NONLINEAR PROGRAMMING FOR LARGE, SPARSE SYSTEMS

BY

B. A. MURTAGH and M. A. SAUNDERS

TECHNICAL REPORT SOL 76-15
AUGUST 1976

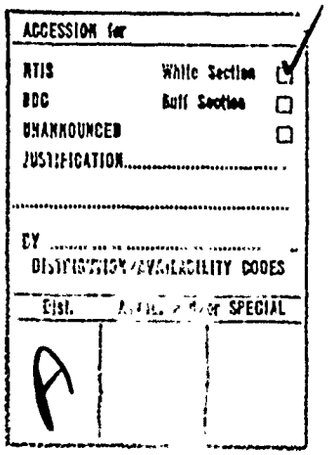
Systems Optimization Laboratory

Department of Operations Research

Stanford University

Stanford
California
94305

NONLINEAR PROGRAMMING FOR LARGE, SPARSE SYSTEMS



by

B.A. Murtagh and M.A. Saunders

TECHNICAL REPORT SOL 76-15

August 1976

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH

Stanford University
Stanford, California

Research and reproduction of this report were partially supported by the Office of Naval Research under Contract N00014-75-C-0267; the National Science Foundation Grants MCS71-03341 A04, DCR75-04544; U.S. Energy Research and Development Administration Contract E(04-3)-326 PA #18.

NONLINEAR PROGRAMMING FOR LARGE, SPARSE SYSTEMS

B. A. Murtagh and M. A. Saunders

ABSTRACT

An algorithm for solving large-scale nonlinear programs with linear constraints is presented. The method combines efficient sparse-matrix techniques as in the revised simplex method with stable variable-metric methods for handling the nonlinearities. A general-purpose production code (MINOS) is described, along with computational experience on a wide variety of problems.

NONLINEAR PROGRAMMING FOR LARGE, SPARSE SYSTEMS*

B. A. Murtagh and M. A. Saunders

1. Introduction

This paper describes our efforts to develop a nonlinear programming algorithm for problems characterized by a large sparse set of linear constraints and a significant degree of nonlinearity in the objective function. It has been our experience that many linear programming problems are inordinately large because they are attempting to approximate, by piecewise linearization, what is essentially a nonlinear problem. It also appears that many real-life problems are such that only a small percentage of the variables are involved nonlinearly in the objective function. Thus we are led to consider problems which have the following canonical form:

$$\text{minimize} \quad F(\underset{\sim}{x}) = f(\underset{\sim}{x}^N) + \underset{\sim}{c}^T\underset{\sim}{x} \tag{1}$$

$$\text{subject to} \quad A\underset{\sim}{x} = \underset{\sim}{b} \tag{2}$$

$$\underset{\sim}{\ell} \leq \underset{\sim}{x} \leq \underset{\sim}{u} \tag{3}$$

where A is $m \times n$, $m \leq n$. We partition $\underset{\sim}{x}$ into a linear portion $\underset{\sim}{x}^L$ and a nonlinear portion $\underset{\sim}{x}^N$:

$$\underset{\sim}{x} = \begin{bmatrix} \underset{\sim}{x}^N \\ \underset{\sim}{x}^L \end{bmatrix} .$$

*An earlier version of this paper was presented at the 8th Int. Symp. Math. Prog., Stanford, California, August 1973.

The components of $\underset{\sim}{x}^N$ will normally be called the nonlinear variables. Note that A and $\underset{\sim}{c}$ operate on all variables $\underset{\sim}{x}$. In some cases the part of $\underset{\sim}{c}^T\underset{\sim}{x}$ involving $\underset{\sim}{x}^N$ may be incorporated into $f(\underset{\sim}{x}^N)$; in other cases $\underset{\sim}{c}$ may be zero. We assume that the function $f(\underset{\sim}{x}^N)$ is continuously differentiable in the feasible region, with gradient

$$\nabla f(\underset{\sim}{x}^N) = \underset{\sim}{g}(\underset{\sim}{x}^N).$$

The research work reported here was stimulated by some of the deficiencies in the algorithm of Murtagh and Sargent [44], [50], especially when applied to large-scale systems. The resulting algorithm is related to the reduced-gradient method of Wolfe [56] and the variable-reduction method of McCormick [41], [42]. It also draws much from the unconstrained and linearly-constrained optimization methods of Gill and Murray [21], [22], [25].

In essence the algorithm is an extension of the revised simplex method (Dantzig [12]). To use some of the associated terminology, it might be described as an extension which permits more than m variables to be basic. Because of the close ties with linear programming (LP) we have been able to incorporate into our implementation many of the recent advances in LP technology. The result is a computer program which has many of the capabilities of an efficient LP code and is also able to deal with nonlinear terms with the power of a quasi-Newton procedure.

1.1 Notation and Terminology

Partitioning $\underset{\sim}{x}$ and $F(\underset{\sim}{x})$ into linear and nonlinear terms is of considerable practical importance; for descriptive purposes however, it is convenient to denote $F(\underset{\sim}{x})$ and $\nabla F(\underset{\sim}{x}) = \underset{\sim}{g}(\underset{\sim}{x}) + \underset{\sim}{c}$ simply by $f(\underset{\sim}{x})$ and $\underset{\sim}{g}(\underset{\sim}{x})$.

With a few conventional exceptions, we use upper-case letters for matrices, lower-case for vectors and Greek lower-case for scalars. The quantity $\epsilon > 0$ represents the precision of floating-point arithmetic.

The terms "variable-metric" and "quasi-Newton" will be used synonymously, as will the adjectives "reduced" and "projected."

2. Basis of the Method

2.1. Variable-metric projection

Before turning to the canonical form, consider the problem

$$\text{minimize} \quad f(\underset{\sim}{x}) = f(x_1, \ \ldots \ , x_n) . \tag{4}$$

$$\text{subject to} \quad A\underset{\sim}{x} \leq \underset{\sim}{b} \tag{5}$$

We will assume that $f(\underset{\sim}{x})$ can be expanded in a Taylor's series with remainder of second order:

$$f(\underset{\sim}{x}_{k+1}) = f(\underset{\sim}{x}_k) + g_k^T \Delta\underset{\sim}{x}_k + \frac{1}{2} \Delta\underset{\sim}{x}_k^T G(\underset{\sim}{x}_k + \gamma\Delta\underset{\sim}{x}_k)\Delta\underset{\sim}{x}_k \tag{6}$$

where

$$\Delta\underset{\sim}{x}_k = \underset{\sim}{x}_{k+1} - \underset{\sim}{x}_k$$

$$g_k = \nabla f(\underset{\sim}{x}_k)$$

and $G(\underset{\sim}{x}_k + \gamma\Delta\underset{\sim}{x}_k)$ is the Hessian matrix of second partial derivatives evaluated at some point between $\underset{\sim}{x}_k$ and $\underset{\sim}{x}_{k+1}$. Note that G is a constant matrix if $f(\underset{\sim}{x})$ is a quadratic function.

Suppose that the current point, $\underset{\sim}{x}_k$, is on the boundary of m constraints $(m < n)$. Denoting the corresponding m rows of A by the submatrix B we have:

$$B\underset{\sim}{x}_k = \underset{\sim}{b}_m \tag{7}$$

Variable-metric projection methods (Goldfarb [30], Murtagh and Sargent [44]) are based on two properties required of the step $\Delta\underset{\sim}{x}_k$:

Property 1.

$$g_k + G(\underset{\sim}{x}_k)\,\Delta\underset{\sim}{x}_k = B^T\underset{\sim}{\lambda} \tag{8}$$

i.e., the step Δx_k is to a stationary point on the surface of the m active constraints. The gradient of $f(x)$ at x_{k+1} (given by the left hand side of equation (8) if x_k is sufficiently close to the stationary point for a quadratic approximation to be valid) is orthogonal to the surface and thus denoted by a linear combination of the constraint normals (given by the right hand side of equation (8)).

Property 2.

$$B\Delta x_k = 0 \tag{9}$$

i.e. the step remains on the surface given by the intersection of the m active constraints.

The implementation put forward by Murtagh and Sargent [44] used these two properties to produce a step given by

$$\Delta x_k = -\alpha_k S_k (g_k - B^T \lambda_k) \tag{10}$$

where λ_k (an estimate of the Lagrange multipliers for B) is obtained by substituting equation (9) into (8):

$$\lambda_k = (BS_kB^T)^{-1} BS_k g_k , \tag{11}$$

α_k is a step-size parameter used to adjust the length of the step Δx_k, and S_k is a variable-metric approximation to $G^{-1}(x_k)$. A rank-1 updating procedure is used to modify S_k each iteration. This allows the matrix $(BS_kB^T)^{-1}$ to be updated by a rank-1 correction also. Note however that no advantage is taken of either B or G being sparse, since the matrices G^{-1} and $(BG^{-1}B^T)^{-1}$ are in general dense.

The procedure works well in many cases (see [50]), but storage limitations prevent application to large problems, quite regardless of numerical accuracy in the updating procedures. The motivation for the present work, therefore, is to use Property 1 and Property 2 in a more efficient manner, particularly for the case of large sparse systems with relatively few nonlinear variables.

2.2. Extension to Large Sparse Systems

We return now to the canonical form of equations (1)-(3). The key to success with the simplex method lay in adopting such a canonical form and working with so-called basic feasible solutions, which are characterized by having n-m "nonbasic" variables equal to their upper or lower bound. With nonlinear problems we cannot expect an optimal solution to be of this kind. As a simple generalization we introduce the notion of "superbasic" variables and partition the set of general constraints (2) as follows:

$$A\underset{\sim}{x} = \overset{\text{m}}{B_1} \quad \overset{\text{s}}{B_2} \quad \overset{\text{n-m-s}}{B_3} \begin{bmatrix} \underset{\sim}{x}_1 \\ \underset{\sim}{x}_2 \\ \underset{\sim}{x}_3 \end{bmatrix} = \underset{\sim}{b} \qquad (12)$$

basics; superbasics; nonbasics

The matrix B_1 is square and nonsingular and corresponds to the usual basis matrix of the simplex method; B_2 is $m \times s$ with $0 \leq s \leq n-m$, and the associated variables x_2 are called the superbasics as shown. Both basics and superbasics are free to vary between their bounds.

The motivation for this partitioning is provided by the following:

Theorem 1. If a nonlinear program has an optimal solution and if it involves t variables nonlinearly, an optimal solution exists in which the number of superbasic variables s satisfies $s \leq t$.

Proof (due to A. Jain). Let the nonlinear variables be fixed at their optimal values. The remaining problem is a linear program for which a basic solution exists ($s = 0$). The result follows trivially.

