(1) compute the QR factorization of $\boldsymbol{Z}$;
(2) compute $\boldsymbol{Q}_Z^T\mathbf{y}'$;
(3) solve $\boldsymbol{R}\mathbf{x} = \boldsymbol{Q}_Z^T\mathbf{y}'$ for $\mathbf{x}$.
(4) compute the residual sum of squares by $\mathbf{y}'^T\mathbf{y}' - \mathbf{y}'\boldsymbol{Q}_Z\boldsymbol{Q}_Z^T\mathbf{y}'$.
(5) compute the variance–covariance matrix from $\boldsymbol{V}_\mathbf{x} = \boldsymbol{R}^{-1}(\boldsymbol{R}^{-1})^T$.

8.1.3.2. *The normal equations*

Let us now consider the relationship of the QR procedure for solving the linear least-squares problem to the classical method based on the normal equations. The normal equations can be derived by differentiating (8.1.3.2) and equating the result to a null vector. This yields

$$\boldsymbol{Z}^T\boldsymbol{Z}\mathbf{x} = \boldsymbol{Z}^T\mathbf{y}'. \tag{8.1.3.5}$$

The algorithm is therefore to compute the cross-product matrix, $\boldsymbol{B} = \boldsymbol{Z}^T\boldsymbol{Z}$, and the right-hand side, $\mathbf{d} = \boldsymbol{Z}^T\mathbf{y}'$, and to solve the resulting system of equations, $\boldsymbol{B}\mathbf{x} = \mathbf{d}$. This is usually accomplished by computing the Cholesky decomposition of $\boldsymbol{B}$, that is $\boldsymbol{B} = \boldsymbol{C}^T\boldsymbol{C}$, where $\boldsymbol{C}$ is upper triangular, and then solving the two triangular systems $\boldsymbol{C}^T\mathbf{v} = \mathbf{d}$ and $\boldsymbol{C}\mathbf{x} = \mathbf{v}$. Because $\boldsymbol{Z} = \boldsymbol{Q}_Z\boldsymbol{R}$, equation (8.1.3.5) becomes

$$\boldsymbol{R}^T\boldsymbol{Q}_Z^T\boldsymbol{Q}_Z\boldsymbol{R}\mathbf{x} = \boldsymbol{R}^T\boldsymbol{Q}_Z^T\mathbf{y}', \tag{8.1.3.6}$$

or

$$\boldsymbol{R}^T\boldsymbol{R}\mathbf{x} = \boldsymbol{R}^T\boldsymbol{Q}_Z^T\mathbf{y}'. \tag{8.1.3.7}$$

It is clear that $\boldsymbol{R}$ is the Cholesky factor of $\boldsymbol{Z}^T\boldsymbol{Z}$, although it is formed in a different way. This procedure requires of order $(np^2)/2$ operations to form the product $\boldsymbol{Z}^T\boldsymbol{Z}$ and $p^3/3$ operations for the Cholesky decomposition. In some situations, the extra time to compute the QR factorization is justified because of greater stability, as will be discussed below. Most other quantities of statistical interest can be computed directly from the QR factorization.

8.1.3.3. *Conditioning*

The condition number of $\boldsymbol{Z}$, which is defined (Subsection 8.1.1.1) as the square root of the ratio of the largest to the smallest eigenvalue of $\boldsymbol{Z}^T\boldsymbol{Z}$, is an indicator of the effect a small change in an element of $\boldsymbol{Z}$ will have on the elements of $(\boldsymbol{Z}^T\boldsymbol{Z})^{-1}$and of $\hat{\mathbf{x}}$. A large value of the condition number means that small errors in computing an element of $\boldsymbol{Z}$, owing possibly to truncation or roundoff in the computer, can introduce large errors into the elements of the inverse matrix. Also, when the condition number is large, the standard uncertainties of some estimated parameters will be large. A large condition number, as defined in this way, can result from either scaling or correlation or some combination of these. To illustrate this, consider the matrices

$$\boldsymbol{Z}^T\boldsymbol{Z} = \begin{pmatrix} 2+\varepsilon & 0 \\ 0 & \varepsilon \end{pmatrix}$$

and

$$\boldsymbol{Z}^T\boldsymbol{Z} = \begin{pmatrix} 1 & 1-\varepsilon \\ 1-\varepsilon & 1 \end{pmatrix},$$

