead of solving the standard Newton equations $\nabla^2 f(x_k)p = -\nabla f(x_k)$, we solve instead the following system to obtain the search direction $p_k^{\text{GN}}$:

$$J_k^T J_k p_k^{\text{GN}} = -J_k^T r_k. \tag{10.23}$$

This simple modification gives a number of advantages over the plain Newton's method. First, our use of the approximation

$$\nabla^2 f_k \approx J_k^T J_k \tag{10.24}$$

saves us the trouble of computing the individual residual Hessians $\nabla^2 r_j$, $j = 1, 2, \ldots, m$, which are needed in the second term in (10.5). In fact, if we calculated the Jacobian $J_k$ in the course of evaluating the gradient $\nabla f_k = J_k^T r_k$, the approximation (10.24) does not require any additional derivative evaluations, and the savings in computational time can be quite significant in some applications. Second, there are many interesting situations in which the first term $J^T J$ in (10.5) dominates the second term (at least close to the solution $x^*$), so that $J_k^T J_k$ is a close approximation to $\nabla^2 f_k$ and the convergence rate of Gauss–Newton is similar to that of Newton's method. The first term in (10.5) will be dominant when the norm of each second-order term (that is, $|r_j(x)|\|\nabla^2 r_j(x)\|$) is significantly smaller than the eigenvalues of $J^T J$. As mentioned in the introduction, we tend to see this behavior when either the residuals $r_j$ are small or when they are nearly affine (so that the $\|\nabla^2 r_j\|$ are small). In practice, many least-squares problems have small residuals at the solution, leading to rapid local convergence of Gauss–Newton.

A third advantage of Gauss–Newton is that whenever $J_k$ has full rank and the gradient $\nabla f_k$ is nonzero, the direction $p_k^{\text{GN}}$ is a descent direction for $f$, and therefore a suitable direction for a line search. From (10.4) and (10.23) we have

$$(p_k^{\text{GN}})^T \nabla f_k = (p_k^{\text{GN}})^T J_k^T r_k = -(p_k^{\text{GN}})^T J_k^T J_k p_k^{\text{GN}} = -\|J_k p_k^{\text{GN}}\|^2 \leq 0. \tag{10.25}$$

The final inequality is strict unless $J_k p_k^{\text{GN}} = 0$, in which case we have by (10.23) and full rank of $J_k$ that $J_k^T r_k = \nabla f_k = 0$; that is, $x_k$ is a stationary point. Finally, the fourth advantage of Gauss–Newton arises from the similarity between the equations (10.23) and the normal equations (10.14) for the linear least-squares problem. This connection tells us that $p_k^{\text{GN}}$ is in fact the solution of the linear least-squares problem

$$\min_p \tfrac{1}{2}\|J_k p + r_k\|^2. \tag{10.26}$$

Hence, we can find the search direction by applying linear least-squares algorithms to the subproblem (10.26). In fact, if the QR or SVD-based algorithms are used, there is no need to calculate the Hessian approximation $J_k^T J_k$ in (10.23) explicitly; we can work directly with the Jacobian $J_k$. The same is true if we use a conjugate-gradient technique to solve (10.26). For this method we need to perform matrix-vector multiplications with $J_k^T J_k$, which can be done by first multiplying by $J_k$ and then by $J_k^T$.

If the number of residuals $m$ is large while the number of variables $n$ is relatively small, it may be unwise to store the Jacobian $J$ explicitly. A preferable strategy may be to calculate the matrix $J^T J$ and gradient vector $J^T r$ by evaluating $r_j$ and $\nabla r_j$ successively for $j = 1, 2, \ldots, m$ and performing the accumulations

$$J^T J = \sum_{i=1}^{m} (\nabla r_j)(\nabla r_j)^T, \qquad J^T r = \sum_{i=1}^{m} r_j (\nabla r_j). \tag{10.27}$$

The Gauss–Newton steps can then be computed by solving the system (10.23) of normal equations directly.

The subproblem (10.26) suggests another motivation for the Gauss–Newton search direction. We can view this equation as being obtained from a linear model for the the vector function $r(x_k + p) \approx r_k + J_k p$, substituted into the function $\frac{1}{2}\|\cdot\|^2$. In other words, we use the approximation

$$f(x_k + p) = \tfrac{1}{2}\|r(x_k + p)\|^2 \approx \tfrac{1}{2}\|J_k p + r_k\|^2,$$

and choose $p_k^{\text{GN}}$ to be the minimizer of this approximation.

Implementations of the Gauss–Newton method usually perform a line search in the direction $p_k^{\text{GN}}$, requiring the step length $\alpha_k$ to satisfy conditions like those discussed in Chapter 3, such as the Armijo and Wolfe conditions; see (3.4) and (3.6).

### CONVERGENCE OF THE GAUSS–NEWTON METHOD

The theory of Chapter 3 can applied to study the convergence properties of the Gauss–Newton method. We prove a global convergence result under the assumption that the Jacobians $J(x)$ have their singular values uniformly bounded away from zero in the

region of interest; that is, there is a constant $\gamma > 0$ such that

$$\|J(x)z\| \geq \gamma\|z\| \tag{10.28}$$

for all $x$ in a neighborhood $\mathcal{N}$ of the level set

$$\mathcal{L} = \{x \mid f(x) \leq f(x_0)\}, \tag{10.29}$$

where $x_0$ is the starting point for the algorithm. We assume here and in the rest of the chapter that $\mathcal{L}$ is bounded. Our result is a consequence of Theorem 3.2.

