

Practical Newton Methods

We saw in Chapter 3 that the pure Newton method with unit steps converges rapidly once it approaches a minimizer x^* . This simple algorithm is inadequate for general use, however, since it may fail to converge to a solution from remote starting points. Even if it does converge, its behavior may be erratic in regions where the function is not convex. Our first goal in designing a practical Newton-based algorithm is to make it robust and efficient in all cases.

Recall that the basic Newton step p_k^N is obtained by solving the symmetric $n \times n$ linear system

$$\nabla^2 f(x_k) p_k^N = -\nabla f(x_k). \quad (6.1)$$

To obtain global convergence we require the search direction p_k^N to be a descent direction, which will be true here if the Hessian $\nabla^2 f(x_k)$ is positive definite. However, if the Hessian is not positive definite, or is close to being singular, p_k^N may be an ascent direction or may be excessively long. In this chapter, we describe two strategies for ensuring that the step is always of good quality. In the first strategy we solve (6.1) using the conjugate gradient (CG) method (see Chapter 5), terminating if negative curvature is encountered. There are both line search and trust-region implementations of this strategy, which we call the *Newton–CG*

method. The second approach consists in modifying the Hessian matrix $\nabla^2 f(x_k)$ before or during the solution of the system (6.1) so that it becomes sufficiently positive definite. We call this the *modified Newton method*.

Another concern in designing practical Newton methods is to keep the computational cost as small as possible. In the Newton–CG method, we accomplish this goal by terminating the CG iteration before an exact solution to (6.1) is found. This *inexact* Newton approach thus computes an approximation p_k to the pure Newton step p_k^N . When using a direct method to solve the Newton equations we can take advantage of any sparsity structure in the Hessian by using sparse Gaussian elimination techniques to factor $\nabla^2 f(x_k)$ and then using the factors to obtain an exact solution of (6.1).

The computation of the Hessian matrix $\nabla^2 f(x_k)$ usually represents a major task in the implementation of Newton’s method. If the Hessian is not available in analytic form, we can use automatic differentiation techniques to compute it, or we can use finite differences to estimate it. A detailed treatment of these topics is deferred to Chapter 7. With all these ingredients—modification of the pure Newton iteration, exploitation of sparsity, and differentiation techniques—the Newton methods described in this chapter represent some of the most reliable and powerful methods for solving both small or large unconstrained optimization problems.

Since inexact Newton steps are useful in both line search and trust-region implementations, we discuss them first.

6.1 INEXACT NEWTON STEPS

We have noted that a drawback of the pure Newton method is the need to solve the equations (6.1) exactly. Techniques based on Gaussian elimination or another type of factorization of the coefficient matrix can be expensive when the number of variables is large. An accurate solution to (6.1) may not be warranted in any case, since the quadratic model used to derive the Newton equations may not provide a good prediction of the behavior of the function f , especially when the current iterate x_k is remote from the solution x^* . It is therefore appealing to apply an iterative method to (6.1), terminating the iterations at some approximate (inexact) solution of this system.

Most rules for terminating the iterative solver for (6.1) are based on the residual

$$r_k = \nabla^2 f(x_k)p_k + \nabla f(x_k), \quad (6.2)$$

where p_k is the inexact Newton step. Since the size of the residual changes if f is multiplied by a constant (i.e., r is not invariant to scaling of the objective function), we consider its size relative to that of the right-hand-side vector in (6.1), namely $\nabla f(x_k)$. We therefore terminate the iterative solver when

$$\|r_k\| \leq \eta_k \|\nabla f(x_k)\|, \quad (6.3)$$

where the sequence $\{\eta_k\}$ (with $0 < \eta_k < 1$ for all k) is called the *forcing sequence*.

We now study how the rate of convergence of inexact Newton methods based on (6.1)–(6.3) is affected by the choice of the forcing sequence. Our first result says that local convergence is obtained simply by ensuring that η_k is bounded away from 1.

Theorem 6.1.

Suppose that $\nabla f(x)$ is continuously differentiable in a neighborhood of a minimizer x^ , and assume that $\nabla^2 f(x^*)$ is positive definite. Consider the iteration $x_{k+1} = x_k + p_k$ where p_k satisfies (6.3), and assume that $\eta_k \leq \eta$ for some constant $\eta \in [0, 1)$. Then, if the starting point x_0 is sufficiently near x^* , the sequence $\{x_k\}$ converges to x^* linearly. That is, for all k sufficiently large, we have*

$$\|x_{k+1} - x^*\| \leq c \|x_k - x^*\|,$$

for some constant $0 < c < 1$.

Note that the condition on the forcing sequence $\{\eta_k\}$ is not very restrictive, in the sense that if we relaxed it just a little, it would yield an algorithm that obviously does not converge. Specifically, if we allowed $\eta_k \geq 1$, the step $p_k = 0$ would satisfy (6.3) at every iteration, but the resulting method would set $x_k = x_0$ for all k and would not converge to the solution.

Rather than giving a rigorous proof of Theorem 6.1, we now present an informal derivation that contains the essence of the argument and motivates the results that follow.

Since the Hessian matrix $\nabla^2 f(x^*)$ is assumed to be positive definite, there is a positive constant L such that $\|\nabla^2 f(x_k)^{-1}\| \leq L$ for all x_k sufficiently close to x^* . We therefore have from (6.2) that the inexact Newton step satisfies

$$\|p_k\| \leq L(\|\nabla f(x_k)\| + \|r_k\|) \leq 2L\|\nabla f(x_k)\|,$$

where the second inequality follows from (6.3) and $\eta_k < 1$. By using this expression together with Taylor's theorem we obtain

$$\begin{aligned} \nabla f(x_{k+1}) &= \nabla f(x_k) + \nabla^2 f(x_k)p_k + O(\|p_k\|^2) \\ &= \nabla f(x_k) - (\nabla f(x_k) - r_k) + O(L^2\|\nabla f(x_k)\|^2) \\ &= r_k + O(\|\nabla f(x_k)\|^2). \end{aligned} \tag{6.4}$$

By taking norms and recalling (6.3), we have that

$$\|\nabla f(x_{k+1})\| \leq \eta_k \|\nabla f(x_k)\| + O(\|\nabla f(x_k)\|^2). \tag{6.5}$$

Therefore, if x_k is close to x^* , we can expect $\|\nabla f(x)\|$ to decrease by a factor of approximately $\eta_k < 1$ at every iteration. More precisely, we have that

$$\limsup_{k \rightarrow \infty} \frac{\|\nabla f(x_{k+1})\|}{\|\nabla f(x_k)\|} \leq \eta < 1.$$

Relation (6.4) also suggests that if $r_k = o(\|\nabla f(x_k)\|)$, then the rate of convergence in the gradient will be superlinear, for in this case the limit satisfies

$$\limsup_{k \rightarrow \infty} \frac{\|\nabla f(x_{k+1})\|}{\|\nabla f(x_k)\|} = 0.$$

Similarly, if $r_k = O(\|\nabla f(x_k)\|^2)$, we have

$$\limsup_{k \rightarrow \infty} \frac{\|\nabla f(x_{k+1})\|}{\|\nabla f(x_k)\|^2} = c,$$

for some constant c , suggesting that the quadratic rate of convergence of the exact Newton method has been recaptured.

It is not difficult to show that the iterates $\{x_k\}$ converge to x^* at the same rate as the sequence of gradients $\{\nabla f(x_k)\}$ converges to zero.

Theorem 6.2.

Suppose that the conditions of Theorem 6.1 hold and assume that the iterates $\{x_k\}$ generated by the inexact Newton method converge to x^ . Then the rate of convergence is superlinear if $\eta_k \rightarrow 0$ and quadratic if $\eta_k = O(\|\nabla f(x_k)\|)$.*

The proof makes use of Taylor's theorem and the relation (6.4). We leave the details as an exercise.

To obtain superlinear convergence we can set, for example, $\eta_k = \min(0.5, \sqrt{\|\nabla f(x_k)\|})$, while the choice $\eta_k = \min(0.5, \|\nabla f(x_k)\|)$ would yield quadratic convergence.

All the results presented in this section, which are proved by Dembo, Eisenstat, and Steihaug [66], are local in nature: They assume that the sequence $\{x_k\}$ eventually enters the near vicinity of the solution x^* . They also assume that the unit step length $\alpha_k = 1$ is taken, and hence that globalization strategies (line search, trust-region) do not get in the way of rapid convergence. In the next sections we show that inexact Newton strategies can, in fact, be incorporated in practical line search and trust-region implementations of Newton's method, yielding algorithms with good local and global convergence properties.

