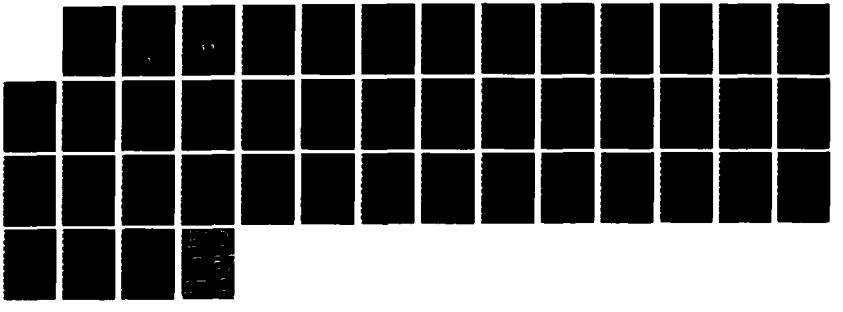
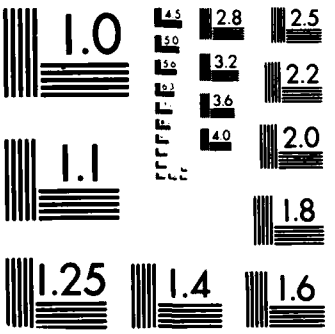
AD-A181 320 USING PARALLEL FUNCTION EVALUATIONS TO IMPROVE HESSIAN APPROXIMATIONS FOR (U) COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE R H BYRD ET AL 30 MAR 87 1/1

UNCLASSIFIED CS-CU-361-87 ARO-21453 9-MA F/G 12/6 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

(2)

UNIVERSITY OF COLORADO

USING PARALLEL FUNCTION EVALUATIONS TO IMPROVE HESSIAN APPROXIMATIONS FOR UNCONSTRAINED OPTIMIZATION

Richard H. Byrd*
Robert B. Schnabel*
Gerald A. Shultz**

CS-CU-361-87 March 1987

DEPARTMENT OF COMPUTER SCIENCE
CAMPUS BOX 430
UNIVERSITY OF COLORADO, BOULDER
BOULDER, COLORADO 80309-0430

Technical Report

(2)

USING PARALLEL FUNCTION EVALUATIONS TO IMPROVE HESSIAN APPROXIMATIONS FOR UNCONSTRAINED OPTIMIZATION

Richard H. Byrd*
Robert B. Schnabel*
Gerald A. Shultz**

CS-CU-361-87 March 1987

*Department of Computer Science, University of Colorado, Boulder, Colorado 80309

**Department of Mathematical Sciences, Metropolitan State College, Denver, Colorado 80204 and Department of Computer Science, University of Colorado, Boulder, Colorado 80309

Research supported by AFOSR grant AFOSR-85-0251, ARO contract DAAG 29-84-K-0140, NSF grant DCR-8403483, and NSF cooperative agreement DCR-84200944

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR AND DO NOT NECESSARILY REFLECT THE VIEWS OF THE NATIONAL SCIENCE FOUNDATION.

THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

Abstract

This paper presents a new class of methods for solving unconstrained optimization problems on parallel computers. The methods are intended to solve small to moderate dimensional problems where function and derivative evaluation is the dominant cost. They utilize multiple processors to evaluate the function, (finite difference) gradient, and a portion of the finite difference Hessian simultaneously at each iterate. We introduce three types of new methods, which all utilize the new finite difference Hessian information in forming the new Hessian approximation at each iteration; they differ in whether and how they utilize the standard secant information from the current step as well. We present theoretical analyses of the rate of convergence of several of these methods. We also present computational results which illustrate their performance on parallel computers when function evaluation is expensive.

1. Introduction.

This paper presents a new class of methods for solving unconstrained optimization problems on parallel computers. The methods are intended to solve small to moderate dimensional problems where function and derivative evaluation is the dominant cost. They utilize multiple processors to evaluate the function, (finite difference) gradient, and a portion of the finite difference Hessian simultaneously at each iterate. We present theoretical analyses of the rate of convergence of several of these methods. We also present computational results which illustrate their performance on parallel computers when function evaluation is expensive.

The unconstrained optimization problem is

$$\min_{x \in R^n} f : R^n \rightarrow R \tag{1.1}$$

where $f(x)$ is assumed to be at least twice continuously differentiable. This problem occurs commonly in many applications, including modeling, data fitting, and planning calculations in most areas of science and engineering. This paper is solely concerned with finding a *local minimizer* of $f(x)$, the lowest point of $f(x)$ in some open neighborhood of the variable space. This is the most common unconstrained optimization calculation in practice. For a discussion of parallel methods for global optimization, the problem of finding the lowest among multiple local minimizers of $f(x)$, see Byrd, Dert, Rinnooy Kan, and Schnabel (1986).

Unconstrained optimization problems often are expensive to solve. One reason is that the objective function, $f(x)$, often is itself a complex computer code, for example the solution of a system of partial differential equations. It is not unusual for each evaluation of $f(x)$ to require many seconds, or minutes, on a powerful computer. In addition, in many instances when $f(x)$ is expensive, the derivatives of $f(x)$

are not available analytically. In this case, optimization codes approximate the gradient of $f(x)$ at a point x_c by using the finite difference approximation

$$\nabla f(x_c)_i = \frac{f(x_c + h_i e_i) - f(x_c)}{h_i} \tag{1.2}$$

where h_i is a small stepsize and e_i denotes the i^{th} unit vector. This means that each gradient evaluation requires n function evaluations in addition to $f(x_c)$. Since the solution to the optimization problem usually requires many evaluations of $f(x)$ and $\nabla f(x)$, it becomes an expensive process. Usually, no higher derivatives are used by optimization algorithms when $f(x)$ is expensive.

If the number of variables is not too large, say $n \leq 100$, then the time required by the remainder of the optimization algorithm often is insignificant in comparison to the time required for the function and gradient calculations. This is the class of problems we consider in this paper. We orient our discussions to the common case when gradients are calculated by finite differences. We will point out, however, that our techniques can also be applied to instances where $f(x)$ is expensive but the analytic gradient is available.

Methods for solving unconstrained optimizations problems with a small or moderate number of variables on sequential computers are quite well understood (see e.g. Fletcher (1980), Gill, Murray, and Wright (1981), or Dennis and Schnabel (1983)). When second derivatives are available analytically or affordable by finite differences, variants of Newton's method are used. When analytic second derivatives are unavailable and function evaluation is expensive, variants of the BFGS method are most commonly used, using either analytic or finite difference gradients. A very brief description of such methods is included in Section 2. The Newton's method based algorithms, which are locally quadratically convergent on most problems, generally require fewer iterations than the superlinearly convergent BFGS based methods, but they generally require more function evaluations and hence more computer time on

problems where function evaluation is expensive. It will be seen that our new parallel methods try to incorporate advantages of both approaches.

Due to the high cost of solving many optimization problems, there is ample incentive to devise methods for solving them on parallel computers if they can lead to significantly faster or more cost effective solution of these problems. For problems with expensive function evaluations, there are two obvious types of approaches. One can use standard sequential optimization methods but apply a parallel algorithm to evaluate $f(x)$, or one can devise methods that make effective use of evaluating $f(x)$ at multiple points concurrently. In this paper we consider the latter approach. The former approach, applying a parallel algorithm to evaluate $f(x)$, is dependent upon the actual objective function $f(x)$ and is not under the control of the optimization algorithm designer. It should be noted, however, that the two approaches often are quite compatible. For example, in cases where the evaluation of $f(x)$ vectorizes well, a computer which consists of multiple vector processors would allow multiple evaluations of $f(x)$ to be performed concurrently with each evaluation performed by a vector processor.

The concurrent evaluation of $f(x)$ at multiple points is well suited to any computer that can execute multiple, different instruction streams concurrently. Such machines are known as *Multiple Instruction Multiple Data* (MIMD) computers. This class includes both shared memory multiprocessors, and local memory multiprocessors such as hypercubes; the algorithms we discuss are well suited to any such computer. Our approach is not generally suited to *Single Instruction Multiple Data* (SIMD) computers, such as processor arrays, whose processors can execute the same instruction on different data in lockstep. This is because different evaluations of an expensive function $f(x)$ usually entail different sequences of instructions, due to data dependent branches in the code for $f(x)$, and thus cannot easily be performed concurrently on an SIMD machine.

The most obvious way to parallelize unconstrained optimization algorithms when the evaluation of $f(x)$ is expensive and the gradient is calculated by finite differences is to perform the n function evaluations required by the finite difference gradient in parallel. (If the number of processors, p, is less than n, then $\lceil n/p \rceil$ groups of p function evaluations each can be performed in parallel.) Simple approaches along these lines are discussed in Schnabel (1986). He demonstrates that if one also incorporates the technique of always evaluating the finite difference gradient when the function value at a new trial point is being evaluated (or $p-1$ components of the finite difference gradient if $p < n+1$), before it is known whether this point will be accepted as an iterate and its gradient needed, then one can make very efficient use of up to $n+1$ processors on most problems with expensive functions. We refer to this technique as *speculative* gradient evaluation.