**Theorem 10.1.**

*Suppose each residual function $r_j$ is Lipschitz continuously differentiable in a neighborhood $\mathcal{N}$ of the bounded level set (10.29), and that the Jacobians $J(x)$ satisfy the uniform full-rank condition (10.28) on $\mathcal{N}$. Then if the iterates $x_k$ are generated by the Gauss–Newton method with step lengths $\alpha_k$ that satisfy (3.6), we have*

$$\lim_{k\to\infty} J_k^T r_k = 0.$$

PROOF. First, we note that the neighborhood $\mathcal{N}$ of the bounded level set $\mathcal{L}$ can be chosen small enough that the following properties are satisfied for some positive constants $L$ and $\beta$:

$$|r_j(x)| \leq \beta \quad \text{and} \quad \|\nabla r_j(x)\| \leq \beta,$$
$$|r_j(x) - r_j(\tilde{x})| \leq L\|x - \tilde{x}\| \quad \text{and} \quad \|\nabla r_j(x) - \nabla r_j(\tilde{x})\| \leq L\|x - \tilde{x}\|,$$

for all $x, \tilde{x} \in \mathcal{N}$ and all $j = 1, 2, \ldots, m$. It is easy to deduce that there exists a constant $\bar{\beta} > 0$ such that $\|J(x)^T\| = \|J(x)\| \leq \bar{\beta}$ for all $x \in \mathcal{L}$. In addition, by applying the results concerning Lipschitz continuity of products and sums (see for example (A.43)) to the gradient $\nabla f(x) = \sum_{j=1}^m r_j(x)\nabla r_j(x)$, we can show that $\nabla f$ is Lipschitz continuous. Hence, the assumptions of Theorem 3.2 are satisfied.

We check next that the angle $\theta_k$ between the search direction $p_k^{\text{GN}}$ and the negative gradient $-\nabla f_k$ is uniformly bounded away from $\pi/2$. From (3.12), (10.25), and (10.28), we have for $x = x_k \in \mathcal{L}$ and $p^{\text{GN}} = p_k^{\text{GN}}$ that

$$\cos\theta_k = -\frac{(\nabla f)^T p^{\text{GN}}}{\|p^{\text{GN}}\|\|\nabla f\|} = \frac{\|Jp^{\text{GN}}\|^2}{\|p^{\text{GN}}\|\|J^T J p^{\text{GN}}\|} \geq \frac{\gamma^2\|p^{\text{GN}}\|^2}{\bar{\beta}^2\|p^{\text{GN}}\|^2} = \frac{\gamma^2}{\bar{\beta}^2} > 0.$$

It follows from (3.14) in Theorem 3.2 that $\nabla f(x_k) \to 0$, giving the result. □

If $J_k$ is rank-deficient for some $k$ (so that a condition like (10.28) is not satisfied), the coefficient matrix in (10.23) is singular. The system (10.23) still has a solution, however, because of the equivalence between this linear system and the minimization problem (10.26).

In fact, there are infinitely many solutions for $p_k^{\mathrm{GN}}$ in this case; each of them has the form of (10.22). However, there is no longer an assurance that $\cos\theta_k$ is uniformly bounded away from zero, so we cannot prove a result like Theorem 10.1.

The convergence of Gauss–Newton to a solution $x^*$ can be rapid if the leading term $J_k^T J_k$ dominates the second-order term in the Hessian (10.5). Suppose that $x_k$ is close to $x^*$ and that assumption (10.28) is satisfied. Then, applying an argument like the Newton's method analysis (3.31), (3.32), (3.33) in Chapter 3, we have for a unit step in the Gauss–Newton direction that

$$\begin{aligned} x_k + p_k^{\mathrm{GN}} - x^* &= x_k - x^* - [J^T J(x_k)]^{-1} \nabla f(x_k) \\ &= [J^T J(x_k)]^{-1} \left[ J^T J(x_k)(x_k - x^*) + \nabla f(x^*) - \nabla f(x_k) \right], \end{aligned}$$

where $J^T J(x)$ is shorthand notation for $J(x)^T J(x)$. Using $H(x)$ to denote the second-order term in (10.5), we have from (A.57) that

$$\begin{aligned} \nabla f(x_k) - \nabla f(x^*) = &\int_0^1 J^T J(x^* + t(x_k - x^*))(x_k - x^*)\, dt \\ &+ \int_0^1 H(x^* + t(x_k - x^*))(x_k - x^*)\, dt. \end{aligned}$$

A similar argument as in (3.32), (3.33), assuming Lipschitz continuity of $H(\cdot)$ near $x^*$, shows that

$$\begin{aligned} &\|x_k + p_k^{\mathrm{GN}} - x^*\| \\ &\quad \le \int_0^1 \|[J^T J(x_k)]^{-1} H(x^* + t(x_k - x^*))\| \|x_k - x^*\|\, dt + O(\|x_k - x^*\|^2) \\ &\quad \approx \|[J^T J(x^*)]^{-1} H(x^*)\| \, \|x_k - x^*\| + O(\|x_k - x^*\|^2). \end{aligned} \tag{10.30}$$

Hence, if $\|[J^T J(x^*)]^{-1} H(x^*)\| \ll 1$, we can expect a unit step of Gauss–Newton to move us much closer to the solution $x^*$, giving rapid local convergence. When $H(x^*) = 0$, the convergence is actually quadratic.

