CONJUGATE GRADIENT VERSUS SPARSITY EXPLOITING QUASI NEWTON ALGORITHMS IN UNCONSTRAINED MINIMIZATION

Janin T. Jadotte

A Thesis

in

The Department

of

Mathematics

Presented in Partial Fulfillment of the Requirements
for the degree of Master of Science at
Concordia University
Montreal, Quebec, Canada

July 1980

ABSTRACT

CONJUGATE GRADIENT VERSUS SPARSITY EXPLOITING QUASI NEWTON ALGORITHMS IN UNCONSTRAINED MINIMIZATION

Janin T. Jadotte

This thesis examines unconstrained minimization problems from the particular point of view of sparsity. We study the two main types of gradient methods, namely Conjugate Gradient and Quasi Newton, with emphasis on a class of Sparsity Exploiting Quasi Newton algorithms. This class has been the subject of many recent papers and the resulting algorithms represent a significant improvement over the Conjugate Gradient and the standard Quasi Newton algorithms. We present some numerical results for a selection of these algorithms. Finally we compare the respective advantages and inconveniences of using either one of the different classes of algorithms.

ACKNOWLEDGEMENTS

This thesis is gratefully dedicated to my whole family, especially Lunie and to all who have been of some help during my studies.

My special thanks to Dr. A. Buckley for his time, encouragement and help during the last year. He initiated me to the subject of numerical analysis and suggested the present topic to me. His knowledge of the field of computation was appreciated.

Thanks also to Dr. D. Shanno of the University of Arizona at Tucson for his collaboration and thanks as well to Gail Wolfenden for her diligent typing.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

1.1 Formulation of the Problem

The problem we are interested in is to

$$\text{minimize} \quad f(x) \ , \ x \in E^n \tag{1.1}$$

where f is smooth, i.e. has at least continuous second derivatives. Many papers dealing with this problem have been published in the last few years. Our interest in the subject is based on the conjugate gradient (CG) methods as introduced by Hestenes and Stiefel [21] for solving linear systems and adapted to (1.1) by Fletcher and Reeves [17], and on the Quasi-Newton (QN) methods as introduced by Davidon [8] and clarified by Fletcher and Powell [18]. Together the two classes of algorithms form the main gradient types of algorithms. They require the computation of the objective function f and of its first derivatives only, although f may be assumed to have second or higher derivatives.

Recently McCormick and Ritter [25] considered the question of the relative merits of using each of these classes. Their analysis showed that QN methods converge faster than CG methods in general. The result confirmed what experimentation had already suggested. On the other hand, CG algorithms require only order n storage locations to be implemented while their QN counterparts normally require order n^2. So, for large n, a user may not be able to use QN methods when sufficient storage is not available. In this case, the usual alternative is to turn to CG algorithms because of their low requirement in terms of storage. In this thesis we will be particularly interested in these large problems.

In the last few years, two possible approaches to large problems have been investigated. The first one uses the fact that for large n, the matrix of second derivatives (the Hessian) is very often sparse. Assuming the exact sparsity pattern to be known, QN methods have been modified to take advantage of that fact in ~~order~~ to reduce storage requirements. In particular they do not revise the zero elements (or even known constants) of the Hessian at each iteration, unlike the standard QN methods. The resulting algorithms developed by Powell [37], Toint [43], Shanno [41] and Marwil [24] preserve the relatively fast convergence of the standard QN algorithms and at the same time do substantially reduce the order n^2 storage requirement. Meanwhile, new CG algorithms have been developed. A new class, the mixed CG and QN algorithms, uses to advantage some recently stressed features of the CG methods [27]. The modified CG algorithms preserve the order n storage requirement of their predecessors while improving their convergence. We will discuss those algorithms elaborated by Powell [34], Shanno [40] and Buckley [6].

In light of these developments, we ask ourselves: Is the CG class comparable to the QN class for certain functions of large size? To answer this question, we will first review in Chapter II the standard CG and QN methods. Then, in Chapters III and IV, we will present the ideas upon which the modified QN and CG methods are based as well as some specific algorithms. Finally, following considerations on the implementation of both classes, we will compare their relative effectiveness on some known test functions.

For the sake of comparison we select criteria which we feel is relevant. They are of two kinds: the first includes standard comparative statistics on the performance of an algorithm such as the number of iterations and the number of function evaluations needed to reach the minimum, and the execution time whenever possible. The second kind of criteria refers to the two classes of algorithms as such. They take into account the specific difficulties in implementing and using each class as will be discussed in Chapter V. As to our selection of algorithms to be tested, it includes those belonging to the modified CG class, the Sparsity Exploiting QN (SE QN) class and two standard algorithms from the CG and QN families. This choice, with our question in mind, was partially guided by the availability of test results for the algorithms.

Our idea is to compare as far as possible the two classes of methods, not specific algorithms. In doing so, we are aware of the difficulty of evaluating algorithms or classes of algorithms. Besides that, there are limitations such as the non-uniform implementation of the algorithms, the scarcity of results for problems of the size being considered and the limitation of time. Nevertheless, within those considerations, we will hopefully come up with a brief guideline for users of minimization routines which they may use to help in making a choice of algorithm. It is understood that a good decision will always depend on the user's evaluation of the situation.

1.2 Preliminaries

We will use a standard notation encountered in many publications. Unless otherwise specified, capital letters will denote $n \times n$ matrices and lower case letters will indicate column vectors or scalars. The context should make the distinction clear in the latter case. In the same vein, we now set the meaning of some symbols often used in this thesis.

Let B_k represent the approximation to the Hessian H at x_k. Similarly H_k approximates H^{-1} at x_k. We also set $s_k = x_{k+1} - x_k$, $f_k \equiv f(x_k)$, $g_k \equiv g(x_k) \equiv \nabla f(x_k)$ and $y_k = g_{k+1} - g_k$. Finally the notation $[d_1, d_2, \cdots, d_k]$ represents the subspace spanned by the set of vectors d_i, $i = 1,2,\ldots,k$.

Throughout this thesis we will consider iterative methods of approximating the solution to (1.1). Thus, given an initial estimate $x_1 \in E^n$, they generate a sequence of points $x_2, x_3, \ldots$ which hopefully will converge to a local minimum of f, say x^*. The k-th iteration sets

$$x_{k+1} = x_k + \lambda_k d_k , \qquad (1.2)$$

where the point x_k and the search direction d_k are known from the previous iteration. We find x_{k+1} by minimizing $\phi(\lambda)$, where

$$\phi(\lambda) = f(x_k + \lambda d_k) . \qquad (1.3)$$

This particular minimization problem is called a line search. If it is solved exactly (as is possible for quadratic f), it is said that it is an Exact Line Search (ELS).

Now, assuming that the sequence of points defined by (1.2)-(1.3) converges, it is of prime importance to know the rate of convergence. This gives a measure of the effectiveness of the algorithm and is as important as the fact that the algorithm converges.

Definition 1. Assume that an algorithm generates a sequence of points $\{x_k\}$ convergent to x^*. If for some norm $\|\cdot\|$, there is an $\alpha \in [0,1)$ and $k_0 \geq 0$ such that

$$\| x_{k+1} - x^* \| \leq \alpha \| x_k - x^* \| \quad , \; k \geq k_0 \tag{1.4}$$

then the algorithm is said to be linearly convergent.

Definition 2. Consider the same sequence of points as in *Definition 1.* If for some norm $\| \cdot \|$, there is a scalar α and $k_0 \geq 0$ such that

$$\| x_{k+1} - x^* \| \leq \alpha \| x_k - x^* \|^2 \quad , \; k \geq k_0 \tag{1.5}$$

then the algorithm is quadratically convergent.

Both rates of convergence defined by (1.4) and (1.5) are extreme; linear convergence is considered poor and quadratic convergence is rather an ideal for an algorithm and rare. Most of the known algorithms exhibit a convergence between the two extremes: it is called a superlinear rate of convergence.

Definition 3. An algorithm is superlinearly convergent if there exists a set of α_k, where $\{\alpha_k\}_{k=1}^{\infty} \to 0$, such that the relation

$$\| x_{k+1} - x^* \| \leq \alpha_k \| x_k - x^* \| \tag{1.6}$$

is true.

Finally, in all the occurences where the objective function f will be assumed quadratic (there will be many), we will consider it to have the form

$$f(x) = q(x) = \frac{1}{2} x^T A x + b^T x + c \; .$$

CHAPTER II

STANDARD CG AND QN ALGORITHMS

2.1 Standard CG Algorithms

The CG method was first developed in 1952 by Hestenes and Stiefel [21] for the solution of linear systems $Ax = c$. Their procedure amounted to an n-step path to the solution. In 1964, Fletcher and Reeves [17] adapted the method to the closely related unconstrained minimization problem (1.1). Indeed, assuming that the objective function is quadratic, we can see that solving the necessary conditions for a local minimum, $g(x) = Ax + b = 0$, is equivalent to solving a linear system. Furthermore their method was developed so as to be applicable to non-quadratic functions. The original CG algorithm for problem (1.1) showed some advantages over QN algorithms and many variants on its basic form have ensued in the last few years. The whole CG class shares the property of requiring only a few n-vectors of storage and they attain the minimum in at most n steps for f quadratic. The former feature makes them particularly valuable for large problems.

We will introduce in this chapter the concept of conjugacy in relation to the conjugate direction methods. Then the original CG algorithm will be presented along with some of the basic properties of the CG class in the quadratic case. Also we will derive an important generalization of the original CG algorithm. Finally a discussion of the application of CG methods to general functions will follow.

Definition. A set of vectors d_k in E^n is said to be conjugate with respect to the symmetric positive definite matrix A if and only if $d_i^T A d_j = 0$ for $i \neq j$.

Lemma 1. If a set of nonzero vectors $d_k (k=1,\dots,n)$ is conjugate with respect to a symmetric positive definite matrix A, then they are linearly independent.

Proof: See Luenberger [23], for instance. ■

The idea of conjugacy is the basis for the conjugate direction methods. Consider the problem of minimizing a quadratic function. As we have mentioned earlier, it is equivalent to solving a linear system $Ax + b = 0$. Then, given an arbitrary point x_1 and $d_1,\dots,d_n$ conjugate, the solution, x^* say, according to *Lemma 1*, can be written as a linear combination of them, i.e.

$$x^* - x_1 = \alpha_1 d_1 + \dots + \alpha_n d_n \tag{2.1}$$

or

$$x^* = x_1 + \sum_{i=1}^{n} \alpha_i d_i \tag{2.2}$$

for some set of α_i. Assuming that we can construct the d_i's, the problem of finding x^* would reduce to finding the coefficients α_i. Usually the directions are not given but defined iteratively. Thus finding the solution x^* can be interpreted as an n-step iterative procedure where, at each step, a new term $\alpha_i d_i$ is determined.

The conjugate direction methods are based on this interpretation and accordingly, any quadratic function can be minimized in at most n steps. Furthermore, progress toward the minimum value of f is constant as this theorem states:

Theorem 1. (Expanding Subspace Theorem) Let $\{d_i\}_{i=1}^{n}$ be a sequence of conjugate directions with respect to A. Then for any $x_1 \in E^n$, the sequence generated by

$$x_{k+1} = x_k + \lambda_k d_k \tag{2.3}$$

has the property that x_{k+1} minimizes the quadratic f on the linear variety $x_1 + B_k$ (where $B_k = [d_1, d_2, \ldots, d_k]$), providing each line search is exact as in (1.3).

Proof. The reader is referred to Luenberger [23] for a detailed proof. ∎

The proof is based on the fact that x_{k+1} minimizes a function over a linear variety only if $g(x_{k+1}) \equiv g_{k+1}$ is orthogonal to that space, i.e. $g_{k+1} \perp B_k$. Thus it follows from the theorem that

$$g_{k+1}^T d_i = 0 \quad , \quad i \le k \quad . \tag{2.4}$$

Also the theorem shows that if $k = n$, then B_n is the entire space E^n and hence x_{n+1} is the overall minimum of f.

The conjugate gradient method is a conjugate direction method where the successive d_i are defined iteratively and are given by fairly simple formulae. The current search direction is taken as a linear combination of the current gradient and of the previous search direction subject to a conjugacy requirement. We give a definition of the original CG algorithm of Fletcher and Reeves.

Given an arbitrary $x_1 \in E^n$, set $d_1 = -g_1$. Then for $k \ge 1$, until $g_{k+1} = 0$, iterate on the steps:

$$x_{k+1} = x_k + \lambda_k d_k \ , \tag{2.5a}$$

$$\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k} \ , \tag{2.5b}$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k \ , \tag{2.5c}$$

where λ_k in (2.5a) is determined by a line search.

Recently, Polak and Ribière [32] considered an alternate form of β_k. Their implementation is similar to that of Fletcher and Reeves except that

$$\beta_k = \frac{g_{k+1}^T(g_{k+1} - g_k)}{g_k^T g_k} \quad . \tag{2.6}$$

In fact, both choices of β_k are made to produce conjugate search directions d_k and are equivalent for f quadratic, as we will see later. Now we state and prove the fundamental theorem of CG algorithms.

Theorem 2. Consider applying algorithm (2.5) to a quadratic f. Then, if it does not terminate at x_k, we have

a) $[g_1, g_2, \ldots, g_k] = [g_1, Ag_1, \ldots, A^{k-1}g_1]$; (2.7)

b) $[d_1, d_2, \ldots, d_k] = [g_1, Ag_1, \ldots, A^{k-1}g_1]$; (2.8)

c) $d_k^T A d_i = 0 \, , \quad i \le k - 1$. (2.9)

Proof. We prove a), b) and c) simultaneously by induction. First they are true for $k = 1$, by default. Now suppose that they are true for $j \le k$.

Then consider (2.5a). If we multiply by A and add the column vector b on both sides, we obtain

$$g_{k+1} = g_k + \lambda_k A d_k \, . \tag{2.10}$$

By the induction hypotheses, both g_k and Ad_k belong to $[g_1, \ldots, A^k g_1]$. Thus,

$$g_{k+1} \in [g_1, Ag_1, \ldots, A^k g_1] \, . \tag{2.11}$$

At the same time, we have

$$g_{k+1} \notin [\, g_1, Ag_1, \ldots, A^{k-1}g_1] = [\, d_1, d_2, \ldots, d_k] \tag{2.12}$$

since by *Theorem 1*, g_{k+1} is linearly independent of d_i for $i \le k$ providing $g_{k+1} \neq 0$. Now (2.10) together with (2.11) and (2.12) imply that

$$[\, g_1, g_2, \ldots, g_{k+1}] = [\, g_1, Ag_1, \ldots, A^k g_1] \ , \tag{2.13}$$

which proves a).