Such algorithms, which evaluate $f(x)$ and the n function evaluations for the finite difference gradient concurrently, can utilize at most $n+1$ processors (assuming that each evaluation of $f(x)$ uses only one processor). In this paper we consider strategies that would be appropriate when $p > n+1$. Since there are many unconstrained optimization problems where the evaluation of $f(x)$ is very expensive but the number of variables is small, say $n \leq 25$, and since many parallel computers already have scores or hundreds of processors, this is a reasonably common scenario. It can be expected to become even more common as the number of processors in MIMD computers grows. It will be seen that our strategies can also be applied to computers with fewer than n processors in the case when $f(x)$ and $\nabla f(x)$ are both evaluated analytically by one processor.

If $p \geq (n^2 + 5n + 2)/2$, then it is possible to evaluate the function, the finite difference gradient, and a finite different Hessian approximation simultaneously, and the best strategy is probably just a standard Newton's method based algorithm. A very likely situation, however, is that $p \in (n+1), (n^2 + 5n + 2)/2)$, so that there are more than enough processors to evaluate the finite

difference gradient but not enough to calculate the full finite difference Hessian as well. For example, if $p = 64$, any problem with $n \in [10, 63]$ falls in this class. This is the main situation we consider in this paper.

The remainder of this paper considers ways to use multiple processors to evaluate the function, (finite difference) gradient, and a portion of the information comprising the finite difference Hessian at each iteration. In Section 2 we propose a number of possible ways to do this. They are based upon using the extra processors to calculate extra (finite difference) gradients which determine $\nabla^2 f(x_c)u_i$ in carefully chosen directions u_i, and incorporating this information either by overwriting part of the Hessian approximation or by secant updates. Some of our algorithms also incorporate the standard secant equation into the Hessian approximation. In Section 3 we analyze the local convergence of some of these methods. Several are shown to be m-step quadratically convergent, for p large enough that n extra gradients can be calculated in the course of m steps. In Section 4 we discuss several considerations involved in the implementation of these algorithms. Section 5 contains computational results of running many of these methods on standard unconstrained optimization test problems. While the results are obtained on a sequential computer, they are easily used to show what speedup would be obtained on parallel computers when function evaluation is the overriding cost. In Section 6 we summarize our conclusions and briefly discuss our plans for continuing this research.

We conclude this section by giving some notation that we will use in this paper.

Notation

Let $g(x) = \nabla f(x)$ and $H(x) = \nabla^2 f(x)$.

Define $\| \ \|$ to be the Euclidean norm $\| \ \|_2$, and let $\kappa(A) = \|A\| \, \|A^{-1}\|$ for any non-singular $n \times n$ real matrix A.

For any $n \times n$ real symmetric matrix B, let $\lambda_1(B), \ldots, \lambda_n(B)$ be the n real eigenvalues of B, in increasing order.

2. Algorithms.

In this section we describe a number of alternative algorithms for unconstrained optimization in a parallel processing environment. They include straightforward parallelizations of the usual BFGS and Newton's methods, as well as a number of new algorithms. All of these algorithms may be regarded as interpolations between Newton's method and a quasi-Newton method. First we present a basic algorithmic framework for all these methods (Algorithm 2.1), and then we motivate and describe several specific algorithms as versions of this basic framework.

All our algorithms are intended for the case when function evaluation is expensive. We orient our discussion in this section to the case when the evaluation of $g(x)$ is by the finite difference formula (1.2), so that it requires n function evaluations, plus the evaluation of $f(x)$. Our algorithms will perform these function evaluations concurrently. For simplicity, we assume that the number of processors, p, is $(q+1)(n+1)$, for some $q \geq 0$, and that $m = \frac{n}{q}$ is an integer. Our algorithms can also be applied to the case where the gradient is available analytically and each component $g_i(x)$ can be evaluated by a separate processor; in this case, they require $p = n(q+1)+1$ processors. They could also be used in a case where $g(x)$ and $f(x)$ are evaluated on the same processor and $p = q+1$ processors are available.

Algorithm 2.1.

0) Let $\alpha \in (0, \frac{1}{2})$, $\beta \in (\alpha, 1)$, $x_1 \in R^n$, $q \geq 0$, B_0 be a positive definite, symmetric matrix, and $k := 1$.

1) Evaluate $f_1 := f(x_1)$, $g_1 := g(x_1)$, and possibly other values in parallel.

2) Determine B_1.

3) Using B_k, compute search direction d_k.

4) Determine the set U_{k+1} of q finite difference directions.

5) Set steplength $\mu_k := 1$.

6) $x_{k+1} := x_k + \mu_k d_k$.

7) Compute $f(x_{k+1})$, $g(x_{k+1})$, and V_{k+1} = finite difference approximation to $H(x_{k+1})U_{k+1}$ in parallel. (requires q additional gradient values)

8) If (x_{k+1} does not satisfy conditions (2.1) and (2.2)) then adjust μ_k and go to 6).

9) Determine B_{k+1}.

10) $k := k + 1$.

11) Stop or go to 3).

Note that this algorithmic framework includes the standard methods, in that it allows B_{k+1} to be set to the values chosen by Newton's method or standard secant methods in step 9). However, computing additional information in step 7) and using this information in step 9) results in some new algorithms that can take advantage of the ability to perform computations in parallel. Steps 1), 2), 3), 4), 7), and 9) are left unspecified, and by making specific choices in these steps we will specify the new methods.

The key issue is that additional function evaluations can be done simultaneously with the evaluation of $f(x)$ and $g(x)$ at step 7), if sufficiently many processors are available. One of the main ideas of the new algorithms is to utilize the extra processors to improve the Hessian approximation at the iterate x_{k+1}. In step 4), the option is available to pick directions U_{k+1} from x_{k+1} along which extra gradient information will be computed in step 7). Note that these finite difference directions, U_{k+1}, must be chosen without knowledge of $f(x_{k+1})$ or $g(x_{k+1})$. Then, in step 7), the gradients $g(x_{k+1} + h_i U_{k+1} e_i)$, for

$i = 1, \ldots, q$, where h_i is a small stepsize, may be approximated from function evaluations. These gradient evaluations, which require $q(n+1)$ additional function evaluations, are used to determine a finite difference approximation to $V_{k+1} = H(x_{k+1})U_{k+1}$ as detailed in Section 4.

The question of how to utilize the extra gradient information obtained in step 7) to help form B_{k+1} is left unspecified in step 9). This step, combined with the choice of finite difference directions in step 4), is the crucial part of the new algorithms. Note that steps 1) and 7) are the only locations in Algorithm 2.1 at which function evaluations are performed, except for Newton's method, which also performs some function evaluations in step 9). Step 1) provides for the possibility of doing extra function evaluations, in parallel with the computation of $f(x_1)$ and $g(x_1)$, to produce B_1 that is a better approximation to $H(x_1)$ than the a priori estimate B_0.

For some of the algorithms, the search direction d_k is taken to be $-B_k^{-1}g_k$, but in leaving step 3) unspecified we provide for two other cases that occur in some of the algorithms. Depending on how B_k is determined in step 9), the approximate Hessian may not be positive definite, and d_k is then taken as $-(B_k + \sigma_k I)^{-1}g_k$, for some σ_k that makes $B_k + \sigma_k I$ positive definite. Also, several algorithms use a temporarily updated version of B_k in computing d_k.

Algorithm 2.1 includes a linesearch that satisfies the standard conditions,

$$f(x_k) - f(x_{k+1}) \leq \alpha g(x_k)^T(x_{k+1} - x_k), \tag{2.1}$$

and

$$g(x_{k+1})^T(x_{k+1} - x_k) \geq \beta g(x_k)^T(x_{k+1} - x_k). \tag{2.2}$$

We do not detail how the steplength parameter μ_k is to be adjusted so that at any iterate conditions (2.1) and (2.2) will be satisfied upon reaching step 9); our implementation uses the procedure in Dennis and Schnabel (1983). Accumulated computational experience indicates that, for reasonable values of α and β,

these conditions are satisfied by the initial value $\mu_k = 1$ most of the time. When that condition occurs, we are able to use immediately all the information gathered in step 7), so that all the algorithms described below, except for Newton's method, require only the time for one concurrent set of function evaluations to carry out the step.

We now describe, in a natural order, the algorithms to be considered in this paper.

Quasi-Newton and Newton methods

We first discuss how two standard algorithms, Newton's method and the BFGS method, may be advantageously implemented on parallel processors. The BFGS method algorithm, which we designate by "S" for "step update", is obtained from Algorithm 2.1 as follows. The matrix B_1 is taken to be some scaled multiple of the identity matrix. No finite difference directions are chosen at step 4), and no extra function evaluations are done in step 7). The approximate Hessian is determined in step 9) by the BFGS step update, namely

$$B_{k+1} := B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{s_k^T y_k}, \tag{2.3}$$

where $s_k = \mu_k d_k$ and $y_k = g_{k+1} - g_k$. The step is computed simply by $d_k = -B_k^{-1} g_k$, since B_k is guaranteed to be positive definite. This algorithm, as discussed in Schnabel (1986), attains near-optimal speedup on $n+1$ or fewer processors by evaluating the gradient at each trial point in step 7) in parallel with the evaluation of $f(x_{k+1})$. Thus, if the linesearch conditions are satisfied by $\mu_k = 1$, and $p \geq n+1$, that iteration of the algorithm requires only the time of one function evaluation. However, this method cannot use more than $n+1$ processors. We turn, therefore, to consideration of methods that can utilize $(q+1)(n+1)$ processors available when $q \geq 1$.