When $n$ and $m$ are both large and the Jacobian $J(x)$ is sparse, the cost of computing steps exactly by factoring either $J_k$ or $J_k^T J_k$ at each iteration may become quite expensive relative to the cost of function and gradient evaluations. In this case, we can design *inexact* variants of the Gauss–Newton algorithm that are analogous to the inexact Newton algorithms discussed in Chapter 7. We simply replace the Hessian $\nabla^2 f(x_k)$ in these methods by its approximation $J_k^T J_k$. The positive semidefiniteness of this approximation simplifies the resulting algorithms in several places.

### THE LEVENBERG–MARQUARDT METHOD

Recall that the Gauss–Newton method is like Newton's method with line search, except that we use the convenient and often effective approximation (10.24) for the Hessian. The Levenberg–Marquardt method can be obtained by using the same Hessian approximation, but replacing the line search with a trust-region strategy. The use of a trust region avoids one of the weaknesses of Gauss–Newton, namely, its behavior when the Jacobian $J(x)$ is rank-deficient, or nearly so. Since the same Hessian approximations are used in each case, the local convergence properties of the two methods are similar.

The Levenberg–Marquardt method can be described and analyzed using the trust-region framework of Chapter 4. (In fact, the Levenberg–Marquardt method is sometimes considered to be the progenitor of the trust-region approach for general unconstrained optimization discussed in Chapter 4.) For a spherical trust region, the subproblem to be solved at each iteration is

$$\min_p \tfrac{1}{2}\|J_k p + r_k\|^2, \quad \text{subject to } \|p\| \leq \Delta_k, \tag{10.31}$$

where $\Delta_k > 0$ is the trust-region radius. In effect, we are choosing the model function $m_k(\cdot)$ in (4.3) to be

$$m_k(p) = \tfrac{1}{2}\|r_k\|^2 + p^T J_k^T r_k + \tfrac{1}{2} p^T J_k^T J_k p. \tag{10.32}$$

We drop the iteration counter $k$ during the rest of this section and concern ourselves with the subproblem (10.31). The results of Chapter 4 allow us to characterize the solution of (10.31) in the following way: When the solution $p^{\text{GN}}$ of the Gauss–Newton equations (10.23) lies strictly inside the trust region (that is, $\|p^{\text{GN}}\| < \Delta$), then this step $p^{\text{GN}}$ also solves the subproblem (10.31). Otherwise, there is a $\lambda > 0$ such that the solution $p = p^{\text{LM}}$ of (10.31) satisfies $\|p\| = \Delta$ and

$$\left(J^T J + \lambda I\right) p = -J^T r. \tag{10.33}$$

This claim is verified in the following lemma, which is a straightforward consequence of Theorem 4.1 from Chapter 4.

**Lemma 10.2.**

*The vector $p^{\text{LM}}$ is a solution of the trust-region subproblem*

$$\min_p \|Jp + r\|^2, \qquad \text{subject to } \|p\| \leq \Delta,$$

*if and only if $p^{\text{LM}}$ is feasible and there is a scalar $\lambda \geq 0$ such that*

$$(J^T J + \lambda I)p^{\text{LM}} = -J^T r, \tag{10.34a}$$

$$\lambda(\Delta - \|p^{\text{LM}}\|) = 0. \tag{10.34b}$$

PROOF. In Theorem 4.1, the semidefiniteness condition (4.8c) is satisfied automatically, since $J^T J$ is positive semidefinite and $\lambda \geq 0$. The two conditions (10.34a) and (10.34b) follow from (4.8a) and (4.8b), respectively. □

Note that the equations (10.33) are just the normal equations for the following linear least-squares problem:

$$\min_p \tfrac{1}{2} \left\| \begin{bmatrix} J \\ \sqrt{\lambda} I \end{bmatrix} p + \begin{bmatrix} r \\ 0 \end{bmatrix} \right\|^2. \tag{10.35}$$

Just as in the Gauss–Newton case, the equivalence between (10.33) and (10.35) gives us a way of solving the subproblem without computing the matrix–matrix product $J^T J$ and its Cholesky factorization.

### IMPLEMENTATION OF THE LEVENBERG–MARQUARDT METHOD

To find a value of $\lambda$ that approximately matches the given $\Delta$ in Lemma 10.2, we can use the rootfinding algorithm described in Chapter 4. It is easy to safeguard this procedure: The Cholesky factor $R$ is guaranteed to exist whenever the current estimate $\lambda^{(\ell)}$ is positive, since the approximate Hessian $B = J^T J$ is already positive semidefinite. Because of the special structure of $B$, we do not need to compute the Cholesky factorization of $B + \lambda I$ from scratch in each iteration of Algorithm 4.1. Rather, we present an efficient technique for finding the following QR factorization of the coefficient matrix in (10.35):

$$\begin{bmatrix} R_\lambda \\ 0 \end{bmatrix} = Q_\lambda^T \begin{bmatrix} J \\ \sqrt{\lambda} I \end{bmatrix} \tag{10.36}$$

($Q_\lambda$ orthogonal, $R_\lambda$ upper triangular). The upper triangular factor $R_\lambda$ satisfies $R_\lambda^T R_\lambda = (J^T J + \lambda I)$.