where ε represents machine precision, which can be defined as the smallest number in machine representation that, when added to 1, gives a result different from 1. By the conventional definition, both of these matrices have a condition number for $\boldsymbol{Z}$ of $[(2+\varepsilon)/\varepsilon]^{1/2}$. Because numbers of order ε can be perfectly well represented, however, the first one can be inverted without loss of precision, whereas an inverse for the second would be totally meaningless. It is good practice, therefore, to factor the design matrix, $\boldsymbol{Z}$, into the form

$$\boldsymbol{Z} = \boldsymbol{T}\boldsymbol{S}, \tag{8.1.3.8}$$

where $\boldsymbol{S}$ is a $p \times p$ diagonal matrix whose elements define some kind of 'natural' unit appropriate to the parameter represented in each column of $\boldsymbol{Z}$. The ideal natural unit would be the standard uncertainty of that parameter, but this is not available until after the calculation has been completed. If correlation is not too severe, suitable values for the elements of $\boldsymbol{S}$, of the same order of magnitude as those derived from the standard uncertainty, are the column Euclidean norms, that is

$$S = \|\mathbf{z}_j\| = (\mathbf{z}_j^T\mathbf{z}_j)^{1/2}, \tag{8.1.3.9}$$

where $\mathbf{z}_j$ denotes the jth column of $\boldsymbol{Z}$. This scaling causes all diagonal elements of $\boldsymbol{Z}^T\boldsymbol{Z}$ to be equal to one, and errors in the elements of $\boldsymbol{Z}$ will have roughly equal effects.

Ill conditioning that results from correlation, as in the second example above, is more difficult to deal with. It is an indication that some linear combination of parameters, some eigenvector of the normal equations matrix, is poorly determined by the available data. Use of the QR factorization of $\boldsymbol{Z}$ to compute the Cholesky factor of $\boldsymbol{Z}^T\boldsymbol{Z}$ may be advantageous, in spite of the additional computation time, because better numerical stability is obtained in marginal situations. As a practical matter, however, it is important to recognize that an ill conditioned matrix is a symptom of a flaw in the model or in the experimental design (or both). Use can be made of the fact that, although determining the entire set of eigenvalues and eigenvectors of a large matrix is computationally an inherently difficult problem, a relatively simple algorithm, known as a *condition estimator* (Anderson *et al.*, 1992), can produce a good approximation to the eigenvector that corresponds to the smallest eigenvalue of a nearly singular matrix. This information can be used in either or both of two ways. First, without any fundamental modification to the model or the experiment, a simple, linear transformation of the parameters so that the problem eigenvector is one of the independent parameters, followed by rescaling, can resolve the numerical difficulties in computing the estimates. A common example is the situation where a phase transition results in the doubling of a unit cell, with pairs of atoms almost but not quite related by a lattice translation. A transformation that makes the estimated parameters the sums and differences of corresponding parameters in related pairs of atoms can make a dramatic improvement in the condition number. Alternatively, the problem eigenvector can be set to some value determined from theory or from some other experiment (see Section 8.3.1), or additional data can be collected that are selected to make that combination of parameters determinate.

8.1.4. Methods for nonlinear least squares

Recall (equation 8.1.2.1) that the general, nonlinear problem can be stated in the form: find the minimum of

$$S(\mathbf{x}) = \sum_{i=1}^{n} w_i[y_i - M_i(\mathbf{x})]^2, \tag{8.1.4.1}$$

where $\mathbf{x}$ is a vector of p parameters, and $\boldsymbol{M}(\mathbf{x})$ represents a set of model functions that predict the values of observations, $\mathbf{y}$. In this section, we discuss two useful ways of solving this problem and consider the relative merits of each. The first is based on iteratively linearizing the functions $M_i(\mathbf{x})$ and approximating

(8.1.4.1) by one of the form in (8.1.2.5). The second uses an unconstrained minimization algorithm, based on Newton's method, to minimize $S(\mathbf{x})$.