Since Newton's method is known to be an effective method in the sequential case, it is natural to consider a parallel version of this algorithm. We will designate our version of Newton's method by "N". In Newton's method B_1, as well as each B_{k+1}, is taken to be the finite difference Hessian matrix at the corresponding iterate. This requires $\frac{(n^2+3n)}{2}$ additional evaluations of $f(x)$. Since the true Hessian need not be positive definite, s_k is taken to be $-(B_k+\sigma_k I)^{-1}g_k$, where σ_k is chosen, as in Moré and Sorensen (1983), so that $B_k+\sigma_k I$ is positive definite and well-conditioned. For an efficient parallel version of Newton's method, at step 7), in parallel with the evaluation of $f(x_{k+1})$ and $g(x_{k+1})$, the extra $q(n+1)$ processors are used to compute some of the elements of $H(x_{k+1})$. Then, if the trial point x_{k+1} satisfies conditions (2.1) and (2.2), at step 9) the remaining elements of the finite difference Hessian are calculated, perhaps requiring many cycles of parallel function evaluations. If the number of processors is at least $\frac{(n^2+5n+2)}{2}$, then f, g, and H may be evaluated in one cycle, which would make Newton's method quite competitive. However, Algorithm 2.1(N) may be quite inefficient when n is large relative to q, since it requires several cycles of parallel function evaluations for each iteration.

If p is between $n+1$ and $\frac{(n^2+5n+2)}{2}$, there is room for a method which gathers more information per iteration than a secant method but less than Newton's method. In the rest of this section we consider several methods that, like the BFGS method, take only one cycle of parallel function evaluations for each trial point, but also utilize effectively the extra $q(n+1)$ function evaluations per cycle to approximate the Hessian more accurately than does the BFGS method.

Finite difference update of part of the Hessian

Perhaps the most natural idea satisfying our goals is to simply evaluate as many elements of $H(x_{k+1})$ as possible using the extra processors, and to update B_k to B_{k+1} by simply overwriting the

appropriate

components of B_k by the elements of $H(x_{k+1})$. It is interesting to note that the PSB update

$$B_{k+1} = B_k + \frac{(v - B_k u)u^T + u(v - B_k u)^T}{u^T u} - \frac{u^T(v - B_k u)uu^T}{(u^T u)^2},$$

is equivalent to overwriting row j and column j of B_k with $v \in R^n$, if u is the j^{th} standard basis direction. Similarly, if we take U_{k+1} to be q columns of the identity matrix, and V_{k+1} contains the corresponding columns of $H(x_{k+1})$, then we can determine B_{k+1} that overwrites the appropriate rows and columns of B_k with V_{k+1} by the generalized PSB formula, as in Schnabel (1983),

$$\begin{aligned} B_{k+1} = B_k &+ (V_{k+1} - B_k U_{k+1})(U_{k+1}^T U_{k+1})^{-1} U_{k+1}^T \\ &+ U_{k+1}(U_{k+1}^T U_{k+1})^{-1}(V_{k+1} - B_k U_{k+1})^T \\ &- U_{k+1}(U_{k+1}^T U_{k+1})^{-1}(V_{k+1} - B_k U_{k+1})^T U_{k+1}(U_{k+1}^T U_{k+1})^{-1} U_{k+1}^T . \end{aligned} \tag{2.4}$$

Since this method of determining B_{k+1} clearly need not result in B_{k+1} being positive definite, d_k must be calculated in the same way as in Algorithm 2.1(N). We designate this method as Algorithm 2.1(UP), where "U" indicates that the finite difference directions are chosen as "unit" vectors, and "P" indicates that the finite difference information is incorporated into B_k through the "PSB" update. Specifically, "U" means that we partition the identity matrix into $m = \frac{n}{q}$ blocks of q adjacent columns, and in step 4) chose U_{k+1} to be the next block of q columns.

Algorithm 2.1(UP) has several obvious theoretical properties. First, it is trivial to see that if $f(x)$ is a quadratic function, with Hessian H, then Algorithm 2.1(UP) will terminate after m steps, since after m steps we will have $B_m = H$. It is also fairly easy to show that this method has a local m-step Q-quadratic convergence rate, and also is 1-step Q-superlinearly convergent. One disadvantage of this method is that, even when $H(x_{k+1})$ is positive definite, it can produce an indefinite B_{k+1}. Also, it is commonly accepted

that the BFGS method is superior to the PSB method among secant algorithms for unconstrained optimization, so it is natural to consider incorporating the finite difference information into B_k by means of the BFGS update.

Finite difference update of part of the Hessian using the BFGS formula

We now consider two methods that are similar to Algorithm 2.1(UP), but that use the BFGS update to incorporate the new finite difference information, rather than the PSB. First, we consider the method obtained by simply modifying Algorithm 2.1(UP) by updating through the generalized BFGS formula (Schnabel (1983)), obtaining

$$B_{k+1} = B_k - B_k U_{k+1}(U_{k+1}^T B_k U_{k+1})^{-1} U_{k+1}^T B_k + V_{k+1}(U_{k+1}^T V_{k+1}) V_{k+1}^T . \tag{2.5}$$

Since V_{k+1} is a finite difference approximation to $H(x_{k+1})U_{k+1}$, it may be necessary to symmetrize the matrix $U_{k+1}^T V_{k+1}$. Sufficiently close to a strict local minimum, the $q \times q$ matrix $U_{k+1}^T V_{k+1}$ will be positive definite, and thus B_{k+1} will be positive definite if B_k is. In the case that $U_{k+1}^T V_{k+1}$ is not positive definite, we will use a maximal subset of the columns of U_{k+1} that yields a positive definite matrix and save the rest of the columns to use as finite difference directions at the next step. We designate this algorithm as Algorithm 2.1(UB), where "U" indicates that in step 4), U_{k+1} is taken to be the next block of q unit vectors, and "B" indicates that in step 10), B_{k+1} is determined from B_k and the finite difference information by using the generalized BFGS update. We include Algorithm 2.1(UB) only to show how much its performance is improved by an idea to be discussed later. Although this method does maintain positive definiteness of the matrices $\{B_k\}$, it does not terminate in m steps on a quadratic function, and it performs very poorly in practice.

However, by choosing the finite difference directions more intelligently we can preserve the quadratic termination property and obtain a considerably better method. If we choose U_{k+1} to be orthogonal

to the $m-1$ previous matrices V_{k-j}, for $0 \le j \le m-2$, then the finite difference directions asymptotically become block conjugate with respect to the Hessian of f. In particular, if $f(x)$ is a quadratic function with Hessian H, then the sets of directions $U_1, U_2, \ldots, U_m$ are block conjugate with respect to H, i.e., for $1 \le i < j \le m$, $U_i^T H U_j = 0$. Thus, it is easy to show that $B_m = H$, and so this method terminates in m steps on a quadratic function. We designate this algorithm as Algorithm 2.1(CB), where "C" indicates that the finite difference directions are chosen as above to be approximately conjugate, and "B" indicates that B_{k+1} is obtained from B_k and the finite difference information by (2.5). We will show in the next section that Algorithm 2.1(CB) is m-step Q-quadratically and 1-step Q-superlinearly convergent. Also, the conjugate method for choosing the finite difference update directions is invariant under linear transformations, and, of course, by updating through (2.5), the matrices B_k are all positive definite.

Step update algorithms

Algorithm 2.1(UP) and Algorithm 2.1(CB) are still somewhat unsatisfactory, in that they do not use the step information contained in g_{k+1} and g_k in forming B_{k+1}. In fact, the computational results in Section 5 show that Algorithm 2.1(S), the parallel BFGS method, which uses just the gradient differences at successive iterates to approximate the Hessian, performs better than Algorithms 2.1(UP), (UB), and (CB), which do not make use of this information and instead approximate the Hessian by using just the gradients along the finite difference directions. Thus, this information appears to be important, and we would like to approximate B_{k+1} in a way that utilizes all of the available gradient information.

One way to do this would be to simply update B_k twice at each iteration, along the step direction, as in standard secant methods, and along the finite difference directions. To do this we modify Algorithms 2.1(UP), (UB), and (CB) as follows. In step 9), the counterparts of Algorithms 2.1(UB) and (CB) first do the step update (2.3) using the direction $s_k = x_{k+1} - x_k$; the counterpart of Algorithm 2.1(UP) does

the step update

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)s_k^T + s_k(y_k - B_k s_k)^T}{s_k^T s_k} - \frac{s_k^T(y_k - B_k s_k)s_k s_k^T}{(s_k^T s_k)^2}. \tag{2.6}$$

The resulting matrix in either case is then updated by the finite difference update as in methods (UP), (UB), or (CB). We designate these algorithms by (UPS), (UBS), and (CBS), where "S" indicates that a step update (2.3) or (2.6) is done. Note that the extra update uses information already available, so that still only one concurrent function evaluation is required per step.

This approach has the drawback that there is little hope of choosing the finite difference directions effectively as was done with (UP) and (CB). In particular, we have seen no way to choose U_{k+1} so as to maintain finite termination on a quadratic function or m-step quadratic convergence on general $f(x)$. However, these methods perform well in the experiments of Section 5, and clearly merit further consideration.