To prove b), we consider (2.5c). By (2.13) and the induction hypothesis (2.8), the result follows.

Now to prove c) for $k+1$, we write, using (2.5c)

$$d_{k+1}^T A d_i = -g_{k+1}^T A d_i + \beta_k d_k^T A d_i \ . \tag{2.14}$$

For $i < k$, (2.14) is zero since the second term in the right-hand side vanishes because of the induction hypothesis c) and the first is also zero because of (2.4) and since $Ad_i \in [\, d_1, d_2 \ldots, d_{i+1}]$. For the case $i = k$, we need to show that β_k in (2.5b) is equivalent to $(g_{k+1}^T A d_k / d_k^T A d_k)$ and it will follow that (2.14) vanishes.

We mention that a) and b) imply that

$$g_k^T g_i = 0 \quad \text{for } i < k \ . \tag{2.15}$$

Then

$$g_{k+1}^T g_{k+1} = g_{k+1}^T (g_{k+1} - g_k) \tag{2.16}$$

$$= \lambda_k g_{k+1}^T A d_k \tag{2.17}$$

since $g_{k+1} - g_k = A(x_{k+1} - x_k) = \lambda_k A d_k$. Similarly

$$g_k^T g_k = (g_{k+1} - g_k)^T g_k = (g_{k+1} - g_k)^T (d_k + \beta_k d_{k-1})$$

$$= (g_{k+1} - g_k)^T d_k \qquad (2.18)$$

using (2.4). Finally

$$g_k^T g_k = \lambda_k d_k^T A d_k \ , \qquad (2.19)$$

and β_k defined by the ratio (2.16) over (2.19) makes $d_{k+1}^T A d_k = 0$. ■

The theorem, in particular c), shows that the algorithm (2.5) generates successive conjugate search directions. It also follows that the choice of β_k (2.6) will do it too, since (2.17) implies that both definitions of β_k are equivalent. So both algorithms (2.5), (2.6) will minimize a quadratic function in at most n iterations and in fact, the sequence of x_k will be exactly the same.

One surprising feature of these CG algorithms is that they lose their finite termination property if the starting search direction is not the steepest descent direction, i.e. if $d_1 \neq -g_1$. In this case, the rate of convergence is usually only linear [7]. However they can be modified so as to recover that property. We will study two different ways of doing it, which are referred to as the generalized or pre-conditioned CG algorithm and Beale's method.

The first method uses a transformation of variables,

$$z = H^{-\frac{1}{2}} x \ , \qquad (2.20)$$

H being any positive definite matrix. Then we apply the standard CG algorithm in the z-coordinates from $z_1 = H^{-\frac{1}{2}} x_1$. The resulting algorithm is a CG algorithm in the new coordinates and hence, finite termination holds. Thus we obtain a new set of search directions $\hat{d}_k$ which may be transformed back into the x-coordinates. We will show that the CG algorithm which then results is equivalent to a modified CG algorithm applied directly in the x-coordinates.

We will assume that f is quadratic and exact line searches are being used. The following relations are important for the proof:

$$\hat{g}(z) = H^{\frac{1}{2}} g(x) \quad , \tag{2.21}$$

$$\hat{d}(z) = H^{-\frac{1}{2}} d(x) \quad , \tag{2.22}$$

where $\hat{g}$ and $\hat{d}$ are respectively the gradient and the search direction in the z-coordinates. In that new space, the steps (2.5a), (2.5c) and (2.6) become

$$z_{k+1} = z_k + \lambda_k \hat{d}_k \quad , \tag{2.23a}$$

$$\hat{\beta}_k = \frac{\hat{g}_{k+1}^T (\hat{g}_{k+1} - \hat{g}_k)}{\hat{g}_k^T \hat{g}_k} \quad , \tag{2.23b}$$

and

$$\hat{d}_{k+1} = -\hat{g}_{k+1} + \hat{\beta}_k \hat{d}_k \quad , \tag{2.23c}$$

using the Polak-Ribière form of β_k . Note that the line searches are equivalent in both coordinates since

$$\hat{g}_{k+1}^T \hat{d}_k = g_{k+1}^T H^{\frac{1}{2}} H^{-\frac{1}{2}} d_k = g_{k+1}^T d_k \quad . \tag{2.24}$$

Then using the transformations (2.21), (2.22) backwards and assuming an ELS at each iteration, we obtain the new algorithm in the x-coordinates.

Given x_1 and an arbitrary symmetric positive definite matrix H , set $d_1 = -Hg_1$. Then for $k \geq 1$, iterate according to

$$x_{k+1} = x_k + \lambda_k d_k \quad , \tag{2.25a}$$

$$\beta_k = \frac{g_{k+1}^T H (g_{k+1} - g_k)}{g_k^T H g_k} \quad , \tag{2.25b}$$

$$d_{k+1} = -Hg_{k+1} + \beta_k d_k \quad . \tag{2.25c}$$

Theorem 3. The pre-conditioned CG algorithm (2.25) with $d_1 = -Hg_1$ has the finite termination property.

Proof. That follows by considering (2.21). If $\hat{g}(z) = 0$, then $g(x) = 0$ for the corresponding x since $H^{\frac{1}{2}}$ is non-singular. And, as we said earlier, $\hat{g}(z_m) = 0$ for an $m \leq n$ if $f(z) = f(H^{-\frac{1}{2}}x)$ is quadratic. Hence we have that $g(x_m) = 0$ for the same index m . ■

The pre-conditioned CG algorithm has some other properties similar to those of the standard CG algorithms in the quadratic case. For instance, the gradient vectors are conjugate with respect to the metric H ,

$$g_k^T H g_i = 0 \quad \text{for} \quad k > i \; . \tag{2.26}$$

But as a CG algorithm and in order to be practical, the matrix H must not be stored. Moreover, it must be chosen so that the number of arithmetic operations required to compute expressions of the form Hv stays within order n . Those ideas will be exploited in the definition of the mixed algorithms studied in Chapter IV.

Now to introduce Beale's method, we will do it, as is usual, in the context of applying CG algorithms to non-quadratic functions. Although we have so far examined only the quadratic case, the definition of the CG algorithms as stated is applicable to general functions since the matrix A is not explicit anywhere in them. However, we need to make some new considerations.

When applied to quadratic f , the standard CG algorithms terminate in at most n steps, provided that $d_1 = -g_1$. If not, it is known [7] that their convergence is usually only linear, even for a quadratic f. Thus for a general function, although the quadratic finite termination no longer holds, it is certainly necessary to start with the negative gradient. More can be said. Recall that a general smooth function, by Taylor's theorem, is approximately quadratic near the minimum. So, assuming that we start outside of the nearly quadratic region Q, we need a steepest descent step once we reach Q in order to obtain good ultimate convergence. This is especially clear if we imagine a smooth function which is precisely quadratic in Q. If we did not do the steepest descent step, convergence would be linear since we may consider the first point in Q as x_1. However if we have a mechanism to eventually ensure a steepest descent step in Q, finite termination will occur. The periodic use of the negative gradient direction is called a restart. In fact, CG algorithms using a restart strategy have been established to be superlinearly convergent every n steps. The problem is that we do not know when the nearly quadratic region Q has been reached.

Fletcher and Reeves [17] suggested a cycle of $(n+1)$ iterations, i.e.

$$d_{k+1} = -g_{k+1} \quad \text{for} \quad k=0,\ n+1,\ 2(n+1),\ \ldots \tag{2.27}$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k \quad \text{otherwise} . \tag{2.28}$$

The scheme proved to be superior in practice to applying the algorithms in a continuous fashion. But it has the disadvantage that when restarting the decrease in the value of the function along direction (2.27) is usually less than that along the corresponding d_k defined by (2.28). Note indeed that, since d_{k+1} and d_k are conjugate, by *Theorem 1*, a line search along (2.28) leads to the least value of f in the two-dimensional affine set that contains $-g_{k+1}$ and d_k, whereas in (2.27) the minimum is along $-g_{k+1}$ only. So one would like to be allowed to restart with $d_{k+1} \neq -g_{k+1}$ and still have the finite termination property.

Beale [1] addressed this problem and proposed a method which achieves the objective. He introduced the three-term recurrence formula

$$d_{k+1} = -g_{k+1} + \beta_k d_k + \gamma_k d_t , \tag{2.29}$$

where $t < k$ is the index of the last restart direction. Now the restart step is defined by (2.28) and the other steps by (2.29). Some features of Beale's method will be studied later in Chapter IV along with Powell's restart procedure. Here we just mention that in the quadratic case, the method develops conjugate search directions and finds the minimum of a quadratic f in at most n iterations.

2.2 Standard QN Algorithms

QN methods are based on Newton's method for solving a system of non-linear equations $F(x) = 0$, $x \in E^n$ and $F(x) = [f_1(x),\ldots,f_n(x)]^T$. Newton's method proceeds iteratively, i.e. from an initial estimate x_1 of the minimum x^*, it attempts by successive approximations to improve x_1. Starting with $k = 1$, it sets

$$x_{k+1} = x_k + d_k , \tag{2.30}$$

where the equation

$$F'(x_k)\cdot d_k = -F(x_k) \tag{2.31}$$

determines d_k, F' being the matrix of the first derivatives of F (the Jacobian). This algorithm can be easily related to the solution of the problem (1.1). Indeed we can look at the system $F(x) = 0$ as the necessary first-order condition to be satisfied by any stationary point x^* of a function f, when $F \equiv \nabla f$. Then we have that $F'(x) = \nabla^2 f(x)$ provided that f is at least C^2. Thus (2.31) can be rewritten

$$\nabla^2 f(x_k) \cdot d_k = -\nabla f(x_k) , \tag{2.32}$$

or

$$d_k = -[\nabla^2 f(x_k)]^{-1} \cdot \nabla f(x_k) , \tag{2.33}$$

assuming that the inverse Hessian exists at that point.

Newton's method has been analyzed extensively. It is known to converge quadratically under reasonable assumptions (see Ortega and Rheinboldt [30]). But that does not go without disadvantages. Broyden [3] has listed three of them and the most serious, in his view, is its failure to converge from a poor initial estimate x_1. Second, each iteration requires the solution of the system of linear equations (2.32) to get d_k. This is quite a costly operation since it requires order n^3 arithmetic operations. Finally, (2.32) assumes computation of the matrix $\nabla^2 f(x)$ at every iteration. This is not always possible or in any case, usually easy or convenient. Thus many modifications to the original Newton algorithm have been developed. The Newton-like family, as it is called sometimes, uses different techniques to take full advantage of the computation of the Hessian; they will not be considered here. Instead we are concerned with a particular class of variants, the Quasi-Newton (QN) class. There, computation of the Hessian will not be required. Also we will not have to solve systems of linear equations and a line search will be introduced in the definition of these algorithms to make them stable in the sense of a regular progress toward the minimum as in *Theorem 1*. We now present an introduction to a particular sub-class of QN algorithms; specific members of that family and other

QN algorithms will be considered later in the chapter.

Given an initial estimate x_1 and a positive definite matrix H_1 which is an approximation to the inverse Hessian at x_1, set

$$d_1 = -H_1 g_1 . \tag{2.34}$$

Then for $k \geq 1$, iterate on the steps

$$x_{k+1} = x_k + \lambda_k d_k , \tag{2.35a}$$

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{a_k} + \frac{s_k s_k^T}{b_k} + \beta_k a_k w_k w_k^T \tag{2.35b}$$

$$d_{k+1} = -H_{k+1} g_{k+1} , \tag{2.35c}$$

where step (2.35a) implies a line search along d_k to determine λ_k. In (2.35b), $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$ as set in the preliminaries; and

$$w_k = \frac{H_k y_k}{a_k} - \frac{s_k}{b_k} \tag{2.35d}$$

where $a_k = y_k^T H_k y_k$ and $b_k = s_k^T y_k$. Finally, the parameter β_k in (2.35b) (not to be confused with the same notation in 2.5b) defines the family of updates known as the Broyden family. For simplicity, we will refer to it as

$$H_{k+1} = U(x_{k+1}, H_k, \beta_k) . \tag{2.36}$$

The case $\beta_k = 1$ is known as the BFGS update and $\beta_k = 0$ gives the DFP update.

Historically, QN methods were developed to preserve, as much as possible, the fast convergence of their Newton predecessors while removing their main disadvantages. For instance the line search implied by

(2.35a) and absent in Newton's algorithm makes the algorithm more stable in that the value of the objective function f decreases at each iteration. The line search can require a substantial amount of computation, but it is considered as the price to pay for stability. To overcome the second disadvantage, QN algorithms usually approximate, in some sense, the inverse Hessian and so do not have to solve linear systems. Thus comparing (2.35c) to (2.33), we can identify H_k with the inverse Hessian $[\nabla^2 f(x_k)]^{-1}$, but in (2.35b), H_{k+1} is given by a simple formula. So, instead of solving a system of linear equations, we only have a matrix-vector multiplication. Finally, as we just said, the inverse Hessian is approximated; this avoids the task of computing the matrix $\nabla^2 f(x_k)$. This approximation H_k is revised or updated at each iteration as the definition (2.35b) indicates. We now stress an important property of the update procedure.

We first consider the case of a quadratic function, $f(x) = q(x)$, so that we have $\nabla^2 f(x) \equiv A$. Recalling that $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$, it is then deduced that

$$As_k = y_k \quad , \tag{2.37}$$

or

$$A^{-1} y_k = s_k \quad . \tag{2.38}$$

These relations are important in that they give us information about the Hessian or its inverse by evaluating the gradient at two points.

For general functions where the Hessian and its inverse are no longer constant, clearly equations (2.37) and (2.38) do not hold. But to the extent that a general function can be locally approximated by a quadratic, the same considerations as in the quadratic case apply. In

(2.38), A^{-1} may be replaced by an approximation to the true inverse Hessian H^{-1}, an approximation which will hopefully reflect the change in gradient from x_k to x_{k+1}. Thus any approximation to H^{-1} is often required to satisfy a relation similar to (2.38), namely

$$H_{k+1}y_k = s_k . \tag{2.39}$$

This equation is known as the QN or the Secant equation. In it, H_{k+1} simulates property (2.38) of A^{-1} ; in this sense, it approximates the inverse Hessian.