8.1.4.1. *The Gauss–Newton algorithm*

Let $\mathbf{x}_c$ be the current approximation to $\hat{\mathbf{x}}$, the solution to (8.1.4.1). We construct a linear approximation to $M_i(\mathbf{x})$ in the vicinity of $\mathbf{x} = \mathbf{x}_c$ by expanding it in a Taylor series through the linear terms, obtaining

$$M_i(\mathbf{x}) = M_i(\mathbf{x}_c) + \sum_{j=1}^{p} J_{ij}(\mathbf{x} - \mathbf{x}_c)_j, \tag{8.1.4.2}$$

where $\boldsymbol{J}$ is the Jacobian matrix, defined by $J_{ij} = \partial M_i(\mathbf{x})/\partial x_j$. A straightforward procedure, known as the *Gauss–Newton algorithm*, may be formally stated as follows:

(1) compute $\mathbf{d}$ as the solution of the linear system

$$\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J}\mathbf{d} = \boldsymbol{J}^T\boldsymbol{W}[\mathbf{y} - \boldsymbol{M}(\mathbf{x}_c)];$$

(2) set $\mathbf{x}_c = \mathbf{x}_c + \mathbf{d}$;

(3) if not converged (see Subsection 8.1.4.4), go to (1), else stop.

The convergence rate of the Gauss–Newton algorithm depends on the size of the residual, that is, on $S(\hat{\mathbf{x}})$. If $S(\hat{\mathbf{x}}) = 0$, then the convergence rate is quadratic; if it is small, then the rate is linear; but if $S(\hat{\mathbf{x}})$ is large, then the procedure is not locally convergent at all. Fortunately, this procedure can be altered so that it is always locally convergent, and even globally convergent, that is, convergent to a relative minimum from any starting point. There are two possibilities. First, the procedure can be modified to include a line search. This is accomplished by computing the direction $\mathbf{d}$ as above and then choosing α such that $S(\mathbf{x}_c + \alpha\mathbf{d})$ is sufficiently smaller than $S(\mathbf{x}_c)$. In order to guarantee convergence, one uses the test

$$S(\mathbf{x}_c + \alpha\mathbf{d}) < S(\mathbf{x}_c) - \alpha\gamma\mathbf{d}^T\boldsymbol{J}(\mathbf{x}_c)^T[\mathbf{y} - \boldsymbol{M}(\mathbf{x}_c)], \tag{8.1.4.3}$$

where, as actually implemented in modern codes, γ has values of the order of 10^{-4}. [In theory, a slightly stronger condition is necessary, but this is usually avoided by other means. See Dennis & Schnabel (1983) for details.] While this improves the situation dramatically, it still suffers from the deficiency that it is very slow on some problems, and it is undefined if the rank of the Jacobian, $\boldsymbol{J}(\hat{\mathbf{x}})$, is less than p. $\boldsymbol{J}(\mathbf{x})$ usually has rank p near the solution, but it may be rank deficient far away. Also, it may be 'close' to rank deficient, and give numerical difficulties (see Subsection 8.1.3.3).

8.1.4.2. *Trust-region methods – the Levenberg–Marquardt algorithm*

These remaining problems can be addressed by the utilization of a trust-region modification to the basic Gauss–Newton algorithm. For this procedure, we compute the step, $\mathbf{d}$, as the solution to the linear least-squares problem subject to the constraint that $\|\mathbf{d}\| \le \tau_c$, where τ_c is the *trust-region radius*. Here, the linearized model is modified by admitting that the linearization is only valid within a limited region around the current approximation. It is clear that, if the trust region is sufficiently large, this constrained step will in fact be unconstrained, and the step will be the same as the Gauss–Newton step. If the constraint is active, however, the step has the form

$$[\boldsymbol{J}(\mathbf{x}_c)^T\boldsymbol{W}\boldsymbol{J}(\mathbf{x}_c) + \mu\boldsymbol{I}]\mathbf{d}(\mu) = \boldsymbol{J}(\mathbf{x}_c)\boldsymbol{W}[\mathbf{y} - \boldsymbol{M}(\mathbf{x}_c)], \tag{8.1.4.4}$$

where μ is the Lagrange multiplier corresponding to the constraint (see Section 8.3.1), that is, μ is the value such that $\|\mathbf{d}(\mu)\| = \tau_c$. Formula (8.1.4.4) is known as the *Levenberg–Marquardt* equation. It can be seen from this formula that the step direction is intermediate between the Gauss–Newton direction and the direction of steepest descent, for which reason it is frequently known as ''Marquardt's compromise'' (Draper & Smith, 1981).