Thus in many cases s can be guaranteed to remain small.

Using the partitioning given by equation (12), Property 1 becomes:

$$\begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} + G \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{bmatrix} = \begin{bmatrix} B_1^T & 0 \\ B_2^T & 0 \\ B_3^T & I \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} \tag{13}$$

and Property 2 becomes:

$$\begin{bmatrix} B_1 & B_2 & B_3 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{bmatrix} = 0 \tag{14}$$

where Δx_k and g_k have been partitioned corresponding to the partitioning of A (and the subscript k, referring to the iteration number, dropped for convenience).

From (14) we have

$$\Delta x_3 = 0 \tag{15}$$

and
$$\Delta x_1 = -W\Delta x_2 \tag{16}$$

where
$$W = B_1^{-1}B_2 \tag{17}$$

Thus,

$$\Delta x = \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x_2 .$$

Equation (13) simplifies when multiplied by the matrix

$$\begin{bmatrix} I & 0 & 0 \\ -W^T & I & 0 \\ 0 & 0 & I \end{bmatrix} \tag{18}$$

First it provides an expression for estimates of the Lagrange multipliers for the general constraints:

$$B_1^T\lambda_1 = g_1 + [I\ 0\ 0]\ G \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x_2 . \tag{19}$$

Note that when $\|\Delta x_2\| = 0$ (which will mean x is stationary) we have

$$B_1^T\lambda_1 = g_1 \tag{20}$$

in which case λ_1 is analogous to the pricing vector π in the revised simplex method. (From now on we shall denote the solution of (20) by π .) Next we have from (13) that

$$\lambda_2 = g_3 - B_3^T\lambda_1 + [0\ 0\ I]\ G \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x_2 \tag{21}$$

and again when $\|\Delta x_2\| = 0$ this equation reduces to

$$\underset{\sim}{\lambda}_2 = \underset{\sim}{g}_3 - B_3^T \underset{\sim}{\pi} \tag{22}$$

which is analogous to the vector of reduced costs in linear programming.

The third result from equation (13), following pre-multiplication by the matrix (18), is an expression for the appropriate step:

$$[-W^T \quad I \quad 0]\, G \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta \underset{\sim}{x}_2 = -\underset{\sim}{h} \tag{23}$$

where

$$\underset{\sim}{h} = [-W^T \quad I \quad 0]\underset{\sim}{g} = \underset{\sim}{g}_2 - W^T \underset{\sim}{g}_1 = \underset{\sim}{g}_2 - B_2^T \underset{\sim}{\pi} \ . \tag{24}$$

The form of equation (23) suggests that

$$[-W^T \quad I \quad 0]\, G \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \tag{25}$$

can be regarded as a "projected" (or "reduced") Hessian and $\underset{\sim}{h} = [-W^T \quad I \quad 0]\underset{\sim}{g}$ a projected gradient, with (23) giving a Newton step in the independent variables $\Delta \underset{\sim}{x}_2$. Note that $\|\underset{\sim}{h}\| = 0$ becomes a necessary condition for a stationary point in the current set of active constraints, which, if the projected Hessian is nonsingular, implies that $\|\Delta \underset{\sim}{x}_2\| = 0$.

While the above derivation is based on the two properties which characterize variable-metric projection algorithms, the resultant relations could be equally regarded as a reduced-gradient method in which the number of independent variables is reduced to s, the dimension of $\Delta \underset{\sim}{x}_2$. We will not spend time here discussing the relationship between

the two seemingly different approaches, but refer the reader to two review papers by Fletcher [17] and Sargent [49].

Recently Gill and Murray [25] have considered a class of algorithms in which the search direction along the surface of active constraints is characterized as being in the range of a matrix Z which is orthogonal to the matrix of constraint normals. Thus, if $\hat{A}\underset{\sim}{x} = \hat{\underset{\sim}{b}}$ is the current set of $n-s$ active constraints, Z is an $n \times s$ matrix such that

$$\hat{A}Z = 0 . \tag{26}$$

In the notation of [25], the main steps to be performed each iteration are as follows. (They generate a feasible descent direction $\underset{\sim}{p}$.)

A. Compute the projected (reduced) gradient $\underset{\sim}{g}_A = Z^T\underset{\sim}{g}$.

B. Form some approximation to the projected Hessian, viz.

$$G_A \doteq Z^T G Z$$

C. Obtain an approximate solution to the system of equations

$$Z^T G Z \underset{\sim}{p}_A = -Z^T\underset{\sim}{g} \tag{27}$$

by solving the system

$$G_A\underset{\sim}{p}_A = -\underset{\sim}{g}_A$$

D. Compute the search direction $\underset{\sim}{p} = Z\underset{\sim}{p}_A$.

E. Perform a line-search to find an approximation to α^*, where

$$f(\underset{\sim}{x} + \alpha^* \underset{\sim}{p}) = \min_{\substack{\alpha \\ \{\underset{\sim}{x} + \alpha \underset{\sim}{p} \text{ feasible}\}}} f(\underset{\sim}{x} + \alpha \underset{\sim}{p})$$

Apart from having full column rank, equation (26) is (algebraically) the only constraint on Z and thus Z may take several forms. The particular Z corresponding to our own procedure is of the form

$$Z = \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} = \begin{bmatrix} -B_1^{-1}B_2 \\ I \\ 0 \end{bmatrix} \begin{matrix} \} \, m \\ \} \, s \\ \} \, n-m-s \end{matrix} \qquad (28)$$

This is a convenient representation which we will refer to for exposition purposes in later sections, but we emphasize that computationally we work only with B_2 and a triangular (LU) factorization of B_1.

For many good reasons Gill and Murray [25] advocate a Z whose columns are orthonormal ($Z^TZ = I$). The principal advantage is that transformation by such a Z does not introduce unnecessary ill-conditioning into the reduced problem (see steps A through D above, in particular equation (27)). The approach has been implemented in programs described by Gill, Murray and Picken (e.g. [27]) in which Z is stored explicitly as a dense matrix. Extension to large sparse linear constraints would be possible via an LDV factorization (see Gill, Murray and Saunders [29]) of the matrix $[B_1 \;\; B_2]$:

$$[B_1 \;\; B_2] = [L \;\; 0]DV$$

where L is triangular, D is diagonal and $D^{1/2}V$ is orthonormal, with L and V being stored in product form. However if B_2 has more than 1 or 2 columns, this factorization will always be substantially more dense than an LU factorization of B_1. Thus on the grounds of efficiency we proceed with the Z in (28). At the same time we are conscious (from the unwelcome appearance of B_1^{-1}) that B_1 must be kept as well-conditioned as possible.

3. Implementation

The basic ideas were presented in the previous section; their actual implementation in a computer code requires a good deal more effort. The code itself is a Fortran program called MINOS* which is designed to be almost machine-independent and to operate primarily within main memory. The central part of MINOS is an efficient implementation of the revised simplex method which incorporates several recent advances in linear programming technology. These include:

1. Fast input of the constraint data in standard MPS format** using hash tables (in particular, the method of Brent [6]) for storing row-names and distinct matrix coefficients.
2. Compact in-core storage of the constraint matrix A using an elementary version of Kalan's super-sparseness techniques [36].
3. Upper and lower bounds on all variables.
4. A version of Hellerman and Rarick's "bump and spike" algorithm P^4 [33] for determining a sparse LU factorization of the basis matrix B_1.***
5. Imbedding of non-spike columns of L within A.
6. Stable updating of the LU factors of B_1 by the method of Bartels and Golub [2], [3] as implemented by Saunders [52].
7. An improved "CHUZR" procedure# for phase 1 of the simplex method, following ideas due to Rarick [48] and Conn [10].

---

*MINOS (my'-noss) = a Modular In-core Nonlinear Optimization System.

**This is the CONVERT data format described in user's manuals for the IBM systems MPS/360, MPSX and MPSX/370.

***The block-triangular structure of B_1 is currently being found using subroutines MC13 and MC16 from the Harwell Subroutine Library (Duff [14], Duff and Reid [15]). Hellerman and Rarick's P^3 [32] is then applied to each block.

#Implemented by J. A. Tomlin.

For optimization of the reduced function we have implemented a quasi-Newton procedure using the factorization $G_A = R^T R$ (R upper triangular) to approximate $Z^T G Z$. This parallels the methods described by Gill and Murray [21], [22], Gill, Murray and Pitfield [28] which are based on the Cholesky factorization $G_A = LDL^T$ (L lower triangular, D diagonal). Stable numerical methods based on orthogonal transformations are used for modifying R during unconstrained steps and for certain other modifications to R whenever the basis matrices B_1 and B_2 change. (Operations on R rather than L and D are somewhat easier to implement and involve little loss of efficiency in this context.)

Another module which is fundamental to the success of the present algorithm is an efficient and reliable line-search. The particular routine used is a Fortran translation of Gill and Murray's Algol 60 procedure delinsearch, which uses successive cubic interpolation with safeguards as described in [24]. This routine evaluates the objective function and its gradient simultaneously when required. We have left just one parameter available to the user to change at his/her discretion, namely, eta ($0.0 \leq$ eta < 1.0) which controls the accuracy of the search. This flexibility has proved to be very satisfactory in practice.

3.1. Summary of procedure

An outline of the optimization algorithm is given in this section; some of the finer points of implementation are discussed in later sections.

Assume we have the following:

(a) A feasible vector $\underset{\sim}{x} = [\underset{\sim}{x}_1 \quad \underset{\sim}{x}_2 \quad \underset{\sim}{x}_3]^T$ satisfying $[B_1 \quad B_2 \quad B_3]\underset{\sim}{x} = \underset{\sim}{b}$, $\underset{\sim}{\ell} \leq \underset{\sim}{x} \leq \underset{\sim}{u}$.

(b) The corresponding function value $f(\underset{\sim}{x})$ and gradient vector $\underset{\sim}{g}(\underset{\sim}{x}) = [\underset{\sim}{g}_1 \quad \underset{\sim}{g}_2 \quad \underset{\sim}{g}_3]^T$.

(c) The number of superbasic variables, s $(0 \leq s \leq n-m)$.

(d) A factorization, LU, of the $m \times m$ basis matrix B_1.

(e) A factorization, R^TR, of a variable-metric approximation to the $s \times s$ matrix Z^TGZ.

(f) A vector $\underset{\sim}{\pi}$ satisfying $B_1^T\underset{\sim}{\pi} = \underset{\sim}{g}_1$.

(g) The reduced gradient vector $\underset{\sim}{h} = \underset{\sim}{g}_2 - B_2^T\underset{\sim}{\pi}$.

(h) Small positive convergence tolerances TOLRG and TOLDJ.

