Sequential and Parallel Methods for Unconstrained Optimization

Robert B. Schnabel

CU-CS-414-88 October 1988

Department of Computer Science
Campus Box 430
University of Colorado,
Boulder, Colorado, 80309 USA

This research was supported by AFOSR grant AFOSR-85-0251, ARO grant DAAL 03-88-K-0086, and NSF cooperative agreement CCR-870243.

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 reviews some interesting recent developments in the field of unconstrained optimization. First we discuss some recent research regarding secant (quasi-Newton) methods. This includes analysis that has led to an improved understanding of the comparative behavior of the BFGS, DFP, and other updates in the Broyden class, as well as computational and theoretical work that has led to a revival of interest in the symmetric rank one update. Second we discuss recent research in methods that utilize second derivatives. We describe tensor methods for unconstrained optimization, which have achieved considerable gains in efficiency by augmenting the standard quadratic model with low rank third and fourth order terms, in order to allow the model to interpolate some function and gradient information from previous iterations. Finally, we will review some work that has been done in constructing general purpose methods for solving unconstrained optimization problems on parallel computers. This research has led to a renewed interest in various ways of performing the linear algebra computations in secant methods, and to new algorithms that make use of multiple concurrent function evaluations.

1. Introduction

This paper reviews some interesting developments in the solution of unconstrained optimization problems over the last few years. The unconstrained optimization problem is

$$\text{given } f : R^n \to R \text{ , find } x_* \text{ for which } f(x_*) \le f(x) \text{ for all } x \in D, \tag{1.1}$$

where D is an open neighborhood containing x_*. We assume that the function f is at least twice continuously differentiable, even though the analytic derivatives may not be readily available. Our orientation is towards problems where the number of variables, n, is not too large, say $n \le 100$. Even if n is not large, it is often the case in practice that the evaluation of $f(x)$ is very expensive, so it is important that algorithms make as few evaluations of $f(x)$ and its derivatives as possible.

In Section 2 we give a very brief and superficial summary of the standard methods for solving unconstrained optimization problems, and their relative advantages and disadvantages. Readers familiar with unconstrained optimization may wish to skip this section. More extensive references can be found in various books, including Fletcher [1980], Gill, Murray, and Wright [1981], and Dennis and Schnabel [1983].

Section 3 discusses a number of recent interesting research developments in *secant methods* for unconstrained optimization. By secant methods, we mean methods that update an approximation to the Hessian matrix at each iteration, using only values of the gradient at current and previous iterates. The developments we discuss include experiments and analysis that have led to a better understanding of the comparative behavior of the BFGS, DFP, and other updates in the Broyden class, as well as new theoretical and experimental research related to the symmetric rank one update.

In Section 4 we review some recent research into *second derivative methods*, methods that utilize the analytic or finite difference value of the Hessian matrix at each iteration. We concentrate on tensor methods, a new class of methods that augment the standard quadratic model with low rank approximations to higher derivatives at each iteration. We briefly describe these methods in the contexts of both nonlinear equations and unconstrained optimization.

Finally, Section 5 summarizes some research in the fairly new field of parallel methods for unconstrained optimization. We concentrate on parallel secant methods. We briefly discuss both some ways to parallelize the linear algebra computations in these methods, and some approaches that make use of multiple concurrent function evaluations.

2. Background

Algorithms for solving the unconstrained optimization problem (1.1) are iterative. The basic framework of an iteration of the methods that are used when the number of variables is not too large is shown in Algorithm 2.1.

At the highest level, two aspects of Algorithm 2.1 require elaboration. The first is the method for calculating or approximating the Hessian matrix $\nabla^2 f(x_+)$. The second is the method for chosing the new iterate x_+. In this section, we give a very brief description of the main alternatives that are used in practice, and how they compare.

Methods for calculating or approximating the Hessian matrix can be divided into two classes, second derivative methods and secant methods. In second derivative methods, one calculatesthe Hessian matrix $\nabla^2 f(x)$, analytically or by finite differences, at each iteration. If finite differences are used, this costs either n additional evaluations of the gradient $\nabla f(x)$ or $n^2 + \frac{3n}{2}$ additional evaluations of the function $f(x)$ at each iteration.

In secant methods, one forms an approximation H_+ to the Hessian $\nabla^2 f(x_+)$ using only values of the gradient at the current and previous iterates. This approximation is chosen so that the model of $f(x)$ around the new iterate x_+,

---

Algorithm 2.1 -- Basic Unconstrained Optimization Iteration

given current iterate x_c , $f(x_c)$,
$g_c = \nabla f(x_c)$ or finite difference approximation,
$H_c = \nabla^2 f(x_c)$ or finite difference approximation
or secant approximation

select new iterate x_+ by a line search or trust region method
(often $x_+ = x_c - H_c^{-1} g_c$)
evaluate $g_+ = \nabla f(x_+)$ or finite difference approximation if not done in previous step
($f(x_+)$ is evaluated in previous step)
decide whether to stop; if not
calculate $H_+ = \nabla^2 f(x_+)$ or finite difference approximation
or secant approximation

$$m(x_+ + d) = f(x_+) + g(x_+)^T d + \frac{1}{2} d^T H_+ d, \tag{2.2}$$

interpolates not only the function and gradient values at x_+, but also the gradient value at the previous iterate x_c. This interpolation condition is satisfied if the new Hessian approximation H_+ satisfys the secant equation

$$H_+ s = y \tag{2.3}$$

$$\text{where } s = x_+ - x_c \ , y = g(x_+) - g(x_c) \ .$$

The most commonly used Hessian approximation is the BFGS update (Broyden [1970], Fletcher [1970], Goldfarb [1970], Shanno [1970])

$$H_+ = H_c - \frac{H_c s s^T H_c}{s^T H_c s} + \frac{y y^T}{y^T s} \ . \tag{2.4}$$

This update makes a symmetric, rank two change to the previous Hessian approximation H_c, and if H_c is positive definite and $s^T y > 0$, then H_+ is positive definite. In practice, the initial approximation is symmetric and positive definite, and usually $s^T y$ is greater than 0, otherwise the update is skipped. Thus each Hessian approximation in a BFGS method is symmetric and positive definite.

A brief comparison between the costs and theoretical properties of second derivative and secant methods is given in Table 2.5. It shows three main differences between these two classes of methods. First, second derivative methods require an evaluation of the Hessian matrix at each iteration, while secant methods do not. Second, second derivative methods require $O(n^3)$ arithmetic operations at each iteration, because they must factorize a symmetric matrix, while secant methods can be implemented in $O(n^2)$ operations at each iteration, because they can use techniques to update the factorization of H_c into the factorization of H_+ (see e.g. Gill, Murray, and Wright [1981] or Dennis and Schnabel [1983], or Section 5.1). Third, if the Hessian matrix at the solution is nonsingular, the eventual rate of convergence of the second derivative methods is quadratic, while secant methods such as the BFGS converge at a slower superlinear rate.

Thus secant methods cost less per iteration than second derivative methods, but can be expected to require more iterations. Computational experience confirms that this is usually the case. In experiments by Schnabel, Koontz, and Weiss [1985] on a set of standard test problems, it was found that second derivative methods and secant methods had roughly the same reliability (ability to find the solution) , and that the number of iterations required by secant methods was usually a relatively small multiple (less than

Table 2.5 -- Comparison, Second Derivative Methods vs Secant Methods

		2nd Derivative	Secant
Evaluations per Iteration	$f(x)$	Usually 1 or 2	
	$\nabla f(x)$	Usually 1	
	$\nabla^2 f(x)$	1	0
Arithmetic Operations per Iteration		$\frac{n^3}{6}+0(n^2)$	$(2\text{–}6)\,n^2$
Storage		$n^2/2$ *or* n^2	
Rate of Local Convergence		Quadratic	Superlinear

$\frac{n}{2}$) times the number of iterations required by second derivative methods. This implies that if the cost of function evaluation is dominant, and the cost of an analytic or finite difference Hessian evaluation is at least $\frac{n}{2}$ times the cost of a gradient evaluation, then it is probably preferable to use secant methods. This is the case when the Hessian is approximated by finite differences. If function evaluation is not expensive, or if the cost of Hessian evaluation is not much more than the cost of gradient evaluation, then either method is probably satisfactory with second derivative methods possibly having a slight advantage in their reliability. It is our understanding that other computational studies have come to similar conclusions.

The other aspect of Algorithm 2.1 that we discuss briefly is the method for choosing the new iterate x_+. Two important classes of methods, line search methods and trust region methods, are used in practice. Roughly speaking, the objective of either type of method is that close to the solution, the new iterate should be chosen to be the minimizer $-H_c^{-1}\,g(x_c)$ of the model

$$m(x_c+d)=f(x_c)+g(x_c)^T d+\frac{1}{2}d^T H_c\,d \tag{2.6}$$

and that otherwise the new iterate $x_+=x_c+d$ should be some value for which $f(x_+)<f(x_c)$.

In a standard line search algorithm, the new iterate x_+ is chosen to be

$$x_+=x_c-\lambda_c\,(H_c+D_c)^{-1}\,g_c \tag{2.7}$$

where $\lambda_c>0$ is the step length and $(H_c+D_c)^{-1}\,g_c$ is the search direction. Here the diagonal matrix D_c is 0 if H_c is safely positive definite, and is a non-negative matrix such that H_c+D_c is positive definite otherwise (see Gill, Murray, and Wright [1981], Dennis and Schnabel [1983]). Thus the search direction is

always a descent direction for $f(x)$, and close to the solution, the search direction is the Newton direction. The step length λ_c generally is chosen so that two conditions, slightly stronger than $f(x_+) < f(x_c)$ and $s^T y > 0$, are satisfied, and so that $\lambda_c = 1$ is used if it is acceptable. It has been shown that it is always possible to choose λ_c to satisfy such conditions, and that many such strategies for choosing the step length and the search direction cause line search algorithms to be both globally convergent, and to retain fast local convergence.