6.2 LINE SEARCH NEWTON METHODS

We now describe line search implementations of Newton's method that are practical for small and large problems. Each iteration has the form $x_{k+1} = x_k + \alpha_k p_k$, where α_k is the step length and p_k is either the pure Newton direction p_k^N or an approximation to it. We have dealt with the issue of selecting the step length α_k in Chapter 3. As stated there, α_k can be chosen to satisfy the Wolfe conditions (3.6) or the Goldstein conditions (3.11), or it can be obtained by an Armijo backtracking line search described in Section 3.1. We stress again that the line search should always try the unit step length $\alpha_k = 1$ first, so that this step length is used when acceptable. We now consider two different techniques for computing the search direction p_k .

LINE SEARCH NEWTON-CG METHOD

In the line search Newton-CG method, also known as the *truncated Newton method*, we compute the search direction by applying the CG method to the Newton equations (6.1), and attempt to satisfy a termination test of the form (6.3). Note, however, that the CG method is designed to solve positive definite systems, and that the Hessian could have negative eigenvalues away from the solution. Therefore, we terminate the CG iteration as soon as a direction of negative curvature is generated. This adaptation of the CG method ensures that the direction p_k is a descent direction, and that the fast convergence rate of the pure Newton method is preserved.

We discuss now the details of this inner CG iteration, which is a slight modification of Algorithm CG described in Chapter 5. The linear system to be solved is (6.1), and thus in the notation of Chapter 5, the coefficient matrix is $A = \nabla^2 f_k$ and the right-hand-side vector is $b = -\nabla f_k$. We will use superscripts to denote the iterates $\{x^{(i)}\}$ and search directions $\{p^{(i)}\}$ generated by the CG iteration, so as to distinguish them from the Newton iterates x_k and search directions p_k . We impose the following three requirements on the CG iteration for solving (6.1).

- (a) The starting point for the CG iteration is $x^{(0)} = 0$.
- (b) Negative curvature test. If a search direction $p^{(i)}$ generated by the CG iteration satisfies

$$(p^{(i)})^T A p^{(i)} \leq 0, \quad (6.6)$$

then we check whether this is the first CG iteration. If so ($i = 0$), we complete the first CG iteration, compute the new iterate $x^{(1)}$, and stop. If (6.6) holds and $i > 0$, then we stop the CG iteration immediately and return the most recently generated solution point, $x^{(i)}$.

- (c) The Newton step p_k is defined as the final CG iterate $x^{(f)}$.

We could also choose $x^{(0)}$ to be the optimal solution of the previous linear system, which is the same as the most recently generated Newton–CG step. Both choices seem to perform equally well in this line search context.

If the condition (6.6) is not satisfied, then the i th CG iteration is exactly as described in Algorithm CG of Chapter 5. A vector $p^{(i)}$ satisfying (6.6) with a strict inequality is said to be a *direction of negative curvature* for A . Note that if negative curvature is encountered during the first CG iteration, then the Newton–CG direction p_k is the steepest descent direction $-\nabla f_k$. This is the reason for choosing the initial estimate in the CG iteration as $x^{(0)} = 0$. On the other hand, if the CG method performs more than one iteration, the Newton–CG direction will also be a descent direction; see Exercise 2. Preconditioning can be used in the CG iteration.

The Newton–CG method does not require explicit knowledge of the Hessian $\nabla^2 f(x_k)$. Rather, it requires only that we can supply matrix–vector products of the form $\nabla^2 f(x_k)p$ for any given vector p . This fact is useful for cases in which the user cannot easily supply code to calculate second derivatives, or where the Hessian requires too much storage. In these cases, the techniques of Chapter 7, which include automatic differentiation and finite differences, can be used to calculate the Hessian–vector products.

To illustrate the finite-differencing technique briefly, we note that we can use the approximation

$$\nabla^2 f(x_k)p \approx \frac{\nabla f(x_k + hp) - \nabla f(x_k)}{h}, \quad (6.7)$$

for some small differencing interval h . It is easy to prove that the accuracy of this approximation is $O(h)$; appropriate choices of h are discussed in Chapter 7. The price we pay for bypassing the computation of the Hessian is one new gradient evaluation per CG iteration. An alternative to finite differencing is automatic differentiation, which can in principle be used to compute $\nabla^2 f(x_k)p$ *exactly*. (Again, details are given in Chapter 7.) Methods of this type are known as *Hessian-free* Newton methods.

We can now give a general description of the Newton–CG method. For concreteness, we choose the forcing sequence as $\eta_k = \min(0.5, \sqrt{\|\nabla^2 f_k\|})$, so as to obtain a superlinear convergence rate, but other choices are possible.

Algorithm 6.1 (Line Search Newton–CG).

Given initial point x_0 ;

for $k = 0, 1, 2, \dots$

 Compute a search direction p_k by applying the CG method to

$\nabla^2 f(x_k)p = -\nabla f_k$, starting from $x^{(0)} = 0$. Terminate when
 $\|r_k\| \leq \min(0.5, \sqrt{\|\nabla f_k\|})\|\nabla f(x_k)\|$, or if negative curvature is
 encountered, as described in (b);

 Set $x_{k+1} = x_k + \alpha_k p_k$, where α_k satisfies the Wolfe, Goldstein, or
 Armijo backtracking conditions;

end

This method is well suited for large problems, but it has a minor weakness, especially in the case where no preconditioning is used in the CG iteration. When the Hessian $\nabla^2 f_k$ is nearly singular, the Newton–CG direction can be excessively long, requiring many function evaluations in the line search. In addition, the reduction in the function may be very small in this case. To alleviate this difficulty we can try to normalize the Newton step, but good rules for doing so are difficult to determine. (They run the risk of undermining the rapid convergence of Newton’s method in the case where the pure Newton step is well scaled.) It is preferable to introduce a threshold value in (6.6), but once again, good choices of this value are difficult to determine. The trust-region implementation of the Newton–CG method described in Section 6.4 deals more effectively with this problematic situation, and is therefore slightly preferable, in our opinion.

MODIFIED NEWTON’S METHOD

In many applications it is desirable to use a direct linear algebra technique, such as Gaussian elimination, to solve the Newton equations (6.1). If the Hessian is not positive definite, or is close to being singular, then we can modify this matrix before or during the solution process to ensure that the computed direction p_k satisfies a linear system identical to (6.1) except that the coefficient matrix is replaced with a positive definite approximation. The modified Hessian is obtained by adding either a positive diagonal matrix or a full matrix to the true Hessian $\nabla^2 f(x_k)$. Following is a general description of this method.

Algorithm 6.2 (Line Search Newton with Modification).

```

given initial point  $x_0$ ;
for  $k = 0, 1, 2, \dots$ 
    Factorize the matrix  $B_k = \nabla^2 f(x_k) + E_k$ , where  $E_k = 0$  if  $\nabla^2 f(x_k)$ 
        is sufficiently positive definite; otherwise,  $E_k$  is chosen to
        ensure that  $B_k$  is sufficiently positive definite;
    Solve  $B_k p_k = -\nabla f(x_k)$ ;
    Set  $x_{k+1} = x_k + \alpha_k p_k$ , where  $\alpha_k$  satisfies the Wolfe, Goldstein, or
        Armijo backtracking conditions;
end

```

The choice of Hessian modification E_k is crucial to the effectiveness of the method. Some approaches do not compute E_k explicitly, but rather introduce extra steps and tests into standard factorization procedures, modifying these procedures “on the fly” so that the computed factors are in fact the factors of a positive definite matrix. Strategies based on modifying a Cholesky factorization and on modifying a symmetric indefinite factorization of the Hessian are described in the next section.

We can establish fairly satisfactory global convergence results for Algorithm 6.2, provided that the strategy for choosing E_k (and hence B_k) satisfies the *bounded modified factorization* property. This property is that the matrices in the sequence $\{B_k\}$ have bounded

condition number whenever the sequence of Hessians $\{\nabla^2 f(x_k)\}$ is bounded; that is,

$$\text{cond}(B_k) = \|B_k\| \|B_k^{-1}\| \leq C, \quad \text{for some } C > 0 \text{ and all } k = 0, 1, 2, \dots \quad (6.8)$$

If this property holds, global convergence of the modified line search Newton method follows directly from the results of Chapter 3, as we show in the following result.

Theorem 6.3.

