NONLINEAR PROGRAMMING FOR LARGE, SPA - 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 NO0014-75-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.

Reproduction in whole or in part is permitted for any purposes of the United States Government. This document has been approved for public release and sale; its distribution is unlimited.
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.
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:

\[
\text{minimize } F(x) = f(x^N) + x^T \bar{c} \tag{1}
\]

subject to

\[
Ax = b \tag{2}
\]

\[
L \leq x \leq U \tag{3}
\]

where \( A \) is \( m \times n \), \( m \leq n \). We partition \( x \) into a linear portion \( x^L \) and a nonlinear portion \( x^N \):

\[
x = \begin{bmatrix}
    x^N \\
    x^L \\
    x^U
\end{bmatrix}.
\]

*An earlier version of this paper was presented at the 8th Int. Symp. Math. Prog., Stanford, California, August 1973.
The components of $\mathbf{x}^N$ will normally be called the nonlinear variables. Note that $A$ and $c$ operate on all variables $\mathbf{x}$. In some cases the part of $\mathbf{g}^T \mathbf{x}$ involving $\mathbf{x}^N$ may be incorporated into $f(\mathbf{x}^N)$; in other cases $\mathbf{g}$ may be zero. We assume that the function $f(\mathbf{x}^N)$ is continuously differentiable in the feasible region, with gradient

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

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

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

\[
\text{minimize } f(x) = f(x_1, \ldots, x_n). \tag{4}
\]

subject to \( Ax \leq b \) \( \tag{5} \)

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

\[
f(x_{k+1}) = f(x_k) + \nabla f(x_k)^T \Delta x_k + \frac{1}{2} \Delta x_k^T G(x_k) \Delta x_k + \gamma \Delta x_k^T \lambda \tag{6}
\]

where

\[
\Delta x_k = x_{k+1} - x_k
\]

\[
G_k = \nabla^2 f(x_k)
\]

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

Suppose that the current point, \( x_k \), is on the boundary of \( m \) constraints \( (m < n) \). Denoting the corresponding \( m \) rows of \( A \) by the submatrix \( B \) we have:

\[
B x_k = b_m \tag{7}
\]

Variable-metric projection methods (Goldfarb [30], Murtagh and Sargent [44]) are based on two properties required of the step \( \Delta x_k \):

**Property 1.**

\[
G_k + G(x_k) \Delta x_k = B^T \lambda \tag{8}
\]
i.e., the step $\Delta x_k$ 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 (3) 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.

$$B \Delta x_k = 0$$

(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

$$\Delta x_k = -\alpha_k S_k(e_k - B^T \lambda_k)$$

(10)

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

$$\lambda_k = (BS_k B^T)^{-1} BS_k e_k$$

(11)

$\alpha_k$ is a step-size parameter used to adjust the length of the step $\Delta x_k$, 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 $(BS_k B^T)^{-1}$ 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:

\[
Ax = \begin{bmatrix} B_1 & B_2 & B_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = b \quad (12)
\]

\[
\begin{array}{c|c|c|c}
\text{basics} & \text{superbasics} & \text{nobasics} \\
\hline
m & s & n-m-s
\end{array}
\]
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:

$$
\begin{bmatrix}
\Delta x_1 \\
\Delta x_2 \\
\Delta x_3
\end{bmatrix} =
\begin{bmatrix}
B_1^T & 0 \\
B_2^T & 0 \\
B_3^T & I
\end{bmatrix}
\begin{bmatrix}
\lambda_1 \\
\lambda_2
\end{bmatrix}
$$

(13)

and Property 2 becomes:

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

(14)

where $\Delta x_k$ 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
\[ \Delta x_3 = 0 \] (15)

and
\[ \Delta x_1 = -W \Delta x_2 \] (16)

where
\[ W = B_1^{-1} B_2 \] (17)

Thus,
\[ \Delta x = \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x_2. \]

Equation (13) simplifies when multiplied by the matrix
\[
\begin{bmatrix}
I & 0 & 0 \\
-W^T & I & 0 \\
0 & 0 & I
\end{bmatrix}
\] (18)

First it provides an expression for estimates of the Lagrange multipliers for the general constraints:
\[
B_1^T \lambda_1 = \xi_1 + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} G \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x_2. \] (19)

Note that when \( \|\Delta x_2\| = 0 \) (which will mean \( \tilde{x} \) is stationary) we have
\[
B_1^T \lambda_1 = \xi_1 \] (20)

in which case \( \lambda_1 \) is analogous to the pricing vector \( \xi \) in the revised simplex method. (From now on we shall denote the solution of (20) by \( \xi \).)