The trust region approach is somewhat different. Rather than first choosing a search direction and then a step length, the trust region algorithm first chooses an approximate step length Δ_c, and then chooses the next iterate x_+ to be an approximate solution to the problem

$$\text{minimize } f(x_c) + g(x_c)^T d + \frac{1}{2} d^T H_c d \quad \text{subject to } \| d \| \leq \Delta_c \ . \tag{2.8}$$

If this step does not result in a satisfactory decrease in $f(x)$, the trust region Δ_c is reduced and problem (2.8) is solved again. If x_+ is satisfactory, the trust region is adjusted for the next iteration.

The solution to the trust region problem (2.8) is $d = -H_c^{-1} g_c$ if H_c is positive definite and $\| H_c^{-1} g_c \| \leq \Delta_c$, otherwise it is a pair (d, μ_c) for which

$$(H_c + \mu_c^2 I) d = -g_c \ , \tag{2.9}$$

$H_c + \mu_c^2 I$ is at least positive semi-definite, and $\| d \| = \Delta_c$. Since this step is expensive to calculate, trust region algorithms usually approximate it in one of two ways. Either they calculate a d which satisfies (2.9) for some μ_c and has length approximately equal to the trust radius, or they restrict the choice of d to a two-dimensional subspace, as in the "dogleg" algorithms. Many strategies for approximately solving the trust region problem (2.8), and for adjusting the trust radius Δ_c, have been shown to lead to algorithms that are globally convergent and retain fast local convergence. For more information on trust regions methods, see for example Dennis and Schnabel [1983], Moré and Sorensen [1983], or Shultz, Schnabel, and Byrd [1985].

In our computational experience (see e.g., Schnabel, Koontz, and Weiss [1985]), we have found no systematic differences between line search and trust region algorithms. This is true both in the case where H_c is the analytic or finite difference Hessian, and where it is a secant approximation. Sometimes, however, there are substantial differences between the efficiency of line search and trust region algorithms on specific problems, so that it may be useful to have both options available in software. Line search methods may enjoy a small advantage when used in conjunction with secant approximations to the Hessian, because they can assure that the necessary and sufficient condition for a positive definite secant

update, $s^T y > 0$, is satisfied at each iteration, while trust region methods cannot do this. Conversely, trust region methods may enjoy a small advantage when used with analytic or finite difference Hessians, because they seem to deal more directly with indefinite Hessians.

3. Recent Research on Secant Methods

In this section, we review some interesting recent developments concerning secant methods for unconstrained optimization. These developments fall into two categories. First, in Section 3.1, we discuss several research contributions that have helped explain the differences between the two best known secant updates, the BFGS and the DFP. Some of this work has also considered other updates in the Broyden class, which consists of all linear combinations of the BFGS and DFP. Secondly, in Section 3.2, we discuss some research that has caused a revival of interest in the symmetric rank one update.

3.1 Understanding the difference in performance between the BFGS, DFP, and other updates in the Broyden class

As we mentioned in Section 2, secant methods are commonly used to solve unconstrained optimization problems when function evaluation is expensive and analytic values of the Hessian are not available. These methods update an approximation H_c to $\nabla^2 f(x_c)$ into an approximation H_+ to $\nabla^2 f(x_+)$, using values of the gradient at x_c and x_+.

The most commonly used secant update for unconstrained optimization is the BFGS update (2.4). Among its important properties are that it obeys the secant equation (2.3), that H_+ differs from H_c by a symmetric matrix of rank two, and that H_+ is positive definite as long as H_c is positive definite and $s^T y > 0$. It has long been known that many other updates have these same algebraic properties. In particular, the DFP update

$$H_+ = H_c + \frac{(y - H_c s)y^T + y(y - H_c s)}{y^T s} - \frac{yy^T (y - H_c s)^T s}{(y^T s)^2} \tag{3.1}$$

(Davidon [1959], Fletcher and Powell [1963]), the oldest known secant update for unconstrained optimization, is another rank two update that obeys the secant equation and preserves symmetry and positive definiteness under the same conditions as the BFGS. In addition, any update in the Broyden class, which consists of all linear combinations of the BFGS and DFP

$$H_+(\phi) = (1-\phi) H_+^{BFGS} + \phi H_+^{DFP} \tag{3.2}$$

also is a symmetric rank two update that obeys the secant equation. Furthermore, if $\phi \in [0,1]$, then $H_+(\phi)$ also is positive definite if H_c is positive definite and $s^T y > 0$. All the updates in the Broyden class also share the important property that they are invariant under linear transformations of the variable space (see e.g. Dennis and Schnabel [1983]).

For some time, the conventional wisdom has been that the BFGS is the best update in the Broyden class in practice, and that it is significantly superior to the DFP. There has been little theoretical analysis, however, that helps explain this superiority. Two types of convergence analysis have been successfully applied to secant methods. One type, which originated with the work of Broyden, Dennis, and Moré [1973], proves local superlinear convergence for a direct prediction algorithm (where each iterate $x_{k+1} = x_k - H_k^{-1} \nabla f(x_k)$) under the initial assumptions that $\| x_0 - x_* \| \leq \varepsilon$ and $\| H_0 - \nabla^2 f(x_0) \| \leq \delta$ for ε and δ sufficiently small. Under these assumptions, Broyden, Dennis, and Moré [1973] proved the superlinear convergence of the iterates produced by either the BFGS or the DFP update. Stachurski [1981] and Griewank and Toint [1982] proved analogous results for any $\phi \in [0,1]$ in (3.2) under the same assumptions. Thus this convergence theory does not differentiate at all between the BFGS, DFP, and their convex combinations.

A second type of convergence result has been proven by Powell [1976]. Under the assumption that $f(x)$ is convex, he establishes both global and local superlinear convergence of a BFGS algorithm that uses a standard line search. While he does not establish this result for the DFP or any other update in the Broyden class, until recently there was no clear understanding of whether or not the result could be extended to other updates. Therefore it was not clear whether this result helped distinguish between various secant updates.

The first recent contribution that we discuss towards understanding the difference between the BFGS and other updates was provided by Powell [1986]. He compares the behavior of the BFGS and the DFP when solving the problem

$$f(x) = x^T x, \quad n = 2, \quad H_0 = \begin{bmatrix} 1 & 0 \\ 0 & \lambda \end{bmatrix} \tag{3.3}$$

under the assumption that the iterates are chosen by $x_{k+1} = x_k - H_k^{-1} \nabla f(x_k)$. (Note that due to the scale invariance of these methods, (3.4) is equivalent to $f(x) = x_1^2 + \frac{x_2^2}{\lambda}, H_0 = I$.) Powell shows that if λ is much greater than 1, and if a particularly difficult starting point,

$$x_0 = \begin{pmatrix} \cos \psi \\ \sin \psi \end{pmatrix}, \quad \psi = \arctan \sqrt{\lambda} \tag{3.4}$$

is chosen, then the BFGS will require about 2.4 $\log_{10}\lambda$ iterations to achieve convergence, whereas the DFP will require about λ iterations. If λ is much less than 1, either method requires at most about 10 iterations.

This analysis shows a potentially huge difference between the performance of the BFGS and the DFP in some situations. Table 3.5 summarizes some computations performed by Powell [1986] to confirm this analysis. The first two lines show the number of iterations required by both methods to reduce $\| x \|$ by a factor of 10^4, using x_0 given in (3.4). They correspond very closely to the estimates from Powell's analysis. The last two rows show the number of iterations required to reduce $\| x \|$ by a factor of 10^4 when $x_0 = \begin{pmatrix} \cos 40^o \\ \sin 40^o \end{pmatrix}$. While they indicate that the worst case is somewhat extreme, they still show a clear superiority of the BFGS.

Thus the analysis by Powell [1986] of the behavior of the BFGS and DFP on the problem (3.3) gives some indication of a fundamental difference between these two updates. A second recent paper, by Byrd, Nocedal, and Yuan [1987], sheds additional light on the subject.

Byrd, Nocedal, and Yuan extend the global and local superlinear convergence of Powell [1976] for the BFGS update to any update in the Broyden class (3.2) with $\phi \in [0,1)$, that is any convex combination

Table 3.5 -- Iterations Required for $\| x_k \| \leq 10^{-4} \| x_0 \|$ on Problem (3.4)
(from Powell [1986])

	λ	10^1	10^2	10^3	10^4	10^6
Bad x_o	BFGS	8	10	-	15	20
	DFP	16	107	1006	-	-
Average x_o	BFGS	6	7	-	7	7
	DFP	10	15	-	19	24

of the BFGS and the DFP except for the DFP itself. The techniques they use to prove this result give some interesting insight into the behavior of these methods. A key quantity in their analysis is $\cos(\theta_k)$, where θ_k is the angle between the step direction and the negative gradient direction at the k^{th} iteration. If this angle is less than any $\sigma < 90°$ infinitely often, then the iterates produced by any standard line search method will be globally convergent (Wolfe [1969, 1971]). Byrd, Nocedal, and Yuan show that, if $f(x)$ is uniformly convex and $x_{k+1} = x_k - \lambda_k H_k^{-1} \nabla f(x_k)$, then

$$\cos(\theta_k) \geq \frac{\lambda_k}{c \cdot \text{Trace}(H_k)} \tag{3.6}$$

for some constant c. This indicates that the method can fail to have the desired global convergence properties only if the step lengths λ_k become arbitrarily close to 0, or if the trace of H_k becomes arbitrarily large. Next they show that for any $\phi \in [0,1]$, the geometric mean of the step lengths $\{\lambda_k\}$ produced by the algorithm is bounded below. This means that the step lengths do not converge to 0 for any $\phi \in [0,1]$, so that the only possible impediment to convergence for any such update is Trace(H_k). Finally, Byrd, Nocedal, and Yuan show that

$$\text{Trace}(H_{k+1}) \leq \text{Trace}(H_k) - \frac{(1-\phi)\lambda_k}{c \cdot (\cos(\theta_k))^2} + t(\phi_k, \lambda_k, \theta_k, H_k) \tag{3.7}$$

where $t(\phi_k, \lambda_k, \theta_k, H_k)$ are some additional terms that are less crucial to the analysis. Equation (3.7) indicates that if the method takes a bad step (i.e., $\cos(\theta_k)$ close to 0), then the trace of H_{k+1} will be significantly less than the trace of H_k, as long as $\phi < 1$. This in turn can be used to show that for any method with $\phi \in [0,1]$, there cannot be too many bad steps, which leads to both global and local superlinear convergence.