Let f be twice continuously differentiable on an open set \mathcal{D} , and assume that the starting point x_0 of Algorithm 6.2 is such that the level set $L = \{x \in \mathcal{D} : f(x) \leq f(x_0)\}$ is compact. Then if the bounded modified factorization property holds, we have that

$$\lim_{k \rightarrow \infty} \nabla f(x_k) = 0.$$

PROOF. The line search ensures that all iterates x_k remain in the level set \mathcal{D} . Since $\nabla^2 f(x)$ is assumed to be continuous on \mathcal{D} , and since \mathcal{D} is compact, we have that the sequence of Hessians $\{\nabla^2 f(x_k)\}$ is bounded, and therefore (6.8) holds. Since Algorithm 6.2 uses one of the line searches for which Zoutendijk's theorem (Theorem 3.2) holds, the result follows from (3.16). \square

We now consider the convergence rate of Algorithm 6.2. Suppose that the sequence of iterates x_k converges to a point x^* where $\nabla^2 f(x^*)$ is sufficiently positive definite in the sense that the modification strategies described in the next section return the modification $E_k = 0$ for all sufficiently large k . By Theorem 3.5, we have that $\alpha_k = 1$ for all sufficiently large k , so that Algorithm 6.2 reduces to a pure Newton method, and the rate of convergence is quadratic.

For problems in which ∇f^* is close to singular, there is no guarantee that the modification E_k will eventually vanish, and the convergence rate may be only linear.

6.3 HESSIAN MODIFICATIONS

We now discuss procedures for modifying the Hessian matrices $\nabla^2 f(x_k)$ that implicitly or explicitly choose the modification E_k so that the matrix $B_k = \nabla^2 f(x_k) + E_k$ in Algorithm 6.2 is sufficiently positive definite. Besides requiring the modified matrix B_k to be well conditioned (so that Theorem 6.3 holds), we would like the modification to be as small as possible, so that the second-order information in the Hessian is preserved as far as possible. Naturally, we would also like the modified factorization to be computable at moderate cost.

To set the stage for the matrix factorization techniques that will be used in Algorithm 6.2 we will begin by describing an “ideal” strategy based on the eigenvalue decomposition of $\nabla^2 f(x_k)$.

EIGENVALUE MODIFICATION

Consider a problem in which, at the current iterate x_k , $\nabla f(x_k) = (1, -3, 2)$ and $\nabla^2 f(x_k) = \text{diag}(10, 3, -1)$, which is clearly indefinite. By the spectral decomposition theorem (see the Appendix) we can define $Q = I$ and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \lambda_3)$, and write

$$\nabla^2 f(x_k) = Q \Lambda Q^T = \sum_{i=1}^n \lambda_i q_i q_i^T. \quad (6.9)$$

The pure Newton step—the solution of (6.1)—is $p_k^N = (-0.1, 1, 2)$, which is not a descent direction, since $\nabla f(x_k)^T p_k^N > 0$. One might suggest a modified strategy in which we replace $\nabla^2 f(x_k)$ by a positive definite approximation B_k , in which all negative eigenvalues in $\nabla^2 f(x_k)$ are replaced by a small positive number δ that is somewhat larger than machine accuracy \mathbf{u} ; say $\delta = \sqrt{\mathbf{u}}$. Assuming that machine accuracy is 10^{-16} , the resulting matrix in our example is

$$B_k = \sum_{i=1}^2 \lambda_i q_i q_i^T + \delta q_3 q_3^T = \text{diag}(10, 3, 10^{-8}), \quad (6.10)$$

which is numerically positive definite and whose curvature along the eigenvectors q_1 and q_2 has been preserved. Note, however, that the search direction based on this modified Hessian is

$$p_k = -B_k^{-1} \nabla f_k = - \sum_{i=1}^2 \frac{1}{\lambda_i} q_i (q_i^T \nabla f_k) - \frac{1}{\delta} q_3 (q_3^T \nabla f(x_k)) \approx -(2 \times 10^8) q_3. \quad (6.11)$$

For small δ , this step is nearly parallel to q_3 (with relatively small contributions from q_1 and q_2) and very long. Although f decreases along the direction p_k , its extreme length violates the spirit of Newton's method, which relies on a quadratic approximation of the objective function that is valid in a neighborhood of the current iterate x_k . It is therefore questionable that this search direction is effective.

A different type of search direction would be obtained by flipping the signs of the negative eigenvalues in (6.9), which amounts to setting $\delta = 1$ in our example. Again, there is no consensus as to whether this constitutes a desirable modification to Newton's method. Various other strategies can be considered: We could set the last term in (6.11) to zero, so that the search direction has no components along the negative curvature directions, or we could adapt the choice of δ to ensure that the length of the step is not excessive. (This last strategy has the flavor of trust-region methods.) We see that there is a great deal of freedom in devising modification strategies, and there is currently no agreement on which is the ideal strategy.

Setting the issue of the choice of δ aside for the moment, let us look more closely at the process of modifying a matrix so that it becomes positive definite. The modification (6.10) to the example matrix (6.9) can be shown to be optimal in the following sense. If A is a symmetric matrix with spectral decomposition $A = Q\Lambda Q^T$, then the correction matrix ΔA of *minimum Frobenius norm* that ensures that $\lambda_{\min}(A + \Delta A) \geq \delta$ is given by

$$\Delta A = Q \operatorname{diag}(\tau_i) Q^T, \quad \text{with} \quad \tau_i = \begin{cases} 0, & \lambda_i \geq \delta, \\ \delta - \lambda_i, & \lambda_i < \delta. \end{cases} \quad (6.12)$$

Here $\lambda_{\min}(A)$ denotes the smallest eigenvalue of A , and the Frobenius norm of a matrix is defined as $\|A\|_F^2 = \sum_{i,j=1}^n a_{ij}^2$ (see (A.40)). Note that ΔA is not diagonal in general, and that the modified matrix is given by

$$A + \Delta A = Q(\Lambda + \operatorname{diag}(\tau_i))Q^T.$$

By using a different norm we can obtain a *diagonal* modification. Suppose again that A is a symmetric matrix with spectral decomposition $A = Q\Lambda Q^T$. A correction matrix ΔA with minimum Euclidean norm that satisfies $\lambda_{\min}(A + \Delta A) \geq \delta$ is given by

$$\Delta A = \tau I, \quad \text{with} \quad \tau = \max(0, \delta - \lambda_{\min}(A)). \quad (6.13)$$

The modified matrix now has the form

$$A + \tau I, \quad (6.14)$$

which happens to have the same form as the matrix occurring in (unscaled) trust-region methods (see Chapter 4). All the eigenvalues of (6.14) have thus been shifted, and all are greater than δ .

These results suggest that both diagonal and nondiagonal modifications can be considered. Even though we have not answered the question of what constitutes a good modification, various practical diagonal and nondiagonal modifications have been proposed and implemented in software. They do not make use of the spectral decomposition of the Hessian, since this is generally too expensive to compute. Instead, they use Gaussian elimination, choosing the modifications indirectly and hoping that somehow they will produce good steps. Numerical experience indicates that the strategies described next often (but not always) produce good search directions.

ADDING A MULTIPLE OF THE IDENTITY

Perhaps the simplest idea is to find a scalar $\tau > 0$ such that $\nabla^2 f(x_k) + \tau I$ is sufficiently positive definite. From the previous discussion we know that τ must satisfy (6.13), but we

normally don't have a good estimate of the smallest eigenvalue of the Hessian. A simple idea is to recall that the largest eigenvalue (in absolute value) of a matrix A is bounded by the Frobenius norm $\|A\|_F$. This suggests the following strategy; here a_{ii} denotes a diagonal element of A .

Algorithm 6.3 (Cholesky with Added Multiple of the Identity).

```

set  $\beta \leftarrow \|A\|_F$ ;
if  $\min_i a_{ii} > 0$ 
     $\tau_0 \leftarrow 0$ 
else
     $\tau_0 \leftarrow \beta/2$ ;
end (if)
for  $k = 0, 1, 2, \dots$ 
    Attempt to apply the incomplete Cholesky algorithm to obtain

```

$$LL^T = A + \tau_k I;$$

```

    if factorization is completed successfully
        stop and return  $L$ 
    else
         $\tau_{k+1} \leftarrow \max(2\tau_k, \beta/2)$ ;
    end (if)
end (for)

```

This strategy is quite simple and may be preferable to the modified factorization techniques described next, but it suffers from two drawbacks. The value of τ generated by this procedure may be unnecessarily large, which would bias the modified Newton direction too much toward the steepest descent direction. In addition, every value of τ_k requires a new factorization of $A + \tau_k I$, which can be quite expensive if several trial values are generated. (The symbolic factorization of A is performed only once, but the numerical factorization must be performed from scratch every time.)

MODIFIED CHOLESKY FACTORIZATION

A popular approach for modifying a Hessian matrix that is not positive definite is to perform a Cholesky factorization of $\nabla^2 f(x_k)$, but to increase the diagonal elements encountered during the factorization (where necessary) to ensure that they are sufficiently positive. This *modified Cholesky* approach is designed to accomplish two goals: It guarantees that the modified Cholesky factors exist and are bounded relative to the norm of the actual Hessian, and it does not modify the Hessian if it is sufficiently positive definite.

