NORTHWESTERN UNIVERSITY
Department of Electrical Engineering
and Computer Science

LARGE SCALE UNCONSTRAINED OPTIMIZATION

by

Jorge Nocedal[1]

June 23, 1996

ABSTRACT

This paper reviews advances in Newton, quasi-Newton and conjugate gradient methods for large scale optimization. It also describes several packages developed during the last ten years, and illustrates their performance on some practical problems. Much attention is given to the concept of partial separability which is gaining importance with the arrival of automatic differentiation tools and of optimization software that fully exploits its properties.

Categories and Subject Descriptors: G.1.6 [**Numerical Analysis**]: Optimization – *gradient methods*; G.4 [**Mathematics of Computing**]: Mathematical Software.

General Terms: Algorithms

Additional Key Words and Phrases: variable metric method, Newton's method, large scale optimization, nonlinear optimization, limited memory methods, quasi-Newton methods, nonlinear conjugate gradient methods.

[1] Department of Electrical and Computer Engineering, Northwestern University, Evanston Il 60208. This work was supported by National Science Foundation Grant CCR-9400881, and by Department of Energy Grant DE-FG02-87ER25047.

1 Introduction.

This survey focuses on recent developments in large scale unconstrained optimization. I will not discuss advances in methods for small and medium scale problems because fairly comprehensive reviews of this work are given in [53] and [24]. I should stress, however, that small scale optimization remains an active area of research, and that advances in this field often translate into new algorithms for large problems.

The problem under consideration is

$$\min f(x), \tag{1.1}$$

where f is a smooth function of n variables. We assume that n is large, say $n > 500$, and we denote the gradient of f by g.

An important recent development has been the appearance of effective tools for automatically computing derivatives and for detecting partially separable structures. These programs are already having a significant impact in the practice of optimization and in the design of algorithms, and their influence is certain to grow with time. Automatic differentiation provides an excellent alternative for finite differences, which can be unreliable, and for hand-coded derivatives, which are error prone and labor intensive. The automatic detection of partially separable structure has only recently become available and its full impact is yet to be seen. It has the potential of making automatic differentiation practical for many large problems, and it may also popularize partially separable quasi-Newton methods.

This paper is organized around the three main classes of algorithms for unconstrained optimization: Newton, quasi-Newton and conjugate gradients. To set the stage for the description and analysis of algorithms, we begin with a discussion of some problem structures that arise in many important areas of application. We devote much attention to this topic because understanding the characteristics of the objective function is crucial in large scale optimization.

2 Problem Structure.

A well-known type of structure is that of sparsity in the Hessian matrix $\nabla^2 f$. Functions with sparse Hessians occur often in practice, and algorithms that exploit it have been developed since the 1970s [16], [14]. Sparse finite difference techniques [20],[15] have played a crucial role in making these methods economical and practical.

But many problems do not possess a sparse Hessian, and it is useful to divide these into two broad categories: problems that have some kind of structure, such as partial separability, and problems that do not possess any useful structure. In this section we will give a close look at partially separable problems because they are not as well understood as they should, and because significant advances in solving them have been made in the last ten years. We will see that partial separability leads to economical problem representation, efficient automatic differentiation and effective quasi-Newton updating. An interesting question is whether there are other types of structures that are as amenable to optimization as partial separability.

2.1 Partial Separability

In many large problems the objective function can be written as a sum of much simpler functions; such an objective function is called partially separable. All functions with sparse Hessians can be shown to have this property, but so do many functions whose Hessian is not sparse. Some other problems cannot be written as the sum of simple functions, but still possess a structure similar to partial separability called group partial separability [17]. The goal of this section is to explain why this kind of structure is important in optimization.

The simplest form of partial separability arises when the objective function can be written as

$$f(x) = \sum_{i=1}^{ne} f_i(x), \tag{2.1}$$

where each of the *element functions* f_i depends only on a few variables.

An example is the minimum surface problem which is described in many calculus textbooks. We wish to find the surface of minimum area that interpolates a given continuous function on the boundary of the unit square. By discretizing this problem we obtain an objective function of the form (2.1), where the element functions are defined as

$$f_i(x) = \frac{1}{m^2}\left[1 + \frac{m^2}{2}[(x_j - x_{j+m+1})^2 + (x_{j+1} - x_{j+m})^2]\right]^{\frac{1}{2}}. \tag{2.2}$$

Thus each f_i, which is formally a function of all the variables $x_1, ..., x_n$, depends only on 4 variables, which are called the *element variables*. In (2.2) m is a constant that determines the fineness of the discretization; the number of element functions ne and the total number of variables n are both equal to m^2. (The precise relationship between the indices i and j in (2.2) is not important in the following discussion; it suffices to say that j is determined by i and m.) The gradient of f_i with respect to all the variables in the problem is

$$\nabla f_i(x) = \frac{1}{2m^2} f_i^{-1}(x) \begin{bmatrix} 0 \\ \vdots \\ x_j - x_{j+m+1} \\ x_{j+1} - x_{j+m} \\ \vdots \\ -x_{j+1} + x_{j+m} \\ -x_j + x_{j+m+1} \\ \vdots \\ 0 \end{bmatrix}. \tag{2.3}$$

We note that, of the four non-zero components, two are negatives of each other. The Hessian of the element function f_i has the sparsity pattern

$$\begin{bmatrix} & & & & \\ & * & * & * & * \\ & * & * & * & * \\ & & & & \\ & & & & \\ & * & * & * & * \\ & * & * & * & * \\ & & & & \end{bmatrix}. \tag{2.4}$$

A close examination shows that some of the nonzero entries differ only in sign, and that only three different magnitudes are represented. Moreover the 4×4 matrix formed by the non-zero elements happens to be singular.

The fact that repeated information is contained in the gradient and Hessian suggests that there is a compact representation that avoids these redundancies. The key observation is that f_i is *invariant* in the subspace

$$N_i = \{w \in R^n : w_j = w_{j+m+1} \quad \text{and} \quad w_{j+1} = w_{j+m}\}, \tag{2.5}$$

which means that for any x and for any $w \in N_i$ we have that $f_i(x + w) = f_i(x)$. In other words, any move along a direction in N_i does not change the value of the

function, and it is not useful to try to gather curvature information about f_i along N_i. Since dimension$(N_i) = n - 2$, we seek a compact representation of f_i involving only two variables.

To accomplish this we define the *internal variables* u_j and u_{j+1} by

$$u_j = x_j - x_{j+m+1} \qquad u_{j+1} = x_{j+1} - x_{j+m}, \tag{2.6}$$

and the *internal function* ϕ_i,

$$\phi_i(u_j, u_{j+1}) = \frac{1}{m^2}\left[1 + \frac{m^2}{2}(u_j^2 + u_{j+1}^2)\right]^{\frac{1}{2}}. \tag{2.7}$$

We have achieved our goal of finding a minimal representation of f_i since the gradient of ϕ has only two components and $\nabla^2 f_i$ has only 3 distinct elements.

It is useful to introduce the $2 \times n$ matrix U_i, which has zero elements except for

$$U_{1,j} = 1, \quad U_{1,j+m+1} = -1, \quad U_{2,j+1} = 1, \quad U_{2,j+m} = -1.$$

We can now write the objective function as

$$f(x) = \sum_{i=1}^{ne} \phi_i(U_i x). \tag{2.8}$$

The gradient and Hessian of f are given by

$$g(x) = \sum_{i=1}^{ne} U_i^T \nabla \phi_i(U_i x) \quad \text{and} \quad \nabla^2 f(x) = \sum_{i=1}^{ne} U_i^T \nabla^2 \phi_i(U_i x) U_i, \tag{2.9}$$

which clearly exhibits the structure of the objective function. In §4.1 we describe a quasi-Newton method that updates approximations to each of the Hessians $\nabla^2 \phi_i$. Since these are 2×2 matrices a good approximation can often be obtained after only a few iterations. We will also see that this representation of the gradient suggests efficient automatic differentiation techniques.

We now generalize the minimum surface problem and give the following definition.

> A function f is said to be *partially separable* if it is the sum of element functions, $f(x) = \sum_{i=1}^{ne} f_i(x)$, each of which has a nontrivial invariant subspace. This means that f can also be written in the form (2.8), where the matrices U_i have dimension $n_i \times n$, with $n_i < n$.