An interesting aspect of the convergence of Byrd, Nocedal, and Yuan [1987] is that it shows that secant methods with $\phi<1$ have a "self-correcting" property with respect to Trace(H_k) that becomes less strong as ϕ gets closer to 1, and is not present for $\phi=1$, the DFP. This analysis does not show that the DFP fails to possess the same global and local convergence properties, but it does seem to point out a fundamental deficiency of the DFP update.

Byrd, Nocedal, and Yuan [1987] also provide a simple example that shows the deterioration of the computational performance of secant methods as ϕ goes from 0 to 1. They consider the function

$$f(x) = \frac{1}{2} x^T x + (0.1)\left(\frac{1}{2} x^T \begin{bmatrix} 5 & 1 \\ 1 & 3 \end{bmatrix} x\right)^2, \quad n = 2 \tag{3.8}$$

with the starting values $x_o = \begin{bmatrix} \cos 70^o \\ \sin 70^o \end{bmatrix}$, $H_o = \begin{bmatrix} 1 & 0 \\ 0 & 10^4 \end{bmatrix}$. Table 3.9 shows the number of iterations

required by an unconstrained optimization method with a modern line search to achieve $\| x_k \| \leq 10^{-4} \| x_o \|$ for various values of ϕ. This example clearly exhibits a deterioration in efficiency as ϕ goes from 0 to 1, but shows that this deterioration is most marked very close to the DFP (ϕ=1).

The analysis and computational example of Byrd, Nocedal, and Yuan [1987] also naturally suggest that one might try values of $\phi < 0$. This possibility has been investigated in a paper by Zhang and Tewarson [1986]. They suggest a heuristic for chosing a value of $\phi < 0$ in (3.2), and show that on a set of test problems, their method is about 10% to 15% more efficient than the BFGS on the average. They are able to prove global and r-linear convergence as long as $\theta_k \geq c_k$, where $c_k < 0$ is a quantity that is computable in practice, but can only show superlinear convergence if $\theta_k \geq \bar{c}_k$, where $\bar{c}_k < 0$ is not computable in practice. Thus, their computational and theoretical results are very interesting, but given the heuristic nature of the choice of ϕ_k and the lack of a fully satisfactory superlinear convergence result, there is probably not yet strong enough computational evidence to warrant switching from the BFGS to their new method.

It will be seen that the same contrast, between slightly improved computational performance and the lack of fully satisfactory superlinear convergence results, exists for some methods to be discussed in Section 3.2.

3.2 Recent Research on the Symmetric Rank One Update

It has long been known that there is one rank one update that satisfies the secant equation (2.3) and preserves symmetry. This update is known as the symmetric rank one (SR1) update,

---

Table 3.9 -- Iterations required for $\| x_k \| \leq 10^{-4} \| x_o \|$ on Problem 3.8
(from Byrd, Nocedal, Yuan [1987])

ϕ	0	0.2	0.4	0.6	0.8	0.9	0.99	0.999	1
iterations	15	21	26	32	66	115	630	2223	4041

$$H_+ = H_c + \frac{(y - H_c s)(y - H_c s)^T}{(y - H_c s)^T s} \quad . \tag{3.10}$$

While this update has not been used much in practice, some recent research is leading to a revival of interest in it. In this section we briefly review the properties of this update, and then discuss this recent research.

It is straightforward to show that the SR1 update is a member of the Broyden class (3.2), with

$$\phi = \frac{y^T s}{(y - H_c s)^T s} \quad . \tag{3.11}$$

This value of ϕ is always outside the range [0,1], as long as H_c is positive definite and $s^T y > 0$. Thus the SR1 update is not covered by the convergence theory mentioned in Section 2 or 3.1, nor is it guaranteed to be positive definite even if $s^T y > 0$. These properties figure prominently in the main advantages and disadvantages of the update.

The SR1 update has two main advantages. First, it is a rank one modification whereas all the other members of the Broyden class are rank two modifications; this may make it cheaper to implement. Second, it is well known that the SR1 possesses *quadratic termination*, meaning that if it is applied to a quadratic function $f(x)$ and the step $-H_c^{-1} \nabla f(x_c)$ is used at each iteration, then in exact arithmetic, the minimizer will be found exactly in $n+1$ or fewer iterations. Furthermore, if $n+1$ iterations are required, then the final Hessian approximation will equal the exact Hessian $\nabla^2 f(x)$. It can be shown that no update in the Broyden class that always preserves positive definiteness has this quadratic termination property. This at least raises the possibility that the SR1 update may produce more accurate Hessian approximations than other updates on general functions, and that it may have attractive local convergence properties.

On the other hand, the SR1 update has several disadvantages. First, there is no reason why the denominator $(y - H_c)^T s$ cannot be zero or nearly zero, even close to the solution. This indicates a potential instability in the update. Secondly, aside from the quadratic termination result mentioned above, no global or local convergence results analogous to those mentioned in Sections 2 and 3.1 have been established for the SR1.

Finally, we have already said that the SR1 will not necessarily yield a positive definite H_+ even when $s^T y > 0$. This could be an advantage if it allows the update to better model the actual Hessian when it is indefinite, or it could be a disadvantage if it leads to an indefinite approximation in a region where the actual Hessian is positive definite.

The revival of interest in the SR1 update was started by the research of Conn, Gould, and Toint [1986, 1987, 1988]. The main focus of their research is somewhat different than that considered in this paper. Conn, Gould, and Toint consider the bound-constrained optimization problem

$$\text{minimize } f(x) \quad \text{subject to } l_i \le x_i \le u_i \text{ , } i = 1, ..., n \text{ .} \tag{3.12}$$

Their research has many interesting and novel aspects, including the generalization of the notion of a Cauchy point for unconstrained optimization to the problem (3.12), the use of inexact Newton methods (methods that solve the linear system of equations associated with each iteration inexactly) in the context of problem (3.12), the introduction of new techniques that allow large changes in the set of active constraints at each iteration, and the extension of the known global convergence theory for trust region methods for unconstrained optimization to problem (3.12). We will not discuss these aspects of their research further since they are outside the scope of this paper.

The part of the research of Conn, Gould, and Toint that interests us most from the perspective of this paper is one of their computational experiments. Conn, Gould, and Toint [1986] ran their algorithm for problem (3.12) (a trust region method using secant updates and an inexact Newton method) on 50 problems, of which 15 are unconstrained. They tried both the BFGS and the SR1 updates. A summary of their results is given in Table 3.13.

The results of Conn, Gould, and Toint [1986] show a large overall advantage for the SR1 update in comparison to the BFGS. The advantage is great on problems where bounds are present, while the two updates appear similar on unconstrained problems.

These unconstrained optimization results interested us considerably, because if the SR1 is even competitive with the BFGS in general, then there are situations where it may be preferable due to its simpler form, quadratic termination, and its ability to reflect indefiniteness. Thus we decided to experiment with using the SR1 update instead of the BFGS update in the UNCMIN code (Schnabel, Koontz, and Weiss [1985]), a fairly standard unconstrained optimization method. The results of running UNCMIN, using the BFGS and the SR1, on the same unconstrained problems as were used by Conn, Gould, and Toint [1986] are shown in Table 3.14.

The UNCMIN results in Table 3.14 are very different than the unconstrained optimization results in Table 3.13, with the UNCMIN results strongly favoring the BFGS over the SR1. The reasons for the difference in the comparative performance of the BFGS and SR1 updates within the algorithms of Conn, Gould, and Toint [1986] and in UNCMIN are not clear. It should be stressed that these two algorithm are

Table 3.13 -- Computational Results from Conn, Gould, and Toint [1986]

Problems	BFGS Better	SR1 Better	BFGS, SR1 Similar	total SR1 iterations / total BFGS iterations
All (50)	9	30	11	0.71
Unconstrained (15)	6	7	2	1.12
Bounds Present (35)	3	23	9	0.56

Table 3.14 -- UNCMIN Results on the Unconstrained Problems from Table 3.13

Global Method	BFGS Better	SR1 Better	BFGS, SR1 Similar	total SR1 iterations / total BFGS iterations
Trust Region	10	1	3	2.66
Line Search	12	0	2	1.93

considerably different. Most importantly, Conn, Gould, and Toint [1986] use an inexact Newton strategy while UNCMIN finds the minimizer of the quadratic model exactly. It is also important to mention that the performances of the BFGS versions of the two algorithms are fairly similar; the big difference between the unconstrained optimization results in Tables 3.13 and 3.14 stems from the fact that the Conn, Gould, and Toint algorithm performs much better using the SR1 than UNCMIN does using the SR1. We do not yet understand why this is so, nor why the comparative advantage of the SR1 over the BFGS in Conn, Gould, and Toint's tests is so much bigger for problems with simple bounds. However all these results do seem to indicate that the SR1 update in general, and the above questions in particular, warrant additional research.