Temporary step update algorithms

A way to preserve the progress made on the Hessian approximation by the finite difference updates while using the Hessian information along s_k is to only *temporarily* update B_k along the step direction. More specifically, we can modify each of the algorithms (UP), (UB), and (CB), to obtain corresponding algorithms designated (UPT), (UBT), and (CBT), as follows. In step 9), B_{k+1} is updated from B_k along the finite difference directions by (2.5) or (2.4). Then, in step 3), we either calculate

$$\bar{B}_{k+1} = B_{k+1} - \frac{B_{k+1}s_k s_k^T B_{k+1}}{s_k^T B_{k+1} s_k} + \frac{y_k y_k^T}{s_k^T y_k},$$

for (UBT) and (CBT), or

$$\bar{B}_{k+1} = B_{k+1} + \frac{(y_k - B_{k+1}s_k)s_k^T + s_k(y_k - B_{k+1}s_k)^T}{s_k^T s_k} - \frac{s_k^T(y_k - B_{k+1}s_k)s_k s_k^T}{(s_k^T s_k)^2},$$

for (UPT), where $y_k = g_{k+1} - g_k$. We then use $\bar{B}_{k+1}$ to compute d_k exactly as B_{k+1} is used in the corresponding algorithm without the temporary step update. The "T" in the designation for these algorithms indicates the fact that the matrices $\bar{B}_{k+1}$ obtained through the step direction update are "temporary" in that they are only used for the next step computation and then never referenced again; i.e., B_{k+2} is calculated from B_{k+1}, not from $\bar{B}_{k+1}$. This idea was suggested to us by a related method of Li (1986) for nonlinear equations.

This approach has the advantage that the theoretical properties of the unmodified algorithms are preserved, since the step update only affects the next step, and is not incorporated into B_k. Thus, as we will show, Algorithm 2.1(UPT) and Algorithm 2.1(CBT) maintain their m-step Q-quadratic and 1-step Q-superlinear convergence rates. Also, the addition of the temporary step update provides significantly improved performance over the pure finite difference algorithms in practice, as seen in Section 5. This is perhaps due to the observation that, in practice, successive steps in optimization algorithms often tend to lie along roughly the same direction, so that in only using the most recent step direction in updating the matrix to be used to compute the next step, most of the relevant step information is retained.

3. Convergence Analysis.

In this section we analyze the local convergence properties of two of the algorithms discussed in the previous section, namely Algorithm 2.1(CB) and Algorithm 2.1(CBT). We also discuss some of the theoretical properties of the other algorithms from Section 2. For simplicity we assume that the finite difference values used are exact, i.e. that each $g_{k+1} = g(x_{k+1})$, and that $V_{k+1} = H(x_{k+1})U_{k+1}$ at step 7) of

Algorithm 2.1.

The first two algorithms in Section 2, parallel versions of the BFGS method and Newton's method, differ from the standard sequential methods only in that the new function value and (some of) the function evaluations for the new finite difference derivative values are evaluated concurrently. The sequence of iterates produced, and hence the convergence properties, are identical to those of the corresponding sequential methods.

The first new algorithm discussed in Section 2, Algorithm 2.1(UP), simply overwrites q rows and columns of the Hessian with accurate information at each iteration. Thus it clearly is m-step Q-quadratically and 1-step Q-superlinearly convergent, and terminates with the exact solution in at most m steps if f is a quadratic function. It is not to be expected that any of these properties apply to Algorithm 2.1(UB), which as we mentioned is included only to make some computational points in Section 5.

The next algorithm, Algorithm 2.1(CB), is not as easy to analyze because the update directions are block conjugate only asymptotically. It is straightforward to show that this again leads to m-step termination on quadratics, since the BFGS update is invariant under linear transformations. The following analysis shows that this method, like Algorithm 2.1(UP), is m-step Q-quadratically and 1-step Q-superlinearly convergent.

We now state the standard assumptions under which we will prove the two theorems of this section.

Assumptions 3.1.

Let S be an open subset of R^n and suppose that $f(x)$, $g(x)$, and $H(x)$ are continuous on S. Let $x_* \in S$ be a strict local minimum of $f(x)$. Assume that $H(x)$ is Lipschitz continuous in some neighborhood of x_*.

To prove our m-step Q-quadratic convergence results, it will be necessary to show that the error in the approximate Hessian at an iterate is of the order of the errors in the previous m iterates. In showing this, the following two definitions will be useful.

Definition

Given sequences $\{x_k\}$ and $\{B_k\}$, let

$$\delta_k = \max\{\, \| x_{k+j} - x_* \| : 1 \le j \le m \}$$

and

$$\kappa_k = \max\{\kappa(B_{k+j}) : 1 \le j \le m\}\,.$$

The following lemma shows that if m steps are taken, with approximate Hessians that are not too ill-conditioned, and the iterates remain close enough to x_*, then the error in the mth approximate Hessian is of the order of the errors in the iterates.

Lemma 3.1.

Let $\{x_k\}$ and $\{B_k\}$ be generated by Algorithm 2.1(CB), and suppose that $f(x)$ and x_* satisfy Assumptions 3.1. Then for any $M > 0$, there is an $r > 0$ and $c > 0$ such that for any k, if $\delta_k < r$ and $\kappa_k \le M$, then $\| B_{k+m} - H_* \| \le c\,\delta_k$.

Proof. Let $M > 0$. Since $H(x)$ is Lipschitz continuous on some neighborhood of x_* and H_* is positive definite, there is an $r > 0$ and an $M_1 > 0$ such that if $\| x - x_* \| < r$ and $\| \bar{x} - x_* \| < r$ then $\lambda_1(H(x)) \ge \frac{1}{M_1}$, $\lambda_n(H(x)) \le M_1$, and $\| H(x) - H(\bar{x}) \| \le M_1 \| x - \bar{x} \|$.

Consider any k, and assume that $\delta_k < r$ and $\kappa_k \le M$. In this proof for notational convenience and clarity, we omit the "$k+$" expressions in all subscripts, e.g. we write "B_{j+1}" for "B_{k+j+1}". Also, define $H_j = H(x_j)$.

Consider $1 \le j \le m-1$. By the BFGS update formula,

$$B_{j+1} = B_j - B_j U_{j+1}(U_{j+1}^T B_j U_{j+1})^{-1} U_{j+1}^T B_j + H_{j+1} U_{j+1}(U_{j+1}^T H_{j+1} U_{j+1})^{-1} U_{j+1}^T H_{j+1} .$$

Thus, for $1 \le i \le j$,

$$B_{j+1}U_i - H_* U_i = B_j U_i - H_* U_i -$$

$$B_j U_{j+1}(U_{j+1}^T B_j U_{j+1})^{-1}(U_{j+1}^T B_j U_i - U_{j+1}^T H_* U_i + U_{j+1}^T H_* U_i - U_{j+1}^T H_i U_i) +$$

$$H_{j+1} U_{j+1}(U_{j+1}^T H_{j+1} U_{j+1})^{-1}(U_{j+1}^T H_i U_i + U_{j+1}^T (H_* - H_i) U_i) ,$$

since $U_{j+1}^T H_i U_i = 0$. Thus, since $\| U_l \| = 1$ for all l,

$$\| B_{j+1}U_i - H_* U_i \| \le \| B_j U_i - H_* U_i \| + \| B_j U_i - H_* U_i \| \kappa(B_j)$$

$$+ \| H_i - H_* \| \kappa(B_j) + \| H_{j+1} - H_i \| \kappa(H_{j+1})$$

$$\le \| B_j U_i - H_* U_i \| (1+M) + M_1 \| x_i - x_* \| M + M_1 \| x_{j+1} - x_i \| M_1$$

$$\le (1+M) \| B_j U_i - H_* U_i \| + M M_1 \delta_0 + 2nM_1^2 \delta_0 \le L (\| B_j U_i - H_* U_i \| + \delta_0) ,$$

where $L = \max\{1+M , MM_1 + 2nM_1^2\}$. Thus, it is easy to show by induction that for $1 \le i \le m-1$,

$$\| B_m U_i - H_* U_i \| \le L^{m-i} \| B_i U_i - H_* U_i \| + (\sum_{q=1}^{m-i} L^q)\delta_0 .$$

Also, for $1 \le i \le m$, $B_i U_i = H_i U_i$ so that

$$\| B_i U_i - H_* U_i \| = \| B_i U_i - H_i U_i + H_i U_i - H_* U_i \|$$

$$= \| H_i U_i - H_* U_i \| \le M_1 \| x_i - x_* \| \le \delta_0 M_1 .$$

Thus, with $c_1 = L^m (M_1 + m)$, $\| B_m U_i - H_* U_i \| \le c_1 \delta_0$ for $1 \le i \le m$.

Since U_{i+1} is chosen orthogonal to $H_l U_l$, for $1 \le l \le i$, if we let U be the $n \times n$ matrix obtained by concatenating the matrices U_i, for $1 \le i \le m$, let V be the matrix obtained by concatenating the matrices $H_i U_i$, for $1 \le i \le m$, and let $D = U^T V$, then D is block diagonal, with diagonal blocks $U_i^T H_i U_i$. Further, for each i,

$$\| (U_i^T H_i U_i)^{-1} \| \leq \frac{1}{\lambda_1(H_i)} \leq M_1 ,$$

since $\| U_i \| = 1$, so $\| D^{-1} \| \leq M_1$. Thus,

$$\| U^{-1} \| = \| D^{-1} V^T \| \leq \| D^{-1} \| \, \| V^T \| \leq M_1 \| V^T \| .$$

Note that

$$\| V^T \| = \max_{\| x \| = 1} \| V^T x \| \leq n^{1/2} M_1 ,$$

since for any x with $\| x \| = 1$, if we let $v = H(x_l)u$ be a typical column of V, where $\| u \| = 1$, and $1 \leq l \leq m$, then $v^T x \leq \| u \| \, \| H(\xi) \| \, \| x \| \leq M_1$. So, $\| U^{-1} \| \leq n^{1/2} M_1^2$.