Next we have from (13) that
\[
\lambda_2 = \xi_3 - B_2^T \lambda_1 + \begin{bmatrix} 0 & 0 & I \end{bmatrix} G \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x_2 \] (21)

and again when \( \|\Delta x_2\| = 0 \) this equation reduces to
\[ \lambda_2 - \zeta_2 = \frac{-W^T}{2} \]

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:

\[ \begin{bmatrix} -W^T & I & 0 \end{bmatrix} G \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta x_2 = -h \]

where

\[ h = \begin{bmatrix} -W^T & I & 0 \end{bmatrix} g = g_2 - W^T \alpha_1 = g_2 - B_2^T \zeta \]

The form of equation (23) suggests that

\[ \begin{bmatrix} -W^T & I & 0 \end{bmatrix} G \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \]

can be regarded as a "projected" (or "reduced") Hessian and \( h = [W^T \ I \ O]_2 \) a projected gradient, with (23) giving a Newton step in the independent variables \( \Delta x_2 \). Note that \( \|h\| = 0 \) becomes a necessary condition for a stationary point in the current set of active constraints, which, if the projected Hessian is nonsingular, implies that \( \|\Delta x_2\| = 0 \).

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 \( \Delta x_2 \). We will not spend time here discussing the relationship between...
the two seemingly different approaches, but refer the reader to two review
topics 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 $A\hat{x} = \hat{b}$ is the current
set of $n$-s active constraints, $Z$ is an $n \times s$ matrix such that

$$A\hat{Z} = 0.$$ \hspace{1cm} (26)

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

A. Compute the projected (reduced) gradient $g_A = Z^T \hat{g}$.

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

$$G_A \approx Z^T G Z$$

C. Obtain an approximate solution to the system of equations

$$Z^T G Z p_A = -Z^T \hat{g}$$ \hspace{1cm} (27)

by solving the system

$$G_A p_A = -g_A$$

D. Compute the search direction $\hat{p} = Z p_A$. 

10
E. Perform a line-search to find an approximation to $\alpha^*$, where

$$ f(\tilde{x} + \alpha^* \mathbf{p}) = \min_{\alpha} f(\tilde{x} + \alpha \mathbf{q}) $$

$$(\tilde{x} + \alpha \mathbf{q} \text{ feasible})$$

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

$$Z = \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} = \begin{bmatrix} -B_1^{-1}B_2 \\ I \\ 0 \end{bmatrix} \begin{bmatrix} m \\ s \\ n-m-s \end{bmatrix} \tag{28}$$

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-spareness 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.

15
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 [23] 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, $\eta$ ($0.0 < \eta < 1.0$) 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 \( \bar{x} = [x_1, x_2, x_3]^T \) satisfying \([B_1, B_2, B_3] \bar{x} = b\), \( \ell \leq \bar{x} \leq u \).

(b) The corresponding function value \( f(\bar{x}) \) and gradient vector \( \bar{g}(\bar{x}) = [g_1, g_2, g_3]^T \).

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

(d) A factorization, \( L U \), of the \( m \times m \) basis matrix \( B_1 \).

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

(f) A vector \( \bar{p} \) satisfying \( B_1^T \bar{p} = \bar{g}_1 \).

(g) The reduced gradient vector \( \bar{h} = \bar{g}_2 - B_2^T \bar{p} \).

(h) Small positive convergence tolerances TOLRG and TOLDJ.

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

If \( \| \bar{h} \| > \text{TOLRG} \) go to step 3.

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

(a) Calculate \( \lambda = \bar{g}_2 - B_2^T \bar{p} \).

(b) Select \( \lambda_1 < -\text{TOLDJ} \) \( (\lambda_2 > +\text{TOLDJ}) \), the largest elements of \( \lambda \) 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 $a_q$ as a new column of $B_2$;

(iii) add $\lambda_q$ as a new element of $\mathbf{b}$;

(iv) add a suitable new column to $R$.

(d) Increase $s$ by 1.

**Step 3. (Compute direction of search, $p = Zp_2$)**

(a) Solve $R^T R p_2 = -b$.

(b) Solve $L U p_2 = -B_2 b_2$.

(c) Set $p = \begin{bmatrix} p_1 \\ p_2 \\ 0 \end{bmatrix}$.