Note that the QN equation does not determine H_{k+1} uniquely. So more constraints are usually imposed on the matrix. Assuming that H_k was a good approximation to H^{-1}, H_{k+1} should retain as much as possible the desirable properties of H_k. In other words, H_{k+1} is updated from H_k and it is done in such a way that the information gathered at the last iteration, e.g. the value of g_{k+1}, is taken into account.

Let us consider the matrix updating problem in a more general framework which will lead us back to the Broyden update class (2.35b). We are concerned with various ways of arriving at an update formula. For instance, Broyden [2] suggested defining

$$H_{k+1} = H_k + a_k u_k z_k^T \tag{2.40}$$

where the scalar a_k and the vectors u_k and z_k are chosen so that H_{k+1} satisfies the QN equation. More generally, we write

$$H_{k+1} = H_k + E_k \tag{2.41}$$

where E_k is the correction or update matrix. It can be determined as a rank-1 matrix as in (2.40), or a rank-2 matrix as in (2.35b) or as a minimum norm matrix as will be discussed later. Thus, the update matrix in (2.40) is of rank one but is, in general, non symmetric. Often the update matrix is required to be symmetric and to satisfy the QN equation; then it is of rank two, in general. The minimum norm approach is somewhat different. There, given H_k, a formula for $E_k = H_{k+1} - H_k$ is derived so that $\|E_k\|$ is minimum for some norm and so that H_{k+1} satisfies (2.39). But it turns out that most of the minimum norm matrices are of rank one or of rank two. In this thesis, we will examine separately the minimum norm approach which proves effective in undertaking problems with specific requirements, such as sparsity in the Hessian of the objective function.

The rank-1 and rank-2 algorithms are closely related in many ways. For instance, it is shown in Huang [22] that both kinds of algorithms can be obtained from a method of developing algorithms so that they satisfy certain requirements, like the finite termination property in the quadratic case. Thus rank-1 and rank-2 algorithms are members of a broad class due to Huang. Alternately, most of the well-known rank-2 algorithms may be derived from the appropriate rank-1 algorithm using a symmetrization technique suggested by Powell [33] and developed by Dennis [9].

For minimization problems where the Hessian is usually symmetric, all rank-1 algorithms but one have the undesirable feature of generating non-symmetric updates. The unique symmetric rank-1 update is derived by considering an update of the form

$$H_{k+1} = H_k + a_k z_k z_k^T \qquad (2.42)$$

subject to the QN equation. It follows that

$$H_{k+1} = H_k + \frac{(H_k y_k - s_k)(H_k y_k - s_k)^T}{(H_k y_k - s_k)^T y_k} . \tag{2.43}$$

The resulting algorithm, the symmetric single rank (SSR), has been studied by many authors, for example, Dixon [13] and showed good promise. It had some desirable properties such as finite termination, since $H_{n+1} = A^{-1}$ when f is quadratic, and it performed well without exact line searches. But usually the updates H_{k+1} were not positive definite and the search directions (2.35c) did not define descent directions. To circumvent this numerical difficulty, some updating strategies were suggested. Although the modified versions indeed improved the performance of the original SSR, even better results are obtained by some rank-2 algorithms. We will discuss possible reasons for that.

We study the family of rank-2 algorithms (2.34)-(2.35) for many reasons. One of its members, the BFGS algorithm, is considered currently as the most effective for the solution of (1.1). The same BFGS has some specific properties which are being used in mixed CG and QN algorithms and also it performs well after appropriate modifications for sparse problems. First we examine the properties of the whole Broyden family.

Broyden's family of algorithms is that subclass of the Huang class defined by (2.34)-(2.35) where the correction matrix to H_k is symmetric, of rank two and satisfies the QN equation. It can be seen from (2.35b) that H_{k+1} is symmetric provided that H_k is. Thus, i) if H_1 is chosen symmetric, then every $H_k (k \geq 1)$ in the sequence of updates generated by the Broyden family of updates is symmetric. Similarly (although not so obviously), it can be proved that H_{k+1} as given by (2.35b) preserves positive definiteness. Indeed, if H_k is positive definite, if ELS are carried out and if $\beta_k \geq 0$, then H_{k+1} is also positive definite (see Powell [36]). Even the assumption of ELS can be relaxed; the line search is required to be accurate enough for H_{k+1} to be positive definite. Hence, ii) if H_1 is chosen positive definite, line searches are exact or sufficiently accurate and β_k non-negative for all k, then every $H_k (k > 1)$ is positive definite in the sequences of updates generated by (2.35b).

An immediate consequence of ii) is that all search directions (2.35c) are downhill directions as

$$d_{k+1}^T g_{k+1} = -g_{k+1}^T H_{k+1} g_{k+1} < 0 . \tag{2.44}$$

This insures, at least in theory, that the algorithm considered is stable. Finally, as an important property of the Broyden family, we prove that they have the finite termination property for f quadratic.

Theorem 4. Assume that $f(x) = q(x)$ and ELS are used at each iteration. Consider applying the family of algorithms (2.34)-(2.35) to f with H_1 positive definite and $\beta_k \geq 0$ for all $k \geq 1$. Then there exists an integer $1 \leq m \leq n + 1$ such that $x_m = x^*$; if $m = n + 1$, then $H_{n+1} = A^{-1}$.

Proof. The proof will be done by induction; we will assume that we are at an iteration number $m - 1$. We will prove that

$$s_i^T A s_j = 0 \quad 1 \le j < i < m \tag{2.45}$$

and $$H_m y_j = s_j \quad 1 \le j < m \quad . \tag{2.46}$$

Relation (2.45) indicates that the search directions are conjugate with respect to A since $s_k = x_{k+1} - x_k = \lambda_k d_k$. By the conjugacy of the vectors d_i , the minimum will be attained in at most n iterations, i.e. for some $1 \le m \le n + 1$, we will have $x_m = x^*$ by *Theorem 2*.

First we state a relation which we need in the proof. Thus,

$$H_{k+1} A s_k = H_{k+1} y_k = s_k \tag{2.47}$$

which is true by (2.37) and the QN equation.

Now the start the proof, (2.45) and (2.46) are true for $m = 1$, by default. Assuming now that they are true for $m = p$, we prove that they hold for $m = p + 1$. Using the definition of s_i , i.e. $s_i = x_{i+1} - x_i$, we can write

$$g_p = Ax_p + b = b + A(x_{k+1} + s_{k+1} + s_{k+2} + \dots + s_{p-1}) \ ,$$

$$= g_{k+1} + A(s_{k+1} + s_{k+2} + \dots + s_{p-1}) \ . \tag{2.48}$$

By the induction hypothesis (2.45) and the Expanding Subspace Theorem,

$$s_k^T g_p = 0 \ , \quad 1 \le k < p - 1 \tag{2.49}$$

Then $$s_k^T A H_p g_p = s_k^T g_p = 0 \ , \tag{2.50}$$

using (2.47). Relation (2.50) can be written

$$s_k^T A s_p = 0 \quad \text{for} \quad 1 \le k < p - 1 \tag{2.51}$$

which proves (2.45) for $m = p + 1$.

Finally, assuming ELS and using update formula (2.35b), we have

$$H_{p+1}As_k = H_pAs_k \quad \text{for} \quad 1 \le k < p \tag{2.52}$$

$$= s_k \quad , \quad 1 \le k < p \tag{2.53}$$

by induction.

Now by taking $i = n + 1$ in (2.46) implies that $H_{n+1}A\Delta = \Delta$, where $\Delta = (s_1, s_2, \ldots, s_n)$ is a $n \times n$ non-singular matrix. Thus, we obtain

$$H_{n+1} = A^{-1} \quad . \quad \blacksquare$$

In practice, algorithms frequently depart from the theoretically expected behaviour. For instance, H_k in (2.35b) may be positive definite in theory and yet $y_k^T H_k y_k$ may be practically zero if H_k is nearly singular because of round-off errors. Then H_{k+1} is no longer computable and the algorithm needs a strategy to go on. For that and other reasons, there exist many implementations of the same algorithm using different techniques to circumvent numerical difficulties. One important factor in implementing an algorithm is the line search used. Although the proofs of theorems usually assume exact line searches, it is well known that in practice most implementations do not perform exact line searches. Instead they use various strategies to compute λ_k; for example, given x_k and d_k, the next point $x_k + \lambda_k d_k$ may be acceptable whenever it gives an acceptable reduction in the function value.

Considering Broyden's family of algorithms, Dixon [12] proved that the whole family generates an identical sequence of points x_k provided that they are started with the same x_1, that ELS are used and

$\beta_k \geq 0$ for all k in (2.35b). Thus, Dixon's result suggested that the kind of line search used must play a great role in the reported differences among members of Broyden's family. This has been borne out in [13]. In this report, Dixon tested three members of the family with various implementations of line searches. Those members were the BFGS, the DFP and the SSR. From his numerical results, he concluded the superiority of the BFGS over the other two. And he partially attributes the effectiveness of the BFGS to not being as sensitive to inexact searches as is the DFP, for instance. Other experiments have been conducted that have reinforced these views, as for example in Shanno and Phua [42].

We conclude this introduction to the standard CG and QN methods by mentioning the relationship between the two classes. First, many well-known algorithms of both classes belong to the same Huang class that has the property of converging in at most n iterations for f quadratic. Moreover it is also well known that the two classes distinguish each other by their storage requirements. However we will study, in Chapter IV, a mixed class of algorithms which combine the CG properties such as the order n storage requirement to the QN features of using a matrix satisfying the QN equation (2.39). Some members of the new class also have the finite termination property, as we will see.

CHAPTER III

DERIVING SPARSE ANALOGUES OF FULL UPDATES

3.1 Introduction

We mentioned in Chapter II that an alternate way of looking at the update problem is the minimum norm approach. In this chapter we will examine this approach along with its application to sparse problems. The idea was first introduced by Greenstadt [20] for the minimization problem (1.1) and it can be used to derive most of the well-known rank-1 and rank-2 updates such as the BFGS, the DFP and the SSR. It also provides a geometrical interpretation of the update procedure and leads to new methods for solving sparse problems.

Thus we will study the concept of minimum norm update first in the non-sparse or full case. In particular, the BFGS and the DFP update will be derived through this approach as an example. Then we will present the basic ideas of row updating as well as a technique of symmetrization of rank-1 updates first suggested by Powell [23] and generalized by Dennis [9]. Finally we will see how norm minimization and symmetrization are applied to the derivation of sparse updates. In that matter we will examine Toint's derivation of his sparse algorithm and a new method to generate a sparse analogue to any given symmetric variable metric update due to Shanno [41].

3.2 Norm Minimization and Symmetrization in Non-sparse Problems

The general form of the update problem is convenient here, i.e. given H, the current approximate inverse Hessian, find $H^* = H + E$ where E is the update matrix. E is chosen so that H^* satisfies the QN equation, $H^*y = s$. Since the choice is not unique, Greenstadt reasoned that from all the solutions, H^* should be the closest to H, in some sense. This prevents too large an increase in H, thus enhancing the stability of the algorithm. He formulated the problem in terms of a minimum norm problem:

$$\min\|E\| \equiv \min\|H^* - H\| \tag{3.1}$$

subject to

$$Ey = s - Hy , \tag{3.2a}$$

$$E = E^T . \tag{3.2b}$$

Relation (3.2a) is equivalent to the QN equation. The most commonly used norm to define (3.1) is certainly the Frobenius norm where

$$\|A\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}^2 = \mathrm{Tr}(AA^T) . \tag{3.3}$$

The abbreviation Tr is for trace. In his paper, Greenstadt used a weighted Frobenius norm defined by

$$\|E\|_P^2 = \|PEP\|_F^2 = \mathrm{Tr}(WEWE^T) . \tag{3.4}$$

Here P is a symmetric non-singular matrix and $W = P^TP$ is positive definite.

To solve (3.1)-(3.2), setting $r = s - Hy$, he defined the Lagrangian function

$$\begin{aligned}\phi(E,\lambda,\Omega) &= \tfrac{1}{2}\mathrm{Tr}(WEWE^T) + \lambda^T(Ey-r) + \sum_{i=1}^{n}\sum_{j=i}^{n}\delta_{ij}(E_{ij}-E_{ji}) , \\ &= \tfrac{1}{2}\mathrm{Tr}(WEWE^T) + \mathrm{Tr}[(Ey-r)\lambda^T] + \mathrm{Tr}[\Omega(E-E^T)] .\end{aligned} \tag{3.5}$$

By minimizing ϕ with respect to E, he then obtained the rank-2 solution

$$E = \frac{1}{y^T M y}\left\{ sy^T M + Mys^T - Hyy^T M - Myy^T H - \frac{1}{y^T M y}[y^T s - y^T M y] Myy^T M \right\} \qquad (3.6)$$

where $M = W^{-1}$. One possible choice for M , suggested by Goldfarb [19], is H^* and that amounts to weighing E in (3.4) by $(B^*)^{\frac{1}{2}}$, where B^* is the update of the current approximate Hessian. This specific weight can be justified theoretically [11] . In fact, replacing M by H^* in (3.6) gives the famous BFGS formula which has been derived independently by Broyden [4] , Fletcher [6] and Shanno [39] using different approaches. Another member of the family (3.6) is the DFP update. It is a degenerate member since M has to be replaced by the non-positive definite matrix

$H - \left(\frac{y^T H y}{s^T y}\right)^{\frac{1}{2}} \frac{ss^T}{s^T y}$ to obtain the DFP formula. We will see later how a broader family of updates than (3.6) can be generated by the symmetrization technique.

Let us now look at the geometrical interpretation of the updating procedure. We said earlier that there are many symmetric solutions to the problem of finding $H^* = H + E$ such that $H^*y = s$. Since H^* simulates the inverse Hessian, it seems reasonable to restrict it to the set of matrices satisfying some specific property of the true inverse Hessian. In our example above, H^* must obey the QN equation and be symmetric. Thus one considers the set of matrices E satisfying (3.2); this is easily seen to form an affine set, say A . Then the minimum norm problem (3.1), i.e. find the closest H^* to H which is in A , amounts to projecting H orthogonally onto A . Dennis and Schnabel [11] have been able to derive many well-known updates in this way, i.e.

$$H^* = P_G^A(H) \quad , \tag{3.7}$$

or

$$B^* = P_F^A(B) \tag{3.8}$$

where P_G^A is the projection operator onto the affine set A with respect to the norm induced by the matrix G. They have also generated sparse updates in a similar way. But before going to the sparse case, we will examine the symmetrization technique of Dennis [9]. The technique is interesting in that it may be used to generate a broad class of full updates, as we will see. It is also readily applied to sparse updating.