Now, each column of $(B_n - H_*)U$ has norm less than $c_1 \delta_0$, so clearly there is a constant c_2 such that $\| (B_n - H_*)U \| \leq c_2 \delta_0$. Hence,

$$\| B_n - H_* \| = \| (B_n - H_*)UU^{-1} \| \leq \| (B_n - H_*)U \| \, \| U^{-1} \| \leq c_2 n^{1/2} M_1^2 \delta_0$$

and the desired result follows with $c = c_2 n^{1/2} M_1^2$. □

In the proof of our main local convergence result below we make the strong assumption that x_* is a strict global minimum of $f(x)$. We do this in order to rule out the possibility of taking an unreasonably long step during the first m iterations. Alternatively, we could prove m-step Q-quadratic convergence under the assumption that B_1 is sufficiently close to H_*.

Theorem 3.1.

Suppose that $f(x)$ and x_* satisfy Assumptions 3.1. Suppose in addition that x_* is a strict global minimum of $f(x)$. Then for any positive definite matrix B_1, there is an $\varepsilon > 0$ such that if $\| x_1 - x_* \| < \varepsilon$, then the sequence $\{x_k\}$ generated by Algorithm 2.1(CB) converges m-step Q-quadratically and 1-step Q-superlinearly to x_*.

Proof.

Let B_1 be a given positive definite matrix.

We first obtain a neighborhood in which $H(x)$ is well-behaved and show that in this neighborhood, the condition numbers $\kappa(B_1), \ldots, \kappa(B_m)$ are bounded.

Since $H(x)$ is Lipschitz continuous on some neighborhood of x_* and H_* is positive definite, there are constants r_1 and γ such that if $\|x - x_*\| < r_1$, then $\lambda_1(H(x)) > 0$, $\|H(x)\| \le 2\|H_*\|$, and $\|H(x)^{-1}\| \le 2\|H_*^{-1}\|$, and if also $\|\bar{x} - x_*\| < r_1$, then $\|H(x) - H(\bar{x})\| \le \gamma\|x - \bar{x}\|$. Define

$$M_1 = \min\{\gamma, 2\|H_*\|, 2\|H_*^{-1}\|, \|B_1\|, \|B_1^{-1}\|, 1\} .$$

Let $\eta_1 > 0$ be such that if $\|B - H_*\| < \eta_1$ for a matrix B, then $\|B^{-1}\| \le M_1$ and $\|B\| \le M_1$. Let $N_1 = \{x \in R^n : \|x - x_*\| < r_1\}$.

It is easy to show, in similar fashion to the argument in Fletcher (1970), that if H and B are positive definite matrices, U has q independent columns, and

$$B_+ = B - BU(U^TBU)^{-1}U^TB + HU(U^THU)^{-1}U^TH ,$$

then

$$\lambda_1(B_+) \ge \min\{\lambda_1(H), \frac{\lambda_1(H)}{\lambda_n(H)}\lambda_1(B)\}$$

and

$$\lambda_n(B_+) \le \max\{\lambda_n(H), \frac{\lambda_n(H)}{\lambda_1(H)}\lambda_n(B)\} .$$

Further, if $H = H(\xi)$ for some $\xi \in N_1$, then clearly

$$\lambda_1(B_+) \ge \min\{\frac{1}{M_1}, \frac{1}{M_1^2}\lambda_1(B)\}$$

and

$$\lambda_n(B_+) \le \max\{M_1, M_1^2 \lambda_n(B)\} .$$

So, a trivial induction shows that if $\|B_{k+1}\| \le M_1$, $\|B_{k+1}^{-1}\| \le M_1$, and $\delta_k \le r_1$ then for $1 \le j \le m$, $\|B_{k+j}\| \le M_1^{3j}$ and $\|B_{k+j}^{-1}\| \le M_1^{3j}$, and so $\kappa_k \le M_1^{6m}$. Let $M_2 = M_1^{6m}$.

We now construct a neighborhood N_2 contained in N_1 in which Lemma 3.1 applies, and show that the iterates remain in this neighborhood. By Lemma 3.1, there are r and c such that for any k, if $\delta_k < r$ and $\kappa_k \le M_2$, then $\|B_{k+m} - H_*\| \le c\,\delta_k$. Let $r_2 = \min\{r_1, r, \frac{\eta_1}{c}\}$. Take

$$F = \min\{f(x) : r_2 \le \|x - x_*\|\} ,$$

and let $N_2 = \{x \in S : f(x) < F\}$. Then, since x_* is the strict global minimum of $f(x)$ and condition (2.1) is satisfied, if $x_1 \in N_2$ then for all k, $x_k \in N_2$. Thus, $\delta_k < r_2 \le r_1$ for all k. Now, since $\|B_1\| \le M_1$ and $\|B_1^{-1}\| \le M_1$, $\kappa_0 \le M_2$, and so $\|B_m - H_*\| \le c\,\delta_0 \le c r_2 < \eta_1$. So, $\|B_m\| \le M_1$ and $\|B_m^{-1}\| \le M_1$. Hence, by a simple induction, we have that $\|B_j\| \le M_2$ and $\|B_j^{-1}\| \le M_2$ for all j, and $\|B_{km} - H_*\| \le c\,\delta_{k(m-1)}$ for all k.

Thus, $\{x_j\}$ is contained in N_2 and $\kappa(B_j) \le M_2$ for all j, so Lemma 2.1 of Byrd and Nocedal (1986) implies that $\{x_j\}$ converges to x_*, since

$$\frac{-g_k^T s_k}{\|g_k\| \, \|s_k\|} = \frac{g_k^T B_k^{-1} g_k}{\|g_k\| \, \|B_k^{-1} g_k\|} \ge \frac{1}{\kappa(B_k)} .$$

We have shown that for all k, $\delta_k \le r_2 < r$ and $\kappa_k \le M_2$, so for any k, $\|B_{k+m} - H_*\| \le c\,\delta_k < \eta_1$. Also, by Theorem 6.4 of Dennis and Moré (1977) it follows that for all large k, $\mu_k = 1$.

Consider any k with $\mu_k = 1$. Then for some ξ_{k+m} between x_{k+m} and x_{k+m+1},

$$\|x_{k+m+1} - x_*\| = \|x_{k+m} - x_* - B_{k+m}^{-1} g_{k+m}\|$$

$$\le \|B_{k+m}^{-1}\| \, \|B_{k+m}(x_{k+m} - x_*) - H(\xi_{k+m})(x_{k+m} - x_*)\|$$

$$\leq M_1 (\| B_{k+m} - H_* \| + \| H(x_{k+m}) + H_* \|) \| x_{k+m} - x_* \|$$

$$\leq M_1 (c \delta_k + M_1 \| \xi_{k+m} - x_* \|) \| x_{k+m} - x_* \| .$$

Since $g_{k+m} = g(x_*) + H(\bar{\xi}_{k+m})(x_{k+m} - x_*)$ for some $\bar{\xi}_{k+m}$ between x_* and x_{k+m},

$$\| \xi_{k+m} - x_* \| \leq \| x_{k+m} - x_* \| + \| B_{k+m}^{-1} g_{k+m} \| \leq (1 + M_1^2) \| x_{k+m} - x_* \| .$$

So, we have that

$$\| x_{k+m+1} - x_* \| \leq M_1 c \delta_k \| x_{k+m} - x_* \| + M_1^2 (1 + M_1^2) \| x_{k+m} - x_* \|^2$$

$$\leq (M_1 c + M_1^2 (1 + M_1^2)) \delta_k \| x_{k+m} - x_* \| .$$

Thus, by definition, $\{x_k\}$ converges to x_* at an m-step Q-quadratic rate, and since δ_k converges to 0, clearly also $\{x_k\}$ converges to x_* at a 1-step Q-superlinear rate. □

Theorem 3.2 shows that Algorithm 2.1(CBT), which adds a temporary BFGS update of the standard secant information to the partial Hessian information update of Algorithm 2.1(CB), has the same local convergence properties as Algorithm 2.1(CB). This is of interest because Algorithm 2.1(CBT) turns out to be the better of the two methods in practice. The technique of proof is related to one used by Li (1986) for a related temporary update method for solving systems of nonlinear equations.

Theorem 3.2.

Suppose that $f(x)$ and x_* satisfy Assumptions 3.1. Suppose in addition that x_* is a strict global minimum of $f(x)$. Then for any positive definite matrix B_1, there is an $\varepsilon > 0$ such that if $\| x_1 - x_* \| < \varepsilon$, then the sequence $\{x_k\}$ generated by Algorithm 2.1(CBT) converges m-step Q-quadratically and 1-step Q-superlinearly to x_*.

Proof.

We prove this result by indicating the necessary modifications to the proof of Theorem 3.1.

First, note that the proof of Lemma 3.1 need not be changed at all in order to apply to Algorithm 2.1(CBT), since the steps $\{s_k\}$ do not enter in to the proof, and the matrices $\{B_k\}$ are unchanged.

Next, let $\bar{B}_j$ be the approximate Hessian with which the step is computed, i.e.

$$\bar{B}_j = B_j - \frac{B_j s_{j-1} s_{j-1}^T B_j}{s_{j-1}^T B_j s_{j-1}} + \frac{y_{j-1} y_{j-1}^T}{s_{j-1}^T y_{j-1}} .$$

Clearly $\|\bar{B}_j\| \le M_1^3 \|B_j\|$ and $\|\bar{B}_j^{-1}\| \le M_1^3 \|B_j^{-1}\|$, so if we take $M_2 = M_1^{6(m+1)}$ and replace B_j by $\bar{B}_j$ throughout in the proof of Theorem 3.1, we still obtain the fact that $\{x_j\}$ converges to x_*, and $\|B_{k+m} - H_*\| \le c\,\delta_k$.