Efficient numerical calculation of $\mathbf{d}$ for a given value of μ is accomplished by noting that (8.1.4.4) is equivalent to the linear least-squares problem, find the minimum of

$$S = \left\| \begin{pmatrix} \boldsymbol{J}(\mathbf{x}_c) \\ \sqrt{\mu}\boldsymbol{I} \end{pmatrix}\mathbf{d} - \begin{pmatrix} [\mathbf{y} - \boldsymbol{M}(\mathbf{x}_c)] \\ \boldsymbol{O} \end{pmatrix} \right\|^2. \tag{8.1.4.5}$$

This problem can be solved by saving the QR factorization for $\mu = 0$ and updating it for various values of μ greater than 0. The actual computation of μ is accomplished by a modified Newton method applied to the constraint equation (see Dennis & Schnabel, 1983, for details). Having calculated the constrained value of $\mathbf{d}$, we set $\mathbf{x}_+ = \mathbf{x}_c + \mathbf{d}$. The algorithm is completed by specifying a procedure for updating the trust-region parameter, τ_c, after each step. This is done by comparing the actual value of $S(\mathbf{x}_+)$ with the predicted value based on the linearization. If there is good agreement between these values, τ is increased. If there is not good agreement, τ is left unchanged, and if $S(\mathbf{x}_+) > S(\mathbf{x}_c)$, the step is rejected, and τ is reduced. Global convergence can be shown under reasonable conditions, and very good computational performance has been observed in practice.

8.1.4.3. *Quasi-Newton, or secant, methods*

The second class of methods for solving the general, nonlinear least-squares problem is based on the unconstrained minimization of the function $S(\mathbf{x})$, as defined in (8.1.4.1). Newton's method for solving this problem is derived by constructing a quadratic approximation to S at the current trial point, $\mathbf{x}_c$, giving

$$S_c(\mathbf{d}) = S(\mathbf{x}_c) + \mathbf{g}(\mathbf{x}_c)^T\mathbf{d} + (1/2)\mathbf{d}^T\boldsymbol{H}(\mathbf{x}_c)\mathbf{d}, \tag{8.1.4.6}$$

from which the Newton step is obtained by minimizing S_c with respect to $\mathbf{d}$. Here, $\mathbf{g}$ represents the gradient of S and $\boldsymbol{H}$ is the Hessian matrix, the $p \times p$ symmetric matrix of second partial derivatives, $H_{jk} = \partial^2 S/\partial x_j \partial x_k$. Thus, $\mathbf{d}$ is calculated by solving the linear system

$$\boldsymbol{H}(\mathbf{x}_c)\mathbf{d} = -\mathbf{g}(\mathbf{x}_c). \tag{8.1.4.7}$$

[Note: In the literature on optimization, the notation $\nabla^2 S(\mathbf{x})$ is often used to denote the Hessian matrix of the function $S(\mathbf{x})$. This should not be confused with the Laplacian operator.] While Newton's method is locally quadratically convergent, it suffers from well known drawbacks. First, it is not globally convergent without employing some form of line search or trust region to safeguard it. Second, it requires the computation of the Hessian of S in each iteration.

The Hessian, however, has the form

$$\boldsymbol{H}(\mathbf{x}_c) = \boldsymbol{J}(\mathbf{x}_c)^T\boldsymbol{W}\boldsymbol{J}(\mathbf{x}_c) + \boldsymbol{B}(\mathbf{x}_c), \tag{8.1.4.8}$$

where $\boldsymbol{B}(\mathbf{x})$ is given by

$$B(\mathbf{x}_c)_{jk} = \sum_{i=1}^{n} w_i[y_i - M_i(\mathbf{x}_c)]\partial M_i^2(\mathbf{x}_c)/\partial x_j \partial x_k. \tag{8.1.4.9}$$

The first term of the Hessian, which is dependent only on the Jacobian, is readily available, but $\boldsymbol{B}$, even if it is available analytically, is, typically, expensive to compute. Furthermore,

in some situations, such as in the vicinity of a more symmetric model, $\boldsymbol{H}(\mathbf{x}_c)$ may not be positive definite.