**Step 4. (Ratio test, "CHUZ")**

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

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

**Step 5. (Line-search)**

(a) Find $\alpha$, an approximation to $\alpha^*$, where

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

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

16
Step 6. (Compute reduced gradient, $\tilde{h} = \nabla^T g$)

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

(b) Compute the new reduced gradient, $\tilde{h} = g_2 - B_T^2 \eta$.

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

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

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

Step 7. (Change basis, delete one superbasic)

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

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

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

\[
\begin{bmatrix}
B_1 \\
\xi_1 \\
\end{bmatrix} \quad \text{and} \quad \begin{bmatrix}
B_2 \\
\xi_2 \\
\end{bmatrix}
\]

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

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

(iii) compute the new reduced gradient $\tilde{h} = g_2 - B_T^2 \eta$.

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

17
(c) Make the $q$-th variable in $B_2$ nonbasic at the appropriate bound, thus:

(i) delete the $q$-th columns of

\[
\begin{bmatrix} B_2 \\ T \\ X_2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} R \\ t \end{bmatrix};
\]

(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 $\|h\|$ is sufficiently small, which means an average of about once every 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 $f(x)$ and $g(x)$ 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 $B_1x = b$ or $B_1^Tz = c$. 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 $R^TR \approx Z^TGZ$ whenever $x$ and/or $Z$ change. The notation $\tilde{R}$ 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

$$
\tilde{R}^T\tilde{R} = R^TR + \frac{1}{\alpha\xi} \frac{\chi\chi^T}{\alpha\xi^2} + \frac{1}{\xi^2} \frac{hh^T}{\alpha\xi^2}
$$

The Rank-one Formula:

$$
\tilde{R}^T\tilde{R} = R^TR + \frac{1}{\alpha\xi} \frac{\chi\chi^T}{\alpha\xi^2}
$$

where $\chi = \tilde{z} - z$, the change in reduced gradient, and $\chi = \chi + \alpha h$. 

19
The COMDFP formula can be used on both constrained and unconstrained steps \( (\alpha = \alpha_{\text{max}} \text{ and } \alpha < \alpha_{\text{max}}, \text{ resp.}) \). 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 \( \alpha = \alpha_{\text{max}} \) and \( \alpha_{\text{max}} \) is very small it is possible that the computed value of \( \chi \) 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

\[
\bar{h}^T \bar{p}_2 > 0.9 \, h^T p_2
\]

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

\[
\eta = \frac{h^T p_2}{|h^T p_2|} < 0.9,
\]

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

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

\[
\hat{R}^T \bar{R} = R^T R + vv^T
\]

\[
\hat{R}^T \bar{R} = R^T R - vv^T
\]

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

\[ Z = Z(I + e_q y^T) \]  

where \( e_q \) is the q-th unit vector and \( y \) is defined by the equations

\[ B_{2-p}^T \pi_p = e_p \]
\[ \chi = B_{2-p}^T \pi_p \]
\[ y_q = y_{2-p} \]
\[ y = -\frac{1}{y_q} (\chi + e_q) \]

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 \( \chi \), viz.

\[ y_j = \chi^T e_j = \pi_p B_2 e_j = e_p^{-1}(B_2 e_j) \]

is the pivot element that would arise if the j-th column of \( B_2 \) were selected for the basis change. Hence \( \chi \) 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 $B_C$ 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:

\[ y_{\text{max}} = \max |y_j| \]

\[ d_j = \min_{(x_j \text{ superbasic})} |x_j - l_j|, |x_j - u_j| \]

\[ d_q = \max \{d_j | y_j| \geq 0.1 y_{\text{max}} \} \]

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. $\pi_p$ can be used to update the vector $\pi$ that is computed in Step 6(a). (after the last move but before the current basis change). Thus

\[ \tilde{\pi} = \pi + (\tilde{h}_q/y_q)\pi_p \]

where $\tilde{h}_q$ is the appropriate element of the reduced gradient $\tilde{h}$ 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. $\pi_p$ can also be used to update the LU factors of $B_L$ (see Tomlin [54], Goldfarb [31]).