Thus, by Theorem 3.2 of Broyden, Dennis, and Moré (1973), it is easy to see that there is a constant c_1 such that for all k, $\|\bar{B}_{k+m} - H_*\| \le c_1 \delta_k$. Thus, since $\{\delta_k\}$ converges to 0, we have that for all large k, $c_1 \delta_k < \eta_1$, and we can finish the proof with $\bar{B}_{k+m}$ in place of B_{k+m} and c_1 in place of c. □

A similar analysis could be used to show that Algorithm 2.1(UPT) is also m-step Q-quadratically and 1-step Q-superlinearly convergent.

4. Computer Implementation.

In this section we discuss our computer implementation of the algorithms described in Section 2. These algorithms are similar to well-known sequential algorithms in most respects, so we will concentrate on the aspects that result from a parallel processing context.

We have considered algorithms that compute q extra gradients at each point, but at the present time we have only implemented a version that uses 1 extra gradient at each point. Because this is the case that is reported on in Section 5, and because this case is simpler to describe and understand, we will restrict

ourselves to this case in this section.

Since we assume that $g(x)$ and $H(x)$ are not available analytically, but must instead be approximated using function values, we need to describe how we approximate the various derivative quantities by finite differences. First, at a point x, we approximate $g(x)$ by the usual finite difference formula (1.2), where

$$h_i = (\text{machine precision})^{1/2} \max\{|x_i|, 1\} .$$

Then, we approximate the extra Hessian information $v_{k+1} = H(x_{k+1})u_{k+1}$ by

$$v_{k+1} = \frac{g(x_{k+1} + \eta u_{k+1}) - g(x_{k+1})}{\eta} ,$$

where $\eta = (\text{machine precision})^{1/4}$, and the new gradient is again computed by (1.2) and requires $n+1$ new function values. It can be shown that this choice of the stepsize η tends to optimize the trade-off between truncation error and rounding error in V_{k+1}.

We now consider the issue of computing the finite difference update direction u_{k+1}. Our algorithms either use unit directions or approximately conjugate directions.

The choice of unit directions is straightforward. We simply cycle through the directions $e_{j \pmod n}$, for $j = 1, 2, \ldots$. However, when we are doing the BFGS update, to preserve positive definiteness we need $u_{k+1}^T v_{k+1} > 0$. Therefore, if the test

$$u_{k+1}^T v_{k+1} > (\text{machine precision})^{1/2} \; \|u_{k+1}\| \; \|v_{k+1}\|$$

is not satisfied then we do not perform the BFGS update at this step, and we continue to use the same direction at successive iterates until the test is satisfied.

We now consider our method of computing the approximately conjugate finite difference update directions. Let $\bar{V}_k$ be the matrix with the $n-1$ columns $H(x_{k-j})u_{k-j}$, for $j = 0, \ldots, n-2$. We do a QR

factorization of $\bar{V}_k$, obtaining $\bar{V}_k = Q_k R_k$, where Q_k is an $n \times n$ orthogonal matrix and R_k is an $n \times t$ upper triangular matrix, where t is the number of columns of $\bar{V}_k$ that were included in the factorization. In the course of the factorization, if a column of $\bar{V}_k$ makes too small of an angle with the subspace spanned by the previously included columns of $\bar{V}_k$, then it is not included in the factorization. Then, we take u_{k+1} to be column $t+1$ of Q_k. Thus, u_{k+1} is orthogonal to all the columns of $\bar{V}_k$ that were included in the factorization. Note that if $\bar{V}_k$ has rank $n-1$, then u_{k+1} is orthogonal to each $H(x_{k-j})u_{k-j}$, for $j = 0, \ldots, n-2$. Next, we let $v_{k+1} = H(x_{k+1})u_{k+1}$. If B_{k+1} is updated by the PSB, then we obtain $\bar{V}_{k+1}$ by shifting the columns of $\bar{V}_k$ one to the right and taking column 1 to be v_{k+1}. If B_{k+1} is updated by the BFGS, and the update along u_{k+1} succeeded, then we determine $\bar{V}_{k+1}$ as for the PSB update. If the update was not done, because $u_{k+1}^T v_{k+1} \leq 0$, then $\bar{V}_{k+1} = \bar{V}_k$. Thus, we continue to use a finite difference direction until the update along it is successful. Initially, we take $\bar{V}_1$ to be the first $n-1$ columns of the identity matrix. Note that the theory of Section 2 is local, and in that context the matrices $\bar{V}_k^{-1}$ can be uniformly bounded and $H(x_{k+1})$ is always positive definite, so the BFGS updates always succeed.

We now discuss the determination of B_1 in the various algorithms. All the algorithms, except of course for Newton's method, take the initial Hessian to be the identity. Then, before the first step direction is calculated, those algorithms that perform updates along finite difference directions update the identity matrix to a matrix $\bar{B}_1$, by equation (2.4) or (2.5), using the finite difference information $H(x_1)u_1$. This information is available at no cost since the algorithm has to evaluate $f(x_1)$ and $g(x_1)$, so the extra gradient information $H(x_1)u_1$ might as well be calculated concurrently and used to improve the initial Hessian approximation. Then, $\bar{B}_1$ is used to compute s_1. After obtaining x_2 and $g(x_2)$, we finally obtain B_1 by the scaling

$$B_1 = \frac{s_1^T \bar{B}_1 s_1}{s_1^T y_1} \bar{B}_1 ,$$

where $y_1 = g_2 - g_1$. The effect of this scaling, first suggested by Oren (1974), is to ensure that $s_1^T B_1 s_1 = s_1^T y_1$, which is desirable since we would like to have $B_1 s_1 = y_1$. Note that our implementation of the usual BFGS algorithm also does this initial scaling of B_1. We experimented with other strategies for determining B_1 in our new algorithms, but found no uniform improvement over the strategy described above.

The implementation of these algorithms uses code for the linesearch, perturbed Cholesky factorization, and stopping conditions as described in Dennis and Schnabel (1983). We implemented the version of their linesearch method that obtains a step satisfying conditions (2.1) and (2.2).

5. Computational Results.

We now present and discuss some computational results comparing the performance on a set of standard test problems of the algorithms we have described. As mentioned in Section 4, so far we have only tested the versions of our new algorithms that utilize $2(n+1)$ processors (i.e. one extra finite difference gradient).

Our test set is taken from the standard set of small dimensional problems in Moré , Garbow, and Hillstrom (1981). We omitted some of their functions because the problems were either badly scaled or were not solved by any of our methods due to floating point arithmetic overflows. The 15 functions in our test set are listed in abbreviated form in Table A.1 in the Appendix, in the column labeled "func." Each function was tested with three choices of x_1, namely the standard starting point given by Moré, Garbow, and Hillstrom (1981), multiplied by 1, 10, and 100. The column in Table A.1 labeled "sp" contains the multiple of the standard starting point that was used in the corresponding test problem. Thus, our test set consists of 42 problems, since Watson's function was only tested with one starting point, the zero

vector, and the Chebyquad function from the farthest point was not solved by any of our methods.

The stopping conditions used in the code are as described in Dennis and Schnabel (1983). The algorithms successfully terminated when either the relative size of the gradient was less than 10^{-5} or the linesearch failed to find a lower point than the current iterate while backtracking. The algorithms failed to solve a problem when either the iteration limit of 500 iterations was reached or a floating point arithmetic overflow occurred. In Table A.1, an overflow on a problem for a method is indicated by "***" listed for the number of iterations and for the number of failed trial points, while "—" in these same locations indicates that the method reached the iteration limit on the problem. The tests were performed on a sequential machine, a VAX 11/780, using double precision arithmetic.

For each problem that was solved, we recorded the number of steps required and the number of trial points at which the linesearch conditions (2.1) and (2.2) were not satisfied. These numbers are recorded in Table A.1. Note that the number of steps and the number of failed trial points is enough information to simulate the performance of our algorithms on a parallel machine with $2(n+1)$ processors, if we assume that only the time for function and derivative evaluations is relevant. Define a cycle of parallel function evaluations, or "f-cycle," to be a step in Algorithm 2.1 at which up to $2(n+1)$ function evaluations are performed in parallel. Thus, on our simulated parallel machine, an f-cycle takes the same amount of time as one function evaluation. For all of our algorithms except Newton's method, the number of f-cycles required to solve a problem is simply the number of points x at which $f(x)$ is calculated in steps 1) and 7) in Algorithm 2.1. This is clear, since all the function evaluations required for the derivative approximations are performed in parallel with the evaluation of the function at each trial point, or are done in parallel with the evaluation of $f(x_1)$. Note that each trial point in the linesearch is either an iterate, if it satisfies conditions (2.1) and (2.2), or is a failed trial point, if it does not satisfy these conditions. Thus, for all our algorithms except Algorithm 2.1(N),

$$\textit{number of } f\textit{-cycles} = 1 + \textit{number of iterations} + \textit{number of failed trial points} .$$

Newton's method is somewhat different, since more than one f-cycle is required to compute the Hessian at successful trial points. In parallel with the computation of $f(x)$ and $g(x)$ at trial points, the extra $n+1$ processors can clearly be used to compute $n+1$ function values for the finite difference approximation to $H(x)$. Then, if the trial point satisfies conditions (2.1) and (2.2), the remaining $\frac{n^2+n-2}{2}$ function values for the finite difference Hessian approximation must be computed. This requires $\left\lceil \frac{n^2+n-2}{4(n+1)} \right\rceil$ f-cycles. Hence, for Algorithm 2.1(N),

$$\textit{number of } f\textit{-cycles} = (\,1 + \textit{number of iterations}\,)(\,1 + \left\lceil \frac{n^2+n-2}{4(n+1)} \right\rceil\,)$$
$$+ \textit{number of failed trial points} .$$

Thus, for all of our algorithms we can compute the simulated number of f-cycles needed to solve each problem on a parallel machine with $2(n+1)$ processors, using the raw data given in Table A.1. If function evaluation is expensive, this is a very close indication of the total cost of solving the problem.