Consider the general form of a rank-1 update for the inverse Hessian. It is given by solving the update problem: find $H^* = H + uz^T$ subject only to the condition that $H^*y = s$; here u and z are arbitrary. The solution can be seen to be

$$H^* = H + \frac{(s-Hy)z^T}{z^Ty} \tag{3.9}$$

for any z. Although the update (3.9) leads to good numerical results in solving systems of non-linear equations, it has the drawback of not being symmetric in general, a usual requirement in minimization problems. So Dennis constructed

$$\hat{H}^* = H + \frac{(s-Hy)z^T + z(s-Hy)^T}{z^Ty} - \theta zz^T \quad , \tag{3.10}$$

where the last two terms in the right-hand side of (3.10) make it symmetric, assuming that H is symmetric. Then θ is chosen so that $\hat{H}^*y = s$. To find this value, we post-multiply both sides of (3.10) by y and use the condition on $\hat{H}^*$. Thus,

$$s = Hy + (s-Hy) + \frac{z(s-Hy)^T y}{z^T y} - \theta z z^T y \;; \tag{3.11}$$

and
$$\theta = \frac{(s-Hy)^T y}{(z^T y)^2} \;. \tag{3.12}$$

Then, (3.10) becomes

$$\hat{H}^* = H + \frac{(s-Hy)z^T + z(s-Hy)^T}{z^T y} - \frac{(s-Hy)^T y}{z^T y} \cdot \frac{zz^T}{z^T y} \;. \tag{3.13}$$

After manipulation, we can see that the update terms in (3.13) and (3.6) are identical provided that there exists a positive definite $M = W^{-1}$ such that $z = My$. However such an M exists if and only if $z^T y > 0$. Hence, Dennis's family of updates (3.13) for H is larger than Greenstadt's (3.6). Note that the symmetrization technique is applicable to the rank-1 update for B . We will apply it in the same way in the sparse case later in this chapter. But to make the transition to the sparse case easy, we state some basic results of linear algebra.

First, we have that $\sum_{i=1}^{n} \ell_i \ell_i = I$ where ℓ_i is the column vector with 1 as its i-th entry and 0 elsewhere, and I is the $n \times n$ identity matrix. Then if D is an arbitrary $n \times n$ matrix, we can rewrite it as

$$D = \sum_{i=1}^{n} \ell_i \ell_i^T D = \sum_{i=1}^{n} \ell_i D_{i*}^T \tag{3.14}$$

or
$$= \sum_{i=1}^{n} D \ell_i \ell_i^T = \sum_{i=1}^{n} D_{*i} \ell_i^T \;, \tag{3.15}$$

where D_{i*} and D_{*i} are respectively the i-th row and the i-th column of D . The two sets of equalities are referred to as the row and the column forms of D .

3.3 History of Sparsity Exploiting QN Updates

The topic of sparse updating problems was first initiated by Schubert [38] for the solution of non-linear equations $F(x) = 0$ by QN methods, where we recall that $F(x) = [f_1(x),\ldots,f_n(x)]^T$ and $x \in E^n$. His motivation was that, especially for large problems, the expression for some f_j frequently happens to be free of some other variables. Explicitly, if $f_j(x)$ would have the form $x_k x_j^3$, for example, then it would follow that $\frac{\partial f_j(x)}{\partial x_i} = 0$ for $i \neq j$, $i \neq k$. Thus the Jacobian could have many zero entries in known locations. Schubert's idea was to reflect any sparsity present in the rows or columns of the Jacobian in its approximation. The observed convergence of the modified QN methods should be faster than that of the standard QN methods since the approximate Jacobian takes into account specific features of the true Jacobian.

The technique developed by Schubert may be applied to QN methods for the minimization problem (1.1). In this case, the Jacobian of F corresponds to the Hessian B of the objective function f, as we explained in introducing the standard QN methods. But, unlike those, the modified QN methods to be discussed now will directly approximate the sparse Hessian since, in general, sparsity is not preserved by the inversion of a matrix. Now, assuming the zero pattern of the Hessian to be known, the initial approximate Hessian B_1 is set to follow the same sparsity pattern. Thus its entries are set equal to zero whenever the corresponding element of the true Hessian is known; the other entries are arbitrary, but usually non-zero. Schubert's technique amounts to not updating the zero entries of $B_k (k > 1)$ throughout the iterations.

Based on that idea, the Sparsity Exploiting QN algorithms reduce the order n^2 storage locations requirement of the standard QN algorithms. Such updates can be generated through the symmetrization technique and the minimum norm approach.

3.4 Norm Minimization and Symmetrization in Sparse Problems

In our notation, while considering both methods, we will let S be a symmetric matrix each of whose entries is 0 or 1; it will represent the sparsity pattern of the Hessian. We also define $\mathcal{S}$ as the set of all matrices having the sparsity pattern of S . Thus for the rest of the chapter we will assume the current approximate Hessian B to be in $\mathcal{S}$.

Consider Broyden's rank-1 update for B ; it is the analogue of (3.9) for B ,

$$B^* = B + \frac{(y-Bs)z^T}{z^T s} \quad . \tag{3.16}$$

The second term in the right-hand side of (3.16) is the update matrix E which we want to be in $\mathcal{S}$, i.e. we want

$$E_{ij} = 0 \text{ whenever } S_{ij} = 0 \ . \tag{3.17}$$

For that, define a vector $z(i) = D_i z$ where D_i is a diagonal matrix and $(D_i)_{jj} = S_{ij}$. The matrix D_i is in fact reflecting the sparseness of the i-th row of B in $z(i)$, and for any v , $v \cdot z(i)^T$ has its j-th column equal to 0 if $S_{ij} = 0$, for then $(D_i)_{jj} = 0$ and $z(i)_j = 0$, where $z(i)_j$ is the j-th element of the vector $z(i)$. It is then easily verified that the update

$$B^* = B + \frac{(y-Bs)z(i)^T}{z(i)^T s} \tag{3.18}$$

satisfies the sparsity requirement (3.17) and the QN equation for B^*, i.e. $B^*s = y$. Although the algorithms using this update have been successful in solving sparse non-linear equations, as in (3.9), it again has the major drawback of being generally non-symmetric.

Recently, Shanno [41] adapted the symmetrization technique of Dennis to generate a family of sparse updates. Starting with (3.18), the sparse analogue to the general full rank-1 update, he rewrote it in row form

$$B^* = B + \sum_{i=1}^{n} \ell_i \left\{ \frac{\ell_i^T(y-Bs)}{z(i)^T s} \cdot z(i)^T \right\}, \tag{3.19}$$

$$= B + \sum_{i=1}^{n} \alpha_i \ell_i z(i)^T , \tag{3.20a}$$

where
$$\alpha_i = \frac{\ell_i^T(y-Bs)}{z(i)^T s} . \tag{3.20b}$$

Since (3.20) is not symmetric for general $z(i)$, he considered

$$B^* = B + \sum_{i=1}^{n} \gamma_i (\ell_i z(i)^T + z(i)\ell_i^T) . \tag{3.21}$$

The choice of the scalars γ_i to provide symmetry but yet to satisfy the QN equation in (3.21) is totally analogous to the choice of θ in (3.10). Shanno pointed out that the new update is in S since the $z(i)$ reflect the sparseness of B, because of the definition of $z(i)$ and since B is symmetric. The γ_i's are determined by solving $B^*s = y$, i.e.

$$\sum_{i=1}^{n} \gamma_i (\ell_i z(i)^T s + z(i)\ell_i^T s) = y - Bs . \tag{3.22}$$

In matrix form, (3.22) becomes

$$\hat{S}\gamma = y - Bs \quad , \tag{3.23a}$$

where
$$\hat{S} = \sum_{i=1}^{n} z(i)^T s \ell_i \ell_i^T + \sum_{i=1}^{n} \ell_i^T s z(i) \ell_i^T \quad . \tag{3.23b}$$

Here the matrix $\hat{S}$ can be seen to be in $\mathcal{S}$.

In short, the symmetrization technique allows one, at least theoretically, to derive a sparse update from any given full rank-1 update. We will discuss later in the chapter the advantages and inconveniences of the technique. Now we examine how Toint derived a sparse update using the minimum norm approach.

Toint's formulation of the updating problem is similar to Greenstadt's but since the Hessian is approximated, the minimum norm matrix is in terms of B , unlike in (3.1). Thus he solved the problem:

$$\min \| \hat{E} \| = \min \| B^* - B \| \tag{3.24a}$$

subject to

$$\hat{E}s = y - Bs \quad , \tag{3.24b}$$

$$\hat{E} = \hat{E}^T \quad , \tag{3.24c}$$

$$\hat{E} \in \mathcal{S} \quad . \tag{3.24d}$$

For the time being, we assume with Toint that the diagonal elements of B are not zeroes, i.e. for any i , f is at least quadratic in its i-th variable. The solution to (3.24) is found by minimizing a Lagrangian function similar to (3.5) with an additional term $\sum_{i,j} \varepsilon_{ij} \hat{E}_{ij}$, where the sum is over the set of (i,j) such that $S_{ij} = 0$, to take into account the sparsity condition. Toint then deduced a simple update formula for $\hat{E}$,

$$\hat{E}_{ij} = \gamma_i s(i)_j + \gamma_j s(j)_i \quad . \tag{3.25}$$

The vector γ is given by the system

$$Q\gamma = y - Bs \tag{3.26a}$$

where
$$Q_{ij} = s(i)_j s(j)_i + \sum_{k=1}^{n} [s(i)_k]^2 \delta_{ij} \ , \tag{3.26b}$$

and δ_{ij} is the Kronecker delta. He went on by noting that $Q \in \mathcal{S}$ provided that the diagonal elements of $\hat{E}$ are updated. Indeed the definition of $s(i)$ and D_i , i.e. $s(i) = D_i s$ implies that we have $s(i)_j = 0$ whenever $S_{ij} = 0$, and the second term of (3.26b) refers only to diagonal elements of Q . Also the solution to (3.26) is well defined since Q has been proved in [43] to be positive definite if none of the vectors $s(i)$ is identically zero.

Toint's update is a member of the family of sparse updates defined by (3.23) where $z(i) = s(i)$ for all i . Indeed it can be seen that the matrices Q and $\hat{S}$ are identical if $z(i) = s(i)$. Thus, as in the non-sparse case, the symmetrization technique generates a broader class of updates than norm minimization. Note that, unlike the matrix Q , $\hat{S}$ may be singular and the solution to the system (3.26) is then undefined. On the other hand, Shanno [41] noted that a scaling of $z(i)$ may prove useful, i.e. again $z(i) = D_i z$ but now $(D_i)_{jj} = a_{ij} S_{ij}$ where the choice of the scaling factors a_{ij} is made to render $\hat{S}$ both symmetric and positive definite under some conditions. However, it is not clear in which cases this scheme may be applied and what the cost would be.

Now, the sparse algorithms resulting from both methods, especially Toint's, share the feature that they reduce the update problem to that of solving a sparse system, for which there exist good computer packages. Indeed, the bulk of the computation required during a single iteration amounts to solving two sparse systems of linear equations. First, to update B, the vector γ is determined by solving (3.26). Second, the search direction is found by the step

$$B^*d = -g \quad , \tag{3.27}$$

the equivalent step to (2.35c) where the approximate inverse Hessian was available. The crucial point is that both matrices Q(or $\hat{S}$) and B^* have the same sparsity pattern, as we noted earlier, so the same package can be used to solve both systems.

In Toint's algorithm, Q and B^* have exactly the same sparsity pattern provided that the diagonal elements of B are updated, as he assumed. This restriction is not really needed, as Shanno [41] pointed out provided that Q' is still sparse. For instance, if we know that so diagonal elements of the true Hessian are known constants, we naturally do not wish to update those entries once they are properly set in B_1. We will therefore treat them as zeroes while updating. Explicitly, if $(B_1)_{ii}$ is constant, we will set $S_{ii} = 0$ and require $\hat{E}$ to be in the new set of sparse matrices $\mathcal{S}$. But Q_{ii} will not be zero because of the term $\| s(i) \|^2$. However, Q will still have the same sparsity pattern as B and the system (3.26) will be sparse. The case where the known constant is an off-diagonal element is treated similarly and in this case, Q can only have fewer zeroes than the sparse B.

The question of the amount of computation required in a sparse algorithm compared to that in a standard QN algorithm will be discussed in Chapter V. Also, considerations will be given to the relative cost of using a sparse algorithm.

3.5 Shanno's Sparse Update

In light of the known effectiveness of the BFGS algorithm, it would be desirable to have a sparse version of it using one of the approaches of the last section. On the other hand, Shanno [41] found that the BFGS update for B is not a member of the symmetrization class. Since this class contains the norm minimization class of updates for B, as was the case for H, he concluded that the latter class can not be used to generate the BFGS update for B. That has the consequence that, for sparse problems, the class of updates derived by considering the weighted norm case of (3.24) with the same constraints will not contain a sparse version of the BFGS update. Alternately, Shanno generated such an analogue by a technique similar to (3.21). However he noted that the algorithm proves unstable and performs poorly. Hence, a sparse version of the BFGS had to be found another way, which is part of what Shanno [41] and Toint [45] have done.

Consider Toint's method; the update B^* can be seen as

$$B^* = P_I^V(B) \quad , \tag{3.28}$$

where V is the affine set of matrices which are sparse, symmetric and satisfy the QN equation, and I is the identify matrix (unweighted case). The new method is a two-stage procedure that will reduce to (3.28) if there is no sparsity constraint. First, a standard symmetric QN update of the form $B^* = B + E$ is considered. Then its sparse analogue is derived as

$$\hat{B}^* = P_I^V(B^*) \quad . \tag{3.29}$$

Although the update (3.29), at least at first sight, looks more complicated than (3.28), it does not require more computation as we will see.

The update has been obtained independently by Toint [45] and Shanno [41]; the latter gave numerical results for the sparse BFGS. However, the presentation he gave is general and applies to any symmetric QN update.