We begin our description of this approach by briefly reviewing the Cholesky factorization. Every symmetric and positive definite matrix A can be written as

$$A = LDL^T, \quad (6.15)$$

where L is a lower triangular matrix with unit diagonal elements and D is a diagonal matrix with positive elements on the diagonal. By equating the elements in (6.15), column by column, it is easy to derive formulas for computing L and D .

□ **EXAMPLE 6.1**

Consider the case $n = 3$. The equation $A = LDL^T$ is given by

$$\begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} \begin{bmatrix} 1 & l_{21} & l_{31} \\ 0 & 1 & l_{32} \\ 0 & 0 & 1 \end{bmatrix}.$$

(The notation indicates that A is symmetric.) By equating the elements of the first column, we have

$$\begin{aligned} a_{11} &= d_1, \\ a_{21} &= d_1 l_{21} \Rightarrow l_{21} = a_{21}/d_1, \\ a_{31} &= d_1 l_{31} \Rightarrow l_{31} = a_{31}/d_1. \end{aligned}$$

Proceeding with the next two columns, we obtain

$$\begin{aligned} a_{22} &= d_1 l_{21}^2 + d_2 \Rightarrow d_2 = a_{22} - d_1 l_{21}^2, \\ a_{32} &= d_1 l_{31} l_{21} + d_2 l_{32} \Rightarrow l_{32} = (a_{32} - d_1 l_{31} l_{21})/d_2, \\ a_{33} &= d_1 l_{31}^2 + d_2 l_{32}^2 + d_3 \Rightarrow d_3 = a_{33} - d_1 l_{31}^2 - d_2 l_{32}^2. \end{aligned}$$

□

This procedure is generalized in the following algorithm.

Algorithm 6.4 (Cholesky Factorization, LDL^T Form).

```

for    $j = 1, 2, \dots, n$ 
     $c_{jj} \leftarrow a_{jj} - \sum_{s=1}^{j-1} d_s l_{js}^2$ ;
     $d_j \leftarrow c_{jj}$ ;
    for    $i = j + 1, \dots, n$ 
         $c_{ij} \leftarrow a_{ij} - \sum_{s=1}^{j-1} d_s l_{is} l_{js}$ ;

```

```

         $l_{ij} \leftarrow c_{ij}/d_j;$ 
    end
end

```

One can show (see, for example, Golub and Van Loan [115, Section 4.2.3]) that the diagonal elements d_{jj} are all positive whenever A is positive definite. The scalars c_{ij} have been introduced only to facilitate the description of the modified factorization discussed below. We should note that Algorithm 6.4 differs a little from the standard form of the Cholesky factorization, which produces a lower triangular matrix M such that

$$A = MM^T. \quad (6.16)$$

In fact, we can make the identification $M = LD^{1/2}$ to relate M to the factors L and D computed in Algorithm 6.4. The technique for computing M appears as Algorithm A.2 in the Appendix.

If A is indefinite, the factorization $A = LDL^T$ may not exist. Even if it does exist, Algorithm 6.4 is numerically unstable when applied to such matrices, in the sense that the elements of L and D can become arbitrarily large. It follows that a strategy of computing the LDL^T factorization and then modifying the diagonal after the fact to force its elements to be positive may break down, or may result in a matrix that is drastically different from A .

Instead, we will modify the matrix A during the course of the factorization in such a way that all elements in D are sufficiently positive, and so that the elements of D and L are not too large. To control the quality of the modification, we choose two positive parameters δ and β and require that during the computation of the j th columns of L and D in Algorithm 6.4 (that is, for each j in the outer loop of the algorithm) the following bounds be satisfied:

$$d_j \geq \delta, \quad |m_{ij}| \leq \beta, \quad i = j + 1, \dots, n. \quad (6.17)$$

To satisfy these bounds we only need to change one step in Algorithm 6.4: The formula for computing the diagonal element d_j in Algorithm 6.4 is replaced by

$$d_j = \max \left(|c_{jj}|, \left(\frac{\theta_j}{\beta} \right)^2, \delta \right), \quad \text{with } \theta_j = \max_{j < i \leq n} |c_{ij}|. \quad (6.18)$$

To verify that (6.17) holds, we note from Algorithm 6.4 that $c_{ij} = l_{ij}d_j$, and therefore

$$|m_{ij}| = |l_{ij}\sqrt{d_j}| = \frac{|c_{ij}|}{\sqrt{d_j}} \leq \frac{|c_{ij}|\beta}{\theta_j} \leq \beta, \quad \text{for all } i > j.$$

We note that θ_j can be computed prior to d_j because the elements c_{ij} in the second **for** loop of Algorithm 6.4 do not involve d_j . In fact, this is the reason for introducing the quantities c_{ij} into the algorithm. This observation also suggests that the computations

should be reordered so that the c_{ij} are computed before the diagonal element d_j . We use this procedure in Algorithm 6.5, the modified Cholesky factorization algorithm described below. To try to reduce the size of the modification, symmetric interchanges of rows and columns are introduced, so that at the j th step of the factorization, the j th row and column are those that yield the largest diagonal element. We also note that the computation of the elements c_{jj} can be carried out recursively after every column of the factorization is calculated.

Algorithm 6.5 (Modified Cholesky [104]).

```

given  $\delta > 0, \beta > 0$ :
for  $k = 1, 2, \dots, n$ 
     $c_{kk} = a_{kk}$ ;          (initialize the diagonal elements)
end
Find index  $q$  such that  $|c_{qq}| \geq |c_{ii}|, i = j, \dots, n$ ;
Interchange row and column  $j$  and  $q$ ;
for  $j = 1, 2, \dots, n$     (compute the  $j$ th column of  $L$ )
    for  $s = 1, 2, \dots, j - 1$ 
         $l_{js} \leftarrow c_{js}/d_s$ ;
    end
    for  $i = j + 1, \dots, n$ 
         $c_{ij} \leftarrow a_{ij} - \sum_{s=1}^{j-1} l_{js}c_{is}$ ;
    end
     $\theta_j \leftarrow 0$ ;
    if  $j \leq n$ 
         $\theta_j \leftarrow \max_{j < i \leq n} |c_{ij}|$ ;
    end
     $d_j \leftarrow \max\{|c_{jj}|, (\theta_j/\beta)^2, \delta\}$ ;
    if  $j < n$ 
        for  $i = j + 1, \dots, n$ 
             $c_{ii} \leftarrow c_{ii} - c_{ij}^2/d_j$ ;
        end
    end
end.

```

The algorithm requires approximately $n^3/6$ arithmetic operations, which is roughly the same as the standard Cholesky factorization. However, the row and column interchanges require movement of data in the computer, and the cost of this operation may be significant on large problems. No additional storage is needed beyond the amount required to store A ; the triangular factors L and D , as well as the intermediate scalars c_{ij} , can overwrite the elements of A .

Suppose that we use P to denote the permutation matrix associated with the row and column interchanges that occur during Algorithm 6.5. It is not difficult to see that the algorithm produces the Cholesky factorization of the permuted, modified matrix $PAP^T + E$,

that is,

$$PAP^T + E = LDL^T = MM^T, \quad (6.19)$$

where E is a nonnegative diagonal matrix that is zero if A is sufficiently positive definite. From an examination of the formulae for c_{jj} and d_j in Algorithm 6.5, it is clear that the diagonal entries of E are $e_j = d_j - c_{jj}$. It is also clear that incrementing c_{jj} by e_j in the factorization is equivalent to incrementing a_{jj} by e_j in the original data.

It remains only to specify the choice of the parameters δ and β in Algorithm 6.5. The constant δ is normally chosen to be close to machine accuracy \mathbf{u} ; a typical choice is

$$\delta = \mathbf{u} \max(\gamma(A) + \xi(A), 1),$$

where $\gamma(A)$ and $\xi(A)$ are, respectively, the largest-magnitude diagonal and off-diagonal elements of the matrix A , that is,

$$\gamma = \max_{1 \leq i \leq n} |a_{ii}|, \quad \xi = \max_{i \neq j} |a_{ij}|.$$

Gill, Murray, and Wright [104] suggest the following choice of β :

$$\beta = \max \left(\gamma(A), \frac{\xi(A)}{\sqrt{n^2 - 1}}, \mathbf{u} \right)^{1/2},$$

whose intent is to minimize the norm of the modification $\|E\|_\infty$.