We now describe our statistical comparisons of our algorithms. We present a statistical comparison of all the algorithms, as well as a number of statistical summaries comparing the relative performance of pairs of the leading methods. Each of these pairwise summaries is in the format of Table 5.1. The column headings give the abbreviated designations of the two algorithms being compared, and the rows are calculated as follows. The row labeled "# solved" contains the number of problems out of the 42 in the test set on which the method successfully terminated, while the rows labeled "# overflow" and "# hit itnlim" contain the number of problems on which each method failed, respectively, from floating point arithmetic overflow and by reaching iteration 500. The row labeled "# compared" contains the number of problems that were solved by both methods. The comparative statistics in the last two rows of the tables are computed only over this set of comparison problems. The row labeled "Ave. score" is calculated as

follows. For each comparison problem, the method which required fewer f-cycles is assigned a score of "1," while the other method is assigned a score equal to the number of f-cycles it required divided by the number of f-cycles required by the better method. Then, the average of the scores for each method over the set of comparison problems is recorded in the row labeled "Ave. score." For each comparison problem, if the score of a method is less than or equal to 1.1, then that method is counted as "# best" for the problem, and the total of such problems for each method is recorded in the row labeled "# best."

Table 5.1
$p = 2(n+1)$

Algorithm:	(S)	(N)
# solved:	36	37
# overflow:	4	4
# hit itnlim:	2	1
# compared:	34	
# best:	19	20
Ave. score:	1.42	2.09

We first consider the comparative performance of Newton's method and the BFGS method. It seems clear from Table 5.1 that the BFGS method is somewhat superior to Newton's method when $2(n+1)$ processors are available. Table 5.2, on the other hand, compares these two methods under the assumption that p is large enough that the entire finite difference Hessian approximation can be calculated concurrently with evaluation of the function and gradient values. It shows that the extra Hessian information available in Newton's method substantially reduces the number of iterations required in this case. Together, these two tables show that it certainly is reasonable to consider the type of algorithms that we are discussing in this paper, when the number of processors is large enough to allow an extra gradient evaluation at each trial point but not large enough to allow evaluation of the finite difference Hessian in one f-cycle, because Newton's method is not optimal, but the Hessian information does seem to

help.

Table 5.2 $p \geq (n^2+5n+2)/2$		
Algorithm:	(S)	(N)
# solved:	36	37
# overflow:	4	4
# hit itnlim:	2	1
# compared:	34	
# best:	8	27
Ave. score:	3.07	1.36

We now compare the relative performance of all our algorithms by the following analysis technique. Define M_i to be the number of f-cycles required for algorithm M to solve problem i. If the algorithm fails, either due to overflow or by exceeding the iteration limit, then $M_i = \infty$. We then define the average performance a_i for problem i to be the median of the values M_i, over all methods M, except that if the median is ∞ then we instead take a_i to be the largest M_i that is not ∞. Next, define the performance P_M of method M

$$P_M = \sum_{M_i \leq a_i} \frac{M_i}{a_i} + \sum_{M_i > a_i} (2 - \frac{a_i}{M_i}),$$

where if $M_i = \infty$ then $\frac{a_i}{M_i} = 0$. This measure tends to compress the performance measure of all the methods but is reasonably good at ordering them.

Table 5.3 shows the performance of each algorithm as measured by the above technique.

Table 5.3

Algorithm	S	N	UP	UB	CB	UPT	UBT	CBT	UPS	UBS	CBS
Performance	1.08	1.15	1.16	1.69	1.26	1.14	1.21	1.07	0.95	0.98	0.90

We can make some interesting observations from this table. First, we note that Algorithms 2.1(UP), (UB), and (CB), which omit the standard secant equation, perform worse than the BFGS method, Algorithm 2.1(S), even though they use some finite difference Hessian information. Also, note that Algorithm 2.1(UB) performs much worse than Algorithm 2.1(CB), which indicates the importance of choosing the finite difference update directions to be approximately conjugate when using the BFGS update to insert finite difference Hessian information. Next, we see that the addition of the temporary step update is clearly worthwhile, since each of Algorithms 2.1(UPT), (UBT), and (CBT) performs better than the corresponding algorithm without the temporary step update. However, two of these methods still perform worse than the BFGS method, and Algorithm 2.1(CBT) performs about the same as the BFGS method. Finally, we observe that the methods that use alternating step and finite difference updates, Algorithms 2.1(UPS), (UBS), and (CBS), perform the best of our algorithms. They perform significantly better than the BFGS method.

In Table 5.4 and Table 5.5 we give pairwise comparisons of the BFGS method, Algorithm 2.1(S), with the three best of our new methods, Algorithms 2.1(UPS), (UPS), and (CBS). These statistics confirm the conclusion that the alternating update methods perform better than the BFGS method. Also, it is interesting to note that the choice of approximately conjugate finite difference update directions that yielded such an improvement of Algorithm 2.1(CB) over Algorithm 2.1(UB) does not give a similar improvement for these methods, since Algorithms 2.1(UBS) and (CBS) perform very similarly. This is probably linked to the fact that the m-step quadratic convergence of methods (CB) and (CBT), which

depended on using conjugate rather than orthogonal directions, is destroyed by making permanent step updates. Finally, we see that the conventional wisdom that the BFGS update is in some sense superior to the PSB update is supported here, since Algorithms 2.1(UBS) and (CBS) perform somewhat better than Algorithm 2.1(UPS).

Table 5.4

Algorithm:	UPS	S	UBS	S	CBS	S
# solved:	38	36	33	36	36	36
# overflow:	3	4	7	4	6	4
# hit itnlim:	1	2	2	2	0	2
# compared:	33		32		34	
# best:	21	14	27	12	24	12
Ave. score:	1.39	3.22	1.23	1.66	1.45	1.75

Table 5.5

Algorithm:	UBS	UPS	CBS	UPS	UBS	CBS
# solved:	33	38	36	38	33	36
# overflow:	7	3	6	3	7	6
# hit itnlim:	2	1	0	1	2	0
# compared:	30		33		33	
# best:	21	12	26	16	20	20
Ave. score:	1.23	1.66	1.46	1.73	1.14	1.15

Tables 5.4 and 5.5 indicate that our best new methods have achieved about a 30% improvement over the BFGS method when there are twice the number of processors needed by the BFGS method. This is not a perfect utilization of processors but may be about as well as one can do on these small problems, especially considering the comparisons in Table 5.2 which show a fairly small improvement by Newton's method over the BFGS method on these problems.

6. Conclusions.

We have introduced three types of new algorithms that utilize extra function evaluations to obtain part of the finite difference Hessian at each iteration. The first type (Algorithms 2.1(UP), (UB), and (CB)) uses the finite difference Hessian information to update the Hessian approximation and omits the standard secant update that is made by methods such as the BFGS method. The second type (Algorithms 2.1(UPT), (UBT), and (CBT)) uses the finite difference Hessian information and makes a temporary standard secant update as well. The third type (Algorithms 2.1(UPS), (UBS), and (CBS)) makes updates both with the finite difference Hessian information and with the standard secant information at each iteration. Each algorithm type has three variants: using standard basis finite difference directions with PSB updates (first two letters UP) or BFGS updates (first two letters UB), or using conjugate finite difference directions with BFGS updates (first two letters CB).

We have shown m-step Q-quadratic and 1-step Q-superlinear convergence rates for Algorithms 2.1(CB) and (CBT); the same results clearly hold for Algorithms 2.1(UP) and (UPT). Here $m = \left\lceil \frac{n}{q} \right\rceil$, where q is the number of extra gradient evaluations available per iteration. Our experimental results with $q = 1$ show that of these algorithms, only Algorithm 2.1(CBT) performs roughly as well as the BFGS method. These algorithms may perform better than the BFGS method when $q > 1$ or on large dimensional problems; we plan to experiment with these cases.

Algorithms 2.1(UPS), (UBS), and (CBS) with $q = 1$ appear to perform better than the BFGS method. We have not been able to show m-step Q-quadratic convergence for these methods and suspect that they do not possess this property. However, they appear to be a promising approach to utilizing extra processors in solving unconstrained optimization problems. Also, we believe that these methods are 1-

step Q-superlinearly convergent. We intend to continue to experiment with these methods with $q > 1$ and on larger-dimensional problems, and to analyze their convergence properties.

References.

C. G. Broyden, J. E. Dennis, and J. J. Moré , "On the local and superlinear convergence of quasi-Newton methods", *Journal of the Institute of Mathematics and its Applications* 12(1973) 223-245.