Step 1. (Test for convergence in the current subspace)

If $\|\underset{\sim}{h}\| >$ TOLRG go to step 3.

Step 2. ("PRICE", i.e. estimate Lagrange multipliers, add one superbasic)

(a) Calculate $\underset{\sim}{\lambda} = \underset{\sim}{g}_3 - B_3^T\underset{\sim}{\pi}$.

(b) Select $\lambda_{q_1} <$ -TOLDJ ($\lambda_{q_2} >$ + TOLDJ), the largest elements of $\underset{\sim}{\lambda}$ corresponding to variables at their lower (upper) bound. If none, STOP; the Kuhn-Tucker necessary conditions for an optimal solution are satisfied.

(c) Otherwise,

(i) Choose $q = q_1$ or $q = q_2$ corresponding to

$$|\lambda_q| = \max|\lambda_{q_1}|, |\lambda_{q_2}|;$$

(ii) add $\underset{\sim}{a}_q$ as a new column of B_2;

(iii) add λ_q as a new element of $\underset{\sim}{h}$;

(iv) add a suitable new column to R.

(d) Increase s by 1.

Step 3. (Compute direction of search, $\underset{\sim}{p} = Z\underset{\sim}{p}_2$)

(a) Solve $R^T R\underset{\sim}{p}_2 = -\underset{\sim}{h}$.

(b) Solve $LU\underset{\sim}{p}_1 = -B_2\underset{\sim}{p}_2$.

(c) Set $$\underset{\sim}{p} = \begin{bmatrix} \underset{\sim}{p}_1 \\ \underset{\sim}{p}_2 \\ 0 \end{bmatrix}.$$

Step 4. (Ratio test, "CHUZR")

(a) Find $\alpha_{max} \geq 0$, the greatest value of α for which $\underset{\sim}{x} + \alpha\underset{\sim}{p}$ is feasible.

(b) If $\alpha_{max} = 0$ go to step 7.

Step 5. (Line-search)

(a) Find α, an approximation to α^*, where

$$f(\underset{\sim}{x} + \alpha^*\underset{\sim}{p}) = \min_{0 < \theta \leq \alpha_{max}} f(\underset{\sim}{x} + \theta\underset{\sim}{p}).$$

(b) Change $\underset{\sim}{x}$ to $\underset{\sim}{x} + \alpha\underset{\sim}{p}$ and set f and $\underset{\sim}{g}$ to their values at the new $\underset{\sim}{x}$.

Step 6. (Compute reduced gradient, $\bar{h} = Z^T g$)

(a) Solve $U^T L^T \pi = g_1$.

(b) Compute the new reduced gradient, $\bar{h} = g_2 - B_2^T \pi$.

(c) Modify R to reflect some variable-metric recursion on $R^T R$, using α, p_2 and the change in reduced gradient, $\bar{h} - h$.

(d) Set $h = \bar{h}$.

(e) If $\alpha < \alpha_{max}$ go to step 1. No new constraint was encountered so we remain in the current subspace.

Step 7. (Change basis, delete one superbasic)

Here $\alpha = \alpha_{max}$ and for some p $(0 < p \leq m+s)$ a variable corresponding to the p-th column of $[B_1 \quad B_2]$ has reached one of its bounds.

(a) If a basic variable hit its bound $(0 < p \leq m)$,

(i) interchange the p-th and q-th columns of

$$\begin{bmatrix} B_1 \\ x_1^T \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} B_2 \\ x_2^T \end{bmatrix}$$

respectively, where q is chosen to keep B_1 non-singular (this requires a vector π_p which satisfies $U^T L^T \pi_p = e_p$);

(ii) modify L, U, R and π to reflect this change in B_1;

(iii) compute the new reduced gradient $h = g_2 - B_2^T \pi$.

(b) If a superbasic variable hit its bound $(m < p \leq m+s)$, define $q = p-m$.

(c) Make the q-th variable in B_2 nonbasic at the appropriate bound, thus:

(i) delete the q-th columns of

$$\begin{bmatrix} B_2 \\ \underset{\sim}{x}_2^T \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} R \\ \underset{\sim}{h}^T \end{bmatrix} ;$$

(ii) restore R to triangular form.

(d) Decrease s by 1 and go to step 1.

3.2 Work per iteration

The work involved in one pass through the above procedure is roughly equivalent to

(a) one iteration of the revised simplex method on a linear program of dimensions $m \times n$, plus

(b) one iteration of a quasi-Newton algorithm on an unconstrained optimization problem of dimension s.

Note that the PRICE operation (Step 2) is performed only when $\|\underset{\sim}{h}\|$ is sufficiently small, which means an average of about once very 5 iterations. This is a typical frequency in commercial LP systems using multiple pricing. The extra work involved in the quasi-Newton steps is somewhat offset by the fact that a basis change (Step 7(a)) occurs only occasionally, so the growth of nonzeros in the LU factors of B_1 is minimal. Thus if s is of reasonable size and if $f(\underset{\sim}{x})$ and $g(\underset{\sim}{x})$ are inexpensive to compute, iterations on a large problem will proceed at about the same rate as if the problem were entirely linear.

3.3 Updating the Matrix Factorizations

As in the simplex method, a stable factorization of the basis matrix B_1 is important for solving equations of the form $B_1 y = b$ or $B_1^T z = c$. Here we use an implementation of the method of Bartels and Golub [2], [3] for updating the factorization $B_1 = LU$. Details are given in Saunders [52]. We normally re-factorize B_1 every 50 iterations regardless of the number of modifications that have been made to L and U.

The remainder of this section is devoted to the methods used for modifying R in the approximation $R^T R \approx Z^T G Z$ whenever x and/or Z change. The notation $\bar{R}$ will be used to represent R after any particular modification. To ensure stability, all modifications to R have been implemented using elementary orthogonal matrices Q_j (plane rotations) whose non-trivial elements are

$$\begin{bmatrix} c_j & s_j \\ s_j & -c_j \end{bmatrix}$$

where $c_j^2 + s_j^2 = 1$.

3.3.1. Variable-metric updates

Any of the usual updating formulas (e.g. Davidon [13], Fletcher and Powell [18], Broyden [7]) can be used to account for a nonzero change in the superbasic variables (Step 6). The two we have experimented with are:

The Complementary DFP formula

COMDFP:
$$\bar{R}^T \bar{R} = R^T R + \frac{1}{\alpha y^T R_2} y y^T + \frac{1}{h^T R_2} h h^T$$

The Rank-one Formula:

RANK1:
$$\bar{R}^T \bar{R} = R^T R + \frac{1}{\alpha w^T R_2} w w^T$$

where $y = \bar{h} - h$, the change in reduced gradient, and $w = y + \alpha h$.

The COMDFP formula can be used on both constrained and unconstrained steps ($\alpha = \alpha_{max}$ and $\alpha < \alpha_{max}$, resp.). An alternative is to use RANK1 on constrained steps as long as it results in a positive definite recursion, otherwise COMDFP. Systematic testing may perhaps reveal a slight advantage for one strategy over another, but in the interest of simplicity we now use COMDFP in either case.

If $\alpha = \alpha_{max}$ and α_{max} is very small it is possible that the computed value of y will be meaningless. Following the suggestion of M. J. D. Powell (private communication) we allow for this by monitoring the change in directional derivative and modifying R only if

$$\tilde{h}^T p_2 > 0.9\, h^T p_2 .$$

The same test is used even if $\alpha < \alpha_{max}$. Since $h^T p_2 < 0$, this means that R is modified if

$$\eta \equiv \frac{\tilde{h}^T p_2}{|h^T p_2|} < 0.9 ,$$

which will normally be true if a value $\underline{\text{eta}} < 0.9$ is given to the parameter of procedure delinsearch, which uses $|\eta| \leq \underline{\text{eta}}$ as one criterion for a successful search. (Note that $g^T p = g^T Z p_2 = h^T p_2$.)

Both COMDFP and RANK1 are implemented by means of the following routines:

R1ADD: $\bar{R}^T \bar{R} = R^T R + vv^T$

R1SUB: $\bar{R}^T \bar{R} = R^T R - vv^T$

These use forward and backward sweeps of plane rotations respectively, as described in Saunders [51, Ch. 7], Gill, Golub, Murray and Saunders [20].

3.3.2 Basis change (Step 7(a))

Suppose that the p-th basic variable is interchanged with the q-th superbasic variable. Once R has been updated to account for the move which is causing the basis change (Step 6), a further "static" update is required to allow for a corresponding change in the definition of Z. The relationship between the new null-space matrix and the old is given by

$$\bar{Z} = Z(I + \underset{\sim}{e}_q \underset{\sim}{v}^T) \qquad (29)$$

where $\underset{\sim}{e}_q$ is the q-th unit vector and $\underset{\sim}{v}$ is defined by the equations

$$B_1^T \underset{\sim}{\pi}_p = \underset{\sim}{e}_p$$

$$\underset{\sim}{y} = B_2^T \underset{\sim}{\pi}_p$$

$$y_q = \underset{\sim}{y}^T \underset{\sim}{e}_q$$

$$\underset{\sim}{v} = -\frac{1}{y_q}(\underset{\sim}{y} + \underset{\sim}{e}_q)$$

Derivation of this result is rather lengthy but the quantities involved are easily computed and they serve several purposes:

1. The j-th element of $\underset{\sim}{y}$, viz.

$$y_j = \underset{\sim}{y}^T \underset{\sim}{e}_j = \underset{\sim}{\pi}_p^T B_2 \underset{\sim}{e}_j = \underset{\sim}{e}_p^T B_1^{-1}(B_2 \underset{\sim}{e}_j)$$

is the pivot element that would arise if the j-th column of B_2 were selected for the basis change. Hence $\underset{\sim}{y}$ can be used as a guide for determining q. Broadly speaking, the condition of B_1 will be preserved as well as possible if y_q is the largest available pivot

element (assuming the columns of B_2 have similar norm). In practice it is reasonable to relax this condition slightly in favor of choosing a superbasic variable that is away from its bounds. Thus we define q by the following:

$$y_{max} = \max|y_j|$$

$$d_j = \min_{\{x_j \text{ superbasic}\}} |x_j - \ell_j|, |x_j - u_j|$$

$$d_q = \max\{d_j \mid |y_j| \geq 0.1\, y_{max}\}$$

This rule is numerically more reliable than that suggested by Abadie [1], which in the above notation is equivalent to maximizing $|y_j|d_j$.