We can save computer time in the calculation of the factorization (10.36) by using a combination of Householder and Givens transformations. Suppose we use Householder transformations to calculate the QR factorization of $J$ alone as

$$J = Q \begin{bmatrix} R \\ 0 \end{bmatrix}. \tag{10.37}$$

We then have

$$\begin{bmatrix} R \\ 0 \\ \sqrt{\lambda} I \end{bmatrix} = \begin{bmatrix} Q^T & \\ & I \end{bmatrix} \begin{bmatrix} J \\ \sqrt{\lambda} I \end{bmatrix}. \tag{10.38}$$

The leftmost matrix in this formula is upper triangular except for the $n$ nonzero terms of the matrix $\lambda I$. These can be eliminated by a sequence of $n(n+1)/2$ Givens rotations, in which the diagonal elements of the upper triangular part are used to eliminate the nonzeros of $\lambda I$ and the fill-in terms that arise in the process. The first few steps of this process are as follows:

rotate row $n$ of $R$ with row $n$ of $\sqrt{\lambda}I$, to eliminate the $(n, n)$ element of $\sqrt{\lambda}I$;

rotate row $n-1$ of $R$ with row $n-1$ of $\sqrt{\lambda}I$ to eliminate the $(n-1, n-1)$ element of the latter matrix. This step introduces fill-in in position $(n-1, n)$ of $\sqrt{\lambda}I$, which is eliminated by rotating row $n$ of $R$ with row $n-1$ of $\sqrt{\lambda}I$, to eliminate the fill-in element at position $(n-1, n)$;

rotate row $n-2$ of $R$ with row $n-2$ of $\sqrt{\lambda}I$, to eliminate the $(n-2)$ diagonal in the latter matrix. This step introduces fill-in in the $(n-2, n-1)$ and $(n-2, n)$ positions, which we eliminate by $\cdots$

and so on. If we gather all the Givens rotations into a matrix $\bar{Q}_\lambda$, we obtain from (10.38) that

$$\bar{Q}_\lambda^T \begin{bmatrix} R \\ 0 \\ \sqrt{\lambda}I \end{bmatrix} = \begin{bmatrix} R_\lambda \\ 0 \\ 0 \end{bmatrix},$$

and hence (10.36) holds with

$$Q_\lambda = \begin{bmatrix} Q & \\ & I \end{bmatrix} \bar{Q}_\lambda.$$

The advantage of this combined approach is that when the value of $\lambda$ is changed in the rootfinding algorithm, we need only recalculate $\bar{Q}_\lambda$ and not the Householder part of the factorization (10.38). This feature can save a lot of computation in the case of $m \gg n$, since just $O(n^3)$ operations are required to recalculate $\bar{Q}_\lambda$ and $R_\lambda$ for each value of $\lambda$, after the initial cost of $O(mn^2)$ operations needed to calculate $Q$ in (10.37).

Least-squares problems are often poorly scaled. Some of the variables could have values of about $10^4$, while other variables could be of order $10^{-6}$. If such wide variations are ignored, the algorithms above may encounter numerical difficulties or produce solutions of poor quality. One way to reduce the effects of poor scaling is to use an *ellipsoidal* trust region in place of the spherical trust region defined above. The step is confined to an ellipse in which the lengths of the principal axes are related to the typical values of the corresponding variables. Analytically, the trust-region subproblem becomes

$$\min_p \tfrac{1}{2}\|J_k p + r_k\|^2, \quad \text{subject to } \|D_k p\| \le \Delta_k, \tag{10.39}$$

where $D_k$ is a diagonal matrix with positive diagonal entries (cf. (7.13)). Instead of (10.33), the solution of (10.39) satisfies an equation of the form

$$\left(J_k^T J_k + \lambda D_k^2\right) p_k^{\text{LM}} = -J_k^T r_k, \tag{10.40}$$

and, equivalently, solves the linear least-squares problem

$$\min_p \left\| \begin{bmatrix} J_k \\ \sqrt{\lambda} D_k \end{bmatrix} p + \begin{bmatrix} r_k \\ 0 \end{bmatrix} \right\|^2. \tag{10.41}$$

The diagonals of the scaling matrix $D_k$ can change from iteration to iteration, as we gather information about the typical range of values for each component of $x$. If the variation in these elements is kept within certain bounds, then the convergence theory for the spherical case continues to hold, with minor modifications. Moreover, the technique described above for calculating $R_\lambda$ needs no modification. Seber and Wild [280] suggest choosing the diagonals of $D_k^2$ to match those of $J_k^T J_k$, to make the algorithm invariant under diagonal scaling of the components of $x$. This approach is analogous to the technique of scaling by diagonal elements of the Hessian, which was described in Section 4.5 in the context of trust-region algorithms for unconstrained optimization.

For problems in which $m$ and $n$ are large and $J(x)$ is sparse, we may prefer to solve (10.31) or (10.39) approximately using the CG-Steihaug algorithm, Algorithm 7.2 from Chapter 7, with $J_k^T J_k$ replacing the exact Hessian $\nabla^2 f_k$. Positive semidefiniteness of the matrix $J_k^T J_k$ makes for some simplification of this algorithm, because negative curvature cannot arise. It is not necessary to calculate $J_k^T J_k$ explicitly to implement Algorithm 7.2; the matrix-vector products required by the algorithm can be found by forming matrix-vector products with $J_k$ and $J_k^T$ separately.

### CONVERGENCE OF THE LEVENBERG–MARQUARDT METHOD

It is not necessary to solve the trust-region problem (10.31) exactly in order for the Levenberg–Marquardt method to enjoy global convergence properties. The following convergence result can be obtained as a direct consequence of Theorem 4.6.