□ EXAMPLE 6.2

Consider the matrix

$$A = \begin{bmatrix} 4.0 & 2.0 & 1.0 \\ 2.0 & 6.0 & 3.0 \\ 1.0 & 3.0 & -0.004 \end{bmatrix},$$

whose eigenvalues are, to three digits, -1.25 , 2.87 , 8.38 . Algorithm 6.5 gives the following Cholesky factor M and diagonal modification E :

$$M = \begin{bmatrix} 0.8165 & 1.8257 & 0 \\ 2.4495 & 0 & 0 \\ 1.2247 & -1.2 \times 10^{-16} & 1.2264 \end{bmatrix}, \quad E = \text{diag}(0, 0, 3.008).$$

The modified matrix is

$$A' = MM^T = \begin{bmatrix} 4.00 & 2.00 & 1.00 \\ 2.00 & 6.00 & 3.00 \\ 1.00 & 3.00 & 3.004 \end{bmatrix},$$

whose eigenvalues are 1.13, 3.00, 8.87, and whose condition number of 7.8 is quite moderate. □

One can show (Moré and Sorensen [171]) that the matrices B_k obtained by applying Algorithm 6.5 to the exact Hessians $\nabla^2 f(x_k)$ have bounded condition numbers, that is, the bound (6.8) holds for some value of C .

GERSHGORIN MODIFICATION

We now give a brief outline of an alternative modified Cholesky algorithm that makes use of Gershgorin circle estimates to increase the diagonal elements as necessary. The first step of this strategy is to apply Algorithm 6.5 to the matrix A , terminating with the usual factorization (6.19). If the perturbation matrix E is zero, we are finished, since in this case A is already sufficiently positive definite. Otherwise, we compute two upper bounds b_1 and b_2 on the smallest eigenvalue $\lambda_{\min}(A)$ of A . The first estimate b_1 is obtained from the Gershgorin circle theorem; it guarantees that $A + b_1 I$ is strictly diagonally dominant. The second upper bound b_2 is simply

$$b_2 = \max_{1 \leq i \leq n} e_{ii}.$$

We now define

$$\mu = \min(b_1, b_2),$$

and conclude the algorithm by computing the factorization of $A + \mu I$, taking the Cholesky factor of this matrix as our modified Cholesky factor. The use of b_2 gives a much needed control on the process, since the estimate b_1 obtained from the Gershgorin circle theorem can be quite loose.

It is not known whether this alternative is preferable to Algorithm 6.5 alone. Neither strategy will modify a sufficiently positive definite matrix A , but it is difficult to quantify the meaning of the term “sufficient” in terms of the smallest eigenvalue $\lambda_{\min}(A)$. Both strategies may in fact modify a matrix A whose minimum eigenvalue is greater than the parameter δ of Algorithm 6.5.

MODIFIED SYMMETRIC INDEFINITE FACTORIZATION

Another strategy for modifying an indefinite Hessian is to use a procedure based on a symmetric indefinite factorization. Any symmetric matrix A , whether positive definite or not, can be written as

$$PAP^T = LBL^T, \quad (6.20)$$

where L is unit lower triangular, B is a block diagonal matrix with blocks of dimension 1 or 2, and P is a permutation matrix (see Golub and Van Loan [115, Section 4.4]). We mentioned earlier that attempting to compute the LDL^T factorization of an indefinite matrix (where D is a *diagonal* matrix) is inadvisable because even if the factors L and D are well-defined, they may contain entries that are larger than the original elements of A , thus amplifying rounding errors that arise during the computation. However, by using the block diagonal matrix B , which allows 2×2 blocks as well as 1×1 blocks on the diagonal, we can guarantee that the factorization (6.20) always exists and can be computed by a numerically stable process.

□ EXAMPLE 6.3

The matrix

$$A = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 2 & 2 \\ 2 & 2 & 3 & 3 \\ 3 & 2 & 3 & 4 \end{bmatrix}$$

can be written in the form (6.20) with $P = [e_1, e_4, e_3, e_2]$,

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{9} & \frac{2}{3} & 1 & 0 \\ \frac{2}{9} & \frac{1}{3} & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 3 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & \frac{7}{9} & \frac{5}{9} \\ 0 & 0 & \frac{5}{9} & \frac{10}{9} \end{bmatrix}. \quad (6.21)$$

Note that both diagonal blocks in B are 2×2 .

□

The symmetric indefinite factorization allows us to determine the *inertia* of a matrix, that is, the number of positive, zero, and negative eigenvalues. One can show that the inertia of B equals the inertia of A . Moreover, the 2×2 blocks in B are always constructed to

have one positive and one negative eigenvalue. Thus the number of positive eigenvalues in A equals the number of positive 1×1 blocks plus the number of 2×2 blocks.

The first step of the symmetric indefinite factorization proceeds as follows. We identify a submatrix E of A that is suitable to be used as a pivot block. The precise criteria that can be used to choose E are described below, but we note here that E is either a single diagonal element of A (giving rise to a 1×1 pivot block), or else it consists of two diagonal elements of A (say, a_{ii} and a_{jj}) together with the corresponding off-diagonal element (that is, a_{ji} and a_{ij}). In either case, E is chosen to be nonsingular. We then find a permutation matrix Π that makes E a leading principal submatrix of A , that is,

$$\Pi A \Pi^T = \begin{bmatrix} E & C^T \\ C & H \end{bmatrix}, \quad (6.22)$$

and then perform a block factorization on this rearranged matrix, using E as the pivot block, to obtain

$$\Pi A \Pi^T = \begin{bmatrix} I & 0 \\ C E^{-1} & I \end{bmatrix} \begin{bmatrix} E & 0 \\ 0 & H - C E^{-1} C^T \end{bmatrix} \begin{bmatrix} I & E^{-1} C^T \\ 0 & I \end{bmatrix}.$$

The next step of the factorization consists in applying exactly the same process to $H - C E^{-1} C^T$, known as the *remaining matrix* or the *Schur complement*, which has dimension either $(n - 1) \times (n - 1)$ or $(n - 2) \times (n - 2)$. The same procedure is applied recursively, until we terminate with the factorization (6.20).

The symmetric indefinite factorization requires approximately $n^3/3$ floating-point operations—the same as the cost of the Cholesky factorization of a positive definite matrix—but to this we must add the cost of identifying the pivot blocks E and performing the permutations Π , which can be considerable. There are various strategies for determining the pivot blocks, which have an important effect on both the cost of the factorization and its numerical properties. Ideally, our strategy for choosing E at each step of the factorization procedure should be inexpensive, should lead to at most modest growth in the elements of the remaining matrix at each step of the factorization, and should not lead to excessive fill-in (that is, L should not be too much more dense than A).

A well-known strategy, due to Bunch and Parlett [31], searches the whole working matrix and identifies the largest-magnitude diagonal and largest-magnitude off-diagonal elements, denoting their respective magnitudes by ξ_{dia} and ξ_{off} . If the diagonal element whose magnitude is ξ_{dia} is selected to be a 1×1 pivot block, the element growth in the remaining matrix is bounded by the ratio $\xi_{\text{dia}}/\xi_{\text{off}}$. If this growth rate is acceptable, we choose this diagonal element to be the pivot block. Otherwise, we select the off-diagonal element whose magnitude is ξ_{off} (a_{ij} , say), and choose E to be the 2×2 submatrix that

includes this element, that is,

$$E = \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{bmatrix}.$$

This pivoting strategy of Bunch and Parlett is numerically stable and guarantees to yield a matrix L whose maximum element is bounded by 2.781. Its drawback is that the evaluation of ξ_{dia} and ξ_{off} at each iteration requires many comparisons between floating-point numbers to be performed: $O(n^3)$ in total during the overall factorization. Since each comparison costs roughly the same as an arithmetic operation, this overhead is not insignificant, and the total run time may be roughly twice as long as that of Algorithm 6.5.

The more economical pivoting strategy of Bunch and Kaufman [30] searches at most two columns of the working matrix at each stage and requires just $O(n^2)$ comparisons in total. Its details are somewhat tricky, and we refer the interested reader to the original paper or to Golub and Van Loan [115, Section 4.4] for details. Unfortunately, this algorithm can give rise to arbitrarily large elements in the lower triangular factor L , making it unsuitable for use with a modified Cholesky strategy.

The bounded Bunch–Kaufman strategy is essentially a compromise between the Bunch–Parlett and Bunch–Kaufman strategies. It monitors the sizes of elements in L , accepting the (inexpensive) Bunch–Kaufman choice of pivot block when it yields only modest element growth, but searching further for an acceptable pivot when this growth is excessive. Its total cost is usually similar to that of Bunch–Kaufman, but in the worst case it can approach the cost of Bunch–Parlett.

So far, we have ignored the effect of the choice of pivot block E on the sparsity of the final L factor. This consideration is an important one when the matrix to be factored is large and sparse, since it greatly affects both the CPU time and the amount of storage required by the algorithm. Algorithms that modify the strategies above to take account of sparsity have been proposed by Duff et al. [76], Duff and Reid [74], and Fourer and Mehrotra [93].