2. π_p can be used to update the vector π that is computed in Step 6(a). (after the last move but before the current basis change). Thus

$$\bar{\pi} = \pi + (\bar{h}_q/y_q)\pi_p$$

where $\bar{h}_q$ is the appropriate element of the reduced gradient $\bar{h}$ in Step 6(b). This is the updating formula suggested by Tomlin [54] for use within the simplex method. Nonlinearity is irrelevant here since the basis change is simply a redefinition of Z.

3. π_p can also be used to update the LU factors of B_1 (see Tomlin [54], Goldfarb [31]).

The modification to R corresponding to equation (29) is accomplished as follows:

$$\text{R1PROD:} \quad \bar{R}^T\bar{R} = (I + \underset{\sim}{v}\underset{\sim}{e}_q^T)\, R^TR(I + \underset{\sim}{e}_q\underset{\sim}{v}^T)$$

If $\underset{\sim}{r}_q$ is the q-th column of R, this expression may be written

$$\bar{R}^T\bar{R} = (R^T + \underset{\sim}{v}\underset{\sim}{r}_q^T)(R + \underset{\sim}{r}_q\underset{\sim}{v}^T)$$

A partial backward sweep of plane rotations Q_j $(j = q, q-1, \ldots, 1)$ reduces $\underset{\sim}{r}_q$ to a multiple of $\underset{\sim}{e}_1$, and then a full forward sweep restores R to triangular form.

3.3.3 Removal of one superbasic variable (Step 7(c))

Removal of the q-th superbasic variable implies deletion of the corresponding column of R. The resulting upper-Hessenberg matrix is restored to triangular form $\bar{R}$ by a partial forward sweep of plane rotations Q_j $(j = q+1, \ldots, s)$:

$$\text{DELCOL:} \quad Q_s \cdots Q_{q+2}Q_{q+1} \times \begin{bmatrix} R \text{ with} \\ q\text{-th column} \\ \text{deleted} \end{bmatrix} = \begin{bmatrix} \bar{R} \\ \\ 0 \end{bmatrix}$$

3.3.4 Addition of one superbasic variable (Step 2(c))

When a vector $\underset{\sim}{a}_q$ is added to B_2 the new null-space matrix is

$$\bar{Z} = [Z \quad \underset{\sim}{z}] \qquad \text{where} \quad \underset{\sim}{z} = \begin{bmatrix} -B_1^{-1}\underset{\sim}{a}_q \\ \underset{\sim}{e}_s \\ 0 \end{bmatrix}$$

Following Gill and Murray ([25], pp. 76-77) we approximate the vector $G\underset{\sim}{z}$ by finite differences, thus:

$$\underset{\sim}{y} = \frac{\underset{\sim}{g}(\underset{\sim}{x} + \delta\underset{\sim}{z}) - \underset{\sim}{g}(\underset{\sim}{x})}{\delta} = G\underset{\sim}{z} + O(\delta\|\underset{\sim}{z}\|^2)$$

where δ is a small step in the direction $\underset{\sim}{z}$, for example, $\delta = \epsilon^{1/2}/\|\underset{\sim}{z}\|$. The following procedure can then be used to generate a new column for R:

ADDCOL:

Solve $\quad R^T\underset{\sim}{r} = Z^T\underset{\sim}{y}$

Compute $\quad \sigma = \underset{\sim}{z}^T\underset{\sim}{y} - \|\underset{\sim}{r}\|^2, \quad \rho = |\sigma|^{1/2}$

Take $\quad \bar{R} = \begin{bmatrix} R & \underset{\sim}{r} \\ & \rho \end{bmatrix}$

(Note that $\underset{\sim}{z}^T\underset{\sim}{y}$ is best computed as the last element of $\bar{Z}^T\underset{\sim}{y}$ rather than from $\underset{\sim}{z}$ and $\underset{\sim}{y}$ directly.)

Comparison of

$$\bar{R}^T\bar{R} = \begin{bmatrix} R^T & \\ \underset{\sim}{r}^T & \rho \end{bmatrix}\begin{bmatrix} R & \underset{\sim}{r} \\ & \rho \end{bmatrix} = \begin{bmatrix} R^TR & Z^T\underset{\sim}{y} \\ \underset{\sim}{y}^TZ & \underset{\sim}{z}^T\underset{\sim}{y} \end{bmatrix}$$

and

$$\bar{Z}^T G \bar{Z} = \begin{bmatrix} Z^T \\ \underset{\sim}{z}^T \end{bmatrix} G \begin{bmatrix} Z & \underset{\sim}{z} \end{bmatrix} = \begin{bmatrix} Z^T G Z & Z^T G \underset{\sim}{z} \\ \underset{\sim}{z}^T G Z & \underset{\sim}{z}^T G \underset{\sim}{z} \end{bmatrix}$$

shows that if $R^T R$ provides a good approximation to $Z^T G Z$ then $\bar{R}^T \bar{R}$ has some chance of being a useful approximation to $\bar{Z}^T G \bar{Z}$. The main work involved here is in computing $B_1^{-1} \underset{\sim}{a}_q$, the gradient vector $\underset{\sim}{g}(\underset{\sim}{x} + \delta \underset{\sim}{z})$ and the projection $\bar{Z}^T \underset{\sim}{y}$. This work is essentially wasted if the expression for σ is not positive, which may happen for many reasons, e.g. if $\bar{Z}^T G \bar{Z}$ is not positive definite at the current point, if R is a poor approximation or if R is very ill-conditioned. In such cases we set $\underset{\sim}{r} = 0$ and take ρ to be either $(\underset{\sim}{z}^T \underset{\sim}{y})^{1/2}$ or 1.0, thus:

$$\bar{R} = \begin{bmatrix} R & 0 \\ & \rho \end{bmatrix} \qquad (30)$$

One advantage, at least, is that the subsequent search direction will move the new superbasic variable x_q away from its bound, so there is no danger of cycling on x_q.

With many problems the condition $\sigma \leq 0$ occurs only occasionally or not at all. Computing $\underset{\sim}{r}$ and ρ as shown then leads to significantly fewer iterations than if (30) were used all the time. On the other hand, $\sigma > 0$ is not a sufficient condition for success. In particular if the current point is near a singularity in $\underset{\sim}{g}(\underset{\sim}{x})$ the difference approximation to $G\underset{\sim}{z}$ is unlikely to be good. (An example is when $f(\underset{\sim}{x})$ has terms of the form $x_j \log x_j$ and the constraints include bounds such as $x_j \geq 10^{-10}$.) In such cases, $\underset{\sim}{r}$ and ρ prove to be consistently very large, resulting

in an $\bar{R}$ which is much more ill-conditioned than R. Subsequent iterations make little progress until the associated quasi-Newton updates restore the condition of $\bar{R}$. In contrast, use of (30) with $\rho = 1.0$ gives rapid progress.

Let d_{max} and d_{min} be the largest and smallest diagonals of R. As a heuristic means of detecting the above situation we monitor $\|\underset{\sim}{v}\|$ and resort to (30) whenever $\|\underset{\sim}{v}\|$ is significantly large than d_{max} or smaller than d_{min}. (As a side benefit, the expense of computing $\bar{Z}^T\underset{\sim}{v}$ and $\underset{\sim}{r}$ is then avoided.) A final similar test is made on ρ.

In contrast to all previous discussion, the ADDCOL procedure just described embodies a discernible level of ad hoc strategy. However our experience with it has been good in general, and the combined use of R1PROD, DELCOL and ADDCOL is almost certainly better than resetting $\bar{R} = I$ at every change to the set of active constraints.

3.4 Convergence tests

Another area in which strategy plays an important practical role is in deciding when to stop optimizing in the current subspace and consider moving away from one of the active constraints. Here we must enlarge on the use of TOLRG in Section 3.1; recall that in Step 1 of the algorithm, TOLRG was tested to determine if it was time to compute Lagrange multipliers (reduced costs, $\underset{\sim}{\lambda}$) and add one more superbasic variable.

Suppose that after a particular iteration we have

$\Delta \underset{\sim}{x}_2$ = the change in the superbasic variables

Δf = the change in f

$\underset{\sim}{\pi}$ = the new pricing vector

$\underset{\sim}{h} = Z^T g$, the new reduced gradient

ϵ_x, ϵ_f, TOLRG, ϵ_g = positive scalars

ϵ = machine precision

and let T_i be a set of tests (with values true or false) defined as follows:

T_1: $\quad \|\Delta \underset{\sim}{x}_2\| \leq (\epsilon_x + \epsilon^{1/2})(1 + \|\underset{\sim}{x}_2\|)$

T_2: $\quad |\Delta f| \leq (\epsilon_f + \epsilon)(1 + |f|)$

T_3: $\quad \|\underset{\sim}{h}\| \leq \text{TOLRG}$

T_4: $\quad \|\underset{\sim}{h}\| \leq \epsilon_g \|\underset{\sim}{\pi}\|$

In place of the simple test

$$\underline{\text{if}}\ \ T_3\ \ \underline{\text{then}}\ \text{compute}\ \ \lambda\ ,$$

the following combined test is used:

$$\underline{\text{if}}\ \ (T_1\ \ \underline{\text{and}}\ \ T_2\ \ \underline{\text{and}}\ \ T_3)\ \ \underline{\text{or}}\ \ T_4\ \ \underline{\text{then}}\ \text{compute}\ \ \lambda\ .$$

The general form of this test follows that used in the algorithm lcmna of Gill, Murray and Picken [27], in which the scalars identified here by ϵ_x, ϵ_f, TOLRG and ϵ_g are fixed at certain "loose" values initially and are then reset to "tight" values once it appears that the optimal set of active constraints has been identified. Use of ϵ_x and ϵ_f in this way is justified in the sense that it seems reasonable to remain on the present set of active constraints as long as significant progress is being made. Use of ϵ_g in T_4 allows for the possibility that the last step, though significant, may have moved x very close to an optimum in the current subspace (e.g. the quasi-Newton procedure should achieve this regularly if $f(x)$ is quadratic).

In adopting the above strategy we have found it beneficial to vary TOLRG dynamically. In the current version of MINOS this is done as follows. Suppose that the "best" Lagrange multiplier at some stage is $\lambda_q = g_q - \pi^T a_q$. If the corresponding variable x_q becomes superbasic, the reduced gradient for the expanded subspace will be

$$\bar{h} = \begin{bmatrix} h \\ \lambda_q \end{bmatrix}$$

Now recall from equation (21) that unless h is reasonably small, even one further iteration could change π and hence λ_q significantly.