Given B, Shanno first updated it using any symmetric QN update, i.e. $B^* = B + D$. In general, the update D does not satisfy the sparsity requirement, $D \notin \mathcal{S}$ and hence, B^* will not preserve the sparsity pattern of B. So he considered modifying this update B^* to

$$\hat{B}^* = B^* + \hat{E} \quad , \tag{3.30}$$

where $\hat{E}$ is the solution to the unweighted minimum norm problem:

$$\min\|\hat{E}\| = \min\|\hat{B}^* - B^*\| \tag{3.31a}$$

subject to

$$\hat{E}s = 0 \tag{3.31b}$$

$$\hat{E} = \hat{E}^T \tag{3.31c}$$

$$B^* + \hat{E} \in \mathcal{S} \quad . \tag{3.31d}$$

The three constraints are to satisfy the QN equation for $\hat{B}^*$, the sparseness and the symmetry of $\hat{B}^*$. The solution is again found by minimizing a Lagrangian function

$$\phi(\hat{E},\mu,\Omega,\Delta) = \frac{1}{2}\mathrm{Tr}(\hat{E}\hat{E}^T) - \mathrm{Tr}(\hat{E}s\mu^T) - \mathrm{Tr}[\Omega(\hat{E}-\hat{E}^T)] - \mathrm{Tr}[\Delta(B^*+\hat{E})], \tag{3.32}$$

where $\Delta_{ij} \neq 0$ if $S_{ij} = 0$ otherwise. After finding $\hat{E}$, Shanno rewrote (3.30) as

$$\hat{B}^* = \overline{B}^* + \sum_{i=1}^{n} \mu_i(\ell_i s(i)^T + s(i)\ell_i^T) \quad . \tag{3.33}$$

Here $\overline{B}^*_{ij} = B^*_{ij}$ whenever $S_{ij} = 1$, and 0 otherwise; the vectors $s(i)$ are defined as previously. The μ_i's represent the elements of the vector μ and are determined by solving the system $\hat{E}s = 0$. Thus,

$$\sum_{i=1}^{n} \mu_i (\ell_i s(i)^T + s(i)\ell_i^T)s = 2\tilde{B}^*s \quad , \qquad (3.34)$$

where, unlike $\bar{B}^*$, $\tilde{B}^*_{ij} = B^*_{ij}$ if $S_{ij} \neq 0$, and 0 otherwise.

Looking at (3.34) and the equivalent of (3.27) for $\hat{B}^*$, the computation again involves mainly the solution of two sparse systems. In particular, Shanno noted that (3.34) always has a well-defined solution, as the coefficient matrix of μ is the same as its equivalent in Toint's algorithm and so is positive definite. Moreover, in terms of the full matrix B^*, equation (3.33) indicates that its entries corresponding to non-zero entries of $\hat{B}^*$ are updated. More important, only those elements need to be computed and stored. The other entries of B^* must be computed but are not stored, as (3.34) shows that only the vector $\omega = 2\tilde{B}^*s$ has to be stored. Thus Shanno's method uses no more storage than Toint's algorithm.

Analytically, the method has the interesting feature that both update matrices B^* and $\hat{B}^*$ satisfy the QN equation, as implied by (3.31b). Shanno observed: "As s is the direction in which we have obtained information about the function, this means that B^* and $\hat{B}^*$ retain the same information along this direction". At the same time, $\hat{B}^*$ has incorporated the information about the sparseness of the Hessian.

Finally note that the class of updates generated by (3.33)-(3.34) has the feature that $\hat{B}^*$ can not be guaranteed to be positive definite, in general. Considering the importance of positive definiteness in the standard QN algorithms, Toint proved a theorem about the existence of such positive definite updates $\hat{B}^*$. The theorem can be stated this way:

Theorem 5. (Toint [45]). Assuming some mild condition on the sparse matrix B and that $y^T s > 0$, then if B^* is any matrix satisfying the QN equation, symmetry and the sparsity conditions of B, there exists a $\bar{\sigma} \geq 0$ such that for all $\sigma \geq \bar{\sigma}$, the matrix $\hat{B}^* = B^* + \sigma\hat{E}$ satisfies the same conditions and is positive definite for a given $\hat{E}$.

This is just an existence theorem but it would be interesting to see its consequences on the sparse update of B corresponding to the SSR. In this case, B^* would be found by making B sparse and symmetrizing it as in (3.23). Then a positive definite $\hat{B}^*$ should be generated as the theorem indicates. The non-sparse symmetric single rank (SSR) is appealing because of its simplicity, although rank-2 updates were preferred for non-sparse problems as they usually generate positive definite B^*, unlike the SSR. But for sparse problems and in particular in relation to Toint's theorem, we note that the intermediate B^* is not required to be positive definite. Thus, the choice of the SSR update is possible, even though it is not positive definite. Moreover the existence of a $\hat{B}^*$ such that the system $\hat{B}^*d = -g$ is well defined and guaranteed under mild conditions. But the cost of such an algorithm in terms of computation required is still to be assessed.

Toint also voiced the opinion that sparse analogues to full updates such as the Self Scaling Variable Metric (SSVM) should be tried. Those numerical results would give a better idea of the performance of the class of updates defined by (3.33)-(3.34) and tell whether there is a member performing substantially better than the others.

CHAPTER IV

MODIFIED CG ALGORITHMS

4.1 Introduction

The standard CG algorithms, as we have seen in Chapter II, have the important property that they do not require storage of any matrices, unlike the standard QN algorithms. Only order n locations of storage are needed for their implementation. This is of crucial importance for large n since then n^2 is far larger than n . But their observed rate of convergence is slower than that of their QN counterparts and in general, more iterations are required by the CG algorithms to attain a comparable accuracy in approximating the minimum. In order to partially overcome this disadvantage, many improved CG algorithms have been recently devised with the common feature of not storing matrices. In this chapter we will look over three of them. The first is based on Beale's method which we introduced in Chapter II and which we will now examine further. We will then see how Powell's considerations on the problem of restarting an algorithm, based on Beale's method, have led to an effective algorithm known as the Powell restart algorithm. The second algorithm belongs to a class which combines CG and QN properties and which exploits a special relationship between the BFGS and the CG algorithms discovered by Nazareth [27] and Buckley [5] . A particular mixed algorithm due to Buckley [6] will be examined. Finally a CG algorithm due to Shanno [40] will be presented; it was inspired by Perry's work [31] .

4.2 Powell's Restart Algorithm

First we recall how Beale had come to his three-term recurrence (2.29). If we apply, for instance, the Fletcher-Reeves algorithm to a general function, we mentioned in Chapter II that it is necessary to restart with the steepest descent direction in order to obtain n-step superlinear convergence. The strategy was justified by the fact that a general function is approximately quadratic near the minimum. So restarting with the steepest descent direction at a point inside the nearly quadratic region Q would ensure a good ultimate convergence. On the other hand we do not know when Q has been reached. Fletcher and Reeves' procedure made such restarts every n or $n+1$ iterations. This has two main disadvantages. First, the times of the restarts are set in advance and are the same for any function. But it may be convenient and worthwhile to restart after fewer than n iterations, especially if n is large. We will see how Powell treated this problem in his algorithm. Secondly, as we mentioned in Chapter II, when restarting with the steepest descent direction, the decrease in the function value is usually less than it would have been along the usual search direction (2.28). Thus progress is, at least temporarily, retarded when a restart is done.

Now Beale's method can be seen as the answer to the following question: if d_t is an arbitrary downhill restarting direction, if f is quadratic and if $d_k (k>t)$ has to be a linear combination of g_k and of the previously calculated $d_t, d_{t+1}, d_{t+2}, \ldots, d_{k-1}$, what linear combination will make $d_t, d_{t+1}, \ldots$ mutually conjugate? The solution is found by using the Gram-Schmidt orthogonalization process. Thus, assume that for $k > t$,

$$d_k = -g_k + \sum_{j=t}^{k-1} a_{jk} d_j \quad , \tag{4.1}$$

where the coefficients a_{jk} are to be chosen to make $d_k^T A d_i = 0$ for $i=t, t+1, \ldots, k-1$. (Recall that A is the constant Hessian of the quadratic f .) This implies that

$$-g_k^T A d_i + \sum_{j=t}^{k-1} a_{jk} d_j^T A d_i = 0 \quad . \tag{4.2}$$

But assuming inductively that the d_j's are mutually conjugate for $t \le j \le k-1$, (4.2) reduces to

$$-g_k^T A d_i + a_{ik} d_i^T A d_i = 0 \quad , \tag{4.3}$$

so

$$a_{ik} = \frac{g_k^T A d_i}{d_i^T A d_i} \quad . \tag{4.4}$$

Now recalling that for f quadratic, $A d_i = \frac{1}{\lambda_i} A(x_{i+1} - x_i)$ $= \frac{1}{\lambda_i}(g_{i+1} - g_i)$, we can rewrite (4.4) as

$$a_{ik} = \frac{g_k^T (g_{i+1} - g_i)}{d_i^T (g_{i+1} - g_i)} \quad , \tag{4.5}$$

where the Hessian does not appear explicitly. Beale noted that (4.5) can be further reduced because of the orthogonality of the successive gradient vectors, i.e.

$$g_k^T g_i = 0 \quad \text{for} \quad k > i \quad , \tag{4.6}$$

Thus $a_{ik} = 0$ for $i = t+1, t+2, \ldots, k-2$ and for $k > t$, (4.1) becomes

$$d_k = -g_k + \beta_k d_{k-1} + \gamma_k d_t \quad , \tag{4.7a}$$

where $$\beta_k = \frac{g_k^T(g_k - g_{k-1})}{d_{k-1}^T(g_k - g_{k-1})} , \tag{4.7b}$$

and $$\gamma_k = \frac{g_k^T(g_k - g_{k-1})}{d_k^T(g_{t+1} - g_t)} . \tag{4.7c}$$

For $k = t + 1$, γ_k is set equal to zero and we obtain the usual CG direction (2.5c).

McGuire and Wolfe [26] implemented an algorithm based on Beale's findings. This algorithm is the same as Fletcher and Reeves', except that they restart with (2.28) as the new search direction after a fixed number of iterations, say t . Otherwise, the d_k's are defined by the three-term recurrence (4.7). The authors noted two undesirable features of the method. First, because $d_{k+1} (k \geq t)$ is conjugate to d_t , we have that all the points $x_{t+1}, x_{t+2}, \ldots$ lie in the manifold

$$L = \{x : d_t^T A(x - x_t) = 0\} , \tag{4.8}$$

since using (2.5a) recursively with $p \geq 2$,

$$x_{t+p} = x_t + \sum_{j=1}^{p-1} \lambda_{t+j} d_{t+j} . \tag{4.9}$$

While in the quadratic case, it is known that the minimum is in L , in the general case it may not be. In that event the sequence $x_{t+1}, x_{t+2}, \ldots$ may converge to a point other than the minimum. Secondly, unlike the Fletcher-Reeves or Polak-Ribière algorithm where all d_k were downhill, it may happen that the search direction (4.7) is not. To overcome that difficulty, McGuire and Wolfe decided to take the negative of those calculated uphill directions where they occur. This strategy did not seem to work and in general, their algorithm showed poor results.

Recently, Powell [34] developed an automatic restart procedure where the frequency of the restarts depends on the objective function. His algorithm puts to advantage the difficulties reported by McGuire and Wolfe. For instance he claimed that a suitable time to restart is when the ratio $g_k^T g_{k-1} / \| g_k \|^2$ is much different from zero. This criterion is equivalent to a restart when the successive g_k's have lost enough orthogonality. Since it is known that the CG algorithm applied to a quadratic function develops orthogonal g_k's, the failure to pass the test would indicate that the algorithm is going through a non-quadratic region. In this case, a restart will not cause any harm to the algorithm. Another test is applied to verify whether the search directions are downhill. If they are not, Powell's algorithm restarts with (2.28). Explicitly, he described his algorithm this way.

Given x_1, set $d_1 = -g_1$ and let $k = t = 1$; then begin the iterations. If $k \geq 2$, test the inequality

$$|g_{k-1}^T g_k| \geq 0.2 \| g_k \|^2 . \tag{4.10}$$

If it holds, set $t = k - 1$ and restart with (2.5c). We also restart by setting $t = k - 1$ if $(k - t) \geq n$ because, in this case, n mutually conjugate search directions will have been used when f is quadratic. For $k \geq 2$, the search direction is defined by (4.7) except that $\gamma_k = 0$ when $k = t + 1$. If $k > t + 1$, check if d_k is sufficiently downhill, i.e. if the inequalities

$$-1.2 \| g_k \|^2 < d_k^T g_k < -0.8 \| g_k \|^2 \tag{4.11}$$

are satisfied. If not, let $t = k - 1$ and redefine d_k by letting $\gamma_k = 0$ in (4.7). Thus d_k is conveniently defined for all values of k. Finally it is assumed that x_{k+1} is found along d_k after an exact line search. The iterations finish if $\| g_{k+1} \|$ or $|f_{k+1}|$ is sufficiently small. Otherwise another iteration is done with k increased by one.

As we will see in the next chapter, Powell's restart algorithm outperforms a standard CG algorithm. The cost of that, Powell noted, is only an extra 2 vectors of storage and is small compared to the advantages. This restart procedure is now used in other algorithms, including the Shanno CG algorithm to be studied in section 4 of this chapter.

4.3 Mixed CG and QN Algorithms

We first point out the relationship existing between the BFGS and the generalized CG algorithm (2.25), as developed by Nazareth [27]. Consider applying any member of Broyden's class of algorithms (2.34)-(2.35) and the generalized CG algorithm to a quadratic function. Assume that the initial approximate inverse Hessian H_1 is the same as H in (2.25) and that ELS are used at each iteration in both cases. Then it is known that both algorithms generate conjugate search directions; as we will denote by d_k^{CG} and d_j^{β}, respectively, the k-th direction generated by the CG algorithm and by the member of the Broyden class given by the value β_k.

If we start them at the same point x_1, obviously we have

$$d_1^{CG} = d_1^{\beta} = -Hg_1 \quad , \tag{4.12}$$

and x_2 will be the same for both algorithms if the line search is exact along $d_1 \equiv d_1^{CG} = d_1^{\beta}$. By the definitions of d_2^{CG} (2.25c), of d_2^{β} (2.35c) and using the ELS assumption, we have that both search directions belong to the subspace spanned by Hg_1 and Hg_2 . Thus the subspace $[d_1, d_2^{CG}, d_2^{\beta}]$ is contained in the plane $[Hg_1, Hg_2]$. But d_1 and d_2^{CG} are conjugate by construction; so are d_1 and d_2^{β} . Therefore, the vectors d_2^{CG} and d_2^{β} must be linearly dependent. As previously, x_3 will be the same in both methods if the line search is unambiguously defined.