As for the Cholesky factorization, the efficient and numerically stable indefinite symmetric factorization algorithms discussed above can be modified to ensure that the modified factors are the factors of a positive definite matrix. The strategy is first to compute the factorization (6.20), as well as the spectral decomposition $B = Q\Lambda Q^T$, which is very inexpensive to compute because B is block diagonal (Exercise 6). Then we construct a modification matrix F such that

$$L(B + F)L^T$$

is sufficiently positive definite. Motivated by the modified spectral decomposition (6.12) we will choose a parameter $\delta > 0$ and define F to be

$$F = Q \operatorname{diag}(\tau_i) Q^T, \quad \tau_i = \begin{cases} 0, & \lambda_i \geq \delta, \\ \delta - \lambda_i, & \lambda_i < \delta, \end{cases} \quad i = 1, 2, \dots, n, \quad (6.23)$$

where λ_i are the eigenvalues of B . The matrix F is thus the modification of minimum Frobenius norm that ensures that all eigenvalues of the modified matrix $B + F$ are no less than δ . This strategy therefore modifies the factorization (6.20) as follows:

$$P(A + E)P^T = L(B + F)L^T, \quad \text{where } E = P^T L F L^T P^T;$$

note that E will not be diagonal, in general. Hence, in contrast to the modified Cholesky approach, this modification strategy changes the entire matrix A and not just its diagonal. The aim of strategy (6.23) is that the modified matrix satisfies $\lambda_{\min}(A + E) \approx \delta$ whenever the original matrix A has $\lambda_{\min}(A) < \delta$. It is not clear, however, whether it always comes close to attaining this goal.

6.4 TRUST-REGION NEWTON METHODS

Unlike line search methods, trust-region methods do not require the Hessian of the quadratic model to be positive definite. Therefore, we can use the exact Hessian $B_k = \nabla^2 f(x_k)$ directly in this model and obtain steps p_k by solving

$$\min_{p \in \mathbb{R}^n} m_k(p) \stackrel{\text{def}}{=} f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p \quad \text{s.t. } \|p\| \leq \Delta_k. \quad (6.24)$$

In Chapter 4 we described a variety of techniques for finding an approximate or accurate solution of this subproblem. Most of these techniques apply when B_k is *any* symmetric matrix, and we do not need to say much about the specific case in which $B_k = \nabla^2 f(x_k)$. We will, however, pay attention to the implementation of these trust-region Newton methods when the number of variables is large. Four techniques for solving (6.24) will be discussed: (i) the dogleg method; (ii) two-dimensional subspace minimization; (iii) accurate solution using iteration; (iv) the conjugate gradient (CG) method. For the CG-based method, we study the choice of norm defining the trust region, which can be viewed as a means of preconditioning the subproblem.

NEWTON-DOGLEG AND SUBSPACE-MINIMIZATION METHODS

If B_k is positive definite, the dogleg method described in Chapter 4 provides an approximate solution of (6.24) that is relatively inexpensive to compute and good enough to allow the method to be robust and rapidly convergent. However, since the Hessian matrix may not always be positive definite, the dogleg method is not directly applicable. To adapt it for the minimization of nonconvex functions, we can use one of the Hessian modifications described in Section 6.3 as B_k , in place of the true Hessian $\nabla^2 f(x_k)$, thus guaranteeing that we are working with a convex quadratic function in (6.24). This strategy for choosing B_k

and for finding an approximation Newton step allows us to implement the dogleg method exactly as in Chapter 4. That is, we choose the approximate solution of (6.24) to be the minimizer of the modified model function m_k in (6.24) along the dogleg path defined by

$$\tilde{p}(\tau) = \begin{cases} \tau p^u, & 0 \leq \tau \leq 1, \\ p^u + (\tau - 1)(p^b - p^u), & 1 \leq \tau \leq 2, \end{cases}, \quad (6.25)$$

where p^u is defined as in (4.12) and p^b is the unconstrained minimizer of (6.24); see Figure 4.3.

Similarly, the two-dimensional subspace minimization algorithm of Chapter 4 can also be applied, when B_k is the exact Hessian or a modification that ensures its positive definiteness, to find an approximate solution of (6.24), as

$$\min_p m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p \quad \text{s.t. } \|p\| \leq \Delta_k, \quad p \in \text{span}\{\nabla f_k, p^b\}. \quad (6.26)$$

The dogleg and two-dimensional subspace minimization approaches have the advantage that all the linear algebra computations can be performed with direct linear solvers, a feature that can be important in some applications. They are also globally convergent, as shown in Chapter 4. We hope, too, that the modified factorization strategy used to obtain B_k (and hence p^b) does not modify the exact Hessian $\nabla^2 f(x_k)$ when it is sufficiently positive definite, allowing the rapid local convergence that characterizes Newton's method to be observed.

The use of a modified factorization in the dogleg method is not completely satisfying from an intuitive viewpoint, however. A modified factorization perturbs $\nabla^2 f(x_k)$ in a somewhat random manner, giving more weight to certain directions than others, and the benefits of the trust-region approach may not be realized. In fact, the modification introduced during the factorization of the Hessian is redundant in some sense because the trust-region strategy introduces its own modification: The solution of the trust-region problem results in the factorization of the positive (semidefinite) matrix $\nabla^2 f(x_k) + \lambda I$, where λ depends on the size of the trust-region radius Δ_k .

We conclude that the dogleg method is most appropriate when the objective function is convex (that is, $\nabla^2 f(x_k)$ is always positive semidefinite). The techniques described next may be more suitable for the general case.

ACCURATE SOLUTION OF THE TRUST-REGION PROBLEM

We can also find a solution of (6.24) using Algorithm 4.4 described in Chapter 4. This algorithm requires the repeated factorization of matrices of the form $B_k + \lambda I$, for different values of λ . Practical experience indicates that between 1 and 3 such systems need to be solved, on average, at each iteration, so the cost of this approach may not be prohibitive. The resulting

trust-region algorithm is quite robust—it can be expected to converge to a minimizer, not just to a stationary point. In some large-scale situations, however, the requirement of solving more than one linear system per iteration may become onerous.

TRUST-REGION NEWTON–CG METHOD

We can approximately solve the trust-region problem (6.24) by means of the conjugate gradient method (CG) with the termination tests proposed by Steihaug; see the CG–Steihaug algorithm, Algorithm 4.3 in Chapter 4. The step computation of this Newton–CG algorithm is obtained by setting $B_k = \nabla^2 f(x_k)$ at every iteration in Algorithm 4.3. This procedure amounts to applying the CG method to the system

$$B_k p_k = -\nabla f_k \quad (6.27)$$

and stopping if (i) the size of the approximate solution exceeds the trust-region radius, (ii) the system (6.27) has been solved to a required accuracy, or (iii) if negative curvature is encountered. In the latter case we follow the direction of negative curvature to the boundary of the trust region $\|p\| \leq \Delta_k$. This is therefore the trust-region analogue of Algorithm 6.1.

Careful control of the accuracy in the inner CG iteration is crucial to keeping the cost of the Newton–CG method as low as possible. Near a well-behaved solution x^* , the trust-region constraint becomes inactive, and the Newton–CG method reduces to the inexact Newton method analyzed in Section 6.1. The results of that section, which relate the choice of forcing sequence $\{\eta_k\}$ to the rate of convergence, become relevant during the later stages of the Newton–CG method.

As discussed in the context of the line search Newton–CG method, it is not necessary to have explicit knowledge of the Hessian matrix, but we can compute products of the form $\nabla^2 f(x_k)v$ by automatic differentiation or using the finite difference approximation (6.7). Once again, the result is a *Hessian-free* method.

The trust-region Newton–CG method has a number of attractive computational and theoretical properties. First, it is globally convergent. Its first step along the direction $-\nabla f(x_k)$ identifies the Cauchy point for the subproblem (6.24) (see (4.5)), and any subsequent CG iterates only serve to improve the model value. Second, it requires no matrix factorizations, so we can exploit the sparsity structure of the Hessian $\nabla^2 f(x_k)$ without worrying about fill-in during a direct factorization. Moreover, the CG iteration—the most computationally intensive part of the algorithm—may be executed in parallel, since the key operation is a matrix–vector product. When the Hessian matrix is positive definite, the Newton–CG method approximates the pure Newton step more and more closely as the solution x^* is approached, so rapid convergence is also possible, as discussed above.

Two advantages, compared with the *line search* Newton–CG method, are that the lengths of the steps are controlled by the trust region and that directions of negative curvature are explored. Our computational experience shows that the latter is beneficial in practice, as it sometimes allows the iterates to move away from nonminimizing stationary points.