Therefore as a safeguard (which is admittedly heuristic) we accept λ_q and move into the new subspace only if $\|\underset{\sim}{h}\|_\infty \leq 0.9|\lambda_q|$, which implies

$$\|\underset{\sim}{h}\|_\infty \leq 0.9\|\bar{\underset{\sim}{h}}\|_\infty .$$

We then reset TOLRG for the new subspace to be

$$\text{TOLRG} = \eta_g \|\bar{\underset{\sim}{h}}\|_\infty$$

where $\eta_g \in (0,1)$ is a parameter which is available to the user to set at his own will (and peril!). A typical value is $\eta_g = 0.2$ and its function is analogous to that of the parameter eta in procedure delinsearch. For example a small value of η_g allows the user to insist on an accurate optimization within each subspace.

4. Use of first and second derivatives

We have assumed throughout that the gradient $g(\underset{\sim}{x})$ is available and we have avoided explicit use of the Hessian matrix $G(\underset{\sim}{x})$. Some discussion of alternatives is in order. The principal factor here is the expense of transforming even one vector by Z or Z^T. In fact if the constraint matrix A has many rows, most of the work per iteration lies in computing $\underset{\sim}{p} = Z\underset{\sim}{p}_2$ and $\underset{\sim}{h} = Z^T\underset{\sim}{g}$. (These calculations are analogous to the FTRAN and BTRAN operations in linear programming.)

1. When g is not available it would often be practical to form an approximation $\hat{g}$ using finite differences along the coordinate directions, e.g.,

$$\hat{g}_j = \frac{f(\underset{\sim}{x} + \delta\underset{\sim}{e}_j) - f(\underset{\sim}{x})}{\delta} \approx g_j .$$

(The number of $\hat{g}_j$'s to be computed this way is equal to the number of nonlinear variables.) Just one transformation with Z^T is then required, viz. $\underset{\sim}{h} \approx Z^T\hat{g}$.

2. An alternative which is normally viable would be to difference $f(\underset{\sim}{x})$ along the directions $\underset{\sim}{z}_j$:

$$\hat{h}_j = \frac{f(\underset{\sim}{x} + \delta\underset{\sim}{z}_j) - f(\underset{\sim}{x})}{\delta} \approx \underset{\sim}{z}_j^T\underset{\sim}{g} = h_j$$

where $\underset{\sim}{z}_j = Z\underset{\sim}{e}_j$, $j = 1, \ldots, s$. Unfortunately this approach is not practical for large problems, since storage limitations prevent saving all s vectors $\underset{\sim}{z}_j$, and the work involved rules out recomputing them when required.

3. If $g(x)$ and perhaps $G(x)$ are available, the system of equations

$$Z^T G Z p_2 = - Z^T g \qquad (31)$$

could sometimes be treated by a modified Newton method (Gill and Murray [23], Gill, Murray and Picken [27]). This involves either computing $Z^T G Z$ directly:

$$Z^T G Z = [z_i^T G z_j]$$

or differencing $g(x)$ thus:

$$v_j = \frac{g(x + \delta z_j) - g(x)}{\delta} \equiv V e_j ,$$

$$Z^T G Z \approx \frac{1}{2} (Z^T V + V^T Z) .$$

However the need for the vectors z_j again presents severe difficulties for large problems.

4. If G is large and sparse, equation (31) could sometimes be solved iteratively by the method of conjugate gradients (e.g. see Gill and Murray [25, p. 133]). Storage is minimal since the method avoids forming the matrix $Z^T G Z$ or any approximation to it. However if Z has s columns the method usually requires $O(s)$ products of the form $Z^T(G(Zy))$.

5. A final (more promising) alternative is to abandon equation (31) and to generate a search direction by a nonlinear conjugate-gradient type method such as that of Fletcher and Reeves [19] (e.g. see Gill and Murray ([25], p. 134)). This takes the form

(a) $\bar{h} = -Z^T\bar{g}$.

(b) <u>if</u> restart <u>then</u> $\bar{p}_2 = -\bar{h}$

<u>else</u> $\bar{p}_2 = -\bar{h} + \beta p_2$

(c) $p = Z\bar{p}_2$

where p_2, $\bar{p}_2$ are the previous and current search directions for the superbasics. Several methods have been suggested for determining the scalar β, e.g.

Fletcher and Reeves [19]:	$\beta = \|\bar{h}\|^2/\|h\|^2$
Polak and Ribiere [46]:	$\beta = \bar{h}^T(\bar{h} - h)/\|h\|^2$
Perry [45],	$\beta = \bar{h}^T(\bar{h} - h - \alpha p_2)/p_2^T(\bar{h} - h)$

In MINOS, a switch is made to one of these methods if the number of superbasics s becomes larger than the dimension specified for the matrix R. A restart occurs whenever the set of active constraints changes; also every s+1 iterations in the (rare) event that more than s consecutive steps are unconstrained. More refined restart procedures (e.g. Powell [47]) will require future investigation. In the present environment the above formulas for β have all performed rather similarly (though seldomly as well as quasi-Newton). An example is given in §5.2.4.

To summarize: the reduced-gradient approach allows maximum efficiency in dealing with large sparse linear constraints, but at the same time it alters our perspective on the relative merits of Newton, quasi-Newton and conjugate gradient methods for handling the nonlinear objective. Strangely enough, even if the <u>exact</u> Hessian matrix were available (no matter how sparse)

we could not afford to use it. In this context we find that quasi-Newton methods take on a new and unexpected importance. The storage required for the Hessian approximation is often moderate even when there are many linear or nonlinear variables, as long as the total number of superbasic variables is of order 100 (say) or less. Otherwise, a conjugate-gradient method remains the only viable alternative.

4.1. Quadratic programs

The above statements do not hold if G happens to be a constant matrix. In this case the relation

$$R^TR = Z^TGZ \tag{32}$$

can often be maintained exactly without recomputing Z^TGZ every iteration. Such a specialization has been described by Gill and Murray [26], along with the measures required to allow for Z^TGZ being indefinite. The present quasi-Newton algorithm could be specialized as follows:

1. Initialize R at the start of a run to satisfy (32). (This is trivial if there are no superbasics; it may not be possible for an arbitrary set of superbasics since Z^TGZ could be indefinite.)
2. In procedure ADDCOL (§3.3.4) compute the vector $y = Gz$ directly rather than by differencing the gradient.
3. Suppress the variable-metric updates to R (COMDFP and RANK1 in §3.3.1).

However it is worth noting that the difference approximation to $\underset{\sim}{y} = G\underset{\sim}{z}$ will be essentially exact, so that if (32) ever holds at any stage then ADDCOL will maintain (32) almost exactly when a column is added to Z. A step $\alpha = 1.0$ along the next search direction will then move $\underset{\sim}{x}$ to the new subspace minimum. Now it is easily verified that the subsequent variable metric updates will cause no net change to R (ignoring slight rounding error in the case of COMDFP). The scene is therefore set for another exact minimization during the next iteration.

The above sequence will be broken if a constraint forces some step α to be less than 1.0. The variable-metric updates will then alter R slightly, (32) will cease to hold and the next subspace minimization may require more than one iteration. In certain applications this could be undesirable, but more generally the robustness and self-correcting properties of quasi-Newton methods offer compensating advantages, including the ability to start with any matrix R (such as I). Suffice to say that the general algorithm comes close to being "ideal" on quadratic programs, without undue inefficiency or any specialized code.

7. Computational Experience

Although the prime application of this research is to large-scale linear programs with a nonlinear objective function, we have endeavored to attack a comprehensive range of problems to aid development of the algorithm. It is unfortunate that large-scale nonlinear problems are not widely reported in the literature, so that many of the results discussed here refer to problems which are solely within the authors' own purview. A brief description of each problem is given. Fuller details of constraint data, starting points, etc. must be left to a future report.

Three of the starting options provided in MINOS are as follows:

1. (CRASH) A triangular basis matrix is extracted from the matrix A, without regard to feasibility or optimality. The number of superbasic variables is set to zero.
2. (Initialization of nonlinears) The user specifies values for any number of the nonlinear variables. These are made superbasic. CRASH is then applied to the linear variables in A.
3. (Restart) A previously-saved bit-map is loaded (specifying the state of all variables), along with values for any superbasic variables. This allows continuation of a previous run, or an advanced start on a different but related problem (for example the bounds $\ell \leq x \leq u$ may be changed).

Options 2 and 3 normally reduce run time considerably, but the results reported here were obtained using the "cold start" option 1 unless otherwise stated. A normal phase 1 simplex procedure was used to obtain an initial feasible solution.

5.1. Description of Test Problems

1. Colville No. 1. This is problem no. 1 in the Colville series of test problems [9]. The objective is a cubic function of 5 variables.

2. Colville No. 7. This is a quartic function of 16 variables.

3. Chemical Equilibrium Problem. This particular example of the chemical equilibrium problem was obtained from Himmelblau [34], problem 6. The objective is of the form

$$f(\underset{\sim}{x}) = \sum_{k} [\sum_{j} x_{jk}(c_{jk} + \ell n(x_{jk}/\sum_{i} x_{ik}))] \ .$$

(Note. Slight corrections were made to the constraint data in [34, p. 401]. The group of coefficients $\{-1,-2,-3,-4\}$ in column 13 was moved to column 14, and a similar group in column 12 was moved to column 13.)

4. Weapon Assignment Problem. This problem appeared originally in Bracken and McCormick's book on nonlinear programming applications [5], and more recently in Himmelblau [34], problem 23. The objective function is

$$f(\underset{\sim}{x}) = \sum_{j=1}^{20} u_j \left(\prod_{i=1}^{5} a_{ij}^{x_{ij}} - 1 \right)$$

with unknowns $x_{ij} \geq 0$. We have ignored the requirement that the x_{ij} be integers.

5. Structures Optimization (Q.P.). This is a series of quadratic programming problems in structures design [58].

6. Oil Refinery Investment Model. This is typical of many linear programming based oil refinery models, but has the added feature that nonlinear returns to scale of capital equipment costs are defined explicitly. The particular problem cited in the results has 15 nonlinear variables of this kind.

7. Energy Submodel. A related research project on the development a national energy model [43] has given rise to a fairly complex submodel of the electricity sector. The 24 nonlinear variables are mainly the capacities of the different types of generating equipment.

8. Expanded Energy System Model. An expanded model which covers all aspects of energy production and distribution on a national level has been developed [53]. This is a medium-scale linear program with 91 nonlinear variables in the objective; again these are mainly nonlinear returns to scale of capital equipment costs of the form

$$\sum_{i=1}^{91} c_i x_i^{p_i} \qquad \text{with} \quad 0 < p_i < 1 \text{ (around 0.6 to 0.7).}$$

9. Energy Model RS8. This is a 16-period energy model which was formulated from the outset as a nonlinear programming problem (see Manne [38], [39]). The objective is of the form

$$\sum_{i=3}^{16} \frac{a_i}{x_i y_i^2} + \text{linear terms}$$

with one pair of nonlinear variables x_i, y_i for each time period (those for the first two periods being known). This was the first large problem available to us and is of interest for several reasons. In particular it provides a comparison with a (considerably larger) linear approximation to the problem, in which each term $a_i/x_i y_i^2$ was discretized over a two-dimensional grid. Further details are given in §5.2.2.