**Theorem 10.3.**

*Let $\eta \in \left(0, \frac{1}{4}\right)$ in Algorithm 4.1 of Chapter 4, and suppose that the level set $\mathcal{L}$ defined in (10.29) is bounded and that the residual functions $r_j(\cdot)$, $j = 1, 2, \ldots, m$ are Lipschitz continuously differentiable in a neighborhood $\mathcal{N}$ of $\mathcal{L}$. Assume that for each $k$, the approximate solution $p_k$ of (10.31) satisfies the inequality*

$$m_k(0) - m_k(p_k) \geq c_1 \|J_k^T r_k\| \min\left(\Delta_k, \frac{\|J_k^T r_k\|}{\|J_k^T J_k\|}\right), \tag{10.42}$$

*for some constant $c_1 > 0$, and in addition $\|p_k\| \leq \gamma \Delta_k$ for some constant $\gamma \geq 1$. We then have that*

$$\lim_{k\to\infty} \nabla f_k = \lim_{k\to\infty} J_k^T r_k = 0.$$

PROOF. The smoothness assumption on $r_j(\cdot)$ implies that we can choose a constant $M > 0$ such that $\|J_k^T J_k\| \leq M$ for all iterates $k$. Note too that the objective $f$ is bounded below (by zero). Hence, the assumptions of Theorem 4.6 are satisfied, and the result follows immediately. □

As in Chapter 4, there is no need to calculate the right-hand-side in the inequality (10.42) or to check it explicitly. Instead, we can simply require the decrease given by our approximate solution $p_k$ of (10.31) to at least match the decrease given by the Cauchy point, which can be calculated inexpensively in the same way as in Chapter 4. If we use the iterative CG-Steihaug approach, Algorithm 7.2, the condition (10.42) is satisfied automatically for $c_1 = 1/2$, since the Cauchy point is the first estimate of $p_k$ computed by this approach, while subsequent estimates give smaller values for the model function.

The local convergence behavior of Levenberg–Marquardt is similar to the Gauss–Newton method. Near a solution $x^*$ at which the first term of the Hessian $\nabla^2 f(x^*)$ (10.5) dominates the second term, the model function in (10.31), the trust region becomes inactive and the algorithm takes Gauss–Newton steps, giving the rapid local convergence expression (10.30).

### METHODS FOR LARGE-RESIDUAL PROBLEMS

In large-residual problems, the quadratic model in (10.31) is an inadequate representation of the function $f$ because the second-order part of the Hessian $\nabla^2 f(x)$ is too significant to be ignored. In data-fitting problems, the presence of large residuals may indicate that the model is inadequate or that errors have been made in monitoring the observations. Still, the practitioner may need to solve the least-squares problem with the current model and data, to indicate where improvements are needed in the weighting of observations, modeling, or data collection.

On large-residual problems, the asymptotic convergence rate of Gauss–Newton and Levenberg–Marquardt algorithms is only linear—slower than the superlinear convergence rate attained by algorithms for general unconstrained problems, such as Newton or quasi-Newton. If the individual Hessians $\nabla^2 r_j$ are easy to calculate, it may be better to ignore the structure of the least-squares objective and apply Newton's method with trust region or line search to the problem of minimizing $f$. Quasi-Newton methods, which attain a superlinear convergence rate without requiring calculation of $\nabla^2 r_j$, are another option. However, the behavior of both Newton and quasi-Newton on early iterations (before reaching a neighborhood of the solution) may be inferior to Gauss–Newton and Levenberg–Marquardt.

Of course, we often do not know beforehand whether a problem will turn out to have small or large residuals at the solution. It seems reasonable, therefore, to consider *hybrid algorithms*, which would behave like Gauss–Newton or Levenberg–Marquardt if the residuals turn out to be small (and hence take advantage of the cost savings associated with these methods) but switch to Newton or quasi-Newton steps if the residuals at the solution appear to be large.

There are a couple of ways to construct hybrid algorithms. One approach, due to Fletcher and Xu (see Fletcher [101]), maintains a sequence of positive definite Hessian approximations $B_k$. If the Gauss–Newton step from $x_k$ reduces the function $f$ by a certain fixed amount (say, a factor of 5), then this step is taken and $B_k$ is overwritten by $J_k^T J_k$. Otherwise, a direction is computed using $B_k$, and the new point $x_{k+1}$ is obtained by performing a line search. In either case, a BFGS-like update is applied to $B_k$ to obtain a new approximation $B_{k+1}$. In the zero-residual case, the method eventually always takes Gauss–Newton steps (giving quadratic convergence), while it eventually reduces to BFGS in the nonzero-residual case (giving superlinear convergence). Numerical results in Fletcher [101, Tables 6.1.2, 6.1.3] show good results for this approach on small-, large-, and zero-residual problems.

A second way to combine Gauss–Newton and quasi-Newton ideas is to maintain approximations to just the second-order part of the Hessian. That is, we maintain a sequence of matrices $S_k$ that approximate the summation term $\sum_{j=1}^m r_j(x_k)\nabla^2 r_j(x_k)$ in (10.5), and then use the overall Hessian approximation

$$B_k = J_k^T J_k + S_k$$

in a trust-region or line search model for calculating the step $p_k$. Updates to $S_k$ are devised so that the approximate Hessian $B_k$, or its constituent parts, mimics the behavior of the corresponding exact quantities over the step just taken. The update formula is based on a *secant equation*, which arises also in the context of unconstrained minimization (6.6) and nonlinear equations (11.27). In the present instance, there are a number of different ways to define the secant equation and to specify the other conditions needed for a complete update formula for $S_k$. We describe the algorithm of Dennis, Gay, and Welsch [90], which is probably the best-known algorithm in this class because of its implementation in the well-known NL2SOL package.