Another interesting aspect of the research of Conn, Gould, and Toint [1987] is an examination of the convergence of the sequence of matrices generated by the SR1 update. They show that if the sequence of iterates $\{x_k\}$ converges to a strong local minimizer x_*, if each set of n consecutive steps $\{x_{i+1}-x_i\}$, $i=k, \cdots, k+n-1$ is uniformly linearly independent, and if the denominators of (3.10) are bounded below in the sense that $| (y_k - H_k s_k)^T s_k | \geq c \, || y_k - H_k s_k || \, || s_k ||$ for all k, then the

sequence of Hessian approximations $\{H_k\}$ generated by the SR1 algorithm converges to $\nabla^2 f(x_*)$. This in turn implies that the rate of local convergence is at least superlinear. While these assumptions are strong, and can probably not be guaranteed to be satisfied in theory, this convergence result still gives an indication of what might often happen in practice. Indeed, Conn, Gould, and Toint [1987] conduct some experiments, using a fourth order polynomial for $f(x)$ and running to a very tight convergence tolerance, where they show that the SR1 produces final Hessian approximations that agree with the actual Hessian at the solution to within between 10^{-8} and 10^{-13}, whereas the final approximations produced by the BFGS only agree to about 10^{-3}. This research supports the hypothesis that the quadratic termination property of the SR1 might lead to better final Hessian approximations in practice. It also seems to further indicate a need for continued research on the role of the SR1 update in unconstrained optimization.

We conclude this section by discussing a second, somewhat different, new algorithm involving the SR1 update that was recently proposed by Osborne and Sun [1988], motivated in part by the work of Conn, Gould, and Toint. Osborne and Sun's approach is to use the SR1 in such a way that it always produces positive definite Hessian approximations. They do this by first multiplying the current Hessian approximation H_c by a scale factor $\gamma > 0$, and then applying the SR1 update to γH_c. That is

$$H_+ = \gamma H_c + \frac{(y - \gamma H_c s)(y - \gamma H_c s)^T}{(y - \gamma H_c s)^T s} . \tag{3.15}$$

It is fairly easy to see that if H_c is positive definite and $s^T y > 0$, then H_+ given by (3.15) will be positive definite if γ is either sufficiently large or sufficiently close to 0. In fact, Osborne and Sun [1988] show that H_+ is positive definite if $s^T y > 0$ and

$$\gamma \in (0, \frac{s^T y}{s^T H_c s}) \text{ or } \gamma \in (\frac{y^T H_c^{-1} y}{s^T y}, \infty). \tag{3.16}$$

Note that the standard SR1 update, $\gamma = 1$, may or may not be contained in one of the two intervals in (3.16).

Osborne and Sun [1988] propose using the standard SR1 update, $\gamma = 1$ in (3.15), if it satisfies (3.16). Otherwise, they propose choosing the value of γ that satisfies (3.16) and that leads to the optimally conditioned update in the sense proposed by Davidon [1975], namely that it minimizes the l_2 condition number of $H_c^{-1/2} H_+ H_c^{-1/2}$ among all the updates of the form (3.15). Osborne and Sun derive a closed form for this optimal value of γ; actually there are two values of γ that yield equally optimal solutions, one in each of the intervals in (3.16).

Osborne and Sun report promising computational results using this scaled SR1 method on a small set of test problems. We have tried using their update in the UNCMIN code, and have found that on the average it leads to a 10-15% improvement over the BFGS update on a standard set of test problems. Therefore, since the Osborne and Sun algorithm has the additional advantage in comparison to the BFGS that it only requires a rank one update, it appears to merit further consideration. On the other hand, the scaled SR1 update (3.15) shares with the standard SR1 update (3.10) the apparent disadvantage that no global or local convergence results have been proven for it under the standard assumptions that are used in the convergence analysis of many secant methods, including the BFGS (see Section 3.1). Thus more research seems necessary to understand both the practical and theoretical properties of all the SR1 methods discussed in this section.

4. Recent Research on Second Derivative Methods -- Tensor Methods

In this section we turn our attention to second derivative methods, methods where the analytic or finite difference Hessian is available at each iteration. We discuss one recent development, the development of *tensor methods*. This is a class of methods that bases each iteration upon a higher order model than is used by standard methods. The higher order terms in this model are chosen so that the model is hardly more expensive to form, store, and solve than the standard model.

Tensor methods were first developed by Schnabel and Frank [1984] in the context of solving systems of nonlinear equations. These methods base each iteration upon a quadratic model, rather than the linear model that is standard for solving systems of nonlinear equations. Since an understanding of tensor methods for nonlinear equations is helpful in understanding the more complicated tensor methods for unconstrained optimization, we review tensor methods for nonlinear equations in Section 4.1. Then in Section 4.2 we describe tensor methods for unconstrained optimization. These methods base each iteration upon a fourth order model, rather than the standard quadratic model (2.6).

We note that another recent body of research has considered the use of higher ordered models when the objective function is a "factorable function." See for example Jackson [1983], Jackson and McCormick [1986], and McCormick [1983].

4.1 Tensor Methods for Nonlinear Equations

The nonlinear equations problem is

$$\text{given } F : R^n \to R^n \text{, find } x_* \text{ for which } F(x_*) = 0 \ . \tag{4.1}$$

When the Jacobian matrix $F'(x_c)$ is available, algorithms for solving (4.1) generally base each iteration upon the linear model

$$M(x_c + d) = F(x_c) + F'(x_c)d \ . \tag{4.2}$$

This model requires n^2 storage locations, and $\frac{n^3}{3}$ arithmetic operations to solve it at each iteration. Its use leads to quadratic convergence for problems where $F'(x_*)$ is non-singular. but at best linear convergence for problems where $F'(x_*)$ is singular (see for example Decker and Kelly [1980a, b], Griewank [1980]).

The tensor method proposed by Schnabel and Frank [1984] instead bases each iteration upon the model

$$M(x_c + d) = F(x_c) + F'(x_c)d + \tfrac{1}{2} T_c\, dd \tag{4.3}$$

where $T_c \in R^{n \times n \times n}$ is a three-dimensional object often referred to as a tensor. If $T_c = F''(x_c)$, then (4.3) is just a second order Taylor series model. However using (4.3) with $T_c = F''(x_c)$ is not practical, as it leads to huge increases in the costs to form, store, and solve the model. Instead, Schnabel and Frank chose T_c in (4.3) to be a very low rank approximation to $F''(x_c)$. We now briefly summarize how they make this choice, and some of its consequences.

Schnabel and Frank [1984] choose T_c in (4.3) by requiring the model to interpolate the values of $F(x)$ at p (not necessarily consecutive) previous iterates $x_{-1}, \cdots, x_{-p}$. They impose the limit $p \le \sqrt{n}$, and also require that the steps $s_i = x_c - x_{-i}$ from x_c to these p previous iterates be strongly linearly independent. The latter condition is usually much more restrictive than the limit $p \le \sqrt{n}$, and most often results in the choice $p = 1$, meaning that only information from the most recent previous iterate is used. Schnabel and Frank then choose the smallest T_c, in the Frobenius norm, that satisfies the p interpolation conditions $M(x_c - s_i) = F(x_{-i})$, $i = 1, \cdots, p$. The result is a rank p tensor T_c of the form $T_c = \sum_{i=1}^{p} a_i\, s_i\, s_i$, for some $a_i \in R^n$, $i = 1, \cdots, p$. Thus the model (4.3) becomes

$$M(x_c + d) = F(x_c) + F'(x_c)d + \tfrac{1}{2} \sum_{i=1}^{p} a_i\, (s_i^T d)^2 \, . \tag{4.4}$$

The additional cost of forming this model is about $n^2 p$ arithmetic operations, while the additional storage

cost is about $4np$ locations. Both of these are small in comparison to the basic $O(n^3)$ arithmetic per iteration and n^2 storage costs of the linear model.

Due to the special form of the model (4.4), the problem of finding its root (or of minimizing $\| M(x_c + d) \|_2^2$ if there is no root) can be reduced to solving p quadratic equations in p unknowns plus $n-p$ linear equations in $n-p$ unknowns. This is hardly more expensive than finding the root of the linear model (4.2), requiring only about n^2p additional arithmetic operations (recall that usually $p=1$). Furthermore, Schnabel and Frank [1984] show that the solution of (4.4) is usually well posed as long as the rank of $F'(x_c)$ is at least $n-p$, whereas the solution of the linear model (4.2) only is well posed if rank $(F'(x_c)) = n$.

The above simply shows that, by utilizing a low rank quadratic term, it is possible to add a small amount of information to the linear model at a small cost. What is perhaps surprising is that this small amount of information seems to lead to fairly large improvements in the cost of solving problem (4.1) in practice. Schnabel and Frank [1984] compare an implementation of their tensor method, using a standard line search, to a standard method for nonlinear equations that uses the linear model (4.2) and the same line search, on a set of problems from Moré, Garbow, and Hillstrom [1981]. Their results are summarized in Table 4.5; for more details, see Schnabel and Frank [1984].

Table 4.5 indicates that the tensor method leads to consistent and rather substantial improvements in efficiency over a standard derivative-based method for solving systems of nonlinear equations.