In many practical problems the n_i are very small (say 2 or 4), and are independent of the number of variables n which can be in the thousands or more. To take advantage of partial separability we define the internal variables (and thus U_i) so that a change in any one of these variables causes a change in the element function f_i. It would be wasteful to define an internal variable that lies in the subspace N_i, since we know in advance that the derivatives of f_i with respect to this variable will be zero; moreover this would be harmful to the quasi-Newton method that exploits partial separability described in §4.1. Thus the number of internal variables equals the dimension of $N_i^{\perp}$, or

$$n - \text{dimension}(N_i).$$

Sometimes the invariant subspace N_i is easy to find, as in the minimum surface example, but this is not always the case. Software tools for automatically detecting partially separable structures are currently under development [29].

2.1.1 Sparsity vs Partial Separability

The concept of partial separability is more general than the notion of sparsity. It is shown in [34] that every twice continuously differentiable function $f : R^n \to R$ with a sparse Hessian is partially separable. But the converse is not true.

Consider the element function [17]

$$f_i(x) = (x_1 + ... + x_n)^2, \tag{2.10}$$

whose gradient and Hessian are dense. Since the invariant subspace of (2.10) is the set

$$N_i = \{w \in R^n \mid e^T w = 0\}, \tag{2.11}$$

where $e = (1, 1, \ldots, 1)^T$, and since the dimension of N_i is $n - 1$, we see that f_i can be considered as a function of only one variable. Thus we can write it in the form (2.8), where

$$U = [1, ..., 1], \quad \text{and} \quad \phi(u) = u^2.$$

This is an example of a function with a dense Hessian, but a very large invariant subspace and very simple structure.

A similar, but more realistic example, is a protein folding problem that has received much attention from the optimization community [50], [38]. The objective is to minimize the energy of a configuration of atoms. If the positions of the atoms are expressed in Cartesian coordinates, then the element functions depend only on the differences of the coordinates of pairs of atoms, $p_i - p_j$, and every row of U_i contains only two nonzeros. Typical values for n_i are 3, 6 or 9. This protein folding problem is thus partially separable but one can show that its Hessian is completely dense. The components of the Hessian that correspond to atoms that end up being widely separated will be very small (and could be set to zero), but the locations of these small entries is not known beforehand.

2.1.2 Group Partial Separability

Partial separability is an important concept, but it is not quite as general as we would like. Consider a nonlinear least squares problem of the form

$$f(x) = \sum_{k=1}^{l} (f_k(x) + f_{k+1}(x) + c)^2, \tag{2.12}$$

where the functions f_j are partially separable and where c is a constant. The definition of partial separability given above forces us to regard the whole k-th term in the summation as the k-th element function. However there is clearly a lot of structure inside that term — it contains two element functions, grouped together and squared. We can extend the definition of partial separability slightly to make better use of this type of structure.

We say that a function $f : R^n \to R$ is *group partially separable* if it can be written in the form

$$f(x) = \sum_{k=1}^{l} \gamma_k(h_k(x)) \tag{2.13}$$

where γ_k is twice continuously differentiable function on the range of $h_k(\cdot)$, and where each h_k is a partially separable function from R^n to R.

We can use the form of f to find compact representations of the derivatives. By using the chain rule we have

$$\nabla[\gamma_k(h_k(x))] = \gamma_k'(h_k(x))\nabla h_k(x), \tag{2.14}$$

$$\nabla^2[\gamma_k(h_k(x))] = \gamma_k''(h_k(x))\nabla h_k(x)\nabla h_k(x)^T + \gamma_k'(h_k(x))\nabla^2 h_k(x). \tag{2.15}$$

We already know how to represent the derivatives of the partially separable function h_k (that is, ∇h_k and $\nabla^2 h_k$) in compact form. To get a compact representation of the derivatives of $\gamma_k(h_k(x))$, we simply have to include the two scalar quantities γ_k' and γ_k'', both evaluated at $h_k(x)$.

Group partial separability is a very general concept since it applies directly to nonlinear least squares problems and to penalty and merit functions arising in constrained optimization. The LANCELOT package [17] is designed to fully exploit its structure.

2.1.3 Automatic Differentiation of Partially Separable Functions

Automatic differentiation is based on the observation that any function f that can be evaluated by a computer program is executed as a sequence of elementary operations (such as additions, multiplications and trigonometric functions). By systematically applying the chain rule to the composition of these elementary functions, the gradient ∇f can be computed to machine accuracy [35]. There are two basic strategies for applying the chain rule: the *forward* and *reverse modes* of differentiation.

In the forward mode one proceeds in the direction determined by the evaluation of the function; as we evaluate f, we compute the derivatives of all the intermediate quantities with respect to the variables x of the problem. In the reverse mode one first evaluates the function; when this is completed we move backwards, computing the derivates of f with respect to the intermediate quantities arising in the evaluation of f, until we obtain the derivatives of f with respect to the problem variables x.

Various automatic differentiation codes have been developed (see [5] for references), and have proved to be effective in practice. I will now discuss two recent proposals on how to take advantage of partial separability to make automatic differentiation more efficient.

The approach described in [5] concerns the computation of the gradient g. It begins with the simple representation (2.1), and associates with it the vector function

$$F(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_{ne}(x) \end{pmatrix}.$$

Since the Jacobian $F'(x)$ is given by $F'(x)^T = (\nabla f_1(x), ..., \nabla f_{ne}(x))$ we have from (2.1) that

$$g(x) = F'(x)^T e, \tag{2.16}$$

where $e = (1, 1, ..., 1)$. Therefore the gradient of a partially separable function could be computed via (2.16) if we had the associated Jacobian $F'(x)$. It would seem that the latter is expensive to compute, but since each f_i depends only on a few variables, $F'(x)$ is sparse. By analyzing the sparsity structure of $F'(x)$ it is possible to find a set of structurally orthogonal columns [20], [15], i.e. columns that do not have a nonzero entry in the same row. This information is provided to the automatic differentiation code, which is then able to compute $F'(x)$ in a compressed form that economizes storage and computation [5]. The gradient $g(x)$ is then computed by means of (2.16).

Another recent proposal [29] we wish to discuss concerns the computation of the Hessian matrix. It is based on the fact that Hessian-vector products can easily be computed by automatic differentiation, without the need to calculate Hessians [35]. To compute $\nabla^2 f(x_k)d$ automatically is conceptually straightforward. We can simply apply backward automatic differentiation to compute the gradient of $d^T \nabla f(x)$, considering d constant.

The complete Hessian $\nabla^2 f(x)$ can therefore be obtained column by column by applying automatic differentiation to compute the n products $\nabla^2 f(x)e_i$. But it

is preferable to consider the representation (2.8) and compute the small internal Hessians $\nabla^2\phi_i$ explicitly. This is done, for each element function ϕ_i, by means of a few products $\nabla^2\phi_i e_j$. The total Hessian is obtained via (2.9), which is a sum of outer products [29].

An automatic procedure for detecting the partially separable decomposition (2.8) is also given in [29]. It consists of analyzing the composition of elementary functions forming f, and finding the transformations U_i and the internal functions ϕ_i. This procedure finds one particular partially separable representation of f (the "finest"), and it is too early to tell if it will be of wide applicability. If successful, it could popularize partially separable optimization methods.

We conclude this section on partial separability by noting that sparsity plays a major role in many existing codes for large scale optimization. Partial separability is now being proposed as a structure of wider applicability that could supersede sparsity, but as we have seen, the approach (2.16) based on the compressed Jacobian matrices makes use of partial separability *and* sparsity. It is difficult to predict what the relative importance of these concepts will be ten years from now.

3 Newton's Method

This is the most powerful algorithm for solving large nonlinear optimization problems. It normally requires the fewest number of function evaluations, is very good at handling ill-conditioning, and is capable of giving the most accurate answers. It may not always require the least computing time – this depends on the characteristics of the problem and the implementation of the Newton iteration – but it represents the most reliable method for solving large problems.

As is well known, to be able to apply Newton's method, the gradient g and Hessian $\nabla^2 f$ must be available. The gradient could be computed analytically or by automatic differentiation, and there are several options for providing the Hessian matrix: it could be supplied in analytic form, could be approximated by sparse finite differences [20], [15], or could be computed by automatic differentiation techniques. Future problem-solving environments will allow the user to choose from these alternatives.

The newest implementations of Newton's method are simple and elegant, and deal well with the case when the Hessian matrix is indefinite. They construct a quadratic model of the objective function and define a trust region over which the model is considered to be reliable. The model is approximately minimized over the trust region by means of an ingenious adaptation of the conjugate gradient (CG) algorithm. Careful attention is given to the issue of preconditioning which is crucial for solving ill-conditioned problems. The routine SBMIN of the LANCELOT package, and the code TRCG that will be part of the Minpack-2 package follow this approach, which we now describe in some detail.