10. Energy Model ETA (Manne [40]). This is a further development of the previous model. The objective is the same as in RS8 with the addition of $\Sigma_{i=1}^{16} z_i^2$ for 16 variables z_i.

5.2. Results

The results summarized in Table 1 were obtained on a Burroughs B6700 computer using single-precision arithmetic ($\epsilon \approx 10^{-11}$). The standard time ratios quoted are relative to the processor time required for a standard timing program given in Colville [9]. The standard time for unoptimized B6700 Fortran is 83.07 seconds.

The results in Table 2 onwards were obtained using double precision arithmetic on an IBM 370/168 ($\epsilon \approx 10^{-15}$). The standard time for this machine with the IBM Fortran IV (H extended) compiler with full optimization is 3.92 seconds. A fairly accurate line-search was normally used (eta = 0.01) and the quantity $\|\underset{\sim}{h}\|/\|\underset{\sim}{x}\|$ was reduced to 10^{-6} or less at optimality.

Problem no.	Rows	Columns	Nonzero elements	Nonlinear variables	Iterations*	Evaluations of $f(\underset{\sim}{x})$, $g(\underset{\sim}{x})$	Final no. of superbasics	Time (secs.)	Standard time ratio
1	10	5	47	5	8	9	1	0.63	0.008
2	8	16	80	16	15	16	3	1.50	0.018
4	12	100	147	100	133	296	18	48.30	0.58
5a	10	24	240	24	8	8	14	1.65	0.019
5b	17	52	884	52	13	13	35	6.21	0.075
5c	19	78	1482	78	21	21	59	13.47	0.16
6	74	83	529	15	80	40	3	37.03	0.45
7	95	200	5[illegible]	24	103	72	0	42.43	0.51
8	324	425	[illegible]	91	348	215	0	538.3	6.48

Table 1. Solution of problems 1-2, 4-8 on Burroughs B6700

*Includes Phase 1 iterations

Problem no.*	Rows	Columns	Nonzero elements	Nonlinear variables	Iterations	Evaluations of $f(x)$, $g(x)$	Final no. of superbasics	Time (secs.)	Standard time ratio
3	16	45	99	45	103	452	24	2.9	0.74
4	12	100	147	100	139	355	18	2.6	0.66
9a	356	1134	4180	0	539	0	0	33.3	8.5
9b	314	631	2122	28	2027	4536	21	119.5	30.5
9c	314	631	2122	28	1397	2862	21	83.6	21.3
10a	320	679	2519	44	948	1768	27	56.6	14.4
10b	320	679	2519	44	236	570	29	17.5	4.5
10c	320	679	2519	44	350	902	26	26.9	6.9

Table 2. Solution of problems 3-4, 9-10 on IBM 370/168.

* 9a = RS8, linearized
9b = RS8, cold start
9c = RS8, cold start, scaled
10a = ETA, BOUNDS = Q2NONE, cold start
10b = ETA, BOUNDS = Q2NOFB, restart from 10a
10c = ETA, BOUNDS = Q2BOTH, restart from 10b

5.2.1. The chemical equilibrium problem (problem 3)

This example provided useful experience in dealing with logarithmic singularities in $g(x)$. The objective consists of functions of the form

$$f = \sum_i x_i g_i$$

whose gradient components are

$$g_i = c_i + \ln \frac{x_i}{\sum_j x_j} .$$

If some x_i is zero, the corresponding term in f may be correctly programmed as $(x_i g_i) = 0$. However, g_i itself is then analytically minus infinity (unless all $x_j = 0$), and any particular numerical value given to it in the gradient subroutine will result in a discontinuity in g_i as x_i moves (even slightly) away from zero. To avoid this difficulty we ran the problem with a uniform lower bound $\epsilon_k \equiv 10^{-k}$ on all variables, for various values of k in the range 4 to 10. (The problem is infeasible with $x_j \geq 10^{-3}$.) Results are summarized in Table 3, where each run continued from the run before using starting option 3. The minimal change in $f(x)$ is typical of dual geometric programs, but values $x_j = 10^{-6}$ and $x_j = 10^{-10}$ (say) have very different physical interpretations and therefore warrant more than the usual degree of resolution.

In Table 4 we list the largest solution value x_{13} and the 8 smallest values in the order by which they became superbasic. The most significant variation is in x_{45}. Most values have stabilized by the time k reaches 10.

Lo-bound ϵ_k	No. of superbasics	f(x)	Iterations*	Evaluations* of f, g	Estimate of** $\kappa(R^TR)$
10^{-4}	10	-1910.366249932	46	130	6×10^5
10^{-5}	14	-1910.381531984	21	75	5×10^6
10^{-6}	17	-1910.382772060	22	72	1×10^8
10^{-7}	19	-1910.382872190	22	88	1×10^9
10^{-8}	23	-1910.382880402	22	90	6×10^7
10^{-9}	24	-1910.382881101	22	90	4×10^8
10^{-10}	24	-1910.382881161	5	27	8×10^7
			160	572	

Table 3. Solution of problem 3 with various bounds $x_j \geq \epsilon_k$.

*Additional to previous run.

**A lower bound on the condition number of the projected Hessian approximation R^TR is the square of the ratio of the largest and smallest diagonals of R.

x_j / ϵ_k	j=13 ($x_{9,2}$)	45 ($x_{2,7}$)	9 ($x_{5,2}$)	28 ($x_{11,3}$)	8 ($x_{4,2}$)	22 ($x_{5,3}$)	32 ($x_{15,3}$)	40 ($x_{2,5}$)	21 ($x_{4,3}$)
10^{-4}	44.096011	2.229e-4							
10^{-5}	44.139306	2.257e-5							
10^{-6}	44.143326	2.261e-6							
10^{-7}	44.143643	2.261e-7	4.192e-7	1.033e-7					
10^{-8}	44.143644	2.602e-8	4.192e-7	1.033e-7	3.708e-8	2.241e-8	2.302e-8	1.091e-8	
10^{-9}	44.143645	7.493e-9	4.192e-7	1.033e-7	3.708e-8	2.241e-8	2.169e-8	3.155e-9	2.426e-9
10^{-10}	44.143645	5.462e-9	4.192e-7	1.033e-7	3.708e-8	2.241e-8	2.098e-8	5.462e-9	2.562e-9
SUMT	44.58	2.476e-6	5.441e-6	3.437e-6	2.794e-6	3.102e-6	3.264e-6	0.0	1.546e-6

Table 4. Selected solution values for Problem 3. Blank entries mean x_j was nonbasic at the appropriate bound ϵ_k.

For interest, the last row of Table 4 shows the values obtained by the program SUMT as reported by Himmelblau [34]. There appear to be no accurate significant digits in the 8 smallest x_j. (This may be due to differences in the constraint data, errors in satisfying the general constraints, or simply different machine precisions.)

Note that when x_j is small the diagonal elements of the Hessian matrix are $\partial g_j/\partial x_j = O(1/x_j)$. However these large elements affect the reduced Hessian only when x_j is basic or superbasic. The safest strategy for these problems therefore appears to be the following:

(a) Solve the problem with relatively large lower bounds, e.g. $x_j \geq 10^{-4}$. A near-optimal objective value will be obtained quickly because the reduced Hessian remains reasonably well-conditioned.

(b) Reduce the lower bounds, perhaps in stages, to $O(\epsilon^{1/2})$ or $O(\epsilon^{2/3})$. There will be essentially no further basis changes, and in roughly descending order the small x_j will leave their bounds one by one to become superbasic.

Solution of problem 3 with $x_j \geq 10^{-4}$ followed by $x_j \geq 10^{-10}$ required a total of 103 iterations and 452 function/gradient evaluations as shown in Table 2. Solution with $x_j \geq 10^{-10}$ directly required 188 iterations and 886 evaluations, primarily because the Hessian approximation became very ill-conditioned before a near-optimal point was reached.

As a natural precaution against rounding error the linesearch procedure delinsearch avoids evaluating $f(\underset{\sim}{x} + \alpha \underset{\sim}{p})$ with values of α that are very close together. On the IBM 370/168 this prevented resolution below 10^{-10}, although for this special case $f(\underset{\sim}{x})$ could

easily be evaluated using higher precision arithmetic. The limiting factor would then become the condition of the reduced Hessian.

5.2.2. Energy model RS8

Problem 9a in Table 2 refers to the original linearized version of the energy model, in which each term of the form

$$f(x,y) = \frac{a}{xy^2}$$

was approximated over a 6×6 grid. It has twice as many columns and matrix coefficients as the nonlinear version 9b. Note that construction of the small but reasonably fine grid required good prior estimates of the optimal values for the 14 (x,y) pairs.

Run 9b is included to illustrate the rather poor performance that could be encountered during early "de-bugging" of a nonlinear problem. Some relevant facts follow.

(a) The bounds on nonlinear variables were conservative in the sense that the lower bounds were far removed from the optimal solution values and there were no upper bounds.

(b) No attempt was made to initialize the nonlinears at reasonable values between their bounds.

(c) The y variables proved to be badly scaled.

To enlarge on the last point, the Hessian matrix of $f(x,y)$ above is

$$G(x,y) = \frac{2}{x^3 y^4}\begin{bmatrix} y^2 & xy \\ xy & 3x^2 \end{bmatrix} = \frac{2}{x^3 y^4}\begin{bmatrix} y & \\ x & \sqrt{2}\,x \end{bmatrix}\begin{bmatrix} y & x \\ & \sqrt{2}\,x \end{bmatrix}$$

and it follows from the diagonal elements of the triangular factor that G has a condition number $\kappa(G) \geq y^2/2x^2$. Now the optimal values for the x and y variables are all $O(1)$ and $O(100)$ respectively, which might normally be considered well-scaled; however it means that $\kappa(G)$ is at least $O(10^4)$, which in this case is unnecessarily large. Replacing each y by a variable $\bar{y} = y/100$ gave a significant improvement as shown by run 9c in Table 2.

5.2.3. Energy model ETA

It is in runs 10a-10c that the real benefits from a nonlinear optimizer become apparent. This is an example of the model-builder's standard mode of operation wherein numerous runs are made on a sequence of closely related problems with the solution from one run providing a starting point for the next. Here, problem 10a (the base case) was solved from a cold start with certain variables fixed at zero; for run 10b the bounds were relaxed on 16 of these variables, and for run 10c a further 10 variables were freed. (In this particular sequence the starting solutions for 10b and 10c were clearly feasible. This is desirable but not essential.)