Table 4.5 -- Comparison of Tensor and Standard Methods for Nonlinear Equations
(from Schnabel and Frank [1984])

Rank $F'(x_*)$	Problem Set	Tensor Better	Standard Better	Two Methods similar	Average Ratio of Tensor Iterations/Standard Iterations
n	All	21	1	6	0.77
	Harder Only*	14	0	0	0.61
$n-1$	All	15	0	2	0.58
	Harder Only*	9	0	0	0.39
$n-2$	all	11	2	0	0.63
	Harder Only*	7	0	0	0.50

*Only those problems where slower method required at least 10 iterations.

Furthermore, in these tests the number of past points, p, used in the tensor model was generally 1 and never more than 3, so that the additional cost of using the tensor method was low. A Fortran code for solving systems of nonlinear equations, and nonlinear least squares problems, by tensor methods is available from the author. Our positive experience with tensor methods for nonlinear equations also has motivated us to consider using tensor methods for unconstrained optimization, which are described in the next section.

4.2 Tensor Methods for Unconstrained Optimization

The extension of the tensor method for nonlinear equations that we have just described into a tensor method for unconstrained optimization brings up an interesting general issue. In some cases, such as the basic Newton method, methods for nonlinear equations and for unconstrained optimization are very closely related. In some other cases, such as secant methods, methods for nonlinear equations and unconstrained optimization are considerably different. These differences are generally due to the considerations of symmetry, convexity, and positive definiteness which are present for unconstrained optimization but not for nonlinear equations. In the case of tensor methods, the differences between unconstrained optimization and nonlinear equations cause the tensor methods for the two problems to differ significantly.

The obvious extension of the tensor method for nonlinear equations to the unconstrained optimization problem would be to base each iteration upon the quadratic model of the gradient

$$\nabla m\,(x_c + d) = \nabla f\,(x_c) + \nabla^2 f\,(x_c)\,d + \tfrac{1}{2} T_c\,dd \quad . \tag{4.6}$$

that would result from simply substituting $\nabla f\,(x)$ for $F(x)$ in the method of Section 4.1. This model is unappealing for unconstrained optimization for several reasons. First, the tensor T_c produced by the method described in Section 4.1 is not symmetric, but for optimization it should be. Second, we will want to interpolate past values of $f\,(x)$ as well as $\nabla f\,(x)$, so a procedure based solely on modeling $\nabla f\,(x)$ is too restrictive. Most importantly, the model (4.6) corresponds to using a third order model of $f\,(x_c)$, and a third order model has two basic deficiencies for unconstrained optimization. One is that it does not have a global minimizer. A second is that it does supply enough information to lead to faster than linear convergence for problems where $\nabla^2 f\,(x_*)$ is singular. For this, fourth order information is necessary as well.

For these reasons, Schnabel and Chow [1988] propose using a fourth order model of $f\,(x)$

$$M\,(x_c + d) = f\,(x_c) + \nabla f\,(x_c)^T d + \frac{1}{2} d^T\,\nabla^2 f\,(x_c) d + \frac{1}{6} T_c\,ddd + \frac{1}{24} V_c\,dddd \tag{4.7}$$

where $T_c \in R^{n \times n \times n}$ is a symmetric approximation to $\nabla^3 f\,(x_c)$ and $V_c \in R^{n \times n \times n \times n}$ is a symmetric

approximation to $\nabla^4 f(x_c)$. While this model may appear complicated, the crucial point is that T_c and V_c will again be low rank tensors. We will see that this again makes the model hardly more expensive to form, store, and solve than the standard quadratic model.

To form the tensor model (4.7), Schnabel and Chow require it to interpolate the values of $f(x)$ and $\nabla f(x)$ at p previous iterates x_{-i}, $i=1,\cdots,p$. These iterates are chosen by the same criteria as in the tensor method for nonlinear equations, namely that $p \leq \sqrt{n}$ and that the steps $s_i = x_c - x_{-i}$ are strongly linearly independent. In practice, usually the second criterion limits p to 1, meaning that only information from the most recent previous iteration is used. Next Schnabel and Chow choose the smallest V_c, in the Frobenius norm, that is consistent with these interpolation conditions. The result is a symmetric rank p tensor V_c of the form

$$V_c = \sum_{i=1}^{p} \alpha_i \, s_i s_i s_i s_i \,, \tag{4.8}$$

where each α_i is a scalar. Finally they choose T_c to be the smallest symmetric tensor, in the Frobenius norm, that is consistent with the choice of the V_c and the interpolation conditions. This yields a rank $2p$ tensor T_c of the form

$$T_c = \sum_{i=1}^{p} (b_i s_i s_i + s_i b_i s_i + s_i s_i b_i) \tag{4.9}$$

where each $b_i \in R^n$. This order of choosing V_c and T_c causes the model to interpolate as much information as possible with a third order model, and to use the fourth order term only where a third order model is insufficient.

Even though T_c is a rank $2p$ tensor, Schnabel and Chow [1988] show that the minimizer of the resultant tensor model (4.7) can still be found by solving p cubic equations in p unknowns and $n-p$ linear equations in $n-p$ unknowns. Thus the size of the system of nonlinear equations that must be solved to minimize the tensor model for unconstrained optimization is the same as for nonlinear equations, with the difference being that the p nonlinear equations are cubic rather than quadratic. Since p generally is 1, this is not significant. The total additional costs of using the tensor model rather than the standard quadratic model are again $O(np)$ storage locations, and $O(n^2 p)$ arithmetic operations per iteration. This is again minor in comparison to the n^2 storage locations and at least $\frac{n^3}{6}$ arithmetic operations per iteration that are required by the standard method.

Schnabel and Chow [1988] compare an implementation of their tensor method for unconstrained optimization to a standard method based on a quadratic model, on the problems from Moré, Garbow, and

Hillstrom [1981]. Both algorithms use the same, fairly standard trust region strategy. A summary of their computational results is given in Table 4.10. The lines labeled "$p \geq 1$" show the results when p, the number of previous iterates whose function and gradient is interpolated is interpolated, is allowed to exceed one, while the lines labeled "p=1" show the results when only information from the most recent iterate is used.

These results indicate that once again, the tensor method seems to obtain a considerable improvement in efficiency from using a rather small additional amount of information. Indeed, the tensor method for unconstrained optimization rarely chooses $p > 1$ even when we allow this possibility, and Table 4.10 shows if we require $p = 1$, then the results do not change appreciably. Thus it appears that we can achieve substantial savings at a very low cost.

It is not yet clear to us why tensor methods for both unconstrained optimization and nonlinear equations achieve rather large improvements in efficiency from utilizing a rather small additional amount of information. For one class of problems, nonlinear equations with rank$(F'(x_*)) = n-1$, the observed improvement is probably due to a faster local convergence rate, since Frank [1984] shows that the tensor method achieves 3-step superlinear convergence as opposed to the linear rate of standard methods. This pattern of convergence is observed in practice, and may also occur in the analogous case for unconstrained optimization. But the rate of convergence of the tensor methods is almost certainly not superior to standard methods when $F'(x_*)$, or $\nabla^2 f(x_*)$, is non-singular, and yet the tensor methods usually are considerably faster in these cases too. Our conjecture is that this improvement occurs because the

Table 4.10 -- Comparison of Tensor and Standard Methods for Unconstrained Optimization
(from Schnabel and Chow [1988])

Rank $\nabla^2 f(x_*)$	Value of p in Tensor Method	Tensor Better	Standard Better	Two Methods Similar	Average Ratio of Tensor Iterations to Standard Iterations
n	$p \geq 1$	27	4	5	0.65
	$p = 1$	27	1	7	0.62
$n-1$	$p \geq 1$	27	2	4	0.56
	$p = 1$	30	3	2	0.55
$n-2$	$p \geq 1$	26	2	4	0.59
	$p = 1$	28	5	3	0.58

directions of two consecutive steps are often rather similar in practice, so that the additional information provided by the tensor model, which is along the previous step direction, turns out to be especially useful. In any case, it appears to us that the use of higher order models seems fruitful, and that additional research on such approaches is warranted.

5. Parallel Methods for Unconstrained Optimization

It is becoming clear that the fastest computers in the future will be parallel computers. Therefore, it is natural that there has been increased interest recently in designing parallel optimization algorithms. In this section we discuss one aspect of this work, parallel methods for the general unconstrained optimization problem. Some of this research will be seen to consist of the development of new algorithms motivated by the consideration of parallel computers, while other parts simply consist of determining good ways to parallelize existing sequential methods.

Before we begin, we need to clarify which of the various types of parallel computers we are concerned with. At a high level, currently available parallel computers can be divided into three categories, vector computers, processor arrays, and multiprocessors. Vector computers can perform pairwise addition or multiplication of long vectors of numbers quickly. They are best suited to a low level, fine grain type of parallelism. Processor arrays (SIMD computers) are computers with many processors that can perform the same operation in lock step on multiple data at the same time. Thus they are also suited to a fairly low level, fine to medium grain type of parallelism. By multiprocessors (MIMD computers), we mean computers that can perform entirely different operations on different data at the same time. These include both shared memory multiprocessors, where all the processors share access to some of the memory, and distributed memory multiprocessors, where each processor has its own memory and there is no shared memory. These computers support a higher level, medium to coarse grain type of parallelism.

Among these various types of parallel architectures, the type of optimization problems we consider in this paper seem best suited to solution on (MIMD) multiprocessors. This is because parallel algorithms for general unconstrained optimization problems generally seem to entail a high level, coarse grain type of parallelism. Largely this is because the evaluation of the objective function $f(x)$, an atomic operation in these algorithms, is itself often a lengthy calculation. Furthermore, if we wish to perform two or more computations of $f(x)$ concurrently, this will often require an MIMD computer since for two different values of x, the program that evaluates $f(x)$ may well execute different sequences of instructions. Thus the discussion in the remainder of this section is mainly oriented towards parallel computation on

multiprocessors, though the specific type of multiprocessor, for example shared or distributed memory, is relatively unimportant.