Let us denote the Hessian matrix by B, i.e. $\nabla^2 f(x_k) = B_k$. At the iterate x_k we formulate the trust region subproblem

$$\min \quad \phi(p) = g_k^T p + \frac{1}{2} p^T B_k p \tag{3.1}$$

$$\text{subject to} \qquad \|p\|_2 \leq \Delta_k, \tag{3.2}$$

where the trust region radius Δ_k is updated at every iteration. Obtaining the exact solution of this subproblem is expensive when n is large – even when B_k is positive definite. It is more efficient to compute an approximate solution p that is relatively inexpensive (at least in the early stages of the algorithm) and gives a fast rate of convergence. This is done by first ignoring the trust region constraint and attempting

to minimize the model (3.1) by means of the CG method. Thus we apply CG to the symmetric linear system

$$B_k p = -g_k, \tag{3.3}$$

starting with the initial guess $p_0 = 0$. The key points are how to take into account the trust region constraint and how to avoid failure of the CG iteration when B_k is not positive definite. For this purpose three different stopping steps are used, which we discuss later on. Once the approximate solution p has been obtained, we test whether $f(x_k + p_k)$ is sufficiently less than $f(x_k)$. If so, we define the new iterate as $x_k + p_k$, and update the trust region Δ_k according to how well the model ϕ estimates the nonlinear objective function f. On the other hand, if sufficient reduction in f is not obtained, we reduce the trust region radius Δ_k and find a new approximate solution to (3.1)-(3.2).

We now focus on the case when the Hessian matrix B_k is not positive definite. The first stopping test stipulates that the CG iteration will be terminated as soon as negative curvature is detected. When this occurs, the direction of negative curvature is followed to the boundary of the trust region, and the resulting step is returned as the approximate solution. The second test monitors the length of the approximate solutions $\{p_j\}$ generated by the CG iteration and terminates when one of them exceeds the trust region radius. A third test is included to end the CG iteration if the linear system (3.3) has been solved to the required precision.

This strategy for computing the search direction of the trust region Newton method is summarized in Algorithm I. The iterates generated by the CG method are denoted by $\{p_j\}$, whereas the conjugate directions computed by the CG iteration are $\{d_j\}$. The differences between this algorithm and standard conjugate gradient are the two extra stopping conditions formulated as the first two IF statements within the LOOP. When negative curvature is encountered or when p_{j+1} violates the trust region constraint, a final estimate p is found by intersecting the current search direction with the the trust region boundary.

Algorithm I Approximate computation of the Newton step by CG.

Constant $\epsilon > 0$ is given
Start with $p_0 = 0$, $r_0 = g_k$, and $d_0 = -r_0$
LOOP, starting with $j = 0$
  IF $d_j^T B_k d_j \le 0$
    THEN find τ so that $p = p_j + \tau d_j$ minimizes $\phi(p)$
      and satisfies $\|p\|_2 = \Delta$, and RETURN p
  $\alpha_j = r_j^T r_j \,/\, d_j^T B_k d_j$
  $p_{j+1} = p_j + \alpha_j d_j$
  IF $\|p_{j+1}\|_2 \ge \Delta$
    THEN find $\tau \ge 0$ so that $\|p_j + \tau d_j\|_2 = \Delta$,
      and RETURN $p = p_j + \tau d_j$
  $r_{j+1} = r_j + \alpha_j B_k d_j$
  IF $\|r_{j+1}\|_2 \,/\, \|r_0\|_2 < \epsilon$ THEN RETURN $p = p_{j+1}$
  $\beta_{j+1} = r_{j+1}^T r_{j+1} \,/\, r_j^T r_j$
  $d_{j+1} = r_{j+1} + \beta_{j+1} d_j$
CONTINUE, after incrementing j

The third IF statement ensures that the CG iteration is terminated when the residual is sufficiently small compared to the initial residual – which equals the current gradient g_k of the objective function. Therefore as the Newton algorithm approaches the solution and g_k converges to zero, the termination test becomes ever

more stringent, ensuring that the rate of convergence is fast [21]. Indeed, as the iterates approach the solution, the Hessian B_k will become positive definite, and one can show [25] that the standard strategies for updating the trust region radius [22] ensure that the trust region constraint becomes inactive. Thus, asymptotically, this method reduces to a pure truncated Newton method with unit steplengths.

Note that first estimate p_1 generated by the inner CG iteration is given by

$$p_1 = \alpha_0 d_0 = -\alpha_0 g_k,$$

where α_0 is the steplength that minimizes the quadratic model ϕ along the steepest descent direction $-g_k$ at the current iterate x_k. This is also called the Cauchy step, and guarantees that the trust region algorithm is globally convergent [57]. Subsequent inner CG iterations reduce ϕ and improve the quality of the search direction. But no attempt is made to try ensure that the iterates converge only to solution points where the Hessian is positive definite.

It is important that the first estimate in the inner CG iteration be $p_0 = 0$, for in this case one can show [61] that each estimate is longer than the previous one. To be more precise, we have

$$0 = \|p_0\|_2 < \cdots < \|p_j\|_2 < \|p_{j+1}\|_2 < \cdots < \|p\|_2 \leq \Delta.$$

This property shows that it is acceptable to stop iterating as soon as the trust region boundary is reached because all subsequent estimates will lie outside the trust region.

When the Hessian matrix B is positive definite, this approach is similar to the dogleg method [56] because the estimates generated by the inner CG iteration sweep out points that move on some interpolating path from the Cauchy step p_1 to the Newton step $p_N = -B_k^{-1} g_k$.

3.1 Preconditioning

An important question is how to precondition the CG iteration, i.e. how to find a nonsingular matrix D such that the eigenvalues of $D^{-T} B_k D^{-1}$ are clustered. The LANCELOT package [17] has the excellent feature of providing a suite of built-in preconditioners with which the user can experiment. The default is an 11-band modified Cholesky factorization, but many other choices can easily be activated. The choice of preconditioner has a marked effect on performance, but it is difficult to know in advance what the best choice is for a particular problem. This is a complex issue; for example there is a trade-off between the quality of the preconditioner and the computational work involved.

In LANCELOT the preconditioned conjugate gradient method is applied to (3.3), the length of the estimates $\|p_j\|$ is monitored, and the CG iteration is terminated if $\|p_j\| \geq \Delta_k$. However, this is not totally consistent with the ideas embodied in Algorithm I because when preconditioning is used the quantity that grows monotonically is $\|Dp_j\|$ and not $\|p_j\|$. Therefore it is possible for LANCELOT to terminate the inner CG iteration even though a subsequent iterate would fall inside the trust region and give a lower value of the model ϕ.

The routine TRCG from Minpack-2 takes a different approach, in that preconditioning is seen as a scaling of the trust region. Let D be any nonsingular matrix and consider the subproblem

$$\min_{p \in \mathbf{R}^n} \quad \phi(p) = g_k^T p + \frac{1}{2} p^T B_k p \tag{3.4}$$

$$\text{subject to} \qquad \|Dp\|_2 \leq \Delta_k. \tag{3.5}$$

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

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

we obtain a problem of the form (3.1)-(3.2) to which Algorithm I can be applied directly; the answer is then transformed back into the original variables p. The length of $\|\hat{p}\| = \|Dp\|$ is monitored, and this is the quantity that grows monotonically. Therefore, unlike the implementation in LANCELOT, once the CG iteration generates an estimate that leaves the trust region, it will never return. It is not clear how this difference in the handling of the trust region affects performance, and this question deserves some investigation.

Minpack-2 uses an incomplete (and possibly modified) Cholesky factorization as a preconditioner because computational experience in linear algebra has shown that this type of preconditioner can be effective for a large class of matrices. The incomplete Cholesky factorization of a positive definite matrix B_k finds a lower triangular factor L such that $B_k = LL^T - R$, where L reflects the sparsity of B_k. The scaling factor used in (3.5) is set to $D = L$. Since in Newton's method B_k may not be positive definite, it may be necessary to modify it so that the Cholesky factorization can be performed. A typical strategy for doing this is described below. It begins by scaling the matrix B, since numerical experience indicates that this can be beneficial.

Incomplete and Modified Cholesky Factorization.
The symmetric matrix B is given.