In [90], the secant equation is motivated in the following way. Ideally, $S_{k+1}$ should be a close approximation to the exact second-order term at $x = x_{k+1}$; that is,

$$S_{k+1} \approx \sum_{j=1}^{m} r_j(x_{k+1})\nabla^2 r_j(x_{k+1}).$$

Since we do not want to calculate the individual Hessians $\nabla^2 r_j$ in this formula, we could replace each of them with an approximation $(B_j)_{k+1}$ and impose the condition that $(B_j)_{k+1}$

should mimic the behavior of its exact counterpart $\nabla^2 r_j$ over the step just taken; that is,

$$\begin{aligned}(B_j)_{k+1}(x_{k+1} - x_k) &= \nabla r_j(x_{k+1}) - \nabla r_j(x_k) \\ &= (\text{row } j \text{ of } J(x_{k+1}))^T - (\text{row } j \text{ of } J(x_k))^T.\end{aligned}$$

This condition leads to a secant equation on $S_{k+1}$, namely,

$$\begin{aligned}S_{k+1}(x_{k+1} - x_k) &= \sum_{j=1}^{m} r_j(x_{k+1})(B_j)_{k+1}(x_{k+1} - x_k) \\ &= \sum_{j=1}^{m} r_j(x_{k+1}) \left[(\text{row } j \text{ of } J(x_{k+1}))^T - (\text{row } j \text{ of } J(x_k))^T\right] \\ &= J_{k+1}^T r_{k+1} - J_k^T r_{k+1}.\end{aligned}$$

As usual, this condition does not completely specify the new approximation $S_{k+1}$. Dennis, Gay, and Welsch add requirements that $S_{k+1}$ be symmetric and that the difference $S_{k+1} - S_k$ from the previous estimate $S_k$ be minimized in a certain sense, and derive the following update formula:

$$S_{k+1} = S_k + \frac{(y^\sharp - S_k s)y^T + y(y^\sharp - S_k s)^T}{y^T s} - \frac{(y^\sharp - S_k s)^T s}{(y^T s)^2} yy^T, \tag{10.43}$$

where

$$\begin{aligned}s &= x_{k+1} - x_k, \\ y &= J_{k+1}^T r_{k+1} - J_k^T r_k, \\ y^\sharp &= J_{k+1}^T r_{k+1} - J_k^T r_{k+1}.\end{aligned}$$

Note that (10.43) is a slight variant on the DFP update for unconstrained minimization. It would be identical if $y^\sharp$ and $y$ were the same.

Dennis, Gay, and Welsch use their approximate Hessian $J_k^T J_k + S_k$ in conjunction with a trust-region strategy, but a few more features are needed to enhance its performance. One deficiency of its basic update strategy for $S_k$ is that this matrix is not guaranteed to vanish as the iterates approach a zero-residual solution, so it can interfere with superlinear convergence. This problem is avoided by scaling $S_k$ prior to its update; we replace $S_k$ by $\tau_k S_k$ on the right-hand-side of (10.43), where

$$\tau_k = \min\left(1, \frac{|s^T y^\sharp|}{|s^T S_k s|}\right).$$

A final modification in the overall algorithm is that the $S_k$ term is omitted from the Hessian approximation when the resulting Gauss–Newton model produces a sufficiently good step.

## 10.4 ORTHOGONAL DISTANCE REGRESSION

In Example 10.1 we assumed that no errors were made in noting the time at which the blood samples were drawn, so that the differences between the model $\phi(x; t_j)$ and the observation $y_j$ were due to inadequacy in the model or measurement errors in $y_j$. We assumed that any errors in the ordinates—the times $t_j$—are tiny by comparison with the errors in the observations. This assumption often is reasonable, but there are cases where the answer can be seriously distorted if we fail to take possible errors in the ordinates into account. Models that take these errors into account are known in the statistics literature as *errors-in-variables models* [280, Chapter 10], and the resulting optimization problems are referred to as *total least squares* in the case of a linear model (see Golub and Van Loan [136, Chapter 5]) or as *orthogonal distance regression* in the nonlinear case (see Boggs, Byrd, and Schnabel [30]).

We formulate this problem mathematically by introducing perturbations $\delta_j$ for the ordinates $t_j$, as well as perturbations $\epsilon_j$ for $y_j$, and seeking the values of these $2m$ perturbations that minimize the discrepancy between the model and the observations, as measured by a weighted least-squares objective function. To be precise, we relate the quantities $t_j$, $y_j$, $\delta_j$, and $\epsilon_j$ by

$$y_j = \phi(x; t_j + \delta_j) + \epsilon_j, \qquad j = 1, 2, \ldots, m, \tag{10.44}$$

and define the minimization problem as

$$\min_{x,\delta_j,\epsilon_j} \ \tfrac{1}{2} \sum_{j=1}^{m} w_j^2 \epsilon_j^2 + d_j^2 \delta_j^2, \qquad \text{subject to (10.44)}. \tag{10.45}$$

The quantities $w_i$ and $d_i$ are weights, selected either by the modeler or by some automatic estimate of the relative significance of the error terms.

It is easy to see how the term "orthogonal distance regression" originates when we graph this problem; see Figure 10.2. If all the weights $w_i$ and $d_i$ are equal, then each term in the summation (10.45) is simply the shortest distance between the point $(t_j, y_j)$ and the curve $\phi(x; t)$ (plotted as a function of $t$). The shortest path between each point and the curve is orthogonal to the curve at the point of intersection.