The remainder of this section will discuss parallel methods that are related to the most commonly used general purpose unconstrained optimization method, the BFGS method with a line search. A high level description of this method is given in Algorithm (5.1).

Since presumably we are interested in parallel optimization in order to solve expensive problems, we need to focus on why the BFGS method can be expensive. There are two main reasons. One is that the function and derivative evaluations are expensive. The second is that the linear algebra computations, namely updating the Hessian approximation and calculating the search direction, are expensive because the number of variables is large.

There are several obvious possibilities for adapting these potentially expensive portions of the BFGS algorithm to parallel computers. The first is to parallelize the individual calculations of the objective function $f(x)$. This may or may not be feasible, depending upon the form of $f(x)$, the availability of pre-existing parallel routines to calculate it, or the interest of the user in devising new parallel routines to evaluate it. In any case, it is outside the scope of basic optimization research. Thus for the remainder of this section we will assume that $f(x)$ is evaluated on one processor.

Algorithm 5.1 -- Iteration of the BFGS Method with Line Search

given current iterate x_c , $f(x_c)$,
$g_c = \nabla f(x_c)$ or finite difference approximation,
H_c = symmetric positive definite Hessian approximation

calculate search direction d_c:
solve $H_c\, d_c = -g_c$
line search:
find $\lambda > 0$ for which $x_+ = x_c + \lambda_c\, d_c$ is satisfactory next iterate
evaluate $g_+ = \nabla f(x_+)$ or finite difference approximation if not done in previous step
($f(x_+)$ is evaluated in previous step)
decide whether to stop; if not
update Hessian approximation by BFGS formula (2.4):
$H_+ = H_c$ + rank two matrix

The second possibility is to parallelize the dominant linear algebraic computations of the BFGS method, mainly the Hessian updates and the calculations of the step directions. We consider this topic in Section 5.1. While it basically consists of ways to parallelize the existing method, it brings up some interesting fundamental issues in unconstrained optimization.

The third possibility is to perform multiple evaluations of the objective function $f(x)$ concurrently. The possibilities here range from calculating several components of the finite difference gradient concurrently to new algorithms that make use of concurrent function evaluations. We consider both of these possibilities in Section 5.2.

The material in this section is taken largely from Schnabel [1987] and Byrd, Schnabel, and Shultz [1988a, b].

5.1 Parallel Methods for the Linear Algebra Calculations in Secant Methods

In this section, we consider the parallelization of the linear algebra calculations in the standard BFGS method. This topic is interesting because it leads us to re-examine various ways for implementing these calculations. It also leads us to interpret a recent proposal of Han [1986] in a different light.

It is convenient to view the linear algebra calculations involved in an iteration of the BFGS method as the update of the Hessian approximation followed by the calculation of the next search direction. Table 5.2 summarizes four different ways of implementing these calculations, along with their costs. These four options arise from choosing between two possibilities each for two orthogonal attributes of how the Hessian approximation may be stored. First, one can store either an approximation to the Hessian itself or an approximation to the inverse of the Hessian. Second, this approximation can either be kept in the straightforward, unfactored, form, or one can store a matrix M from a factorization MM^T of the Hessian or the inverse Hessian approximation.

The upper left variant in Table 5.2 is the most obvious way to implement the BFGS method. It simply consists of performing the update (2.4) to obtain H_+, and then performing a Cholesky factorization of H_+ to solve $H_+ d_+ = -g_+$ for d_+. This is the only variant that requires $O(n^3)$ arithmetic operations; all the others require only $O(n^2)$.

The upper right and lower left variants in Table 5.2 are the two well-known ways for performing the linear algebra calculations in a BFGS algorithm in $O(n^2)$ operations. The lower left variant was the first known $O(n^2)$ implementation of the BFGS; it consists of using the formula that gives the effect of

Table 5.2 -- Four Possible Implementations of the Linear Algebra Calculations :

$$H_{k+1} = H_k + \text{rank-two-matrix}$$
$$\text{solve } H_{k+1}\, d_{k+1} = -g_{k+1} \text{ for } d_{k+1}$$

	Matrix Stored Unfactored	Matrix Stored Factored
Direct (H_k) Update	(H_k stored, updated to H_{k+1}) $H_{k+1} = H_k$ + rank-two Cholesky factor H_{k+1} 2 triangular solves to find d_{k+1} $\frac{n^3}{6} + 2n^2$	(L_k lower triangular stored, for which $H_k = L_k L_k^T$, updated to L_{k+1} lower triangular for which $H_{k+1} = L_{k+1} L_{k+1}^T$) $J_{k+1} = L_k$ + rank-one $J_{k+1} = Q_{k+1} L_{k+1}$ by Givens rotations 2 triangular solves to find d_{k+1} $6n^2$ $(2.5n^2)$
Inverse (H_k^{-1}) Update	(H_k^{-1} stored, updated to H_{k+1}^{-1}) $H_{k+1}^{-1} = H_k^{-1}$ + rank-two Matrix-vector multiply to find d_{k+1} $2n^2$	(M_k stored for which $H_k^{-1} = M_k M_k^T$, updated to M_{k+1} for which $H_{k+1}^{-1} = M_{k+1} M_{k+1}^T$) $M_{k+1} = M_k$ + rank-one 2 Matrix-vector multiples to find d_{k+1} $4n^2$

the BFGS update on the inverses of H_c and H_+,

$$H_+^{-1} = H_c^{-1} + \frac{(s - H_c^{-1}y)s^T + s(s - H_c^{-1}y)^T}{s^T y} - \frac{ss^T(s - H_c^{-1}y)^T y}{(s^T y)^2}, \tag{5.3}$$

and then multiplying $-g_+$ by H_+^{-1} to get d_+. The upper right variant was subsequently discovered (Gill and Murray [1972], Goldfarb [1976]), and has become the favored $O(n^2)$ method for implementing the BFGS. It consists of directly updating the Cholesky factorization LL^T of H_c into the Cholesky factorization $L_+L_+^T$ of H_+. This is accomplished by expressing the BFGS update as a rank one change to the Cholesky factor L, resulting in a non-triangular matrix J_+, and then transforming J_+ into a lower triangular matrix L_+ through a series of $2n-2$ Given's rotations. Then d_+ is found by using two backsolves to solve $L_+L_+^T d_+ = -g_+$ for d_+. An advantage of this implementation is that by always keeping a Cholesky factorization, it implicitly guarantees that the Hessian approximation is numerically positive definite.

(For details on the operation counts, see Byrd, Schnabel, and Shultz [1988b].)

The fourth, lower right, variant is closely related to the upper right variant. It consists of updating a factorization MM^T of H_c^{-1} directly into a factorization $M_+M_+^T$ of H_+^{-1} by making a rank one change to M, and then multiplying $-g_+$ by $M_+M_+^T$ to obtain d_+. Since the savings that would occur in these matrix vector multiplications if M_+ were triangular would be exceeded by the cost of restoring M_+ to triangularity at each iteration, it is preferable to omit the Given's rotations that are used in the upper right variant and simply store M as a full matrix.

To our knowledge, the lower right variant has hardly been considered in the form in which we have just described it, but it is equivalent to a suggestion of Han [1986]. Han proposes implementing the linear algebra of the BFGS on parallel computers by keeping a matrix Z_c for which

$$Z_c^T H_c Z_c = I, \tag{5.4}$$

and then updating it to a matrix Z_+ for which

$$Z_+^T H_+ Z_+ = I \tag{5.5}$$

by making a rank one change to Z_c to obtain Z_+. But since equations (5.4) and (5.5) are equivalent to $H_c^{-1} = Z_c Z_c^T$ and $H_+^{-1} = Z_+ Z_+^T$, this is simply another way of interpreting the lower right variant in Table 5.2, and it is easy to confirm that the calculations in Han's method and the lower right variant in Table 5.2 are identical.

These four possible ways of implementing the linear algebra calculations of the BFGS method produce identical results in exact arithmetic. They differ in the number of arithmetic operations they require, in their suitability to parallel computation, and perhaps in their accuracy when implemented in computer arithmetic. None appears to be best in all these regards. There has long been a belief that the two left hand variants in Table 5.2 might have problems in finite precision arithmetic because they might produce numerically non-positive definite Hessian approximations. Powell [1987] has also shown that the two bottom variants may have difficulties in finite precision arithmetic handling very badly scaled problems. The combination of these two observations thus leads one to favor the upper right variant and this is the conventional wisdom.

In several computational tests that have compared the use of these four variants in a BFGS method, however, the differences between them have turned out to be negligible, even on somewhat extreme test problems. These include tests performed by Grandinetti [1978], Connolly and Nocedal [1987], and tests we recently performed using all four variants of the BFGS shown in Table 5.2 in the UNCMIN code. In

our tests, on no problem was there more than a 1-2% variation between the costs of solving it using these four different implementations.

Thus we consider any of the variants in Table 5.2 to be a suitable candidate for the implementation of the BFGS method on a parallel computer. Among these four, the bottom two variants clearly are the most amenable to parallelization. Both require only matrix vector multiplications and rank one updates, computations that can be easily and efficiently parallelized over a large variety of parallel architectures and problem sizes. The upper right variant appears far more difficult to parallelize efficiently due to the need to perform the $2n-2$ Given's rotations, on vectors of size 2 through n, sequentially. The upper left variant is clearly too expensive to consider since some of the much less expensive sequential variants parallelize well. The difference in efficiency between the two bottom variants on parallel computers can be expected to be about a factor of two (but only 4/3 on a distributed memory computer, see Byrd, Schnabel, and Shultz [1988b]), so we would tend to favor the lower left variant, but experiments using the two bottom variants on parallel computers need to be performed.