The same arguments can be used to show that in general, d_k^{CG} and d_k^{β} are a multiple of each other for all $k \geq 1$ and that all the points x_k are identical. The relation between search directions is even stronger for one specific member of Broyden's class. Indeed, Nazareth proved that the magnitudes of the search directions are also equal in the BFGS case, i.e.

$$d_k^{CG} = d_k^1 \tag{4.13}$$

for all $k \geq 1$, where d_k^1 is generated by the BFGS algorithm (i.e. $\beta_k = 1$). In fact, Buckley [5] established an even more general result, as we will see.

For a general function, a result somewhat similar to (4.13) has been proved by Nazareth. In his proof, he used a theorem due to Powell [35] which we will quote. First, let H_k^{β} denote the matrix updated from a given H_{k-1} using the β-member of the Broyden class (2.35b). The notation H_k^1 then corresponds to a BFGS updated of H_{k-1} , i.e. $H_k^1 = U(x_k, H_{k-1}, 1)$.

Theorem 6. (Powell). If Broyden's class of algorithms is used to minimize a differentiable function $f(x)$, if each iteration calculates λ_j in (2.35a) by an ELS and if this λ_j is unambiguously defined, then the sequence of points x_j and the sequence of matrices H_{j+1} defined by (2.36) are independent of the parameter values $\beta_j (j = 1,2,\ldots)$, provided that no iteration sets β_j equal to $(-1/g^T_{j+1}H_jg_{j+1})$.

So, assume that we are at the j-th iteration of the BFGS algorithm, with

$$d^1_{j+1} = -H^1_{j+1}g_{j+1} . \tag{4.14}$$

Using the hypothesis of an ELS at that iternation, i.e. $g^T_{j+1}s_j = 0$, (4.14) is rewritten

$$d^1_{j+1} = -H^\beta_j g_{j+1} + \frac{y^T_jH^\beta_jg_j}{s^T_jy_j} d^1_j \tag{4.15}$$

since, according to the theorem, the index $\beta \equiv \beta_j$ for the previous updates H_j can have any value (with the one exception).

The similarity between (4.15) and (2.25c) indicates that the BFGS can be seen as a CG algorithm where the metric H, instead of being kept fixed, is updated at each iteration by a member of the Broyden class. This interpretation motivates a new class of mixed CG and QN algorithms. But before presenting a member of that class, we put forward the main features of the whole family.

Consider applying the preconditioned CG algorithm to a quadratic f with the constant inverse Hessian of f as the metric H. It is then seen that we obtain a special case of the original Newton method and given an arbitrary initial approximation x_1, it takes exactly one iteration to reach the minimum. For a general function where the Hessian is not known and depends on x, it then seems reasonable to choose H as an approximation of the true inverse Hessian in the sense that H should satisfy the QN equation. Thus, useful information is brought into the algorithm and the hope is that it will improve the observed convergence compared to that of the standard CG algorithms. At the same time, as a CG algorithm, no storage of a matrix should be required. Hence, assuming the metric variable, a strategy to save storage is needed.

Recently, Buckley ([5], [6]) exploited similar ideas to develop a mixed algorithm which iterates on the CG steps (2.25) with intermittent changes to the metric according to a certain criterion. This corresponds to a need to change the current metric H if the decrease in the function value along the step (2.25c) is not considered satisfactory. Thus the algorithm adjusts itself to the local behaviour of the objective function. As we will see later, the new metric H^* corresponding to a QN update of the current H will be defined by storing only certain vectors. Since the number of those updates will not be fixed in advance, the number of storage locations required to run the algorithm is variable, but of order n. More precisely, the implementation will usually require at least 7 or 8 vectors of length n. This may be more than the 7 n-vectors of the Powell restart algorithm, but it is far less than the order n^2 storage locations of the standard QN algorithms. Also, Buckley [5] proved a theorem which strongly suggests updating by the BFGS formula instead of any other member of the Broyden class. The result will be stated after a brief description of the new algorithm.

Consider applying a modified version of the generalized CG algorithm (2.25) where the metric H is intermittently changed. Let $\bar{x}_k$ and $\bar{d}_k$ be respectively the sequence of points and search directions so generated. Starting with a positive definite H_1 (usually, $H_1 = I$) and an initial point $\bar{x}_1$, we keep this metric until the t-th iteration, say, and we update it to H_{t+1} by the BFGS formula at x_{t+1}. (Thus, in our notation, $H_i \doteq H_1$ for $i = 2,3,\ldots,t$.) Then set

$$\bar{d}_{t+1} = -H_{t+1}g_{t+1} \tag{4.16}$$

The matrix H_{t+1} is recorded by storing the vectors $s_t = x_{t+1} - x_t$ and $H_t y_t = H_t(g_{t+1} - g_t)$. The CG steps are resumed with the metric H_{t+1} and the same procedure is repeated in the next iterations with the current metric periodically updated. For convenience we will denote by H the current metric and by H^* its update.

Now, to stress an important feature of Buckley's algorithm, we consider again the hypothetical example of a general function which is exactly quadratic in a region Q around the minimum. In order to obtain finite termination property once it enters Q, the algorithm would need to take a restart step similar to (4.16) and then to keep this metric throughout the next n iterations (see *Theorem 3*). On the other hand, we do not know when Q is entered and the restarts are done intermittently. So it may happen that more than one restart is accomplished within Q within n steps. Then it is important to know what the effect of such restarts will be on the termination of the algorithm. To know that, Buckley considered a standard CG algorithm and let x_k and d_k be respectively the sequence of points and search directions obtained in that case, from $x_1 = \overline{x}_1$. Comparing this algorithm to his in the quadratic case, he then proved

Theorem 7. (Buckley). Suppose $x_i = \overline{x}_i$ for $i = 1,2,\ldots,k$; assume

$$g_{k-1}^T H g_{k-1} = g_{k-1}^T g_{k-1} \quad , \tag{4.17a}$$

and

$$H g_j = g_j \quad \text{for } j \geq k \; , \tag{4.17b}$$

where $g_j = g(x_j)$ and $x_{k+1}, x_{k+2}, \ldots$ are the points which would be reached from x_k (= $\overline{x}_k$ by hypothesis) by the standard CG algorithm. Then,

$$\overline{d}_{k+1} = d_{k+1} \quad , \tag{4.18a}$$

$$g_k^T H^* g_k = g_k^T g_k \; . \tag{4.18b}$$

and

$$H^* g_j = g_j \quad \text{for } j > k \; . \tag{4.18c}$$

The reader is referred to Buckley [5] for a detailed proof. However we mention that applied inductively, the theorem shows that the two algorithms generate identical sequences of points from the beginning to the end. That means that the change to the metric along the QN step (4.16) does not prevent the mixed algorithm from terminating in n steps or less, in the quadratic case or in case of the quadratic region discussed earlier. Finally, an important point of the paper is that this property is specific to the BFGS update formula. For any other member of the class (2.35b), finite termination will be lost (or at least delayed) if the restart is made before the n-th iteration.

The new algorithm presents some other interesting features. First, as a CG algorithm, it does not need to store any matrix. To do so, Buckley rewrote the Broyden class (2.35b) at x_{j+1} as

$$H_{j+1} = H_j - \eta_j\left[\frac{\beta_j}{b_j}s_j + \frac{1-\beta_j}{a_j}\eta_j\right]^T + s_j\left[\frac{b_j + \beta_j a_j}{b_j^2}s_j - \frac{\beta_j}{b_j}\eta_j\right]^T , \quad (4.19)$$

where $\eta_j = H_j y_j$ and s_j, a_j, b_j are defined as previously. For $\beta_j = 1$, we obtain the BFGS update we are interested in. Thus for each update, $2n + 2$ locations are required to store the two vectors and two scalars. From (4.19), it can then be seen that H_{j+1} does not have to be stored as a matrix provided that H_1 was not stored in matrix form (as for $H_1 = I$) and that the expressions s_t, η_t, a_t and b_t were recorded for $t < j$, where t is the index corresponding to an update of the metric. Since those are done only intermittently, this strategy should save a lot of storage, especially when the number of updates is much less than n. Moreover, note that at x_{t+1}, only the vector $\bar{d}_{t+1} = -H_{t+1}g_{t+1}$ is needed. In the standard QN algorithms, this computation requires, in general, order n^2 operations. In this algorithm, because of the way in which H_{t+1} is stored, d_{t+1} can clearly be evaluated from (4.19) in order n operations. This last feature makes the mixed algorithm comparable to a standard CG algorithm in terms of the number of operations required per iteration.

Since the algorithm has been devised to run under limited storage, one must decide what action should be taken when the storage available has been exhausted. Buckley opted for resetting the current metric to the identity matrix. His main argument concerns the positive definiteness of the H_i's, an important property of the method. Indeed, suppose that, at some stage of the algorithm, the allocated storage limit is reached and we need to record new vectors defining a QN update. Then some old vectors have to be deleted and this has for effect to make the next QN update almost always non-positive definite. If this is not detected, the next search direction may be downhill, an undesirable feature for an algorithm. The simple strategy of resetting H_t to I avoids potential numerical difficulties which may ensue; Buckley noted that his strategy produced good results.

To end this section, we mention some important details related to the implementation of the mixed algorithm. First, it generates a sequence of positive definite matrices H_{j+1}, assuming that $s_{j-1}^T y_j > 0$ as it is for an exact line search. Moreover Buckley showed that all of the search directions d_{j+1} defined by (4.16) or (2.25c) are downhill if the line searches are exact or sufficiently accurate. Finally he devised a test similar to (4.10) in the Powell restart algorithm, which indicates the need to update the current metric H and restart by (4.16). In particular, if

$$\left| \frac{g_{k-1}^T H g_k}{g_k^T H g_k} \right| < 0.2 \quad , \tag{4.20}$$

the algorithm is restarted at x_k. This criterion makes use of the fact that, in the quadratic case, $g_k^T H g_j = 0$ for $j \neq k$. It then corresponds to a sufficient loss of orthogonality of the successive gradients to allow a restart.

Within the class of mixed algorithms, the concept of intermittent updates is particular to Buckley's algorithm. This is unlike another mixed algorithm due to Nazareth, which updates the metric at each iteration.

No extensive comparison of the two algorithms is known to the author. However, the whole class is expected to represent a significant improvement over the standard CG algorithms without performing as well as the best standard QN algorithms. Numerical results for Buckley's algorithm are shown in Chapter V.

4.4 Shanno's CG Algorithm

This CG algorithm was developed by Shanno [40] so as to perform better than the standard CG algorithms under inexact line searches. Moreover, assuming exact line searches (ELS), it reduces to the usual algorithm. Shanno's algorithm was inspired by Perry's work [31], but combines many concepts. It also uses the restart procedure of Powell.

Starting with the standard CG algorithm, Perry noted that in (2.5c), the parameter β_k was chosen so as to make d_k and d_{k+1} conjugate assuming ELS. Since in general, line searches are not exact, Perry relaxed this requirement and rewrote (2.5c) in an equivalent form, but with no ELS assumption. Thus, he obtained

$$d_{k+1} = -\left(I - \frac{d_k y_k^T}{y_k^T d_k}\right) g_{k+1} . \tag{4.21}$$

But the projection matrix multiplying g_{k+1} is not of full rank; so (4.21) is modified to

$$d_{k+1} = -\left(I - \frac{s_k y_k^T}{y_k^T s_k} + \frac{s_k s_k^T}{s_k^T y_k}\right) g_{k+1} \tag{4.22}$$

$$\equiv -Q_{k+1} g_{k+1} . \tag{4.23}$$

Other reasons support his choice of the new term in (4.22). First, the matrix Q_{k+1} satisfies a relation similar to the QN equation, namely

$$Q^T_{k+1} y_k = s_k \quad . \tag{4.24}$$

Also, (4.22) reduces to (4.21) if an ELS is carried out at this iteration since then $s^T_k g_{k+1} = 0$.

Perry's experiments with his algorithm show that it performs only slightly better than the standard CG algorithms under inexact searches. To somewhat justify the relatively poor performance of the new algorithm using (4.23), Shanno noted that the matrix Q_{k+1} is not symmetric or positive definite and hence, (4.23) may not define a downhill direction. Also the similarity with the QN equation is not perfect. So Shanno symmetrized Q_{k+1} by adding an appropriate term. Thus,

$$\hat{Q}_{k+1} = \left(I - \frac{s_k y^T_k}{y^T_k s_k} - \frac{y_k s^T_k}{y^T_k s_k} + \frac{s_k s^T_k}{y^T_k s_k} \right) \quad . \tag{4.25}$$

But the new symmetric matrix does not satisfy (4.24) or the true QN equation, so it is again modified so as to do so. He then obtained

$$Q^*_{k+1} = I - \frac{s_k y^T_k + y_k s^T_k}{y^T_k s_k} + \left(1 + \frac{y^T_k y_k}{s^T_k y_k} \right) \cdot \frac{s_k s^T_k}{y^T_k s_k} \quad . \tag{4.26}$$

This new form of the projection matrix Q_{k+1} has a special relationship with the BFGS update formula, as Shanno pointed out. Indeed the BFGS formula (2.35) can be written as

$$H_{k+1} = H_k - \frac{H_k y_k s^T_k + s_k y^T_k H_k}{s^T_k y_k} + \left(1 + \frac{y^T_k H_k y_k}{s^T_k y_k} \right) \frac{s_k s^T_k}{s^T_k y_k} \quad . \tag{4.27}$$

It is then easily seen that (4.26) is equivalent to (4.27) with $H_k = I$. In fact, a similar dual relationship (4.26)-(4.27) can be exhibited for any member of the Broyden class of updates. Thus a CG algorithm corresponds to a QN algorithm where the approximate inverse Hessian is reset to the identity matrix at each iteration and no storage is used to develop a better approximation to the inverse Hessian. For that reason, this CG algorithm is referred to as a memoryless BFGS algorithm.