The modification to $R$ corresponding to equation (29) is accomplished as follows:
If \( \tilde{r}_q \) is the \( q \)-th column of \( R \), this expression may be written

\[
\tilde{R}^T \tilde{R} = (I + v\tilde{e}_q^T) R^T R (I + \tilde{c} \tilde{v}^T)
\]

A partial backward sweep of plane rotations \( Q_j \) (\( j = q, q-1, \ldots, 1 \)) reduces \( \tilde{r}_q \) to a multiple of \( \tilde{e}_q \), 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 \( \tilde{R} \) by a partial forward sweep of plane rotations \( Q_j \) (\( j = q+1, \ldots, s \)):

\[
\text{DELCOL: } Q_s \cdots Q_{q+2} Q_{q+1} \times \begin{bmatrix} R \text{ with} \\
q\text{-th column} \end{bmatrix} = \begin{bmatrix} \tilde{R} \\
0 \end{bmatrix}
\]

23
3.3.4 Addition of one superbasic variable (Step 2(c))

When a vector $a_q$ is added to $B_2$ the new null-space matrix is

$$
\tilde{Z} = [z \ g]
$$

where $z = \begin{bmatrix} e_s \\ 0 \end{bmatrix}$

Following Gill and Murray ([25], pp. 76-77) we approximate the vector $Gz$ by finite differences, thus:

$$
\chi = \frac{g(x + \delta z) - g(x)}{\delta} = Gz + o(\delta \|z\|^2)
$$

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

ADDCOL:

\begin{align*}
\text{Solve} & \quad R^T \chi = \tilde{Z}^T \chi \\
\text{Compute} & \quad \sigma = \tilde{z}^T \chi - \|z\|^2, \quad \rho = |\sigma|^{1/2} \\
\text{Take} & \quad \tilde{R} = \begin{bmatrix} R \\ \tilde{z} \\ \rho \end{bmatrix}
\end{align*}

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

Comparison of

$$
\tilde{R}^T \tilde{R} = \begin{bmatrix} R^T \\ \tilde{z}^T \\ \rho \end{bmatrix} \begin{bmatrix} R & \tilde{z} \\ \tilde{z}^T & \rho \end{bmatrix} = \begin{bmatrix} R^T R & \tilde{z}^T \chi \\ \tilde{z}^T Z & \|z\|^2 \end{bmatrix}
$$

and

$$
24
$$
shows that if $R^T R$ provides a good approximation to $Z^T G Z$
then $R^T \tilde{R}$ has some chance of being a useful approximation to $Z^T G Z$.
The main work involved here is in computing $B^{-1}_{q}$, the gradient vector $g(x + \delta x)$
and the projection $\tilde{Z}^T x$. This work is essentially wasted if
the expression for $\sigma$ is not positive, which may happen for many reasons,
e.g. if $Z^T G Z$ 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 $\chi = 0$ and take $\rho$ to be either $(\tilde{Z}^T x)^{1/2}$ or 1.0, thus:

$$ \bar{R} = \begin{bmatrix} R & 0 \\ 0 & \rho \end{bmatrix} $$

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 $\sigma < 0$ occurs only occasionally
or not at all. Computing $\chi$ and $\rho$ 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 $g(x)$ the difference approximation
to $G_Z$ is unlikely to be good. (An example is when $f(x)$ 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, $\chi$ and $\rho$ 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 \( d_{\text{max}} \) and \( d_{\text{min}} \) be the largest and smallest diagonals of \( R \). As a heuristic means of detecting the above situation we monitor \( \|y\| \) and resort to (30) whenever \( \|y\| \) is significantly large than \( d_{\text{max}} \) or smaller than \( d_{\text{min}} \). (As a side benefit, the expense of computing \( z^T y \) and \( y \) is then avoided.) A final similar test is made on \( \rho \).

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 RIPROD, DEICOL 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, \( \lambda \)) and add one more superbasic variable.

Suppose that after a particular iteration we have

- \( \Delta x_2 \) = the change in the superbasic variables
- \( \Delta f \) = the change in \( f \)
- \( \pi \) = the new pricing vector
- \( h = Z^T g \), the new reduced gradient
- \( \epsilon_x, \epsilon_f, TOLRG, \epsilon_g \) = positive scalars
- \( \epsilon \) = machine precision

and let \( T_1 \) be a set of tests (with values true or false) defined as follows:

- \( T_1 : \| \Delta x_2 \| \leq (\epsilon_x + \epsilon^{1/2})(1 + \| x_2 \|) \)
- \( T_2 : | \Delta f | \leq (\epsilon_f + \epsilon)(1 + | f |) \)
- \( T_3 : \| h \| \leq TOLRG \)
- \( T_4 : \| h \| \leq \epsilon_g \| \pi \| \)

27
In place of the simple test

\[
\text{if } T_3 \text{ then compute } \lambda ,
\]

the following combined test is used:

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

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 \( \epsilon_x, \epsilon_f, \text{TOLRG and } \epsilon_g \) 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 \( \epsilon_x \) and \( \epsilon_f \) 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 \( \epsilon_g \) 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 \( \lambda_q = g_q - \pi a_q \). If the corresponding variable \( x_q \) becomes superbasic, the reduced gradient for the expanded subspace will be

\[
\bar{b} = \begin{bmatrix} b \\ \lambda_q \end{bmatrix}
\]

Now recall from equation (21) that unless \( b \) is reasonably small, even one further iteration could change \( \pi \) and hence \( \lambda_q \) significantly.
Therefore as a safeguard (which is admittedly heuristic) we accept $\lambda_q$ and move into the new subspace only if $\|b\|_\infty \leq 0.9|\lambda_q|$, which implies

$$\|b\|_\infty \leq 0.9\|\tilde{b}\|_\infty.$$ 

We then reset TOLRG for the new subspace to be

$$TOLRG = \eta_g \|\tilde{b}\|_\infty$$

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 $\eta_g$ 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 \( g(x) \) is available and we have avoided explicit use of the Hessian matrix \( G(x) \). 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 \( p = Zp_2 \) and \( h = Z^T g \). (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.,

\[
\hat{g}_j = \frac{f(x + \delta e_j) - f(x)}{\delta} \approx g_j .
\]

(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. \( h \approx Z^T \hat{g} \).

2. An alternative which is normally viable would be to difference \( f(x) \) along the directions \( \delta \)

\[
\hat{h}_j = \frac{f(x + \delta e_j) - f(x)}{\delta} \approx \delta^T \hat{g} = h_j
\]

where \( \delta_j = \delta e_j, j = 1, \ldots, s \). Unfortunately this approach is not practical for large problems, since storage limitations prevent saving all \( s \) vectors \( \delta_j \), and the work involved rules out recomputing them when required.
3. If \( g(x) \) and perhaps \( G(x) \) are available, the system of equations

\[
Z^T G z = - Z^T g
\]

(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:

\[ Z^T G Z = [z_1^T g z_j] \]

or differencing \( g(x) \) thus:

\[
x_j = \frac{g(x + \delta z_j) - g(x)}{\delta} = \delta z_j,
\]

\[
Z^T G Z \approx \frac{1}{2} (Z^T v + v^T z).
\]

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 \( Z^T (G(z_j)) \).

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) $\bar{h} = -Z^T \tilde{R}.$

(b) \text{if} \ restart \ \text{then} \quad \tilde{p}_2 = -\bar{h} \\
\text{else} \quad \tilde{p}_2 = -\bar{h} + \beta \bar{p}_2

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

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

- Fletcher and Reeves [19]: $\beta = \|\bar{h}\|^2 / \|\bar{h}\|^2$
- Polak and Ribiere [46]: $\beta = \bar{h}^T (\bar{h} - \bar{h}) / \|\bar{h}\|^2$
- Perry [45]: $\beta = \bar{h}^T (\bar{h} - \bar{h} - \alpha \bar{p}_2) / \bar{p}_2^T (\bar{h} - \bar{h})$

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 $\beta$ 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 exact 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

$$R^T R = Z^T G Z$$  \hfill (32)

can often be maintained exactly without recomputing $Z^T G Z$ every iteration.

Such a specialization has been described by Gill and Murray [26], along with the measures required to allow for $Z^T G Z$ 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 $Z^T G Z$ could be indefinite.)

2. In procedure ADDCOL (§3.3.4) compute the vector $y = G z$ 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 $\chi = Cz$
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 $\chi$ 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 $\alpha$ 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.
5. **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 $L \leq x \leq U$ 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

\[ f(x) = \sum_k^J \sum_j^K \left( c_{jk} x_{jk} + \ln(x_{jk}/\sum_i x_{ik}) \right) \]

(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

\[ f(x) = \sum_{j=1}^{20} \sum_{i=1}^{5} a_{ij} x_{ij} - 1 \]

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 of 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