1. (Scale B.) Let $T = \text{diag}(\|Be_i\|)$, where e_i is the i-th unit vector. Define $\bar{B} = T^{-\frac{1}{2}} B T^{-\frac{1}{2}}$, and $\beta = \|\bar{B}\|$.
2. (Compute a shift to attempt to ensure positive definiteness.) If $\min(B_{i,i}) > 0$, set $\alpha_0 = 0$; otherwise set $\alpha_0 = \beta/2$.
3. Loop starting with $k = 0$.
   - Attempt to compute the incomplete Cholesky factorization [41] of $\bar{B} + \alpha_k I$. If the factorization is completed successfully exit and return L; otherwise set $\alpha_{k+1} = \max(2\alpha_k, \frac{1}{2}\beta)$, and continue loop.

We should note that scaling alters the incomplete Cholesky factorization, and that several other choices for the scaling matrix T can be used.

Following are some results obtained by Bouaricha and Moré [7] on two problems from the Minpack-2 collection [4]. GL2 is a problem arising in the modeling of superconducting materials, and MSA is a minimum surface problem.

Table 1. Performance of a Preconditioned Newton Method

Problem	n	iter	feval	time
GL2	2500	6	12	13.0
GL2	10,000	6	10	63.4
GL2	40,000	8	12	537.7
MSA	2500	6	7	2.8
MSA	10,000	6	7	2.8
MSA	40,000	10	14	91.1

These are very good results in that the number of iterations stays nearly constant as the dimension of the problem increases, and computing time grows only a little faster than linearly. The results also compare very favorably with those obtained with the limited memory BFGS code described in §4.2, which has considerable difficulty solving the problems as the dimension increases. I do not know, however, if this kind of performance is typical of the Minpack-2 code, and more computational experience is necessary.

3.2 Current Research and Open Questions

The Newton method in LANCELOT takes advantage of partial separability by computing the Hessians of the internal functions ϕ_i defined in (2.8). This can give significant savings in storage in some problems, but the computational overhead required by this structured representation of the function can sometimes be onerous. Similarly, in the Minpack-2 code the preconditioned CG iteration can sometimes be quite expensive. Therefore, even though these Newton codes are highly successful, it is interesting to ask if the information generated in the course of the step computation can be used more thoroughly.

In the *Iterated Subspace Minimization Method* [18] a search direction is first computed using Algorithm I. However, instead of accepting this step, one selects a few of the estimates $\{p_j\}$ generated during the inner CG iteration, in order to define a subspace S_k over which the nonlinear objective function f is minimized further. Since the subspace S_k is chosen to be of small dimension, this is a low-dimensional nonlinear optimization problem that is solved by means of the BFGS method (using second derivatives would require projections onto the subspace S_k which can be unnecessarily costly). The Iterated Subspace Minimization method therefore requires several evaluations of the objective function to produce a new iterate, and one can expect it to be most effective when the function is not too costly to evaluate. This approach is based on the assumption that CG determines important directions over which it is worth exploring the nonlinear objective function, and that the benefit of this exploration outweighs the cost of additional function evaluations.

Numerical tests with the Iterated Subspace Minimization method show some promise, but a clear improvement in performance over the standard Newton method has not been observed. There is room, however, for further research. For example, there are many possible choices for the subspace S_k, and this selection could be crucial to the efficiency of the method. One could also try to develop an automatic mechanism that determines when the exploration of the subspace S_k is to be performed.

There are several open questions concerning the implementation of Algorithm I. The first is the use of the residual in the stopping test

$$\|r_{j+1}\| \leq \epsilon \|r_0\|, \tag{3.6}$$

that determines that the linear system (3.3) has been approximately solved. On ill-conditioned problems, the residual can oscillate greatly during the course of the CG iteration, and only drop sharply at the end. Some researchers (see e.g. [48]) propose stopping tests based on the reduction of the model ϕ, and others use an angle test [42], but no systematic comparison of these alternatives has been undertaken in the setting of nonlinear optimization.

Assuming that a residual-based stopping test is used, (3.6) may not be its best implementation. It is known [39] that the test (3.6) may be too stringent when ϵ is a small and when the approximate solution p of (3.3) is large compared with r_0. It may be preferable to use $\|r_k\| \leq \epsilon(\|B_k\|\|p\| + \|r_0\|)$. The effects of rounding errors can also be important. Since r_k is not computed directly, but recurred, $\|r_k\|$ can differ from its true value by several orders of magnitude. A point in favor of a residual-based stopping test is that it allows us to easily control the rate of convergence of the optimization algorithm [21].

Another open question concerning the implementation of Algorithm I is the use of the conjugate gradient method. Since B_k can be indefinite, the CG iteration can become unstable [55], and it is not clear that the direction given by Algorithm I is always of good quality. An alternative [46] is to use the Lanczos iteration and continue past the point where negative curvature is first detected. It is also possible to alter the CG iteration, after encountering negative curvature, so that it can

continue exploring other subspaces [3]. I do not know if significant improvements in performance can be realized with these proposals, but this question is important and deserves careful investigation.

Perhaps more important than any of these issues is the choice of preconditioner. The Hessian matrix B_k can change drastically during the course of the optimization iteration, and to use the same preconditioning strategy throughout the run is questionable. The idea of having a dynamic preconditioner is appealing but, to the best of my knowledge, has not been studied in the context of large scale nonlinear optimization.

3.3 Hessian Free Newton Method

This is a modification of Newton's method that allows us to solve problems where the Hessian $B_k \equiv \nabla^2 f(x_k)$ is not available. It is based on the observation that the conjugate gradient iteration in Algorithm I only needs the product of $\nabla^2 f(x_k)$ with certain displacement vectors d_j, and does not require the Hessian matrix itself. These products can be approximated by finite differences,

$$\nabla^2 f(x_k)d \approx \frac{g(x_k + \epsilon d) - g(x_k)}{\epsilon}, \tag{3.7}$$

where ϵ is a small differencing parameter. Since each iteration of the conjugate gradient method performs one product $\nabla^2 f(x_k)d_j$, this approach requires a new evaluation of the gradient g of the objective function at every CG inner iteration. A method that uses (3.7) is called a discrete Newton method and has received considerable attention [54],[48],[47],[59].

The finite-difference (3.7) is unreliable and may deteriorate the performance of the Hessian free Newton method. An attractive alternative is to compute the product $\nabla^2 f(x_k)d$ by automatic differentiation [35], as discussed in §2.1.3. This has the important advantage of being accurate (in general, as accurate as the computation of the function).

In general, however, computing the Hessian-vector product by automatic differentiation will be as expensive (or more) as finite differences in terms of computing time. Therefore the Hessian-free Newton method can be made more reliable by automatic differentiation, but its overall cost remains high. Its efficiency will be highly dependent on the termination test used in the inner CG iteration. If only one CG iteration is performed, the step computation is inexpensive but the method reduces to steepest descent, whereas high accuracy in the CG iteration results in an approximation to Newton's method but a high computational cost [49]. Thus the proper termination of the inner CG iteration is even more delicate in this context than in the case when the Hessian is available.

An important open question is how to develop general purpose preconditioners for Hessian free Newton methods. In some applications it is useful to compute part of the Hessian (or a modification of it) and use it as a preconditioner. But in other applications this is not practical or effective, and the design of simple preconditioners that require no user intervention would greatly enhance the value of Hessian free Newton methods.

4 Quasi-Newton Methods

There have been various attempts to extend quasi-Newton updating to the large-scale case, and two of them have proved to be very successful in different contexts. The first idea consists of exploiting the structure of partially separable functions by updating approximations to the Hessians of the internal functions (2.8). This gives rise to a powerful algorithm of wide applicability, and whose only drawback is

the need to fully specify a partially separable representation of the function. The second approach is that of limited memory updating in which only a few vectors are kept to represent the quasi-Newton approximation to the Hessian. Limited memory methods require minimal input from the user and are best suited for problems that do not possess a structure that can be exploited economically. They are not as robust and as rapidly convergent as partially separable quasi-Newton methods, but are probably much more widely used.

An approach that has not yet proved to be successful is that of designing sparse quasi-Newton updating formulae. But some new ideas that deserve attention have recently been proposed.

4.1 Partially Separable Quasi-Newton Methods

Suppose that we know how to break a function f down into partially separable form (2.8), i.e. that we have identified ne, the transformations U_i and ϕ_i. Rather than computing the Hessians of the internal functions ϕ_i, we can store and update quasi-Newton approximations B_i to each individual $\nabla^2\phi_i$. We then estimate the entire Hessian $\nabla^2 f(x)$ (see (2.9)) by

$$B = \sum_{i=1}^{ne} U_i^T B_i U_i. \tag{4.1}$$

The $n \times n$ matrix B can then be used in Algorithm I, resulting in a partially separable quasi-Newton method using trust regions.