In order to overcome these difficulties, various methods have been proposed to approximate $\boldsymbol{H}$ without calculating $\boldsymbol{B}$. Because these methods are based on approximations to Newton's method, they are often referred to as *quasi-Newton methods*. Two popular versions use approximations to $\boldsymbol{B}$ that are known as the Davidon–Fletcher–Powell (DFP) update and the Broyden–Fletcher–Goldfarb–Shanno (BFGS) update, with BFGS being apparently superior in practice. The basic idea of both procedures is to use values of the gradient to update an approximation to the Hessian in such a way as to preserve as much previous information as possible, while incorporating the new information obtained in the most recent step. From (8.1.4.6), the gradient of $S_c(\mathbf{d})$ is

$$\mathbf{g}(\mathbf{x}_c + \mathbf{d}) = \mathbf{g}(\mathbf{x}_c) + \boldsymbol{H}(\mathbf{x}_c)\mathbf{d}. \tag{8.1.4.10}$$

If the true function is not quadratic, or if $\boldsymbol{H}$ is only approximately known, the gradient at the point $\mathbf{x}_c + \mathbf{d}$, where $\mathbf{d}$ is the solution of (8.1.4.7), will not vanish. We make use of the value of $\mathbf{g}(\mathbf{x}_c + \mathbf{d})$ to compute a correction to $\boldsymbol{H}$ to give a Hessian that would have predicted what was actually found. That is, find an *update matrix*, $\boldsymbol{B}'$, such that

$$[\boldsymbol{H}(\mathbf{x}_c) + \boldsymbol{B}'(\mathbf{x}_c)]\mathbf{d} = \mathbf{g}(\mathbf{x}_c + \mathbf{d}) - \mathbf{g}(\mathbf{x}_c). \tag{8.1.4.11}$$

This is known as the *secant*, or *quasi-Newton*, relation. An algorithm based on the BFGS update goes as follows: Given $\mathbf{x}_c$ and $\boldsymbol{H}_c$ (usually a scaled, identity matrix is used as the initial approximation, in which case the first step is in the steepest descent direction),

(1) solve $\boldsymbol{H}_c\mathbf{d} = -\mathbf{g}(\mathbf{x}_c)$ for the direction $\mathbf{d}$;
(2) compute α such that $S(\mathbf{x}_c + \alpha\mathbf{d})$ satisfies condition (8.1.4.3);
(3) set $\mathbf{x}_c = \mathbf{x}_c + \alpha\mathbf{d}$;
(4) update $\boldsymbol{H}_c$ by $\boldsymbol{H}_c + \mathbf{g}(\mathbf{x}_c)\mathbf{g}(\mathbf{x}_c)^T/\mathbf{d}^T\mathbf{g}(\mathbf{x}_c) + \mathbf{q}\mathbf{q}^T/\alpha\mathbf{q}^T\mathbf{d}$ (see Nash & Sofer, 1995), where $\mathbf{q} = \mathbf{g}(\mathbf{x}_c + \alpha\mathbf{d}) - \mathbf{g}(\mathbf{x}_c)$;
(5) if not converged, go to (1), else stop.

If, in step (2), the additional condition is applied that $\mathbf{d}^T\mathbf{g}(\mathbf{x}_c + \alpha\mathbf{d}) = 0$, it becomes an *exact line search*. This, however, is an unnecessarily severe condition, because it can be shown that, if $\mathbf{d}^T[\mathbf{g}(\mathbf{x}_c + \alpha\mathbf{d}) - \mathbf{g}(\mathbf{x}_c)] > 0$, the approximation to the Hessian remains positive definite. The two terms in this expression are values of $\mathrm{d}S/\mathrm{d}\alpha$, and the condition states that S does not decrease more rapidly as α increases, which will always be true for some range of values of $\alpha > 0$. This algorithm is globally convergent; it *will* find a point at which the gradient vanishes, although that point may be a false minimum. It should be noted, however, that the procedure does not produce a final Hessian matrix whose inverse necessarily has any resemblance to a variance–covariance matrix. To get that it is necessary to compute $\boldsymbol{J}(\widehat{\mathbf{x}})$ and from it compute $(\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J})^{-1}$.