Using the constraints (10.44) to eliminate the variables $\epsilon_j$ from (10.45), we obtain the unconstrained least-squares problem

$$\min_{x,\delta} \ F(x, \delta) = \tfrac{1}{2} \sum_{j=1}^{m} w_j^2 [y_j - \phi(x; t_j + \delta_j)]^2 + d_j^2 \delta_j^2 = \tfrac{1}{2} \sum_{j=1}^{2m} r_j^2(x, \delta), \tag{10.46}$$

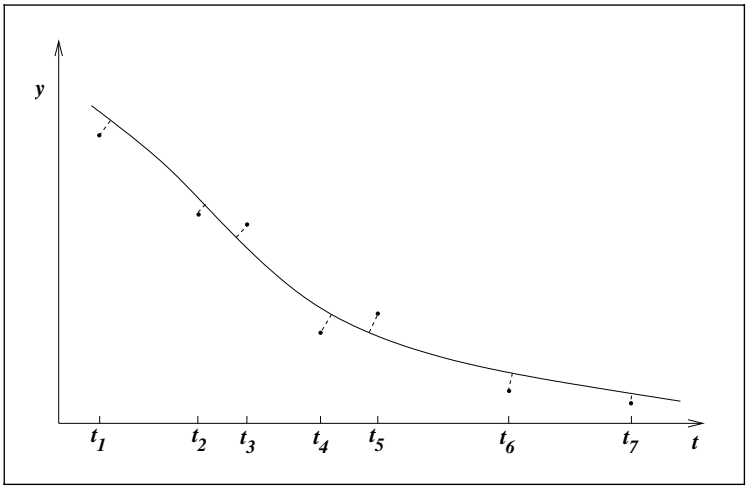


**Figure 10.2** Orthogonal distance regression minimizes the sum of squares of the distance from each point to the curve.

where $\delta = (\delta_1, \delta_2, \ldots, \delta_m)^T$ and we have defined

$$r_j(x, \delta) = \begin{cases} w_j[\phi(x; t_j + \delta_j) - y_j], & j = 1, 2, \ldots, m, \\ d_{j-m}\delta_{j-m}, & j = m+1, \ldots, 2m. \end{cases} \tag{10.47}$$

Note that (10.46) is now a standard least-squares problem with $2m$ residuals and $m + n$ unknowns, which we can solve by using the techniques in this chapter. A naive implementation of this strategy may, however, be quite expensive, since the number of parameters ($2n$) and the number of observations ($m + n$) may both be much larger than for the original problem.

Fortunately, the Jacobian matrix for (10.46) has a special structure that can be exploited in implementing the Gauss–Newton or Levenberg–Marquardt methods. Many of its components are zero; for instance, we have

$$\frac{\partial r_j}{\partial \delta_i} = \frac{\partial[\phi(t_j + \delta_j; x) - y_j]}{\partial \delta_i} = 0, \quad i, j = 1, 2, \ldots, m, \ i \neq j,$$

and

$$\frac{\partial r_j}{\partial x_i} = 0, \quad j = m+1, \ldots, 2m, \ \ i = 1, 2, \ldots, n.$$

Additionally, we have for $j = 1, 2, \ldots, m$ and $i = 1, 2, \ldots, m$ that

$$\frac{\partial r_{m+j}}{\partial \delta_i} = \begin{cases} d_j & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Hence, we can partition the Jacobian of the residual function $r$ defined by (10.47) into blocks and write

$$J(x, \delta) = \begin{bmatrix} \hat{J} & V \\ 0 & D \end{bmatrix}, \tag{10.48}$$

where $V$ and $D$ are $m \times m$ diagonal matrices and $\hat{J}$ is the $m \times n$ matrix of partial derivatives of the functions $w_j \phi(t_j + \delta_j; x)$ with respect to $x$. Boggs, Byrd, and Schnabel [30]) apply the Levenberg–Marquardt algorithm to (10.46) and note that block elimination can be used to solve the subproblems (10.33), (10.35) efficiently. Given the partitioning (10.48), we can partition the step vector $p$ and the residual vector $r$ accordingly as

$$p = \begin{bmatrix} p_x \\ p_\delta \end{bmatrix}, \qquad r = \begin{bmatrix} \hat{r}_1 \\ \hat{r}_2 \end{bmatrix},$$

and write the normal equations (10.33) in the partitioned form

$$\begin{bmatrix} \hat{J}^T \hat{J} + \lambda I & \hat{J}^T V \\ V \hat{J} & V^2 + D^2 + \lambda I \end{bmatrix} \begin{bmatrix} p_x \\ p_\delta \end{bmatrix} = - \begin{bmatrix} \hat{J}^T \hat{r}_1 \\ V \hat{r}_1 + D \hat{r}_2 \end{bmatrix}. \tag{10.49}$$

Since the lower right submatrix $V^2 + D^2 + \lambda I$ is diagonal, it is easy to eliminate $p_\delta$ from this system and obtain a smaller $n \times n$ system to be solved for $p_x$ alone. The total cost of finding a step is only marginally greater than for the $m \times n$ problem arising from the standard least-squares model.

### NOTES AND REFERENCES

Algorithms for linear least squares are discussed comprehensively by Björck [29], who includes detailed error analyses of the different algorithms and software listings. He considers not just the basic problem (10.13) but also the situation in which there are bounds (for example, $x \geq 0$) or linear constraints (for example, $Ax \geq b$) on the variables. Golub and Van Loan [136, Chapter 5] survey the state of the art, including discussion of the suitability of the different approaches (for example, normal equations vs. QR factorization) for different problem types. A classical reference on linear least-squares is Lawson and Hanson [188].