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

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 $\sum_{i=1}^{16} z_i^2$ 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 $\|\nabla f\|/\|\nabla g\|$ was reduced to $10^{-6}$ or less at optimality.
<table>
<thead>
<tr>
<th>Problem no.</th>
<th>Rows</th>
<th>Columns</th>
<th>Nonzero elements</th>
<th>Nonlinear variables</th>
<th>Iterations</th>
<th>Evaluations of $f(x), g(x)$</th>
<th>Final no. of superbasics</th>
<th>Time (secs.)</th>
<th>Standard time ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>5</td>
<td>47</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>0.63</td>
<td>0.008</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>16</td>
<td>80</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>3</td>
<td>1.50</td>
<td>0.013</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>100</td>
<td>147</td>
<td>100</td>
<td>133</td>
<td>296</td>
<td>18</td>
<td>48.30</td>
<td>0.58</td>
</tr>
<tr>
<td>5a</td>
<td>10</td>
<td>24</td>
<td>240</td>
<td>24</td>
<td>3</td>
<td>8</td>
<td>14</td>
<td>1.65</td>
<td>0.019</td>
</tr>
<tr>
<td>5b</td>
<td>17</td>
<td>52</td>
<td>884</td>
<td>52</td>
<td>13</td>
<td>13</td>
<td>35</td>
<td>6.21</td>
<td>0.075</td>
</tr>
<tr>
<td>5c</td>
<td>19</td>
<td>78</td>
<td>1482</td>
<td>78</td>
<td>21</td>
<td>21</td>
<td>59</td>
<td>13.47</td>
<td>0.16</td>
</tr>
<tr>
<td>6</td>
<td>74</td>
<td>83</td>
<td>529</td>
<td>15</td>
<td>80</td>
<td>40</td>
<td>3</td>
<td>37.03</td>
<td>0.45</td>
</tr>
<tr>
<td>7</td>
<td>95</td>
<td>200</td>
<td>*</td>
<td>24</td>
<td>103</td>
<td>72</td>
<td>0</td>
<td>42.43</td>
<td>0.51</td>
</tr>
<tr>
<td>8</td>
<td>324</td>
<td>425</td>
<td>*</td>
<td>91</td>
<td>348</td>
<td>215</td>
<td>0</td>
<td>538.3</td>
<td>6.48</td>
</tr>
</tbody>
</table>

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

*Includes Phase 1 iterations
<table>
<thead>
<tr>
<th>Problem no.</th>
<th>Rows</th>
<th>Columns</th>
<th>Nonzero elements</th>
<th>Nonlinear variables</th>
<th>Iterations</th>
<th>Evaluations of $f(x), g(x)$</th>
<th>Final no. of superbasics</th>
<th>Time (secs.)</th>
<th>Standard time ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>16</td>
<td>45</td>
<td>99</td>
<td>45</td>
<td>103</td>
<td>452</td>
<td>24</td>
<td>2.9</td>
<td>0.74</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>100</td>
<td>147</td>
<td>100</td>
<td>139</td>
<td>355</td>
<td>18</td>
<td>2.6</td>
<td>0.66</td>
</tr>
<tr>
<td>9a</td>
<td>356</td>
<td>1134</td>
<td>4180</td>
<td>0</td>
<td>539</td>
<td>0</td>
<td>0</td>
<td>33.3</td>
<td>8.5</td>
</tr>
<tr>
<td>9b</td>
<td>314</td>
<td>631</td>
<td>2122</td>
<td>28</td>
<td>2027</td>
<td>4536</td>
<td>21</td>
<td>119.5</td>
<td>30.5</td>
</tr>
<tr>
<td>9c</td>
<td>314</td>
<td>631</td>
<td>2122</td>
<td>28</td>
<td>1397</td>
<td>2862</td>
<td>21</td>
<td>83.6</td>
<td>21.3</td>
</tr>
<tr>
<td>10a</td>
<td>320</td>
<td>679</td>
<td>2519</td>
<td>44</td>
<td>948</td>
<td>1768</td>
<td>27</td>
<td>56.6</td>
<td>14.4</td>
</tr>
<tr>
<td>10b</td>
<td>320</td>
<td>679</td>
<td>2519</td>
<td>44</td>
<td>236</td>
<td>570</td>
<td>29</td>
<td>17.5</td>
<td>4.5</td>
</tr>
<tr>
<td>10c</td>
<td>320</td>
<td>679</td>
<td>2519</td>
<td>44</td>
<td>350</td>
<td>902</td>
<td>26</td>
<td>26.9</td>
<td>6.9</td>
</tr>
</tbody>
</table>