Thus the discussion of this section has shown that the consideration of parallel computation may lead to a different choice for implementing the linear algebraic computations in the BFGS method than is generally made on sequential computers. It also makes it clear that, to conclusively resolve this issue, optimization researchers need to carefully consider whether and when the inverse updates, factored or unfactored, have practical computational deficiencies.

5.2 Utilizing Concurrent Function Evaluations for Unconstrained Optimization

In this section we review some possibilities for utilizing concurrent function evaluations in general purpose unconstrained optimization algorithms. These possibilities include both the parallelization of standard optimization algorithms, and new algorithms motivated by the consideration of parallel computation. Our point of departure is the standard BFGS method described in Algorithm 5.1.

Recall first the pattern of function and gradient evaluations in a standard secant method like Algorithm 5.1. Each iteration performs one or more *trial point function evaluations*, evaluations of $f(x_c+\lambda d_c)$ for some values of the step length parameter λ, within the line search. The last of these is at the successful next iterate x_+, and is followed by the gradient evaluation at x_+. Generally this is the only gradient evaluation in the iteration. In our experience, the average number of trial point function evaluations per iteration over the course of the algorithm tends to be between 1.2 and 1.5.

We are most concerned with performing these function and derivative evaluations efficiently on a parallel computer if they are expensive. In our practical experience, when $f(x)$ is expensive, $\nabla f(x)$ usually is calculated by the finite difference approximation

$$\nabla f(x)_i = \frac{f(x + h_i e_i) - f(x)}{h_i}, \quad i = 1, \cdots, n \tag{5.6}$$

where e_i is the i^{th} unit vector and h_i is an appropriately chosen finite difference step size. This requires n additional evaluations of $f(x)$. Clearly these function evaluations can be performed concurrently on a parallel computer. Thus on a machine with p processors, the finite difference gradient evaluation requires $\left\lceil \frac{n}{p} \right\rceil$ *concurrent function evaluation steps*, steps where each processor performs at most one evaluation of $f(x)$ concurrently.

Thus if $f(x)$ is expensive and the number of processors, p, is much less than n, then simply parallelizing the finite difference gradient calculation by performing groups of p function evaluations concurrently leads to very good speedups. If $p >= n$, however, the overall speedup for all the function and gradient evaluation steps will be no better than about half of optimal. This is because all but one of the processors will be unused during the trial point function evaluations, and there will be at least as many steps devoted to trial point function evaluations as to gradient evaluations.

This leads to the obvious question, "How can we utilize additional processors while evaluating $f(x_c + \lambda d_c)$ on one processor in the line search?" There are two obvious possibilities. The most common suggestion (see for example Dixon and Patel [1982], or Lootsma [1984]) is to supplement the line search by evaluating $f(x)$ at $p-1$ other points simultaneously. We refer to this strategy as a *multiple point search.*

A second possibility, originally suggested by Schnabel [1987], is to use the remaining $p-1$ processors to evaluate (part of) $\nabla f(x_c + \lambda d_c)$ by finite differences *before* it is known whether this gradient value will be needed. We refer to this as a *speculative* (partial) gradient evaluation. If $x_c + \lambda d_c$ is accepted as the next iterate, as it usually will be, then the function evaluations that have been performed in the speculative gradient evaluation have all turned out to be necessary ones. If $x_c + \lambda d_c$ is not accepted as the next iterate, then the speculative gradient evaluation has been unnecessary, although Byrd, Schnabel, and Shultz [1988b] describe a new method that makes some beneficial use of this information.

We believe that using the extra processors that are available, while evaluating $f(x_c + \lambda d_c)$ in the line search, to perform a speculative finite difference gradient evaluation will usually result in a more efficient parallel algorithm than utilizing the extra processors to perform a multiple point search. Indeed,

in order for the multiple point search to be the superior strategy, it would need to reduce the overall number of iterations by a factor of almost $\frac{n+p}{n}$ (assuming $p \leq n+1$). In this case, the multiple point search would lead to a better sequential algorithm as well. We consider this unlikely, especially since it is usually very hard to improve upon the choice $\lambda = 1$ close to the solution. We cannot make a more definite assessment because, to our knowledge, proposers of multiple point searches have not provided the data that would allow us to compare their strategies to speculative gradient evaluations.

If $p \leq n+1$ and gradients are evaluated by finite differences, then we feel there is little more to say about utilizing concurrent function evaluations for unconstrained optimization. The remaining interesting cases are when $p > n+1$, or when $f(x)$ and $\nabla f(x)$ are naturally evaluated together utilizing one processor. In either of these cases, additional processors are available while $f(x)$ and $\nabla f(x)$ are being evaluated. Byrd, Schnabel, and Shultz [1988a, b] consider several ways to utilize these additional processors. In the remainder of this section we briefly summarize some of their suggestions.

If there are enough processors so that the function, gradient, and finite difference Hessian can all be evaluated at once, then the analogous idea to that discussed above is to perform speculative finite difference Hessian evaluations. (Finite difference approximation of the Hessian requires $\frac{n^2+3n}{2}$ additional evaluations of $f(x)$ or n additional evaluations of $\nabla f(x)$.) This means evaluating all of $\nabla^2 f(x_c + \lambda d_c)$, as well as $\nabla f(x_c + \lambda d_c)$, concurrently with the trial point function evaluation $f(x_c + \lambda d_c)$. If $x_c + \lambda d_c$ is accepted as the new iterate x_+, as it usually will be, all the speculative evaluations perform useful work. Note that this is basically just a way to parallelize a standard second derivative method, although new algorithmic features could be introduced to attempt to the speculative gradient and Hessian information at unsuccessful trial points.

If there are more processors than necessary to evaluate $f(x)$ and $\nabla f(x)$ simultaneously, but not enough to evaluate the entire finite difference Hessian simultaneously as well, then Byrd, Schnabel, and Shultz [1988a, b] propose utilizing the remaining processors to perform speculative evaluation of *part* of the finite difference Hessian at each iteration. This leads to the consideration of new optimization algorithms. Now we briefly review this work

Byrd, Schnabel, and Shultz [1988a] consider many alternatives for evaluating part of the Hessian at each iteration, and for incorporating this information into an optimization algorithm. The strategy for evaluating part of the Hessian which they find to be most successful is simply to evaluate some set of columns of the Hessian at each iteration, with these sets sweeping through all the columns as the iterations proceed. At any given iteration, this means one evaluates $z_i = \nabla^2 f(x) \cdot e_i$ for some values

$i \in I_c \subseteq [1,n]$. The strategy for incorporating this information that Byrd, Schnabel, and Shultz [1988a, b] find best is to first update H to $\bar{H}$ by the standard BFGS update (2.4), utilizing the normal secant equation (2.3). Then they update $\bar{H}$ to H_+ by performing a multiple BFGS update (Schnabel [1983]), which causes H_+ to satisfy $H_+ e_i = z_i$ for each $i \in I_c$. The rationale for inserting the information in this order is that the normal BFGS update gives information about the Hessian between x_c and x_+, while the finite difference Hessian information gives values at x_+. Thus the finite difference information is inserted last.

Byrd, Schnabel, and Schultz [1988b] show that an algorithm that utilizes the above strategy for incorporating part of the finite difference Hessian at each iteration retains the superlinear convergence rate of the BFGS method. Of course the intent is that the new method should perform better than the BFGS in practice, since it uses more information, but this is about the best theoretical result one can expect.

The results of extensive computational tests utilizing the partial Hessian methods described above are reported in Byrd, Schnabel, and Shultz [1988a, b]. Table 5.7 summarizes their results on a set of test problems with n=20. The second row shows the speedup of the new method, that evaluates the function, gradient, and q columns of the finite difference Hessian at each trial point, over a parallel BFGS method that just evaluates the function and gradient at each trial point, under the assumption that there are enough processors to evaluate $f(x)$, $\nabla f(x)$, and q columns of the finite difference Hessian concurrently. The third row shows the speedups of the same partial Hessian methods, under the same assumptions about the

Table 5.7 -- Average Speedup of a Method Using Speculative Partial Hessian Evaluations on a Test Set with n=20 (speed measured in function evaluations)
(from Byrd, Schnabel, and Shultz [1988b])

Number of columns of Hessian calculated at each iteration	0	1	2	3	4	5	20*
Speedup over Parallel BFGS utilizing speculative gradient evaluation	(1.0)	1.8	2.3	2.9	3.0	3.2	6.0
Speedup over Sequential BFGS with finite difference gradients	17.5	31.5	40.3	50.8	52.5	56.0	105
Number of processors required to evaluate $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)$ concurrently from function values	21	42	62	81	99	116	231

*Newton's method

number of processors, in comparison to a standard sequential BFGS method that performs finite difference gradient evaluations and utilizes only one processor. These numbers in the third row are 17.5 times as large as the numbers in the second row. This reflects the fact that the parallel BFGS is 17.5 times as fast as the sequential BFGS on these problems on the average, or equivalently, that there are an average of 1.21 trial point function evaluations per iteration on these problems. The final row shows the number of processors that would be necessary to implement the parallel partial Hessian algorithms for each value of q, when the finite difference gradient and Hessian are computed from function values.