Very large nonlinear least-squares problems arise in numerous areas of application, such as medical imaging, geophysics, economics, and engineering design. In many instances, both the number of variables $n$ and the number of residuals $m$ is large, but it is also quite common that only $m$ is large.

The original description of the Levenberg–Marquardt algorithm [190, 203] did not make the connection with the trust-region concept. Rather, it adjusted the value of $\lambda$ in (10.33) directly, increasing or decreasing it by a certain factor according to whether or not the previous trial step was effective in decreasing $f(\cdot)$. (The heuristics for adjusting $\lambda$ were analogous to those used for adjusting the trust-region radius $\Delta_k$ in Algorithm 4.1.) Similar convergence results to Theorem 10.3 can be proved for algorithms that use this approach (see, for instance, Osborne [231]), independently of trust-region analysis. The connection with trust regions was firmly established by Moré [210].

Wright and Holt [318] present an inexact Levenberg–Marquardt approach for large-scale nonlinear least squares that manipulates the parameter $\lambda$ directly rather than making use of the connection to trust-region algorithms. This method takes steps $\bar{p}_k$ that, analogously to (7.2) and (7.3) in Chapter 7, satisfy the system

$$\left\| \left( J_k^T J_k + \lambda_k I \right) \bar{p}_k + J_k^T r_k \right\| \leq \eta_k \| J_k^T r_k \|, \qquad \text{for some } \eta_k \in [0, \eta],$$

where $\eta \in (0, 1)$ is a constant and $\{\eta_k\}$ is a forcing sequence. A ratio of actual to predicted decrease is used to decide whether the step $\bar{p}_k$ should be taken, and convergence to stationary points can be proved under certain assumptions. The method can be implemented efficiently by using Algorithm LSQR of Paige and Saunders [234] to calculate the approximate solution of (10.35) since, for a small marginal cost, this algorithm can compute approximate solutions for a number of different values of $\lambda_k$ simultaneously. Hence, we can compute values of $\bar{p}_k$ corresponding to a range of values of $\lambda_k$, and choose the actual step to be the one corresponding to the smallest $\lambda_k$ for which the actual-predicted decrease ratio is satisfactory.

Nonlinear least squares software is fairly prevalent because of the high demand for it. Major numerical software libraries such as IMSL, HSL, NAG, and SAS, as well as programming environments such as Mathematica and Matlab, contain robust nonlinear least-squares implementations. Other high quality implmentations include DFNLP, MINPACK, NL2SOL, and NLSSOL; see Moré and Wright [217, Chapter 3]. The nonlinear programming packages LANCELOT, KNITRO, and SNOPT provide large-scale implementions of the Gauss–Newton and Levenberg–Marquardt methods. The orthogonal distance regression algorithm is implemented by ORDPACK [31].

All these routines (which can be accessed through the web) give the user the option of either supplying Jacobians explicitly or else allowing the code to compute them by finite differencing. (In the latter case, the user need only write code to compute the residual vector $r(x)$; see Chapter 8.) Seber and Wild [280, Chapter 15] describe some of the important practical issues in selecting software for statistical applications.

## EXERCISES

**10.1** Let $J$ be an $m \times n$ matrix with $m \geq n$, and let $y \in \mathbb{R}^m$ be a vector.

**(a)** Show that $J$ has full column rank if and only if $J^T J$ is nonsingular.

**(b)** Show that $J$ has full column rank if and only if $J^T J$ is positive definite.

**10.2** Show that the function $f(x)$ in (10.13) is convex.

**10.3** Show that

**(a)** if $Q$ is an orthogonal matrix, then $\|Qx\| = \|x\|$ for any vector $x$;

**(b)** the matrices $\bar{R}$ in (10.15) and $R$ in (10.17) are identical if $\Pi = I$, provided that $J$ has full column rank $n$.

**10.4**

**(a)** Show that $x^*$ defined in (10.22) is a minimizer of (10.13).

**(b)** Find $\|x^*\|$ and conclude that this norm is minimized when $\tau_i = 0$ for all $i$ with $\sigma_i = 0$.

**10.5** Suppose that each residual function $r_j$ and its gradient are Lipschitz continuous with Lipschitz constant $L$, that is,

$$\|r_j(x) - r_j(\tilde{x})\| \leq L\|x - \tilde{x}\|, \qquad \|\nabla r_j(x) - \nabla r_j(\tilde{x})\| \leq L\|x - \tilde{x}\|$$

for all $j = 1, 2, \ldots, m$ and all $x, \tilde{x} \in \mathcal{D}$, where $\mathcal{D}$ is a compact subset of $\mathbb{R}^n$. Assume also that the $r_j$ are bounded on $\mathcal{D}$, that is, there exists $M > 0$ such that $|r_j(x)| \leq M$ for all $j = 1, 2, \ldots, m$ and all $x \in \mathcal{D}$. Find Lipschitz constants for the Jacobian $J$ (10.3) and the gradient $\nabla f$ (10.4) over $\mathcal{D}$.

**10.6** Express the solution $p$ of (10.33) in terms of the singular-value decomposition of $J(x)$ and the scalar $\lambda$. Express its squared-norm $\|p\|^2$ in these same terms, and show that

$$\lim_{\lambda \to 0} p = \sum_{\sigma_i \neq 0} \frac{u_i^T r}{\sigma_i} v_i .$$