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 $\epsilon_k = 10^{-k}$ on all variables, for various values of $k$ in the range 4 to 10. (The problem is infeasible with $x_j > 10^{-3}$.) 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.
<table>
<thead>
<tr>
<th>Lo-bound $\varepsilon_k$</th>
<th>No. of superbasics</th>
<th>$f(x)$</th>
<th>Iterations*</th>
<th>Evaluations* of $f, g$</th>
<th>Estimate of $\kappa(R^T R)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-4}$</td>
<td>10</td>
<td>-1910.366249932</td>
<td>46</td>
<td>130</td>
<td>$6 \times 10^5$</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>14</td>
<td>-1910.381531984</td>
<td>21</td>
<td>75</td>
<td>$5 \times 10^6$</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>17</td>
<td>-1910.382772060</td>
<td>22</td>
<td>72</td>
<td>$1 \times 10^8$</td>
</tr>
<tr>
<td>$10^{-7}$</td>
<td>19</td>
<td>-1910.382872190</td>
<td>22</td>
<td>88</td>
<td>$1 \times 10^9$</td>
</tr>
<tr>
<td>$10^{-8}$</td>
<td>23</td>
<td>-1910.382880402</td>
<td>22</td>
<td>90</td>
<td>$6 \times 10^7$</td>
</tr>
<tr>
<td>$10^{-9}$</td>
<td>24</td>
<td>-1910.382881101</td>
<td>22</td>
<td>90</td>
<td>$4 \times 10^8$</td>
</tr>
<tr>
<td>$10^{-10}$</td>
<td>24</td>
<td>-1910.382881161</td>
<td>5</td>
<td>27</td>
<td>$8 \times 10^7$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>160</td>
<td>572</td>
<td></td>
</tr>
</tbody>
</table>

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

*Additional to previous run.

**A lower bound on the condition number of the projected Hessian approximation $R^T R$ is the square of the ratio of the largest and smallest diagonals of $R$.**
<table>
<thead>
<tr>
<th>$\varepsilon_k$</th>
<th>$x_j$</th>
<th>(j=13)</th>
<th>(45)</th>
<th>(9)</th>
<th>(28)</th>
<th>(8)</th>
<th>(22)</th>
<th>(32)</th>
<th>(40)</th>
<th>(21)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10^{-4})</td>
<td>(x_{9,2})</td>
<td>44.096011</td>
<td>2.22e-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(10^{-5})</td>
<td>(x_{2,7})</td>
<td>44.139306</td>
<td>2.257e-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(10^{-6})</td>
<td>(x_{5,2})</td>
<td>44.43326</td>
<td>2.261e-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(10^{-7})</td>
<td>(x_{11,3})</td>
<td>44.143614</td>
<td>2.261e-7</td>
<td>4.192e-7</td>
<td>1.033e-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(10^{-8})</td>
<td>(x_{1,2})</td>
<td>44.143614</td>
<td>2.602e-8</td>
<td>4.192e-7</td>
<td>1.033e-7</td>
<td>3.708e-8</td>
<td>2.241e-8</td>
<td>2.302e-8</td>
<td>1.091e-8</td>
<td></td>
</tr>
<tr>
<td>(10^{-9})</td>
<td>(x_{5,3})</td>
<td>44.143614</td>
<td>7.493e-9</td>
<td>4.192e-7</td>
<td>1.033e-7</td>
<td>3.708e-8</td>
<td>2.241e-8</td>
<td>2.169e-8</td>
<td>3.155e-9</td>
<td>2.426e-9</td>
</tr>
<tr>
<td>(10^{-10})</td>
<td>(x_{15,3})</td>
<td>44.143614</td>
<td>5.462e-9</td>
<td>4.192e-7</td>
<td>1.033e-7</td>
<td>3.708e-8</td>
<td>2.241e-8</td>
<td>2.098e-8</td>
<td>5.462e-9</td>
<td>2.562e-9</td>
</tr>
<tr>
<td><strong>SUMT</strong></td>
<td></td>
<td>44.58</td>
<td>2.476e-6</td>
<td>5.441e-6</td>
<td>3.437e-6</td>
<td>2.794e-6</td>
<td>3.102e-6</td>
<td>3.264e-6</td>
<td>0.0</td>
<td>1.546e-6</td>
</tr>
</tbody>
</table>

**Table 4.** Selected solution values for Problem 3.
Blank entries mean $x_j$ was nonbasic at the appropriate bound $\varepsilon_k$. 
For interest, the last row of Table I 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 \( f(x + \alpha \beta) \) with values of \( \alpha \) that are very close together. On the IBM 370/168 this prevented resolution below \( 10^{-10} \), although for this special case \( f(x) \) 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

\[ f(x,y) = \frac{a}{x^2 y^3} \]

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

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 \( O(10^{14}) \), which in this case is unnecessarily large.
Replacing each \( y \) by a variable \( \tilde{y} = y/100 \) gave a significant improve-
ment 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

\[ \eta = \text{linesearch accuracy tolerance (eta in §3)}; \]
\[ \eta_g = \text{the tolerance for minimization within each subspace (see §3.4).} \]