As for any quasi-Newton method, the approximations B_i are updated by requiring them to satisfy the following secant equation for each element:

$$B_i s_{[i]} = y_{[i]}. \tag{4.2}$$

Here

$$s_{[i]} = u^+_{[i]} - u_{[i]} \tag{4.3}$$

is the change in the internal variables corresponding to the i-th element function, and

$$y_{[i]} = \nabla\phi_i(u^+_{[i]}) - \nabla\phi(u_{[i]}) \tag{4.4}$$

is the corresponding change in gradients, where u^+ indicates the most recent iterate and u the previous one.

The success of this element-by-element updating technique can be understood by returning to the minimum surface problem (2.2). In this case, the functions ϕ_i depend only on two internal variables, so that each Hessian approximation is 2×2. After just a few iterations, we will have sampled enough directions $s_{[i]}$ to make B_i an accurate approximation to $\nabla^2\phi_i$. Hence (4.1) will be a very good approximation to $\nabla^2 f(x)$.

It is interesting to contrast this with a quasi-Newton method that ignores the partially separable structure of the objective function. This method will attempt to estimate the total average curvature (i.e. the sum of the individual curvatures) by constructing an $n \times n$ matrix. When the number of variables is large, after k iterations with $k << n$, this quasi-Newton matrix will not resemble the true Hessian well, and will not make very rapid progress towards the solution.

The partially separable quasi-Newton updating cannot always be performed by means of the BFGS formula because there is no guarantee that the curvature condition $s^T_{[i]} y_{[i]} > 0$ will be satisfied; this condition is needed to ensure that the BFGS approximation is well defined [22]. Even if a line search is used to guarantee that the total step $s_k = x_{k+1} - x_k$ satisfies the curvature condition, this would not imply

that the changes of the individual components $u_{[i]}$ would satisfy the it. In fact one of the element Hessians $\nabla^2\phi_i$ could be concave in a region around x_k.

One way to overcome this obstacle is to use SR1 [19] to update each of the element Hessians, with simple precautions to ensure that it is always well-defined. Or one could start with the BFGS update formula and, if at some stage the curvature condition for a particular element is not satisfied, then switch to the SR1 formula and use it until termination [63]. This last strategy is used in LANCELOT, which is designed to take full advantage of partial separability. Computational experience on the CUTE [6] test problem collection suggests that the partially separable quasi-Newton method implemented in LANCELOT is nearly as effective as Newton's method.

4.2 Limited Memory Methods

Various limited memory methods have been proposed; some combine conjugate gradient and quasi-Newton steps, and others are very closely related to quasi-Newton methods. The simplest implementation, and perhaps the most efficient, is the limited memory BFGS method (L-BFGS) [52],[44],[31]. It is a line search method in which the search direction has the form

$$d_k = -H_k g_k. \tag{4.5}$$

The inverse Hessian approximation H_k, which is not formed explicitly, is defined by a small number of BFGS updates. In the standard BFGS method, H_k is updated at every iteration by means of the formula

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T, \tag{4.6}$$

where

$$\rho_k = 1/y_k^T s_k, \qquad V_k = I - \rho_k y_k s_k^T, \tag{4.7}$$

and

$$s_k = x_{k+1} - x_k, \qquad y_k = g_{k+1} - g_k.$$

The $n \times n$ matrices H_k are generally dense, so that storing and manipulating them is impractical when the number of variables is large. To circumvent this problem, the limited memory BFGS method does not form these matrices but only stores a certain number, say m, of pairs $\{s_k, y_k\}$ that define them implicitly through the BFGS update formula (4.6)-(4.7). Two important features of the method, which we now describe, are a rescaling (or resizing) strategy, and the continuous refreshing of the curvature information.

Suppose that the current iterate is x_k and that we have stored the m pairs $\{s_i, y_i\}$, $i = k-m, ..., k-1$. We first define the basic matrix $H_k^{(0)} = \gamma_{k-1} I$ where

$$\gamma_{k-1} = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}. \tag{4.8}$$

We then (formally) update $H_k^{(0)}$ m times using the BFGS formula (4.6)-(4.7) and the m pairs $\{s_i, y_i\}$, $i = k-m, ..., k-1$. The product $H_k g_k$ is obtained by performing a sequence of inner products involving g_k and these m pairs $\{s_k, y_k\}$. After computing the new iterate, we save the most recent correction pair $\{s_k, y_k\}$ – unless the storage is full, in which case we first delete the oldest pair $\{s_{k-m}, y_{k-m}\}$ to make room for the newest one, $\{s_k, y_k\}$.

This approach is suitable for large problems because it has been observed in practice that small values of m, say $m \in [3, 20]$ often give satisfactory results [44], [31]. The numerical performance of the limited memory method L-BFGS is illustrated in Table 2, where we compare it [65] with the Newton method provided by the LANCELOT package on a set of test problems from the CUTE [6] collection.

Table 2. Unconstrained Problems

Problem	n	L-BFGS		LANC/Newt	
		nfg	time	nfg	time
CRAGGLVY	1000	95	13	15	10
FMINSURF	1024	186	11	316	106
DIXMAANI	1500	1237	166	8	9
EIGENCLS	462	2900	563	543	2300

The total number of function and gradient evaluations is denoted by nfg, and time denotes the total execution time on a Sparcstation-2. The memory parameter for L-BFGS was set to $m = 5$, and LANCELOT was run with all its default settings. The four problems typify situations we have observed in practice. In CRAGGLVY Newton's method required much fewer function evaluations, but the execution times of the two methods were similar; this is a common occurrence. FMINSURF is highly atypical in that L-BFGS required fewer function and gradient evaluations. In the ill-conditioned problem DIXMAANI the advantage of the Newton method is striking in both measures, and illustrates a case in which L-BFGS performs quite poorly. Finally EIGENCLS represents a situation that is not uncommon: even though LANCELOT requires much fewer function evaluations it takes longer to solve the problem.

We can draw three conclusions from our computational experience with Newton and limited memory methods. First, it is clear that the quality of the limited memory matrix is rather poor compared with the true Hessian, as is shown by the wide gap in the number of iterations and function evaluations required for convergence. The second observation is that the relative cost of the L-BFGS iteration is so low that one cannot discount the possibility that it will require less computing time than the Newton method. Finally, L-BFGS is not as reliable as a Newton or partially separable quasi-Newton method. On problems with an unfavorable eigenvalue distribution L-BFGS may require a huge number of iterations [8], or may not achieve good accuracy in the answer. These difficulties are sometimes overcome by increasing m, say to 20 or 30, but this is not always the case.

An attractive feature of L-BFGS is that it can easily be generalized to solve bound constrained problems. But in order to obtain an efficient implementation in that case it is necessary to find new representations of limited memory matrices, which we now discuss.

4.2.1 Compact Representations of Limited Memory Matrices

The limited memory techniques described so far only store the difference vectors s_i and y_i, and avoid storing any matrices. We now show that limited memory updating can also be described using outer products of matrices. We begin by describing a result on BFGS updating that is interesting in its own right.

Let us define the $n \times k$ matrices S_k and Y_k by

$$S_k = [s_0, \ldots, s_{k-1}], \qquad Y_k = [y_0, \ldots, y_{k-1}]. \tag{4.9}$$

It can be shown [11] that if H_0 is symmetric and positive definite, and if H_k is obtained by updating H_0 k times using the BFGS formula (4.6) and the pairs $\{s_i, y_i\}_{i=0}^{k-1}$, then

$$H_k = H_0 + \begin{bmatrix} S_k & H_0 Y_k \end{bmatrix} \begin{bmatrix} R_k^{-T}(D_k + Y_k^T H_0 Y_k) R_k^{-1} & -R_k^{-T} \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ Y_k^T H_0 \end{bmatrix}, \tag{4.10}$$

where R_k and D_k are $k \times k$ matrices given by

$$(R_k)_{i,j} = \begin{cases} s_{i-1}^T y_{j-1} & \text{if } i \leq j \\ 0 & \text{otherwise} \end{cases}, \tag{4.11}$$

and

$$D_k = \text{diag}\left[s_0^T y_0, \ldots, s_{k-1}^T y_{k-1}\right]. \tag{4.12}$$

It is easy to describe a limited memory implementation based on this representation. We keep the m most recent difference pairs $\{s_i, y_i\}$ in the matrices S_k and Y_k, and H_0 stands for the basic matrix $H_k^{(0)}$ defined through (4.8). The difference pairs are refreshed at every iteration by removing the oldest pair and adding a new one to S_k and Y_k. After this is done, the matrices R_k and D_k are updated to account for these changes.