If a scaled, identity matrix is used as the initial approximation to $\boldsymbol{H}$, no use is made of the fact that, at least in the vicinity of the minimum, a major part of the Hessian matrix is given by $\boldsymbol{J}(\mathbf{x}_c)^T\boldsymbol{W}\boldsymbol{J}(\mathbf{x}_c)$. Thus, if there can be reasonable confidence in the general features of the model, it is useful to use $\boldsymbol{J}(\mathbf{x}_c)^T\boldsymbol{W}\boldsymbol{J}(\mathbf{x}_c)$ as the initial approximation to $\boldsymbol{H}$, in which case the first step is in the Gauss–Newton direction. The line-search provision, however, guarantees convergence, and, when n and p are both large, as in many crystallographic applications of least squares, a quasi-Newton update gives an adequate approximation to the new Hessian with a great deal less computation.

Another procedure that makes use of the fact that the linear approximation gives a major part of the Hessian constructs an approximation of the form

$$\boldsymbol{H}(\mathbf{x}) = \boldsymbol{J}(\mathbf{x})^T\boldsymbol{W}\boldsymbol{J}(\mathbf{x}) + \boldsymbol{B}(\mathbf{x}), \tag{8.1.4.12}$$

where $\boldsymbol{B}$ is generated by quasi-Newton methods. The quasi-Newton condition that must be satisfied in this case is

$$[\boldsymbol{J}(\mathbf{x}_+)^T\boldsymbol{W}\boldsymbol{J}(\mathbf{x}_+) + \boldsymbol{B}(\mathbf{x}_+)]\mathbf{d}_c = \mathbf{q}_c, \tag{8.1.4.13}$$

where $\mathbf{d}_c = \mathbf{x}_+ - \mathbf{x}_c$, and

$$\mathbf{q}_c = \boldsymbol{J}(\mathbf{x}_+)^T\boldsymbol{W}[\mathbf{y} - \boldsymbol{M}(\mathbf{x}_+)] - \boldsymbol{J}(\mathbf{x}_c)^T\boldsymbol{W}[\mathbf{y} - \boldsymbol{M}(\mathbf{x}_c)].$$

A formula based on the BFGS update (Nash & Sofer, 1995) is

$$\boldsymbol{B}(\mathbf{x}_+) = \boldsymbol{B}(\mathbf{x}_c) - \boldsymbol{G}_c\mathbf{d}_c\mathbf{d}_c^T\boldsymbol{G}_c/\mathbf{d}_c^T\boldsymbol{G}\mathbf{d}_c + \mathbf{q}_c\mathbf{q}_c^T/\mathbf{q}_c^T\mathbf{d}_c, \tag{8.1.4.14}$$

where $\boldsymbol{G}_c = \boldsymbol{J}(\mathbf{x}_+)^T\boldsymbol{W}\boldsymbol{J}(\mathbf{x}_+) + \boldsymbol{B}(\mathbf{x}_c)$. Since $\boldsymbol{J}(\mathbf{x}_+)$ and $\boldsymbol{M}(\mathbf{x}_+)$ must be computed anyway, this technique maximizes the use of information with little additional computing. The resulting approximation to the Hessian, however, may not be positive definite, and precautions must be taken to ensure convergence. In the vicinity of a correct solution, where the residuals are small, the addition of $\boldsymbol{B}$ is not likely to help much, and it can be dropped. Far from the solution, however, it can be very helpful. An implementation of this procedure has been described by Bunch, Gay & Welsch (1993); it appears to be at least as efficient as the trust region, Levenberg–Marquardt procedure, and is probably better when residuals are large.

In actual practice, it is not the Hessian matrix itself that is updated, but rather its Cholesky factor (Prince, 1994). This requires approximately the same number of operations and allows the solution of the linear system for computing $\mathbf{d}$ in a time that increases in proportion to p^2. This strategy also allows the use of the approximate Hessian in convergence checks with no significant computational overhead and no extra storage, as would be required for storing both the Hessian and its inverse.

8.1.4.4. *Stopping rules*

An iterative algorithm must contain some criterion for termination of the iteration process. This is a delicate part of all nonlinear optimization codes, and depends strongly on the scaling of the parameters. Although exceptions exist to almost all reasonable scaling rules, a basic principle is that a unit change in any variable should have approximately the same effect on the sum of squares. Thus, as discussed in Subsection 8.1.3.3 (equation 8.1.3.9), the ideal unit for each parameter is the standard uncertainty of its estimate, which can usually be adequately approximated by the reciprocal of the column norm of $\boldsymbol{J}$. In modern codes, the user has the option of supplying a diagonal scaling matrix whose elements are the reciprocals of some estimate of a typical 'significant' shift in the corresponding parameter.