Note that the CG algorithm implied by (4.25), namely

$$d_{k+1} = -Q^*_{k+1} g_{k+1} \tag{4.28}$$

reduces again to (4.21) assuming ELS. Also it does not require storage of the matrix Q^*_{k+1} since

$$d_{k+1} = -g_{k+1} - \left[\left(1 + \frac{y_k^T y_k}{s_k^T y_k}\right)\frac{s_k^T g_{k+1}}{s_k^T y_k} - \frac{y_k^T g_{k+1}}{s_k^T y_k}\right] s_k + \frac{s_k^T g_{k+1}}{s_k^T y_k} y_k \ . \tag{4.29}$$

Thus no additional information beyond that required by the standard CG is needed to compute d_{k+1}.

However it can be rightly argued that the identity matrix is not likely to have any relation with the true Hessian. In other words, unlike the QN methods, CG methods (4.29) do not build up a sequence of approximations to the inverse Hessian. In that respect, this may account for the superiority of the QN methods established by McCormick and Ritter [25]. In an attempt to partially make up for this undesirable feature of the CG methods, Shanno devised an algorithm based on the above ideas that also takes advantage of his computational experience with the scaled QN algorithms.

The idea of scaling was originally developed in a series of papers by Oren and Luenberger [28] and Oren and Spedicato [29]. Considering Broyden's class of algorithms, it was proposed in [28] to modify the update formula to

$$H_{k+1} = \left(H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \beta_k a_k w_k w_k^T \right) r_k + \frac{s_k s_k^T}{s_k^T y_k} , \tag{4.30}$$

where r_k is the scaling factor. Basically it was chosen so as to improve the stepwise convergence of the algorithm. For example, using a quadratic f, Luenberger [23] showed how a bad eigenvalue structure of the Hessian can be harmful to the stepwise convergence of the DFP algorithm and how that can be improved by scaling H_{k+1} as in (4.30). Oren and Spedicato [29] determined r_k by minimizing the condition number of $H_k^{-1} H_{k+1}$ and obtained a relation between the parameter β_k in (4.30) and r_k,

$$\beta_k = \frac{b(c - br_k)}{r_k(ac - b^2)} \tag{4.31}$$

where $a = y_k^T H_k y_k$, $b = s_k^T y_k$ as in (4.19) and $c = s_k^T H_k s_k$. We will be particularly interested in the case $\beta_k = 1$, the BFGS case; in this case, we have

$$r_k = \frac{s_k^T y_k}{y_k^T H_k y_k} . \tag{4.32}$$

To the question of how often to scale, Shanno and Phua [42] noted that, according to their experience, it seems to be harmful to scale at every iteration. This is because the scaling factor introduces truncation error in the approximation of the inverse Hessian. However they found that scaling at the initial step was critical, especially for large problems, in order to eliminate the error which results from using the identity matrix to estimate the inverse Hessian. But, as we have seen, that is exactly what the CG algorithm in the form (4.29) is doing.

So it seems natural to always scale the CG algorithm using relation (4.29). That amounts to substituting I for H_k in (4.30); in the BFGS case, $\beta_k = 1$ and r_k is given by (4.32). Thus, Shanno defined a modified CG sequence with $H_k = I$ and

$$d_{k+1} = -H_{k+1}g_{k+1} \quad , \tag{4.33}$$

$$= -r_k g_{k+1} - \left[\left(1 + r_k \cdot \frac{y_k^T y_k}{s_k^T y_k}\right) \cdot \frac{s_k^T g_{k+1}}{s_k^T y_k} - r_k \cdot \frac{s_k^T g_{k+1}}{s_k^T y_k}\right] s_k$$

$$+ r_k \cdot \frac{s_k^T g_{k+1}}{s_k^T y_k} y_k \quad .$$

After reduction, he obtained

$$d_{k+1} = -\frac{s_k^T y_k}{y_k^T y_k} g_{k+1} - \left(2\frac{s_k^T g_{k+1}}{s_k^T y_k} - \frac{y_k^T g_{k+1}}{y_k^T y_k}\right) s_k + \frac{s_k^T g_{k+1}}{y_k^T y_k} y_k \quad . \tag{4.34}$$

This corresponds to the scaling by r_k of the memoryless BFGS (4.26) and (4.28). Unfortunately, Shanno found that this CG algorithm did not produce as good results as the one just defined by (4.28) but scaled in a way suggested by Fletcher [15], i.e.

$$\overline{d}_{k+1} = \frac{2(f_{k+1} - f_k)}{g_{k+1}^T d_{k+1}} d_{k+1} \quad . \tag{4.35}$$

The reason might be that applying (4.34) to a general function and not assuming ELS, this sequence of d_k does not include a step similar to a restart.

Considering the good performance of the Powell restart procedure, Shanno then examined the three-term recurrence (4.7) on which it is based. This was done in particular view of the dual CG-QN relationship studied in (4.26)-(4.27). Thus (4.7) is rewritten as

$$d_{k+1} = -\left(I - \frac{d_k y_k^T}{d_k^T y_k} - \frac{d_t y_k^T}{d_t^T y_t}\right) g_{k+1} \quad , \tag{4.36}$$

$$= -P_{k+1} g_{k+1} , \tag{4.37}$$

where t is the index of the last restart and the matrix P_{k+1} uses information from two prior points x_k and x_t . In Beale's method, we recall that the information gathered at x_t was critical and must be retained. And, at the same time, the storage requirement for the CG must stay within order n locations. So Shanno defined for $k > t$

$$\hat{H}_t = \left(I - \frac{s_t y_t^T + y_t s_t^T}{s_t^T y_t} + \frac{y_t^T y_t}{s_t^T y_t} \cdot \frac{s_t s_t^T}{s_t^T y_t}\right) r_t + \frac{s_t s_t^T}{s_t^T y_t} \quad . \tag{4.38}$$

The information at x_t is retained through this matrix which corresponds to a scaled BFGS update of $H_t = I$.

Now the approximate inverse Hessian at the point x_{k+1} is defined as the BFGS update of $\hat{H}_t$. Thus,

$$\hat{H}_{k+1} = \hat{H}_t - \frac{s_k y_k^T \hat{H}_t + \hat{H}_t y_k s_k^T}{s_k^T y_k} + \left(1 + \frac{y_k^T \hat{H}_t y_k}{s_k^T y_k}\right) \cdot \frac{s_k s_k^T}{s_k^T y_k} \quad . \tag{4.39}$$

Then the search direction at x_{k+1} is found by setting

$$d_{k+1} = -\hat{H}_{k+1}g_{k+1}$$

$$= -\hat{H}_t g_{k+1} - \frac{s_k^T g_{k+1}}{s_k^T y_k} \cdot \hat{H}_t y_k - \left[\left(1 + \frac{y_k^T \hat{H}_t y_k}{s_k^T y_k}\right) \cdot \frac{s_k^T g_{k+1}}{s_k^T y_k} - \frac{y_k^T \hat{H}_t g_{k+1}}{s_k^T y_k}\right] s_k \tag{4.40a}$$

where the vectors $\hat{H}_t g_{k+1}$ and $\hat{H}_t y_k$ are given by

$$\hat{H}_t g_{k+1} = \frac{s_t^T y_t}{y_t^T y_t} g_{k+1} - \frac{s_t^T g_{k+1}}{y_t^T y_t} y_t + \left(2\frac{s_t^T g_{k+1}}{s_t^T y_t} - \frac{y_t^T g_{k+1}}{y_t^T y_t}\right) s_t \quad , \tag{4.40b}$$

$$\hat{H}_t y_k = \frac{s_t^T y_t}{y_t^T y_t} y_k - \frac{s_t^T y_k}{y_t^T y_t} y_t + \left(2\frac{s_t^T y_k}{s_t^T y_t} - \frac{y_t^T y_k}{y_t^T y_t}\right) s_t \quad . \tag{4.40c}$$

This definition of the d_k amounts to considering a projection matrix similar to (4.21), but based on the three-term recurrence. This matrix, in view of the dual CG-QN relationship, is scaled in the way initiated by Oren and Luenberger and then updated.

Note that no matrix needs to be stored to compute d_{k+1}. Indeed looking at (4.40), the vectors (4.40b) and (4.40c) fully define the search direction (4.40a). As to the number of storage locations needed, compared to Beale's method, Shanno's CG algorithm requires no extra vectors of storage. The vectors needed are x_{k+1}, x_k, g_{k+1}, g_k, d_k, d_t and y_t. Shanno noted that at the time of an update, the information in x_k is no longer required, so that (4.40c) may be stored in x_k. Once (4.40c) and scalars such as $s_k^T y_k$ and $y_k^T \hat{H}_t y_k$ have been computed, g_k is no longer required, so (4.40b) can be explicitly computed and stored in g_k.

Two other properties of the algorithm are worth mentioning. First, as the updates (4.38) and (4.39) use QN update formulae, the condition $s_k^T y_k > 0$ is sufficient to ensure that the search directions (4.40) define downhill directions. Second, for f quadratic with ELS, the algorithm using (4.40) reduces to Beale's method defined by (4.35).

To end this section, we indicate how Shanno implemented his algorithm. Given x_1, set $d_1 = -g_1/\|g_1\|$. For $k \geq 2$, the algorithm is restarted whenever $|g_{k-1}^T g_k| \geq 0.2\,\|g_k\|^2$ or every n iterations, as in Powell's algorithm. The restarting directions are defined by (4.34), the memoryless BFGS update scaled by (4.32). The other steps use (4.40) and are also scaled by

$$\hat{d}_{k+1} = \frac{2(f_{k+1} - f_k)}{g_{k+1}^T d_{k+1}} d_{k+1} , \qquad (4.41)$$

following Fletcher's suggestion.

To justify the double scaling, Shanno appeals to his computational experience. His experiments showed, as we mentioned earlier, that an algorithm scaled as in (4.41) sometimes performs better than the one using the scaled memoryless BFGS. His algorithm, as it uses the Powell restart procedure, selects automatically the appropriate scaling at any time and takes advantage of both ways of scaling.

CHAPTER V

NUMERICAL RESULTS

5.1 Source of Numerical Results

We will be considering seven routines developed to solve the problem (1.1). First, for purpose of historical reference, we have included implementations of both one standard QN (BFGS) and one standard CG (Fletcher-Reeves) algorithm. The QN routine, which we dub QNS, has been coded by Shanno while the CG routine, VA08, is from the Harwell library. Also, one modified CG routine, Powell's restart VA14, originates from the same library. Three other routines examined are Toint's and Shanno's SE QN algorithms, known, respectively, as PSB and SBFGS and Shanno's CG algorithm for inexact line searches CGILS. Shanno ([40] and [41]) implemented all three. Finally the mixed algorithm CGQNA has been coded by Buckley ([6]).

Note that for the figures given, all the algorithms except those coded by Shanno terminate when the 2-norm of the current gradient is less than 10^{-6}. Shanno used the infinity norm instead, with an accuracy of 10^{-5}. Our experience indicates that the results would not vary significantly from one termination criterion to the other.

As to the test functions, they are quite common and we refer the reader to [40], [42] and [44] for a detailed description. Nevertheless, we mention the meaning of the abbreviated function names we will use:

EROSEN	is the famous Rosenbrock function extended to more than 2 dimensions in an obvious way,
CROSEN	is a non-convex version of EROSEN;
NONDIA	stands for a non-diagonal variant of the same EROSEN;

TRIDIA is a tridiagonal quadratic function;

OREN stands for the Oren power function;

MANCINO stands for the Maneino function.

This choice of algorithms and test functions was made on two main grounds. First, two of the functions are well-known sparse ones (TRIDIA, CROSEN). Second, all the function codes and all the algorithms except the two SE QN ones were accessible for us from a private library. Thus, we could obtain most of the results shown in the tables by running the five algorithms on the University computer. Note that some figures are missing for PSB and SBFGS; in those cases, neither the routines were available in the library nor the figures published to our knowledge.

5.2 Comparison of the Algorithms

The comparison of minimization routines is a rather complex subject, and the art is yet to be fully developed. There are many approaches to the subject and the one we are going to follow, although not the most sophisticated, is quite standard and convenient for our purposes. It mainly tests the efficiency of the routines on a selection of test functions. The measure of efficiency consists in Table I of the number of iterations (ITER) and the number of function evaluations (IFN) needed by the routine to reach an acceptable approximation to the minimum of a test function. If this routine fails to do so, we indicated it by the letter F . Moreover we included in Table II the time (in seconds) spent in evaluating the objective function (FSEC) and the total CPU time (MSEC), whenever available. In those cases, we managed to obtain MSEC figures which do not include the print times.

Within this framework, an extensive comparison study would require a broad variety of test problems arising from different fields with, for instance, various degrees of non-linearity. Also it is usual to test the robustness of routines by considering the results obtained by the routines when started at different points with the same test function. A number of other factors could have been considered and their effect on the performance of the routine studied. However, this thesis is more limited in scope and generally, for a given test function, all the algorithms are started at a common point. Moreover, the specific factor relevant to us is the high dimensional sparse case of problem (1.1). This guided our choice of problems, which consisted of test functions of moderate dimension, some of them being sparse. In that respect, we have already mentioned the lack of results for medium and large size problems. Indeed, looking at the tables, the largest value of n is 50. Therefore, since we are particularly interested in higher dimensions, we will have to extrapolate on the basis of the available figures. However, we will support these extrapolations with known theoretical and numerical results. But first, we discuss our selection of algorithms and the procedure for comparing them.

The seven algorithms can be divided into four classes. The standard CG routine VA08 forms class 1 while class 2 includes the three modified CG routines. The standard QN and the two SE QN routines constitute respectively classes 3 and 4. We will compare the performance of those classes and in particular, we will stress the comparison of the SE QN class and the modified CG class. That should give us a picture of the progress made in the design of algorithms to solve (1.1), when n is large and/or the Hessian is sparse. The analysis will consider, besides the figures shown in Tables I and II, other details relevant to the implementation and use of an algorithm or of a specific class of algorithms.

In general, CG algorithms are the simplest to use and also the least demanding in terms of internal work. In particular, the housekeeping for linear algebra is at its minimum; the number of operations per iteration is of order n only. This is one of their advantages as compared to the QN algorithms, where order n^2 operations per iteration are usually required. On the other hand, it is usually observed that the standard CG algorithms take more function evaluations than their QN counterparts. This may become a critical factor when the functions are expensive to evaluate.