Compared to solving linearized approximations by standard linear programming, some of the obvious advantages are:

1. reduced problem size;
2. reduced volume of output (in the absence of a report writer);
3. ability to prepare data for several runs in advance, since there are no grid variables to be moved or refined;
4. the solution obtained actually solves the correct problem.

5.2.4. Comparison of Quasi-Newton and Conjugate Gradients

The weapon assignment problem (no. 4) was chosen here as a reasonably small but nontrivial example. About 60 changes in the active constraint set occur during the iterations.

The parameters being varied are

η = linesearch accuracy tolerance (eta in §3);

η_g = the tolerance for minimization within each subspace (see §3.4).

Recall that small values of these parameters mean accurate minimization. For Table 5 we set $\eta_g = 0.5$ and compared the normal Quasi-Newton algorithm with each of the conjugate gradient algorithms for various values of η. We find that Quasi-Newton is consistently superior and is quite robust with respect to diminishing linesearch accuracy, in contrast to the conjugate gradient (cg) algorithms. Unfortunately there is no discernible trend that singles out one cg algorithm over another.

For Table 6 the same runs were made with $\eta_g = 0.01$. (A more accurate subspace minimization makes the sequence of constraint changes more consistent between runs.) This smoothed out the iteration and function-evaluation counts, but again there is no evidence to favor any particular cg algorithm.

To illustrate that the cg methods are not to be discarded immediately, in Figure 1 we have plotted the value of $f(\underset{\sim}{x})$ against iteration number for the second row and first two columns of both Tables 5 and 6. Thus a reasonably accurate linesearch was used for all cases ($\eta = 0.01$). Curves 1 and 2 compare Quasi-Newton with Fletcher-Reeves using $\eta_g = 0.5$, and curves 3 and 4 do the same with $\eta_g = 0.01$.

The first two curves show smooth progress for both methods. Note that although the cg method lags behind it has essentially identified the final set of active constraints by the time the Quasi-Newton method converges (iteration 139). The step-function shape of curves 3 and 4 illustrates the work that is wasted in converging to minima within each subspace. Otherwise these curves effectively place a magnifying glass on the tail end of the other runs. The terminal convergence of the cg method is clearly very slow and it is here that better restart procedures such as in Powell [47] should prove to be most valuable.

η	Quasi-Newton		Fletcher-Reeves		Polak-Ribiere		Perry	
0.001	123	375	226	840	222	806	198	713
0.01	139	255	223	728	237	770	259	849
0.1	122	281	227	671	238	709	228	665
0.2	137	300	250	721	252	749	218	578
0.3	148	291	239	648	248	688	307	814
0.4	156	289	282	742	296	853	309	762
0.5	153	242	275	695	394	1079	612	1411
0.9	207	256	694	987	>999	>2748	818	968

Table 5. Iterations and function + gradient evaluations for the weapon assignment problem; $\eta_g = 0.5$; various linesearch tolerances η.

η	Quasi-Newton		Fletcher-Reeves		Polak-Ribiere		Perry	
0.001	220	615	493	1628	440	1514	440	1495
0.01	219	548	498	1520	471	1520	466	1476
0.1	209	461	560	1597	508	1461	530	1568
0.2	218	445	582	1589	531	1517	585	1626
0.3	229	411	612	1557	634	1752	611	1625
0.4	262	441	748	1831	691	1821	752	1788
0.5	262	377	691	1633	818	1993	894	1974
0.9	288	345	>999	>1855	>999	>1658	>999	>1156

Table 6. Iterations and function + gradient evaluations for the weapon assignment problem; $\eta_g = 0.01$ (more accurate minimization within each subspace).

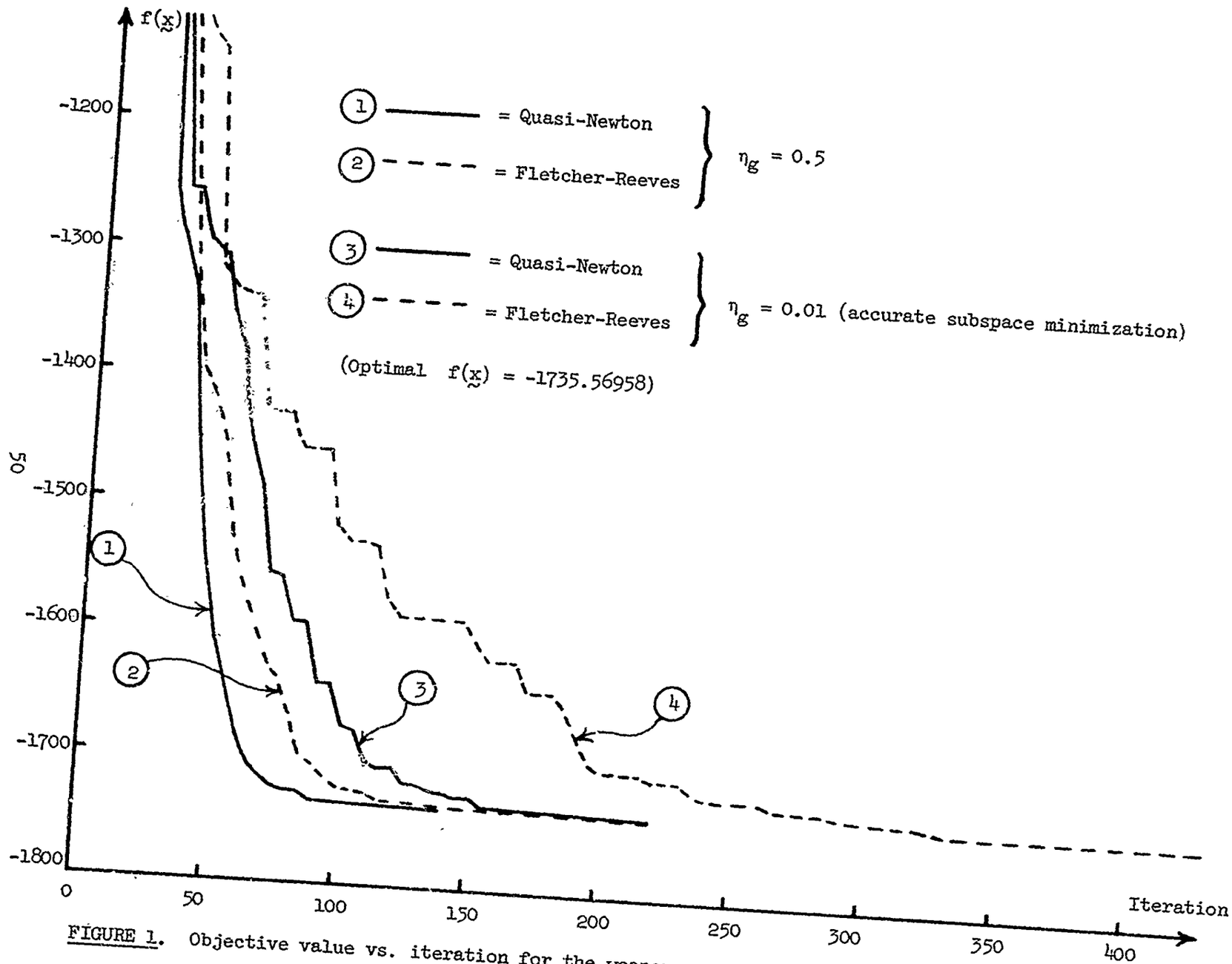


FIGURE 1. Objective value vs. iteration for the weapon assignment problem; η = 0.01 (accurate linesearch).

6. Comparison with other algorithms

Many of the ideas discussed here were either implicit in or anticipated by the work of Wolfe [56], [57], Faure and Huard [16] and McCormick [41], [42]. However there have since been such significant advances in implementation techniques for the numerical methods involved that there is little point in making detailed comparisons. Algorithmically, one important difference is our emphasis on keeping the number of superbasic variables as small as possible and changing that number by at most 1 each iteration.* With the quasi-Newton approach, this strategy retains maximum information about the projected Hessian. Even though the proof of convergence [41] for the variable-reduction method depended on regular resetting of the Hessian approximation, we never set $R = I$ except at the start of a run or in the rare event that the linesearch fails to find an improved point (in which case both R and the true Hessian are normally very ill-condition). Zig-zagging is controlled effectively by the tolerance η_g and the logic described in § 3.4. Rates of convergence within each subspace follow from analogous proofs for unconstrained algorithms.

Since the present algorithm possesses superlinear convergence properties and can handle rather arbitrary sized problems, it should be competitive with other algorithms designed specifically for quadratic

*In the original reduced-gradient algorithm the set of superbasics was effectively redefined each iteration as being the current set plus those nonbasic variables whose reduced costs were of the correct sign.

programming (e.g. Wolfe [55], Beale [4], Cottle [11], Lemke [37]). In particular a comparison with Beale's method would be relevant, since it is reported that his method is efficient for problems which have a small number of quadratic terms. On general (unstructured) problems with m and n large it is doubtful that Wolfe's algorithm or the linear complementarity methods would compare well because they work with linear systems of order $m + n$ rather than m.

A final comment on problems which have a large sparse set of general constraints $A\underset{\sim}{x} \geq b$ in relatively few variables (thus A is $m \times n$ with $m > n$). Ideally, methods designed specifically for this case use an active-constraint strategy and avoid transforming the whole of A each iteration (e.g. the version of the reduced-gradient algorithm in Wolfe [57]), and the implementation of Buckley [8]). The improved efficiency of these methods is analogous to the benefit that might be realized in the purely linear case if the dual simplex method were applied to the dual linear program. Nevertheless, given the use of sparse-matrix techniques, solution by the present (canonical form) method will be quite efficient unless $m \gg n$. In any event, with n moderate by assumption, this is one class of problems where the number of superbasic variables (and hence the dimension of the reduced hessian) will always remain manageably small.

7. Conclusion

Our primary aim has been to combine the simplex algorithm with quasi-Newton techniques in an efficient and reliable computer code for solving large, linearly constrained nonlinear programs. The full potential of conjugate-gradient methods in this context remains to be explored, but the necessary framework now exists; this framework will also accommodate extension to problems with a moderate number of nonlinear constraints (e.g. Jain, Lasdon and Saunders [35]). In the meantime the code is applicable to an important class of potentially very large problems, viz. dual geometric programs, and more generally it should provide a new dimension of utility to an already substantial body of large-scale linear programming models.