In principle, the following conditions should hold when the convergence of a well scaled, least-squares procedure has reached its useful limit:

(1) $S(\mathbf{x}_c) = S(\widehat{\mathbf{x}})$;
(2) $\boldsymbol{J}(\mathbf{x}_c)^T\boldsymbol{W}[\mathbf{y} - \boldsymbol{M}(\mathbf{x}_c)] = \boldsymbol{O}$;
(3) $\mathbf{x}_c = \widehat{\mathbf{x}}$.

The actual stopping rules must be chosen relative to the algorithm used (other conditions also exist) and the particular application. It is clear, however, that, since $\widehat{\mathbf{x}}$ is not known, the last two conditions cannot be used as stated. Also, these tests are dependent on the scaling of the problem, and the variables are not related to the sizes of the quantities involved. We present tests, therefore, that are relative error tests that take into account the scaling of the variables.

The general, relative error test is stated as follows. Two scalar quantities, a and b, are said to satisfy a relative error test with respect to a tolerence T if

$$\frac{|a-b|}{|a|} \leq T. \tag{8.1.4.15}$$

Roughly speaking, if T is of the form 10^{-q} then a and b agree to q digits. Obviously, there is a problem with this test if $a = 0$ and there will be numerical difficulties if a is close to 0. Thus, in practice, (8.1.4.15) is replaced by

$$|a-b| \leq (|a|+1)T, \tag{8.1.4.16}$$

which reduces to an absolute error test as $a \to 0$. A careful examination may be required to set this tolerance correctly, but, typically, if one of the fast, stable algorithms is used, only a few more iterations are necessary to get six or eight digits if one or two are already known. Note also that the actual value depends on ε, the relative machine precision. It is fruitless to seek more digits of accuracy than are expressed in the machine representation.

A test based on condition (1) is often implemented by using the linear approximation to $\boldsymbol{M}$ or the quadratic approximation to S. Thus, using the quadratic approximation to S, we can compute the predicted reduction by

$$\Delta_{\text{pred}} = S(\mathbf{x}_c) - \mathbf{d}^T\boldsymbol{J}^T\boldsymbol{W}[\mathbf{y} - \boldsymbol{M}(\mathbf{x}_c)]. \tag{8.1.4.17}$$

Similarly, the actual reduction is

$$\Delta_{\text{act}} = S(\mathbf{x}_c) - S(\mathbf{x}_+). \tag{8.1.4.18}$$

The test then becomes $\Delta_{\text{pred}} \leq [1 + S(\mathbf{x}_c)]T$, $\Delta_{\text{act}} \leq [1 + S(\mathbf{x}_c)]T$, and $\Delta_{\text{act}} \leq 2\Delta_{\text{pred}}$. That is, we want both the predicted and actual reductions to be small and the actual reduction to agree reasonably well with the predicted reduction. A typical value for T should be 10^{-4}, although the value again depends on ε, and the user is cautioned not to make this tolerance too loose.

For a test on condition (2), we compute the cosine of the angle between the vector of residuals and the linear subspace spanned by the columns of $\mathbf{J}$,

$$\cos\zeta = \{\mathbf{d}^T\boldsymbol{J}^T\boldsymbol{W}[\mathbf{y} - \boldsymbol{M}(\mathbf{x}_+)]\}/[(\mathbf{d}^T\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J}\mathbf{d})S(\mathbf{x}_+)]^{1/2}. \tag{8.1.4.19}$$

The test is $\cos\zeta \leq T$, where, again, T should be 10^{-4} or smaller.

Test 3 above is usually only present to prevent the process from continuing when almost nothing is happening. Clearly, we do not know $\hat{\mathbf{x}}$, thus the test is typically that corresponding elements of $\mathbf{x}_+$ and $\mathbf{x}_c$ satisfy (8.1.4.16), where T is chosen to be 10^{-q}. A recommended value of q is half the number of digits carried in the computation, *e.g.* $q = 8$ for standard 64-bit (double-precision or 16 digit) calculations. Sometimes, the relative error test is of the form $|(\mathbf{x}_+)_j - (\mathbf{x}_c)_j|/\sigma_j \leq T$, where σ_j is the standard uncertainty computed from the inverse Hessian in the last iteration. Although this test has some statistical validity, it is quite expensive and usually not worth the work involved to compute.