Note that the inner matrix in (4.10) is of size $2m \times 2m$, i.e. it is very small, so that the total storage of this representation is essentially the same as storing only the difference pairs $\{s_i, y_i\}$. One can show that updating the limited memory matrix and computing the search direction $H_k g_k$ using the compact representation (4.10) costs roughly the same as in the approach described earlier, so that there is no clear benefit from using the compact form in the unconstrained case.

There are, however, many advantages to this approach if we wish to use updating formulae other than BFGS, or if we need to solve problems with bounds on the variables. For example products of the form $H_k A$, where A is a sparse matrix, occur often in constrained optimization and can be performed efficiently using (4.10). In particular, when using the range-space or dual approach [32] to solve linearly constrained subproblems, we need to compute $A^T H_k A$, whose symmetry can be exploited to give further savings in computation.

In constrained optimization, however, it is more common to work with an approximation B_k to the Hessian matrix, rather than with an inverse approximation H_k. One can derive compact representations for the limited memory Hessian approximation B_k that are similar to (4.10). These give rise to considerable savings compared with the simple-minded approach of storing only the correction vectors arising in BFGS updating. There are, in addition, compact representations for the symmetric rank-one (SR1) updating formula, which is particularly appealing in the constrained setting because it is not restricted by the positive definiteness requirement.

The recently developed code L-BFGS-B [12], [65] uses a gradient projection approach together with compact limited memory BFGS matrices to solve the bound constrained optimization problem

$$\min f(x)$$

$$\text{subject to} \quad l \leq x \leq u.$$

Table 3 illustrates the performance of L-BFGS-B on bound constrained problems from the CUTE collection. Once more we use the Newton code of LANCELOT as a benchmark [65].

Table 3. Bound Constrained Problems

			L-BFGS-B		LANC/Newt	
Problem	n	nbds	nfg	time	nfg	time
JNLBRNGA	15,625	5,657	332	740	22	1503
LINVERSE	999	338	291	57	28	150
OBSTCLAE	5,625	2,724	258	207	6	1423
TORSION6	14,884	12,316	362	707	9	130

Here nbds denotes the number of active bounds at the solution. We find again that the Newton code of LANCELOT requires much fewer function evaluations, but in terms of computing time L-BFGS-B performs quite well. This may be due to the fact that the compact representations allow us to implement the projected gradient method with minimal computational cost. We should note, however, that L-BFGS-B fails to solve a few of the bound constrained problems in the CUTE collection to reasonable accuracy, and that the Newton method is more reliable in this respect.

4.2.2 Current Research and Open Questions

We have devoted much attention so far to the L-BFGS method, but this may not be the most economical limited memory method. New algorithms designed to reduce the amount of storage without compromising performance are proposed in [23], [60], [27], [43]; see also [1]. It is easy to see that if the BFGS method is started with an initial matrix that is a multiple of the identity, then $s_k \in span\{g_0, ..., g_k\}$. This suggests that there is some redundancy in storing both s_i and y_i in a limited memory method, and in most of the recent proposals storage is in fact cut in half.

A variety of new limited memory methods have been proposed to realize these savings; some of them are based on ingenious formulas for updating the information containing the quasi-Newton update information. Even though the algorithm proposed in [60] appears to give good performance compared with L-BFGS more analysis and testing is necessary.

Another recently proposed idea [13] is to combine the properties of Newton and limited memory in the *Discrete Newton Method with Memory*. This method attempts to reduce the computational cost of the Hessian free Newton method by saving information from the inner CG iteration and keeping it in the form of a limited memory matrix. Once this information has been gathered, a sequence of limited memory steps is performed until it is judged that a new Hessian free Newton step is needed.

Thus the algorithm interleaves limited memory and Hessian free Newton steps, but it does not simply alternate them. The key is to view the inner CG iteration in Algorithm I from the perspective of quasi-Newton methods, and to realize that it may probe the function f along directions of small curvature that would normally be ignored by a limited memory method; this information could improve the quality of the limited memory matrix. The Hessian free Newton step therefore serves the dual purpose of giving good progress towards the solution and of gathering important information for the subsequent limited memory steps.

Good results have been obtained with the Discrete Newton Method with Memory, when solving problems in a controlled setting [13]. In these experiments the eigenvalue distribution of the Hessian was known, and the inner CG iteration was designed to take advantage of it. Extensive numerical tests have not yet been performed, and it remains to be seen if these ideas – or variations of them – will prove to be valuable in practice.

4.3 Sparse Quasi-Newton Updates

An interesting idea that had been explored [62] and abandoned in the late 1970's has recently been resurrected [26], [28]. It consists of developing quasi-Newton updates that mimic the sparsity pattern of the Hessian matrix $\nabla^2 f$.

In the approach described in [28], the goal is to construct a symmetric matrix B_{k+1} with the same sparsity pattern as $\nabla^2 f$, and which attempts to satisfy the secant conditions $B_{k+1}s_j = y_j$, $j = k - m + 1, ..., k$, as well as possible along m past directions. The sparse quasi-Newton matrix is constructed using the following variational approach. Let S_k and Y_k denote the matrices containing the m most

recent difference pairs, as in (4.9), and let Ω specify the sparsity pattern of the Hessian matrix. The matrix B_{k+1} will be defined as the solution to

$$\min_B \|BS_k - Y_k\|_F^2$$

$$\text{subject to} \quad B = B^T, \qquad \text{and} \quad B_{i,j} = 0 \quad \text{for all} \quad i,j \in \Omega,$$

where $\|\cdot\|_F$ denotes the Frobenius norm. Thus the secant equation $B_{k+1}s_j = y_j$, may not be satisfied even along the latest search direction s_k.

This convex optimization problem always has a solution, but to compute one is not easy. It is shown in [28] that the solution is unique if S_k satisfies a certain linear independence assumption. In this case B_{k+1} can be computed by solving a positive definite system – but B_{k+1} itself is not guaranteed to be positive definite. The analysis, which is quite novel, also reveals the minimum number m of difference pairs required to estimate a Hessian with a given sparsity pattern. Finding out this minimum number, can be difficult because it requires consideration of all possible orderings of certain sparse matrices. Nevertheless some interesting cases are simple to analyze. For example, it is shown that 2 difference pairs are sufficient to estimate an arrowhead matrix.

A trust region method implementing these ideas is given in [28]. Since the number of elements of the Hessian approximation B_{k+1} that can be estimated is limited by the number m of difference pairs in S_k and Y_k, the algorithm begins by approximating only the diagonal; after the second iteration the diagonal plus one off-diagonal element per column is estimated, and so on. Once sufficient difference pairs have been saved to approximate all the nonzero elements in B_{k+1}, the oldest pair is replaced by a new one, as in limited memory updating.

One of the drawbacks of this approach is that the system that needs to be solved to obtain the new sparse quasi-Newton matrix B_{k+1} can be very large: its dimension equals the number of nonzeros in the lower triangular part of the Hessian. Preliminary numerical tests appear to indicate that this sparse quasi-Newton method requires fewer iterations than the L-BFGS method, but the difference seems to be too small to overcome the much larger expense of the iteration. It is not known if this approach can be made into a competitive algorithm. If this were to be the case, its main use could be in constrained optimization, where the sparse quasi-Newton matrix would be part of a KKT system.

5 Nonlinear Conjugate Gradient Methods

Conjugate gradient methods remain very popular due to their simplicity and low storage requirements. They fall in the same category as limited memory and Hessian free Newton methods, in that they require only gradient information (and no information about the structure of the objective function), and use no matrix storage. Even though limited memory and Hessian free methods tend to be more predictable, robust and efficient, nonliner CG methods require only a fraction of the storage. Most of the recent work in nonlinear CG methods has focused on global convergence properties and on the design of new line search strategies.