The results in Table 5.7 show that the new method that utilizes otherwise idle processors to perform speculative partial Hessian evaluations is more efficient, in terms of concurrent function evaluation steps, than a standard method that doesn't utilize these processors, but that these improvements are not proportional to the amount of new information that is being utilized. This is to be expected, however, because as the table shows, Newton's method, which uses $\frac{n}{2}$ times as much information as the BFGS method, is only about 6 times as fast on the average on these problems. Combining the last two rows of Table 5.7 shows that the speedups are at least 0.45 of the optimal in all cases, which is considered reasonable in parallel computation. These results also indicate, however, that there is still an opportunity to find new algorithms that make better use of partial Hessian information.

6. Concluding Remarks

Recent research in unconstrained optimization has demonstrated that even though the field has reached a fairly mature state, many interesting and potentially fruitful research possibilities still exist. The research on the BFGS, DFP, and their convex combinations show that there are still fundamental theoretical issues concerning secant updates that need to be better understood. The research on the SR1 and on updates beyond the BFGS ($\phi<1$) illustrates that there may be updates that perform better than the BFGS in practice, but that we need to understand these updates better in both practice and theory. It appears increasingly unlikely, however, that any new secant update will result in a large improvement, say greater than 25%, over the BFGS.

The research on derivative tensor methods for nonlinear equations and unconstrained optimization has led to surprisingly large improvements in efficiency over standard methods, often in the range of 30-50%. Since these methods constitute just one of many possibilities for incorporating additional function or derivative information into optimization algorithms by using nonstandard models, this research indicates that there may be other interesting unexplored possibilities for utilizing nonstandard models in

unconstrained optimization algorithms.

Research in parallel optimization methods is still in its infancy. Much of the research described in Section 5 can be considered to be fairly straightforward adaptations or generalizations of standard methods. We consider it likely that more novel parallel optimization research will result from considering parallel methods for specific classes of large scale optimization problems. Indeed, the consideration of specific classes of large scale optimization problems, which has been neglected in this paper, probably holds many of the future challenges for research in unconstrained optimization.

7. References

C. G. Broyden [1970], "The convergence of a class of double-rank minimization algorithms", Parts I and II, *Journal of the Institute of Mathematics and its Applications* 6, pp. 76-90, 222-236.

C. G. Broyden, J. E. Dennis Jr., and J. J. Moré [1973], "On the local and superlinear convergence of quasi-Newton methods", *Journal of the Institute of Mathematics and its Applications* 12, pp. 223-246.

R. H. Byrd, J. Nocedal, and Y. Yuan [1987], "Global convergence of a class of quasi-Newton methods on convex problems", *SIAM Journal on Numerical Analysis* 24, pp. 1171-1190.

R. H. Byrd, R. B. Schnabel, and G. A. Shultz [1988a], "Using parallel function evaluations to improve Hessian approximation for unconstrained optimization", *Annals of Operations Research* 14, pp. 167-193.

R. H. Byrd, R. B. Schnabel, and G. A. Shultz [1988b], "Parallel quasi-Newton methods for unconstrained optimization", Technical Report No. CU-CS-396-88, Department of Computer Science, University of Colorado at Boulder, to appear in *Mathematical Programming.*

A. R. Conn, N. I. M. Gould, and Ph. L. Toint [1986], "Testing a class of methods for solving minimization problems with simple bounds on the variables," Research Report CS-86-45, Faculty of Mathematics, University of Waterloo, Waterloo, Canada.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint [1987], "Convergence of quasi-Newton matrices generated by the symmetric rank one update", Report 87/12, Department of Computer Sciences, University of Waterloo, Waterloo, Canada.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint [1988], "Global convergence of a class of trust region algorithms for optimization with simple bounds", *SIAM Journal on Numerical Analysis* 25, pp. 433-460.

K. A. Connolly and J. Nocedal [1987], "Quasi-Newton methods for parallel processing", Technical Report NAM 05, Department of Electrical Engineering and Computer Science, Northwestern University.

W. C. Davidon [1959], "Variable metric methods for minimization", Argonne National Laboratory Report ANL-5990 (rev.).

W. C. Davidon [1975], "Optimally conditioned optimization algorithms without line searches", *Mathematical Programming* 9, pp. 1-30.

D. W. Decker and C. T. Kelley [1980a], "Newton's method at singular points I", *SIAM Journal on Numerical Analysis* 17, pp. 66-70.

D. W. Decker and C. T. Kelley [1980b], "Newton's method at singular points II", *SIAM Journal on Numerical Analysis* 17, pp. 465-471.

J. E. Dennis Jr. and R. B. Schnabel [1983], *Numerical Methods for Nonlinear Equations and Unconstrained Optimization*, Prentice-Hall, Englewood Cliffs, New Jersey.

L. C. W. Dixon and K. D. Patel [1982], "The place of parallel computation in numerical optimization IV, parallel algorithms for nonlinear optimisation", Technical Report No. 125, Numerical Optimisation Centre, The Hatfield Polytechnic.

R. Fletcher [1970], "A new approach to variable metric methods", *Computer Journal* 13, pp. 317-322.

R. Fletcher [1980], *Practical Method of Optimization, Vol. 1, Unconstrained Optimization*, John Wiley and Sons, New York.

R. Fletcher and M. J. D. Powell [1963], "A rapidly convergent descent method for minimization", *Computer Journal* 6, pp. 163-168.

P. D. Frank [1984], "Tensor methods for solving systems of nonlinear equations", Ph.D. Thesis, Department of Computer Science, University of Colorado at Boulder.

P. E. Gill and W. Murray [1972], "Quasi-Newton methods for unconstrained optimization", *Journal of the Institute of Mathematics and its Applications* 9, pp. 91-108.

P. E. Gill, W. Murray, and M. H. Wright [1981], *Practical Optimization*, Academic Press, London.

D. Goldfarb [1970], "A family of variable metric methods derived by variational means", *Mathematics of Computation* 24, pp. 23-26.

D. Goldfarb [1976], "Factorized variable metric methods for unconstrained optimization", *Mathematics of Computation* 30, pp. 796-811.

L. Grandinetti [1978], "Factorization versus nonfactorization in quasi-Newtonian methods for differentiable optimization," Report N5, Dipartimento di Sistemi, Universita della Calabria.

A. O. Griewank [1980], "Starlike domains of convergence for Newton's method at singularities", *Numerische Mathematik* 35, pp. 95-111.

A. O Griewank and Ph. L. Toint [1982], "On the unconstrained optimization of partially separable functions", in *Nonlinear Optimization 1981*, M. J. D. Powell ed., Academic Press, London, pp. 301-312.

S. P. Han [1986], "Optimization by updated conjugate subspaces," in *Numerical Analysis: Pitman Research Notes in Mathematics Series 140*, D.F. Griffiths and G.A. Watson, eds., Longman Scientific and Technical, Burnt Mill, England, pp. 82-97.

R. H. F. Jackson [1983], "Tensors, polyads, and high-order methods in factorable programming", Ph.D. Dissertaion, The George Washington University, Department of Operations Research, Washington, DC.

R. H. F. Jackson and G. P. McCormick [1986], "The polyadic structure of factorable function tensors with application to high-order minimization techniques", *Journal of Optimization Theory and Applications* 51, pp. 63-94.

F. A. Lootsma [1984], "Parallel unconstrained optimization methods," Report No. 84-30, Department of Mathematics and Informatics, Technische Hogeschool Delft.

G. P. McCormick [1983], *Nonlinear Programming: Theory, Algorithms and Applications*, John Wiley & Sons, New York.

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

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

M. R. Osborne and L. P. Sun [1988], "A new approach to the symmetric rank-one updating algorithm", Report NMO/01, Department of Statistics, Australian National University, Canberra, Australia.

M. J. D. Powell [1976], "Some global convergence properties of a variable metric method without exact line searches", in *Nonlinear Programming*, R. Cottle and C. Lemke, eds. AMS, Providence, R.I., pp. 53-72.

M. J. D. Powell [1986], "How bad are the BFGS and DFP methods when the objective function is quadratic?", *Mathematical Programming* 34, No. 1, pp. 34-47.

M. J. D. Powell [1987], "Updating conjugate directions by the BFGS method", *Mathematical Programming* 38, pp. 29-46.

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

R. B. Schnabel [1987], "Concurrent function evaluations in local and global optimization," *Computer Methods in Applied Mechanics and Engineering* 64, pp. 537-552.

R.B. Schnabel and T. Chow [1988], "Tensor methods for unconstrained optimization," (in preparation).

R. B. Schnabel and P. Frank [1984], "Tensor methods for nonlinear equations", *SIAM Journal on Numerical Analysis* 21, pp. 815-843.

R. B. Schnabel, J. E. Koontz, and B. E. Weiss [1985], "A modular system of algorithms of unconstrained minimization", *ACM Transactions on Mathematical Software* 11, pp. 419-440.

D. F. Shanno [1970], "Conditioning of quasi-Newton methods for function minimization", *Mathematics of Computation* 24, pp. 647-657.

G. A. Shultz, R. B. Schnabel, and R. H. Byrd [1985], "A family of trust region based algorithms for unconstrained minimization with strong global convergence properties", *SIAM Journal on Numerical Analysis*,22, pp. 47-67.

A. Stachurski [1981], "Superlinear convergence of Broyden's bounded Θ-class of methods", *Mathematical Programming*, 20, pp. 196-212.

P. Wolfe [1969], "Convergence conditions for ascent methods", *SIAM Review*, 11, pp. 226-235.

P. Wolfe [1971], "Convergence conditions for ascent methods II: some corrections", *SIAM Review*, 13, pp. 185-188.

Y. Zhang and R. P. Tewarson [1986], "On the development of algorithms superior to the BFGS method in Broyden's family of updates", Department of Applied Mathematics and Statistics Report AMS 86-69, State University of New York, Stony Brook, New York.