8.1.4.5. *Recommendations*

One situation in which the Gauss–Newton algorithm behaves particularly poorly is in the vicinity of a saddle point in parameter space, where the true Hessian matrix is not positive definite. This occurs in structure refinement where a symmetric model is refined to convergence and then is replaced by a less symmetric model. The hypersurface of S will have negative curvature in a finite sized region of the parameter space for the less-symmetric model, and it is *essential* to use a safeguarded algorithm, one that incorporates a line search or a trust region, in order to get out of that region.

On the basis of this discussion, we can draw the following conclusions:

(1) In cases where the fit is poor, owing to an incomplete model or in the initial stages of refinement, methods based on the quadratic approximation to S (quasi-Newton methods) often perform better. This is particularly important when the model is close to a more symmetric configuration. These methods are more expensive per iteration and generally require more storage, but their greater stability in such problems usually justifies the cost.

(2) With small residual problems, where the model is complete and close to the solution, a safeguarded Gauss–Newton method is preferred. The trust-region implementation (Levenberg–Marquardt algorithm) has been very successful in practice.

(3) The best advice is to pick a good implementation of either method and stay with it.

8.1.5. Numerical methods for large-scale problems

Because the least-squares problems arising in crystallography are often very large, the methods we have discussed above are not always the most efficient. Some large problems have special structure that can be exploited to produce quite efficient algorithms. A particular special structure is sparsity, that is, the problems have Jacobian matrices that have a large fraction of their entries zero. Of course, not all large problems are sparse, so we shall also discuss approaches applicable to more general problems.

8.1.5.1. *Methods for sparse matrices*

We shall first discuss large, sparse, linear least-squares problems (Heath, 1984), since these techniques form the basis for nonlinear extensions. As we proceed, we shall indicate how the procedures should be modified in order to handle nonlinear problems. Recall that the problem is to find the minimum of the quadratic form $[\mathbf{y} - \boldsymbol{A}\mathbf{x}]^T\boldsymbol{W}[\mathbf{y} - \boldsymbol{A}\mathbf{x}]$, where $\mathbf{y}$ is a vector of observations, $\boldsymbol{A}\mathbf{x}$ represents a vector of the values of a set of linear model functions that predict the values of $\mathbf{y}$, and $\boldsymbol{W}$ is a positive-definite weight matrix. Again, for convenience, we make the transformation $\mathbf{y}' = \boldsymbol{U}\mathbf{y}$, where $\boldsymbol{U}$ is the upper triangular Cholesky factor of $\boldsymbol{W}$, and $\boldsymbol{Z} = \boldsymbol{U}\boldsymbol{A}$, so that the quadratic form becomes $(\mathbf{y}' - \boldsymbol{Z}\mathbf{x})^T(\mathbf{y}' - \boldsymbol{Z}\mathbf{x})$, and the minimum is the solution of the system of normal equations, $\boldsymbol{Z}^T\boldsymbol{Z}\mathbf{x} = \boldsymbol{Z}^T\mathbf{y}'$. Even if $\boldsymbol{Z}$ is sparse, it is easy to see that $\boldsymbol{H} = \boldsymbol{Z}^T\boldsymbol{Z}$ need not be sparse, because if even one row of $\boldsymbol{Z}$ has all of its elements nonzero, all elements of $\boldsymbol{H}$ will be nonzero. Therefore, the direct use of the normal equations may preclude the efficient exploitation of sparsity. But suppose $\boldsymbol{H}$ *is* sparse. The next step in solving the normal equations is to compute the Cholesky decomposition of $\boldsymbol{H}$, and it may turn out that the Cholesky factor is not sparse. For example, if $\boldsymbol{H}$ has the form

$$\boldsymbol{H} = \begin{pmatrix} x & x & x & x \\ x & x & 0 & 0 \\ x & 0 & x & 0 \\ x & 0 & 0 & x \end{pmatrix},$$

where x represents a nonzero element, then the Cholesky factor, $\boldsymbol{R}$, will not be sparse, but if

references