The search direction in all nonlinear conjugate gradient methods is given by

$$d_k = -g_k + \beta_k d_{k-1}. \tag{5.1}$$

There are two well-known choices [25] for β_k: the Fletcher-Reeves formula

$$\beta_k^{\text{FR}} = \frac{\|g_k\|^2}{\|g_{k-1}\|^2},$$

and the more successful Polak-Ribière formula

$$\beta_k^{\rm PR} = \frac{g_k^T(g_k - g_{k-1})}{\|g_{k-1}\|^2}. \tag{5.2}$$

The new iterate is given by $x_{k+1} = x_k + \alpha_k d_k$, where the steplength α_k (usually) satisfies the *strong Wolfe conditions* [45]

$$\begin{aligned} f(x_k + \alpha_k d_k) &\leq f(x_k) + \sigma_1 \alpha_k g_k^T d_k & (5.3)\\ |g(x_k + \alpha_k d_k)^T d_k| &\leq -\sigma_2 g_k^T d_k, & (5.4) \end{aligned}$$

where $0 < \sigma_1 < \sigma_2 < \frac{1}{2}$.

A major drawback of nonlinear CG methods is that the search directions tend to be poorly scaled, and the line search typically several function evaluations to obtain an acceptable steplength α_k. This is in sharp contrast with quasi-Newton and limited memory methods which accept the unit steplength most of the time. Nonlinear CG methods would therefore be greatly improved if we could find a means of properly scaling d_k. Many studies have suggested search directions of the form

$$d_k = -H_k g_k + \beta_k d_{k-1} \tag{5.5}$$

where H_k is a simple symmetric and positive definite matrix, often satisfying a secant equation [40]. However, if H_k requires several vectors of storage, the economy of the nonlinear CG iteration disappears, and its performance compared with limited memory methods is unlikely to be as good. This is because the second term in (5.5) may prevent d_k from being a descent direction unless the line search is relatively accurate. In addition, the last term in (5.5) can introduce bad scaling in the search direction. So far all attempts to derive an efficient method of the form (5.5) have been unsuccessful.

An interesting framework for studying nonlinear CG, as well as quasi-Newton and Newton methods, is that of *Successive Affine Reduction* [51]. The idea is to make curvature estimates of the Hessian matrix in a low dimensional subspace formed by some of the most recent gradients and search directions. Quadratic termination properties of these methods have been studied in some detail, but practical implementations have not yet been fully developed, and their effectiveness in the context of large scale optimization remains to be demonstrated.

In an effort to improve nonlinear CG methods, some researchers have turned to global convergence studies to gain new insights into their behavior. This work is based on two important results concerning the Polak-Ribière [58] and Fletcher-Reeves methods [2]. In both cases it is assumed that the starting point x_0 is such that the level set $\mathcal{L} := \{x : f(x) \leq f(x_0)\}$ is bounded, and that in some neighborhood $\mathcal{N}$ of $\mathcal{L}$ the objective function f is continuously differentiable, and its gradient is Lipschitz continuous. The nonlinear CG method is also assumed to include no regular restarts.

It is shown in [58] that the Polak-Ribière method may fail to approach a solution point, in the sense that the sequence $\{\|g_k\|\}$ is bounded away from zero. In this analysis the line search always finds the first stationary point of the univariate function $\Psi(\alpha) = f(x_k + \alpha_k d_k)$. Recently, however, it has been shown [36] that these difficulties can be overcome by using a new line search strategy. More specifically, the Polak-Ribière iteration using this line search satisfies

$$\liminf_{k\to\infty} \|g_k\| = 0. \tag{5.6}$$

However numerical results appear to indicate that only a marginal improvement over the standard implementation of the Polak-Ribière method is obtained.

A different approach is motivated by the observation [58] that some undesirable behavior of the Polak-Ribière method occurs if the parameter $\beta_k^{\rm PR}$ becomes negative at regular intervals. It is shown in [30] that if β_k is defined as

$$\beta_k = \max\{\beta_k^{\rm PR}, 0\}, \tag{5.7}$$

and if the line search satisfies a slight modification of the strong Wolfe conditions (5.3)-(5.4), then the global convergence result (5.6) can be established. This analysis has been generalized in [37] by allowing a more flexible line search. Numerical experiments again fail to show a significant improvement in performance over the standard Polak-Ribière method.

The analysis for the Fletcher-Reeves method is simpler. It is shown in [2] that if the line search satisfies the strong Wolfe conditions then the Fletcher-Reeves method is globally convergent in the sense that (5.6) is satisfied. The same result is proved in [64] for all methods of the form (5.1) with a line search satisfying the strong Wolfe conditions, and with any β_k such that $0 \leq \beta_k \leq \beta_k^{\rm FR}$. The analysis is taken one step further in [30], where it is shown that global convergence is obtained for any method with $|\beta_k| \leq \beta_k^{\rm FR}$. Moreover this result is tight in the following sense: there exists a smooth function f, a starting point, and values of β_k satisfying

$$|\beta_k| \leq c\beta_k^{\rm FR},$$

for some $c > 1$, such that the sequence of gradient norms $\{\|g_k\|\}$ is bounded away from zero.

Even though most of these theoretical results are interesting, and some of the proof techniques are innovative, these studies have not lead to significant practical advances in nonlinear CG methods. Their main contribution has been a better understanding of the crucial role played by line searches.

Acknowledgments. I would like to thank Marcelo Marazzi and Guanghui Liu for carefully reading the manuscript and suggesting many improvements.

Bibliography

1. L. Adams and J.L. Nazareth, eds. (1996). *Linear and nonlinear conjugate gradient-related methods*, SIAM.

2. M. Al-Baali (1985). Descent property and global convergence of the Fletcher-Reeves method with inexact line search, *IMA Journal of Numerical Analysis* **5**, 121–124.

3. M. Arioli, T.F. Chan, I. S. Duff, N.I.M. Gould and J. K. Reid (1993). Computing a search direction for large-scale linearly constrained nonlinear optimization calculations, *Technical Report TR/PA/93/94*, CERFACS Toulouse, France.

4. B.M. Averick, R.G. Carter, J.J. Moré and G. Xue (1992). The Minpack-2 test problem collection, Preprint *MCS-P153-0692*, Mathematics and Computer Science Division, Argonne National Laboratory.

5. C.H. Bischof, A. Bouaricha, P.M. Khademi and J.J Moré (1995). Computing gradients in large scale optimization using automatic differentiation. Report ANL/MCS-P488-0195, Argonne National Laboratory.

6. I. Bongartz, A. R. Conn, N.I.M. Gould, Ph.L. Toint (1993). CUTE: constrained and unconstrained testing environment, *Research Report*, IBM T.J. Watson Research Center, Yorktown Heights, NY.

7. A. Bouaricha and J.J. Moré (1996). A preconditioned Newton method for large-scale optimization. Presented at the *Workshop on Linear Algebra in Optimization*, Albi, France.

8. M.G. Breitfeld and D.F. Shanno (1994). Computational experience with modified log-barrier functions for large-scale nonlinear programming, in W.W. Hager, D.W. Hearn and P.M. Pardalos, eds., *Large Scale Optimization: State of the Art*, Kluwer Academic Publishers B.B.

9. A. Buckley and A. LeNir (1985). BBVSCG –A variable storage algorithm for function minimization, *ACM Transactions on Mathematical Software* 11, 2, pp. 103–119.

10. A. Buckley (1989). Remark on algorithm 630, *ACM Transactions on Mathematical Software* 15, 3, pp. 262–274.

11. R. H. Byrd, J. Nocedal and R. B. Schnabel (1994). Representation of quasi-Newton matrices and their use in limited memory methods, *Mathematical Programming* 63, 4, pp. 129–156.

12. R. H. Byrd, P. Lu, J. Nocedal and C. Zhu (1995). A limited memory algorithm for bound constrained optimization, *SIAM Journal on Scientific Computing*, 16, 5, pp. 1190–1208.

13. R. H. Byrd, J. Nocedal and C. Zhu (1995). Towards a discrete Newton method with memory for large scale optimization, to appear in *Nonlinear Optimization and Applications*, G. Di Pillo and F. Giannessi, eds. Plenum.

14. T.F. Coleman (1991). Large-Scale Numerical Optimization: Introduction and Overview, Cornell Theory Center, Advanced Computing Research Institute, *Technical Report* **85**, pp. 1–28.

15. T.F. Coleman and J. Moré (1984). Estimation of sparse Hessian matrices and graph coloring problems, *Mathematical Programming* 28, 243–270.

16. T.F. Coleman (1984). Large Sparse Numerical Optimization, *Lecture Notes in Computer Sciences* 165, Springer Verlag.

17. A.R.Conn, N.I.M. Gould, Ph.L. Toint (1992). LANCELOT: a FORTRAN package for large-scale nonlinear optimization (Release A), Number 17 in Springer Series in *Computational Mathematics*, Springer-Verlag, New York.