Comparing classes 1, 2 and 3, it is seen from Table 1 that on most of the test functions, the modified CG and the standard QN routines outperform the standard CG routine. This is as expected. It is known indeed [25] that the QN algorithms are superior to the CG algorithms, in general. Also, Powell [34] put forward evidence why his restart algorithm performs better than a standard CG algorithm. Both analysis are confirmed by our figures. Thus, ITER and IFN are almost steadily less for classes 2 and 3 than for class 1. On the other hand, looking at the time columns in Table II, the comparison clearly favors the CG routines in general, especially when the function evaluation time FSEC is a small fraction of the total time MSEC. Otherwise, as for the MANCINO function, the significantly smaller number of function evaluations of QNS becomes important. Then MSEC drastically changes the picture in favor of class 3. Finally, note that CGILS performs consistently better than the other members of class 2.

A common feature of the routines of classes 1, 2 and 3 is that they do not require any specific task from the user besides the subprogram to evaluate the objective function and its gradient. More is needed in the case of the SE QN class. To take advantage of the sparseness of the Hessian B, it was assumed that the sparsity pattern was known. Thus, before running a SE QN algorithm, the user must provide the relevant information defining the zero pattern of B . This may be a time-consuming job and is not necessarily easy. However, for large n , that effort should be compensated for by the improvement in performance over a standard QN algorithm. In fact, for very large n , solving (1.1) may be impossible without exploiting the sparseness of the Hessian.

Now, examining the QN classes 3 and 4, there is a clear indication that both SE QN algorithms are better than the standard QN routine. In particular, the number of function evaluations for PSB and SBFGS are almost always less than the corresponding figures for QNS. Moreover, the internal work needed by the SE QN algorithms is usually of order n^2 operations per iteration, depending on the sparsity pattern, i.e. it is the same as that required by the standard QN algorithms. Therefore, in this case, the statistics FSEC and MSEC should still be expected to favour class 4. Unfortunately, the time figures for the SE QN class are not available to confirm that. Furthermore, the order n^2 storage locations requirement of the standard QN algorithms makes them not practical or not usable for large n . Then, the alternative must be between the SE QN class and the modified CG class of algorithms.

That leads us to the comparison of class 2 versus class 4. Specifically, we consider CGILS and SBFGS which may be considered as representatives of each of their classes. Again it is apparent that in general, the SE QN routine requires fewer function evaluations than the modified CG routine. Despite that, for problems of moderate dimensions or reasonably simple functions, the total time MSEC should slightly favor the CG representative mainly because it generally requires less internal work than a SE QN algorithm. On the other hand, when n is large or FSEC dominates within MSEC, the modified CG class should not sustain the comparison. The MANCINO function is an example; in this case, SBFGS is expected to do at least as well as the standard QNS, which in turn outperforms class 2 as far as the time factor is concerned. Again we do not have the FSEC and MSEC figures, or the IFN column to support our argument.

In summary, it is clear that for large n , the ultimate choice is between the modified CG and the SE QN classes. And although the comparison of the two classes favors the latter, the modified CG algorithms can be competitive in some circumstances, particularly when the objective function is not too expensive to evaluate.

TABLE I

	CLASS 1		CLASS 2						CLASS 3		CLASS 4			
	VA08		VA14		CGQNA		CGILS		QNS		PSB		SBFGS	
NONDIA	ITER	IFN	ITER	IFN	ITER	IFN	ITER	IFN	ITER	IFN	ITER	IFN	ITER	IFN
n = 10	33	78	21	63	30	95	23	55	33	46	31	38	31	42
n = 20	32	75	22	70	25	82	23	53	32	43	36	46	33	42
n = 30	30	72	F	43	29	93	17	40	30	41	74	76	37	46
CROSEN														
n = 10	48	100	50	96	50	100	36	75	37	39	22	31	20	27
n = 25	49	100	54	100	48	100	49	100	52	56	25	39	23	36
EROSEN														
n = 5	F	100	F	20	F	79	F	100	35	71				
n = 10	36	95	23	68	24	76	21	54	30	43				
TRIDIA														
n = 10	39	79	33	65	34	66	25	51	20	21	14	17	13	16
n = 20	50	100	55	99	56	99	42	85	32	33	23	26	17	21
n = 30	49	100	58	100	56	100	50	100	40	41	32	40	26	33
OREN														
n = 20	23	68	20	54	34	91	15	31	F	100				
n = 50	38	97	31	76	38	90	34	69	F	100				
MANCINO														
n = 10	6	15	9	13	7	16	5	11	5	7				
n = 20	7	22	8	18	10	22	6	13	7	8				
n = 30	8	25	13	21	12	24	9	18	7	8				

TABLE II

	CLASS 1		CLASS 2						CLASS 3	
	VA08		VA14		CGQNA		CGILS		QNS	
NONDIA	MSEC	FSEC	MSEC	FSEC	MSEC	FSEC	MSEC	FSEC	FSEC	MSEC
n = 10	.28	.09	.35	.07	.67	.11	.27	.06	.60	.05
n = 20	.43	.17	.65	.15	1.17	.20	.47	.11	1.65	.09
n = 30	.54	.23	.57	.14	1.74	.30	.49	.13	3.06	.13
CROSEN										
n = 10	.40	.15	.62	.14	.97	.16	.46	.11	.64	.05
n = 25	.79	.36	1.36	.37	2.01	.37	1.41	.36	3.89	.20
EROSEN										
n = 5	.18	.06	.08	.01	.23	.06	.24	.07	.36	.04
n = 10	.30	.10	.36	.08	.51	.08	.26	.06	.61	.05
TRIDIA										
n = 10	.29	.09	.42	.07	.60	.08	.31	.06	.33	.02
n = 20	.59	.22	1.08	.21	1.50	.20	.95	.18	1.59	.07
n = 30	.81	.32	1.50	.32	2.26	.33	1.61	.32	4.01	.13
OREN										
n = 20	.32	.08	.50	.06	.99	.09	.26	.04	4.91	.12
n = 50	.88	.26	1.15	.21	1.30	.23	26.22	.27	1.06	.16
MANCINO										
n = 10	5.00	4.96	4.41	4.33	5.38	5.24	3.73	3.69	2.34	2.27
n = 20	56.8	56.7	46.39	46.22	55.75	55.43	32.92	32.82	20.49	20.19
n = 30	218.5	218.4	175.8	175.5	205.2	204.5	152.8	152.7	68.14	67.54

CHAPTER VI

CONCLUSION

In this thesis, we have surveyed the two main types of gradient methods for the solution of (1.1): the CG and the QN methods. Although they both solve the same problem, there was a clear division in the sort of situations in which they were used. Basically, the QN algorithms, despite their known effectiveness, were preferred only when n was small enough so that their order n^2 storage locations requirement could be met by the user. Otherwise, the standard CG algorithms were used in spite of their poorer performance.

Recently, the gap between the two standard methods has been somewhat filled by the development of the modified CG and the Sparsity Exploiting QN algorithms. Both classes require only order n storage locations in order to be implemented. Moreover, as we have seen in Chapters III and IV, the first class improves the convergence of the standard CG algorithms and the second class performs at least as well as the standard QN algorithms.

Now, when n is large so that both modified CG and SE QN algorithms can be used, a user may have to choose between the two classes. Then, based on our discussion of Chapter V, the modified CG class should be preferred to the SE QN class if the problem is of medium size and the functions not too expensive to evaluate. Their performance might be inferior, but their simplicity makes them preferable, provided that the function evaluation cost is not excessive. On the other hand, if the functions are expensive to evaluate and computer time is a prime factor, there is no doubt that the SE QN algorithm should be favored. This is particularly true when n becomes very large or when the routine has to be run often. Then, the effort to set up the sparsity code is expected to be returned.

These brief recommendations obviously do not cover all the possibilities. In a real-life situation, a user will have to deal with other factors. Nevertheless, a compromise will likely have to take account of the cost factor. This thesis should provide users with valuable information upon which they may base their choice.

REFERENCES

[1] BEALE, E.M.L., "A derivation of conjugate gradients" in Num. Methods for Nonlinear Optimization, F.A. Lootsma ed. (Academic Press, 1972.)

[2] BROYDEN, C.G., "A class of methods for solving nonlinear simultaneous equations", Math. of Computation 19(1965).

[3] BROYDEN, C.G., "Quasi Newton methods" in Num. Methods for Nonlinear Optimization, F.A. Lootsma ed. (1972).

[4] BROYDEN, C.G., "The convergence of a class of double-rank minimization algorithms", J. Inst. Math. Appl. 6(1971).

[5] BUCKLEY, A.G., "Extending the relationship between the conjugate gradient and BFGS algorithm", Math. Prog. 15(1978).

[6] BUCKLEY, A.G., "A combined Conjugate Gradient Quasi Newton minimization algorithm", Math. Prog. 15(1978).

[7] CROWDER, H. and WOLFE, P., "Linear convergence of the conjugate gradient method", IBM Journal of Research and Development 16(1972).

[8] DAVIDON, W.C., "Variable metric method for minimization", A.E.C. Research and Development Report, ANL-5990(1959).

[9] DENNIS, J.E., "On some methods based on Broyden's secant approximation to the Hessian" in Num. Methods for Nonlinear Opt., F.A. Lootsma ed. (1972).

[10] DENNIS, J.E. and MORE, J.J., "Quasi Newton methods, motivation and theory", SIAM Review 19(1979).

[11] DENNIS, J.E. and SCHNABEL, R.B., "Least change secant updates for Quasi Newton methods", SIAM Review 21(1979).

[12] DIXON, L.C.W., "Variable metric algorithms: Necessary and sufficient conditions for identical behaviour on non-quadratic functions", NOC Tech. Rpt. 26(1971).

[13] DIXON, L.C.W., "The choice of step length, a crucial factor in the performance of variable metric algorithms" in Num. Methods for Nonlinear Opt., F.A. Lootsma ed. (1972).

[14] FLETCHER, R., "Conjugate direction methods" in Num. Methods for Unconstrained minimization, W. Murray ed. (1972).

[15] FLETCHER, R., "A Fortran subroutine for minimization by the method of Conjugate Gradients", Rpt. R-7073, A.E.R.E. Harwell (1972).

[16] FLETCHER, R., "A new approach to variable metric algorithms", Computer Journal, 13(1970).

[17] FLETCHER, R. and REEVES, C.M., "Function minimization by conjugate gradients", Computer Journal, 7(1964).

[18] FLETCHER, R. and POWELL, M.J.D., "A rapidly convergent descent method for minimization", Computer Journal, 7(1963).

[19] GOLDFARB, D., "A family of variable metric methods derived by variational means", Math. of Computation, 29(1974).

[20] GREENSTADT, J., "Variations on variable metric methods", Math. of Computation, 24(1970).

[21] HESTENES, M.R. and STIEFEL, E., "Methods of conjugate gradients for solving linear systems", J. Res. Nat. Bureau Standard, 49(1959).

[22] HUANG, H.U., "Unified approach to quadratically convergent algorithms for function minimization, JOTA, 5(1970).

[23] LUENBERGER, D., "Introduction to Linear and Nonlinear programming", Addison-Wesley (1973).

[24] MARWIL, S.E., "Exploiting sparsity in Newton-like methods", Ph.D. Thesis, Cornell Univ., N.Y., (1978).

[25] MCCORMICK, G. and RITTER, K., "Methods of conjugate directions versus Quasi Newton methods", Math. Prog., 3(1972).

[26] MCGUIRE, M.F. and WOLFE, P., "Evaluating a restart procedure for conjugate gradients", Rpt. RC-4382, IBM Research Center, Yorktown Heights, 1973.

[27] NAZARETH, L., "A relationship between the BFGS and conjugate gradient algorithms", Math. Prog., 16(1979).

[28] OREN, S.S. and LUENBERGER, D., "Self-Scaling variable metric algorithm, part I", Management Science, 20(1974).

[29] OREN, S.S. and SPEDICATO, E., "Optimal conditioning or Self-Scaling variable metric algorithms", Math. Prog., 10(1976).

[30] ORTEGA, J.M. and RHEINBOLDT, W.C., "Iterative solutions of non-linear equations in several variables", (Academic Press, 1970).

[31] PERRY, A., "A modified Conjugate Gradient algorithm", Discussion paper No. 229, Center for Mathematical Studies in Economics and Management Science, Northwestern University, 1976.

[32] POLAK, E. and RIBIERE, G., "Note sur la convergence de méthodes des directions conjuguées, "Univ. of California, Berkeley, Dept. of Electrical Engineering and Comp. Sciences, working paper. (1969)

[33] POWELL, M.J.D., "A new algorithm for unconstrained optimization" in Nonlinear programming, Rosen, J.B., Managarsarian, L. and Ritter, K., eds. (Academic Press, 1970).

[34] POWELL, M.J.D., "Restart procedure for the conjugate gradient methods", Math. Prog., 12(1977).

[35] POWELL, M.J.D., "Unconstrained minimization and extensions for constraints", in Math. Prog. in Theory and Practice, Hammer, P.L. and Zoutendijk, G., eds. (North-Holland, 1974).

[36] POWELL, M.J.D., "Quadratic termination properties of a class of double-rank minimization algorithms", Rpt. T.P. 471, A.E.R.E., Harwell. (1972).

[37] POWELL, M.J.D., "Quasi-Newton formulae for sparse second derivative matrices", Internal Rpt. DAMTP (Univ. of Cambridge, 1979).

[38] SCHUBERT, L.K., "Modification of a Quasi-Newton method for nonlinear equations with a sparse Jacobian", Math. of Computation, 24(1970).

[39] SHANNO, D.F., "Conditioning of a Quasi-Newton method for function minimization", Math. of Computation, 24(1970).

[40] SHANNO, D.F., "Conjugate gradient methods with inexact searches", MIS (Univ. of Arizona, Tucson), Tech. Rpt. No. 22 (1977).

[41] SHANNO, D.F., "On variable metric methods for sparse Hessians", parts 1 and 2, MIS Tech. Rpt. No. 26(1978).

[42] SHANNO, D.F. and PHUA, K.H., "Numerical comparison of several variable metric algorithms", JOTA 25(1978).

[43] TOINT, Ph.L., "On sparse and symmetric updating subject to a linear equation", Math. of Computation, 31(1977).

[44] TOINT, Ph.L., "Some numerical results using a sparse matrix updating formula in unconstrained minimization", Math. of Computation, 32(1978).

[45] TOINT, Ph.L., "A note about Sparsity Exploiting Quasi-Newton Updates", Facultés Universitaires de Namur (Belgium) Tech. Rpt. 79/5 (1979).