A limitation of the trust-region Newton–CG method is that it accepts *any* direction of negative curvature, even when this direction gives an insignificant reduction in the model. Consider, for example, the case where the subproblem (6.24) is

$$m(p) = 10^{-3} p_1 + 10^{-4} p_1^2 - p_2^2 \quad \text{subject to } \|p\| \leq 1,$$

where subscripts indicate elements of the vector p . The steepest descent direction at $p = 0$ is $(10^{-3}, 0)^T$, which is a direction of negative curvature for the model. Algorithm 4.3 would follow this direction to the boundary of the trust region, yielding a reduction in model function m of about 10^{-3} . A step along e_2 —also a direction of negative curvature—would yield a much greater reduction of 1.

Several remedies have been proposed. We have seen in Chapter 4 that when the Hessian $\nabla^2 f(x_k)$ contains negative eigenvalues, the search direction should have a significant component along the eigenvector corresponding to the most negative eigenvalue of $\nabla^2 f(x_k)$. This feature would allow the algorithm to move away rapidly from stationary points that are not minimizers. A variation of the Newton–CG method that overcomes this drawback uses the *Lanczos method*, rather than CG, to solve the linear system (6.27). This approach does not terminate after encountering the first direction of negative curvature, but continues in search of a direction of sufficiently negative curvature; see [177], [121]. The additional robustness in the trust-region algorithm comes at the cost of a more expensive solution of the subproblem.

PRECONDITIONING THE NEWTON–CG METHOD

The unpreconditioned CG method can be inefficient when the Hessian is ill-conditioned, and may even fail to reach the desired accuracy. Hence, it is important to introduce *preconditioning* techniques into the CG iteration. Such techniques are based on finding a nonsingular matrix D such that the eigenvalues of $D^{-T} B_k D$ have a more favorable distribution, as discussed in Chapter 5.

The use of preconditioning in the Newton–CG method requires care because the preconditioned CG iteration no longer generates iterates of increasing ℓ_2 norm; this property holds only for the unpreconditioned CG iteration (see Theorem 4.2). Thus we cannot simply terminate the preconditioned CG method as soon as the iterates reach the boundary of the trust region $\|p\| \leq \Delta_k$, since later CG iterates could return to the trust region.

However, there exists a weighted ℓ_2 norm in which the magnitudes of the preconditioned CG iterates grow monotonically. This will allow us to develop an extension of Algorithm 4.3 of Chapter 4. (Not surprisingly, the weighting of the norm depends on the preconditioner D !) Consider the subproblem

$$\min_{p \in \mathbb{R}^n} m_k(p) \stackrel{\text{def}}{=} f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p \quad \text{s.t. } \|Dp\| \leq \Delta_k. \quad (6.28)$$

By making the change of variables $\hat{p} = Dp$ and defining

$$\hat{g}_k = D^{-T} \nabla f_k, \quad \hat{B}_k = D^{-T} B_k D^{-1},$$

we can write (6.28) as

$$\min_{\hat{p} \in \mathbb{R}^n} f_k + \hat{g}_k^T \hat{p} + \frac{1}{2} \hat{p}^T \hat{B}_k \hat{p} \quad \text{s.t. } \|\hat{p}\| \leq \Delta_k,$$

which has exactly the form of (6.24). We can now apply the CG algorithm without any modification to this subproblem. Algorithm 4.3 now monitors the norm $\|\hat{p}\| = \|Dp\|$, which is the quantity that grows monotonically during the CG iteration, and terminates when $\|Dp\|$ exceeds the bound Δ_k applied in (6.28).

Many preconditioners can be used within this framework; we discuss some of them in Chapter 5. Of particular interest is *incomplete Cholesky* factorization, which has proved to be useful in a wide range of optimization problems. The incomplete Cholesky factorization of a positive definite matrix B finds a lower triangular matrix L such that

$$B = LL^T - R,$$

where the amount of fill-in in L is restricted in some way. (For instance, it is constrained to have the same sparsity structure as the lower triangular part of B , or is allowed to have a number of nonzero entries similar to that in B .) The matrix R accounts for the “inexactness” in the approximate factorization. The situation is complicated a little further by the possible indefiniteness of the Hessian $\nabla^2 f(x_k)$; we must be able to handle this indefiniteness as well as maintain the sparsity. The following algorithm combines incomplete Cholesky and a form of modified Cholesky to define a preconditioner for Newton–CG methods.

Algorithm 6.6 (Inexact Modified Cholesky).

(scale B)

Compute $T = \text{diag}(\|Be_i\|)$, where e_i is the i th coordinate vector;

$\bar{B} \leftarrow T^{-1/2} B T^{-1/2}$; $\beta \leftarrow \|\bar{B}\|$;

(compute a shift to ensure positive definiteness)

if $\min_i b_{ii} > 0$

$\alpha_0 \leftarrow 0$

else

$\alpha_0 \leftarrow \beta/2$;

end

for $k = 0, 1, 2, \dots$

Attempt to apply incomplete Cholesky algorithm to obtain

$$LL^T = \bar{B} + \alpha_k I;$$


```

if    factorization is completed successfully
      stop and return  $L$ 
else
       $\alpha_{k+1} \leftarrow \max(2\alpha_k, \beta/2)$ ;
end
end

```

We can then set the preconditioner to be $D = L$, where L is the lower triangular matrix output from Algorithm 6.6. A Newton–CG trust region method using this preconditioner is implemented in the forthcoming MINPACK-2 package under the name NMTR. The LANCELOT package also contains a Newton–CG method that makes use of slightly different preconditioning techniques.

LOCAL CONVERGENCE OF TRUST-REGION NEWTON METHODS

Since global convergence of trust-region methods that incorporate (possibly inexact) Newton steps is established above, we turn our attention now to local convergence. The key to attaining the fast rate of convergence associated with Newton's method is to show that the trust-region bound eventually does not interfere with the convergence. That is, when we reach the vicinity of a solution, the (approximate) solution of the subproblem is well inside the region defined by the trust-region constraint $\|p\| \leq \Delta_k$ and becomes closer and closer to the pure Newton step. Sequences of steps that satisfy the latter property are said to be *asymptotically exact*.

We first prove a general result that applies to any algorithm of the form of Algorithm 4.1 (see Chapter 4) that generates asymptotically exact steps whenever the true Newton step is well inside the trust region. It shows that the trust-region constraint eventually becomes inactive in algorithms with this property. The result assumes that the exact Hessian $B_k = \nabla^2 f(x_k)$ is used in (6.24) when x_k is close to a solution x^* that satisfies second-order conditions. Moreover, it assumes that the algorithm uses an approximate solution p_k of (6.24) that achieves at least the same decrease in the model function m_k as the Cauchy step. (All the methods discussed above satisfy this condition.)

Theorem 6.4.

Let f be twice Lipschitz continuously differentiable. Suppose the sequence $\{x_k\}$ converges to a point x^ that satisfies the second-order sufficient conditions (Theorem 2.4) and that for all k sufficiently large, the trust-region algorithm based on (6.24) with $B_k = \nabla^2 f(x_k)$ chooses steps p_k that achieve at least the same reduction as the Cauchy point (that is, $m_k(p_k) \leq m_k(p_k^c)$) and are asymptotically exact whenever $\|p_k^N\| \leq \frac{1}{2}\Delta_k$, that is,*

$$\|p_k - p_k^N\| = o(\|p_k^N\|). \quad (6.29)$$

Then the trust-region bound Δ_k becomes inactive for all k sufficiently large.

PROOF. We show that $\|p_k^N\| \leq \frac{1}{2}\Delta_k$ and $\|p_k\| \leq \Delta_k$, for all sufficiently large k , so the near-optimal step p_k in (6.29) will eventually always be taken.

First, we seek a lower bound on the predicted reduction $m_k(0) - m_k(p_k)$ for all sufficiently large k . We assume that k is large enough that the $o(\|p_k^N\|)$ term in (6.29) is less than $\|p_k^N\|$. When $\|p_k^N\| \leq \frac{1}{2}\Delta_k$, we then have that $\|p_k\| \leq \|p_k^N\| + o(\|p_k^N\|) \leq 2\|p_k^N\|$, while if $\|p_k^N\| > \frac{1}{2}\Delta_k$, we have $\|p_k\| \leq \Delta_k < 2\|p_k^N\|$. In both cases, then, we have

$$\|p_k\| \leq 2\|p_k^N\| \leq 2\|\nabla^2 f(x_k)^{-1}\| \|\nabla f(x_k)\|,$$

and so $\|\nabla f(x_k)\| \geq \frac{1}{2}\|p_k\|/\|\nabla^2 f(x_k)^{-1}\|$.