Acknowledgments

Work of this nature is necessarily a gathering together of methods and ideas from many sources. Where possible we have acknowledged the contribution of others within the text, and we wish to thank the individuals concerned. We are also grateful to J. Abadie, S. J. Byrne, A. Jain, L.S. Lasdon, A.S. Manne, J.A. Tomlin and M.H. Wright for assistance in various ways. In particular our thanks go to P. E. Gill and W. Murray for providing their linesearch procedure and for their valuable comments on the draft.

REFERENCES

[1] J. Abadie, "Application of the GRG algorithm to optimal control problems," in: J. Abadie, ed., Integer and nonlinear programming (North Holland, Amsterdam, 1970), pp. 191-211.

[2] R. H. Bartels, "A stabilization of the simplex method," Num. Math. 16 (1971) 414-434.

[3] R. H. Bartels and G. H. Golub, "The simplex method of linear programming using LU decomposition," Comm. ACM 12 (1969) 266-268.

[4] E. M. L. Beale, "Numerical methods," in: J. Abadie, ed., Nonlinear programming (North-Holland, Amsterdam, 1967) pp. 132-205.

[5] J. Bracken and G. P. McCormick, Selected applications of nonlinear programming (Wiley, New York, 1968).

[6] R. P. Brent, "Reducing the retrieval time of scatter storage techniques," Comm. ACM 16 (1973) 105-109.

[7] C. G. Broyden, "Quasi-Newton methods," in: W. Murray, ed., Numerical methods for unconstrained optimization (Academic Press, London and New York, 1972) pp. 87-106.

[8] A. Buckley, "An alternate implementation of Goldfarb's minimization algorithm," Math. Prog. 8 (1975) 207-231.

[9] A. R. Colville, "A comparative study on nonlinear programming codes," IBM New York Scientific Center Report 320-2949 (1968).

[10] A. R. Conn, "Linear programming via a non-differentiable penalty function," SIAM J. Numer. Anal. (to appear).

[11] R. W. Cottle, "The principal pivoting method of quadratic programming," in: G.B. Dantzig and A.F. Veinott, Jr., eds., Mathematics of the decision sciences, Part 1 (American Mathematical Society, 1968) pp. 144-162.

[12] G. B. Dantzig, Linear programming and extensions (Princeton University Press, New Jersey, 1963).

[13] W. C. Davidon, "Variable metric method for minimization," AEC Research and Development Report ANL-5990 (1959).

[14] I.S. Duff, "On algorithms for obtaining a maximum transversal," to appear (1976).

[15] I.S. Duff and J.K. Reid, "An implementation of Tarjan's algorithm for the block triangularizatior of a matrix," AERE Report C.S.S. 29 (1976), Harwell, England.

[16] P. Faure and P. Huard, "Resolution de programmes mathematiques a fonction nonlineaire par la methode du gradient reduit," Revue Francaise de Recherche Operationelle 36 (1965) 167-206.

[17] R. Fletcher, "Minimizing general functions subject to linear constraints," in: F. A. Lootsma, ed., Numerical methods for nonlinear optimization (Academic Press, London and New York, 1972) pp. 279-296.

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

[19] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," Computer J. 7 (1964) 149-154.

[20] P. E. Gill, G. H. Golub, W. Murray and M. A. Saunders, "Methods for modifying matrix factorizations," Math. Comp. 28 (1974) 505-535.

[21] P. E. Gill and W. Murray, "Quasi-Newton methods for unconstrained optimization," J. Inst. Maths. Applics. 9 (1972), 91-108.

[22] P. E. Gill and W. Murray, "Quasi-Newton methods for linearly constrained optimization," Report NAC 32 (1973), National Physical Laboratory, Teddington.

[23] P. E. Gill and W. Murray, "Newton-type methods for unconstrained and linearly constrained optimization," Math. Prog. 7 (1974) 311-350.

[24] P. E. Gill and W. Murray, "Safeguarded steplength algorithms for optimization using descent methods," Report NAC 37 (1974), National Physical Laboratory, Teddington.

[25] P. E. Gill and W. Murray, eds., Numerical methods for constrained optimization (Academic Press, London, 1974).

[26] P. E. Gill and W. Murray, "Linearly constrained optimization including quadratic and linear programming," in: Jacobs and Scriven, eds., Modern numerical analysis (Academic Press, London, 1976), Proc. of conference on "State of the art of numerical analysis," University of York (April 1976).

[27] P. E. Gill, W. Murray and S. M. Picken, "The implementation of two modified Newton algorithms for linearly constrained optimization," to appear.

[28] P. E. Gill, W. Murray and R. A. Pitfield, "The implementation of two revised quasi-Newton algorithms for unconstrained optimization," Report NAC 11 (1972), National Physical Laboratory, Teddington.

[29] P. E. Gill, W. Murray and M. A. Saunders, "Methods for computing and modifying the LDV factors of a matrix," Math. Comp. 29 (1975) 1051-1077.

[30] D. Goldfarb, "Extension of Davidon's variable metric method to maximization under linear inequality and equality constraints," SIAM J. Appl. Math. 17 (1969) 739-764.

[31] D. Goldfarb, "On the Bartels-Golub decomposition for linear programming bases," AERE Report C.S.S. 18 (1975), Harwell, England.

[32] E. Hellerman and D. C. Rarick, "Reinversion with the preassigned pivot procedure," Math. Prog. 1 (1971) 195-216.

[33] E. Hellerman and D. C. Rarick, "The partitioned preassigned pivot procedure," in: D.J. Rose and R.A. Willoughby, eds., Sparse matrices and their applications (Plenum Press, New York, 1972) 67-76.

[34] D. M. Himmelblau, Applied nonlinear programming (McGraw-Hill, 1972).

[35] A. Jain, L.S. Lasdon and M.A. Saunders," An in-core nonlinear mathematical programming system for large sparse nonlinear programs," to be presented at ORSA/TIMS joint national meeting, Miami, Florida (November, 1976).

[36] J. E. Kalan, "Aspects of large-scale in-core linear programming," Proceedings of ACM conference, Chicago (1971) 304-313.

[37] C.E. Lemke, "Bimatrix equilibrium points and mathematical programming," Management Sci. 11 (1965) 681-689.

[38] A. S. Manne, "Waiting for the Breeder," The review of economic studies symposium (1974) 47-65.

[39] A. S. Manne, "U.S. options for a transition from oil and gas to synthetic fuels," presented at the World Congress of the Econometric Society, Toronto (August 1975).

[40] A. S. Manne, "ETA: a model for Energy Technology Assessment," Working paper (1975), John Fitzgerald Kennedy School of Government, Harvard University, Cambridge, Mass.

[41] G. P. McCormick, "The Variable-Reduction Method for nonlinear programming," Management Sci. 17, 3 (1970) 146-160.

[42] G. P. McCormick, "A second order method for the linearly constrained nonlinear programming problem," in: J. B. Rosen, O. L. Mangasarian and K. Ritter, eds., Nonlinear programming (Academic Press, New York, 1970) pp. 207-243.

[43] B. A. Murtagh and P. D. Lucas, "The modelling of energy production and consumption in New Zealand," IBM (N.Z.) 5 (1975) 3-6.

[44] B. A. Murtagh and R. W. H. Sargent, "A constrained minimization method with quadratic convergence," in: R. Fletcher, ed., Optimization (Academic Press, New York, 1969) pp. 215-246.

[45] A. Perry, "An improved conjugate gradient algorithm," Technical note (March 1976), Dept. of Decision Sciences, Graduate School of Management, Northwestern University, Evanston, Illinois.

[46] E. Polak, Computational methods in optimization: a unified approach (Academic Press, 1971).

[47] M. J. D. Powell, "Restart procedures for the conjugate gradient method," AERE Report C.S.S. 24 (1975), Harwell, England.

[48] D. C. Rarick, An improved pivot row selection procedure, implemented in the mathematical programming system MPS III, Management Science Systems, Rockville, Maryland.

[49] R.W.H. Sargent, "Reduced-gradient and projection methods for nonlinear programming," in: P. E. Gill and W. Murray, eds., Numerical methods for constrained optimization (Academic Press, London, 1974) pp. 149-174.

[50] R. W. Sargent and B. A. Murtagh, "Projection methods for nonlinear programming," Math. Prog. 4 (1973) 245-268.

[51] M. A. Saunders, "Large-scale linear programming using the Cholesky factorization," Report STAN-CS-72-252 (1972), Computer Science Dept., Stanford University, Stanford, Calif.

[52] M. A. Saunders, "A fast,stable implementation of the simplex method using Bartels-Golub updating," in: J. R. Bunch and D. J. Rose, eds., Sparse Matrix Computations (Academic Press, New York and London, 1976) pp. 213-226.

[53] B. R. Smith, P. D. Lucas and B. A. Murtagh, "The development of a New Zealand energy model," N.Z.O.R. Journal, to appear.

[54] J. A. Tomlin, "On pricing and backward transformation in linear programming," Math. Prog. 6 (1974) 42-47.

[55] P. Wolfe, "The simplex method for quadratic programming," Econometrica 27 (1959) 382-398.

[56] P. Wolfe, "The Reduced Gradient Method," unpublished manuscript, The RAND Corporation (June 1962).

[57] P. Wolfe, "Methods of nonlinear programming," in: J. Abadie, ed., Nonlinear programming (North-Holland, Amsterdam, 1967) pp. 97-131.

[58] M. J. Wood, "The February 1975 state of BUILD," Ministry of Works and Development report (February 1975), Wellington, New Zealand.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(14) 1. REPORT NUMBER: SOL-76-15 ✓

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

(6) 4. TITLE (and Subtitle): NONLINEAR PROGRAMMING FOR LARGE, SPARSE SYSTEMS.

(9) 5. TYPE OF REPORT & PERIOD COVERED: Technical Report,

6. PERFORMING ORG. REPORT NUMBER

(10) 7. AUTHOR(s): B. A. Murtagh and M. A. Saunders

(15) 8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0267 ✓ ; E(04-3)-326

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Department of Operations Research
Stanford University
Stanford, CA 94305

(16) 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR-047-064

11. CONTROLLING OFFICE NAME AND ADDRESS
Operations Research Program
Office of Naval Research
Arlington, VA 22217

(11) 12. REPORT DATE: August 1976

13. NUMBER OF PAGES: 58

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

(12) 62p.

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

linear programming
nonlinear programming
optimization
sparse matrix

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

An algorithm for solving large-scale nonlinear programs with linear constraints is presented. The method combines efficient sparse matrix techniques as in the revised simplex method with stable variable-metric methods for handling the nonlinearities. A general-purpose production code (MINOS) is described, along with computational experience on a wide variety of problems.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD 765 LB