R. H. Byrd, C. Dert, A. H. G. Rinnooy Kan, and R. B. Schnabel, "Concurrent stochastic methods for global optimization", Technical Report CU-CS-338-86, Department of Computer Science, University of Colorado at Boulder (Boulder, CO, 1986).

R. H. Byrd and J. Nocedal, "Global convergence of a class of quasi-Newton methods on uniformly convex problems", Technical Report #86-01-NAM-01, Department of Electrical Engineering and Computer Science, Northwestern University (Evanston, IL, 1986).

J. E. Dennis, Jr. and J. J. Moré , "Quasi-Newton methods, motivation, and theory", *SIAM Review* 19(1977) 46-89.

J. E. Dennis and R. B. Schnabel, Numerical methods for unconstrained optimization and nonlinear equations (Prentice-Hall, Englewood Cliffs, NJ, 1983).

R. Fletcher, "A new approach to variable metric algorithms", *Computer Journal* 13(1970) 317-322.

R. Fletcher, Practical methods of optimization, Vol. 1: Unconstrained optimization (John Wiley & Sons, New York, 1980).

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

G. Li, "Successive column correction algorithms for solving sparse nonlinear systems of equations", Technical Report 86-12, Department of Mathematics, Rice University (Houston, TX, 1986).

J. J. Moré , B. S. Garbow, and K. E. Hillstrom, "Testing unconstrained optimization software", *ACM Transactions on Mathematical Software* 7(1981) 17-41.

J. J. Moré and D. C. Sorensen, "Computing a trust region step", *SIAM Journal on Scientific and Statistical Computing* 4(1983) 553-572.

S. S. Oren, "On the selection of parameters in self-scaling variable metric algorithms", *Mathematical Programming* 7(1974) 351-367.

R. B. Schnabel, "Quasi-Newton methods using multiple secant equations", Technical Report CU-CS-247-83, Department of Computer Science, University of Colorado at Boulder (Boulder, CO, 1983).

R. B. Schnabel, "Concurrent function evaluations in local and global optimization", Technical Report CU-CS-345-86, Department of Computer Science, University of Colorado at Boulder (Boulder, CO, 1986).

Appendix.

Table A.1

*** = overflow, — = iteration limit

			method										
			S	N	UP	UB	CB	UPT	UBT	CBT	UPS	UBS	CBS
func	n	sp	iterations failures										
HELI	3	1	27	10	19	58	21	21	23	23	18	21	17
			7	6	7	9	4	6	20	22	6	6	5
TRIG	10	1	27	9	20	35	33	24	22	22	19	17	17
			1	0	17	12	11	10	1	8	12	0	2
ROSE	10	1	41	24	124	171	149	***	139	78	***	64	65
			11	8	124	115	206	***	25	38	***	11	30
ROSE	2	1	23	24	35	90	29	23	35	32	27	23	27
			6	8	15	7	8	10	13	13	9	4	8
SING	4	1	47	15	32	97	33	21	24	26	27	22	24
			20	0	2	34	4	1	3	6	0	3	4
SING	8	1	47	15	56	128	54	36	31	29	35	43	29
			20	0	3	78	8	23	6	8	1	5	6
BEAL	2	1	16	7	11	12	8	11	9	8	14	7	8
			2	1	2	1	1	1	2	2	2	1	1
WOOD	4	1	32	57	90	118	49	63	52	49	75	41	46
			9	38	24	42	15	36	19	19	17	13	17
CHEB	9	1	24	13	55	49	48	42	49	29	32	25	34
			6	8	9	5	20	24	6	8	4	3	8
GAUS	3	1	5	1	3	4	3	3	4	2	3	4	3
			2	0	3	3	1	2	2	1	3	3	1
BOX	3	1	38	16	10	61	9	10	15	12	9	12	9
			5	11	0	50	1	1	0	1	0	0	0
VAR	10	1	16	14	37	116	37	33	32	22	31	21	22
			7	0	15	115	6	5	28	1	0	4	1
WATS	9	1	95	***	266	—	129	***	309	35	175	56	34
			6	***	48	—	118	***	4	19	23	4	20
PEN1	10	1	151	30	—	251	138	***	112	69	122	101	78
			63	6	—	111	66	***	30	21	46	18	22
PEN2	10	1	22	99	42	423	111	***	163	263	21	121	154
			11	43	13	304	74	***	63	233	2	15	50
HELI	3	10	29	15	20	54	16	13	19	17	19	9	13
			7	4	5	12	4	4	11	9	4	2	4
TRIG	10	10	72	19	41	41	62	26	74	40	32	39	29
			10	6	2	5	24	6	99	19	2	12	5
ROSE	10	10	54	73	327	410	—	***	210	489	***	154	166
			19	36	510	451	—	***	59	305	***	51	67
ROSE	2	10	49	72	78	266	75	40	92	90	108	57	71
			17	34	47	184	24	20	39	43	47	10	19
SING	4	10	64	20	52	117	46	34	35	33	35	40	31
			11	0	4	43	8	5	12	9	1	9	8
SING	8	10	77	20	87	151	74	50	105	37	47	55	48
			30	0	1	74	11	47	23	11	0	8	14
BEAL	2	10	—	35	—	—	39	39	27	44	45	—	46
			—	22	—	—	24	6	224	17	18	—	26
WOOD	4	10	58	53	106	279	58	64	62	56	98	52	53
			7	28	40	655	15	26	13	20	19	13	15

CHEB	9	10	***	***	320	***	***	176	***	***	153	***	***
			***	***	52	***	***	83	***	***	18	***	***
GAUS	3	10	19	1	1	15	***	1	15	17	1	***	13
			5	0	0	18	***	0	6	1	0	***	1
BOX	3	10	***	***	40	—	—	29	49	54	39	47	36
			***	***	31	—	—	33	11	12	28	14	19
VAR	10	10	41	17	89	140	127	***	26	24	66	23	24
			9	0	21	139	184	***	23	3	9	4	2
PEN1	10	10	171	43	—	491	127	—	256	91	149	115	93
			50	13	—	430	40	—	321	36	64	19	23
PEN2	10	10	570	105	455	***	***	58	***	***	122	***	***
			112	36	121	***	***	19	***	***	26	***	***
HELI	3	100	28	16	10	54	14	7	20	18	13	9	9
			7	4	3	12	6	3	4	15	3	5	5
TRIG	10	100	101	27	112	***	110	***	88	93	114	58	175
			46	10	89	***	55	***	36	79	82	26	53
ROSE	10	100	165	—	—	—	—	***	—	—	—	396	358
			31	—	—	—	—	***	—	—	—	105	225
ROSE	2	100	112	367	206	—	466	74	—	491	274	—	383
			41	249	101	—	455	25	—	253	216	—	255
SING	4	100	99	26	68	119	64	43	43	41	47	39	39
			16	0	5	50	37	3	11	25	4	11	16
SING	8	100	141	26	164	203	106	79	113	50	71	75	51
			18	0	79	172	20	94	41	14	7	16	20
BEAL	2	100	***	78	15	***	***	96	***	***	120	***	***
			***	45	14	***	***	31	***	***	41	***	***
WOOD	4	100	116	44	121	137	45	80	50	41	115	43	41
			16	6	44	14	16	21	7	13	17	12	9
GAUS	3	100	56	136	1	—	—	1	***	***	1	***	***
			34	254	0	—	—	0	***	***	0	***	***
BOX.	3	100	***	***	4	***	***	4	***	***	3	***	***
			***	***	0	***	***	0	***	***	0	***	***
VAR	10	100	59	26	***	250	66	***	37	48	262	66	35
			8	0	***	139	14	***	23	65	132	7	0
PEN1	10	100	185	44	—	—	169	—	107	91	170	111	91
			199	7	—	—	58	—	14	51	98	12	12
PEN2	10	100	—	106	***	***	***	***	***	***	***	***	***
			—	38	***	***	***	***	***	***	***	***	***

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CS-CU-361-87	5. MONITORING ORGANIZATION REPORT NUMBER(S) ARO 21453.9-MA
6a. NAME OF PERFORMING ORGANIZATION University of Colorado 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION U.S. Army Research Office
6c. ADDRESS (City, State and ZIP Code) Computer Science Department Campus Box 430 Boulder, CO 80309	7b. ADDRESS (City, State and ZIP Code) Post Office Box 12211 Research Triangle Park, NC 27709
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAAG-29-84-K-0140
8c. ADDRESS (City, State and ZIP Code) 11.-Title Using Parallel Function Evaluations To Improve Hessian Approximations For Unconstrained Optimization	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
12. PERSONAL AUTHOR(S) Richard H. Byrd, Robert B. Schnabel, Gerald A. Shultz	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 87/3/30 15. PAGE COUNT 38
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) parallel computation, unconstrained optimization, finite difference derivative, secant method

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper presents a new class of methods for solving unconstrained optimization problems on parallel computers. The methods are intended to solve small to moderate dimensional problems where function and derivative evaluation is the dominant cost. They utilize multiple processors to evaluate the function, (finite difference) gradient, and a portion of the finite difference Hessian simultaneously at each iterate. We introduce three types of new methods, which all utilize the new finite difference Hessian information in forming the new Hessian approximation at each iteration; they differ in whether and how they utilize the standard secant information from the current step as well. We present theoretical analyses of the rate of convergence of several of these methods. We also present computational results which illustrate their performance on parallel computers when function evaluation is expensive.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☑ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Jagdish Chandra	22b. TELEPHONE NUMBER (Include Area Code) 617/549-0641 22c. OFFICE SYMBOL

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

END

7-87

DTIC