18. A.R. Conn, N.I.M. Gould, A. Sartenaer and Ph. L. Toint (1996). On iterated-subspace minimization methods for nonlinear optimization, in Linear and Nonlinear Conjugate Gradient-Related Method, L. Adams and L. Nazareth, eds, *SIAM*.

19. A.R. Conn, N.I.M. Gould and Ph. L. Toint (1991). Convergence of quasi-Newton matrices generated by the symmetric rank one update, *Math. Prog.* **2**, 177–195.

20. A. Curtis, M.J.D. Powell and J.K. Reid (1974). On the Estimations of Sparse Jacobian Matrices, *J.I.M.A.* **13**, pp. 117–120.

21. R.S. Dembo, S.C. Eisenstat and T. Steihaug (1982). Inexact Newton methods, *SIAM J. Numer. Anal.* **19**, 400–408.

22. J.E. Dennis, Jr. and R.B. Schnabel (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs, N.J., Prentice-Hall.

23. M.C. Fenelon (1981). Preconditioned conjugate-gradient-type methods for large-scale unconstrained optimization, *Ph.D. Dissertation*, Stanford University.

24. R. Fletcher (1993). An overview of unconstrained optimization, *Tech. Report NA/149*, Department of Mathematics and Computer Science, University of Dundee.

25. R. Fletcher (1987). *Practical Methods of Optimization* **1**, Unconstrained Optimization, John Wiley & Sons (New York).

26. R. Fletcher (1995). An optimal positive definite update for sparse Hessian matrices, *SIAM Journal on Optimization*, 5 (1): 192-218, February 1995.

27. R. Fletcher (1990). Low storage methods for unconstrained optimization, *Computational Solutions of Nonlinear Systems of Equations,* eds. E. L. Allgower and K. Georg, Lectures in Applied Mathematics **26**, AMS Publications, Providence, RI.

28. R. Fletcher, A. Grothey and S. Leyffer (1996). Computing Sparse Hessian and Jacobian approximations with optimal hereditary properties, *Tech. Report*, Department of Mathematics, University of Dundee.

29. D.M. Gay (1996). More AD of nonlinear AMPL models: computing Hessian information and exploiting partial separability, to appear in the *Proceedings of the Second International Workshop on Computational Differentiation*.

30. J. C. Gilbert and J. Nocedal (1990). Global convergence properties of conjugate gradient methods for optimization, *SIAM Journal on Optimization*, 2, (1).

31. J.C. Gilbert and C. Lemaréchal (1989). Some numerical experiments with variable storage quasi-Newton algorithms, *Mathematical Programming* 45, pp. 407–436.

32. P. E. Gill, W. Murray and M. H. Wright (1981). *Practical Optimization*, London, Academic Press.

33. G.H. Golub and C.F. Van Loan (1989). *Matrix Computations (Second Edition)*, The Johns Hopkins University Press, Baltimore and London.

34. A. Griewank and Ph.L. Toint (1982). On the unconstrained optimization of partially separable objective functions, in *Nonlinear Optimization 1981* , M.J.D. Powell, ed., Academic Press (London), 301–312.

35. A. Griewank and G.F. Corliss, eds. (1991). *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, SIAM publications.

36. L. Grippo and S. Lucidi (1995). A globally convergent version of the Polak-Ribière gradient method, Dipartimento di Informatica e Sistematica, Universita degli Studi di Roma "La Sapienza", R. 08-95.

37. J. Han, G. Liu, D. Sun and H. Yin (1996). Relaxing sufficient descent condition for nonlinear conjugate gradient methods. Tech. Report, Institute of Applied Mathematics, Academia Sinica, China.

38. T. Head-Gordon, F.H. Stillinger, D.M. Gay and M.H. Wright (1992). Poly(L-alanine) as a universal reference material for understanding protein energies and structures, *Proceedings of the National Academy of Sciences*, USA, 89, pp. 11513-11517.

39. N.J. Higham (1996). *Accuracy and Stability of Numerical Algorithms*, SIAM Publications.

40. Y.F. Hu and C. Storey (1990). On unconstrained conjugate gradient optimization methods and their interrelationships, *Mathematics Report Number A129*, Loughborough University of Technology.

41. M.T. Jones and P. Plassman (1995). An improved incomplete Cholesky factorization, *ACM Trans. on Mathematical Software*, 21 pp. 5-17.

42. N.K. Karmakar and K.G. Ramakrishnan (1991). Computational results of an interior point algorithm for large scale linear programming, *Mathematical Programming*, 52, pp. 555-586.

43. M.W. Leonard (1995). Reduced Hessian quasi-Newton methods for optimization. *Ph.D. Dissertation*, University of California, San Diego.

44. D. C. Liu and J. Nocedal (1989). On the limited memory BFGS method for large scale optimization, *Mathematical Programming*, pp. 503-528.

45. J. J. Moré and D.J. Thuente (1994). Line search algorithms with guaranteed sufficient decrease, *ACM Transactions on Mathematical Software*, Vol. 20, No. 3, pp. 286-307.

46. S.G. Nash (1984). Newton-type minimization via the Lanczos method, *SIAM. J. Numerical Analysis*, 21,4.

47. S.G. Nash (1984). User's guide for TN/TNBC: FORTRAN routines for nonlinear optimization, *Report 397*, Mathematical Sciences Dept., The Johns Hopkins University.

48. S.G. Nash (1985). Preconditioning of truncated-Newton methods, *SIAM Journal on Scientific and Statistical Computing* **6**, 599–616.

49. S.G. Nash and J. Nocedal (1991). A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization, *SIAM Journal on Optimization*, 1, 3, pp. 358-372.

50. A. Neumaier (1996). Molecular modeling of proteins: a feasibility study of mathematical prediction of protein structure, manuscript.

51. J.L. Nazareth, (1986). The method of successive affine reduction for nonlinear minimization, *Mathematical Programming*, 35, pp. 373-387.

52. J. Nocedal (1980). Updating quasi-Newton matrices with limited storage, *Mathematics of Computation* 35, pp. 773-782.

53. J. Nocedal (1992). Theory of algorithms for unconstrained optimization, *Acta Numerica*, 1, pp.199-242.

54. D.P. O'Leary (1982). A discrete Newton algorithm for minimizing a function of many variables, *Mathematical Programming* 23, 20–33.

55. C.C. Paige and M.A. Saunders (1975). Solution of sparse indefinite systems of linear equations, *SIAM. J. Numer. Anal.*, 12, pp. 617-629.

56. M.J.D. Powell (1970a). A hybrid method for nonlinear equations, in *Numerical Methods for Nonlinear Algebraic Equations*, (P. Rabinowitz, ed.), Gordon and Breach, London, pp. 87–114.

57. M.J.D. Powell (1984). On the global convergence of trust region algorithm for unconstrained optimization, *Mathmatical Programming* 29, pp. 297-303.

58. M.J.D. Powell (1984). Nonconvex minimization calculations and the conjugate gradient method, in *Lecture Notes in Mathematics* **1066**, Springer-Berlag (Berlin), 122–141.

59. T. Schlick and A. Fogelson (1992). TNPACK - A truncated Newton package for large-scale problems: I. Algorithms and usage. *ACM Transactions on Mathematical Software*, 18,1,46-70.

60. D. Siegel (1992). Implementing and modifying Broyden class updates for large scale optimization, *Report DAMPT 1992/NA12*, University of Cambridge.

61. T. Steihaug (1983). The conjugate gradient method and trust regions in large scale optimization, *SIAM J. Num. Anal.* 20, 626–637.

62. Ph. L. Toint (October 1977). On sparse and symmetric matrix updating subject to a linear equation, *Mathematics of Computation*, 31 (140): 954-961.

63. Ph.L. Toint (1983). VE08AD, a routine for partially separable optimization with bounded variables, *Harwell Subroutine Library*, A.E.R.E. (UK).

64. D. Touati-Ahmed and C. Storey (1990), Efficient hybrid conjugate gradient techniques, *Journal of Optimization Theory and Applications* **64**, 379–397.

65. C. Zhu, R.H. Byrd, P. Lu and J. Nocedal (1995). L-BFGS-B: FORTRAN subroutines for large-scale bound constrained optimization, *Tech. Report*, Department of Electrical Engineering and Computer Science, Northwestern University.