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 \( \eta \). 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 \( f(x) \) 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.
Quasi-Newton Fletcher-Reeves Polak-Ribiere Perry

<table>
<thead>
<tr>
<th>$\eta$</th>
<th>Quasi-Newton</th>
<th>Fletcher-Reeves</th>
<th>Polak-Ribiere</th>
<th>Perry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>123</td>
<td>226</td>
<td>222</td>
<td>198</td>
</tr>
<tr>
<td></td>
<td>375</td>
<td>840</td>
<td>806</td>
<td>713</td>
</tr>
<tr>
<td>0.01</td>
<td>139</td>
<td>223</td>
<td>237</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>255</td>
<td>728</td>
<td>770</td>
<td>849</td>
</tr>
<tr>
<td>0.1</td>
<td>122</td>
<td>227</td>
<td>238</td>
<td>228</td>
</tr>
<tr>
<td></td>
<td>281</td>
<td>671</td>
<td>709</td>
<td>665</td>
</tr>
<tr>
<td>0.2</td>
<td>137</td>
<td>250</td>
<td>252</td>
<td>218</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>721</td>
<td>749</td>
<td>578</td>
</tr>
<tr>
<td>0.3</td>
<td>148</td>
<td>239</td>
<td>248</td>
<td>307</td>
</tr>
<tr>
<td></td>
<td>291</td>
<td>648</td>
<td>688</td>
<td>814</td>
</tr>
<tr>
<td>0.4</td>
<td>156</td>
<td>282</td>
<td>296</td>
<td>309</td>
</tr>
<tr>
<td></td>
<td>289</td>
<td>742</td>
<td>855</td>
<td>762</td>
</tr>
<tr>
<td>0.5</td>
<td>153</td>
<td>275</td>
<td>394</td>
<td>612</td>
</tr>
<tr>
<td></td>
<td>242</td>
<td>695</td>
<td>1079</td>
<td>1411</td>
</tr>
<tr>
<td>0.9</td>
<td>207</td>
<td>694</td>
<td>&gt;999</td>
<td>818</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>987</td>
<td>&gt;2748</td>
<td>960</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$\eta$</th>
<th>Quasi-Newton</th>
<th>Fletcher-Reeves</th>
<th>Polak-Ribiere</th>
<th>Perry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>220</td>
<td>493</td>
<td>440</td>
<td>440</td>
</tr>
<tr>
<td></td>
<td>615</td>
<td>1628</td>
<td>1514</td>
<td>1495</td>
</tr>
<tr>
<td>0.01</td>
<td>219</td>
<td>498</td>
<td>471</td>
<td>466</td>
</tr>
<tr>
<td></td>
<td>548</td>
<td>1520</td>
<td>1520</td>
<td>1476</td>
</tr>
<tr>
<td>0.1</td>
<td>209</td>
<td>560</td>
<td>508</td>
<td>550</td>
</tr>
<tr>
<td></td>
<td>451</td>
<td>1597</td>
<td>1461</td>
<td>1568</td>
</tr>
<tr>
<td>0.2</td>
<td>218</td>
<td>582</td>
<td>231</td>
<td>585</td>
</tr>
<tr>
<td></td>
<td>445</td>
<td>1599</td>
<td>1517</td>
<td>1626</td>
</tr>
<tr>
<td>0.3</td>
<td>229</td>
<td>612</td>
<td>634</td>
<td>611</td>
</tr>
<tr>
<td></td>
<td>411</td>
<td>1557</td>
<td>1752</td>
<td>1625</td>
</tr>
<tr>
<td>0.4</td>
<td>262</td>
<td>748</td>
<td>691</td>
<td>752</td>
</tr>
<tr>
<td></td>
<td>441</td>
<td>1831</td>
<td>1821</td>
<td>1788</td>
</tr>
<tr>
<td>0.5</td>
<td>262</td>
<td>691</td>
<td>818</td>
<td>894</td>
</tr>
<tr>
<td></td>
<td>377</td>
<td>1633</td>
<td>1993</td>
<td>1974</td>
</tr>
<tr>
<td>0.9</td>
<td>288</td>
<td>&gt;999</td>
<td>&gt;999</td>
<td>&gt;999</td>
</tr>
<tr>
<td></td>
<td>345</td>
<td>&gt;1855</td>
<td>&gt;1658</td>
<td>&gt;1156</td>
</tr>
</tbody>
</table>

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; 
\( \eta = 0.01 \) (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 \( \eta_g \) 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 $Ax \geq b$ 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 manageable 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


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.