Because we assume that the step p_k achieves at least the same decrease as the Cauchy point, we have from the relation (4.34) that there is a constant $c_1 > 0$ such that

$$\begin{aligned} m_k(0) - m_k(p_k) &\geq c_1 \|\nabla f(x_k)\| \min \left(\Delta_k, \frac{\|\nabla f(x_k)\|}{\|B_k\|} \right) \\ &\geq c_1 \frac{\|p_k\|}{2\|\nabla^2 f(x_k)^{-1}\|} \min \left(\|p_k\|, \frac{\|p_k\|}{2\|\nabla^2 f(x_k)\| \|\nabla^2 f(x_k)^{-1}\|} \right) \\ &= c_1 \frac{\|p_k\|^2}{4\|\nabla^2 f(x_k)^{-1}\|^2 \|\nabla^2 f(x_k)\|}. \end{aligned}$$

Because $x_k \rightarrow x^*$, then by continuity of $\nabla^2 f(x)$ and positive definiteness of $\nabla^2 f(x^*)$, we have for k sufficiently large that

$$\frac{c_1}{4\|\nabla^2 f(x_k)^{-1}\|^2 \|\nabla^2 f(x_k)\|} \geq \frac{c_1}{8\|\nabla^2 f(x^*)^{-1}\|^2 \|\nabla^2 f(x^*)\|} \stackrel{\text{def}}{=} c_3,$$

and therefore

$$m_k(0) - m_k(p_k) \geq c_3 \|p_k\|^2 \tag{6.30}$$

for all sufficiently large k . By Lipschitz continuity of $\nabla^2 f(x)$, we have by the argument leading to (4.41) that

$$|(f(x_k) - f(x_k + p_k)) - (m_k(0) - m_k(p_k))| \leq \frac{L}{2} \|p_k\|^3,$$

where $L > 0$ is the Lipschitz constant for $\nabla^2 f(\cdot)$ in the neighborhood of x^* . Hence, by definition of ρ_k (see (4.4)), we have for sufficiently large k that

$$|\rho_k - 1| \leq \frac{\|p_k\|^3 (L/2)}{c_3 \|p_k\|^2} = \frac{L}{2c_3} \|p_k\| \leq \frac{L}{2c_3} \Delta_k. \tag{6.31}$$

Now, the trust-region radius can be reduced only if $\rho_k < \frac{1}{4}$ (or some other fixed number less than 1), so it is clear from (6.31) that there is a threshold $\tilde{\Delta}$ such that Δ_k cannot be reduced further once it falls below $\tilde{\Delta}$. Hence, the sequence $\{\Delta_k\}$ is bounded away from zero. Since $x_k \rightarrow x^*$, we have $\|p_k^N\| \rightarrow 0$ and therefore $\|p_k\| \rightarrow 0$ from (6.29). Hence the trust-region bound is inactive for all k sufficiently large, and our proof is complete. \square

The conditions of Theorem 6.4 are satisfied trivially if $p_k = p_k^N$. In addition, all three algorithms discussed above also satisfy the assumptions of this theorem. We state this result formally as follows.

Lemma 6.5.

Suppose that $x_k \rightarrow x^$, where x^* satisfies the second-order sufficient conditions of Theorem 2.4. Consider versions of Algorithm TR in which the inexact dogleg method (6.25) or the inexact two-dimensional subspace minimization method (6.26) with $B_k = \nabla^2 f(x_k)$ is used to obtain an approximate step p_k . Then for all sufficiently large k , the unconstrained minimum of the models (6.25) and (6.26) is simply p_k^N , so the conditions of Theorem 6.4 are satisfied.*

When the termination criterion (6.3) is used with $\eta_k \rightarrow 0$ in the Newton–CG method (along with termination when the trust-region bound is exceeded or when negative curvature is detected), then this algorithm also satisfies the conditions of Theorem 6.4.

PROOF. For the case of the dogleg and two-dimensional subspace minimization methods, the exact step p_k^N is one of the candidates for p_k —it lies inside the trust region, along the dogleg path, and inside the two-dimensional subspace. Since in fact, p_k^N is the minimizer of m_k inside the trust region for k sufficiently large (since $B_k = \nabla^2 f(x_k)$ is positive definite), it is certainly the minimizer in these more restricted domains, so we have $p_k = p_k^N$.

For the Newton–CG method, the method does not terminate by finding a negative curvature direction, because $\nabla^2 f(x_k)$ is positive definite for all k sufficiently large. Also, since the norms of the CG iterates increase with iteration number, they do not exceed p_k^N , and hence stay inside the trust region. Hence, the CG iterations can terminate only because condition (6.3) is satisfied. Hence from (6.1), (6.2), and (6.3), we have that

$$\begin{aligned} \|p_k - p_k^N\| &\leq \|\nabla^2 f(x_k)^{-1}\| \|\nabla^2 f(x_k)p_k - \nabla^2 f(x_k)p_k^N\| \\ &= \|\nabla^2 f(x_k)^{-1}\| \|r_k\| \\ &\leq \eta_k \|\nabla^2 f(x_k)^{-1}\| \|\nabla f(x_k)\| \\ &\leq \eta_k \|\nabla^2 f(x_k)^{-1}\| \|\nabla^2 f(x_k)\| \|p_k^N\|. \end{aligned}$$

Hence, the condition (6.29) is satisfied. \square

The nearly exact algorithm of Chapter 4—Algorithm 4.4—also satisfies the conditions of Theorem 6.4, since if the true Newton step p_k^N lies inside the trust region, the initial guesses

$\lambda = 0$ and $p_k = p_k^N$ will eventually satisfy any reasonable termination criteria for this method of determining the step.

Rapid convergence of all these algorithms now follows immediately from their eventual similarity to Newton's method. The methods that eventually take exact Newton steps $p_k = p_k^N$ converge quadratically. The asymptotic convergence rate of the Newton–CG algorithm with termination test (6.3) depends on the rate of convergence of the forcing sequence $\{\eta_k\}$ to zero, as described in Theorem 6.2.


NOTES AND REFERENCES

Newton methods in which the step is computed by an iterative algorithm have received much attention; see Sherman [229] and Ortega and Rheinboldt [185]. Our discussion of inexact Newton methods is based on Dembo, Eisenstat, and Steihaug [66].

For a more thorough treatment of the modified Cholesky factorization see Gill, Murray, and Wright [104] or Dennis and Schnabel [138]. The modified Cholesky factorization based on Gershgorin disk estimates is described in Schnabel and Eskow [223]. The modified indefinite factorization is from Cheng and Higham [41].

Another strategy for implementing a line search Newton method when the Hessian contains negative eigenvalues is to compute a direction of negative curvature and use it to define the search direction (see Moré and Sorensen [169] and Goldfarb [113]).

EXERCISES

 **6.1** Program a pure Newton iteration without line searches, where the search direction is computed by the CG method. Select stopping criteria such that the rate of convergence is linear, superlinear, and quadratic. Try your program on the following convex quartic function:

$$f(x) = \frac{1}{2}x^T x + 0.25\sigma(x^T A x)^2, \quad (6.32)$$


where σ is a parameter and


$$A = \begin{bmatrix} 5 & 1 & 0 & 0.5 \\ 1 & 4 & 0.5 & 0 \\ 0 & 0.5 & 3 & 0 \\ 0.5 & 0 & 0 & 2 \end{bmatrix}.$$


This is a strictly convex function that allows us to control the deviation from a quadratic by means of the parameter σ . The starting point is


$$x_1 = (\cos 70^\circ, \sin 70^\circ, \cos 70^\circ, \sin 70^\circ)^T.$$


Try $\sigma = 1$ or larger values and observe the rate of convergence of the iteration.


 **6.2** Consider the line search Newton–CG method described in Section 6.2. Use the properties of the CG method described in Chapter 5 to prove that the search Newton direction p_k is always a descent direction for f .

 **6.3** Compute the eigenvalues of the 2 diagonal blocks of (6.21), and verify that each block has a positive and diagonal eigenvalue. Then compute the eigenvalues of A and verify that its inertia is the same as that of B .

 **6.4** Describe the effect that the modified Cholesky factorization of Algorithm 6.5 would have on the Hessian $\nabla^2 f(x_k) = \text{diag}(-2, 12, 4)$.

 **6.5** Program a trust-region Newton–CG method in which the step is computed by the CG–Steihaug method (Algorithm 4.3), without preconditioning. Select the constants so that the rate of convergence is superlinear. Try it on (6.32). Then define an objective function that has negative curvature in a neighborhood of the starting point $x_0 = 0$, and observe the effect of the negative curvature steps.

 **6.6** Prove Theorem 6.2.

 **6.7** Consider a block diagonal matrix B with 1×1 and 2×2 blocks. Show that the eigenvalues and eigenvectors of B can be obtained by computing the spectral decomposition of each diagonal block separately.

CHAPTER 7

